

Lauterbach TRACE32 Emulator

1 Abstract

This document describes the usage of the Lauterbach TRACE32 debugger/emulator as target system. Special settings are required in case of targets that need to be flashed.

In case of problems using the target hardware connected to TRACE32, it is recommended to use the TRACE32 simulator to understand the basic interaction between TRACE32 and TESSY (simulators for all supported target platforms are available on the Lauterbach home page <http://www.lauterbach.com>).

Table of Contents

1	Abstract	1
2	Lauterbach TRACE32	2
2.1	Linker/Locator Settings.....	2
2.2	TRACE32 Startup.....	2
2.3	Optional Module Attributes	3
2.4	Flashing the Target Binary	3
2.5	Special Flash Procedures	4
2.6	Parallel Execution.....	5
2.6.1	Using t32start.exe	5
2.6.2	Using the simulator being started for each test object.....	6
2.6.3	Using Pre and Post Execution Command for parallel execution	7
3	Troubleshooting.....	8
3.1	Init/Start/Stop Script Problems	8
3.2	Sync State Polling Delay	8
3.3	Simulator Fails Using VLE.....	8

2 Lauterbach TRACE32

The integration of the Lauterbach TRACE32 emulator uses the API provided by TRACE32 (RCI Remote Control Interface). The RCI interface needs to be enabled to connect to TRACE32. Please ensure the following settings in the **config.t32** file (Normally, they are added automatically during the installation of TRACE32):

```
RCL=NETASSIST  
PACKLEN=1024
```

TESSY will connect to an already running TRACE32 instance on the same computer. If the connection fails, the above settings are missing or commented out within the configuration file of the TRACE32 instance that was actually started (e.g. a config file for TRACE32 may also be specified via command line).

TESSY uses **PRACTICE** commands (the scripting language of TRACE32) for all communication issues between TESSY and TRACE32. You may execute certain CMM scripts via module attribute.

2.1 Linker/Locator Settings

The TESSY generated test binary may either be executed within RAM memory (if enough memory space is available) or from within FLASH memory. Both configurations need to be defined within the linker/locator settings of the module. Refer to the TESSY application notes of the respective compiler for details.

- If the application shall be run within RAM memory, there are no further TRACE32 specific settings required.
- In case of test execution from within FLASH memory, the necessary configuration of the TRACE32 scripts (as described below) depend on the flash procedure required for the specific target. See below for the two configuration possibilities.

2.2 TRACE32 Startup

The TRACE32 needs to be set up for the microcontroller to be used. You may use a CMM file, that you are already using for your normal development and debugging process.

Please note: Start TRACE32 before running the tests within TESSY. TESSY will then connect to the running instance of TRACE32.

Make sure to disable all watchdog or similar functionalities if you are using your own hardware or special evaluation boards.

It is also important to check the memory mapping that you have defined for the compiler (within the module properties). This mapping has to fit to the settings defined in the CMM startup file of TRACE32.

2.3 Optional Module Attributes

During test startup and execution within TRACE32, you may execute several optional scripts which have to be specified within the module attributes. Also setting some options concerning the communication to TRACE32 is possible. The following table shows the possible attributes and their meaning.

Module attribute	Description
Init Script	Script file to be executed after connection to TRACE32
Preload Wait Time	Time (in seconds) to wait after execution of the Init Script , before loading the binary.
Load Auto	If the Boolean attribute is set to true TRACE32 automatically determines the type of the binary program to be loaded.
Load Options	PRACTICE command options to be passed to the data.load command issued by TESSY to load the binary file (e.g. "/NOCODE").
Postload Script	Script file to be executed after loading the binary
Postload Wait Time	Time (in seconds) to wait after execution of the Postload Script , before starting the binary.
Start Script	Script file to be executed after reaching the main() function
Stop Script	Script file to be executed before disconnecting from TRACE32
Wait Timeout	The time to wait for a reaction of TRACE32 (in seconds), the default value is 30 seconds. If the TRACE32 does not respond within this time, a timeout error will occur.

You may also change some default settings:

Module attributes	Description
Debug Host	Host where TRACE32 is running (default "localhost"). Remote testing or debugging is currently not supported by TESSY.
Debug Port	Port on which TRACE32 waits for connections (default 20000)

2.4 Flashing the Target Binary

Use the **Postload Script** together with the **Init Script** for flash programming on target configurations that require flashing the binary before executing the test.

InitScript

```
flash.erase all
wait 1.s
flash.erase off
flash.program all
```

PostloadScript

```
flash.program off
```

In this configuration, the target binary will be loaded by TESSY **after** executing the InitScript and **before** executing the PostloadScript. If your target requires more than the above outlined PRACTICE commands to do the flashing, please refer to the next section for a more flexible loading procedure.

2.5 Special Flash Procedures

If your target hardware requires a special procedure (e.g. written in PRACTICE script code) to flash the target memory, you may do the complete programming and loading within the **Init Script**. The following steps are necessary for this kind of flash loading:

- Add all flash programming commands to the InitScript
- Specify the module attribute **Load Options** with the value **"/NOCODE"**. This causes TRACE32 to load only the symbol information, since the code is already flashed after execution of the InitScript.
- Do **not** specify a PostloadScript

Within the StartScript, the PRACTICE variable **&TESSY_BINARY_FILE** holds the name of the generated test binary (for the currently selected test object), that needs to be loaded into TRACE32. This variable will be added dynamically by TESSY before executing the script.

The example below contains the standard flash loading procedure using variable **&TESSY_BINARY_FILE** within the **Init Script**:

```
; This is a TESSY generated init script for Trace32
; You may edit this script but BE CAREFUL WITH ANY CHANGES!
;
; The following variables will be added automatically by TESSY before
; execution of this script:
;
; &TESSY_BINARY_FILE          The name of the generated binary file
; &TESSY_TESTOBJECT_NAME     The name of the current test object

b.delete

sys.up

flash.erase all
wait 1.s
flash.erase off
flash.program all

d.load.elf &TESSY_BINARY_FILE

flash.program off
```

Important Notice: When flashing the binary within the **Init Script**, you **must not** specify the **Postload Script** to finish the flash programming algorithm!

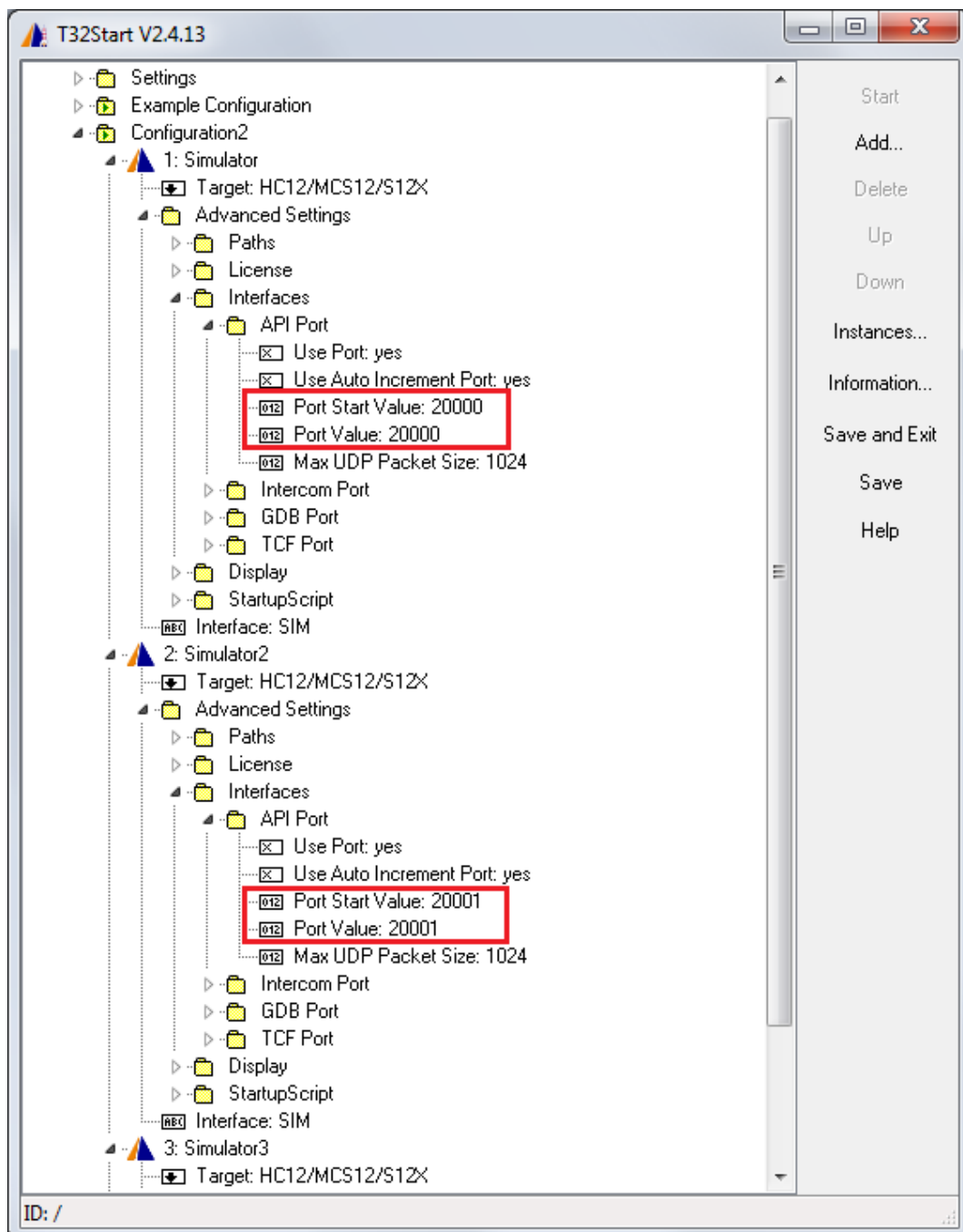
2.6 Parallel Execution

There are multiple ways to accomplish the parallel execution using the TRACE32 debugger in conjunction with TESSY. Our recommended way is shown in chapter 2.6.3.

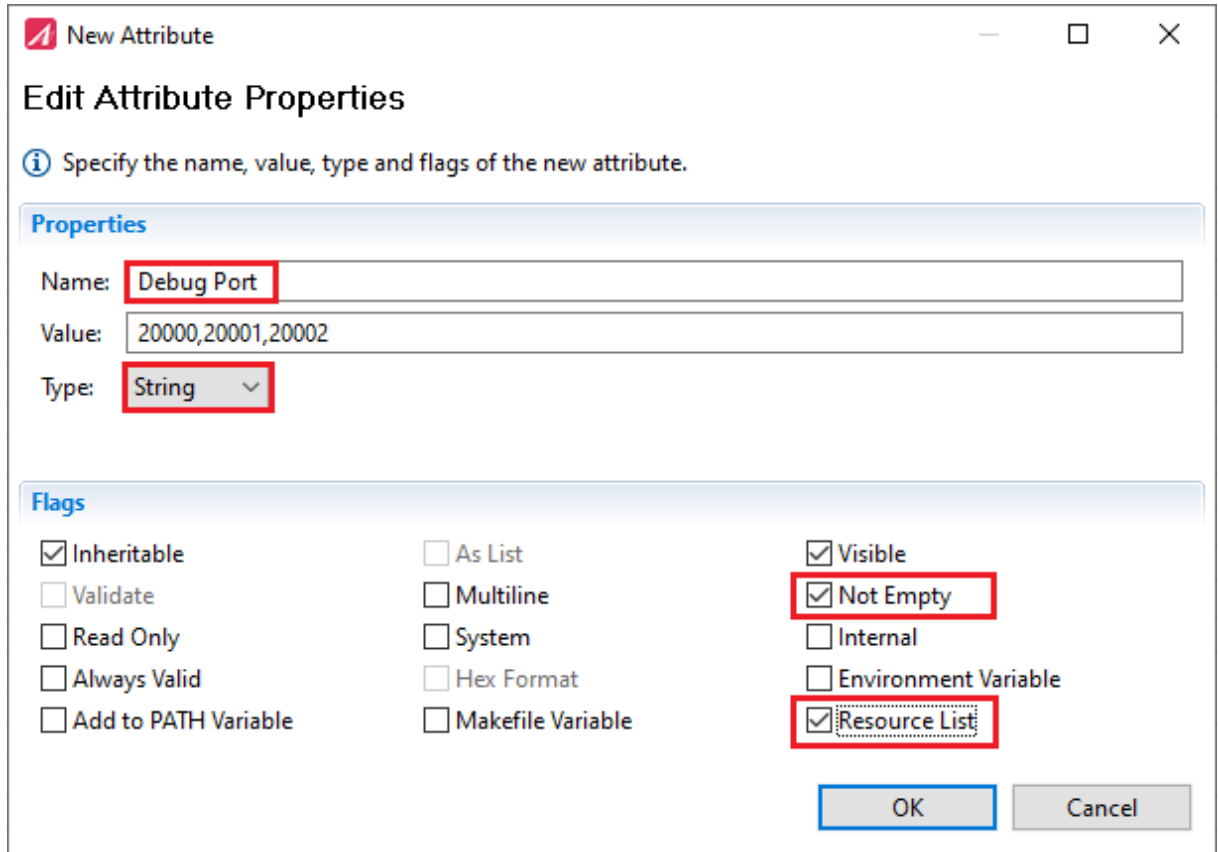
2.6.1 Using t32start.exe

In order to execute test runs in parallel using t32start.exe the following prerequisites have to be met.

- Each TRACE32 debugger instance requires its own TRACE32 license.
- Each TRACE32 debugger instance has its own port number.



- Within TEE create a new attribute **Debug Port**. If the attribute already exists delete it and create a new one as shown below. Enter the port numbers you configured within t32start.exe. The port numbers have to be separated by commas without space in between. The attribute **Debug Port** is of type *String*.



- Set attribute **Target Concurrency** to the amount of port numbers give in attribute **Debug Port**. The attribute is visible in TEE's **Expert Mode**.
- Start all TRACE32 debugger instances before you start your TESSY test runs.

2.6.2 Using the simulator being started for each test object

This solution requires a TRACE32 license for each instance which runs in parallel as well. The config.t32 has to be adapted so that the following lines are contained beside the other settings.

```
RCL=NETASSIST
PACKLEN=1024
PORT=${1}
```

Create TEE attribute **Debug Port** as described in chapter 2.6.1, create TEE attribute **Preload Wait Time**, and create TEE attribute Terminate TRACE32. Fill in the TEE attribute as shown below. The **Preload Wait Time** is needed to prevent a race condition between the launching TRACE32 instance and TESSY's master, which tries to connect to the TRACE32 instance.

Debug Port	20000,20001,20002
Execution Mode	0x61
Preload Wait Time	1
Slave Call	\$(Target Install Path)\t32marm.exe -c

	\$(Target Install Path)\config.t32 \$(Debug Port)
Target Concurrency	3
Target Install Path	C:\TRACE32
Terminate TRACE32	1

The t32marm.exe has to be adapted, if you are using a different TRACE32 instance. Also, the **Target Install Path** may need to be adjusted to match your own settings. When tests are run, TESSY fills attribute **Debug Port** in attribute **Slave Call** with one port number of the given list automatically.

2.6.3 Using Pre and Post Execution Command for parallel execution

This recommended method is based primarily on the **Pre Execution Command** and the **Post Execution Command** TEE attributes. The **Pre Execution Command** is executed before the actual test run is started, while the **Post Execution Command** is called after the test run is finished. Therefore, a batch script may be called to launch all TRACE32 instances automatically at once before the test run. Whereas another batch script may be used to terminate the running TRACE32 instances. Since it may result in a race condition with TESSY's master process, if the TRACE32 instances are too slow during startup, there is a further utility available to cope with this issue, i.e. the master waits for a trigger file given in attribute **Wait For TRACE32 Trigger** before it continues with the test execution. So, the **Pre Execution Command** should finally create that trigger file to indicate to the master, that all TRACE32 instances are ready for testing. TESSY's master does *not* delete the trigger file. This has to be accomplished by the batch script which is given in the **Post Execution Command**. Except for **Target Concurrency**, all of these TEE attributes have to be created manually within the TEE. The following example configuration for the TEE may be used as a basis.

Debug Port	20000,20001,20002
Post Execution Command	\$(Post Execution Script) \$(Wait For TRACE32 Trigger)
Post Execution Script	\$(TESSY_SYSPATH)\targets\trace32\stop_trace32.bat
Pre Execution Command	\$(Pre Execution Script) \$(TRACE32 Debugger Path) \$(TRACE32 Config File) \$(Wait For TRACE32 Trigger) "\$(Debug Port) "
Pre Execution Script	\$(TESSY_SYSPATH)\targets\trace32\start_trace32.bat
Target Concurrency	3
TRACE32 Config File	\$(Target Install Path)\config.t32
TRACE32 Debugger Path	\$(Target Install Path)\simarm\t32marm.exe

You may either import the TEE modification file as follows

- Click Support->Enable Support Mode from TESSY's menu
- Select your TRACE32 environment within TEE's Project Environment view
- From the view's toolbar select the Load Modification icon

\$(TESSY_SYSPATH)\targets\trace32\TRACE32-Parallel Execution.mod

or create all TEE attributes manually: Create attribute **Debug Port** as described in chapter 2.6.1. **Target Concurrency** should be set to the amount of ports given in attribute **Debug Port**. The **script**, **file**, and **path** attributes should be created as

type `File`, the `command` attributes as type `String`. As described in chapter 2.6.2, the TRACE32 `config.t32` file should contain the following lines.

```
RCL=NETASSIST
PACKLEN=1024
PORT=${1}
```

The above given TEE and TRACE32 configurations allow to determine how many TRACE32 instances will run in parallel simply by the given port numbers in attribute **Debug Port** and the amount of running debugger instances given in attribute **Target Concurrency**, which has to be the same amount as the amount of port numbers given in attribute **Debug Port**.

3 Troubleshooting

3.1 Init/Start/Stop Script Problems

If you are using PRACTICE command scripts to be executed by TESSY, please make sure that these scripts **do not** contain the **enddo** statement, because this terminates the script and prevents the execution of the required synchronization commands that are automatically added at the end of the script file.

3.2 Sync State Polling Delay

Some older versions of TRACE32 may require an extra wait time during the polling of the sync state. You can use the TEE attribute **Sync State Polling Delay** to set the wait time in milliseconds if the test run gets stuck in function `ts_trace32_sync()`.

3.3 Simulator Fails Using VLE

The TRACE32 simulator needs a special script command to run binary programs compiled with the VLE feature. So, your initialization script should look similar to the following example. The VLE command has to be added after your CPU commands.

```
; This is a TESSY generated init script for Trace32
; You may edit this script but BE CAREFUL WITH ANY CHANGES!

; The following variables will be added automatically by TESSY before
; execution of this script:
;
; &TESSY_BINARY_FILE      The name of the generated binary file
; &TESSY_TESTOBJECT_NAME  The name of the current test object
sys.CPU MPC5606S
system.UP
system.OPTION DISMODE VLE
b.delete
```