





ZSP Debugger

TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

| | |
|---|---|
| TRACE32 Documents |  |
| ICD In-Circuit Debugger |  |
| Processor Architecture Manuals |  |
| ZSP |  |
| ZSP Debugger | 1 |
| Introduction | 5 |
| Brief Overview of Documents for New Users | 5 |
| Demo and Start-up Scripts | 5 |
| ESD Protection | 6 |
| FAQ | 6 |
| Quick Start JTAG | 7 |
| Troubleshooting | 9 |
| System Up Errors | 9 |
| System Startup for EB403 | 10 |
| System Startup for EB500P (ZSP500) | 10 |
| Hardware and Software Debug Modes | 11 |
| ZSP500 Hardware Debug Mode | 11 |
| ZSP500 Software Debug Mode (Default) | 12 |
| ZSP500 Mixed Software and Hardware Debug Mode | 12 |
| Breakpoints | 13 |
| Breakpoints in ROM (ZSP500) | 13 |
| CPU specific TrOnchip Commands | 15 |
| TrOnchip.CONVert | Adjust range breakpoint in on-chip resource 15 |
| TrOnchip.RESet | Set on-chip trigger to default state 15 |
| TrOnchip.state | Display on-chip trigger window 15 |
| Memory Access | 16 |
| Memory Addressing (ZSP400) | 16 |
| Memory Addressing (ZSP500) | 17 |
| CPU specific SYStem Settings | 18 |
| SYStem.CPU | Selects the processor type 18 |
| SYStem.JtagClock | Selects the frequency for the debug interface 19 |
| SYStem.MemAccess | Select run-time memory access method 20 |

| | | |
|--|---|-----------|
| SYStem.Mode | Selects target reset mode | 21 |
| SYStem.CONFIG | Configure debugger according to target topology | 24 |
| SYStem.Option.ADIAFTERBREAKIN | Handling of external breakpoints | 29 |
| SYStem.Option.BREAKOUTAFTERSWBREAK | Creating a break-out signal | 30 |
| SYStem.Option.EnReset | Allow the debugger to drive nRESET (ZSP400) | 30 |
| SYStem.Option.HardwareDebug | Select debug mode (ZSP5xx) | 31 |
| SYStem.Option.IBOOT | Configure IBOOT board signal (ZSP5xx) | 31 |
| SYStem.Option.IMASKASM | Disable interrupts while ASM single stepping | 31 |
| SYStem.Option.IMASKHLL | Disable interrupts while HLL single stepping | 32 |
| SYStem.Option.IntelSOC | Slave core is part of Intel® SoC | 32 |
| SYStem.Option.MEMDEU | Memory access via DEU (ZSP5xx) | 32 |
| SYStem.Option.RisingTDO | TDO sampled on rising TCK edge (LSI402ZX) | 33 |
| SYStem.Option.SLOWRESET | Slow reset | 33 |
| SYStem.Option.SVTADDR | Configure SVTADDR (ZSP500) | 33 |
| Multicore Debugging | | 34 |
| SYStem.LOCK | Lock debug port (ZSP400) | 34 |
| Design Decisions and Limitations (ZSP5xx) | | 35 |
| Disassembler | | 35 |
| Timer0, Timer1 Registers | | 35 |
| Single Stepping over RETI Fails | | 35 |
| Software Breakpoints in CEXE Blocks | | 35 |
| On-chip Breakpoints (ZSP500 Hardware Erratum) | | 36 |
| Software Debug Mode and Hardware Debug Mode | | 36 |
| Simulator Interface for ZSP5xx Cores | | 37 |
| Limitations of the Simulators | | 37 |
| File I/O with Simulator Targets | | 37 |
| Performance Measurements with Simulator Interface | | 38 |
| JTAG Connector | | 39 |
| Mechanical Description of 20-pin Debug Cable for ZSP400/ZSP500 | | 39 |
| Mechanical Description of JTAG Connector for ZSP400 (obsolete) | | 39 |
| Technical Data | | 40 |
| Operation Voltage | | 40 |

B::SYStem

Mode

Down

NoDebug

Go

Attach

StandBy

Up (Stand

Up

CPU

LSI402ZX

MemAccess

CPU

Denied

CpuAccess

Enable

Denied

Nonstop

JtagClock

5.0MHz

Option

IMASKASM

IMASKHLL

SLOWRESET

MultiCore

B::Var.Frame /Locals /Caller

-000>sieve()

()i = 0x0000

()primz = 0x0003

()k = 0x01C8

()anzahl = 0x0002

end of frame

| B::Data.List | |
|--------------|---|
| addr/line | source |
| | int anzahl; |
| 684 | anzahl = 0; |
| 686 | for (i = 0 ; i <= SIZE ; flags[i++] = TRUE) ; |
| 688 | for (i = 0 ; i <= SIZE ; i++) |
| | { |
| 690 | if (flags[i]) |
| | { |
| 692 | primz = i + i + 3; |
| 693> | k = i + primz; |
| 694 | while (k <= SIZE) |
| | { |
| 696 | flags[k] = |
| 697 | k += primz; |
| 698 | }; LSE; |
| 699 | anzahl++; |
| | } |
| | } |
| 703 | return anzahl; |

Introduction

Please keep in mind that only the **Processor Architecture Manual** (the document you are reading at the moment) is CPU specific, while all other parts of the online help are generic for all CPUs supported by Lauterbach. So if there are questions related to the CPU, the Processor Architecture Manual should be your first choice.

Brief Overview of Documents for New Users

Architecture-independent information:

- **“Training Basic Debugging”** (training_debugger.pdf): Get familiar with the basic features of a TRACE32 debugger.
- **“T32Start”** (app_t32start.pdf): T32Start assists you in starting TRACE32 PowerView instances for different configurations of the debugger. T32Start is only available for Windows.
- **“General Commands”** (general_ref_<x>.pdf): Alphabetic list of debug commands.

Architecture-specific information:

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

Demo and Start-up Scripts

Lauterbach provides ready-to-run start-up scripts for known ZSP based hardware.

To search for PRACTICE scripts, do one of the following in TRACE32 PowerView:

- Type at the command line: **WELCOME.SCRIPTS**
- or choose **File** menu > **Search for Script**.

You can now search the demo folder and its subdirectories for PRACTICE start-up scripts (*.cmm) and other demo software.

You can also manually navigate in the ~/demo/zsp/ subfolder of the system directory of TRACE32.

| | |
|-----------------|--|
| WARNING: | <p>To prevent debugger and target from damage it is recommended to connect or disconnect the Debug Cable only while the target power is OFF.</p> <p>Recommendation for the software start:</p> <ol style="list-style-type: none">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. <p>Power down:</p> <ol style="list-style-type: none">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. |
|-----------------|--|

FAQ

Please refer to <https://support.lauterbach.com/kb>.

Quick Start JTAG

Starting up the Debugger is done as follows:

1. Select the device prompt B: for the ICD Debugger.

```
b:
```

If you are working with the PODPC card, a Podbus-Ethernet interface or a Podbus-Parallel adapter device `b: :` is already selected.

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

```
SYStem.CPU LSI402ZX  
SYStem.o.RisingTDO ON // only for LSI402ZX
```

3. Tell the debugger where's FLASH/ROM on the target.

```
MAP.BOnchip 0x0f800++0xFFFF
```

This command is necessary for the use of on-chip breakpoints.

4. Enter debug mode

```
SYStem.Up
```

This command resets the CPU (RESET) and enters debug mode. after this command is executed, it is possible to access the CPU registers.

5. Load the program.

```
Data.LOAD.Elf diabc.x ; elf specifies the format,  
; diabc.x is the file name
```

The option of the `Data.LOAD` command depends on the file format generated by the compiler. For information on the compiler options refer to the section [Compiler](#). A detailed description of the `Data.LOAD` command is given in the [“General Commands Reference”](#).

The start-up can be automated using the PRACTICE script programming language. A typical start sequence is shown below. This sequence can be written to a PRACTICE script file (*.cmm, ASCII format) and executed with the command **DO** <file>.

```
B::                                ; Select the ICD device prompt

WinCLEAR                          ; Clear all windows

MAP.BOnchip 0x0000++0xFFFF        ; Specify where's ROM

SYStem.cpu LSI402ZX               ; Select the processor type

SYStem.o.RisingTDO ON             ; Create specific option; only LSI402ZX

SYStem.Up                        ; Reset the target and enter debug mode

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

Data.List                        ; Open disassembly window *)

Register.view /SpotLight          ; Open register window *)

Frame.view /Locals /Caller        ; Open the stack frame with
                                ; local variables *)

Var.Watch %Spotlight flags       ; Open watch window for variables *)
ast

PER.view                        ; Open window with peripheral register **)

Break.Set sieve                  ; Set breakpoint to function sieve

Break.Set 0x1200 /Program         ; Set software breakpoint to address 1200
                                ; (address 1200 is in internal memory)

Break.Set 0xf900 /Program         ; Set on-chip breakpoint to address
                                ; 0xf900 (address 0xf900 is in ROM)
```

*) These commands open windows on the screen. The window position can be specified with the **WinPOS** command.

**) There is no predefined peripheral file for ZSP500 as there is no standard peripheral register mapping.

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.

| | |
|----------|---|
| All | The target has no power. |
| All | The target is in reset: The debugger controls the processor reset and uses the RESET line to reset the CPU on every SYStem.Up. |
| All | There is logic added to the JTAG state machine: By default the debugger supports only one processor on one JTAG chain. If the processor is member of a JTAG chain the debugger has to be informed about the target JTAG chain configuration. See Multicore Debugging. |
| All | There are additional loads or capacities on the JTAG lines. |
| All | The pull-up resistor between the JTAG[VCCS] pin and the target VCC is too large |
| LSI402ZX | The system option “RisingTDO” is switched OFF (default!). It must be switched ON before a system up will be done. |
| LSI403.. | The system option “RisingTDO” is switched ON. It must be switched OFF before a system up will be done. |
| EB403 | The system option “EnReset” is switched on (default!). Switch the EnReset option OFF, reset the target by pressing the reset button for at least 2 s and do a system up afterwards. See note below. |

System Startup for EB403

For the EB403 of LSI `SYStem.Up` will not work, if the system option “enreset” was not switched OFF before. This is because the reset signal influences the TRST signal and prevents the target from booting. Therefore the EB403LP requires a special boot sequence which does not assert the reset signal.

1. ; Connect JTAG cable, while target power is off
2. ; Start debugger
3. `SYS.O ENRESET OFF` ; Disable reset
4. ; Switch target power on
5. `SYStem.Up` ; Do system up of target via debugger

After the first `SYStem.Up`, subsequent `SYStem.Up` commands will only succeed after resetting the board by pressing the reset button for at least 2 s.

System Startup for EB500P (ZSP500)

The EB500P evaluation board requires up to 1000 ms delay after releasing the reset line in order to initialize. For taking into account this delay, enable the option for slow reset:

```
sys.option SLOWRESET ON
```

In the default configuration the EB500P will boot from Flash at 0xF800. For correct debugger operation the `IBOOT` and `SVTADDR` variables need to be configured:

```
sys.option IBOOT OFF
sys.option SVTADDR 0xf800
```

If configured incorrectly, the debugger may show internal instruction memory contents (RAM) in the range of the `IBOOT` windows (0xF800-0xFFFF) instead of the Flash memory which is accessed by the core.

In many configurations the EB500P will boot a demo program with flashing LEDs. As this program is located in Flash memory it is not possible to set SW breakpoints and therefore it is not possible to single step this program.

NOTE: The EB500P eval board is equipped with a 20 pin (J13) and a 10 pin (J14) connector for attaching a JTAG debugger. Which of these connectors is actually connected to the core depends on the FPGA configuration (depends on board versions). In case of problems attaching the debugger to the target, please try the other connector.

Hardware and Software Debug Modes

The ZSP cores support two different debug modes:

- Hardware debug mode (HWD) and
- Software debug (SWD) mode.

In SWD mode there must be a debug monitor (DMC) in the target. Technically the DMC is an exception handler which is called when a debug interrupt request is performed by the debugger (e.g. for stopping the core) or as result of program execution (e.g. when hitting a SW breakpoint).

In HWD mode the core resources (registers, memories, ...) are accessed using a hardware device emulation unit (DEU), obviating the need for a SW debug monitor. Using the hardware debug mode requires that the core has a register scan chain (which is not always the case) and that TRACE32 knows its layout.

For ZSP400 only SWD mode and SW breakpoints are supported by TRACE32. There is no support for on-chip breakpoints.

For ZSP500 both modes are supported. The mode is selected by **SYSystem.Option.HardwareDebug**. As HWD mode is specific for **each implementation** and sometimes even depends on the chip revision of a ZSP500 core, the correct core needs to be selected before activating HWD mode. Currently only EB500P is supported in HWD mode.

If you want to debug your core in HWD mode, please **check that the core actually supports HWD mode** (there are ZSP5xx implementations without a scan chain) and contact LAUTERBACH support.

| Selection with sys.cpu | ; Supported Mode |
|------------------------------|------------------|
| ZSP40x | ; SWD |
| ZSP500, ZSP540, ZSP600 | ; SWD |
| EB500P (from LSI Eval Board) | ; SWD + HWD |

ZSP500 Hardware Debug Mode

The main advantage of the HWD mode is that no debug monitor program is required in the target. Also the HWD mode works even if the core is in a state that does not allow it to enter the DMC (e.g. in halt mode). The hardware debug mode requires that the debugger knows the exact layout of the register scan chain. This layout is specific for each core version and describe in a register map file. The mapfile for ZSP500 (from EB500P eval board, COREID 0x0500216D) is known to T32. **For using other cores in HW debug mode please contact LAUTERBACH support for instructions.**

The main drawback of HWD mode is that the shown PC is imprecise. The PC shown is that of the RD pipeline stage, not that of the EX stage. As the core can have up to 32 instructions between the pipeline stages RD and EX, the PC may be wrong by that number of instructions. In general the shown PC will be ahead of actual program execution. The instruction indicated will not necessarily be on the execution path of the core (e.g. in case of conditional branches).

On-chip breakpoints may be used in HW debug mode. Due to their implementation, the core will not stop exactly at the requested instruction (IA breaks) or the when the specified data (DAB, DAV breaks) is accessed. Software breakpoints are not supported in HWD mode.

Single stepping cannot be implemented in HWD mode due to imprecise on-chip breakpoints and program counter (PC). SW breakpoints cannot be used in HWD mode due to the lack of the DMC.

Setting the program counter (PC) is not reliable in hardware mode (ZSP500 hardware issue). In many cases setting the PC will succeed, in other cases the core will ignore the new PC. This behavior seems to be related to the instructions currently in the pipeline. Conditional branches seem to cause problems. For reliably setting the PC proceed as follows (provided there is a DMC in the target):

```
sys.o.hardwaredebug off          ; Switch to SWD mode

r.s PC 0x1234                    ; Set the PC

sys.o.hardwaredebug on          ; Switch back to HWD mode
```

Despite these limitations the HWD mode is useful in particular situations where the SWD mode does not function anymore e.g. when the core entered halt mode and will not execute the DMC anymore. In that case you can switch the debugger into HWD mode and analyze registers and memory. Generally the SWD mode is more convenient than HWD and the latter is employed as a last resort if it fails.

ZSP500 Software Debug Mode (Default)

The advantages of the SWD mode are the unlimited number of (software) breakpoints and the fact that the shown program counter (PC) is precise i.e. it corresponds to the next instruction to be executed. Due to pipelining effects the latter is not the case in hardware debug mode. Software breakpoints are implemented by writing a breakpoint instruction at the requested address and require writable memory.

In SW debug mode the TPC (trap program counter) register is not visible because the PC of the interrupted instruction is passed in the TPC register. The actual PC register points to the SW Debug Monitor and is irrelevant for the debugged program.

ZSP500 Mixed Software and Hardware Debug Mode

For targets that support hardware debug mode, TRAC32 supports the unique so-called mixed hardware and software debug mode. In this debug mode, TRACE32 uses a debug monitor (DMC) as in software debug mode but at the same time allows to use OnChip breakpoints.

| | |
|--------------|--|
| NOTE: | The mixed mode only works for targets that also support the hardware debug mode. |
|--------------|--|

Breakpoints

There are two types of breakpoints available: Software breakpoints and on-chip breakpoints (ZSP500 only).

For SW breakpoints there must be writable memory at the required address. As single stepping is implemented via SW breakpoints, it is not possible to step through code in Flash or ROM.

On-chip breaks are not entirely precise and may stop the core too late: the PC may have advanced beyond the instruction causing the break. SW breakpoints do not have this problem.

Using on-chip breaks in SW mode is implemented by switching the core from HWD mode (entered after hitting the on-chip break) into SWD mode. This transition will take effect after the pipeline is empty i.e. with some delay (normally less than 8 instructions). Therefore the core will stop beyond the breakpoint e.g. inside a subroutine even if the break was set on the `call` instruction. Despite this, the shown PC is always precise in SW mode i.e. points to the next instruction to be executed.

NOTE: the core might encounter additional, closely located on-chip breaks while draining the pipeline. As this would disturb the draining process, the following on-chip breaks are disabled while switching from HWD to SWD mode (and re-enabled before continuing program execution later).

Breakpoints in ROM (ZSP500)

With the command **MAP.BOnchip** *<range>* it is possible to define address ranges in which the debugger uses on-chip breakpoints on default. Commonly the command is used to inform the debugger about ROM (FLASH, EPROM) memories on the target. If a breakpoint is set within the specified address range the debugger uses automatically the available on-chip breakpoints.

Alternatively on-chip breakpoints can be set using the `/OnChip` parameter:

```
break.set 0xf890 /OnChip
```

Assuming a target with ROM from 0xF800 to 0xFFFF, Flash memory from 0x10.0000 to 0x10.FFFF, and otherwise internal memory or RAM. The commands to configure TRACE32 correctly for this configuration are:

```
Map.BOnchip 0xf800++0x0xffff  
Map.BOnchip 0x100000++0x10ffff
```

Based on these settings, TRACE32 will automatically select the appropriate type of breakpoint (on-chip or software) for each address as shown below:

```
Break.Set 0x4000 /Program           ; Software Breakpoint 1  
Break.Set 0xf750 /Program           ; Software Breakpoint 2  
Break.Set 0x1100000 /Program        ; Software Breakpoint 3
```

```
Break.Set 0xf810 /Program          ; On-chip Breakpoint (in ROM)
Break.Set 0x10100 /Program         ; On-chip Breakpoint (in Flash)
```

TrOnchip.CONVert

Adjust range breakpoint in on-chip resource

Format:

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

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

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

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

TrOnchip.RESet

Set on-chip trigger to default state

Format:

TrOnchip.RESet

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

TrOnchip.state

Display on-chip trigger window

Format:

TrOnchip.state

Opens the TrOnchip.state window.

Memory Access

The following memory classes are available:

| Memory Class | Description |
|--------------|----------------|
| P | Program memory |
| D | Data memory |

Whether the debugger accesses internal or external memory is determined by the current setting of the %smode register bits DDR (disable internal data RAM) and DIR (disable internal instruction RAM). Therefore the debugger will always show the memory contents that would be accessed by the core in its current state. Be sure to configure [SYStem.Option.IBOOT](#) and [SYStem.Option.SVTADDR](#) for taking into account the IBOOT window.

Memory Addressing (ZSP400)

Internally the memory addresses are 16 bit addresses. Therefore the ZSP400 addresses always the range 0x0000 - 0xffff. Since the ZSP400 may have up to 5 instruction memories and up to 5 data memories with the same address range the address will be expanded to a 8 digit address. The TRACE32 debugger uses the same address format as the debugger of the SDK development kit.

The address format is as follows:

| Digit 7 | Digit 6 | Digit 5 | Digit 4 | Digits 0-3 |
|---------|---------|---------|------------|------------|
| 0 | pageno | int/ext | instr/data | address |

where

| Digit /Name | Description |
|-------------|--|
| 7 | not relevant |
| 6 / page | external page number (stored in mempcr) |
| 5 / ext | 1 to access external memory 0 to access internal memory |
| 4 / instr | 0 to access instruction memory 2 to access data memory |
| 0..3 | 16 bit address |

Address examples:

```
Data.Set 0x1000 0x0          ; writes 0x0 into internal instruction
                             ; memory at address 0x1000

Data.Set 0x024000 0x1234     ; writes 0x1234 into internal data memory
                             ; at address 0x4000

Data.Set 0x102000 0xf        ; writes 0xf into external instruction
                             ; memory page 0

Data.Set 0x2102500 0xa611    ; writes 0xa611 into external instruction
                             ; memory page 2 at address 0x2500

Data.Set 0xc120000 0x0       ; writes 0x0 into external data memory
                             ; page 0xc at address 0x0

Data.Set 0x12a000 0x55       ; writes 0x55 into external memory page 0
                             ; at address 0xa000
```

Trying to access memory not belonging to the memory map of the processor will be refused with the error message

```
no memory mapped at address D:0002000
```

Memory Addressing (ZSP500)

In ZSP500 mode, there is no special interpretation of the upper bits of the memory by the debugger.

Program or data memory are selected with the P: and D: memory access qualifiers. The distinction between internal and external memory is made using the current value of the %smode processor register: Bit 0 (DDR) and 1 (DIR) disable internal data RAM respectively internal instruction RAM. Therefore memory displayed via [Data.dump](#) or [Data.List](#) will show the same memory area that the core would access in its current state.

ZSP500 may implement a so-called IBOOT (internal boot) window for supporting booting from external memory. If the IBOOT signal is enabled (ON), the core will access internal memory within the IBOOT window's address range (fixed at P:0xF800--0xFFFF), provided %smode=0 (as is the case after a reset). When the IBOOT signal is disabled (OFF), the core will access external memory in this address range (regardless of the %smode register).

The IBOOT signal is external to the core and cannot be determined by the debugger itself. Therefore to show the correct memory area (internal vs. external) in the IBOOT window, `sys.o.iboot` needs to be configured.

SYStem.CPU

Selects the processor type

Format:

SYStem.CPU <cpu>

<cpu>:

ZSP400 | LSI402ZX | LSI403LP | LSI403Z | ZSP500 | ZSP540

Selects the processor type.

| | |
|----------|---|
| Format: | SYStem.JtagClock <rate> SYStem.BdmClock <rate> (deprecated) |
| <fixed>: | 5 000 000...32 000 000 (ZSP400) 1 000 000....25 000 000 (ZSP500) |


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 commonly used frequencies. The default frequency for the fixed clock is 5 MHz.

The following commands are only used in case of multicore systems with ARM CPU. Please refer to **SYStem.JtagClock** in “**Arm Debugger**” (debugger_arm.pdf) for a detailed description

| | |
|--------------|--|
| ARTCK | Accelerated Returned TCK - clock mode. |
| CRTCK | Compensation by RTCK - clock mode. |
| CTCK | Compensation by TCK - clock mode. |
| RTCK | Returned TCK - clock mode. |

NOTE: Buffers, additional loads or high capacities on the JTAG/COP lines reduce the maximum operation frequency of the JTAG clock and should be avoided.

| | |
|---|--|
|  | The mode CRTCK can not be used if a DEBUG INTERFACE (LA-7701) or a debug cable with 14-pin flat cable (LA-7740) is used. And it is required that the target provides an RTCK signal. |
|---|--|

| | |
|---------|--|
| Format: | SYStem.MemAccess <mode> |
| <mode>: | Enable Denied StopAndGo DAP |

The option controls if system memory can be accessed while the core is executing.

| | |
|-----------------------------------|--|
| Enable CPU (deprecated) | A run-time memory access is made without CPU intervention while the program is running. This is only possible on the instruction set simulator. |
| Denied | No memory access is possible while the CPU is executing the program. |
| StopAndGo | Temporarily halts the core(s) to perform the memory access. Each stop takes some time depending on the speed of the JTAG port, the number of the assigned cores, and the operations that should be performed. |
| DAP | A run-time memory access is done via the CoreSight v2 Debug Access Port (DAP). This is only possible if a DAP is available on the chip and if the memory bus is connected to it. NOTE: The debugger accesses the memory bus and cannot see caches. |

ZSP400: The switch is fixed to the denied state because there is no way to access memory while the core is running.

ZSP500: The switch can be set as desired. If enabled, memory is accessed via the device emulation unit (DEU). The DEU is a bus master device that can access all internal and external system memories and busses independently of and without stopping the ZSP500 core. For deciding whether to access internal or external memory the debugger uses the current setting of the %smode register.

For ZSP500 this option should be changed only while the debugger is in **SYStem.Up** mode.

| | |
|---------|---|
| Format: | SYStem.Mode <mode> |
| | SYStem.Attach (alias for SYStem.Mode Attach) SYStem.Down (alias for SYStem.Mode Down) SYStem.Up (alias for SYStem.Mode Up) |
| <mode>: | Down NoDebug Go Attach Up |

Selects the target operation mode. Note that in multicore debugging the actual behavior is also influenced by the option **SYStem.CONFIG Slave**.

| | |
|----------------|---|
| Down | Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tristated. ZSP500: Keeps target in rest by holding the reset line asserted. |
| NoDebug | Disables the debugger. In this mode no debugging is possible. For further use of the debugger, system mode “Attach” must be chosen. The state of the CPU can be stopped or running. ZSP500: Disables on-chip breakpoints in the target, resumes execution and detaches from the target. |
| Go | Resets the target with debug mode enabled and prepares the CPU for debug mode entry. After this command the CPU is running and can be stopped with the break command or until any break condition occurs (breakpoint or external trigger). |
| Attach | This command does not reset the target CPU. After this command the CPU is prepared for debug mode. The state of the CPU can be stopped or running. |
| StandBy | Not available for ZSP. |
| Up | Resets the target, sets the CPU to debug mode and stops the CPU. After execution of this command the CPU is stopped and prepared for debugging. |

ZSP500: In SW debug mode the debugger sets a SW break at the reset vector to stop the core at the first instruction after the reset. For this to work correctly the following prerequisites must be met:

1. There must be RAM at the reset vector so the debugger can write the SW break.
2. A DMC (debug monitor code) must be loaded with the application. The application must be compiled such that SVT (system vector table) is loaded at the reset vector
3. **SYStem.Option.SVTADDR** must be configured to point to the reset vector which is configured for the hardware. **SYStem.Option.IBOOT** must be configured correctly and reflect the hardware's setting of the IBOOT signal (off=external ram, on=internal ram)

If any of these conditions is not met, the core will not hit the SW break after SYStem.Up. The core will therefore execute code before being stopped asynchronously by the debugger after some unavoidable delay. Due to reset delay requirements (50..1000 ms) the amount of code executed may be substantial.

In HW mode no breakpoint is set to the reset vector (on-chip breaks are cleared by the reset). SYStem.Up will therefore stop the core as soon as possible. However, this will normally takes some milliseconds so the core executes a substantial amount of code.


NOTE:

With the EB500P board, configured to boot from address 0xF800 and IBOOT=OFF (demo program with running LEDs), the core will **not** stop at the reset vector (neither in HW mode nor in SW mode). This is because there is flash memory at the reset vector and the SW break cannot be written. The core will therefore stop as soon as the debugger sends the asynchronous stop command. Due to the flash memory it is also not possible to single step through this demo program.

| | |
|-----------------------------|--|
| Format: | SYSystem.CONFIG <parameter> <number_or_address> SYSystem.MultiCore <parameter> <number_or_address> (deprecated) |
| <parameter>: (JTAG) | CORE DRPRE DRPOST IRPRE IRPOST TAPState TCKLevel TriState Slave state CJTAGFLAGS <flags> CJTAGTCA <value> DAPDRPOST <bits> DAPDRPRE <bits> DAPIRPOST <bits> DAPIRPRE <bits> |
| <parameter>: (DebugPort) | DEBUGPORTTYPE [JTAG SWD CJTAG] SWDPIDLEHIGH [ON OFF] SWDPTargetSel <value> |
| <parameter>: (BugFix) | BYPASS <pattern> FILLDRZERO |

The four parameter IRPRE, IRPOST, DRPRE, DRPOST are required to inform the debugger about the position of the TAP controller 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 example below.

TriState has to be used if more than one debugger are connected to the common JTAG port at the same time. 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, EDBG RQ must have a pull-down resistor.

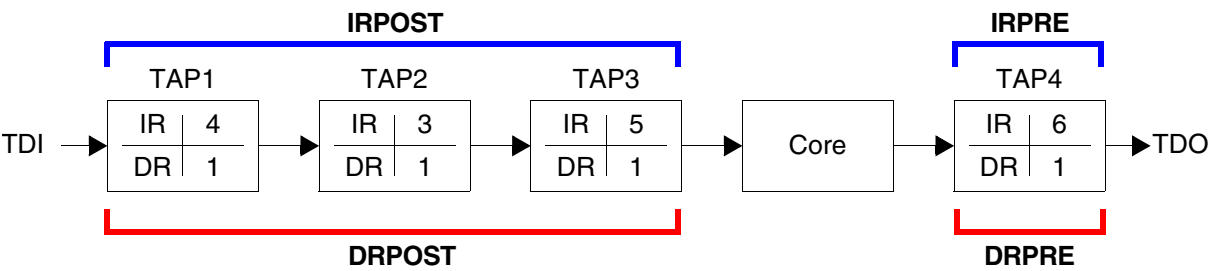


Multicore debugging is not supported for the DEBUG INTERFACE (LA-7701).

| | |
|--------------------------------|---|
| CORE | For multicore debugging one TRACE32 PowerView 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. |
| ... DRPOST <bits> | Defines the TAP position in a JTAG scan chain. Number of TAPs in the JTAG chain between the TDI signal and the TAP you are describing. In BYPASS mode, each TAP contributes one data register bit. See possible TAP types and example below. Default: 0. |
| ... DRPRE <bits> | Defines the TAP position in a JTAG scan chain. Number of TAPs in the JTAG chain between the TAP you are describing and the TDO signal. In BYPASS mode, each TAP contributes one data register bit. See possible TAP types and example below. Default: 0. |
| ... IRPOST <bits> | Defines the TAP position in a JTAG scan chain. Number of Instruction Register (IR) bits of all TAPs in the JTAG chain between TDI signal and the TAP you are describing. See possible TAP types and example below. Default: 0. |
| ... IRPRE <bits> | Defines the TAP position in a JTAG scan chain. Number of Instruction Register (IR) bits of all TAPs in the JTAG chain between the TAP you are describing and the TDO signal. See possible TAP types and example below. Default: 0. |
| TAPState | (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. |
| TCKLevel | (default: 0) Level of TCK signal when all debuggers are tristated. |
| TriState | (default: OFF) If more than one debugger share the same JTAG port, this option is required. The debugger switches to tristate mode after each JTAG access. Then other debuggers can access the port. |
| Slave | (default: OFF) If more than one debugger share the same JTAG port, all except one must have this option active. Only one debugger - the 'master' - is allowed to control the signals nTRST and nSRST (nRESET). |
| state | Show state. |

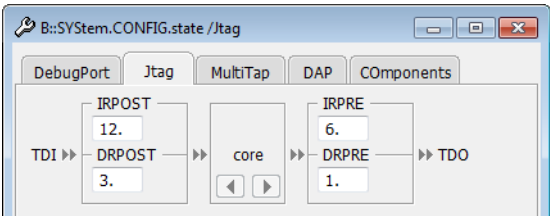
| | |
|--|---|
| BYPASS <pattern> | With this option it is possible to change the bypass instruction pattern for other cores in a multicore environment. It only works for the IRPOST pattern and is limited to 64 Bit. The specified pattern (hexadecimal) will be shifted least significant bit first. If no BYPASS option is used, the default value is '1' for all bits. This is a workaround for a certain chip problem available on ARM9 debugger only. |
| FILLDRZERO | With this option it is possible to change the bypass data pattern for other cores in a multicore environment. It changes the pattern from all '1' to all '0'. This is a workaround for a certain chip problem available on ARM9 debugger only. |
| CJTAGFLAGS <flags> | <p>Activates bug fixes for "cJTAG" implementations.</p> <p>Bit 0: Disable scanning of cJTAG ID.</p> <p>Bit 1: Target has no "keeper".</p> <p>Bit 2: Inverted meaning of SREDGE register.</p> <p>Bit 3: Old command opcodes.</p> <p>Bit 4: Unlock cJTAG via APFC register.</p> <p>Default: 0</p> |
| CJTAGTCA <value> | Selects the TCA (TAP Controller Address) to address a device in a cJTAG Star-2 configuration. The Star-2 configuration requires a unique TCA for each device on the debug port. |
| DEBUGPORTTYPE [JTAG SWD CJTAG] | <p>It specifies the used debug port type "JTAG", "SWD", "CJTAG". It assumes the selected type is supported by the target.</p> <p>Default: JTAG.</p> |
| SWDPIdleHigh [ON OFF] | <p>Keep SWDIO line high when idle. Only for Serialwire Debug mode. Usually the debugger will pull the SWDIO data line low, when no operation is in progress, so while the clock on the SWCLK line is stopped (kept low).</p> <p>You can configure the debugger to pull the SWDIO data line high, when no operation is in progress by using SYStem.CONFIG SWDPIdleHigh ON</p> <p>Default: OFF.</p> |
| SWDPTargetSel <value> | <p>Device address in case of a multidrop serial wire debug port.</p> <p>Default: none set (any address accepted).</p> |

Daisy-Chain Example



IR: Instruction register length **DR:** Data register length **Core:** The core you want to debug

Daisy chains can be configured using a PRACTICE script (*.cmm) or the **SYStem.CONFIG.state** window.



Example: This script explains how to obtain the individual IR and DR values for the above daisy chain.

```
SYStem.CONFIG.state /Jtag      ; optional: open the window

SYStem.CONFIG IRPRE    6.      ; IRPRE: There is only one TAP.
                                ; So type just the IR bits of TAP4, i.e. 6.

SYStem.CONFIG IRPOST   12.     ; IRPOST: Add up the IR bits of TAP1, TAP2
                                ; and TAP3, i.e. 4. + 3. + 5. = 12.

SYStem.CONFIG DRPRE    1.      ; DRPRE: There is only one TAP which is
                                ; in BYPASS mode.
                                ; So type just the DR of TAP4, i.e. 1.

SYStem.CONFIG DRPOST   3.      ; DRPOST: Add up one DR bit per TAP which
                                ; is in BYPASS mode, i.e. 1. + 1. + 1. = 3.
                                ; This completes the configuration.
```

NOTE: In many cases, the number of TAPs equals the number of cores. But in many other cases, additional TAPs have to be taken into account; for example, the TAP of an FPGA or the TAP for boundary scan.

| | |
|----|------------------|
| 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.Option.ADIAFTERBREAKIN [ON | OFF]

Default: OFF.

When the option is enabled, after an external breakpoint TRACE32 asserts a debug interrupt (ADI).

This option is intended for handling external breakpoints in **ZSP systems that do not support HWD mode**. This can be due to an unknown or not implemented register scan chain. External breakpoints are typically created by the cross-triggering logic of a multi-core chip.

The sequence for handling an external break is as follows:

- When attaching to the core, TRACE32 shifts (among other commands)

DEI_ENABLE_ICE (DEI := 0x70c0)
DEI_E0B_CPU (DEI := 0xba8a)
- When an external breakpoint (break signal) is asserted, the ZSP core disables the core clock
- TRACE32 detects the external breakpoint because of the stopped core clock
- TRACE32 shifts the JTAG commands sequence

DEI_RESUME (DEI := 0x2080)
DEI_ADI (DEI := 0x6080)

and thus causes the core to enter the SWD mode

Due to the inevitable delay between DEI_RESUME and DEI_ADI, the core will execute some code before entering the SWD mode. At a JTAG clock of 5 MHz this takes approximately 8 microseconds.

NOTE: The option must be set while in **SYStem.Mode.Down** (before connecting to the core via **SYStem.Up** or **SYStem.Attach**) or an error message “command locked” will be displayed.

NOTE: For systems that **do support HWD mode, the option should not be used**. For these systems TRACE32, after an external breakpoint, automatically switches to SWD mode using a method incurring a delay of only a few clock cycles.

| | |
|---------|---|
| Format: | SYStem.Option.BREAKOUTAFTERSWBREAK [ON OFF] |
|---------|---|

Default: OFF.

When this option is enabled, after hitting a SW breakpoint a break-out signal is created.

Technical background: A break-out signal is created by stopping the ZSP's clock (assuming proper configuration of the chips cross-break logic). On ZSP, a **SW breakpoint** is implemented by an ADI (assert debug interrupt) exception that branches into a debug monitor (DMC). This in itself **does not stop the core clock**. Therefore TRACE32 sets an **auxiliary onchip breakpoint** onto the ADI exception vector. This auxiliary onchip breakpoint will be hit "immediately" after raising the ADI exception and assert the break-out signal. Finally, when TRACE32 detects that the auxiliary onchip breakpoint was hit, it automatically resumes execution of the debug monitor.

| | |
|-------|---|
| NOTE: | The option must be set while in SYStem.Mode.Down (before connecting to the core via SYStem.Up or SYStem.Attach) or an error message "command locked" will be displayed. |
|-------|---|

| | |
|-------|--|
| NOTE: | In order to set the auxiliary onchip breakpoint, it is mandatory to correctly set SYStem.Option.SVTADDR ! If the address is set incorrectly, no break-out signal will be generated. |
|-------|--|

SYStem.Option.EnReset

Allow the debugger to drive nRESET (ZSP400)

| | |
|---------|----------------------------------|
| Format: | SYStem.Option.EnReset [ON OFF] |
|---------|----------------------------------|

Determines, if the debugger may assign/deassign the HRESET signal. Generally the HRESET pin of the JTAG connector resets the CPU and HRESET will be assigned during **SYStem.Up**, **SYStem.NoDebug**, **SYStem.Go** and **SYStem.Down**. If a target board doesn't reset the CPU with the HRESET signal (EB403LP of LSI !) the **SYStem.Option.EnReset** must be switched off before a **SYStem.Up** will be done. It may be necessary after switch off the EnReset that the target board must be reset again with the reset button.

Format:SYStem.Option.HardwareDebug [ON | OFF]

Default: OFF.

The option is used to switch the debugger to hardware debug mode (ON) or software debug mode (OFF). Software debug mode offers better features but requires a DMC in the target. Hardware debug mode has a number of limitations. See section *Hardware and Software Debug Modes* for details.

The option needs to be set **before** connecting the debugger to the target. Changing the option **after** attaching to the target has no effect and will not switch to the requested debug mode.

SYStem.Option.IBOOT

Configure IBOOT board signal (ZSP5xx)

Format:SYStem.Option.IBOOT [ON | OFF]

Default: OFF.

The option needs to be configured according to the actual setting of the IBOOT signal on the target board. This is required so that the debugger can display the correct memory contents in the address range of the so-called IBOOT window. The setting OFF corresponds to a LOW signal on the target board, the setting ON corresponds to a HIGH signal.

Note that also the start address of the IBOOT window needs to be configured via [SYStem.Option.SVTADDR](#).

SYStem.Option.IMASKASM

Disable interrupts while ASM single stepping

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

Mask interrupts during assembler single steps. Useful to prevent interrupt disturbance during assembler single stepping.

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

Mask interrupts during HLL single steps. Useful to prevent interrupt disturbance during HLL single stepping.

| | |
|---------|--|
| Format: | SYStem.Option.IntelSOC [ON OFF] |
|---------|--|

Default: OFF.

Informs the debugger that the core is part of an Intel® SoC. When enabled, all IR and DR pre/post settings are handled automatically, no manual configuration is necessary.

Requires that the debugger for this core is slave in a multicore setup with x86 as the master debugger and that **SYStem.Option.CLTAPOnly** is enabled in the x86 debugger.

| | |
|---------|--|
| Format: | SYStem.Option.MEMDEU [ON OFF] |
|---------|--|

Default: ON.

If set to ON the debugger accesses the memory via the on-chip DEU (Device Emulation Unit). If set to OFF, memory accesses are made through the DMC (debug monitor code). Using the DMC for memory accesses is only possible in SWD mode.

Set the option to OFF when the DEU on your target is not functional e.g. because it is not connected to the memory busses.

| | |
|---------|---|
| Format: | SYSystem.Option.RisingTDO [ON OFF] |
|---------|---|

Determines with which edge of TCK TDO will be sampled. This option is necessary for target boards with LSI402ZX, which shift TDO data out with the rising edge of TCK. Default is RISINGTDO off, i.e. TDO will be sampled at the falling edge of TCK.

SYSystem.Option.SLOWRESET

Slow reset

| | |
|---------|---|
| Format: | SYSystem.Option.SLOWRESET [ON OFF] |
|---------|---|

Default: ON.

ZSP400: If **System.Option.SLOWRESET** is on, the system up routine waits 5000 ms after releasing the Reset line and before initiating the DEI interrupt for change to debug mode. So it is guaranteed that the boot routine will not be interrupted for 5000 ms (time necessary for internal boot routine of LSI402ZX).

ZSP500: If **System.Option.SLOWRESET** is on, the debugger will wait 1000 ms after releasing the reset line (required for correct operation of the EB500P Eval Board). If the option is off, the delay is 50 ms.

SYSystem.Option.SVTADDR

Configure SVTADDR (ZSP500)

| | |
|---------|--|
| Format: | SYSystem.Option.SVTADDR <address> |
|---------|--|

Default: 0xF800

The option needs to be configured according to the actual setting of the SVTADDR signal on the target board. The debugger assumes that the IBOOT window starts at the address configured by this option. Therefore the option must be configured so that the debugger can display the correct memory contents in the address range of the IBOOT window.

When doing a system reset (**SYSystem.Up**) the debugger will set a software breakpoint at SVTADDR+0x0 which is the location of the reset vector. Therefore the option must be set correctly in order to stop the core after a reset.

Note that also the state of the IBOOT signal needs to be configured via **SYSystem.Option.IBOOT**.

For background information on SVTADDR and memory access within IBOOT window, please refer to the documentation for ZSP500 and target board.

Multicore Debugging

If your ZSP processor is the only device connected to the JTAG connector the default system settings should be kept.

If there are more CPUs connected to the same JTAG connector, the debugger has to be informed about its position within the JTAG chain. Following system settings are necessary for multicore debugging.

SYStem.LOCK

Lock debug port (ZSP400)

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

Default: OFF.

If the system is locked (ON) no access to the JTAG port will be done by the debugger. All JTAG connector signals of the debugger are tristated.

LOCK must be switched on, if several debuggers are used to debug several Cores using the same JTAG connector. By locking the debug lines for certain cores another debugger can own mastership on the JTAG interface.

It must be ensured that the state of the ZSP400 JTAG state machine remains unchanged while the system is locked.

Disassembler

The disassembler currently does not warn when it detects an invalid ldu/ldx instruction where source and target register are identical (invalid instruction):

```
ldu r13, r13, 0x1
```

Timer0, Timer1 Registers

In SW debug mode, timer registers timer0, timer1 cannot be changed by the debugger. The passed register values are ignored by the debug monitor (DMC limitation). The reason for this implementation of the DMC is that by not rewriting the timer registers their reload values remain unaffected.

After uploading registers in SWD mode, the debugger will freeze the core and therefore stop the internal timers. During the time required for register upload and download the timer registers will continue to decrement, however.

Single Stepping over RETI Fails

This is a ZSP500 limitation in SW debug mode. As entering the debug monitor (DMC) overwrites the TPC (trap pc) register, the debugger cannot determine where the RETI instruction will continue. Internally the ZSP500 records the previous value of the RPC on a stack so the program will continue correctly after leaving the DMC.

Software Breakpoints in CEXE Blocks

The ZSP500 documentation advises that SW breakpoints shall not be set within CEXE blocks. Currently the debugger does not give a warning when this condition is violated.

On-chip Breakpoints (ZSP500 Hardware Erratum)

As of the writing of this document (June 2005) there is a hardware issue with the ZSP500 core that may cause memory corruption when using on-chip breakpoints. When an on-chip breakpoint is hit the core may incorrectly execute *load* and *store* instructions causing memory corruption for addresses involved in *load* and *store* operations currently in the core pipeline. This is a ZSP500 chip issue.

It is therefore advised by LSI Logic not to use on-Chip breakpoints unless LSI Logic advises otherwise for a specific version of the ZSP500.

Software Debug Mode and Hardware Debug Mode

A number of limitations applies. For specific information refer to the relevant chapters sections.

Simulator Interface for ZSP5xx Cores

T32 supports LSI instruction accurate and cycle accurate simulators for ZSP5xx cores via the MDI interface.

For using T32 with ZSP simulators you have to install the LSI ZSP SDK which contains the core simulators. Add the path to MDI.DLL (from the LSI ZSP SDK) to your %PATH system variable (e.g. c:\LSI_Logicsdk5.1\MDI\) so T32 can load the DLL.

Configure T32 for simulator targets by adding the following line to your T32 configuration file (config.t32):

PBI=MDI

Then start T32, select a core via **SYStem.CPU** and execute the command **SYStem.Up**: T32 will connect to the simulator target. T32 chooses the actual simulator depending on the settings for

```
sys.cpu [ zsp500 | zsp540 | zsp600 ]  
sys.option.CycleAccurate [ON | off]
```

SYStem.Option.CpuAccess allows to chose the type of simulator (cycle accurate vs. instruction accurate). It can be changed only when the debugger is in DOWN state. The default settings is ON (cycle accurate).

As the ZSP5xx, ZSP6xx cores are code compatible, the same instruction accurate simulator (ZISIM) is used for all of them. For cycle accurate simulation each cores has its own simulator (ZSIM, ZSIM540, ZSIM600). ZSP4xx simulators are not supported by T32.

As the debugger can access the state of the simulator via a special software interface (MDI interface), there is no need to link a DMC with the target application. In fact, even if downloaded, the debug monitor will be ignored by the debugger. Also there is no distinction between hardware and software debug modes with simulator targets.

Limitations of the Simulators

Due to the type of simulation, the **cycle accurate** simulator has some limitations similar to hardware debug mode of a silicon target:

- It is not possible to set the PC, it seems to be ignored completely. As workaround you can set the PC:=0 by **SYStem.Down**; **SYStem.Up**.
- The shown PC is imprecise: Like in the actual hardware, in cycle accurate simulation the current PC does not necessarily tell which instruction is executed next (out-of-order, multi-scalar core). In particular the %rpc register may be set to the next return value, even before the PC encountered the respective "call" instruction. This can lead to incorrect stack backtraces.

File I/O with Simulator Targets

The ZSP simulators supports file I/O to the host file system using C standard library calls (fopen, fwrite, fprintf, ...). For supported functions see stdio.h file in your ZSP SDK installation (e.g. c:/LSI_Logicsdk5.1/zspg2/include/stdio.h).

Performance Measurements with Simulator Interface

The T32 statistical performance measurements are not currently supported for simulator targets. The measurement would be inaccurate in any case as the time required for simulation will normally not correlate to the time the target takes for executing it e.g. some Viterbi operation takes a few cycles in a DSP core while the simulator needs to emulate it costly via normal instructions.

Therefore for performance measurements with simulator targets a pseudo register "cycles" was implemented . When connected to a simulator target, there is an additional (pseudo) register "cycles" in the register window. With cycle accurate simulation (zsim500, zsim540, zsim600) this shows the number of elapsed clock cycles. With instruction accurate simulation (zsim500) it shows the number of executed instructions.

Like a normal register, the cycles register can be reset (or set to an arbitrary value). This facilitates to count the number of cycles executed within a code block.

Mechanical Description of 20-pin Debug Cable for ZSP400/ZSP500

| Signal | Pin | Pin | Signal |
|--------|-----|-----|--------|
| VTREF | 1 | 2 | N/C |
| TRST- | 3 | 4 | GND |
| TDI | 5 | 6 | GND |
| TMS | 7 | 8 | GND |
| TCK | 9 | 10 | GND |
| N/C | 11 | 12 | GND |
| TDO | 13 | 14 | GND |
| SRTS- | 15 | 16 | GND |
| N/C | 17 | 18 | GND |
| N/C | 19 | 20 | GND |

This is a standard 20-pin double row connector (pin-to-pin spacing 0.100 inch):

Mechanical Description of JTAG Connector for ZSP400 (obsolete)

| Signal | Pin | Pin | Signal |
|---------|-----|-----|--------|
| TDO | 1 | 2 | GND |
| TDI | 3 | 4 | TRST- |
| N/C | 5 | 6 | VDD |
| TCK | 7 | 8 | N/C |
| TMS | 9 | 10 | N/C |
| N/C | 11 | 12 | N/C |
| HRESET- | 13 | 14 | N/C |
| N/C | 15 | 16 | GND |

This is a standard 16 pin double row (two rows of seven pins) connector (pin-to-pin spacing: 0.100 in.)

NOTE:

This pinout was used with older ZSP400 boards and was superseded by the 20-pin ARM-compatible pinout (see above). In case you need a connector with the old pinout, please contact LAUTERBACH support.

Operation Voltage

This list contains information on probes available for other voltage ranges. Probes not noted here supply an operation voltage range of 4.5 ... 5.5 V.

| Adapter | OrderNo | Voltage Range |
|---|---------|---------------|
| JTAG Debugger for ZSP500 DSP Core (ICD) | LA-3712 | 1.8 .. 3.6 V |
| JTAG Debugger for ZSP400 DSP Core | LA-7832 | 1.8 .. 3.6 V |