LAUTERBACH
DEVELOPMENT TOOLS

dsPIC33 Debugger

MANUAL

dsPIC33 Debugger

dsPIC33 Debugger

Version 06-Jun-2024

History

20-Nov-19 Initial version.

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.

Introduction

This manual serves as a guideline for debugging dsPIC33C/E cores and describes all processor-specific TRACE32 settings and features.

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.

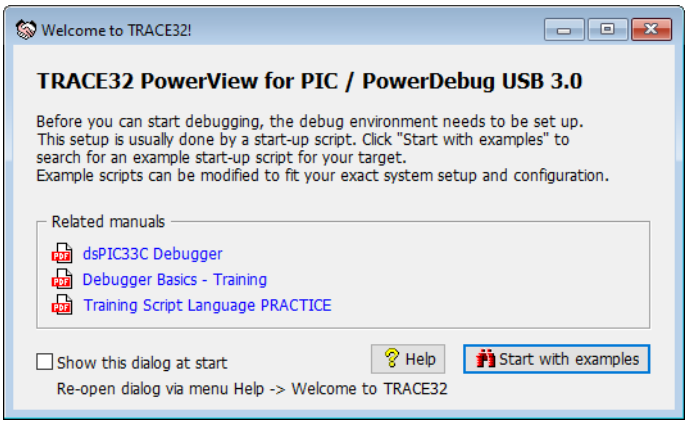
Brief Overview of Documents for New Users

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:

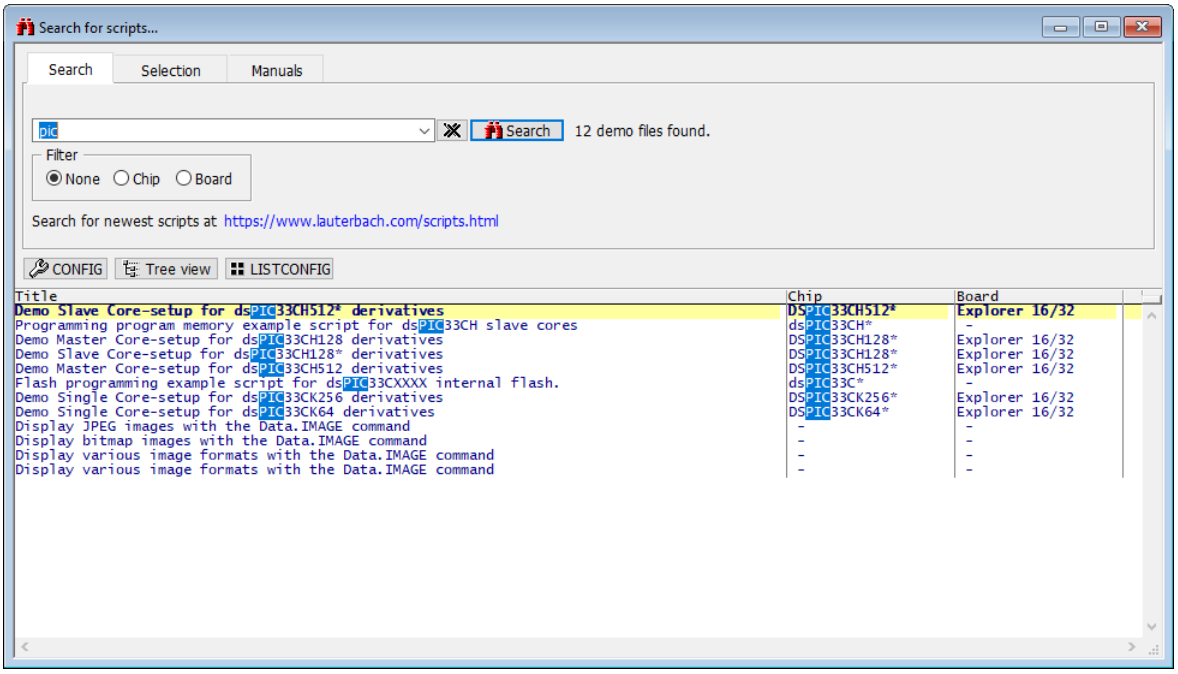
- **"Processor Architecture Manuals"**: These manuals describe commands that are specific for the processor architecture supported by your debug cable. To access the manual for your processor architecture, proceed as follows:
  - Choose **Help** menu > **Processor Architecture Manual**.
- To get started with the most important manuals, use the **Welcome to TRACE32!** dialog (**WELCOME.view**):



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**.

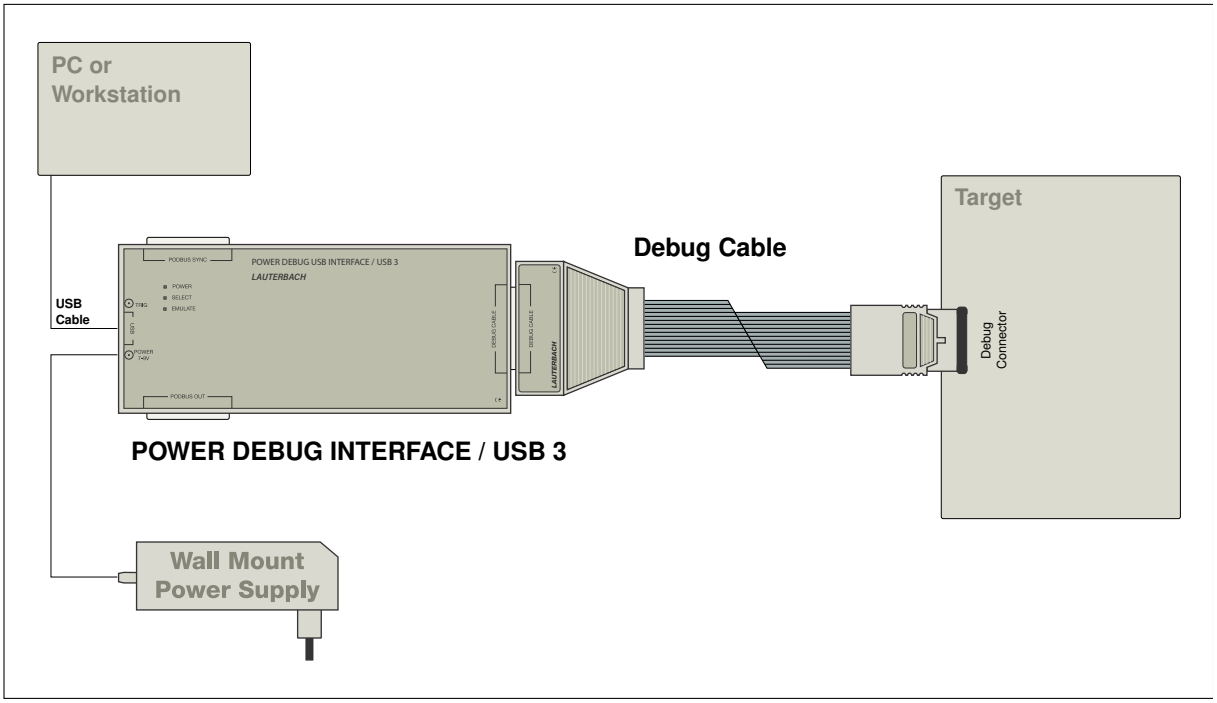
  You can now search the demo folder and its subdirectories for PRACTICE start-up scripts (*.cmm) and other demo software.



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

System Overview

Example configuration for a single core debugger.



POWER DEBUG INTERFACE / USB 3

Please consider the tips given in the chapter **"Connector Type and Pinout"**, page 26.

Start a New Debug Session

Starting up the debugger is done as follows:

1. Select the device prompt B (BDM debugger) and reset TRACE32.

```
B::

RESet
```

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 DSPIC33CH128MP508
```

This command selects the CPU type. In case the exact type of CPU is not known, the command **SYStem.DETECT.CPU** can be used to detect the connected target.

3. Reset the target and enter debug mode.

```
SYStem.Mode Up
```

This command resets the CPU on the target, enables On-Chip-Debug Mode and halts at the first instruction.The CPU stops executing any instruction, and the user is able to download and test the code. After this command is executed, it is possible to access memory and registers.

If this command results in an error, the target might not be prepared for debugging with TRACE32. Either it is configured in productive mode where the program in the flash is executed right after the reset, or an incompatible debug monitor is programmed. In these cases use following command to prepare the target before entering the debug mode.

```
FLASH.UNSECUREerase
```

If the CPU **DSPIC33XXXX** is selected, TRACE32 tries to detect the CPU type of the connected target before the system is brought up.

4. Load the program into the program memory.

```
DO ~~/demo/pic/flash/dspic33cxxxx.cmm
```

A typical start sequence of the PIC 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

RESet                               ; Reset the TRACE32 software

WinCLEAR                            ; Clear all windows

SYStem.Up                           ; Reset the target and enter debug mode

DO                                  ; Load the target application with the
~~/demo/pic/flash/<script>.cm       ; family specific script
m


                                    ; Set the stack pointer to address 8000

PER.view                            ; Show clearly arranged peripherals
                                    ; in window                        *)

List.Mix                            ; Open source code window          *)

Register.view /SpotLight            ; Open register window             *)

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

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

Break.Set 0x101000 /Program         ; Set on-chip breakpoint
/Onchip                             ; to address 101000
```

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

NOTE:	Due to the architecture of the dsPIC33 microcontroller, the on-chip breakpoints halt the target two instructions after the program counter (PC) reached the address of an on-chip breakpoint. This is called skid.

To write an application where the debug access is disabled to the program memory, a different approach must be used. In this case, the target is configured to be in productive mode so that the loaded program is executed right after a reset.

1. Select the device prompt B (BDM debugger) and reset TRACE32.

```
B::

RESet
```

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 by selecting the appropriate script and prepare the program memory

```
DO ~~/demo/pic/flash/dspic33cxxxx.cmm PREPAREONLY
```

3. Reset the target and enter prepare mode.

```
SYStem.Mode Prepare
```

This command resets the CPU on the target, and enables program memory access.

4. Write the configuration information first.

```
FLASH.ReProgram 2.

Data.LOAD.auto * P:0x2B800++0x7FF

FLASH.ReProgram off
```

The address range given in the Data.LOAD.auto command must be modified to point to the last flash page of the target which includes the configuration memory space.

5. Write the remaining program memory.

```
FLASH.ReProgram 1.

Data.LOAD.auto *

FLASH.ReProgram off
```

The program binary selected for the Data.LOAD.auto command should be the same as in the previous step.

6. Reset target

```
SYStem.RESet
```

Error Message	Event	Reason
Target power fail	**SYStem.Mode.Up**	See below.
No clock signal detected.	**SYStem.Mode.Up**	See below.
Target processor in reset	**SYStem.Down**	See below.
The number of *<number>* accessed bytes in memory is not a multiple of the access size *<size>* bytes.	No special event	Internal error, please consult your Lauterbach representative.
Memory address *<address>* is not aligned to access size *<size>*.	No special event	Internal error, please consult your Lauterbach representative.
Invalid memory access size: *<size>* bytes (@ address *<address>*)	No special event	Internal error, please consult your Lauterbach representative.
Memory access timeout: Reading from address *<address>*	No special event	Corrupted debug connection. Check debug hardware and settings.

Typically the **SYStem.Up** command is the first command of a debug session where communication with target is required. If you receive error messages like “debug port fail” or “debug port time out” while executing this command, this may have the reasons below. “target processor in reset” is just a follow-up error message.

- Open the **AREA.view** window to display all error messages.
- If the target has no power or the debug cable is not connected to the target, this results in the error message “target power fail”.
- Did you select the correct core type with **SYStem.CPU** *<cpu>*?
- There is an issue with the debug interface. Maybe there is the need to set jumpers on the target to connect the correct signals to the debug connector. The debugger will not work, for example, if PGEC signal is directly connected to ground on target side.
- The target is in an unrecoverable state. Re-power your target and try again.
- The default debug clock prescaler is too low. In this case try **SYStem.CLockPrescaler 0xA0** and optimize the speed when you got it working.
- The target was not prepared for debugging with TRACE32. In this case try **FLASH.UNSECUREerase**.
- There are no pull-down resistors connected to the communication lines. For further information see the chapter **“Connector Type and Pinout”**, page 26.
- The core has no clock.
- The core is kept in reset.
- There is a watchdog which needs to be deactivated.

FAQ

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

dsPIC33 Debug Monitor

In order to debug a dsPIC33C/E target, a debug monitor is required. The debug monitor is a software program which executes on the target whenever the target receives a halt request, e.g. by a breakpoint or a user initiated break. The debug monitor then communicates with the debugger, which allows access to the target system. Therefore, the debug monitor capabilities have a direct influence on the debugger capabilities.

Lauterbach provides debug monitors which are **not** compatible with the debug tools of other manufacturers. The debug monitor is designed to support all basic and advanced debug features offered by a certain dsPIC33 family.

The Lauterbach debug monitors require up to 2.908 Bytes of memory and must be loaded to the address P:0x800000. This is a separate area in the flash memory and does not affect the space available for user programs. In general, the debug monitor code must be present in the target memory before the debugger can be used. To load the suitable Lauterbach debug monitor into the target's flash memory, the command **FLASH.UNSECUREerase** should be used. This command erases the user code memory and configures the currently used debug port, too.

NOTE:	The application loaded for debugging **must reserve 80 bytes** of data memory at the address **D:0x1000**, which must not be modified by the program. This area is used by the debug monitor to save register data, etc. Modifying the data in this area might cause the debugger to crash. Please check if your tools automatically reserve this area while linking the program binary.

Breakpoints

Software Breakpoints

The Microchip dsPIC33 architecture does support unlimited software breakpoints. But their usage is not recommended as setting them will partially rewrite the flash memory and therefore reduces the number of flash erase cycles. The default breakpoints are On-chip breakpoints.

On-chip Breakpoints for Instructions

Most Microchip dsPIC33 MCUs support a total of up to eight on-chip breakpoint registers which can be used as program breakpoints to stop and debug the program which executes always in the Flash. When debugging the slave core of a dsPIC33CH derivative only three breakpoints are available. Please consider the skid of two assembler instructions when using on-chip breakpoints. That means, the core usually halts two instructions after the on-chip breakpoint.

NOTE:	One of the on-chip breakpoints of a dsPIC33CH slave core can also be used as data breakpoint. When debugging other core types, even five data breakpoints are possible. If data breakpoints are used, the total number of program breakpoints is reduced.

On-chip Breakpoints for Data

Data breakpoints are used to analyze the read and write accesses to global variables. The data breakpoints can be triggered with respect to the data address or access type, i.e. read, write or both, or the data value. Up to five on-chip breakpoints of dsPIC33 MCUs can be used as data breakpoints. On the slave core of a dsPIC33CH derivative one data breakpoint is available.

In case of an on-chip data breakpoint, every load and store instruction is checked with respect to the breakpoint address, access type and the value. The data breakpoints are especially useful to find out when a global variable is written with a certain value. It is not possible to implement a similar breakpoint in software without affecting the real-time behavior of the system. Since the load and store instructions work on RAM, data breakpoints always point to addresses on RAM.

Memory Classes

The dsPIC33 architecture is a Harvard-type processor architecture. Therefore, following different memory access classes are available:

Access Class	Description
D	Data
P	Program

To access a memory class, write the class in front of the address. For example, use D to access the data memory:

```
Data.dump D:0x00
```

The following examples return different results, since the dsPIC architecture uses the Harvard Architecture.

```
Data.dump D:0x100
```

```
Data.dump P:0x100
```

Programming the On-chip FLASH of the dsPIC33

The PRACTICE script for programming of the on-chip FLASH of a dsPIC33 can be found in the TRACE32 demo folder ~~/demo/pic/flash/.

For programming the program memory of a dsPIC33E core, the script dspic33epxxx.cmm should be used. For programming the program memory of a dsPIC33C master core with a single partition, the script dspic33cxxxx.cmm should be used. For dual partition configurations of a dsPIC33C core, the script dspic33cxxxx_dual.cmm is suitable. The dspic33chxxxslave.cmm is intended for flashing a dsPIC33CH slave core.

Please be aware that these are just example scripts. They might need some adaption to fit your MCU.

To debug only the slave core of a dsPIC33CH target, the FLASH of the master core must be programmed at least with a stub function including the hardware configuration words for the master and slave core. Afterwards the slave core can be programmed. For further details see the scripts mentioned above.

Additionally, an application can be flashed to the chip's program memory where the debug ports of the target are disabled. To do so, the target must be brought to Prepare mode before the binary is written to flash memory. In this case, the scripts mentioned above will fail.

Special Hints, Restrictions, and Known Problems

Special Hints

- Due to the architecture of the dsPIC33 microcontrollers, the target will always halt two assembler instructions after an on-chip breakpoint's address. This can lead to imprecisions when doing HLL steps.

Restrictions

- The use of SW breakpoints is discouraged as setting them leads to faster reduction of the target's number of flash erase cycles.
- **Go.Return** will stop the target right after the current function is left. Because of the on-chip breakpoint implementation, the debugger can not stop the target at the function epilog.

Known Problems

- Stack frames not correctly shown when entering library functions.

NOTE:	All problems will be fixed in one of the next SW versions without notice!

SYStem.CLockPrescaler — Select the prescaler for the debug clock

Format: **SYStem.CLockPrescaler** *<value>*

Default: 0x03.

Selects the prescaler for the clock used by the debug port. For a satisfying performance of the debug communication, this value should only be set to a higher value if the debug communication fails.

SYStem.CONFIG.state — Display target confguration

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

<tab>: **DebugPort**

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 (default) — The **DebugPort** tab informs the debugger about the debug connector type and the communication protocol it shall use.

For descriptions of the commands on the **DebugPort** tab, see **DebugPort**.

Format:	**SYStem.CONFIG** *<parameter>*
<parameter>: **(DebugPort)**	**DEBUGPORT [DebugCable0 | DebugCableA | DebugCableB]** **DEBUGPORTTYPE [SPI]** **Slave [ON|OFF]** **TriState [ON|OFF]**

The **SYStem.CONFIG** commands inform the debugger about the available on-chip debug and trace components and how to access them.

The **SYStem.CONFIG** command information shall be provided after the **SYStem.CPU** command, which might be a precondition to enter certain **SYStem.CONFIG** commands, and before you start up the debug session e.g. by **SYStem.Up**.

<parameters> describing the “DebugPort”

DEBUGPORT [DebugCable0 | DebugCableA | DebugCableB]

It specifies which probe cable shall be used e.g. “DebugCableA” or “DebugCableB”. At the moment only the CombiProbe allows to connect more than one probe cable.

Default: depends on detection

DEBUGPORTTYPE [SPI]

It specifies the used debug port type “SPI”. At the moment only “SPI” is selectable.

Default: SPI.

Slave [ON | OFF]

If several debuggers share the same debug port, all except one must have this option active.

Default: OFF.
Default: ON if `CORE=... >1` in the configuration file (e.g. config.t32).

TriState [ON | OFF]

TriState has to be used if several debug cables are connected to a common debug port.

Default: OFF.

Format:	**SYStem.CPU** *<cpu>*
<cpu>:	**DSPIC33CH512MP508** | **DSPIC33CK32MP102** | …

Default: DSPIC33XXX.

Selects the processor type. Most of the current Microchip dsPIC33C and dsPIC33E MCU cores are supported.

SYStem.LOCK 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 connector of the debugger is tristated. The intention of the SYStem.LOCK command is, for example, to give debug access to another tool. The process can also be automated, see **SYStem.CONFIG TriState**

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

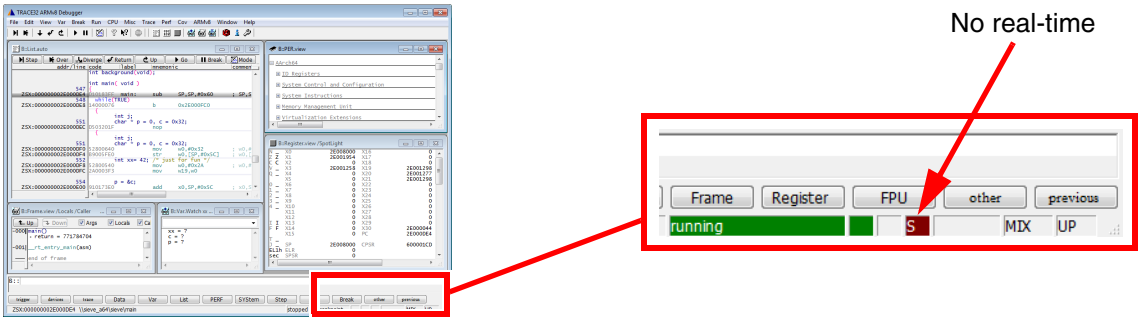
<mode>: **Denied**
StopAndGo

Default: Denied.

Denied No memory access is possible while the CPU is executing the program.

StopAndGo Temporarily halts the core to perform the memory access. Each stop takes some time depending on the speed of the debug port and the operations that should be performed.
For more information, see below.

If **SYStem.MemAccess StopAndGo** is set, it is possible to read from memory, to write to memory and to set software breakpoints while the CPU is executing the program. To make this possible, the program execution is shortly stopped by the debugger. Each stop takes some time depending on the currently active debug port clock prescaler and the operations that should be performed. A white S against a red background in the TRACE32 state line warns you that the program is no longer running in real-time:



To update specific windows that display memory or variables while the program is running, select the memory class **E:** or the format option **%E**.

```
Data.dump E:0x100

Var.View %E first
```

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
Prepare
Go
Attach
Up

Default: Down.

Down Disables the debugger. The CPU is reseted.

NoDebug Disables the debugger. The state of the CPU remains unchanged. The debug port is tristated.

Prepare Resets the target. The debugger initializes the debug port, but does not connect to the CPU. This debug mode is used if the CPU shall not be debugged but programmed with an application binary intended for productive use.

Go Resets the target, initializes the debug port, and starts program execution.

Attach Initializes the debug interface and connects to core while program remains running.

After this command the user program can be stopped with the break command or by any other break condition (e.g a breakpoints).

Up Resets the target and stops the CPU at the reset vector.

StandBy Not available for this architecture.

The **SYStem.Option** commands are used to control special features of the debugger or to configure the target. It is recommended to execute the **SYStem.Option** commands **before** the emulation is activated by a **SYStem.Up** or **SYStem.Mode** command.

SYStem.Option.BReakonWDT Enable break on watchdog time-out

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

Default: OFF.

If enabled, the program execution halts on a Watchdog time-out. If the CPU is in running mode and this option is disabled, a Watchdog time-out resets the CPU.

NOTE: If the program execution is already halted due to a breakpoint or break command, a Watchdog time-out is ignored. See: **SYStem.Option.EnableWDT**

SYStem.Option.CLockSWitch Enable clock group switch

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

Default: ON.

If enabled, a loaded program can switch the clock group used by the CPU. Otherwise a break occurs.

SYStem.Option.ENableWDT Enable watchdog timer

Format: **System.Option.ENableWDT** [**ON** | **OFF**]

Default: ON.

This option enables a global Watchdog timer. The system's reaction to a Watchdog time-out can be configured by using **SYStem.Option.BReakonWDT**.

NOTE:	If the program execution is already halted due to a breakpoint or break command, a Watchdog time-out is ignored.

SYStem.Option.FastRC Use FRC as debug port clock

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

Default: ON.

If enabled, the debug port runs on the Fast RC Oscillator instead of the system clock.

SYStem.Option.FreezePer Freeze peripherals on break or breakpoint

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

Default: OFF.

The on-chip peripherals of a dsPIC33 chip have can be configured to freeze when the program execution is interrupted. Several of those peripherals have no separate FREEZE bit in the configuration registers. All the peripherals lacking such a FREEZE are globally controlled by this configuration bit.

If enabled, the peripherals freeze when the program execution is interrupted. If disabled, the peripherals run normally when a breakpoint or break command occurs.

SYStem.Option.IMASKASM Disable interrupts while single stepping

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

Default: ON.

If enabled, the interrupt enable flag of the EFLAGS register will be cleared during assembler single-step operations. After the single step, the interrupt enable flag is restored to the value it had before the step. It is turned on to make sure that no interrupt routine is serviced between **Break** and **Go** states.

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

Default: ON.

If enabled, the interrupt enable flag of the EFLAGS register will be cleared during HLL single-step operations. After the single step, the interrupt enable flag is restored to the value it had before the step.

SYStem.Option.PARTitionconfig Configure the Flash partitions

Format: **SYStem.Option.PARTitionconfig** [**SinglePARTition** | **DUALpartition** | **PROTectedDualpart** | **PRIVilegedDualpart**]

Default: SinglePARTition.

SinglePARTition The Flash memory will be used as one partition.

DUALpartition The Flash memory will be split in two partitions. Both partitions have the same size but will be loaded with different program code according to the binary file.

PROTectedDualpart The Flash memory will be split in one protected and one normal partition. Similar to the previous configuration but partition 1 will be permanently erase/write-protected. Partition 2 can still be altered.

PRIVilegedDualpart The Flash memory will be split in two partitions. The Boot Segment limitation has special protection to prevent changes. This option is not supported by all dsPIC33 MCUs.

Several dsPIC33 MCUs support an on-chip Flash memory which can be split into two partitions. The active partition begins at address 0x000000 and in case of a dual partition configuration, the inactive partition begins at address 0x400000. Depending on the used target, the size of the partitions can vary. For further details please refer to the target chip's data sheet.

SYStem.Option.PoWeRSaVe Enable PWRSAV instruction

ON

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

Default: ON.

If enabled, the PWRSAV instruction will cause the chip to enter Idle or Sleep mode. Otherwise the program execution will be interrupted.

SYStem.state — Display SYStem.state window

Format: **SYStem.state**

Displays the **SYStem.state** window for system settings that configure debugger and target behavior.

CPU specific TrOnchip Commands

The **TrOnchip** command group is not available for the dsPIC33 debugger.

Probe Cables

For debugging a dsPIC33 single or master core, the following kinds of probe cables can be used to connect the debugger to the target:

- AUTO26 Debug Cable V2/V3
- AUTO26 Whisker for CombiProbe

The debug logic of the dsPIC33CH family only allows to debug a single core at a time. Debugging the second core requires either a second Debug Cable with a second tool set or a CombiProbe with two Whiskers.

Connector Type and Pinout

Debug Cable

	Microchip ICSP	JTAG (future use only)
Pin	**Signal**	**Signal**
1	RESET-	N/C
2	VTREF	VREF-DEBUG
3	GND	GND
4	PGD	TDO
5	PGC	TCK
6	N/C	RESET-
7	N/C	TDI
8	N/C	TMS

A standard 1 x 8 pin header (pin-to-pin spacing: 0.1 inch = 2.54 mm) is required on the target.

- Do not connect the N/C pins. Even if pins 7 and 8 are present on the provided adapter (LA-2773).
- If there are dsPIC33 derivatives which can be debugged via JTAG, the future use pinout will be used.
- VTREF is the processor power supply voltage. It is used to detect if target power is on and it is used to supply the output buffers of the debugger. That means the output voltage of the debugger signals (PGD, PGC) depends directly on this signal.
- If not already present on the target board, consider the use of a pull-up resistor on pin 1 and pull-down resistors on pin 4 and 5, with approximately 5.1 kOhms each. The most stable debug connection is achieved when the pull-down resistors are as close as possible to the board's pin header and connected to pin 3 and pin 4, and to pin 3 and pin 5, respectively.