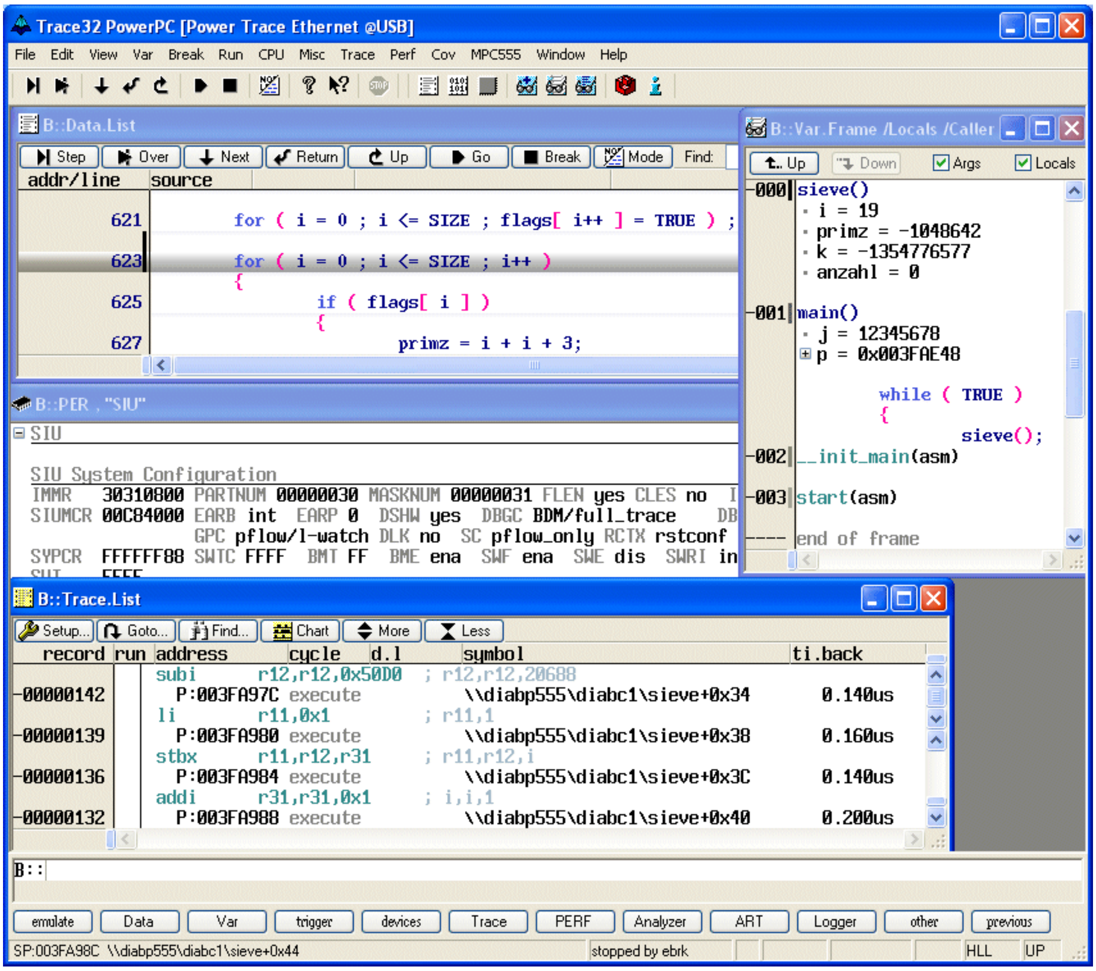
BDM Debugger and Trace for MPC555/8xx

BDM Debugger and Trace for MPC555/8xx

Version February, 07 2007



Trace32 PowerPC [Power Trace Ethernet @USB]
File Edit View Var Break Run CPU Misc Trace Perf Cov MPC555 Window Help
B::Data.List
Step Over Next Return Up Go Break Mode Find:
B::Var.Frame /Locals /Caller
Up Down Args Locals
B::PER , "SIU"
B::Trace.List
Setup... Goto... Find... Chart More Less
emulate Data Var trigger devices Trace PERF Analyzer ART Logger other previous

```
addr/line  source
      621          for ( i = 0 ; i <= SIZE ; flags[ i++ ] = TRUE ) ;
      623          for ( i = 0 ; i <= SIZE ; i++ )
                   {
      625                  if ( flags[ i ] )
                           {
      627                          primz = i + i + 3;
```

```
-000|sieve()
     - i = 19
     - primz = -1048642
     - k = -1354776577
     - anzahl = 0
-001|main()
     - j = 12345678
     + p = 0x003FAE48

            while ( TRUE )
            {
                    sieve();
-002|__init_main(asm)
-003|start(asm)
----|end of frame
```

```
⊟ SIU

 SIU System Configuration
 IMMR   30310800 PARTNUM 00000030 MASKNUM 00000031 FLEN yes CLES no   I
 SIUMCR 00C84000 EARB int   EARP 0   DSHW yes  DBGC BDM/full_trace    DB
                 GPC pflow/l-watch DLK no  SC pflow_only RCTX rstconf
 SYPCR  FFFFFF88 SWTC FFFF   BMT FF   BME ena  SWF ena  SWE dis  SWRI in
```

```
    record|run|address     cycle   |d.l        |symbol                        |ti.back
          |   subi     r12,r12,0x50D0  ; r12,r12,20688
-00000142 |     P:003FA97C execute      \\diabp555\diabc1\sieve+0x34      0.140us
          |   li       r11,0x1         ; r11,1
-00000139 |     P:003FA980 execute      \\diabp555\diabc1\sieve+0x38      0.160us
          |   stbx     r11,r12,r31     ; r11,r12,i
-00000136 |     P:003FA984 execute      \\diabp555\diabc1\sieve+0x3C      0.140us
          |   addi     r31,r31,0x1     ; i,i,1
-00000132 |     P:003FA988 execute      \\diabp555\diabc1\sieve+0x40      0.200us
```

```
B::
```

```
SP:003FA98C  \\diabp555\diabc1\sieve+0x44          stopped by ebrk                HLL   UP
```

Quick Start

Starting up the BDM Debugger is done by the following steps:

1. Select the device prompt B: for the TRACE32 ICD-Debugger, if the device prompt is not active after starting the TRACE32 software.

   ```
   b:
   ```

2. Select the CPU type to load the CPU specific settings.

   ```
   SYStem.CPU MPC563
   ```

   The default CPU is the MPC860.

3. Inform the debugger where's FLASH/ROM on the target, this is necessary for the use of the onchip breakpoints.

   ```
   MAP.BOnchip 0x100000++0x0fffff
   ```

4. Enter debug mode.

   ```
   SYStem.Up
   ```

   This command resets the CPU, enables the debug mode and stops the CPU at the first opfetch (reset vector). After this command is executed it is possible to access memory and registers.

5. Configure the IBUS.

   ```
   SYStem.Option IBUS NONE        ;No show cycles are performed.
                                  ;Recommanded for BDM debugger
                                  ;only

   SYStem.Option IBUS IND         ;Show cycles are generated for all
                                  ;indirect changes in the program
                                  ;flow. Recommanded if a RISC Trace
                                  ;or PowerTrace module is connected.
   ```

6. Set the special function registers to prepare your target memory for program loading.

   ```
   Data.Set SPR:027E %Long 0x800
   ???
   ????
   ```

7. Load the program.

```
Data.LOAD.Elf diabp555.x        ;Load ELF file
```

The extension for the Data.LOAD command (here Elf) depends on the file format generated by the compiler. For information on the compiler specific extensions refer to the section **Compilers**.

The start up sequence can be automated using the script language PRACTICE. A typical start sequence is shown below:

```
b::                               ;Select the ICD-Debugger device prompt

WinClear                          ;Delete all windows

MAP.BOnchip 0x100000++0x0fffff    ;Specify where's FLASH/ROM

SYStem.CPU 0x563                  ;Select the processor type

SYStem.Up                         ;Reset the target and enter debug mode

Data.LOAD.Elf diabp563.x          ;Load the application

Register.Set PC main              ;Set the PC to the function main

Data.List          *)             ;Open a source listing

Register           *)             ;Open the register window

Variable.Local     *)             ;Open a window with local variables

PER                               ;Open a window for the special
                                  ;function registers

Break.Set sieve                   ;Set breakpoint to function sieve

Break.Set 0x1000 /Program         ;Set a software breakpoint to address
                                  ;1000 (address 1000 is in RAM)

Break.Set 0x101000 /Program       ;Set an onchip breakpoint to address
                                  ;101000 (address 101000 is in FLASH)
```

*) These commands open windows on the screen.

Refer to the **PEDIT** command to write a script and to the **DO** command to start a script.

BDM Debugger

Warning

ESD protection

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:** • Disconnect the debug cable from the target while the target power is off. • Connect the host system, the TRACE32 hardware and the debug cable. • Start the TRACE32 software. • Connect the debug cable to the target. • Switch the target power ON. **Power down:** • Switch off the target power. • Disconnect the debug cable from the target.

Target Design Requirement/Recommendations

General

- Locate the **BDM connector** as close as possible to the processor to minimize the capacitive influence of the line length and cross coupling of noise onto the BDM signals.
- Ensure that the debugger signal ($\overline{\text{HRESET}}$) is connected directly to the $\overline{\text{HRESET}}$ of the processor. This will provide the ability for the debugger to drive and sense the status of $\overline{\text{HRESET}}$. The target design should only drive the HRESET with open collector, open drain. $\overline{\text{HRESET}}$ should not be tied to $\overline{\text{PORESET}}$, because the debugger drives the HRESET and DSCK to enable BDM operation.
- The TRACE32 internal buffer/level shifter will be supplied via the VCCS pin. Therefore it is nesecary to reduce the VCCS pull-up on the target board to a value smaller 10 Ohm.

RESET-Configuration

At HRESET the Hard Reset Configuration bits will be sampled. Depending on the $\overline{\text{RSTCONF}}$ pin the external or the internal configuration word is sampled.

$\overline{\textbf{RSTCONF}}$	**configuration word**
0	DATA[0..31] pins
1	internal data default word (0x0000 0000)

The multifunction I/O pins (VFLS0/1) have to be configured correctly for the debugging. Drive actively the following pins:

MPC5xx	DBGC(D9,D10) and DBPC(D11)
MPC8xx	DBGC(D9,D10) and DBPC(D11,D12)

There are two signal schemes possible to indicate the processor status to the debugger. Option A is recommended but Option B is also supported for the BDM functionality.

Option B is used as an alternative to eliminate pin conflicts. Option B is typically used if:

- the internal watchpoints are used
- the amount of signals must be reduced to a minimum
- the target design uses PCMCIA Port B.

Option A: Using the VFLS pins

MPC800: (DBGC=[11]; DBPC=0; FRC=x)
MPC500: (DBGC=[00,10]; DBPC=0; GPC=x)

Comment	Signal Name	PIN	PIN	Signal Name	Comment
	IPB0/IWP0/VFLS0	1	2	/SRESET	
	GND	3	4	DSCK/TCK	
	GND	5	6	IP_BI/IWP1/VFLS1	
	HRESET	7	8	DSDI/TDI	
	VCCS	9	10	DSDO/TDO	

Option B: Using the FREEZE pin

MPC800: (DBGC=[11]; DBPC=0; FRC=0)
MPC500: (DBGC=[00,10]; DBPC=0; GPC=[10,11])

Comment	Signal Name	PIN	PIN	Signal Name	Comment
	FRZ/IRQ6	1	2	/SRESET	
	GND	3	4	DSCK/TCK	
	GND	5	6	FRZ/IRQ6	
	HRESET	7	8	DSDI/TDI	
	VCCS	9	10	DSDO/TDO	

If option B is used, the **SYStem.Option.FreezePin** must be switched on

When the PowerPC's development port (BDM) is used, the JTAG functionality is disabled.

BDM Termination

T32 PU/ PD	Target PU/PD	Signal Name	PIN	PIN	Signal Name	Target PU/PD	T32 PU/PD
-	47kPU	FRZ/VFLS0	**1**	**2**	/SRESET	10kPU	-
-	-	GND	**3**	**4**	DSCK	10kPD	4k7PD
-	-	GND	**5**	**6**	FRZ/VFLS1	47kPU	-
10kPU	10kPU	HRESET	**7**	**8**	DSDI	10kPD	4k7PD
-	<10	VCCS	**9**	**10**	DSDO	>10k	-

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 target has no power.
- The pull-up resistor between the JTAG/COP[VCCS] pin and the target VCC is to 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.
- There is logic added to the JTAG/COP state machine: The debugger supports only one processor on one JTAG chain. Only the debugged processor has to be between TDI and TDO in the scan chain. No further devices or processors are allowed.
- There are additional loads or capacities on the JTAG lines.

ANY Sporadic Debug Port Fail	**The debugger crashes sporadically when a dump window is open or a system up is sometimes not possible.** Be sure that the 'VCC PIN' of the debug port connector is connected directly to the VCC of your target board. The Lauterbach debugger uses this voltage to supply a buffer that drives the debug lines to the CPU. If there is a resistor between the VCC of your board and our VCC pin, our supply voltage might drop too low.
MPC5xx Cannot write to SYPCR	**Writing SYPCR has no effect.** The SYPCR register can only be written one time. If the SYSTEM.OPTION.WATCHDOG is set to OFF then the CPU WATCHDOG function will be disabled by the debugger during a SYSTEM.UP. To disable the WATCHDOG on the CPU the debugger writes to SYPCR and uses the one-time write access to the SYPCR register.
MPC5XX/8XX Step or Go can't be Executed Successful	**Step or go result in a error message!** ... VFLS0/1 pins have wrong status. • Freeze connected? sys.o.freeze • VFLS from MIOS modul used? PU is missing 10 kOhm Right after reset VFLS pins are also inputs! State is non-recoverable!
MPC5XX/8XX With connected debugger program behaves in a different way	**With connected debugger program behaves in a different way** sys.o.ibus == debug register ibus has priority, register will be overwritten. RSTCONF for IBUS will be overwritten. sys.nodebug only will not enable the BDM interface. sys.o.freeze.off (default) assumes VFLS0/1 at BDM connector and overwrites SIUMCR bits. (MPC8XX)

MPC8260 Cannot write to SYPCR	**Writing SYPCR has no effect.** The SYPCR register can only be written one time. If the SYSTEM.OPTION.WATCHDOG is set to OFF then the CPU WATCHDOG function will be disabled by the debugger during a SYSTEM.UP. To disable the WATCHDOG on the CPU the debugger writes to SYPCR and uses the one-time write access to the SYPCR register.
MPC8xx Cannot write to SYPCR	**Writing SYPCR has no effect.** The SYPCR register can only be written one time. If the SYSTEM.OPTION.WATCHDOG is set to OFF then the CPU WATCHDOG function will be disabled by the debugger during a SYSTEM.UP. To disable the WATCHDOG on the CPU the debugger writes to SYPCR and uses the one-time write access to the SYPCR register.
MPC8XX/5XX Exceptions and Stepping	**What happens if I debug my code and an exception occurs?** The MPC8xx/5xx can react in two ways when an exception occurs: • The exception is handled by the exception handler. This way the exception is not detected by the debugger (default). • The program execution is stopped at the exception and the debug mode is entered, if the exception is enabled by the command 'TrOnchip.Set <exception>'. Refer also to the description of the Debug Enable Register in your processor manual. TRACE32 displays the reason for the program stop in the state line (refer also to the Exception Cause Register description in your processor manual). The program execution is stopped in most cases exactly at the instruction that caused the exception, in some cases at the next instruction. On some exceptions it is not possible to continue the debugging.

MPC8XX/5XX Software runs differently with ICD	**The target runs fine without the ICD attached. But with the ICD attached, the target runs for a while and then it hangs up.** If the debug mode is enabled, the serialize control bit and the instruction fetch show cycle control bits are set to SERALL after reset. In SERALL mode the processor is fetch serialized and all internal fetch cycles appear on the external bus. The processor performance is, therefore, much slower. If only a BDM debugger is used perform the command 'SYStem.Option IBUS NONE'. In NONE mode the processor works in normal mode and no show cycles are performed. There is no performance degradation in this mode. If a RISC Trace or a PowerTrace is used, perform the command 'SYStem.Option IBUS IND'. In IND mode the processor works in normal mode and show cycles are performed for all indirect changes in the program flow. The performance degradation in this mode is about 1 %. For more information refer to the description of the ISCT_SER register in your processor manual.
MPC8XX/5XX Using NOTRAP Option	**How do I use the TRAP exeption for my own application?** Use the command SYStem.Option NOTRAP ON With this setting the TRAP exception is no longer used for software breakpoints. UNDEF 0 is used instead. Use the command TrOnchip.Set PRIE OFF With this setting the debug mode is no longer entered when a TRAP occurs. See also the Debug Enable Register in you processor manual. Now your application can handle the TRAP instruction.

MPC8XX/5XX What means 'stopped by SEI'?	**Where can I find more information about the acronyms SEIE, PRIE, MCIE, ...?** These names are the abbreviation for the exceptions handler and identical to the used acronyms in the Freescale user manual. An overview and a detailed description of all possible exception handler could be found in the Freescale MPC500/800 user manual. In a debug session almost all exception could be used/enabled/configured to stop the CPU and enter the debug mode instead of running the corresponding exception handler. This could be set up in the T32 PowerView Menue: Break - OnChip_Trigger - Set - [MCIE] ;MCIE is used as example here or alternatively in the command line or script language: TrOnchip.Set [MCIE] ON If the option is enabled (box is checked) then the CPU will stop and enter the debug mode.
MPCXXX Runtime Accuracy	**When stepping with the ICD debugger, the runtime counter shows too long count values.** The runtime counter unit of the PowerPC debugger is realized using a software counter of the host and a hardware counter of the Lauterbach tool. The accuracy is about 10 us.
MPCXXX Verify Error at Single-Step or Breakpoint	**I get the error message: verify error at address ...,** By default TRACE32-ICD uses software breakpoints to set a breakpoint to an instruction. Software breakpoint means the original instruction is replaced by to TRAP in order to stop the program. This is the reason why a software breakpoint usually requires that the instruction is in RAM. Otherwise the error message verfiy error at address (address) is displayed. The reasons for these errors are: • The instruction is in ROM/FLASH/EPROM. To set software breakpoints in FLASH refer to the command 'FLASH.Auto'. • The appropriate CS is switched to ReadOnly mode. In this case it is not possible to patch the code. It is possible to use a limited number of onchip breakpoints to set a breakpoint to ROM/FLASH/EEPROM or ReadOnly memories. For more information refer to the command 'MAP.BOnchip <range>'.

General Restrictions

The CPU handles the debug mode similar to an exception.

SYStem.Option BRKNOMSK OFF: The program execution isn´t stopped as long as the processor is in a non-recoverable state (RI bit cleared in the Machine Status register).

SYStem.Option BRKNOMSK ON: The program execution can be stopped by a breakpoint even if the processor is in a non-recoverable state. Since the debug exception overwrites SRR0 and SRR1 it is not advisable to continue the debugging process.

Breakpoints

There are two types of breakpoints available: Software breakpoints and onchip breakpoints.

Software Breakpoints

Software breakpoints are the default breakpoints on instructions. Software breakpoints can be set to any instruction address in RAM and after some preparations also to instructions in FLASH. For more information refer to the command **FLASH.AUTO**.

There is no restriction in the number of software breakpoints. Please consider that increasing the number of software breakpoints will reduce the debug speed.

Onchip Breakpoints

The following list give an overview of the usage of the onchip breakpoints by TRACE32:

- **CPU family**
- **Onchip breakpoints:** Total amount of available onchip breakpoints.
- **Instruction breakpoints:** Number of onchip breakpoints that can be used for Program breakpoints.
- **Read/Write breakpoints:** Number of onchip breakpoints that can be used as Read or Write breakpoints.
- **Data breakpoints:** Number of onchip data breakpoints that can be used to stop the program when a specific data value is written to an address or when a specific data value is read from an address.

CPU family	Ochip Breakpoints	Instruction Breakpoints	Read/Write Breakpoints	Data Breakpoints
MPC500/800	4 Instruction 2 Read/Write	4	2	2

Onchip Breakpoints on Instructions

If a breakpoint is set to an instruction, a software breakpoint is used by default. If your code is in FLASH, ROM etc. you can advise TRACE32 to automatically use onchip breakpoint for specific address ranges by using the command **MAP.BOnchip** <range>.

Onchip Breakpoints on Read or Write Accesses

Onchip Breakpoints are always used, if a Read or Write breakpoint is set. For the MPC5xx/8xx it is also possible to define a specific data value. Refer to the **Break.Set** command for more information.

Example for Breakpoints

Assume you have a target with FLASH from `0` to `0xFFFFF` and RAM from `0x100000` to `0x11FFFF`. The command to configure TRACE32 correctly for this configuration is:

```
Map.BOnchip 0x0--0x0FFFFF
```

The following breakpoint combinations are possible.

Software breakpoints:

```
Break.Set 0x100000 /Program                     Software Breakpoint 1
Break.Set 0x101000 /Program                     Software Breakpoint 2
Break.Set 0xx /Program                          Software Breakpoint 3
```

Onchip breakpoints:

```
Break.Set 0x100 /Program                                Onchip Breakpoint 1
Break.Set 0x0ff00 /Program                              Onchip Breakpoint 2
Break.Set   flags /Write                                Onchip Breakpoint 3
Var.Break.Set \flags[3] /Write /DATA.Byte 0x1           Onchip Breakpoint 4
```

Simultaneous FLASH programming for MPC555

Simultaneous programming of the internal FLASH is supported for the masks K1, K2, K3 and M of the MPC555.

The MPC555 supports simultaneous programming of all 14 flash modules.

- 8 64-byte pages in the 8 blocks of FLASH module A
- 6 64-byte pages in the 6 blocks of FLASH module B

Using simultaneous FLASH programming is up to 7 times faster!

Programming procedure

1. Load the application program into the virtual memory of TRACE32-ICD.

   For the simultaneous FLASH programming the code can not directly be loaded from the host. The code has to be loaded into the virtual memory (VM) of TRACE32-ICD first.

   TRACE32-PowerView can recognize empty 64-byte pages and skip them while programming. For this reason the virtual memory should be initialized with 0xff.

```
; initialize the virtual memory of TRACE32-ICD with 0xff
Data.Set VM:<start_address_internal_flash>++0x6ffff %Long 0xffffffff

; load the code for the internal FLASH into the virtual memory
Data.LOAD.ELF <file_name> <start_address_internal_flash>++0x6ffff /
VM
```

2. Start the simultaneous programming.

```
FLASH.MultiProgram <start_address_internal_flash>++0x6ffff
```

If your application program also contains code for the external FLASH, this code has to be loaded separatly.

Memory Classes

The following memory classes are available:

Memory Class	**Description**
P	Program
D	Data
SPR	Special Purpose Register
IC	Instruction Cache (MPC8xx only)
DC	Data Cache (MPC8xx only)
NC	No Cache (only physically memory)

If the cache is disabled, memory accesses to the memory classes IC or DC are realized by TRACE32-ICD as reads and writes to physical memory.

Memory Coherency MPC8xx

Memory coherency on access to the following memory classes. If data will be set to DC, IC, NC, D or P the D-Cache, I-Cache or physical memory will be updated.

	D-Cache	**I-Cache**	**Physical Memory**
DC:	Yes	No	Yes
IC:	No	Yes	Yes
NC:	No	No	Yes
D:	Yes	Yes	Yes
P:	Yes	Yes	Yes

See also **SYStem.Option ICREAD** and **SYStem.Option DCREAD.**

MPC555/MPC553 Pin Multiplexing

CLKOUT	Always required.
A8..A29	Are always required.
D0..D11	Are required for tracing in compressed mode.
WR	Is required.
STS	Is not present when SIUMCR.DBGC== 00. In this case it is assumed that the program trace show cycle for indirect change of flow is appearing directly at the same clock where the indirect change of flow is shown. This should be always the case when running only with internal memories and having only indirect program show cycles active (no data cycles or data show cycles).
PTR	Is not present when SIUMCR.GPC !=00. In this case ALL program cycles are assumed to be program trace cycles. This is always the case when the program is running from internal memory and only indirect show cycles are enabled. When external program memory is used the trace may not be able to take the correct cycle as target for the indirect branch.
AT(2)	Is taken from the WE2/AT2 line when SIUMCR.ATWC==1 (AT0-3 lines enabled) or taken from the dedicated AT(2) line when SIUMCR.ATWC==0 (WE0-3 lines enabled) and SIUMCR.MLRC ==x1 (AT(2) function enabled). When non of the two variants is possible the debugger will assume that ALL cycles are program cycles (no data cycles). The program flow trace will not be affected by this, as long as the PTR line is available. When the AT(2) and PTR lines are both not available the trace will only work when the code is running from internal memory and only “indirect change of flow” show cycles are enabled.
VF0,VF1	Is taken from SIU when SIUMCR.DBGC==10, otherwise from the MIOS pins. MIOS must be configured when MIOS pins are used. If none of the pins are available then the program flow trace will not work. Direct cycle tracing in fully serialized mode with show cycles for all cycles will still work.
VFLS0,VFLS1	Is taken from SIU when SIUMCR.DBGC==x0, otherwise from the MIOS pins. MIOS must be configured when MIOS pins are used.
LWPx, IWPx	Optional lines. Only used when selective tracing features should be used.

Troubleshooting MPC500/MPC800 RISC Trace

Target is not running with trace attached

Some trace adapters use drivers with “Bus Hold” feature. This resistor (about 20KOhms) can pull the lines connected to the trace to VCC or Ground. If the target is using high impedance resistors to select a specific level for the reset configuration it may not work. In this case make either the resistors on the target smaller or disable the external reset configuration. Pulling down the TS line may also cause such effects. Use a pullup resistor (about 10KOhms) in this case.

Nothing recorded (number of records in Analyzer.state window remains 0)

Check that CLKOUT is available on the trace probe. Check that VFLS0 and VFLS1 are correctly configured.

No cycle information displayed in Analyzer.List

Check the TS and STS signals.

Cycle type information in Analyzer.List is wrong

Check the RW and AT lines (CT lines for MPC50x).

Address information is wrong for DRAM accesses

Define DRAM areas with **MAP.DMUX** command.

Flowtrace (Analyzer.List /FT) gives no useful results

Make sure that indirect branch program trace cycles are enabled (**SYStem.Option ICTL IND**). Check that PTR signal is correctly recorded in trace. Check for presence of VF0, VF1 and VF2 signals. Make sure that program has executed an indirect branch while sampling data for the trace.

Used Options for RiscTrace

- SYSTEM.OPTION NODATA ON /OFF
- SYSTEM.OPTION SIUMCR ON /OFF
- SIUMCR Register [DBGC,GPC] (Peripheral Window)

SYStem.Mode — Establish the communication with the CPU

Format:	**SYStem.Mode** *<mode>*
<mode>:	**Down** **NoDebug** **Go** **Attach** **Up**

Select target reset mode.

Down	Disables the Debugger. The state of the CPU remains unchanged.
NoDebug	Resets the target with debug mode disabled. In this mode no debugging is possible. The CPU state keeps in the state of NoDebug
Go	Resets the target with debug mode enabled and prepares the CPU for debug mode entry. The program execution is started then. The program execution is stopped at the next breakpoint.
Attach	Not supported.
Up	Resets the CPU, enables the debug mode and stops the CPU at the first opfetch (reset vector). All register are set to the default value.

SYStem.CPU — Select CPU type

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	*MPC5xx* | *MPC8xx*

SYStem.MemAccess Run-time memory access

Format:	**SYStem.MemAccess**	**Denied**

No run-time memory access is possible for the MPC5xx/8xx family.

SYStem.CpuAccess Run-time memory access (intrusive)

Format:	**SYStem.CpuAccess**	<mode>
<mode>:	**Enable | Denied | Nonstop**	

Enable	In order to perform an update of the memory displayed in the TRACE32 window the debugger stops the program execution about 10 times per second, switches to debug mode, updates the memory and restarts the program execution afterwards. Each short stop takes 1-100 ms depending on the speed of the debug interface and on the size of the read/write accesses required. The run-time memory access has to be activated for each window by using the memory class E: (e.g. Data.dump E:0x100) or by using the format option %E (e.g. Var.View %E var1).

Denied	No memory read or write is possible while the CPU is executing the program.
Nonstop	Nonstop ensures that the program execution can not be stopped and that the debugger doesn´t affect the real-time behaviour of the CPU. 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. spot breakpoints, performance analysis via StopAndGo, conditional breakpoints etc.)

Format:	**SYStem.BdmClock** *<rate>*
<rate>:	**EXT/4 | EXT/8 | EXT/16 |** *<fixed>*
<fixed>:	**1MHz .. 20MHz**

Selects the frequency for the debug interface. A fixed frequency or an diveded external clock can be used.

SYStem.LOADVOC — Load vocabulary for code compression

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

Load vocabulary for code compression. This is usually not required, since the vocabulary is already in the Elf file.

FLASH.MultiProgram — Simultaneous programming of on-chip FLASH

Available on:	**MPC555 (K1, K2, K3)**
Format:	**FLASH.MultiProgram** <*range*>

Allows simultaneous programming of the internal FLASH. For a complete description of the programming procedure see **Simultaneous FLASH programming for MPC555**.

SYStem.Option BRKNOMSK — Allow program stop in a non-recoverable state

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

The CPU handles debug events similar to exceptions. When a debug event (normally a break) OR an exception occurs, the CPU copies the MSR (Machine Status Register) into SRR1 (Machine Status Save/Restore Register 1) and the IP (Instruction Pointer) into SRR0 (Machine Status Save/Restore Register 1). This means, that after an exception occurred, the old values of IP and MSR are as backup in the SRR0 and SRR1 registers. If now a break happens, these values will be overwritten by the new MSR and IP values. So, it is possible to return to the exception routine and stop the processor, **but it’s not possible to return to the main program and continue the user application!** The status after the start of the exception routine is called non recoverable state.

ON	The program execution can be stopped by a breakpoint even if the processor is in a non-recoverable state. Since the debug exception overwrites SRR0 and SRR1 it is not advisable to continue the debugging process.
OFF	The program execution isn´t stopped as long as the processor is in a non-recoverable state (RI bit cleared in the Machine Status register).

SYStem.Option CCOMP — Enable code compression

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

If the code compression unit of the MPC5xx is used, this option must be switched on before the program is loaded. Then correct disassembly is possible.

SYStem.Option CLEARBE — Clear MSR[BE] on step/go

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

If the option CLEARBE is switched on, the BE bit of the MSR register will be cleared before every Go or Step.

Available on:	**MPC505, MPC509**
Format:	**SYStem.Option CSBTOR** [<**value**>]
	SYStem.Option CSBTSBOR [<**value**>]
	SYStem.Option CSBTBAR [<**value**>]
	SYStem.Option CSBTSBBAR [<**value**>]
	SYStem.Option CS0OR [<**value**>]
	SYStem.Option CS1OR [<**value**>]
	SYStem.Option CS2OR [<**value**>]
	SYStem.Option CS3OR [<**value**>]
	SYStem.Option CS4OR [<**value**>]
	SYStem.Option CS5OR [<**value**>]
	SYStem.Option CS6OR [<**value**>]
	SYStem.Option CS7OR [<**value**>]
	SYStem.Option CS8OR [<**value**>]
	SYStem.Option CS9OR [<**value**>]
	SYStem.Option CS10OR [<**value**>]
	SYStem.Option CS11OR [<**value**>]
	SYStem.Option CS0BAR [<**value**>]
	SYStem.Option CS1BAR [<**value**>]

SYStem.Option CS2BAR [<**value**>]
SYStem.Option CS3BAR [<**value**>]
SYStem.Option CS4BAR [<**value**>]

For the flow trace functionality, it is necessary for the software to know the settings of the CS unit. The values of these options must be the same values as the register values of the chip.

SYStem.Option DCREAD — Use DCACHE for data read

Format:	**SYStem.Option DCREAD** [**ON**|**OFF**]**>**

ON (Default)	If data memory is displayed (memory class D:) the memory contents from the D-cache is displayed if the D-cache is valid. If D-cache is not valid the physical memory will be read. Typical command to display data memory are: Data.dump, Var.Watch, Var.View.
OFF	If data memory is displayed (memory class D:) the memory contents from the physical memory is displayed.

SYStem.Option FAILSAVE — Special error handling for debug port

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

The debug interface of the MPC8xx and MPC5xx returns the fatal error emulation debug port fail, when reading incorrect communication data from the debug port. With this option, it is possible to suppress this debug port fail, and recover the communication. This helps debugging in noisy environment.

Available on:	**MPC8xx**
Format:	**SYStem.Option FreezePin** [**ON**|**OFF**]

As default, this option is off and the debugger set all necessary setting for the SIMCR register for the most frequently used **option A**. (VFLS0/1 pins are connected to BDM connector pin 1 and 6). The SYStem.Option.FreezePin can prevent the debugger for resetting/overwriting the SIMCR register to the default settings.

If **option B** is used (FREEZE pin is connected to the BDM connector) this SYStem.Option.FreezePin must be switched on.

Note: For the MPC5xx family all necessary configuration for the correct BDM pin setting have to be done in the RSTCONF word.

SYStem.Option IBUS — Configure the show cycles for the I-BUS

Format: **SYStem.Option IBUS** [**<Value>**]

With this option, you can set the instruction fetch showcycle and serialize control bits of the IBUS support control register.

SERALL	All fetch cycles are visible on the external bus. In this mode the processor is fetch serialized. Therefore the processor performance is much lower then working in regular mode.
SERCHG	All cycles that follow a change in the program flow are visible on the external bus. In this mode the processor is fetch serialized. Therefore the processor performance is much lower then working in regular mode.
SERIND	All cycles that follow an indirect change in the program flow are visible on the external bus. In this mode the processor is fetch serialized. Therefore the processor performance is much lower then working in regular mode.
SERNONE	In this mode the processor is fetch serialized. Therefore the processor performance is much lower then working in regular mode. No information about the program flow is visisble on the external bus.
CHG	All cycles that follow a change in the program flow are visible on the external bus. The performance degradation is small here.
IND	All cycles that follow an indirect change in the program flow are visible on the external bus. The performance degradation is small here. This setting is recommanded if a preprocessor for MPC500/800 is used.
NONE	No show cycles are performed. (Recommanded when only a BDM debugger is used.)
RESERVED	Should not be used.

SYStem.Option ICFLUSH — Flush ICACHE

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

Invalidates the instruction cache and flush the data cache before starting the target program (Step or Go).

This is required when the CACHEs are enabled and software breakpoints are set to a cached location.

MPC5xx: Flushes the Instruction Prefetch Queue before starting the program execution by `Step` or `Go`

SYStem.Option ICREAD — Use ICACHE for program read

Format:	**SYStem.Option ICREAD** [**ON**|**OFF**]>

ON	If program memory is displayed (memory class P:) the memory contents from the I-cache is shown if the I-cache is valid. If I-cache is not valid the physical memory will be read. Typical command for program memory display are: Data.List, Data.dump.
OFF (Default)	If program memory is displayed (memory class P:) the memory contents from the physical memory is displayed.

SYStem.Option LittleEnd — Control for true little endian

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

Normally, the PowerPC debugger displays data big endian style.

With this option data is displayed little endian style.

SYStem.Option MMU — MMU support

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

Enables the usage of the MMU to support **multiple** address spaces. The command should not be used if only one translation table is used. Enabling the option will extend the address scheme of the debugger by a 16 bit memory space identifier. The option can only be enabled when there are no symbols loaded.

SYStem.Option NODATA — The external data bus is not connected to trace

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

ON	No external data bus is connected to the trace connector.
OFF (Default)	The external data bus is connected to the trace connector.

SYStem.Option NOTRAP — Use alternative instruction to enter debug mode

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

ON	With this setting the TRAP exception is no longer used for software breakpoints. UNDEF 0 is used instead. Use the command **TrOnchip.Set PRIE OFF**. With this setting the debug mode is no longer entered when a TRAP occurs. See also the Debug Enable Register in you processor manual. Now your application can handle the TRAP instruction.
OFF (Default)	The TRAP exception is used for software breakpoints.

SYStem.Option PPCLittleEnd — Control for PPC little endian

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

Normally, the PowerPC debugger displays data big endian style.

With this option data is displayed in PPC little endian style.

SYStem.Option SCRATCH — Scratch for FPU access

Format: **SYStem.Option SCRATCH** <address**> | AUTO**

Reading the FPU registers of the MPC5xx requires two memory words in target memory. This option defines which location is used. The content of the memory location will be restored after use. If AUTO is used, two memory words of the on-chip RAM are used for reading the FPU registers.

SYStem.Option SIUMCR — SIUMCR setting for the trace

Format: **SYStem.Option SIUMCR** [<**value>**]

In order to trace the program and data flow, it is necessary for the TRACE32 software to know the settings of some peripheral pins. The value of this option must be the same value as the SIUMCR register of the chip.

SYStem.Option SLOWLOAD — Alternative data load algorithm

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

The debug interface of the MPC8xx and MPC5xx has a special mode for fast download of 32 bit data. For some older versions of the chips, it might be necessary to switch to a slower download mode to get proper results.

SYStem.Option SLOWRESET — Activate SLOWRESET

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

After the debugger resets the CPU (e.g. via SYStem.Up), the debugger senses $\overline{\text{HRESET}}$ for 2-3 seconds before an error message is displayed.

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

If this option is switched off, the watchdog timer of the CPU is disabled after the SYStem.Up.

Otherwise the watchdog will be periodic reseted by the debugger. **Software Watchdog Timer (SWT) —** The SWT asserts a reset or non-maskable interrupt (as selected by the system protection control register) if the software fails to service the SWT for a designated period of time (e.g, because the software is trapped in a loop or lost). After a system reset, this function is enabled with a maximum time-out period and asserts a system reset if the time-out is reached. The SWT can be disabled or its time-out period can be changed in the SYPCR. Once the SYPCR is written, it cannot be written again until a system reset.

Software Watchdog Timer (SWT) — The SWT asserts a reset or non-maskable interrupt (as selected by the system protection control register) if the software fails to service the SWT for a designated period of time (e.g, because the software is trapped in a loop or lost). After a system reset, this function is enabled with a maximum time-out period and asserts a system reset if the time-out is reached. The SWT can be disabled or its time-out period can be changed in the SYPCR. Once the SYPCR is written, it cannot be written again until a system reset.

MMU.TLB

Display MMU TLB entries

Format:	**MMU.TLB** <*tlb*>
<*tlb*>:	**IMMU** **DMMU**

Displays a table of all MMU TLB entries of the specified TLB table.

MMU.TLBSCAN

Load MMU TLB entries

Format:	**MMU.TLBSCAN** **MMU.TLBSCAN** <*tlb*>
<*tlb*>:	**IMMU** **DMMU**

Loads the TLB table entries from the CPU to the debugger internal MMU table. If no TLB table is specified, both are scanned.

TrBus.Out — Define source for the external trigger pulse

Format:	**TrBus.Out Break | ABreak | ATrigger [ON | OFF]**

Define the source for the external trigger pulse.

Break	Generate an external trigger pulse when the program exexution is stopped.
ABreak	Generate an external trigger pulse when the sampling to the trace buffer is stopped.
ATrigger	Generate an external trigger pulse when a trigger is generated for the trace. A trigger for the trace can be used to stop the sampling to the trace buffer after a specified delay **Analyzer.TDelay**.

TrBus.Set — Define the target for the incoming trigger

Format:	**TrBus.Set Break | ATrigger [ON | OFF]**

Select the target for the incoming trigger signal.

Break	Stop the program execution as soon as the external trigger signal becomes active.
ATrigger	Generate a trigger for the trace as soon as the external trigger signal becomes active. A trigger for the trace can be used to stop the sampling to the trace buffer directly or after a specified delay **Analyzer.TDelay**.

TrOnchip.CONVert — Adjust range breakpoint in onchip resource

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

The MPC5xx/MPC8xx family provides the follwing on-chip breakpoints:

MPC5xx	4 Instruction, 2 Read/Write	4 4 single I-bus breakpoints or 2 I-bus breakpoint ranges	2 2 single L-bus breakpoints or 1 L-bus breakpoint ranges	2

ON (default)	If all resources for the onchip breakpoints are already used and if the user wants to set an addtional onchip breakpoint, TRACE32 converts an onchip breakpoint set to a short address range (max. 4 bytes) to a single address breakpoint to free additional resources.
OFF	If all resources for the onchip breakpoints are already used and if the user wants to set an addtional onchip breakpoint, an error message is displayed.

Example:

```
TrOnchip.Convert ON

Break.Set 0x100++0x4 /Write                  ;Set a write breakpoint to the
                                             ;address range 0x100++0x4

Break.Set 0x800 /Write                       ;Set a write breakpoint to the
                                             ;address 0x800. The first set
                                             ;breakpoint is reduced to address
                                             ;0x100
```

Format: **TrOnchip.G.Value** *<hexmask>* | *<float>*
TrOnchip.H.Value *<hexmask>* | *<float>*
TrOnchip.G.Size [**Byte** | **Word** | **Long**]
TrOnchip.H.Size [**Byte** | **Word** | **Long**]
TrOnchip.G.Match [**OFF** | **EQ** | **NE** | **GT** | **LT** | **GE** | **LE**]
TrOnchip.H.Match [**OFF** | **EQ** | **NE** | **GT** | **LT** | **GE** | **LE**]

Defines the two data selectors of the MPC500/800 family.

OFF	Off
EQ	Equal
NE	Not equal
LE	Lower equal
GE	Greater equal
LT	Lower then
GT	Greater then
ULE	Unsigned lower equal
UGE	Unsigned greater equal
ULT	Unsigned lower then
UGT	Unsigned greater then

Example: Stop the program execution if a value between 0x50 and 0x70 is written to the variable vint.

```
Var.Break.Set vint /Alpha                ;Set a breakpoint of the type
                                         ;Alpha to vint

;Program the first L-Bus watchpoint

TrOnchip.RESet                           ;Reset onchip trigger unit

TrOnchip.LW0 LBUS Alpha                  ;The addresses marked with Alpha
                                         ;breakpoints define the L-Bus
                                         ;address

TrOnchip.LW0.CYcle Write                 ;The L-Bus cycle is write
TrOnchip.LW0.Data GANDH                ;The L-Bus data is a logical AND
                                       ;of data selector G and H

;Program the data selector G

TrOnchip.G.Value 0x50                  ;The value for G is 0x50

TrOnchip.G.Size Long                   ;The access size is Long

TrOnchip.G.Match GT                    ;The match is GreaterThen

;Program the data selector H

TrOnchip.H.Value 0x70                  ;The value for H is 0x70

TrOnchip.H.Size Long                   ;The access size is Long

TrOnchip.H.Match LT                    ;The match is LowerThen
```

TrOnchip.IWx.Count — Event counter for I-Bus watchpoint

Format:	**TrOnchip.IW0.Count** <*count*> **TrOnchip.IW1.Count** <*count*> **TrOnchip.IW2.Count** <*count*> **TrOnchip.IW3.Count** <*count*>

The occurence of the specified I-Bus event can be counted.

Example: Stop the program execution after 100. entries to INT5.

```
Break.Set INT5 /Alpha                  ;Set an Alpha breakpoint to
                                       ;the entry of INT5

TrOnchip.RESet                         ;Reset onchip trigger unit

TrOnchip.IW0.Ibus Alpha                ;The addresses marked with Alpha
                                       ;breakpoints define the I-Bus
                                       ;address

TrOnchip.IW0.Count 100.                ;The I-Bus counter is set to 100.

Go
```

TrOnchip.IWx.Ibus — Instructions address for I-Bus watchpoint

Format:	**TrOnchip.IW0.Ibus** *<selector>* **TrOnchip.IW1.Ibus** *<selector>* **TrOnchip.IW2.Ibus** *<selector>* **TrOnchip.IW3.Ibus** *<selector>*
<selector>:	**OFF** **Alpha** **Beta** **Charly** **Delta** **Echo**

Define the instruction for the I-Bus watchpoint.

TrOnchip.IWx.Watch — Activate I-Bus watchpoint pin

Format:	**TrOnchip.IW0.Watch** [**OFF** | **ON**] **TrOnchip.IW1.Watch** [**OFF** | **ON**] **TrOnchip.IW2.Watch** [**OFF** | **ON**] **TrOnchip.IW3.Watch** [**OFF** | **ON**]

ON	A pulse is generated on IWP0/IWP1/IWP2/IWP3 if the I-Bus watchpoint is hit. The processor pins IWP0/IWP1/IWP2/IWP3 serve multiple functions. Please check your target hardware to find out which pin can be used for the trigger pulse. The smallest pulse lenght is one clock cylcle.
OFF	The program execution is stop on a hit of the L-Bus watchpoint.

Example: Generate a pulse on IW0 when the function func5 is entered. Generated a pulse on IW1 on the exit of func5.

```
Break.Set func5 /Alpha                ;Set an Alpha breakpoint to the
                                      ;entry of func5

Break.Set v.end(func5)-3 /Beta        ;Set a Beta breakpoint to the exit
                                      ;of func5

TrOnchip.RESet                        ;Reset the onchip trigger unit
TrOnchip.IWO.Ibus Alpha                ;The addresses marked with Alpha
                                      ;breakpoints define the Ibus
                                      ;address

TrOnchip.IWO.Watch ON                  ;Generate a pulse on IWP0 when
                                      ;IW0 is hit

TrOnchip.IW1.Ibus Beta                 ;The addresses marked with Beta
                                      ;breakpoints define the Ibus
                                      ;address

TrOnchip.IW1.Watch ON                  ;Generate a pulse on IWP1 when
                                      ;IW1 is hit
```

TrOnchip.LW0.Count — Event counter for L-Bus watchpoint

Format: **TrOnchip.LW0.Count** *<count>*
TrOnchip.LW1.Count *<count>*

The occurence of the specified L-Bus event can be counted.

Example: Stop the program execution after 100. write accesses to flags[3].

```
Var.Break.Set flags[3] /Alpha          ;Set an Alpha breakpoint to
                                       ;flags[3]

TrOnchip.RESet                         ;Reset onchip trigger unit

TrOnchip.LW0.Lbus Alpha                ;The addresses marked with Alpha
                                       ;breakpoints define the L-Bus
                                       ;address

TrOnchip.LW0.CYcle Write               ;The L-Bus cycle is write

TrOnchip.LW0 Count 100.                ;The L-Bus counter is set to 100.

Go
```

TrOnchip.LW0.CYcle — Cycle type for L-Bus watchpoint

Format:	**TrOnchip.LW0.CYcle** *<cycle>* **TrOnchip.LW1.CYcle** *<cycle>*
<cycle>:	**Read** **Write** **Access**

Define the cycle type for the L-Bus watchpoint.

TrOnchip.LW0.Data — Data selector for L-Bus watchpoint

Format:	**TrOnchip.LW0.Data** *<selector>* **TrOnchip.LW1.Data** *<selector>*
<selector>:	**OFF** **G** **H** **GANDH** **GORH**

Define the data selector for the L-Bus watchpoint.

TrOnchip.LW0.Ibus — Instructions address for L-Bus watchpoint

Format:	**TrOnchip.LW0.Ibus** *<selector>* **TrOnchip.LW1.Ibus** *<selector>*
<selector>:	**OFF** **Alpha** **Beta** **Charly** **Delta** **Echo**

Define the instruction for the L-Bus watchpoint.

Example: Stop the program execution if func5 writes to flags[3].

```
Var.Break.Set func5 /Alpha              ;Set an Alpha breakpoint to the
                                        ;complete range of func5

Var.Break.Set flags[3] /Beta            ;Set a Beta breakpoint to flags[3]

TrOnchip.RESet                          ;Reset onchip trigger unit

TrOnchip.LW0.Ibus Alpha                 ;The addresses marked with Alpha
                                        ;breakpoints define the instruction
                                        ;address for LW0

TrOnchip.LW0.Lbus /Beta                 ;The addresses marked with Beta
                                        ;breakpoints define the data
                                        ;address for LW0

TrOnchip.LW0.CYcle Write                ;The data cycle is write
```

TrOnchip.LW0.Lbus — Data address for the L-Bus watchpoint

Format:	**TrOnchip.LW0.Lbus** <*selector*> **TrOnchip.LW1.Lbus** <*selector*>
<*selector*>:	**OFF** **Alpha** **Beta** **Charly** **Delta** **Echo**

Defines on which data address for the L-Bus watchpoint.

TrOnchip.LW0.Watch — Activate L-Bus watchpoint pin

Format:	**TrOnchip.LW0.Watch** [**OFF** | **ON**] **TrOnchip.LW1.Watch** [**OFF** | **ON**]

ON	A pulse is generated on LWP0/LWP1 if the L-Bus watchpoint is hit. The processor pins LWP0/LWP1 serve multiple functions. Please check your target hardware to find out which pin can be used for the trigger pulse. The smallest pulse lenght is one clock cylcle.
OFF	The program execution is stop on a hit of the L-Bus watchpoint.

TrOnchip.RESet — Reset onchip trigger unit

Format:	**TrOnchip.RESet**

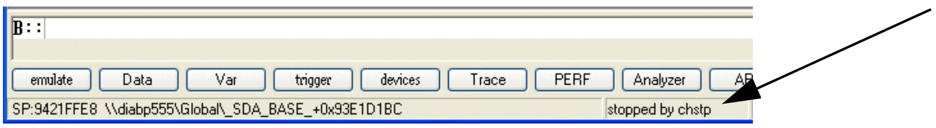
Reset the onchip trigger unit.

TrOnchip.Set — Stop program execution at specified exception

Format:	**TrOnchip.Set** <*item*> [**OFF** | **ON**]
<*item*>:	**CHSTPE** .. **SEIE** (only MPC500/800)

The program execution is stopped at the specified exception. For more details refer to the Debug Enable Register in your processor manual.

If the program execution is stopped at an exception, the name of the exception is displayed in the state line.



SP:9421FFE8 \\diabp555\Global_SDA_BASE_+0x93E1D1BC | stopped by chstp

TrOnchip.TCOMPRESS — Trace data compression

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

Not implemented yet.

TrOnchip.TEnable — Set filter for the trace

Format:	**TrOnchip.TEnable** *<par>*

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

TrOnchip.TOFF — Switch the sampling to the trace to OFF

Format:	**TrOnchip.TOFF**

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

TrOnchip.TON — Switch the sampling to the trace to ON

Format:	**TrOnchip.TON EXT** | **Break**

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

TrOnchip.TTrigger

Set a trigger for the trace

Format: **TrOnchip.TTrigger** <*par*>

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

TrOnchip.VarCONVert

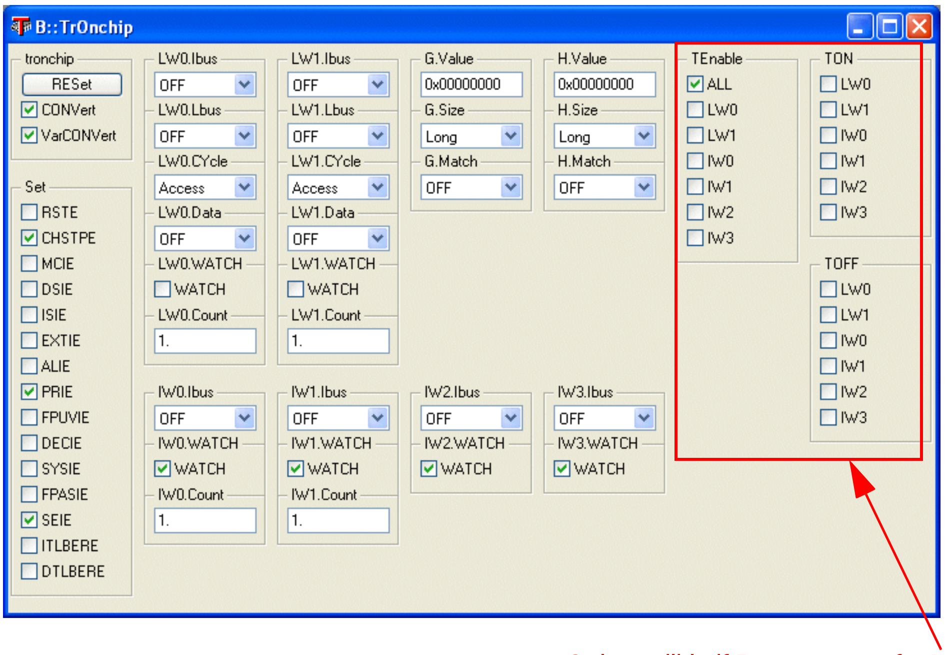
Adjust hll breakpoint in onchip resource

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

Command is of no relevance for the MPC5xx/8xx family.

Format:	**TrOnchip.view**

Display the TrOnchip window.


B::TrOnchip

tronchip: RESet; [✓] CONVert; [✓] VarCONVert

Set: [ ] RSTE; [✓] CHSTPE; [ ] MCIE; [ ] DSIE; [ ] ISIE; [ ] EXTIE; [ ] ALIE; [✓] PRIE; [ ] FPUVIE; [ ] DECIE; [ ] SYSIE; [ ] FPASIE; [✓] SEIE; [ ] ITLBERE; [ ] DTLBERE

LW0.Ibus: OFF; LW0.Lbus: OFF; LW0.CYcle: Access; LW0.Data: OFF; LW0.WATCH: [ ] WATCH; LW0.Count: 1.

LW1.Ibus: OFF; LW1.Lbus: OFF; LW1.CYcle: Access; LW1.Data: OFF; LW1.WATCH: [ ] WATCH; LW1.Count: 1.

G.Value: 0x00000000; G.Size: Long; G.Match: OFF

H.Value: 0x00000000; H.Size: Long; H.Match: OFF

TEnable: [✓] ALL; [ ] LW0; [ ] LW1; [ ] IW0; [ ] IW1; [ ] IW2; [ ] IW3

TON: [ ] LW0; [ ] LW1; [ ] IW0; [ ] IW1; [ ] IW2; [ ] IW3

TOFF: [ ] LW0; [ ] LW1; [ ] IW0; [ ] IW1; [ ] IW2; [ ] IW3

IW0.Ibus: OFF; IW0.WATCH: [✓] WATCH; IW0.Count: 1.

IW1.Ibus: OFF; IW1.WATCH: [✓] WATCH; IW1.Count: 1.

IW2.Ibus: OFF; IW2.WATCH: [✓] WATCH

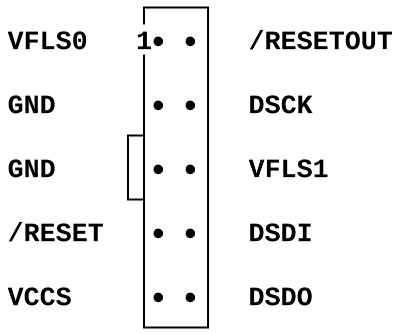
IW3.Ibus: OFF; IW3.WATCH: [✓] WATCH

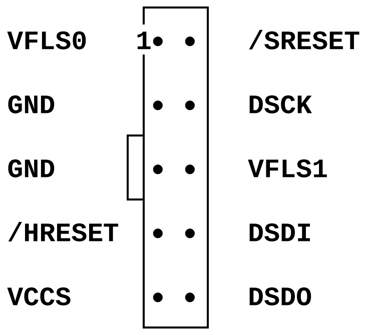
Only availble if Preprocessor for MPC500/800 is used

BDM Connector

Mechanical Description

BDM Connector MPC500/MPC800





Support

Available Tools

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
MGT560			YES		YES		YES
MPC533			YES		YES		YES
MPC534			YES		YES		YES
MPC535			YES		YES		YES
MPC536			YES		YES		YES
MPC555			YES		YES		YES
MPC556			YES		YES		YES
MPC561			YES		YES		YES
MPC562			YES		YES		YES
MPC563			YES		YES		YES
MPC564			YES		YES		YES

CPU	ICE	FIRE	ICD DEBUG	ICD MONITOR	ICD TRACE	POWER INTEGRATOR	INSTRUCTION SIMULATOR
MPC565			YES		YES		YES
MPC566			YES		YES		YES
MPC821	YES	YES	YES		YES		YES
MPC823		YES	YES		YES		YES
MPC850		YES	YES		YES		YES
MPC852T			YES		YES		YES
MPC855		YES	YES		YES		YES
MPC859DSL			YES		YES		YES
MPC859T			YES		YES		YES
MPC860	YES	YES	YES		YES		YES
MPC862		YES	YES		YES		YES
MPC866P			YES		YES		YES
MPC866T			YES		YES		YES
MPC870			YES		YES		YES
MPC875			YES		YES		YES
MPC880			YES		YES		YES
MPC885			YES		YES		YES

Compilers

Language	Compiler	Company	Option	Comment
C	D-CC	Diab-Data	IEEE	
C	D-CC	Diab-Data	COFF	
C	D-CC	Diab-Data	ELF/DWARF	
C	CC	Freescale	XCOFF	
C	XCC-V	Gaio	SAUF	
C	GREEN HILLS C	Greenhills	ELF/DWARF	
C	GCC	HighTec	ELF/DWARF	
C	MCCPPC	Mentor Graphics	ELF/DWARF	
C	HIGH-C	Metaware	ELF/DWARF	
C	CODEWARRIOR	Metrowerks	ELF/DWARF	
C	ULTRA C	Microware	ROF	
C	DCPPC	TASKING	ELF/DWARF	
C++	D-C++	Diab-Data	ELF/DWARF	
C++	GCC	FSF	ELF/DWARF	
C++	GREEN HILLS C++	Greenhills	ELF/DWARF	
C++	CCCPPC	Mentor Graphics	ELF/DWARF	
C++	HIGH-C++	Metaware	ELF/DWARF	
C++	MSVC	Microsoft	EXE/CV5	WindowsCE
C++	GCCPPC	Wind River Systems	ELF/STABS	
GCC	GCC	FSF	ELF/DWARF	
JAVA	FASTJ	Diab-Data	ELF/DWARF	

Realtime Operation System

Name	Company	Comment
OSEK	-	via ORTI
ProOSEK	3Soft	via ORTI
AMX	KADAK Products	
ChorusOS	Sun Microsystems	
CMX-RTX	CMX Company	
CodeWarriorOSEK	Freescale	via ORTI/former MetrowerksOSEK
ECOS	eCosCentric Limited	1.3.1 and 2.0
ERCOSEK	ETAS GmbH	via ORTI
Linux	-	Kernel Version 2.4 and 2.6
Linux	MontaVista	Version 3.0 and 3.1
LynxOS	LynuxWorks	3.1.0, 3.1.0a, 4.0
MQX	MQX Embedded	2.40 and 2.50
NORTi	MISPO	
Nucleus PLUS	Accelerated Tech.	
OSE Delta	Enea OSE Systems	4.x up to 5.2
PikeOS	Sysgo AG	
pSOS+	Integrated Systems	2.1 to 2.5, 3.0, with TRACE32
QNX	QNX Software Systems	6.0 to 6.3
RTXC 3.2	Quadros Systems Inc.	
RTXC Quadros	Quadros Systems Inc.	
SMX	Micro Digital	3.4 to 3.7
ThreadX	Express Logic	3.0, 4.0, 5.0
uC/OS-II	Micrium Inc.	2.0 to 2.7
VRTXsa	Mentor Graphics	
VxWorks	Wind River Systems	5.x and 6.x

Debuggers

CPU	Debugger	Company	Host
ALL	EASYCASE	BKR GmbH	Windows
ALL	X-TOOLS / X32	blue river software	Windows
ALL	ECLIPSE	Eclipse.org	Windows
ALL	ATTOL TOOLS	MicroMax	Windows
ALL	VISUAL BASIC INTERFACE	Microsoft	Windows
ALL	CODEWRIGHT	Premia Corporation	Windows
ALL	DA-C	RistanCASE	Windows

CPU	Debugger	Company	Host
ALL	RHAPSODY IN MICROC	Telelogic	Windows
ALL	WINDOWS CE PLATF. BUILDER	Windows	Windows
POWERPC	GR228X IC-TESTSYSTEME	Battefeld GmbH	Windows
POWERPC	OSE ILLUMINATOR	Enea OSE Systems AB	Windows
POWERPC	DIAB RTA SUITE	WindRiver Systems	Windows

Products

Product Information

OrderNo Code	Text
LA-7722 BDM-MPC500/800	**BDM Debugger for MPC500/800 (ICD)** supports PowerPC MPC505, MPC555, MPC56X MPC801, MPC821, MPC85x, MPC86x, MPC87x and MPC88x includes HLL debugger, operation system, cable includes driver for Windows 3.11, Windows NT, Windows 95/98, Windows2000, Windows-XP requires PODBUS-interface to host and universal Debug Module (Processor BDM inputsignals have to be 3.3V tolerant.)

Order Information

Order No.	Code	Text
LA-7722	**BDM-MPC500/800**	**BDM Debugger for MPC500/800 (ICD)**