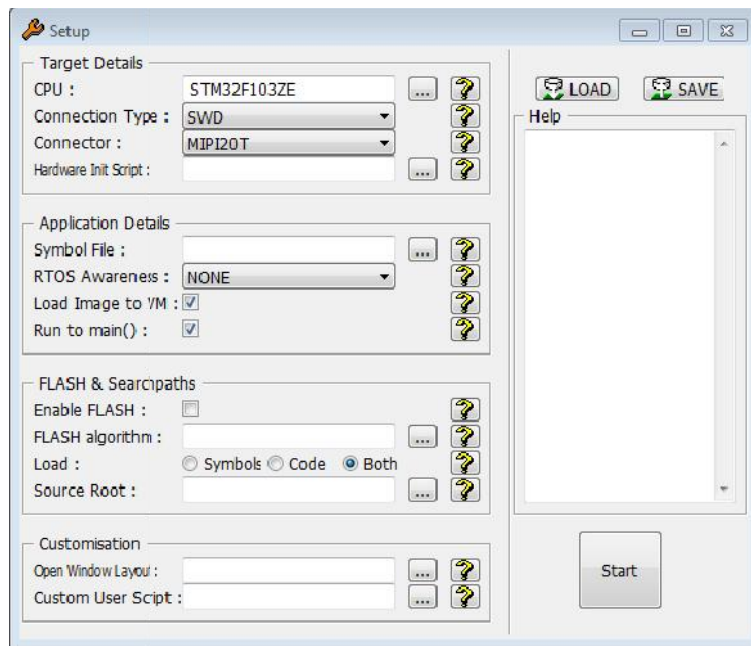


# Using the startup script with $\mu$ Trace



## About this document

This document describes the functionality and provides usage information for the **startup.cmm** script. This script is designed to simplify the setup of a new target for use with the Lauterbach  $\mu$ Trace system.

## Pre-Requisites

The Lauterbach TRACE32 software for  $\mu$ Trace has been installed. It is assumed that this has been installed to the default location of **C:\T32\_uTrace**. It will be referred to as \$T32SYS in the rest of this document. This document assumes no prior knowledge of the Lauterbach TRACE32 software or the  $\mu$ Trace hardware.

## Setup Procedures

The **startup.cmm** script is usually supplied as part of a board support package and is unzipped into the \$T32SYS directory. You may wish to take a backup of this directory beforehand.

Connect the CombiProbe header to Socket A on the  $\mu$ Trace.

Connect the CombiProbe header (using the appropriate adapter) to the debug header on the target board. More information for specific targets will be provided in the startup guide included with the BSP.

It is recommended to power on the  $\mu$ Trace unit first and then power on the target board.

Start the TRACE32 software and you should see something like Figure 1. A link will have been created in the start menu during the program's installation. You may have a shortcut created elsewhere for convenience.

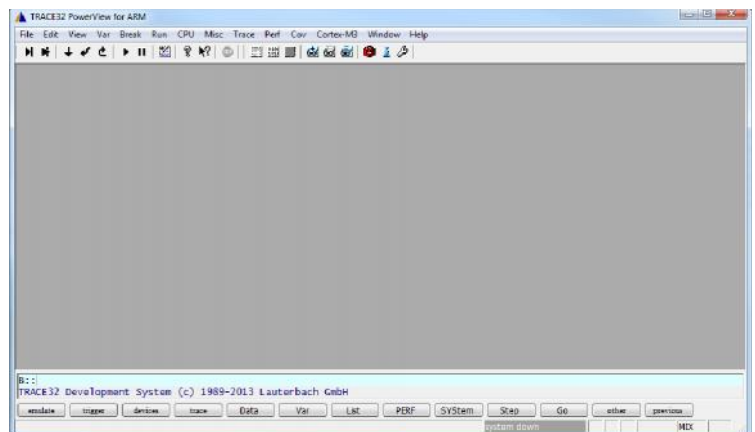


Figure 1: TRACE32

From the **File** menu, select **Run Batchfile...**, browse to \$T32SYS and select **startup.cmm**. You should see a window which looks like Figure 2.

TRACE32 can be configured to call this script each time you start it up. To do this, edit the **t32.cmm** file in the \$T32SYS directory. At the bottom of the file, just before the line that reads "ENDDO" add a line which reads

```
do startup.cmm
```

The startup.cmm script can be called with an optional argument which represents a default configuration to load. For example:

```
do startup.cmm C:\T32_uTrace\my_LPC1768.t32ini
```

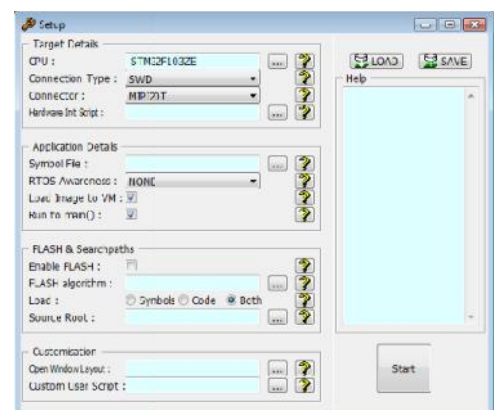


Figure 2: startup.cmm

## Basic Configuration

If you are using the script to load a default configuration for a board support package that you have installed then follow the instructions provided in the guide for your target. In summary, it will be clicking the Load button and browsing for a setup configuration and then clicking the big Start button.

This next section will provide more information on using the interface to create or modify your own configuration file(s).

If at any time you need a reminder or some more information on the options and what they do, click on the “?” button alongside and a help message will be displayed. An example of the CPU help can be seen in Figure 3. All of the options have some help text associated with them.

The TRACE32 software has an on-line help feature. Selecting a window or partially entering a command and then pressing F1 will cause the Adobe Acrobat Reader (if installed) to load the correct help file and display the appropriate page.

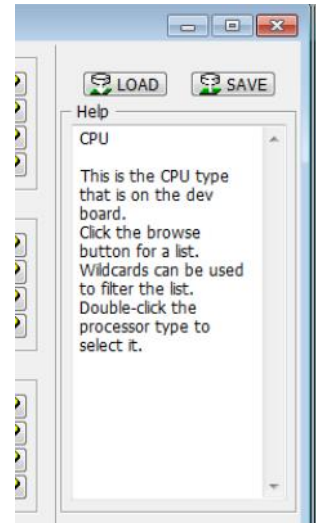


Figure 3: Help display

## Target Details

Figure 4 shows the options available for the initial hardware setup.

The first step is to enter the CPU type that is on the target board. Click the “...” button a browse window will open that allows you to select from a list of supported processors. The list has a text field at the top where you can enter a wildcard pattern and all matches will be displayed. Double-click the correct entry to close the pop-up and fill in the field on the main window with the selected value. An example can be seen in Figure 5.

Pay attention to the letters and numbers that sometimes appear after the main processor name. These can be important for selecting which package you have and will ensure that the correct hardware settings are used by the debugger. This could be things like FLASH and RAM sizes and base addresses or number of breakpoints, etc.

Next, select the connection type to the target board. The options are:

- Serial Wire Debug (SWD)
- JTAG
- cJTAG

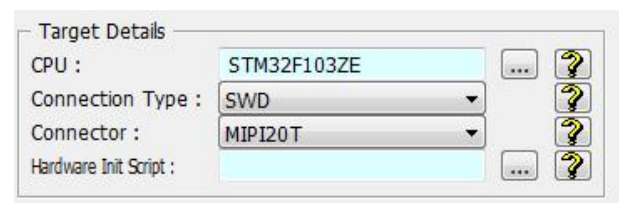


Figure 4: Target Details

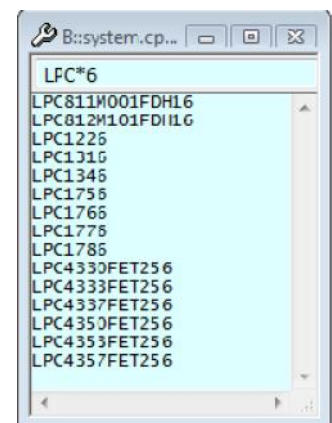


Figure 5: CPU Selection

You may need to refer to the chipset manual and/or board schematics to determine which you have. Many Cortex-M devices start in JTAG mode and then allow the debugger to switch to SWD. If you are using one of the 14 or 20 pin headers pitched at 0.1 inches then it is most likely that you have JTAG. If you are using one of the 10, 20 or 34 pin headers pitched at 0.05 inches then you are probably using SWD.

If you are using one of the MIPI (small) connectors you will need to tell the debugger which one you are using. This is only relevant if your connection type is SWD.

The final option in this section is to provide the name of a hardware initialisation script that will be called after the debugger has established the connection to the core but before it does anything else. An example script can be seen in Figure 6.

This script is designed to initialise any on-chip peripherals or to setup any chip selects or memory interfaces beyond what the basic JTAG/SWD reset would achieve. Here you may disable a watchdog or configure it to halt when the core is in debug mode. The example shown here is for an LPC4357 device which has two cores: a Cortex-M4 and a Cortex-M0. In this case we are only interested in the main core so the script shuts the other one down. It then detects if a  $\mu$ Trace is being used and configures the pin multiplexing options so that trace data is available.

You can create your own script (or ask your local Lauterbach representative if he has one already ☺). For more help with this you will need to refer to your chipset documentation. The following Lauterbach help files will also be useful:

**Training\_practice.pdf**  
**Practice\_ref.pdf**  
**Practice\_user.pdf**

These can be found in \$T32SYS\pdf.

```
;; Clear any pending Cortex-M0 interrupts
data.set ASD:0x40043130 %long 0x00

;; Disable any new Cortex-M0 interrupts
data.set ASD:0xE000E180 %long 0x00000002

IF UTRACE( )
(
    print "uTrace configuring TRACE pins"
    ; Enable trace pins
    ; Enable pin function on PF_[8..4]
    Data.Set SD:0x40086790 %Long 0x00000022
    Data.Set SD:0x40086794 %Long 0x00000023
    Data.Set SD:0x40086798 %Long 0x00000023
    Data.Set SD:0x4008679C %Long 0x00000023
    Data.Set SD:0x400867A0 %Long 0x00000023

    ; Select external trace
    ETM.ON
    ITM.ON

    ETM.PortMode Continuous
    ETM.PortSize 4.

    Trace.Method CAnalyzer
    Trace.Off
)
ELSE
(
    ; Enable ETB SRAM at 0x2000C000
    Data.Set SD:0x40043128 %Long 0x00000000

    ; Select ETB trace
    ETM.ON

    Trace.Method Onchip
    Trace.OFF
)
enddo
```

Figure 6: Hardware Initialisation Script

## Application Details

Figure 7 shows the options for the application that is to be debugged on the target.

The Symbol File is the executable that has been produced by the compiler or IDE that you are using. This should have been compiled with full debug info and minimal optimisation to ensure a good debug experience. Clicking the "..." button will open a file browse dialog to allow users to select the correct file.

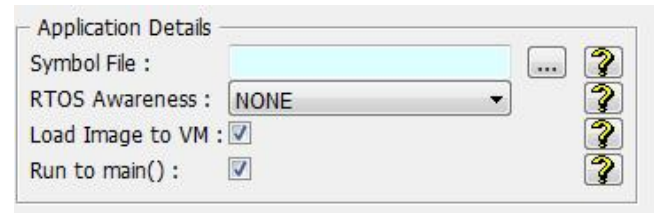


Figure 7: Application Details

From the RTOS Awareness dropdown users can choose which RTOS (if any) they have in their target application. This will cause the debugger to load the correct Kernel Awareness plug-in as part of the initialisation. For some choices additional information will need to be provided after the "Start" button has been clicked. This is normally a confirmation of which version of an RTOS is being used as sometimes the plug-ins are version specific. More information on the awareness for your RTOS can be found in the `$T32SYS\PDF` directory. Look for the file that matches the pattern `rtos_<your_rtos>.pdf`, for example `rtos_freertos.pdf`.

Ticking the "Load Image to VM" box will cause TRACE32 to load a copy of the application code into virtual target memory. This is memory on the host PC that TRACE32 uses to mimic target memory. For some trace analysis features, the performance can be improved by using this feature as the tool doesn't have to continually refer to target memory via a slow JTAG connection. Don't select this option if you have self modifying code or code that relocates at runtime.

Most debug format files (ELF, AXF, OUT, etc.) include information about the entry point to the code. If this is present the debugger will place the Program Counter at this location after loading the file to target. Often, users will not need to debug the low level board bring up code so ticking the "Run to main()" box allows the debugger to skip this and will run the target until the `main()` function is reached.

## FLASH Programming

Figure 8 shows the options for programming FLASH or dealing with code already resident in FLASH on the target.

Tick the "Enable FLASH" box to enable FLASH programming and then click the "..." button to select a FLASH programming algorithm. A file

browser will be opened showing a list of all supported FLASH algorithms. Select the appropriate one for your device. If your device or family is not listed please contact your local Lauterbach representative.

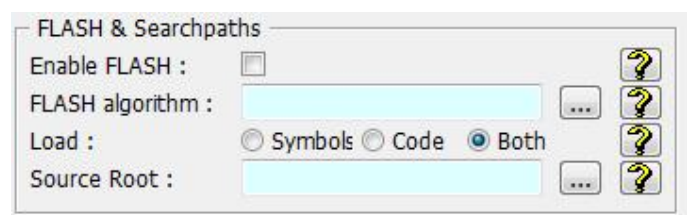


Figure 8: FLASH Details

If the application is already resident in FLASH then the debugger will only need to load the symbols, so select "Symbols".

If the code to be loaded is a binary and does not contain any debug info, select "Code".

If the application is to be loaded to FLASH and then debugged, select "Both".



Source root only needs to be filled in if the target application was built on a different machine from the one doing the debugging. Most debug format files include path information for the source files that were used during the build. TRACE32 uses this information for displaying source code. If the source files are not in the location specified in the debug file a new path needs to be used. This is entered here. Most of the time this will not be needed and can be left blank.

## Customisation

Figure 9 shows the final customisation options.

A Window layout file can be created by opening all of the required windows in TRACE32 and then select **Store Windows to...** from the **Window** menu. Many different workspace layouts can be created and this option allows you to select a default one to start.

Finally, an option is provided to call a custom user script after everything else has been setup. An example is shown in Figure 10. This adds some extra manual configuration for the FreeRTOS Kernel Awareness plug-in and then creates a new button on the toolbar to run specific set of commands to collect ITM trace data on task switches, run for 6 seconds and then show the analysis windows. It also defines a custom bitmap for the new button.

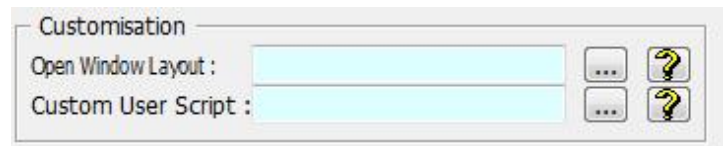


Figure 9: Customisation Details

```
task.config freertos 0. 100.*4
task.stack.pattern 0xa5

menu.rp
(
  add
  toolbar
  (
    toolitem "Task Aware Tracing Demo"
    (
      break.set task.config(magic) /write /TRACEDATA
      ITM.DataTrace CorrelatedData
      trace.autoinit ON
      go
      wait 6.s
      break
      break.delete task.config(magic)
      WinPOS 0.14286 0.33333 149. 8. 25. 3. W006
      WinTABS 9. 52.
      Trace.STATistic.TASK

      WinPOS 0.14286 14.833 149. 8. 25. 2. W007
      Trace.CHART.TASK
    )
  )
  [
    BBBBBBBBBBBBBR
    BWWWWWWWWWWWB
    BWWWWWWWWWWWB
    BXXWWXWWXWWXB
    BWWWWWWWWWWWB
    BWWWWWWXXWWWB
    BWWXXXWWXWWB
    BWWWWWWWWWWWB
    BBBBBBBBBBBBBB

    XXX XXX XXX X X
    X X X X XX
    X XXX XXX XX
    X X X X X X
    X X X XXX X X
  ]
)
)
enddo
```

Figure 10: User Script

**And Finally...**

Once the information for your target has been entered it can be saved to a .t32ini file. These can be re-loaded or passed as an argument to the startup.cmm script when it is called. This has been discussed earlier in this document. These features can be accessed by using the "LOAD" and "SAVE" buttons.

Once everything is complete, click the big "Start" button to run the configuration.