UM1598
User manual

Application Trace Logger

Introduction

The Application Trace Logger (ATL) provides a unified solution for trace capture and visualization in STLinux-based systems. It consists of a set of utilities running on the board (providing support for trace generation) and tools running on the host workstation for trace capture and display.

The ATL software contains all of the software components required for generating and capturing trace data. It supports the visualization of that data from the STM-Probe or from data written to a file by the board application, regardless of the origin of the trace data: user or kernel, live or post-mortem, system trace or Ethernet.

The ATL includes Multi-target trace (MTT), which is a framework running on the board for configuring and generating traces. From the user's point of view, the visible part of MTT is the API, as described in *MTT API Reference manual* (Doc ID 023121).

The ATL provides both a GUI and a command line interface on the host. ATL is suitable for users who do not have specialist knowledge of the underlying trace software.

The Application Trace Logger consists of a set of utilities running on the board (providing support for trace generation) and tools running on the host workstation (providing trace capture and display).

This manual assumes that the reader has a basic knowledge of developing embedded software for STMicroelectronics SoCs and using tracing as a technique for debugging embedded software.

This document is intended to be a "Quick Start" manual to enable users to start working with ATL as soon as possible.

Contents

List of figures

Preface

Comments on this manual should be made by contacting your local STMicroelectronics sales office or distributor.

Document identification and control

Each book in the documentation suite carries a unique identifier of the form:

Doc ID nnnnnnn Rev x

where, nnnnnnn is the document number, and x is the revision.

Whenever making comments on a document, the complete identification nnnnnnn Rev x should be quoted.

Documentation suite

Application Trace Logger user manual (Doc ID 024051)

This document provides information on how to use the STLinux trace implementation.

Application Trace Logger graphical interface HTML documentation

This is the Application Trace logger graphical interface documentation, in HTML format, accessible from a web browser (such as Internet Explorer).

MTT API Reference manual (Doc ID 023121)

This manual contains a full description of the MTT API for instrumenting code in user and kernel space.

Conventions used in this guide

General notation

The notation in this document uses the following conventions:

- `sample code`, `keyboard input` and `file names`
- *variables*, *code variables* and *code comments*
- `equations` and `math`
- **screens**, **windows**, **dialog boxes** and **tool names**, **instructions**

Software notation

- Syntax definitions are presented in a modified Backus-Naur Form (BNF).
- Terminal strings of the language, that is those not built up by rules of the language, are printed in teletype font. For example, `void`.
- Non-terminal strings of the language, that is those built up by rules of the language, are printed in italic teletype font. For example, *name*.
- If a non-terminal string of the language starts with a non-italicized part, it is equivalent to the same non-terminal string without that non-italicized part. For example, `vspace-`*name*.
- Each phrase definition is built up using a double colon and an equals sign to separate the two sides (‘`::=`’).
- Alternatives are separated by vertical bars (‘`|`’).
- Optional sequences are enclosed in square brackets (‘`[`’ and ‘`]`’).
- Items which may be repeated appear in braces (‘`{`’ and ‘`}`’).

Terms and acronyms

Table 1 lists some of the acronyms used in this document that the reader may be less familiar with.

Table 1. Acronyms used in this document

Acronym	Definition
ATL	Application Trace Logger
STM	System Trace Module
GUI	Graphical User Interface
MTT	Multi-Target Trace (the trace engine running on the board)
RPM	RPM Package Manager (see *http://rpm.org*.)
RCP	Rich Client Platform
STP	System Trace Protocol (see *http://www.mipi.org/specifications/debug*.)
YUM	Yellow dog Updater, Modified (see *http://linux.duke.edu/projects/yum/*) **STMYUM** is a version of YUM that has been customised for STLinux.

Acknowledgements

1 Product overview

Application Trace Logger (ATL) is a host application to log trace messages generated by the Multi-Target Trace (MTT) library or other sources, such as `printk`. Trace messages can be logged without the need for any specialist knowledge of the trace software.

Note: For more information about the MTT library, see the MTT API Reference manual (Doc ID 023121)

ATL can log all software traces, regardless of their origins. It can log traces from user or kernel space, live or post-mortem, system trace port or filesystem.

The tool natively supports the STM-Probe trace capture box, and optionally is able to connect to commonly used third party capture probes and import trace data in their native formats.

The tool consists of components that run on the host and the board (as shown in *Figure 1*).

- On the host: the tools **mttatl** and **mttalgui**. These configure and capture traces from a board, either from the command line or from a GUI.
- On the board: the tools for generating traces. (These are shown in more detail in *Figure 2*).

Figure 1. Overview

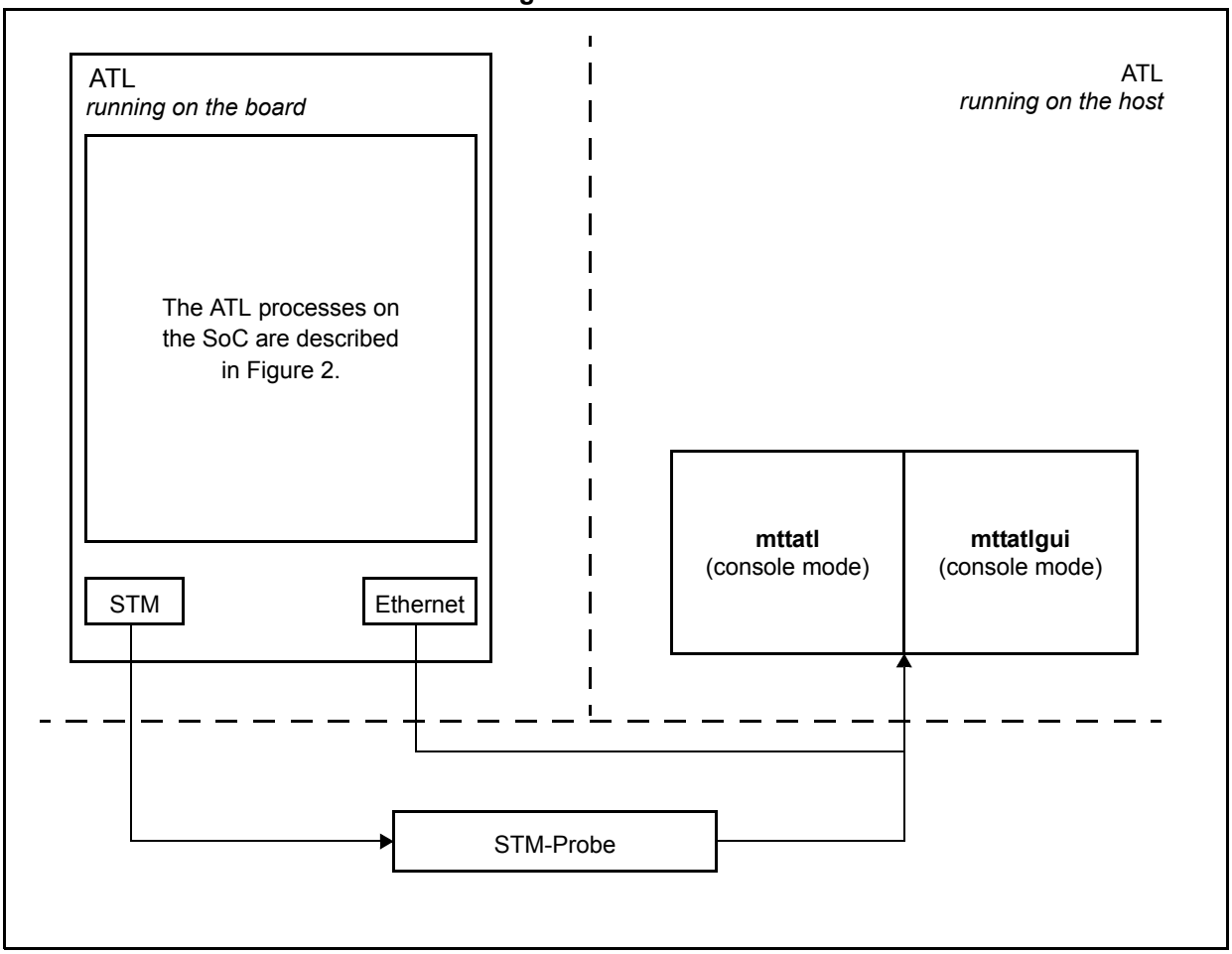
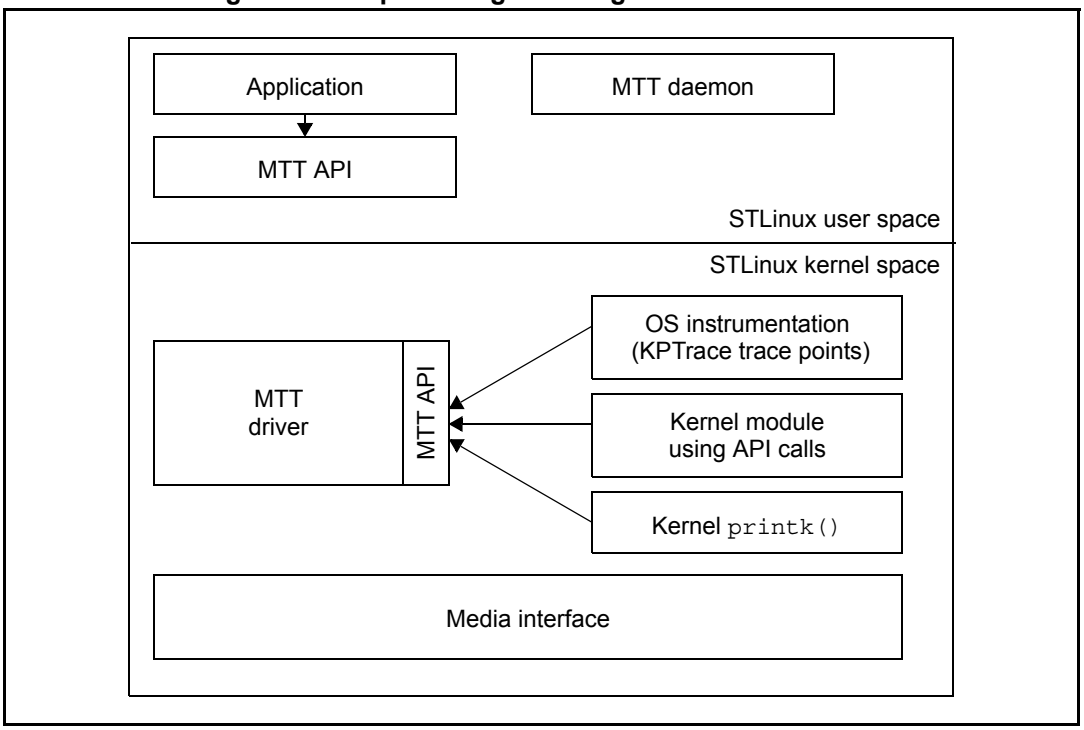


Figure 2. Components generating traces on the board



ATL communicates with the board in two ways.

- **Configuration** is done through a TCP-IP socket connection. When an IP address is specified when using ATL, ATL uses that IP address for configuration. Configuration allows the user to setup KPTrace, trace levels, trace start/stop, and so forth.
- **Data capture** is done though trace transport mechanism. This is typically STM through STM-Probe, but local storage can also be used.

There are exceptional cases, such as boot log or when STLinux is not running (bare-machine only), where no configuration service is listening. In these cases, ATL does not need to connect to the board but simply listens on the STM port.

1.1 Supported use cases

ATL can be used to generate and display traces in any combination of the cases listed in *Table 2*.

Table 2. Supported use cases

Use case	Board	Remote host	Comments
Kernel space: printk	`printk` routed to MTT console service	– Decode MTT protocol – On the fly display of user payload	ATL provides a console service **mttS**: `bootargs console=mttS`
User space: User trace points	Manual instrumentation using MTT-API calls	– Connect to the board – Configure trace generation (items to trace, filter level and so forth) – Decode MTT protocol – on the fly display of user payload	See *MTT API reference manual* (Doc ID 023121).
Kernel space: User trace points			
Kernel space: kernel traces	KPTrace		KPTrace (kprobes) trace points routed to STM port / Using MTT protocol.
User space: function I/Os	Instrumentation through compiler options		Option: `-finstrument-functions`

1.2 Interfaces

ATL provides the following types of interfaces for software traces on STLinux:

- configuration and control, using a socket connection
- trace acquisition, typically using an STM port

The configuration and control of the tracing is performed by a daemon called **mttd**. This daemon is controlled in any of the following ways:

- from the remote host, using ATL either in command line mode (**mttatl**) or from the GUI (**mttatlgui**, which is a front-end to the command line tool)
- using the direct command `/etc/init.d/mttd`

If the daemon is not running, data acquisition only can be launched from the remote host using either a command line user interface or a GUI.

- The graphical user interface is **mttatlgui**. This provides a graphical front end on top of the command line tool.
- The command line interface is **mttatl**.

For backward compatibility, there is also a **kptrace** user agent and a utility named **mtt2kpt** for generating KPTrace text files.

All output data are compliant with the MTT data protocol, and can be displayed by ATL in the host workstation.

2 Product installation

The product is packaged as three RPMs:

- Application Trace Logger for the host workstation
- KPTrace package for the host workstation
- KPTrace package for the board

ATL is an integral part of the SDK2 package. When installing SDK2, ATL and its dependencies are automatically installed by default. The ATL components are listed in the SDK2 file `armv7.rpmlist`, as follows:

- `stlinux24-host-mttatl-<version>`
- `stlinux24-host-kptrace-<version>`
- `stlinux24-${ARCH}-kptrace-<version>`

Section 2.1 and *Section 2.2* describe how to complete a manual installation or update. It is not necessary to complete these steps if SDK2 has already been installed.

2.1 Dependencies

The ATL contains an Eclipse-based Rich Client Platform application that has dependencies upon Java and certain graphical libraries. These dependent packages must be installed before the ATL.

Figure 3 lists the commands to install these packages on a freshly installed Fedora 17-64 machine.

Figure 3. Install Java and graphics libraries

```
# yum -y update
# yum -y install java-1.7.0-openjdk.i\*
# yum -y install perl libICE.i686 ORBit2.i686 libSM.i686 libX11.i686
libXft.i686 libXrender.i686 libXt.i686 libart_lgpl.i686 atk.i686
libbonobo.i686 glibc.i686 cairo.i686 libbonoboui.i686 fontconfig.i686
freetype.i686 libgcc.i686 GConf2.i686 gtk2.i686 gdk-pixbuf2.i686
glib2.i686 libgnome.i686 libgnome-keyring.i686 libgnomecanvas.i686
libgnomeui.i686 gnome-vfs2.i686 xulrunner.i686 pango.i686 nspr.i686
popt.i686 nss-softokn.i686 nss.i686 libstdc++.i686 libusb1.i686
libxml2.i686 zlib.i686 libstdc++PackageKit-gtk3-module.i686
libcanberra-gtk2.i686 gtk2-engines.i686
```

2.2 Installation

When installing ATL, STMicroelectronics recommend that you install corresponding versions of KPTrace to avoid the risk of version mismatch.

- `stlinux24-host-mttatl-<version>.i386.rpm`, containing the host workstation **mttatl** package, including:
  - **mttatl**, the command line interface to launch a trace session on the board
  - **mttatlgui**, the graphical interface launcher

- `stlinux24-armv7-kptrace-4.<version>.armv7.rpm`, containing:
  - the **kptrace** user command
  - **mttd** trace daemon for enabling and disabling trace, to configure the trace output port and the **init.d** component for **mttd**
  - **mtt2kpt** utility for converting a raw trace file to a KPTrace text file
  - `libmtt.{so,a}` and `libkptrace.so` for user-space traces
  - `mtt.h` , `mtt_types.h` and `kptrace.h`
- `stlinux24-host-kptrace-4.<version>.i386.rpm`, containing:
  - **mtt2kpt** utility for converting a raw trace file to a KPTrace text file
  - **Kptrace.pl** utility to display KPTrace trace files with symbol resolution

Use the following command lines to install ATL using **stmyum**:

```
stmyum install -y stlinux24-host-mttatl-<version>.i386
stmyum install -y stlinux24-host-kptrace-4.<version>.i386
stmyum install -y stlinux24-armv7-kptrace-4.<version>.armv7
```

Use the following command lines to install ATL using **rpm**:

```
rpm -Uvh stlinux24-host-mttatl-<version>.i386.rpm
rpm -Uvh stlinux24-host-kptrace-4.<version>.i386.rpm
rpm -Uvh stlinux24-host-armv7-kptrace-4.<version>.armv7.rpm --ignoreach
```

*Note: The ATL package has dependencies on other packages. These packages will be installed by the **stmyum** command, as the dependency rules are predefined.*

Note: There are several alternative ARMv7 packages available, each built in a different way. If a library is not specified in the name of the package, the package is built with `glibc`.

Note: If you have an earlier version of ATL installed, you may see the following error message:

```
error: Failed dependencies:
stlinux24-host-traceinfrastructure = 2012.2-2 is needed by (installed)
stlinux24-host-sdk2-armv7-development-filesystem-<sdk2-version>-int<intern
al-sdk2-buildid-number>.noarch
stlinux24-host-traceinfrastructure = 2012.2-2 is needed by (installed)
stlinux24-host-sdk2-armv7-development-<SDK2-BUILDID>-<sdk2-version>-int<in
ternal-sdk2-buildid-number>.noarch
```

To proceed with the installation as described above, first use the following commands to remove the failed installation:

```
rpm -e stlinux24-host-sdk2-armv7-development-filesystem-<sdk2-version>
rpm -e stlinux24-host-sdk2-armv7-development-<SDK2-BUILDID>-<sdk2-version>
```

Note: *If you have an earlier version of ATL installed, you may see an error similar to the following displayed during installation. This error occurs because the arrangement of the packages has been simplified.*

```
file /opt/STM/STLinux-2.4/devkit/armv7/target/usr/bin/mttd from install of
stlinux24-armv7-kptrace-4.0.0_<version>.armv7 conflicts with file from
package stlinux24-armv7-mttatl-1.1-3.armv7
```

To prevent this error, use the following command to remove the previous package before installing the new one:

```
rpm -e stlinux24-armv7-mttatl-1.1-3
```

Next, re-run the installation.

```
rpm -Uvh stlinux24-armv7-kptrace-4.<version>.armv7.rpm --ignorearch
```

2.3 Product directory tree

The following sections describe the location and the contents of each of these three parts.

In the following sections, `<STLINUXINSTALL>` is the path of the installation directory of the STLinux distribution. (For the standard STLinux 2.4 distribution this defaults to `/opt/STM/STLinux-2.4/`.) The examples throughout *Section 2.3* use the environment variable `${STLINUXINSTALL}` to define the installation path.

2.3.1 Board directory tree

The board binaries are installed in the following location on the host:

`${STLINUXINSTALL}/devkit/armv7/target/`

The files are listed in *Table 3*. The paths given in *Table 3* are in respect of the board filesystem.

Table 3. Location of board binaries

Location	File	Comments
`${STLINUXINSTALL}/devkit/armv7/target/`		
`etc/rc.d/rc3.d`		
	`S99mttd`	Link to `/etc/init.d/mttd`
`etc/init.d`		
	`mttd`	The MTT daemon script.
	`kptrace.conf`	A copy of `kptrace.min.conf`
	`kptrace.full.conf`	KPTrace full configuration.
	`kptrace.min.conf`	KPTrace minimal configuration.
`/usr/bin/`		
	`mtt2kpt`	Utility to extract KPTrace text files from raw trace.
	`mttd`	The MTT daemon.
	`kptrace`	Command line utility to start and stop KPTrace.
`/usr/include/`		
	`kptrace.h`	MTT API header files.
	`mtt.h`	
	`mtt_types.h`	
`/usr/lib/`		
	`libkptrace.so`	A link to `libmtt.so`.
	`libmtt.so`	MTT API implementation for Linux user space.
	`libmtt.a`	
`/usr/share/man/man1`		
	`kptrace.1`	man file for KPTrace.
	`mtt2kpt.1`	man file for the **mtt2kpt** utility.

2.3.2 Host directory tree

The host tools are installed in the following locations: The files are listed in *Table 4*.

Table 4. Location of host tools

Location		File	Comments
/opt/STM/STLinux-2.4/host/			
	bin/		
		mtt2kpt	Utility to extract KPTrace text files from raw trace.
		kptrace.pl	
	man/		
		mtt2kpt.1	Utility to extract KPTrace files from MTT files.
/opt/STM/STLinux-2.4/mttatl/			
	bin/		
		mttatl	ATL command line tool.
		mttatlgui	ATL GUI tool.
	documentation/		Documentation directory.
	firmware/		STM-Probe related software.
	system/		System setup utilities.
	trcviewer/		Graphical user interface.
/opt/STM/STLinux-2.4/host/doc			
		stlinux24-host-mttatl-<*version*>	License files

Note: ***mttatl*** *and* ***mttatlgui*** *are both scripts that set essential environment variables before calling the associated executables. Executables should not be called directly as they need the appropriate environment variables.*

2.4 Configure the kernel

STMicroelectronics recommends that first you copy the directory ${STLINUXINSTALL}/devkit/build to a temporary working directory (<*dev-dir*>) before performing any configuration. Then, run the following commands:

```
cd <dev-dir>/sdk2-build.b2020_a9
make menuconfig
```

The MTT support option is in the category `Kernel hacking: Multi-Target Trace (MTT) infrastructure support` in `menuconfig`.

- Check that an entry for configuring MTT support appears in the `menuconfig` output, as shown in *Figure 4*.
- Enable KPTrace as a built-in, as also shown in *Figure 4*.
- Check that the System Trace Module driver support is enabled, in section **Device Drivers > STM specific devices** as illustrated in *Figure 5*.

Figure 4. Trace infrastructure in the kernel

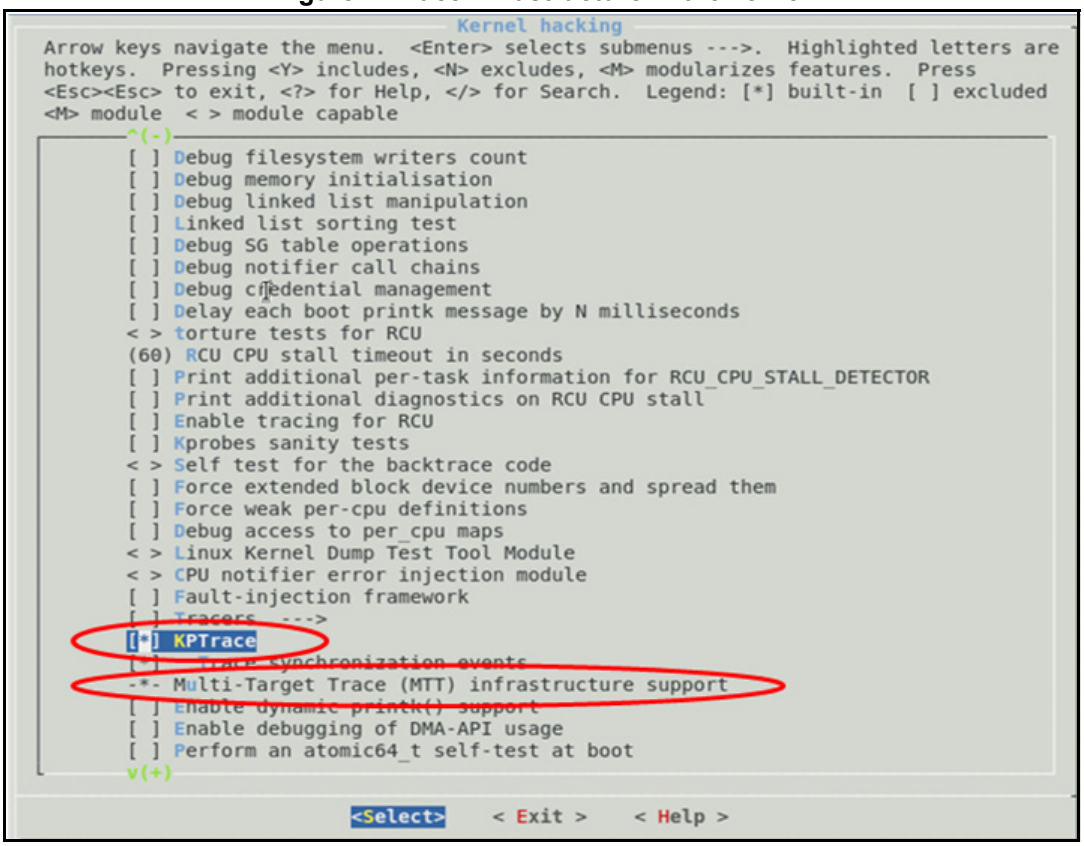
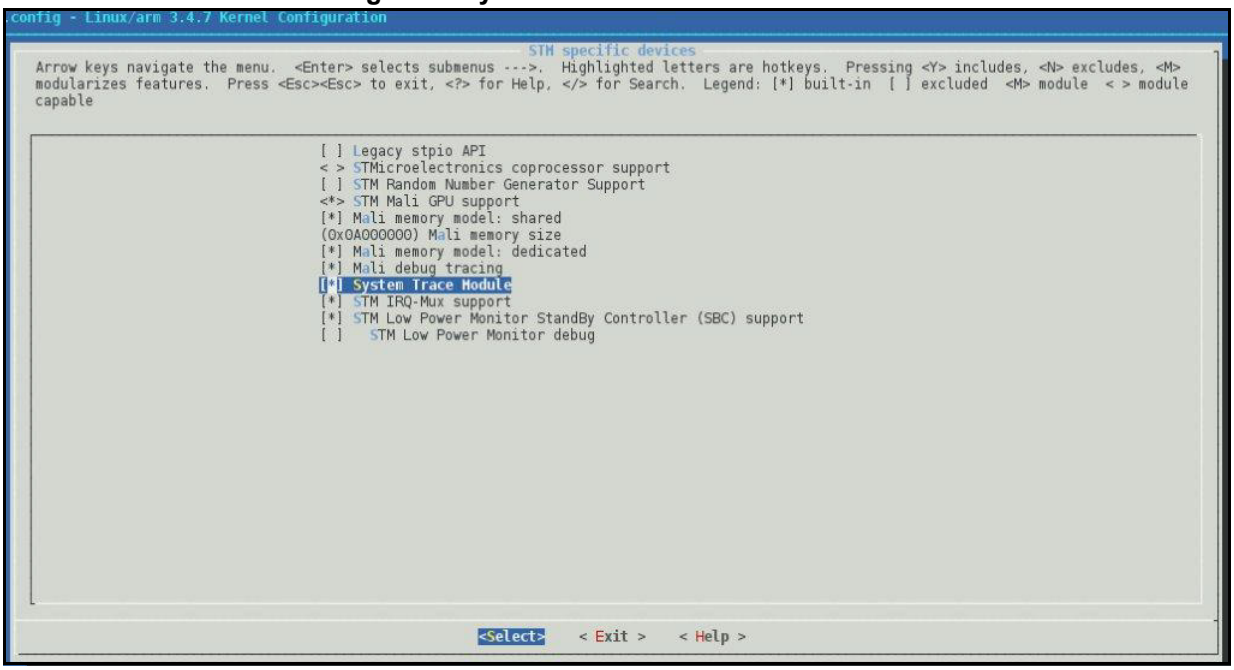


Figure 5. System trace module in the kernel



2.4.1 Rebuild the kernel

Rebuild the kernel to enable the support. The command is:

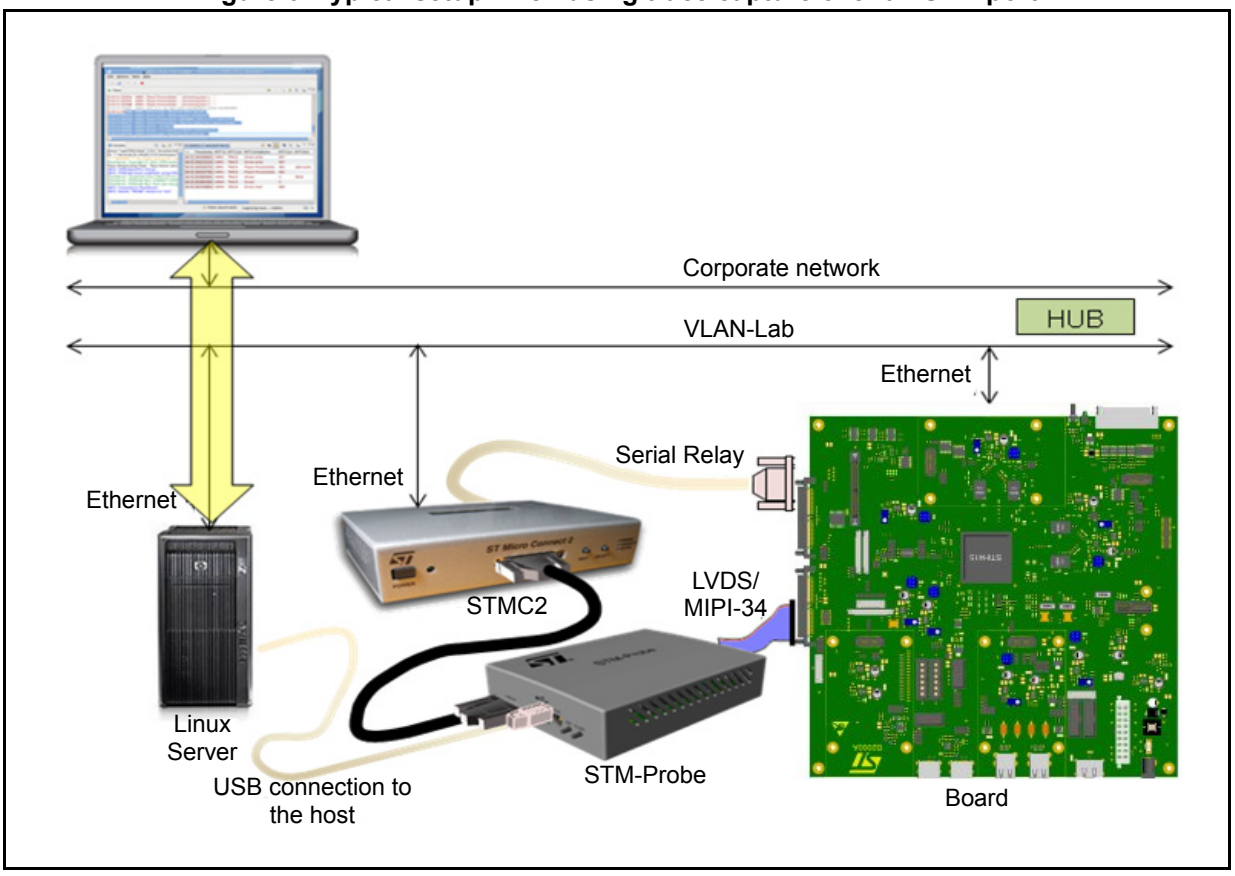
```
make clean all
```

3 Required environment

3.1 Trace capture over an STM port

Figure 6 shows a diagram of the typical setup for using trace capture over the STM port of the SoC.

Figure 6. Typical setup when using trace capture over an STM port



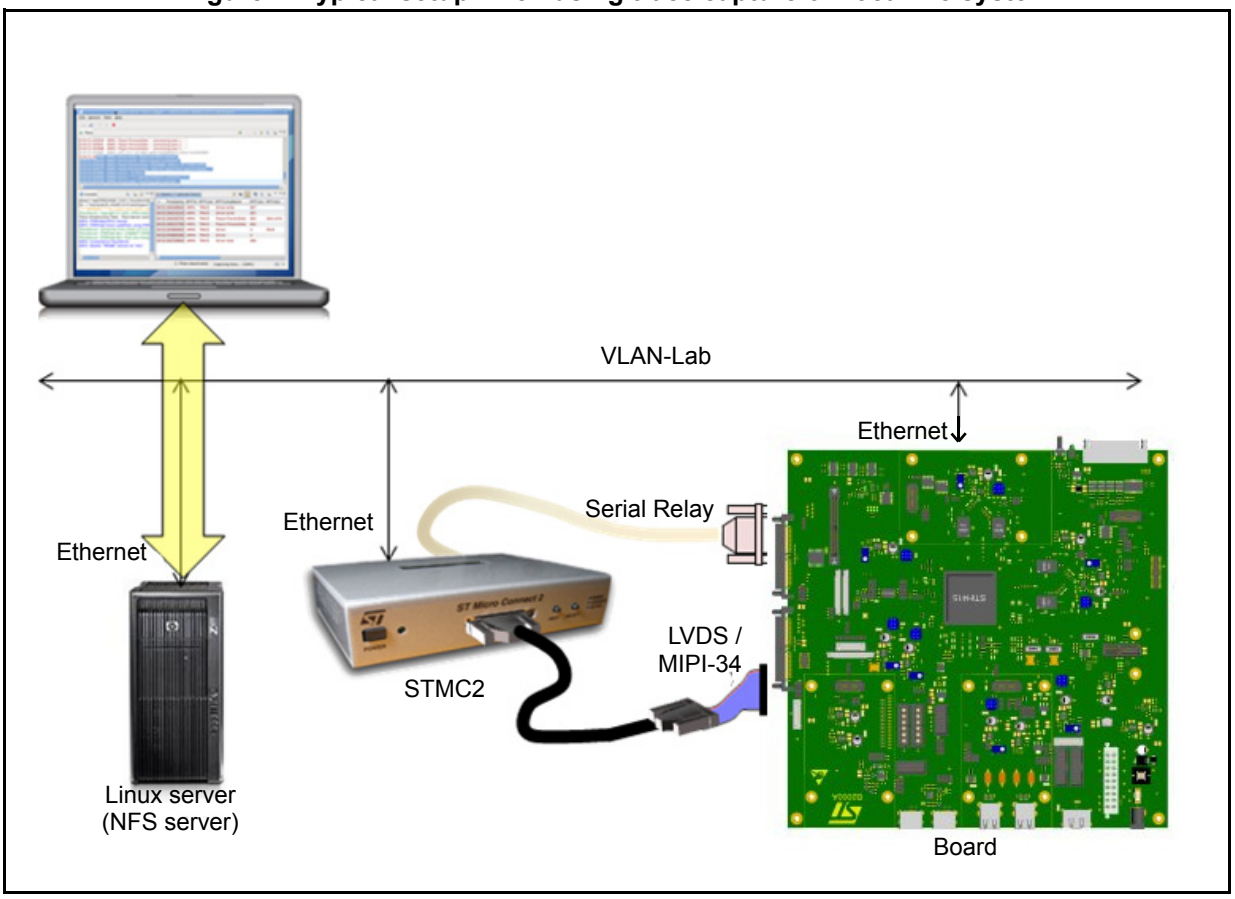
Note: The STM-Probe connects to the host using USB. The STMC2 connection is an electrical relay from the board.

Note: When using a board with a MIPI-34 connecction, it is important to use an IO converter to interface with the LVDS cable. The IO converter must be a "Type-H" as this is the only version that routes the STM signal from MIPI to LVDS.

3.2 Data capture on local storage

Figure 7 shows a diagram the typical setup for using trace capture on local storage on the board file system.

Figure 7. Typical setup when using trace capture on local file system



4 Getting started

This chapter describes how to use the ATL GUI.

4.1 Launch ATL GUI

The launcher is located in `${STLINUXINSTALL}/host/mttatl/bin`, where `${STLINUXINSTALL}` is the path to the STLinux distribution. (In the standard STLinux distribution, this is `/opt/STM/STLinux-2.4/`.)

Start ATL from a shell by entering:

`${STLINUXINSTALL}/host/mttatl/bin/mttatlgui`

When ATL launches, it first displays the **Application Trace Logger Launcher** dialog (see *Figure 8*).

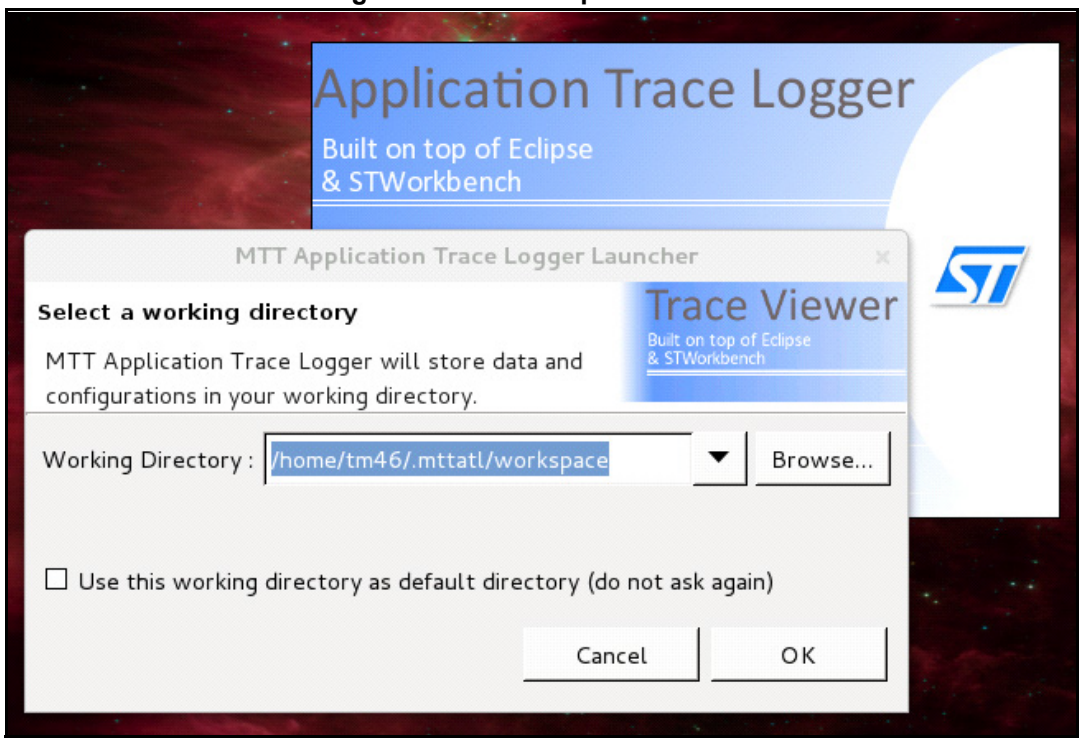
By default the working directory is your home directory, but you may change it to a different directory if you wish. Use this dialog to enter or select the location of the working directory. This is the directory where the project data, files and directories are stored.

If the named working directory does not already exist, it is created. The default name for this directory of `/home/${USER}/.mttlatl/workspace`.

Note: Do not use spaces in the working directory path and name as this may cause problems with the tools.

Note: Trace sessions are stored by default in this directory. Make sure that there is sufficient disk space for the trace files.

Figure 8. ATL workspace launcher



Click OK to display the main ATL window. See *Figure 9*.

4.2 ATL GUI overview

The ATL GUI provides several views for the user. These are:

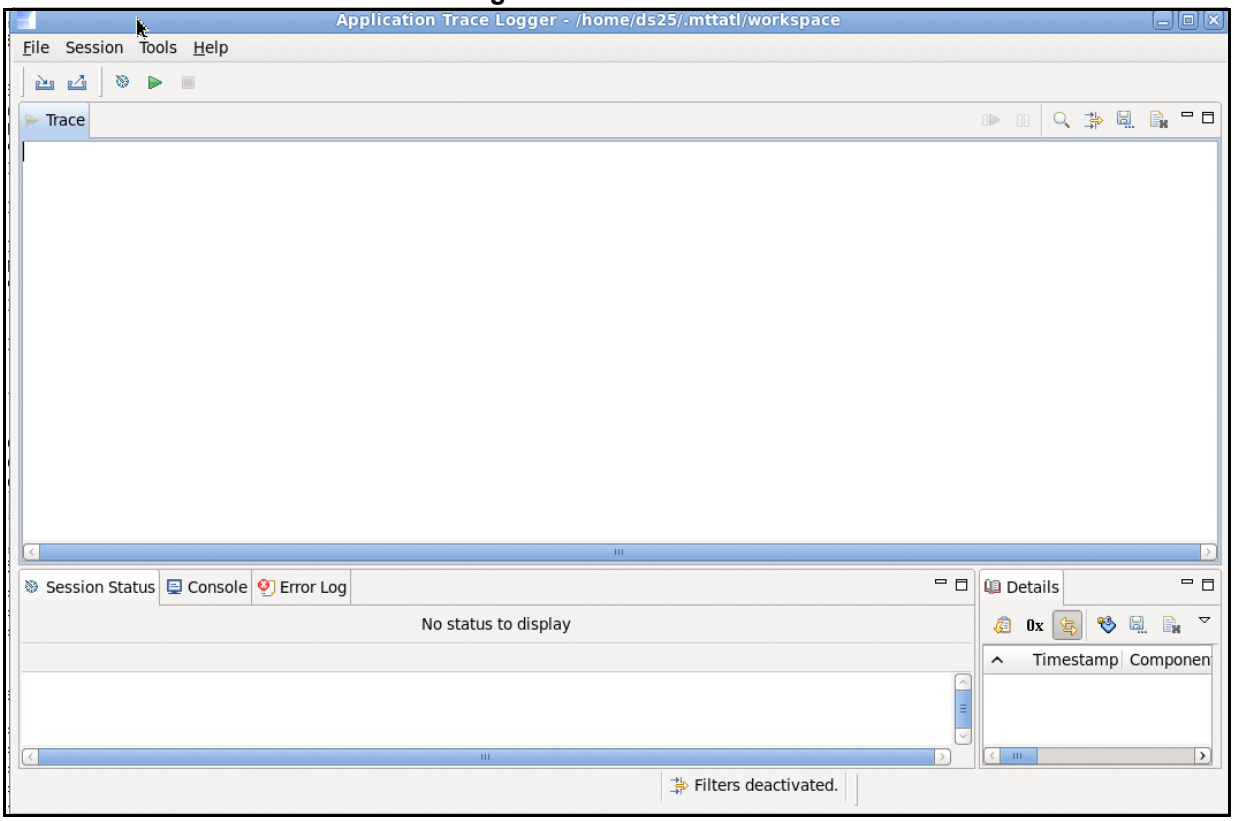
- **Trace** to display the trace data as it is generated (as though displayed in a console)
- **Session Status** to provide information regarding underlying tools, connection status and statistics
- **Details** to provide a detailed view of traces selected in the main trace window
- **Console** to provide information regarding underlying tools
- **Error log** to display messages from the Graphical User Interface
- **Terminal** to provide support to connect to the board using `ssh`

By default, the GUI perspective (that is, the arrangement of views) is organized so that three of these views (**Trace**, **Session Status** and **Details**) are prominent. See *Figure 9*:

This arrangement can be altered by reducing or maximizing any given view or by selecting and moving the window tabs. On exit from the GUI, the GUI saves the current perspective. On re-launch, the GUI opens with the most recent saved perspective.

The perspective can be reset to the default settings by selecting **File > Reset Perspective** from the menus.

Figure 9. ATL main view



4.3 Trace session management

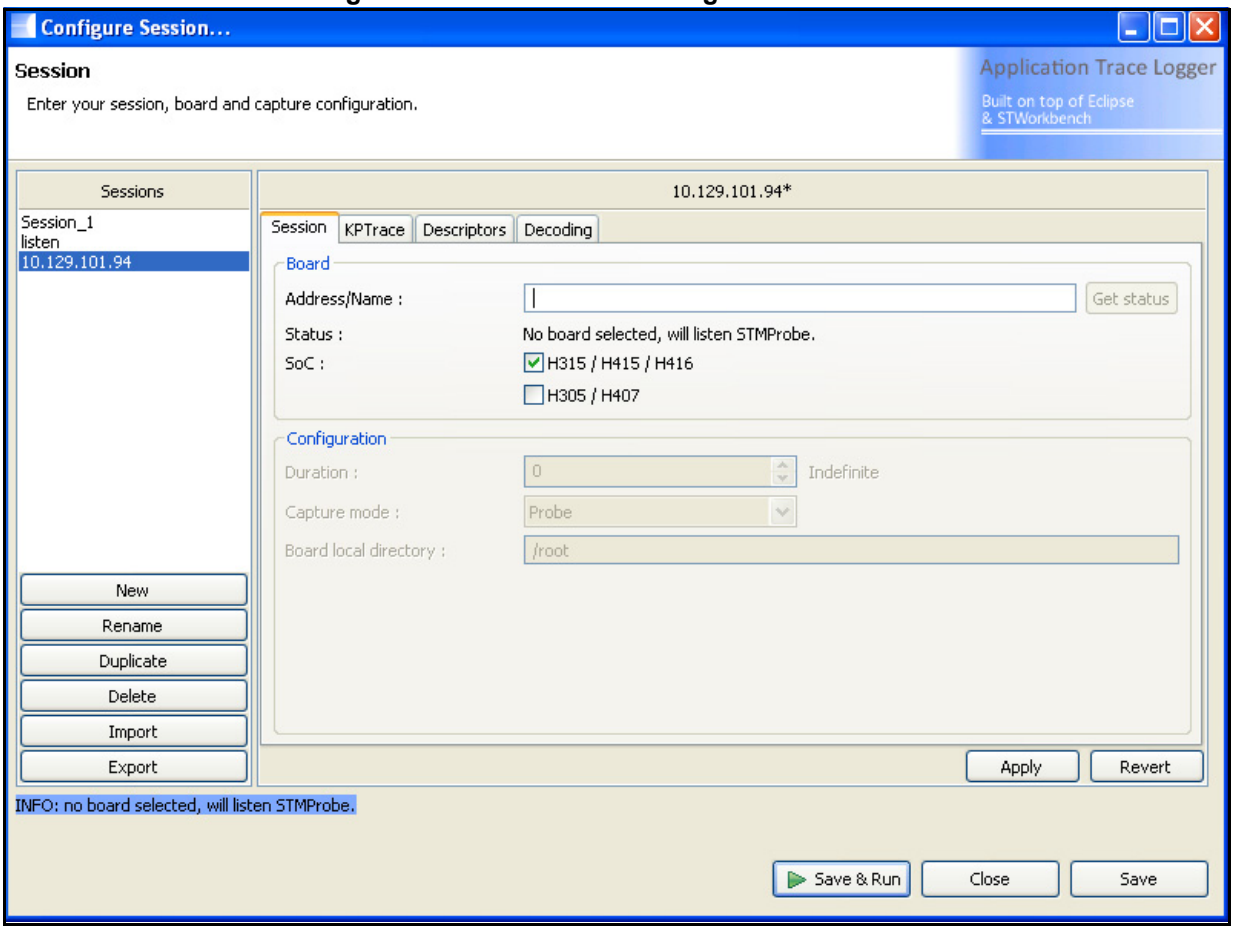
Two main scenarios are available from the GUI.

- Trace capture only (that is, without any on-board configuration). Use this configuration to trace `printk` messages from boot only.
- Trace capture with configuration of the MTT daemon. Use this configuration for all trace use cases *except* tracing `printk` messages from boot.

The first step in both scenarios is to configure and launch a trace session. To do this, select **Session > Configure**.

The ATL GUI displays a dialog box describing the session (see *Figure 10*).

Figure 10. Trace session configuration window



4.3.1 Configure and launch a trace session

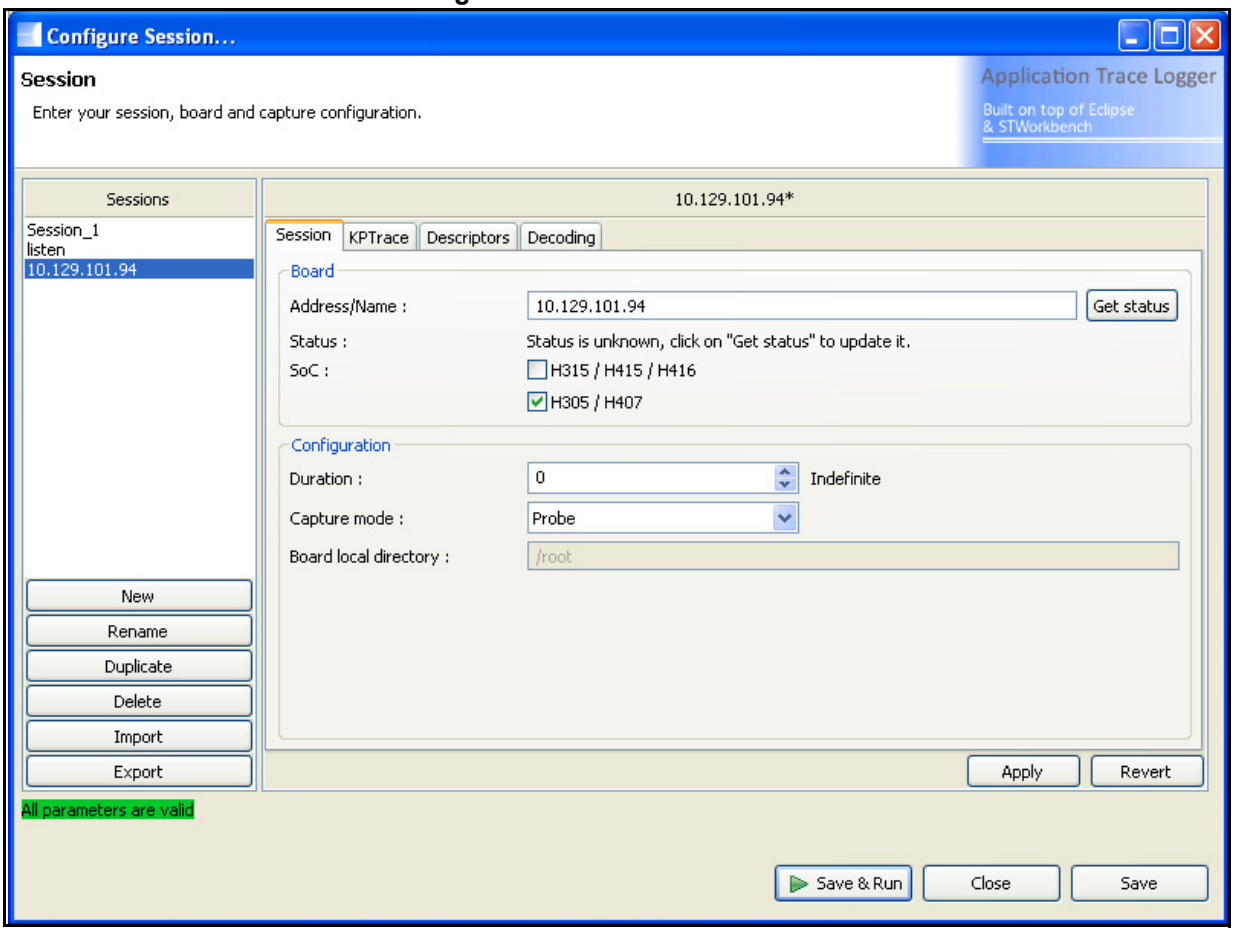
Trace session configuration consists of attaching to the board to send configuration commands to the MTT daemon running on the board. The configuration is carried out through a socket connection.

The procedure is as follows:

1. Enter the board IP address in the **Address/Name** text field.
2. Select the capture mode: If you do not have an STM-Probe connected you must choose “Board FileSystem”.
3. Select the type of SoC that you are using. The System Trace protocol is different for different SoC families, so it is important that the trace capture chain is configured correctly. (This step is only relevant for trace over trace port.)
4. Click **Save & Run**. This action closes the session configuration dialog and launches the trace acquisition chain on the host side. On the board side, it configures the MTT daemon to generate traces according to the selected capture mode and enables trace generation.

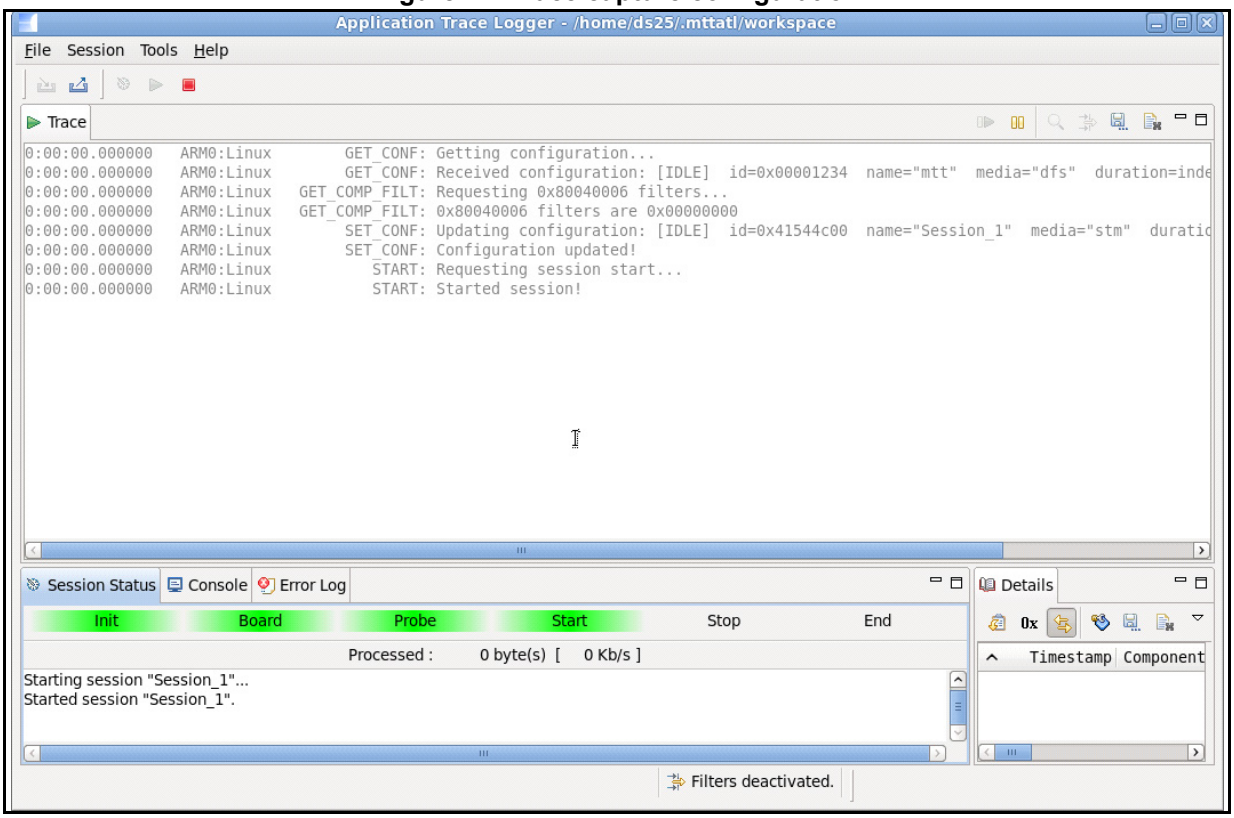
Figure 11 illustrates trace session configuration over the trace port (“Probe” is selected).

Figure 11. Connect to a board



The information that appears in the console window is shown in *Figure 12*.

Figure 12. Trace capture configuration



The GUI may display some data in the trace window. This shows that the trace framework on the board is configured and ready.

From this point, the ATL displays any data logged over the trace port as it is received.

The configuration is saved as a session that can be re-used later.

4.3.2 Stop a trace session

Either press the “Stop” shortcut (red box) or select **Session > Stop**.

If KPTrace is enabled, stopping a trace session may take a few moments to complete as the kprobes trace points must first be disabled.

4.3.3 Restart a trace session

Press the “Start” shortcut (green arrow) or select **Session > Start**.

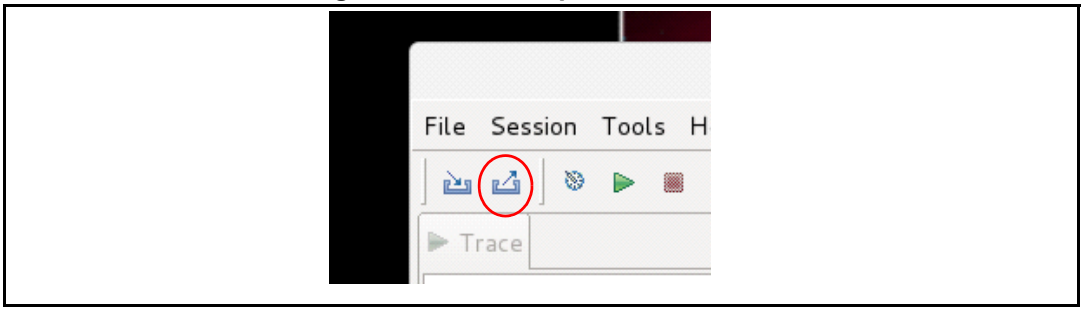
This launches the latest configured trace session.

4.3.4 Archiving a trace session

Trace data can be archived in a database for later use. Because the data have already been processed, this functionality enables the data to be processed quickly.

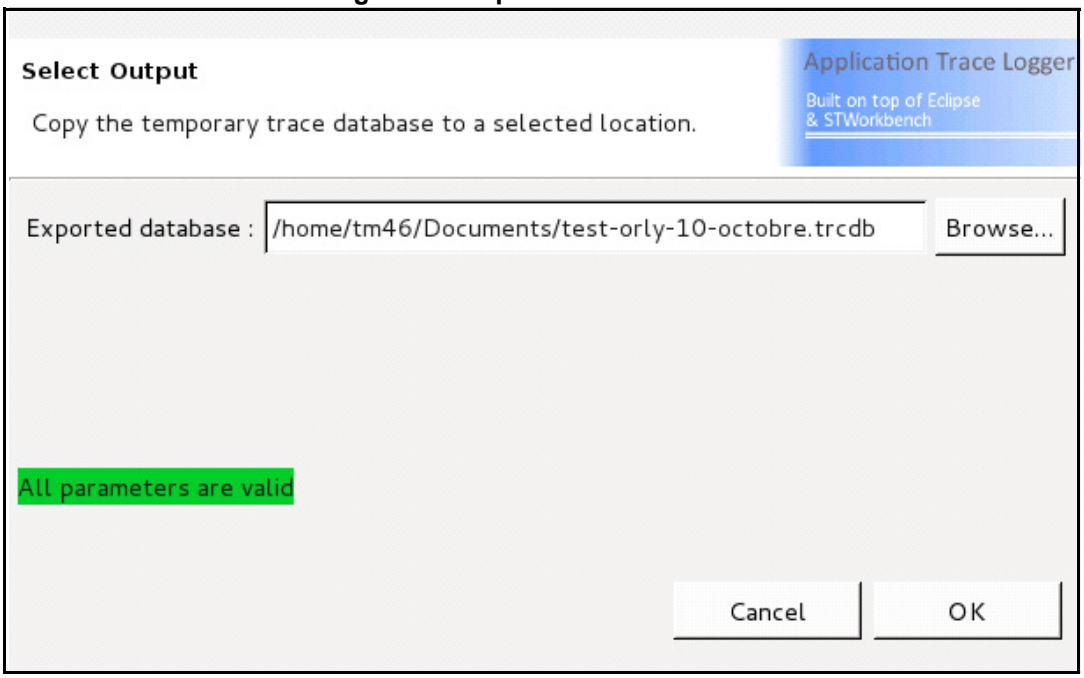
Export trace data by clicking on the export archive icon (see *Figure 13*).

Figure 13. Select export archive icon



This opens the export panel to specify the file path where to export the trace archive (as shown in *Figure 14*).

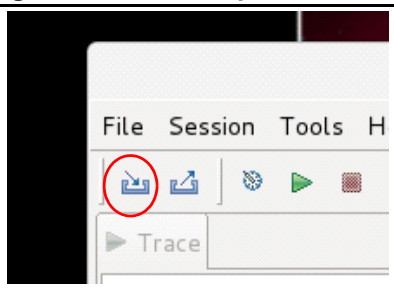
Figure 14. Export archive window



4.3.5 Reading an archive

An exported archive can be re-opened by clicking on the import archive icon. See *Figure 15.*

Figure 15. Select import archive icon



4.3.6 Saving input trace data

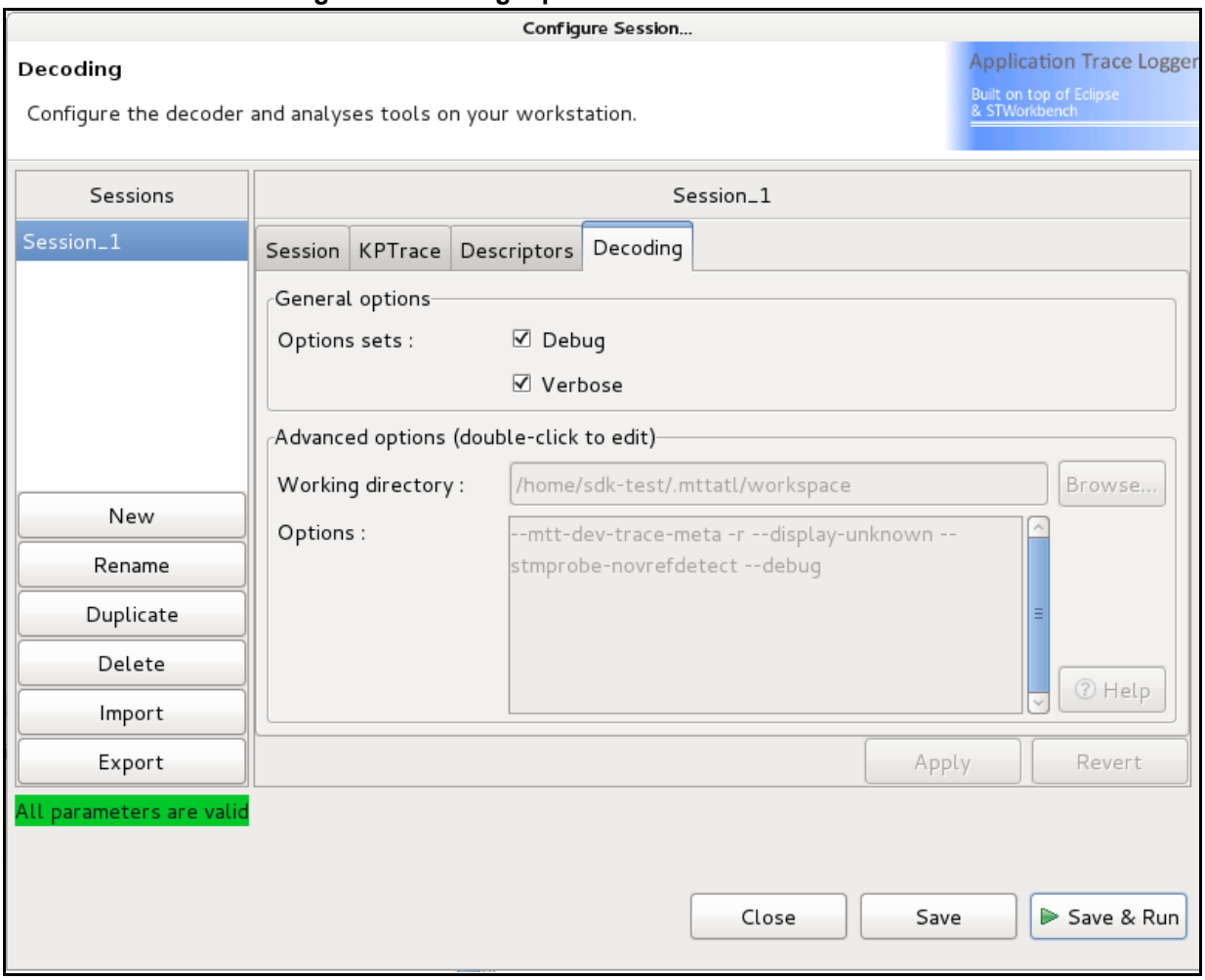
Input traces can also be stored in a raw binary file that can be opened off-line at a later date. The tool supports an STP binary output format that keeps the STM level information (initiator, channel and timestamp) plus MTT trace data.

In the **Decoding** tab (see *Figure 16*), set the option `-O stp` in the **Option** field.

Click on **Save** or **Save & Run** to save the trace data in a file. The file is saved in the working directory and takes the name of the session concatenated with the qualifier `_stp` and the extension `.bin`. In the example shown in *Figure 16*, the trace file takes the name:

`/home/ds25/.mttatl/workspace/Session_1_stp.bin`

Figure 16. Storing input traces in a raw STP format



Note: Make sure that you have sufficient disk space at the destination location when enabling trace storage.

4.3.7 Other options

The trace configuration dialog has other tabs allowing other configuration parameters (see *Figure 10 on page 23*). You can use the **Decoding** tab to pass more options to the underlying decode tools.

Detect unknown frames

In the **Decoding** tab, the **Verbose** button is a shortcut that enables the option `--display-unknown` for the underlying decoder. This can be enabled for diagnostic or display of unrecognized binary frames.

Board specific option

This option is enabled by default and is mandatory for all STiH415 and STiH416 board families. It fixes hardware defects on the MIPI-34 connector. This option enables the `--stmprobe-novrefdetect` option.

More options

The **Options** field of the **Decoding** tab provides the ability to edit the command line. All underlying tool options not handled by shortcuts described above can be passed through the command line.

Refer to the **mttatl** man page for a list of all available options.

4.3.8 Listen mode only

In the situation where the on-board daemon is not running (which will always be the case if the board has not been started) it is not possible to connect and configure the trace generation. This means that ATL can only listen on the trace port and must rely on the board's default configuration.

In these circumstances, use the following procedure.

1. Do not enter a board IP address in the **Address/Name** text field. Because ATL assumes that the board has not yet booted, it generates an error message when attempting to connect to the un-booted board. In this case, the default trace port is STM port / STM-Probe.
2. Click **Save & Run**. This closes the session configuration window and launches the trace acquisition chain on the host side. From this point, ATL listens on the STM port and does not configure anything on the board.

From this point onwards, ATL displays any `printk` messages logged over the trace port as if the MTT console is enabled.

When ATL starts operating in this mode, it is not possible to configure the trace session. The only way to configure a trace session on the board is to stop the current session and configure a new trace session as described in *Section 4.3.1: Configure and launch a trace session on page 24*.

4.4 Tracing using printk

ATL provides a number of useful features to assist with tracing using `printk`. These are described in this section.

MTT provides console support to route `printk` to MTT. This takes advantage of the STM port rather than UART.

To enable MTT as console output, the boot arguments must contain the following:

```
console=mttS
```

STMicroelectronics does not recommend enabling both UART and MTT for console output. This is because UART can introduce huge amounts of latency and increase the interrupt rate.

It is also preferable to remove the standard setting of `console=ttyAS0,115200` when setting up the MTT console.

Note: When using SDK2, bootargs can be tuned using the following command before booting the board.

```
export COMMON_BOOTARGS="console=mttS"
```

4.4.1 Enable printk trace capture when the board is running

When the board has been booted, a connection specifying the board IP enables trace capture for all kind of traces, including `printk` if the `mttS` console is enabled.

To launch trace acquisition, see *Section 4.3.1: Configure and launch a trace session on page 24*.

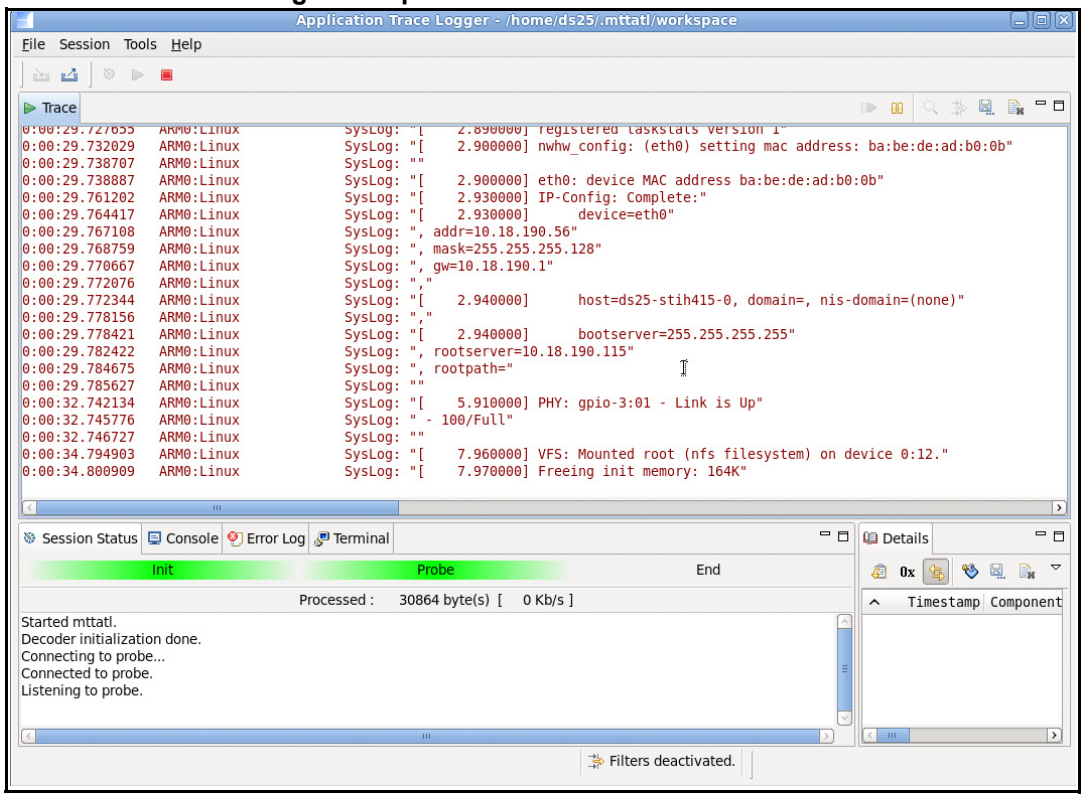
4.4.2 Enable printk from boot time onwards

When using the `mttS` console, `printk` can be traced from boot time. This is because the Multi Target Trace infrastructure support is resident in the kernel as illustrated in *Figure 4*.

If the board has not been booted, or if you are only interested in `printk` traces, it is not necessary to specify the board IP address when launching the tool. To launch trace acquisition in this mode, see *Section 4.3.8: Listen mode only on page 29*.

When using the boot argument `console=mttS`, output from `printk` is logged to the STM port from the boot onwards. *Figure 17* shows `printk` during the boot sequence.

Figure 17. printk routed to MTT at boot time



4.5 KPTrace

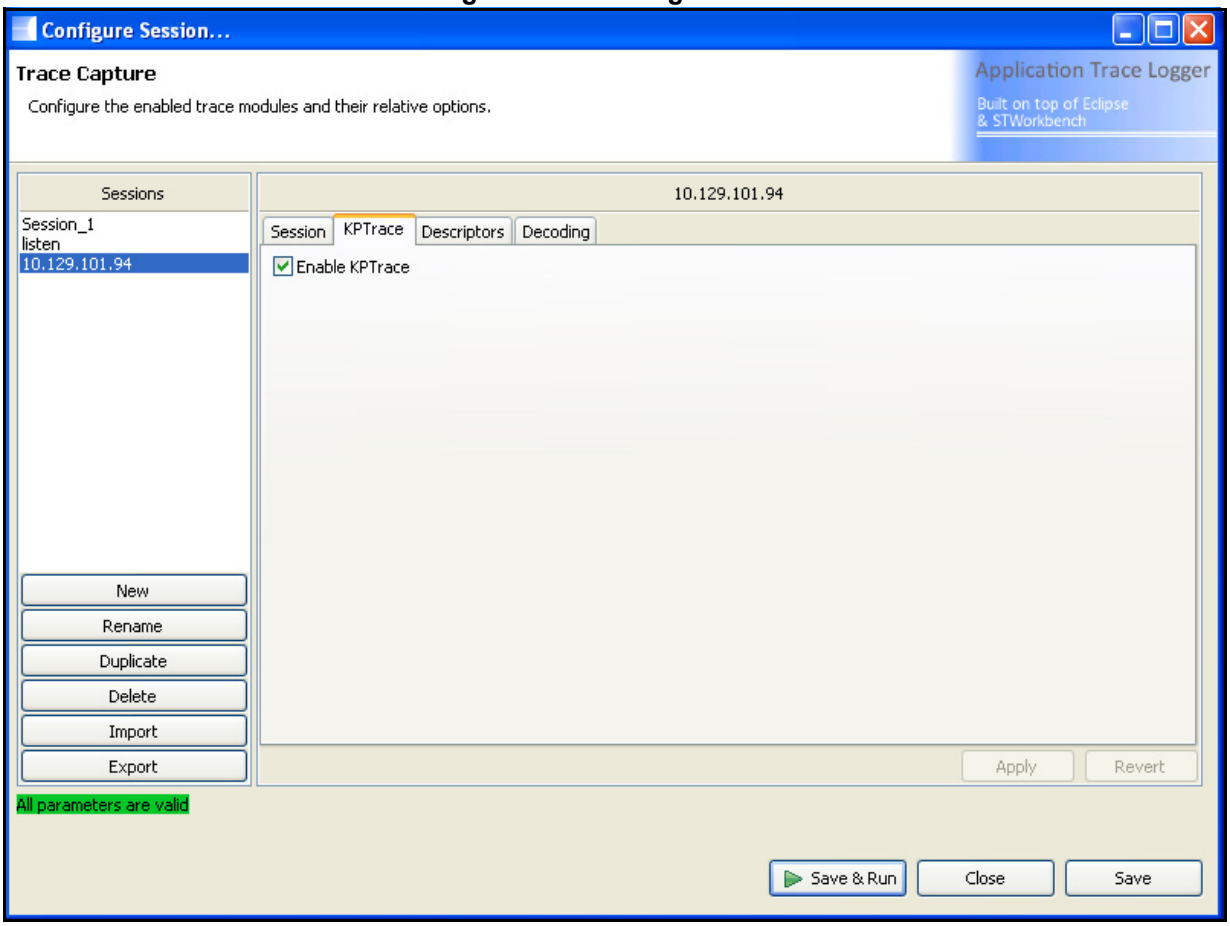
Enable KPTrace from the trace configuration tab (as shown in *Figure 18*).

When a trace session is launched, the core that is running STLinux is sent a notification to enable KPTrace (consisting of enabling a KProbe for each trace point specified in the `kptrace.conf` configuration file).

KPTrace uses the configuration file specified in `/etc`.

To toggle KPTrace activation, stop and restart the session. It cannot be toggled dynamically during a running trace session.

Figure 18. Enabling KPTrace



When the session has been stopped, ATL imports the traces into the working directory specified when launching the tool. (By default, the working directory is `/home/${USER}/.mttatl/workspace`)

Traces can be stored in a custom session directory named as follows:

`<Working Directory>/<SessionName>-<Unique ID>`

*Note: The session directory name is visible in the top of the **Console** view.*

The contents of the session directory is as follows:

- `<SessionName>-00.kpt`: KPTrace file for core 0
- `<SessionName>-01.kpt`: KPTrace file for core 1
- `<SessionName>.kpm`: symbol file (a copy of `/proc/kallsyms`)
- `<SessionName>.meta`: running processes at KPTrace startup
- `<SessionName>_stp.bin`: raw input data received from the trace port.

The session directory can then be imported into STWorkbench to display the kernel activity time-chart.

4.6 User traces

User traces are generated when using MTT API, either from user or kernel space.

User trace capture requires configuring the board to enable trace generation over the selected output port.

To launch trace acquisition, see *Section 4.3.1: Configure and launch a trace session on page 24*.

Refer to *Appendix A: Program example using MTT-API to generate traces on page 38* and *Appendix B: Kernel module example using MTT-API to generate traces on page 40* for examples.

The user trace API also provides support to KPTrace user traces (`kpprintf`, `kptrace_write_record`, `kptrace_mark`).

4.7 Function I/O traces

The user space MTT shared library provides the support to log function I/O traces by implementing the following functions:

```
__cyg_profile_func_enter (void *func , void *callsite)
__cyg_profile_func_exit (void *func , void *callsite)
```

To enable this functionality, compile the code with the option `--finstrument-function`.

It is also necessary to initialize the trace connection by adding the relevant initialization call to the code:

```
mtt_initialize( NULL );
```

This initialization enables the trace mechanism for the current process.

To launch trace acquisition, see *Section 4.3.1: Configure and launch a trace session on page 24*.

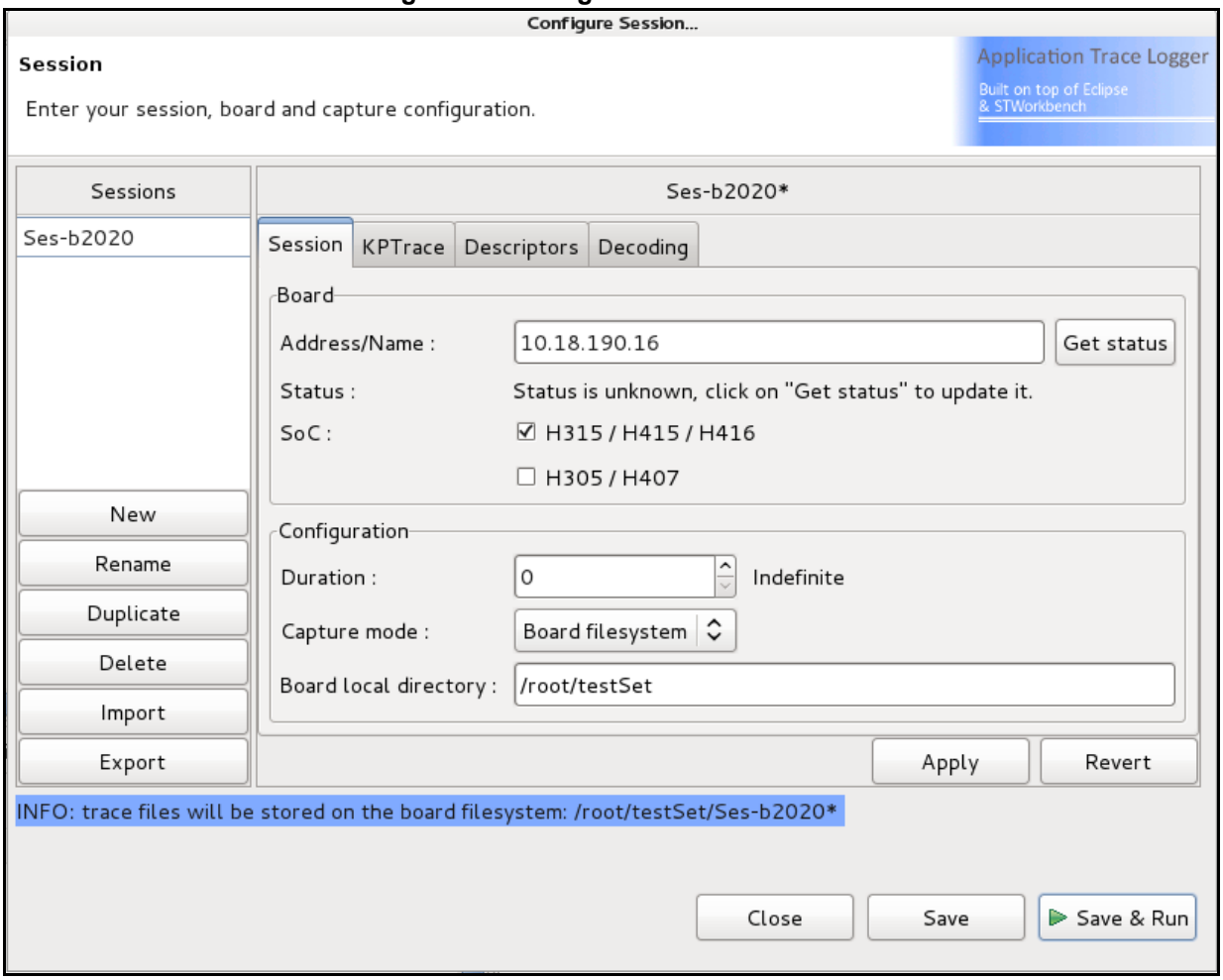
4.8 Generate trace in files

Trace can be generated into a file on the board file system by selecting the “Board filesystem” capture mode.

A board local directory to record the session can be specified too.

Note: The Session directory must be created before launching trace capture.

Figure 19. Configure Session window



There is no display while the session is running because the data is stored locally on the board file system.

When the session is stopped, ATL imports the trace files to the host workstation, displays their contents in the ATL window and saves them in a session directory. The directory is named according to the following naming convention:

`<Working Directory>/<SessionName>-<UniqueSession-ID>`

Note: The session directory name is visible in the top of the “Console” view

The session directory contains the following files:

- `<SessionName>-00.mtt`: Trace file for core 0
- `<SessionName>-01.mtt`: Trace file for core 1
- `<SessionName>.kpm`: Symbol file (image of `/proc/kallsyms`)
- `<SessionName>.meta`: Running processes at session startup

If a process instrumented with MTT or the KPTrace API has been launched during the trace session, it is saved in a file that is named according to the following naming convention:

`<prefixName>_<PIDno>.mtt`

If KPTrace was enabled, the KPTrace text files can be extracted from the MTT trace files by using the **mtt2kpt** utility. This utility is described in *Section 5.7: Convert trace file to KPTrace text file on page 37*.

5 Command line mode

This chapter describes how to use the ATL command line interface. In addition to this chapter, detailed information on the options and command line utilities can be found in the **mttatl** man page and the **mttcontrol** man page.

5.1 Launch ATL and configure daemon

When the board is booted, invoke ATL with the following command to configure and enable trace generation using MTT and capture over STM.

```
mttatl <Board-IP-address> -m probe
```

5.2 Launch ATL in listen-only mode

When there is no board IP address or name specified in the arguments, the ATL launches the trace capture chain and starts listening on the STM port.

The command is the following:

```
mttatl -m probe
```

*Note: Default mode when **mttatl** is used without any option is "`-m probe`".*

This decodes and displays on-the-fly any data sent over the trace port.

5.3 Launch ATL and enable KPTrace

When the board is booted, invoking ATL configures and enables KPTrace, trace generation using MTT and capture over STM.

```
mttatl <Board IP Address> -m probe –k
```

This command generates the same traces as described in *Section 4.5 on page 31*.

5.4 Launch ATL and save trace file

Input traces can be stored in a raw binary file and then re-opened off-line.

The arguments are:

- `-O stp`: enable saving traces in a raw format (including STM header) for post-processing
- `-o <filename>`: name of the file to store. Extension `_stp.bin` is added. The default value is `mtt`
- `-H <directory>`: name of the directory where to store the generated trace file. Default value is "." (current directory)

The following example generates `mtt.bin` in the current directory:

```
mttatl 10.18.190.56 -m probe -O stp
```

This example generates `mytrace.bin` file in the `/tmp` directory.

```
mttatl 10.18.190.56 -m probe -o mytrace -H /tmp -O stp
```

5.5 Reading a trace file

Use the following command to read a raw binary file (called `rawxtitraces_TM2.bin`) and save it in an output file named `outdata` in the directory `/tmp`. The command specifies the message descriptor file to apply when decoding the traces.

```
./mttatl -o outdata -H /tmp rawxtitraces_TM2.bin
```

5.6 Launch KPTrace

KPTrace can be launched from the board command line prompt as follows:

```
#> kptrace
```

By default, the trace data is stored in a file. When tracing begins, KPTrace displays a message on the console giving the path to the location where the trace data is stored (as shown in the following example output. where the location is `/root/kptrace`):

```
Tracing to file
Writing session data to folder /root/kptrace
Started tracing on localhost
Tracing.... Ctrl+C to stop
```

To differentiate between multiple trace sessions, it is possible to specify a custom trace session directory and a session ID, as follows:

```
mkdir -p /root/MySession
kptrace -T /root/MySession -i 34 -o hello
```

In this example, the trace session is created in `/root/MySession/hello-34`.

Note: The custom directory to be used must have been created before launching KPTrace. If not. KPTrace will exit with an error message.

Note: The same session name is re-used for subsequent KPTrace launches until a new name is specified. Use the following command to reset the session name and trace file name to the default:

```
kptrace -T root -i -1 -o kptrace
```

The session folder contains the following files:

- `hell-00.mtt`: KPTrace file for core 0
- `hell-01.mtt`: KPTrace file for core 1
- `hell.kpm` : Symbol file (a copy of `/proc/kallsyms`)
- `hell.meta`: Running processes at KPTrace startup

If processes instrumented with KPTrace or MTT API are executed during the session, they are stored in the same session as follows:

```
<prefixName>_<PIDno>.mtt
```

See the KPTrace man page for more information.

5.7 Convert trace file to KPTrace text file

Since KPTrace V4, KPTrace does not perform any text formatting of the trace output stream of the **kprobes** implementation. The trace events are output in a raw binary format. This has been done in order to reduce intrusiveness in the tracing.

The ATL provides a utility called **mtt2kpt** to reformat the raw binary files as text (ASCII) files. This utility is invoked with the following command line:

```
mtt2kpt --session=/root/ MySession /hello-34
```

This generates the following files

- `hello-00.kpt`: KPTrace text file for core 0
- `hello-01.kpt`: KPTrace text file for core 1

See **mtt2kpt** man page for more details.

Note: The `mtt2kpt` utility is available both on the board and host workstations.

5.8 Launch KPTrace and log trace to trace port

It is possible to launch KPTrace from the board command prompt and log traces on the trace port. To do this, complete the following steps.

1. On the host workstation, launch ATL without specifying a board IP address.

   This is described in *Section 4.3.8: Listen mode only on page 29* (if using the GUI) or *Section 5.2: Launch ATL in listen-only mode on page 35* (if using the command line).
2. On the board command prompt, launch KPTrace specifying that you want to log traces on the trace port, as follows:

```
kptrace -m probe
```

Appendix A Program example using MTT-API to generate traces

A.1 Example overview

In order to use MTT tracing in a program, include the MTT header file:

```
#include "mtt.h"
```

Declare a trace component handle:

```
mtt_comp_handle_t mymtthandle = NULL;
```

Figure 20 shows example code that uses MTT API calls.

Figure 20. Example of MTT API usage

```
   char              hello[]       = "Hello";
   char              world[]       = "World";
   char              helloworld[]  = "Hello World !";
   uint32_t          hwvector1[]   = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
…
   mtt_initialize( NULL );
   mtt_open(MTT_COMP_ID_ANY, "HelloWorld-Player", &mymtthandle);
   mtt_print(mymtthandle, MTT_LEVEL_INFO, "%s %s !", hello, world);
   mtt_trace(mymtthandle, MTT_LEVEL_INFO, MTT_TRACEITEM_STRING(strlen(helloworld)),
           helloworld, NULL);
   mtt_trace(mymtthandle, MTT_LEVEL_INFO, MTT_TRACEVECTOR_UINT32(10), hwvector1,
           "data0");
   mtt_print(mymtthandle, MTT_LEVEL_INFO, "bye.");
   mtt_close(mymtthandle);
   mtt_uninitialize();
…
```

A.2 Building the example

Build this example with the following options:

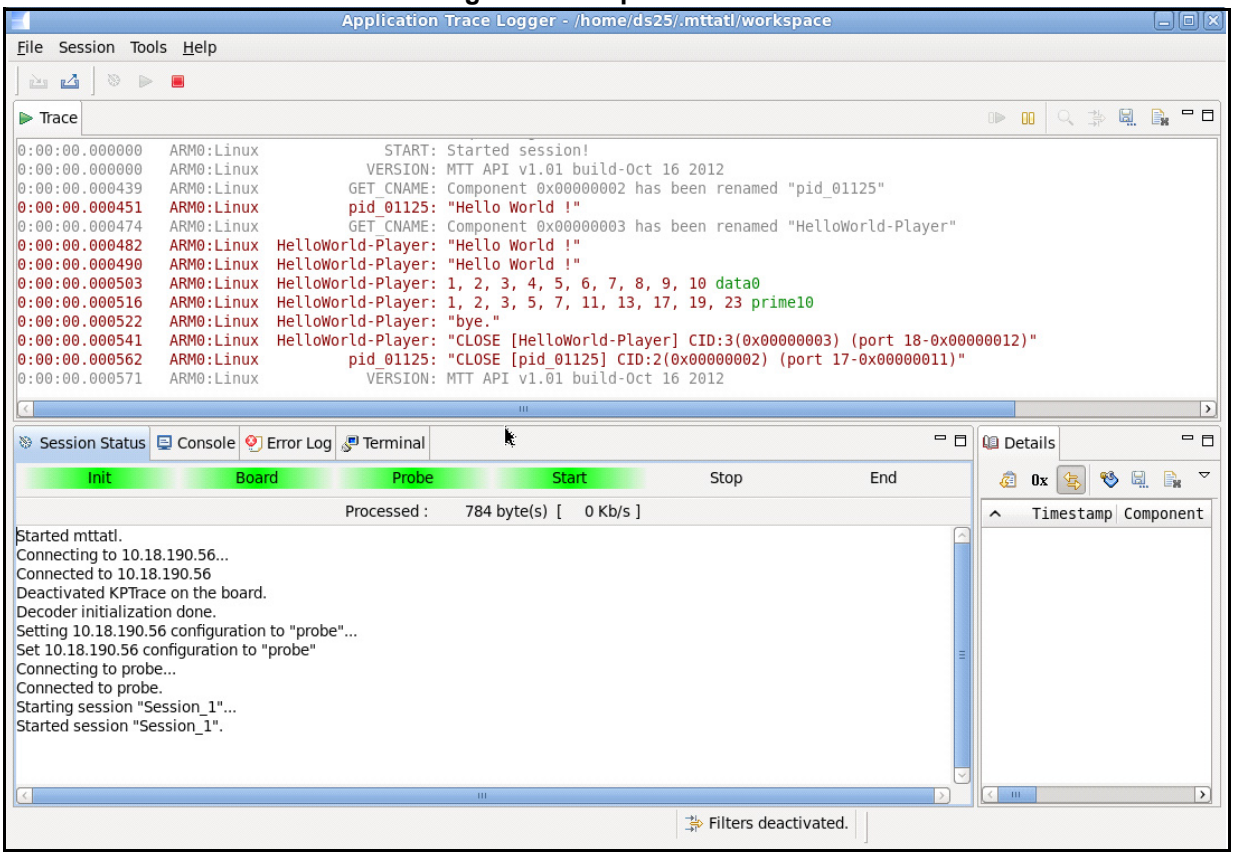
```
-I /opt/STM/STLinux-2.4/devkit/armv7/target/usr/include
-L /opt/STM/STLinux-2.4/devkit/armv7/target/usr/lib -lmtt
```

A.3 Output trace

Figure 21. Output trace

```
0:57:25.717003    ARM0: GET_CNAME: Component 0x00000003 has been renamed "pid_00842"
0:57:25.717023     ARM0: GET_CNAME: Component 0x00000004 has been renamed
"HelloWorld-Player"
0:57:25.717034    ARM0: HelloWorld-Player: "Hello World !"
0:57:25.717047    ARM0: HelloWorld-Player: "Hello World !"
0:57:25.717061    ARM0: HelloWorld-Player: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 data0
0:57:25.717074    ARM0: HelloWorld-Player: 1, 2, 3, 5, 7, 11, 13, 17, 19, 23 prime10
0:57:25.717079    ARM0: HelloWorld-Player: "bye."
0:57:25.717101     ARM0: HelloWorld-Player: "CLOSE [HelloWorld-Player]
CID:4(0x00000004) (port 19-0x00000013)"
0:57:25.717122     ARM0: pid_00842: "CLOSE [pid_00842] CID:3(0x00000003) (port
18-0x00000012)"
```

Figure 22. Output in the GUI



Appendix B Kernel module example using MTT-API to generate traces

B.1 Example overview

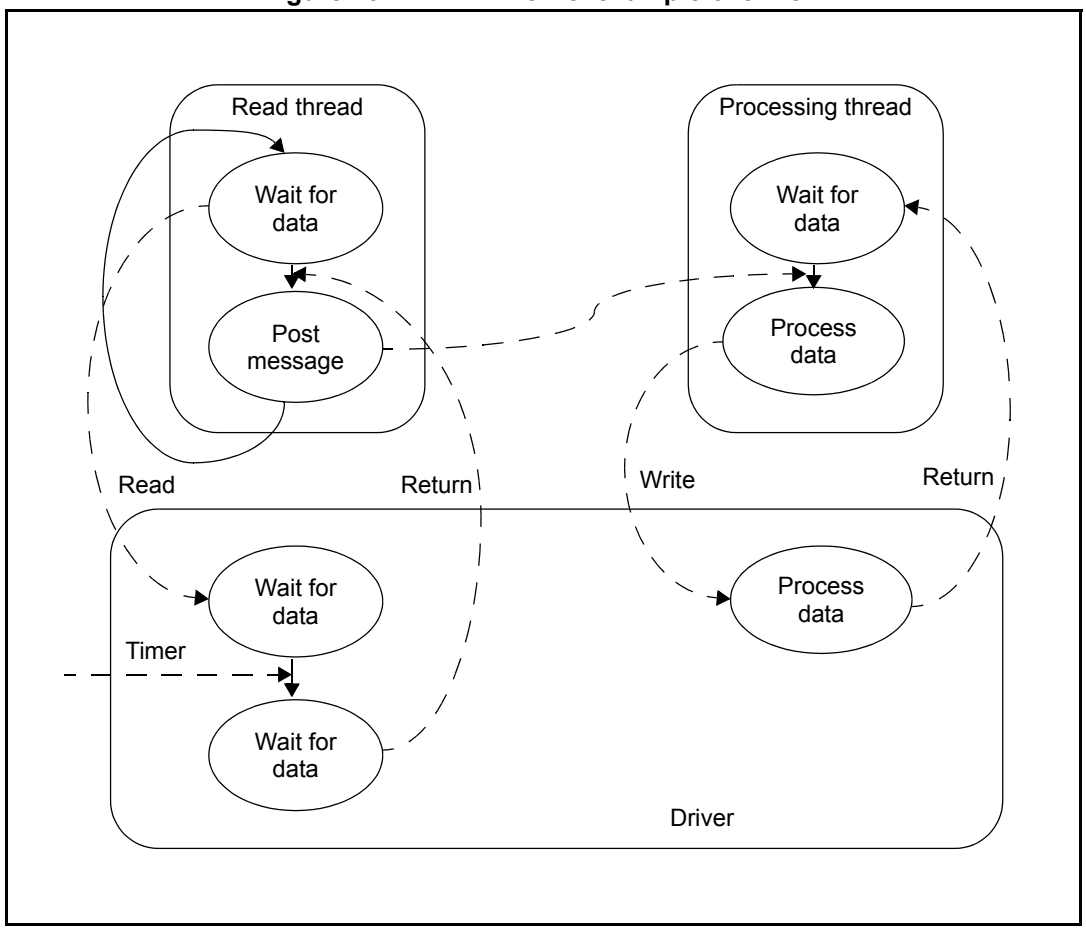
The example in the kernel `sample/mtt` folder simulates a simple block of data streaming with quality of service incidents caused by hardware contention. The purpose of this example is to demonstrate how trace can be used to check the dynamics of the application and to pin-point the source of data dropouts.

The application consists of a driver module and a user mode client. The client creates two threads:

- the reader thread, that performs a blocking read to the driver
- the processing thread (which is awoken whenever the reader completes a read successfully) to process the received data buffer and write the result to the driver

A hardware contention is simulated by creating an anomaly in the period of the kernel timer to trigger the completion of the blocking read. See *Figure 23*.

Figure 23. MTT-API kernel example overview



B.2 MTT API trace points

B.2.1 Checking the frequency and duration of the read operation (user space)

Figure 24 shows a fragment of example code from the user space application.

Figure 24. Example code

```
mtt_print(comp_handle, level, "Wait for data from driver...");

     rdsiz     = read (dev_fd, cur_buf->data, BUFF_SIZE * sizeof (int));

mtt_trace(comp_handle, MTT_LEVEL_USER0, MTT_TRACEITEM_UINT32, &rdsiz, "read_size");
```

The sample code in *Figure 24* shows that when the read operation completes, the size of the read is output to the trace stream by means of a call to `mtt_trace()`.

To help the tools to identify this monitored value, use the hint feature in the API to give it the label "`read_size`". Alternatively, the hint field could be left as `NULL` and the record could be located later using filter or search to isolate the location that issued this trace message, although this is less convenient.

B.2.2 Notifying a quality of service problem with a simple string

A quality of service problem can be signalled using `mtt_print()`.

```
mtt_print(comp_handle, MTT_LEVEL_WARNING, "Pipe underrun, data dropout\n");
```

B.2.3 Non-intrusive flagging of the read wait/wakeup in the kernel module

On the kernel side, instrument the code to flag the entry and exit of the wait list using named signals.

```
mtt_trace(mtt_ior_handle, MTT_LEVEL_DEBUG,0, NULL, "wait_event");
wait_event_interruptible_timeout(wq, got_irq, timeout);
mtt_trace(mtt_ior_handle, MTT_LEVEL_DEBUG,0, NULL, "resume");
```

B.3 Building and installing on the board

Building this example is completed using a typical kernel module compile directive:

```
make -C $KDIR M=`pwd` modules modules_install
```

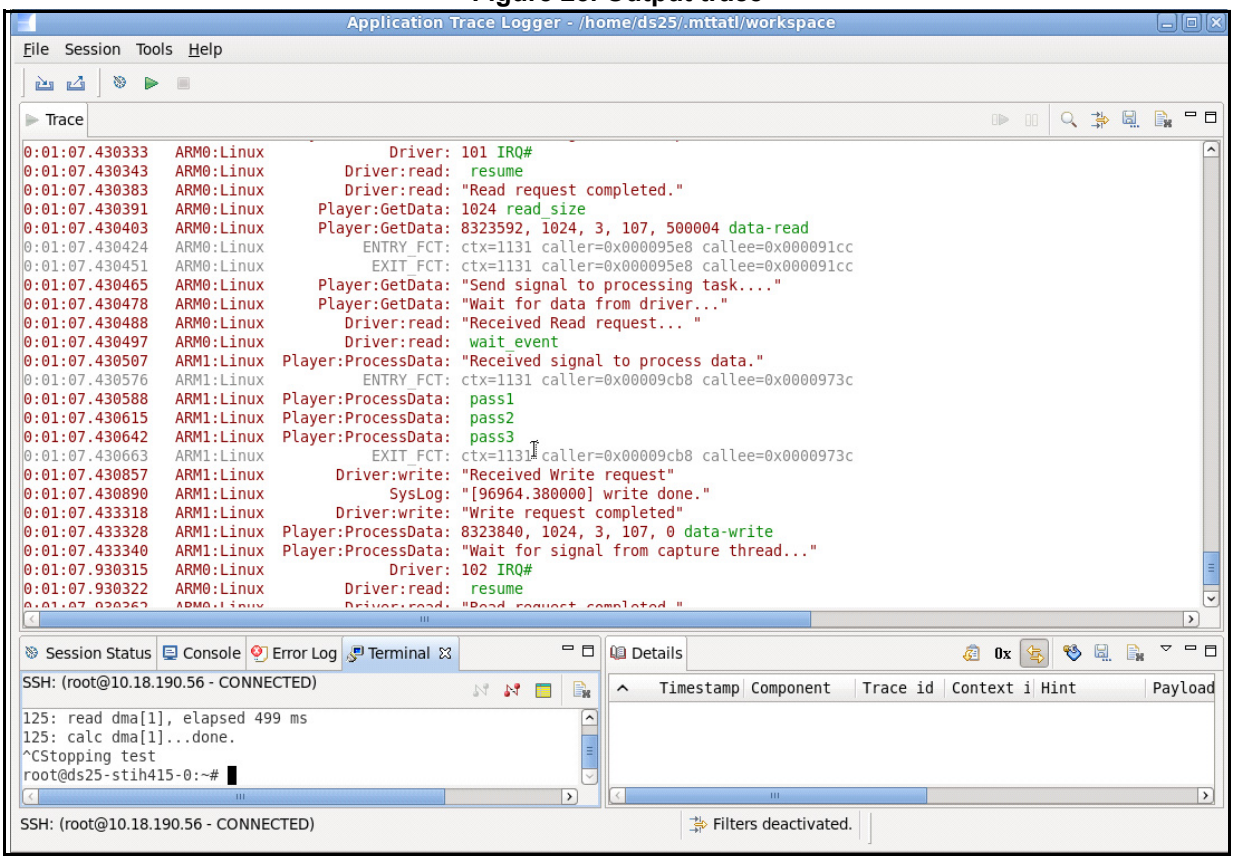
Where `$KDIR` is the path to the configured kernel sources.

B.4 Output trace

Figure 25 shows a typical output.

In this example there are signals labelled `data-read` and `data-write`. The output from these signals dump a sequence of comma separated integers that can subsequently be used for statistical analysis with a tool such as a spreadsheet.

Figure 25. Output trace



Appendix C mttd

By default, the **mttd** trace backend daemon is enabled for run-level 3 in Linux.

Use the configuration script in `/etc/init.d` to issue the regular service management commands:

```
/etc/init.d/mttd {start|stop|reload|restart|force-reload|reload-or-restart}
```

Revision history

Table 5. Document revision history

Date	Revision	Changes
10-Jan-2013	1	Initial release.
22-May-2013	2	Updated for ATL rev 1.2. Updates to *Introduction on page 1*. Updates to *Preface on page 6* Update to *Figure 2 on page 10*. Updates to *Chapter 2: Product installation on page 12* to reflect that fact that the ATL rev 1.2 is distributed as three packages instead of two. Added *Section 2.1: Dependencies on page 12*. Update to *Table 4: Location of host tools on page 16*. Update to *Section 2.4: Configure the kernel on page 16*. Update to *Section 4.2: ATL GUI overview on page 22* to add more information about the GUI perspective. Update to *Section 4.4: Tracing using printk on page 29*. Update to *Section 4.5: KPTrace on page 31* to provide more information about trace files. Updates to *Section 4.6: User traces on page 32*. Added *Section 4.8: Generate trace in files on page 32*. Renamed *Chapter 5: Command line mode on page 35*. Added *Section 5.6: Launch KPTrace on page 36*. Added *Section 5.7: Convert trace file to KPTrace text file on page 37*. Added *Section 5.8: Launch KPTrace and log trace to trace port on page 37*. Changed title of *Appendix A: Program example using MTT-API to generate traces on page 38*. Changed title of *Appendix B: Kernel module example using MTT-API to generate traces on page 40*.
30-Oct-2013	3	Update to *Chapter 2: Product installation on page 12*. Added note following *Figure 6 on page 19*. Update to *Figure 10 on page 23*. Update to *Section 4.3.1: Configure and launch a trace session on page 24*. Update to *Figure 11 on page 24*. Update to *Figure 16 on page 28*. Update to *Figure 18 on page 31*. Update to *Figure 19 on page 33*.

Please Read Carefully:

Information in this document is provided solely in connection with ST products. STMicroelectronics NV and its subsidiaries (“ST”) reserve the right to make changes, corrections, modifications or improvements, to this document, and the products and services described herein at any time, without notice.

All ST products are sold pursuant to ST’s terms and conditions of sale.

Purchasers are solely responsible for the choice, selection and use of the ST products and services described herein, and ST assumes no liability whatsoever relating to the choice, selection or use of the ST products and services described herein.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted under this document. If any part of this document refers to any third party products or services it shall not be deemed a license grant by ST for the use of such third party products or services, or any intellectual property contained therein or considered as a warranty covering the use in any manner whatsoever of such third party products or services or any intellectual property contained therein.

UNLESS OTHERWISE SET FORTH IN ST’S TERMS AND CONDITIONS OF SALE ST DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY WITH RESPECT TO THE USE AND/OR SALE OF ST PRODUCTS INCLUDING WITHOUT LIMITATION IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE (AND THEIR EQUIVALENTS UNDER THE LAWS OF ANY JURISDICTION), OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

ST PRODUCTS ARE NOT DESIGNED OR AUTHORIZED FOR USE IN: (A) SAFETY CRITICAL APPLICATIONS SUCH AS LIFE SUPPORTING, ACTIVE IMPLANTED DEVICES OR SYSTEMS WITH PRODUCT FUNCTIONAL SAFETY REQUIREMENTS; (B) AERONAUTIC APPLICATIONS; (C) AUTOMOTIVE APPLICATIONS OR ENVIRONMENTS, AND/OR (D) AEROSPACE APPLICATIONS OR ENVIRONMENTS. WHERE ST PRODUCTS ARE NOT DESIGNED FOR SUCH USE, THE PURCHASER SHALL USE PRODUCTS AT PURCHASER’S SOLE RISK, EVEN IF ST HAS BEEN INFORMED IN WRITING OF SUCH USAGE, UNLESS A PRODUCT IS EXPRESSLY DESIGNATED BY ST AS BEING INTENDED FOR “AUTOMOTIVE, AUTOMOTIVE SAFETY OR MEDICAL” INDUSTRY DOMAINS ACCORDING TO ST PRODUCT DESIGN SPECIFICATIONS. PRODUCTS FORMALLY ESCC, QML OR JAN QUALIFIED ARE DEEMED SUITABLE FOR USE IN AEROSPACE BY THE CORRESPONDING GOVERNMENTAL AGENCY.

Resale of ST products with provisions different from the statements and/or technical features set forth in this document shall immediately void any warranty granted by ST for the ST product or service described herein and shall not create or extend in any manner whatsoever, any liability of ST.

ST and the ST logo are trademarks or registered trademarks of ST in various countries.
Information in this document supersedes and replaces all information previously supplied.
The ST logo is a registered trademark of STMicroelectronics. All other names are the property of their respective owners.

STMicroelectronics group of companies

Australia - Belgium - Brazil - Canada - China - Czech Republic - Finland - France - Germany - Hong Kong - India - Israel - Italy - Japan - Malaysia - Malta - Morocco - Philippines - Singapore - Spain - Sweden - Switzerland - United Kingdom - United States of America

www.st.com