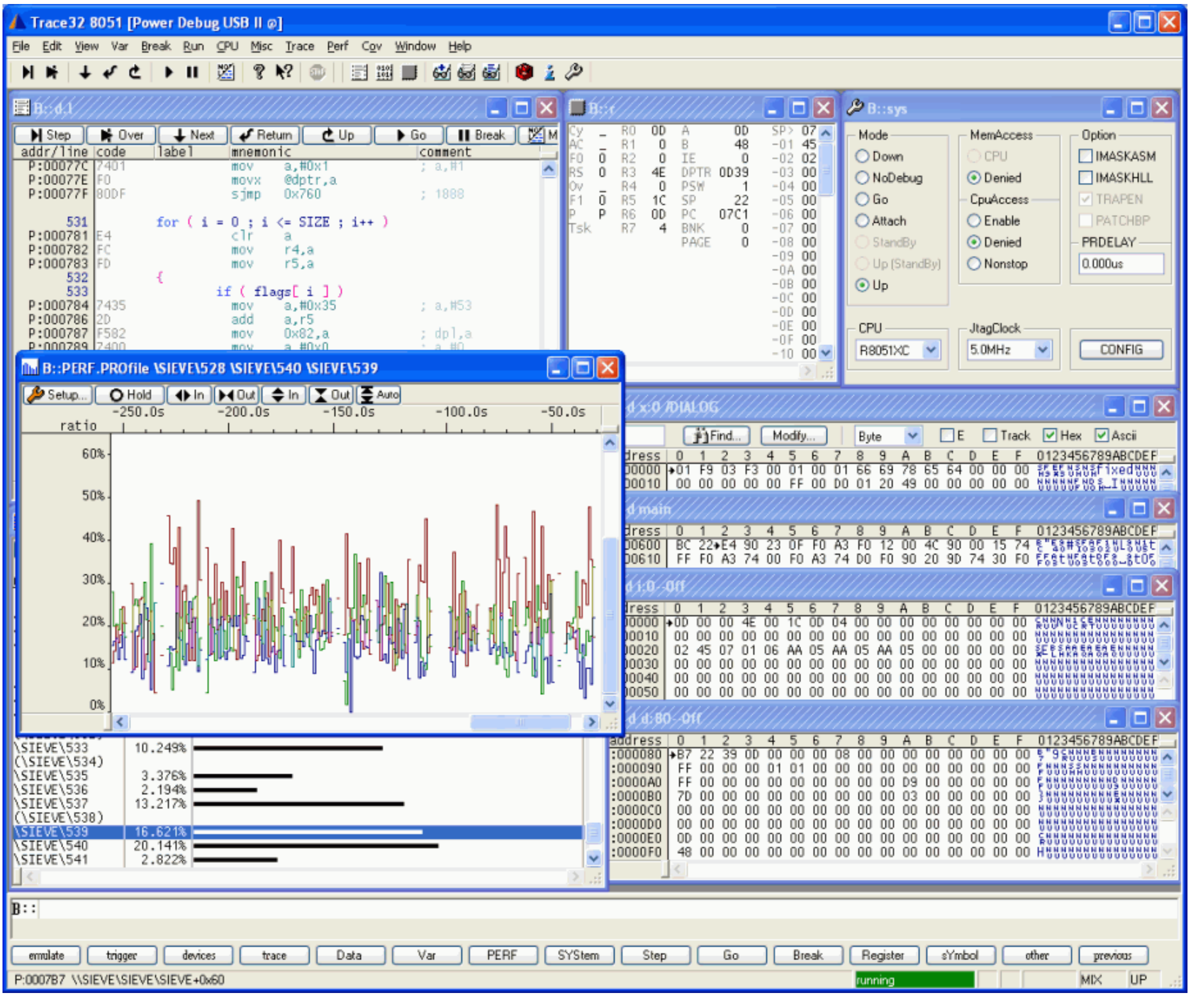
LAUTERBACH
DEVELOPMENT TOOLS

R8051XC Debugger

MANUAL

R8051XC Debugger

Version 06-Jun-2024



Introduction

This document describes the processor specific settings and features of the TRACE32 debugger for the CAST, Inc. / Evatronix SA “R8051XC” IP core CPU family.

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.

Warning

WARNING:	To prevent debugger and target from damage it is recommended to connect or disconnect the Debug Cable only while the target power is OFF. Recommendation for the software start: 1. Disconnect the Debug Cable from the target while the target power is off. 2. Connect the host system, the TRACE32 hardware and the Debug Cable. 3. Power ON the TRACE32 hardware. 4. Start the TRACE32 software to load the debugger firmware. 5. Connect the Debug Cable to the target. 6. Switch the target power ON. 7. Configure your debugger e.g. via a start-up script. Power down: 1. Switch off the target power. 2. Disconnect the Debug Cable from the target. 3. Close the TRACE32 software. 4. Power OFF the TRACE32 hardware.

Quick Start

Starting up the debugger is done as follows:

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

```
b::
```

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

2. Specify the CPU specific settings.

```
SYStem.CPU <cpu_type>
```

The default values of all other options are set to values that should allow to start work without modification. Please consider that these values are possibly not the best configuration for your target.

3. Set up the JTAG electrical interface clock speed.

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

The default frequency is 10 MHz, but please note that the actually usable frequency depends on your chip/FPGA design. If your JTAG connection does not support the RESET signal, please press your target board reset button before the next command to ensure a HARD RESET.

4. Enter debug mode.

```
SYStem.Up
```

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

5. Load your application program.

```
Data.LOAD.OMF2 myprogram /Verify    ; OMF2 specifies the format,
                                    ; myprogram is the file name
```

The format of the **Data.LOAD** command depends on the file format generated by the compiler. It is recommended to use the option /Verify that verifies all written data. This test spots any problems with the electrical connection, wrong chip configurations or linker command file settings.

A detailed description of the **Data.LOAD** command and all available options is given in the **"General Commands Reference"**.

The start-up can be automated using the programming language PRACTICE. A typical start sequence for R8051XC-based CPUs 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

SYStem.CPU R8051XC                 ; Select CPU

SYStem.UP                          ; Reset the target and enter debug mode

Data.LOAD.OMF2 APP.OM2 /VERIFY     ; Load the application, verify the
                                   ; process

Go main                            ; Run and break at main()

Data.List                          ; Open source window

Register.view /SpotLight           ; Open register window

Var.Local                          ; Open window with local variables
```

SYStem.Up Errors

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

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

  The debugger controls the processor reset and use the RESET line to reset the CPU on every SYStem.Up. Additionally it executes an R8051XC soft reset. If you have no RESET line connected, please make sure you manually hard-reset the target board before continuing.
- There is logic added to the JTAG state machine:

  The debugger is configured at start-up to expect only one R8051XC core in the JTAG chain. Please use the **SYStem.CONFIG** command (“CONFIG” button) to configure the JTAG chain position of the core in a multi-core configuration.
- There are additional loads or capacities on the JTAG lines
- The core you want to debug has to be started first by another core, or target board has additional RESET delay logic. Please use **SYStem.Option.PRDELAY**.
- You have additional logic on your board that requires special handling of JTAG lines during or at the end of system RESET. Please make sure the JTAG port is enabled correctly.

KEIL OMF-51 and OMF2

- For R8051XC debugging, the KEIL compiler currently supports only the “Intel MCS-51 Object Module Format” (OMF-51/OMF-251). KEIL extended this format to store some additional information within the OMF file, e.g. to support banking.
- The KEIL linkers can generate OMF (OMF-51) and OMF2 (OMF-251) format, depending on your project settings. Please select the appropriate TRACE32 command for loading OMF or OMF2.

  Load your OMF-51 application program with:

```
Data.LOAD.OMF myprogram /verify      ; OMF specifies the format,
                                     ; myprogram is the file name
```

  Load your OMF2 application program with:

```
Data.LOAD.OMF2 myprogram /verify     ; OMF2 specifies the format,
                                     ; myprogram is the file name
```

  A detailed description of the **Data.LOAD** command and all available options is given in the **“General Commands Reference”**.

- OMF-51 specifies source files by name only, and does not include directories.

  If your project is split into several subdirectories, and your HLL source code is not found, please either provide a list of source directories using the **Data.LOAD** /PATH option, or by using the **sYmbol.SourcePATH.SetRecurseDir** command.

Debugging with Low Target Frequencies

When designing and testing your new chip design with the R8051XC IP core, your design (or an ASIC emulator) might support only a fraction of normal JTAG and processor frequencies. In this case:

- You can reduce the update rate of the TRACE32-PowerView GUI with

  `SETUP.URATE <rate per second | time>`

- You can cache program and data areas with

  `MAP.UpdateOnce P:0--0FFFFF`

  Please remember to access your data with the correct memory type specifiers (D:, I:, P:, X:), do not use C:. The cache is invalidated with each STEP or GO command.

- Please restrict data windows to the minimum required address ranges.
  E.g. instead of “d d:0” and “d i:0”, use “d d:80--0FF” and “d i:00--0FF”.
- Minimizing windows you don’t currently need also reduces the amount of data that has to be transferred between host and target.

Mapping Memory

- Processor designs with Harvard architecture, such as the R8051XC, have separate program and data memory buses.
- For various purposes it may be useful or necessary to map data space to program space and vice versa. Sometimes during development a read-writable data memory area is mirrored into a read-only program memory area, or e.g. program flash is mapped to a read-only data area.
- An “unlimited” number of software breakpoints can only be set within read-writable memory. For read-only memory only a very limited number of hardware on-chip breakpoints can be used.
- If you have a read-writable data area that is mapped into read-only program space, you can redirect the debugger breakpoint setting from program memory to data memory with the **TRANSlation** command.

  TRANSlation.Create *<logical_range>* [*<physical_range>*] [**/***<option>*]

  Example:

```
TRANSlation.Create P:0100--0FFFF X:4100

TRANSlation.ON
```

  MAP.BOnchip *<addressrange>*

  Example:

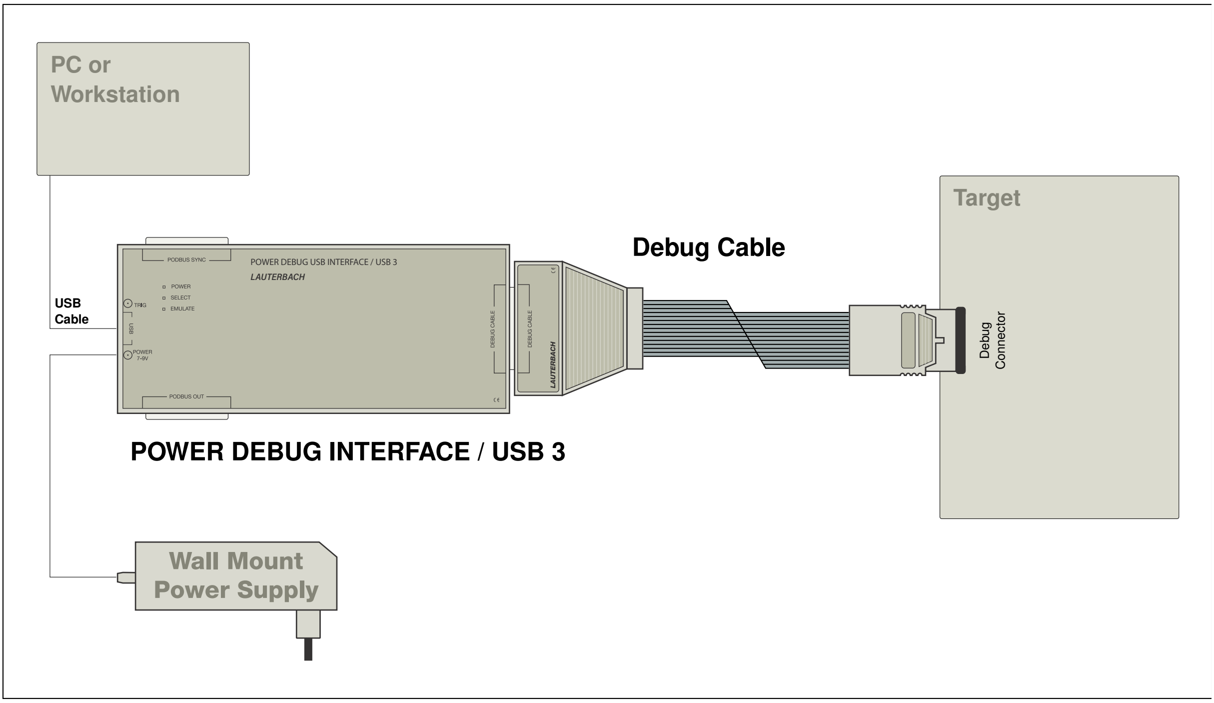
```
MAP.BOnchip P:0--0FF
```

FAQ

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

Configuration

Example configuration for an R8051XC debugger.



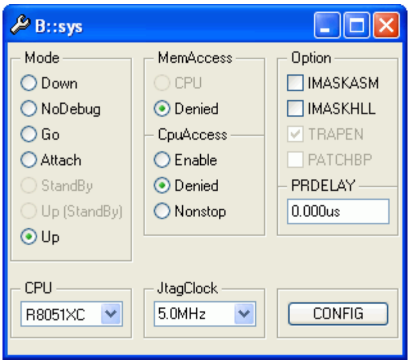
The processor type must be selected by the **SYStem.CPU** command before issuing any other target related commands.

CPU specific SYStem Settings and Restrictions

SYStem.state Open system window

Format:	**SYStem.state**

Opens a window with settings of CPU specific system commands. Settings can also be changed here.



SYStem.CONFIG.state Display target configuration

Format:	**SYStem.CONFIG.state** [**/**<*tab*>]
<*tab*>:	**DebugPort** | **Jtag**

Opens the **SYStem.CONFIG.state** window, where you can view and modify most of the target configuration settings. The configuration settings tell the debugger how to communicate with the chip on the target board and how to access the on-chip debug and trace facilities in order to accomplish the debugger's operations.

Alternatively, you can modify the target configuration settings via the TRACE32 command line with the **SYStem.CONFIG** commands. Note that the command line provides *additional* **SYStem.CONFIG** commands for settings that are *not* included in the **SYStem.CONFIG.state** window.

<*tab*>	Opens the **SYStem.CONFIG.state** window on the specified tab. For tab descriptions, see below.

DebugPort	Informs the debugger about the debug connector type and the communication protocol it shall use.
Jtag	Informs the debugger about the position of the Test Access Ports (TAP) in the JTAG chain which the debugger needs to talk to in order to access the debug and trace facilities on the chip.

SYStem.CONFIG — Configure debugger according to target topology

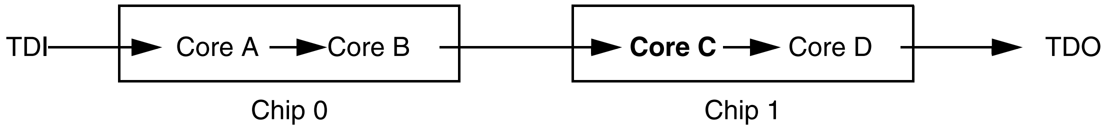
Format:	**SYStem.CONFIG** *<parameter> <number_or_address>* **SYStem.MultiCore** *<parameter> <number_or_address>* (deprecated)
<parameter>:	**CORE** *<core>*
<parameter>: (JTAG):	**DRPRE** *<bits>* **DRPOST** *<bits>* **IRPRE** *<bits>* **IRPOST** *<bits>* **TAPState** *<state>* **TCKLevel** *<level>*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

The four parameters IRPRE, IRPOST, DRPRE, DRPOST are required to inform the debugger about the TAP controller position in the JTAG chain, if there is more than one core in the JTAG chain (e.g. Arm + DSP). The information is required before the debugger can be activated e.g. by a **SYStem.Up**. See **Daisy-chain Example**.
For some CPU selections (**SYStem.CPU**) the above setting might be automatically included, since the required system configuration of these CPUs is known.

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

!	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. Further information can be found in **SYStem.CONFIG.CORE**.
DRPRE	(default: 0) *<number>* of TAPs in the JTAG chain between the core of interest and the TDO signal of the debugger. If each core in the system contributes only one TAP to the JTAG chain, DRPRE is the number of cores between the core of interest and the TDO signal of the debugger.
DRPOST	(default: 0) *<number>* of TAPs in the JTAG chain between the TDI signal of the debugger and the core of interest. If each core in the system contributes only one TAP to the JTAG chain, DRPOST is the number of cores between the TDI signal of the debugger and the core of interest.
IRPRE	(default: 0) *<number>* of instruction register bits in the JTAG chain between the core of interest and the TDO signal of the debugger. This is the sum of the instruction register length of all TAPs between the core of interest and the TDO signal of the debugger.
IRPOST	(default: 0) *<number>* of instruction register bits in the JTAG chain between the TDI signal and the core of interest. This is the sum of the instruction register lengths of all TAPs between the TDI signal of the debugger and the core of interest.
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 several debuggers share the same debug port, this option is required. The debugger switches to tristate mode after each debug port access. Then other debuggers can access the port. JTAG: This option must be used, if the JTAG line of multiple debug boxes are connected by a JTAG joiner adapter to access a single JTAG chain.
Slave	(default: OFF) If more than one debugger share the same debug port, all except one must have this option active. JTAG: Only one debugger - the “master” - is allowed to control the signals nTRST and nSRST (nRESET).



Below, configuration for core C.

Instruction register length of

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

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

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

Format:	**SYStem.CONFIG.CORE** *<core_index> <chip_index>* **SYStem.MultiCore.CORE** *<core_index> <chip_index>* (deprecated)
<chip_index>:	**1 … i**
<core_index>:	**1 … k**

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

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

To provide proper interaction between different parts of the debugger, the systems topology must be mapped to the debugger's topology model. The debugger model abstracts chips and sub cores of these chips. Every GUI must be connect to one unused core entry in the debugger topology model. Once the **SYStem.CPU** is selected, a generic chip or non-generic chip is created at the default *chip_index.*

Non-generic Chips

Non-generic chips have a fixed number of sub cores, each with a fixed CPU type.

Initially, all GUIs are configured with different *chip_index* values. Therefore, you have to assign the *core_index* and the *chip_index* for every core. Usually, the debugger does not need further information to access cores in non-generic chips, once the setup is correct.

Generic Chips

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

Start-up Process

The debug system must not have an invalid state where a GUI is connected to a wrong core type of a non-generic chip, two GUIs are connected to the same coordinate or a GUI is not connected to a core. The initial state of the system is valid since every new GUI uses a new *chip_index* according to its CORE= parameter of the configuration file (config.t32). If the system contains fewer chips than initially assumed, the chips must be merged by calling **SYStem.CONFIG.CORE**.

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**R8051XC**
<cpu>:	**S8051XC3**

Selects the processor type. The available types depend on your adapter type and license.

SYStem.JtagClock

Define JTAG clock

Format:	**SYStem.JtagClock** [*<frequency>*] **SYStem.BdmClock** [*<frequency>*] (deprecated)
<frequency>:	**1.0MHz** | **5.0MHz** | **10.0MHz** | *<other>*

Selects the JTAG port frequency (TCK) used by the debugger to communicate with the processor. The frequency affects e.g. the download speed. It may be required to reduce the JTAG frequency if there are buffers, additional loads or high capacities on the JTAG lines or if VTREF is very low. A very high frequency will not work on all systems and will result in an erroneous data transfer.

<frequency>	• Default is 10MHz • *<other>* is 6kHz … 80MHz The debugger cannot select all frequencies accurately. It chooses the next possible frequency and displays the real value in the **SYStem.state** window. Instead of decimal numbers like “100000.”, short forms like “10kHz” or “15MHz” may be used. The short forms imply a decimal value, although no “.” is used.

When the debugger is not working correctly (e.g. memory display flickers), decrease the JtagClock.

SYStem.LOCK

Lock and tristate the debug port

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

Default: OFF.

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

SYStem.MemAccess Select run-time memory access method

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

Enable **CPU** (deprecated)	The mode "CPU" cannot be selected, because there is no way to do runtime access to the memory while the R8051XC core is running.
Denied	The mode "Enable" cannot be selected, because there is no way to do runtime access to the memory while the R8051XC core is running.
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.

SYStem.Mode Establish communication with the target

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**

Down	The CPU is held in reset (if the RESET signal is attached), debug mode is not active. Default state and state after fatal errors.
NoDebug	Disables the debugger. The state of the CPU remains unchanged. The JTAG port is tri-stated.
Go	Resets the target and enables the debugger and start the program execution. Program execution can be stopped by the break command or if any break condition occurs.

Attach	User program remains running (no reset) and the debug mode is activated. After this command the user program can be stopped with the break command or if any break condition occurs.
Up	Resets the target, sets the CPU to debug mode and stops the CPU. After the execution of this command the CPU is stopped and all registers are set to the default level.
StandBy	Not supported.

SYStem.Option.IMASKASM Disable interrupts while single stepping

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

Default: OFF.

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

SYStem.Option.IMASKHLL Disable interrupts while HLL single stepping

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

Default: OFF.

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

SYStem.Option.LittleEndian Treat memory as little endian

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

Default: OFF.

Treats memory as little endian.

SYStem.Option.DPTREXT Selects the address of DPS register

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

Default: OFF.

Must be enabled if DPTR extension is implemented. Then the DPS register is located at address 0x86 and otherwise at 0x92. Only S8051XC3

Format:	**SYStem.Option.PRDELAY** [<*time*>]
<*time*>:	**0 … 60000ms**

Set a wait time after releasing the RESET signal before JTAG communication with the target is continued. Useful for target boards with an on-board reset delay unit, or if another core has to enable the target core before JTAG communication is possible.

<*time*>	Default is 0us

Instead of decimal numbers like “1000.”, abbreviated forms like “1s” or “500ms” may be used. This command always implies a decimal value, although no “.” is used. Fractional values can be entered (e.g. “1000.250”) but the fractional part is ignored.

Memory Classes

The following memory classes are available:

Memory Class	Description
P	Program
X	External data (XRAM)
I	Internal RAM (Indirect Address)
D	Special Function Registers + Internal RAM (Direct Address)

The low 128 bytes of the internal data memory are mirrored in the memory classes I and D.
The upper 128 bytes in the memory class D represent the Special Function Registers SFR.

If the peripheral configuration of your chip supports SFR banking, then the banked SFR contents are visible in the address range beyond 0x80--0xFF.
E.g. the SFR Bank 5 would be visible in the upper 128 bytes of D:0500--05FF.

Special Function Register (SFR) symbols

Special Function Registers (SFRs) for all 8051 derivatives are located within the memory range `D:80--FF` and accessed via MOV 'direct' memory opcodes.

All SFRs with an address where bits [2..0] are not set (e.g. `D:80`, `D:88`, `D:90`, `D:98`, etc.) are bit-adressable like the memory in the range `D:20--2F`.

One problem for disassembly is to distinguish "normal" addresses and constants in the range 0x80..0xFF from SFR and SFR bit definitions. Some registers (A, B, PSW) are available on all 8051 derivatives. For these, default names and addresses (that can be overwritten by an external definition) are hard-coded into the disassembler. But the majority of platforms will have different peripherals located on different addresses.

PUBSFR section in KEIL OMF-251

KEILs OMF-251 (OMF2) format contains a special PUBSFR section for SFR and SBIT definitions.

Here is an example for a KEIL definition for the PSW and its bit flags:

```
sfr  PSW            = 0xD0; // Program Status Word
sbit P              = 0xD0; // Parity Flag
sbit F1             = 0xD1; // General Purpose Flag 1
sbit OV             = 0xD2; // Overflow Flag
sbit RS0            = 0xD3; // Register Bank Select 0
sbit RS1            = 0xD4; // Register Bank Select 1
sbit F0             = 0xD5; // General Purpose Flag 0
sbit AC             = 0xD6; // Auxiliary Carry Flag
sbit CY             = 0xD7; // Carry Flag
```

When such a definition is included in a C or ASM source file and the output format is set to OMF2, the compiler/linker emits this definition in the ABS file.

After symbol load the special function register is available in the dis/assembler.

Pure symbol definitions (and no code) can be loaded from an OMF-251 file with:

```
DATA.LOAD.OMF2 my_symbols.om2 /NoCODE
```

For R8051XC cores, SFR symbols can be created in PRACTICE with the D: and B: addressing modes.

D:00xx addresses (xx=0x80--0xFF) are SFR byte definitions, B:0yyy bit addresses are computed by multiplying the SFR base address with 8 and then adding the bit offset.

Example: For the PSW at address 0xD0, the PSW_3 bit address (RS0) is (0xD0 * 8) + 3 = 0x683.

This is the PRACTICE definition for the R8051XC PSW:

```
sYmbol.CREATE.RESet       ; erase all user-defined symbols
sYmbol.CREATE             ; start symbol creation
sYmbol.NEW PSW   D:00D0   ; Program Status Word
sYmbol.NEW P     B:0680   ; Parity Flag                (0xD0 * 8 + 0)
sYmbol.NEW F1    B:0681   ; General Purpose Flag 1     (0xD0 * 8 + 1)
sYmbol.NEW OV    B:0682   ; Overflow Flag              (0xD0 * 8 + 2)
sYmbol.NEW RS0   B:0683   ; Register Bank Select 0     (0xD0 * 8 + 3)
sYmbol.NEW RS1   B:0684   ; Register Bank Select 1     (0xD0 * 8 + 4)
sYmbol.NEW F0    B:0685   ; General Purpose Flag 0     (0xD0 * 8 + 5)
sYmbol.NEW AC    B:0686   ; Auxiliary Carry Flag       (0xD0 * 8 + 6)
sYmbol.NEW CY    B:0687   ; Carry Flag                 (0xD0 * 8 + 7)
sYmbol.CREATE.Done        ; finish symbol creation
```

NOTE:	If the **SYStem.CPU** selection is not set to an R8051XC or M8051EW derivative, all D:xxxx definitions will be mapped to I:xxxx definitions. These do not represent SFR addresses.

TrOnchip.state — Display on-chip trigger window

Format: **TrOnchip.state**

Opens the **TrOnchip.state** window.

TrOnchip.CONVert — Adjust range breakpoint in on-chip resource

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

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

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

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

TrOnchip.RESet — Set on-chip trigger to default state

Format: **TrOnchip.RESet**

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

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

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

JTAG Connectors

These are designed by the users of the R8051XC core, i.e. the SoC and PCB designers. If publishable R8051XC JTAG connector pinouts become available, they will be listed here.

For ideas how to layout your own JTAG connector(s), please refer to the next section.

LAUTERBACH Adapters

These are the pin assignments of the LAUTERBACH ARM and R8051XC debug cables:

ARM 20-pin Adapter

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	VSUPPLY (not used)
TRST-	3	4	GND
TDI	5	6	GND
TMS|TMSC|SWDIO	7	8	GND
TCK|TCKC|SWCLK	9	10	GND
RTCK	11	12	GND
TDO|-|SWO	13	14	GND
RESET-	15	16	GND
DBGRQ	17	18	GND
DBGACK	19	20	GND