STMicroelectronics

TRACE32 for Nomadik

User manual

8063903 Rev. A

January 2008

BLANK

User manual

TRACE32 for Nomadik

Nomadik is a registered trademark of STMicroelectronics

Introduction

This user guide is a quick reference to help the installation and use of TRACE32®, the premier solution for debugging multi-cores on Nomadik.

Contents

Preface

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

License and guarantee information

Appendix A: License and guarantee on page 54 shows the licence and guarantee for TRACE32.

MMDSP documentation suite

The Nomadik documentation suite comprises the following volumes:

Nomadik Toolset getting started

ADCS 8087207. This manual describes the principles of the Nomadik Toolset and provides some basic worked examples.

Nomadik MMDSP+ Toolset

ADCS 8086787. This manual describes the Nomadik MMDSP+ Tools for compiling, debugging and simulating code on Nomadik cores.

NMF programming model

ADCS 8071313. This manual describes the Nomadik multimedia framework (NMF) programming model that is used to define the Nomadik component-based projects.

Trace32 for Nomadik

ADCS 8063903. This manual describes how to use TRACE32 to configure and connect to a target board through a PowerTrace or PowerDebug box.

GNU documentation

In addition, there are several GNU documents supplied with the Nomadik Toolset that are published by the Free Software Foundation.

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) unless otherwise specified.

- Terminal strings of the language, that is those not built up by rules of the language, are printed in teletype font. For example, `void`.
- Nonterminal 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 nonterminal string of the language starts with a nonitalicized part, it is equivalent to the same nonterminal string without that nonitalicized 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 ‘`}`’).

Acknowledgements

Multi-ICE® and RealView® are registered trademarks of ARM limited in the EU and other countries.

Windows® is a registered trademarks of Microsoft Corporation in the United States and/or other countries.

Linux® is a registered trademark of Linus Torvalds.

TRACE32® is a registered trademark of Lauterbach Datentechnik GmbH.

Nomadik is a registered trademark of STMicroelectronics.

1 Installation

Perform the installation of TRACE32 in the following main steps.

1. Software installation, see *Section 1.1: Software installation*.
2. Hardware installation, see *Section 1.2: Hardware installation on page 11*.

1.1 Software installation

TRACE32 is a Windows program, to install do the following.

1. Insert the CD into your CD drive. The installation starts automatically. If it does not:

   From the Windows Start menu, select **Run** and type `D:\SETUP.BAT`, where `D` is the drive letter of your CD drive. A welcome page appears.
2. From the destination page, select the installation directory.
3. From the setup type page, select the type of installation:

   If a version of TRACE32 is already installed for the ARM or SxA, download the latest software update. Contact the TRACE32 purchase support for information on where to locate the latest update.
4. From the product type page, select **ICD**.
5. From the ICD interface page, select the interface type. We recommend that you use a USB Interface.
6. From the license selection page, select the licence. If you are using a new TRACE32 hardware device, you do not require a licence key, see *Appendix A: License and guarantee on page 54*.
7. From the OS page, select the OS system.
8. From the CPU page, select each of the target CPUs from the list: ICD ARM, ICD and MMDSP.
9. TRACE32 installs to your PC. A message box opens stating that “The USD driver (T32USB.sys) is on the CD.”, see *Section 1.2: Hardware installation*.
10. From the environment variable T32ID page, type the name of the T32ID environmental variable.
11. From the environment variable T32TMP page, select the path to the T32ID environmental variable. This is the directory for the TRACE32 temporary files. Ensure that you have write permissions for this directory.
12. From the screen configuration page, select the TRACE32 screen configuration (font sizes).
13. From the next screen configuration page, select whether TRACE32 has client windows in a single window (MDI) or multiple windows spread around the whole screen (MWI).
14. From the prepare for integration page, select which products TRACE32 is to integrate with.
15. From the folder selection page, select the Windows start folder for TRACE32.
16. From the program group type, select whether TRACE32 is for all users (common) or for a single user (personal).
17. From the registration page, select the registration method.

This completes the installation.

1.1.1 Installing the run environments

To simplify the Nomadik debug configuration and Trace32 run environment configuration, some configuration script files are delivered in the following folder.

```
<NDKTOOLS_ROOT>\configuration\debugger\trace32
```

There are two ways to use the configuration files:

- Copy the content of the folder into the same location as the TRACE32 installation.
- If you need to keep these files somewhere else, modify the Windows environment variable PATH so that Trace32 debugger programs are reachable. For example:

  ```
  set PATH=%PATH%;<Trace32_location>
  ```

Script files

The folder contains a set of scripts files (cob15_xxx_xxx.cmm, stn8820xx_xxx.cmm) that gives the STn8815 and STn8820 NDK boards configurations.

The following are the most common methods of booting a NDK15 or a NDK20 board.

- Running a configuration script file by an ARM debugger such as Trace32, Realview Debugger, and so on.

  The board configuration script files (cob15_xxx_xxx.cmm, stn8820xx_xxx.cmm) are run using this method. The Trace32-ARM boot file can invoke the configuration files so that when Trace32-ARM starts, the board is configured.
- Running (often by using an ARM debugger) an executable that configures the board.

Note: The first two methods can be used with Trace32-ARM.

- The board boots itself at power-on or at reset by the boot code loaded in a flash memory.

.bat files

The folder contains the `startt32.bat` that starts TRACE32.

Configuration files

The folder contains a set of configuration files that initialize and configure TRACE32, see *Section 2.1.3: Trace32 Configuration files on page 17*.

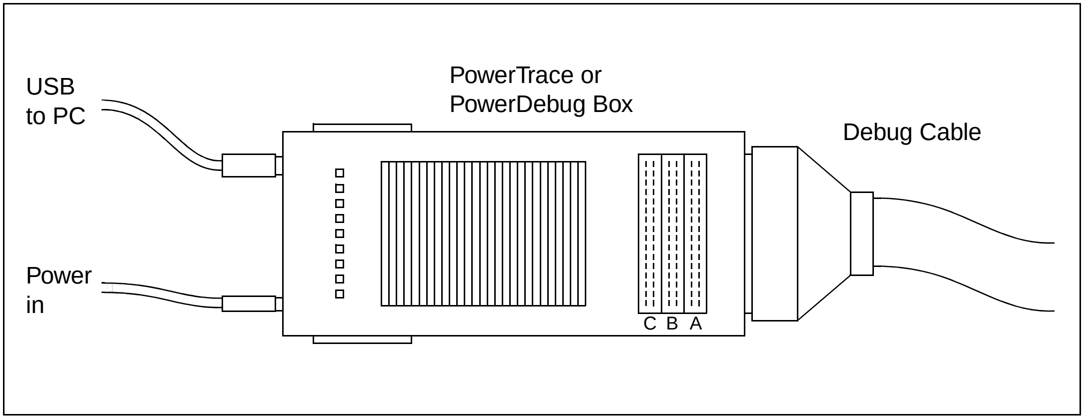
Utility scripts

The folder contains a set of utilities scripts that help with the Nomadik debugging configurations. For example, when configuring cross debugging (also known as cross triggering), changing debug mode and so on.

1.2 Hardware installation

This section describes the installation of the TRACE32 devices.

Figure 1. PowerTrace or PowerDebug connections



1.2.1 Installing the PowerTrace or PowerDebug driver

On first use of the PowerTrace or PowerDebug, do the following.

1. Ensure that the installation CD is in the CD drive.
2. Connect the device through the USB cable to the PC.
3. Connect the device to the power supply and set the device power adaptor switch to On. The PC detects the device automatically and opens the **Found new hardware** dialog box.
4. Install the device driver.

On subsequent use, there is no need to install the device driver.

1.2.2 Configuring the target board

Appendix B: Target board configuration on page 55 shows how to set the switches and jumpers on most target boards. Please refer to the *NDK-15 Core Board user manual* for more information.

For the COB-10(B) and NDK-15 boards, select either “chained” or “unchained” debugging mode (also known as “separated” mode).

In unchained mode, ARM and MMDSP debugging signals are routed through different paths to different connectors on the board. A typical configuration has the debugger connect to the NEXUS Mictor (P3) for SxA debugging and to the MAIN JTAG connector (J6) for ARM debugging.

In chained mode, ARM and MMDSP debugging units (JTAG TAPs) are chained through the same path to the same connector on the board. The debugger can connect to the MAIN JTAG connector (J6) for both ARM and SxA debugging.

1.2.3 Connecting the Trace32 Devices to the target board

Connect the debug devices correctly to the target boards.

For a minimal installation, you require a single PowerDebug box with a debug cable. *Appendix C: TRACE32 devices on page 58* shows the devices to be used in the different configuration cases.

To connect the target board, do the following.

1. Identify the Nexus connector and JTAG interface on the target board, see *Appendix D: Connections on page 59*.
2. Identify the debug mode (chained or unchained) configured by the *Section 1.2.2: Configuring the target board*.
3. Identify the connection name in the Table5 in *Appendix C: TRACE32 devices on page 58*.
4. Make the connection according to the *Appendix D: Connections on page 59*.

Caution: Always power on the Trace32 device first, then the target board. Always power off the target board first, then Trace32 device.

2 Using TRACE32

This chapter describes briefly how to work with TRACE32, and describes only the most commonly used functionality. For information on advanced operation, please consult the TRACE32 help menu.

2.1 Starting TRACE32

To launch up to four Trace32 debuggers simultaneously in chained or unchained debug mode, run the startt32.bat file, see *1.1.1: Installing the run environments on page 10*. Alternately, use the t32start.exe tool to launch the Trace32 debuggers.

To run startt32.bat, ensure that the following are correct before starting.

- Use the correct `startt32` parameters, see *Section 2.1.1: Startt32.bat parameters*.
- Ensure that the configuration file is correct, see *Section 2.1.3: Trace32 Configuration files*.
- Ensure the host connection is correct, see *2.1.4: Configuring the USB or Ethernet host connection on page 19*.

2.1.1 Startt32.bat parameters

start32 has three mandatory parameters and five optional parameters.

```
startt32 [-c] debug_mode cores Nomadik_version [MMDSP cores names]
[2-ports] [config-file] [-b boot-file]
```

To open the on-line help, type one of the following commands:

- startt32
- startt32 /?
- startt32 --help
- startt32 -h

interactive mode

The option “-c” helps generate a startt32 command with the correct parameters. For example:

```
startt32 -c
```

Other parameters

`<debug_mode>`

Debug mode, see *Section 1.2.2: Configuring the target board*.

Use one of the following values.

`chained`: ARM and SxA(MMDSP+) JTAG interfaces are chained.

`unchained`: ARM and SxA(MMDSP+) JTAG interfaces are unchained. (It needs the PODBUS if the debug cores are ARM+MMDSPs).

`<cores>`

For ARM or MMDSP.

Use one of the following values.

`arm:` Starts one Trace32 GUI for ARM debugging.

`arm-mmdsp`: Starts two Trace32 GUI at a time for ARM and one MMDSP debuggers.

`arm-2mmdsp`: Starts three Trace32 GUIs for ARM and two MMDSPs debuggers.

`arm-3mmdsp`: Starts four Trace32 GUIs for ARM and three MMDSPs debuggers.

`mmdsp`: Starts one Trace32 GUI for one MMDSP debugging

`2mmdsp`: Starts two Trace32 GUIs for two MMDSPs debuggers.

`3mmdsp`: Starts three Trace32 GUIs for three MMDSPs debuggers.

`<Nomadik_version>`

Use one of the following values.

`8810`: Configures STn8810 with `init_ndk10.cmm` when Trace32-ARM starts.

`8815`: Configures STn8815 with `init_ndk15.cmm` when Trace32-ARM starts.

`8820`: Configures STn8820 with `init_ndk20.cmm` when Trace32-ARM starts.

none: Do not configure Nomadik at Trace32-ARM start.

[MMDSP cores names]

SAA, SVA or SIA. It's a optional parameter.

Use one of the following values.

`a`: the MMDSP to debug is SAA

`v`: the MMDSP to debug is SVA

`i`: the MMDSP to debug is SIA

`av`: the MMDSPs to debug are SAA and SVA

`ai`: the MMDSPs to debug are SAA and SIA

`vi`: the MMDSPs to debug are SVA and SIA

`avi`: the MMDSPs to debug are SAA, SVA, and SIA

[2-ports]

This is an optional parameter. This parameter is rarely used and not recommended.

It is used in the chained debug mode and the board is configured to output ARM and MMDSP debug signals onto both MAIN JTAG port and NEXUS mictor.

This mode of debug configuration is useful if the Nexus trace is required and the debug mode can only be chained mode.

This option requires supplementary switches configuration on the NDK boards, see *NDK15 boards on page 55*.

`[config-file]`

Displays the configuration file appropriate to the Trace32 settings. The command prompt displays the name of the configuration file but does not start TRACE32, see *Section 2.1.3: Trace32 Configuration files*.

[-b boot-file]

Specifies the `.cmm` file that each Trace32 debugger runs at start up. Without this option, the Trace32 uses the `ndk_t32.cmm` file.

2.1.2 startt32.bat usage examples

This section gives several examples of the `start32.bat` usage to start a Trace32-ARM and a Trace32-MMDSP for debugging ARM and SAA in STn8815 in chained debug mode.

Example one

To open Trace32 in interactive mode, type `startt32 -c`. Trace32 displays the following options.

```
<startt32 -c

The available debug modes are:
(c) chained:    ARM and SxA(MMDSP+) JTAG interfaces are chained.
(u) unchained:  ARM and SxA(MMDSP+) JTAG interfaces are unchained. (It needs the
 PODBUS if the debug cores are ARM+MMDSPs)
Please choose a debug mode:c

The available cores to debug are:
(a) arm         start 1 Trace32 GUI for ARM debugging
(am) arm-mmdsp  start 2 Trace32 GUI at a time for ARM and 1 MMDSP debuggings
(amm) arm-2mmdsp        start 3 Trace32 GUIs for ARM and 2 MMDSPs debuggings
```

```
(ammm) arm-3mmdsp       start 4 Trace32 GUIs for ARM and 3 MMDSPs debuggings
(m) mmdsp       start 1 Trace32 GUI for 1 MMDSP debugging
(mm) 2mmdsp     start 2 Trace32 GUIs for 2 MMDSPs debuggings
(mmm) 3mmdsp    start 3 Trace32 GUIs for 3 MMDSPs debuggings
Please chose the core(s) to debug:am

The available Nomadik versions are:
8810            configure STn8810 with init_ndk10.cmm when Trace32-ARM starts
8815            configure STn8815 with init_ndk15.cmm when Trace32-ARM starts
8820            configure STn8820 with init_ndk20.cmm when Trace32-ARM starts
(n)none         do not configure Nomadik at Trace32-ARM start
Please choose a version:8815

The available SxA(MMDSP+) are:
a               the MMDSP to debug is SAA
v               the MMDSP to debug is SVA
i               the MMDSP to debug is SIA
av              the MMDSPs to debug are SAA and SVA
ai              the MMDSPs to debug are SAA and SIA
vi              the MMDSPs to debug are SVA and SIA
avi             the MMDSPs to debug are SAA, SVA, and SIA
Please choose SxA:a

Each Trace32 GUI is started according to its configuration file. You may need to
 edit it.
Do you want to get the configuration file(s) name(s)?[Y/N](N)

Each Trace32 GUI runs a .cmm file at boot.
Do you want to specify the boot file (ndk_t32.cmm by default)?[Y/N](N)

Running the command: startt32 chained arm-mmdsp 8815 a

start t32marm.exe -c ndk_mc_config_arm.t32, ndk_t32.cmm chained 8815 a C:\T32

Starting the second Trace32, please wait ...
start t32mmdsp.exe -c ndk_mc_config_mmdsp.t32, ndk_t32.cmm chained 8815 a C:\T32
```

Example two

Open Trace32 by specifying all the startt32 parameters.

```
>startt32 chained arm-mmdsp 8815 a
start t32marm.exe -c ndk_mc_config_arm.t32, ndk_t32.cmm chained 8815 a C:\T32

Starting the second Trace32, please wait ...
start t32mmdsp.exe -c ndk_mc_config_mmdsp.t32, ndk_t32.cmm chained 8815 a C:\T32
```

Example three

Find the available configuration files names of example one.

To find the configuration names in interactive mode, type *startt32 -c* or by the complete command:

```
>startt32 chained arm-mmdsp 8815 a config-file
Config file for Trace32-ARM: ndk_mc_config_arm.t32
Config file for Trace32-MMDSP: ndk_mc_config_mmdsp.t32
```

Note: *Trace32 debugger does not start.*

Example four

Use `my_t32_boot_file.cmm` as the Trace32 boot file instead of ndk_t32.cmm in the example one.

```
startt32 chained arm-mmdsp 8815 a -b my_t32_boot_file.cmm
start t32marm.exe -c ndk_mc_config_arm.t32, my_t32_boot_file.cmm chained 8815 a
C:\T32

Starting the second Trace32, please wait ...
start t32mmdsp.exe -c ndk_mc_config_mmdsp.t32, my_t32_boot_file.cmm chained 8815
 a C:\T32
```

2.1.3 Trace32 Configuration files

Use the appropriate configuration files to configure Trace32. Depending on the debug mode and connections, use:

- PODBUS for the debug connection
- Environment variables referenced by the debugger
- USB or Ethernet for the connection to the host
- Printer settings
- Debugger's GUI setting

Use the `config-file` parameter to find the configuration file to the target, see *Section 2.1.1: Startt32.bat parameters*.

It is possible that the configuration file needs to be modified to correspond with the host environment. Ensure that the following lines are correct:

- `SYS=` the location of TRACE32
- `HELP=` the location of TRACE32 help
- `TMP=` the location of the temporary files folder

Table 1 lists the configuration files.

Table 1. Configuration files

Configuration file	Description
`ndk_config_arm.t32`	the configuration file of Trace32-ARM in the single-core debug mode (without Trace32-MMDSP launched)
`ndk_config_mmdsp.t32`	the configuration file of Trace32-MMDSP in the single-core mode (without Trace32-ARM launched).
`ndk_config_mmdsp_core2.t32`	the configuration file of second instance of Trace32-MMDSP in the single-core mode (without Trace32-ARM launched).
`ndk_config_mmdsp_core3.t32`	the configuration file of third instance of Trace32-MMDSP in the single-core mode (without Trace32-ARM launched).
`ndk_mc_config_arm.t32`	in the chained debug mode, the configuration file of Trace32-ARM in multi-cores debug mode (at least one Trace32-MMDSP will be launched).

Table 1. Configuration files

Configuration file	Description
ndk_mc_config_mmdsp.t32	In the chained debug mode, the configuration file of Trace32-MMDSP in multi-cores debug mode (the Trace32-ARM has been launched before hand).
ndk_mc_config_mmdsp_core2.t32	In the chained debug mode, the configuration file of the second instance of Trace32-MMDSP in multi-cores debug mode (the Trace32-ARM and a Trace32-MMDSP have been launched).
ndk_mc_config_mmdsp_core3.t32	in the chained debug mode, the configuration file of the third instance of Trace32-MMDSP in multi-cores debug mode (the Trace32-ARM and two Trace32-MMDSP have been launched).
ndk_mc_podbus_config_arm.t32	In the unchained debug mode, the configuration file of the Trace32-ARM in multi-cores debug mode (at least one Trace32-MMDSP will be launched).
ndk_mc_podbus_config_mmdsp.t32	In the unchained debug mode, the configuration file of Trace32-MMDSP in multi-cores debug mode (the Trace32-ARM has been launched before hand).
ndk_mc_podbus_config_mmdsp_core2.t32	In the unchained debug mode, the configuration file of the second instance of Trace32-MMDSP in multi-cores debug mode (the Trace32-ARM and a Trace32-MMDSP have been launched).
ndk_mc_podbus_config_mmdsp_core3.t32	in the unchained debug mode, the configuration file of the third instance of Trace32-MMDSP in multi-cores debug mode (the Trace32-ARM and two Trace32-MMDSP have been launched).

2.1.4 Configuring the USB or Ethernet host connection

There are two ways to connect a PowerDebug or PowerTrace box to the host machine.

USB cable

By default, the configuration files use the USB connection, see *Section 2.1.3: Trace32 Configuration files*.

Ethernet

To configure the Ethernet connection, do the following:

1. Initially, use the default USB connection to start a Trace32-ARM or a Trace32-MMDSP.
2. Either, from the **Misc** menu select **Ethernet config...**, or type **ifconfig** in the command line and click **ok**. The **ifconfig** dialog box opens, see*Figure 2*.
3. Select the DHCP option and type a name for the connection.

   Because the “_” and upper case characters are sometimes not recognized, we recommend using lowercase characters and “-” instead of “_”.
4. Modify the appropriate configuration file (use **config-file** parameter to find the configuration file name, see *Section 2.1.1: Startt32.bat parameters*):

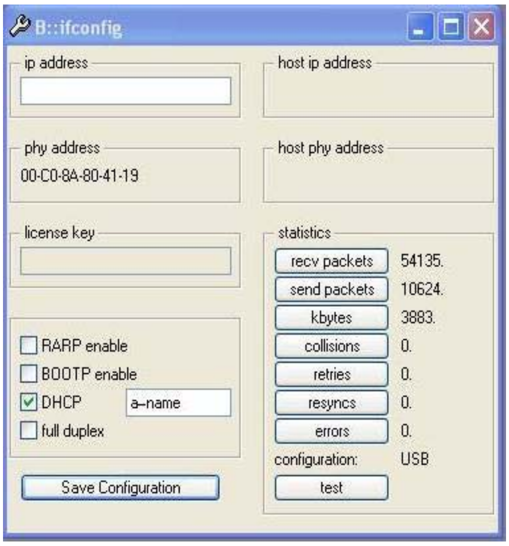
   Uncomment the ETH section.

   Comment the USB section.

   Change the connection name in the **NODE** line, like

   ```
   ==== ETH ======
   PBI=
   NET
   NODE=a-name.gnb.st.com
   ```
5. Click the **Save Configuration** button and close the dialog box.

Figure 2. ifconfig dialog box



B::ifconfig

ip address:

host ip address:

phy address: 00-C0-8A-80-41-19

host phy address:

license key:

□ RARP enable

□ BOOTP enable

☑ DHCP a–name

□ full duplex

statistics

recv packets 54135.

send packets 10624.

kbytes 3883.

collisions 0.

retries 0.

resyncs 0.

errors 0.

configuration: USB

2.1.5 TRACE32 start up

On startup, each TRACE32 debugger goes through the following tasks.

1. **startt32** calls`t32marm.exe` or `t32mmdsp.exe`, or both (as appropriate to the “cores” parameter of the **startt32** command).
2. `t32arm.exe` or `t32mmdsp.exe`, or both start TRACE32 and the self tests.
3. When the self test procedure completes, `t32arm.exe` or `t32mmdsp.exe`, or both execute the boot script given by the “-b” parameter or `ndk_t32.cmm` by default.
4. `ndk_t32.cmm` script performs the tasks such as:
   - configuring the menus, speed buttons, GUI settings and so on
   - adding the current folder (where **startt32** is called) into the Trace32 internal PATH environment variable so that the command scripts located in the current folder can be called anywhere
   - configuring the JTAG chain according to the “Nomadik_version” and “debug_mode” parameters
   - for SxA, set the target CPU to be the selected one
   - for SxA, plug in RTOS awareness feature
   - for ARM, call init_ndk10.cmm or init_ndk15.cmm or init_ndk20.cmm to configure selected Nomadik according to the “Nomadik_version” parameter
   - activating the autostore and history feature

To enable customized procedures or configurations that need to be performed at Trace32 start up, the boot script possibly needs to be edited.

2.1.6 Nomadik configuration files

As mentioned in the *Section 2.1.1: Startt32.bat parameters* by the option “**-b**”, the boot file ndk_t32.cmm file is run by default whenTrace32 debuggers start.

If the debugger is Trace32-ARM, according to the chosen Nomadik version, the init_8810.cmm, init_8815.cmm or init_8820.cmm is called by the ndk_t32.cmm boot file.

These scripts configure Nomadik and NDK board for debug use. You probably need to modify these scripts to adapt to your project. Without modifications, the scripts do by default:

- call target board init scripts, like
  ```
  do cob15_b06_100mhz.cmm
  ```
- If the “Nomadik_version” parameter is “8820”, power on the selected SxA core according to the “cores” parameter in the startt32 command
- configure the AHB bases/tops addresses of the external memory area allocated for the selected SxA, if the <cores> parameter on the startt32 command line contains mmdsp.
- configure GPIOs for SxA debug in unchained mode, if the mode is given by the startt32 command and the <cores> parameter contains mmdsp.
- enable Nexus trace if the “unchained” mode is given by the startt32 command and the the <cores> parameter contains mmdsp.

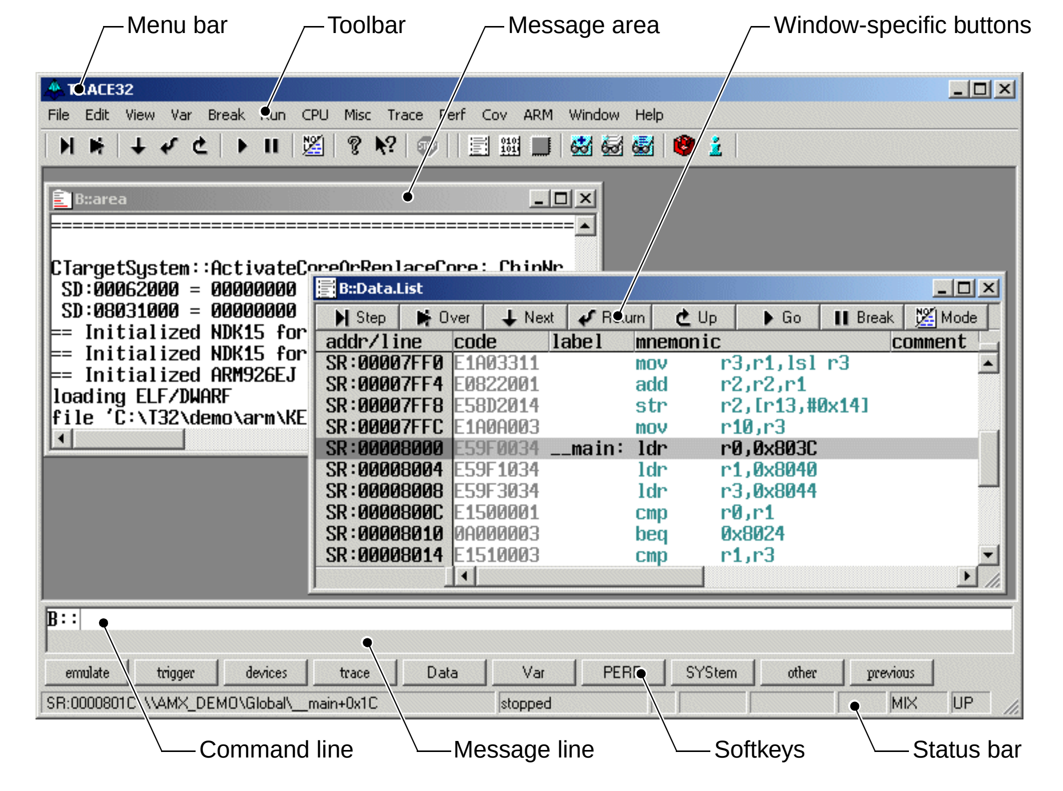
2.1.7 Stopping TRACE32

At power off, always turn the board off first before the TRACE32 devices. Do not disconnect any devices until power has been switched off on all devices.

2.2 The graphical user interface (GUI)

Figure 3 shows the layout of the TRACE32 application window.

Figure 3. TRACE32 GUI



```
==================================================
CTargetSystem::ActivateCoreOrReplaceCore: ChipNr
 SD:00062000 = 00000000
 SD:08031000 = 00000000
== Initialized NDK15 for
== Initialized NDK15 for
== Initialized ARM926EJ
loading ELF/DWARF
file 'C:\T32\demo\arm\KE
```

```
addr/line    code      label    mnemonic             comment
SR:00007FF0  E1A03311           mov       r3,r1,lsl r3
SR:00007FF4  E0822001           add       r2,r2,r1
SR:00007FF8  E58D2014           str       r2,[r13,#0x14]
SR:00007FFC  E1A0A003           mov       r10,r3
SR:00008000  E59F0034  __main:  ldr       r0,0x803C
SR:00008004  E59F1034           ldr       r1,0x8040
SR:00008008  E59F3034           ldr       r3,0x8044
SR:0000800C  E1500001           cmp       r0,r1
SR:00008010  0A000003           beq       0x8024
SR:00008014  E1510003           cmp       r1,r3
```

```
B::
```

```
SR:0000801C \\AMX_DEMO\Global\__main+0x1C   stopped   MIX   UP
```

Menu bar

Gives access to all the TRACE32 functionality. If you use **startt32** to connect to a target, an additional menu item with the CPU name appears. For example, 8815a.

Toolbar

Gives access to TRACE32 common tasks.

Message area

Displays relevant messages during debugging.

Window-specific buttons

Each window can have window-specific buttons.

Command line

Either type commands directly into the command line or use the softkeys to construct commands.

Message line

Displays any errors or other messages.

Softkeys

Softkeys enable you to construct a string of commands in the command line. The softkeys are context-sensitive, so that as you add commands the keys change to other appropriate commands.

Because there are often more softkeys than can fit in the screen, use the **other** or **previous** softkeys to show the available commands.

Status bar

Shows the current status of the target.

2.3 Connecting to a target core

For a debug session, connect to the appropriate target using **startt32**, see *Section 2.1: Starting TRACE32*. For example:

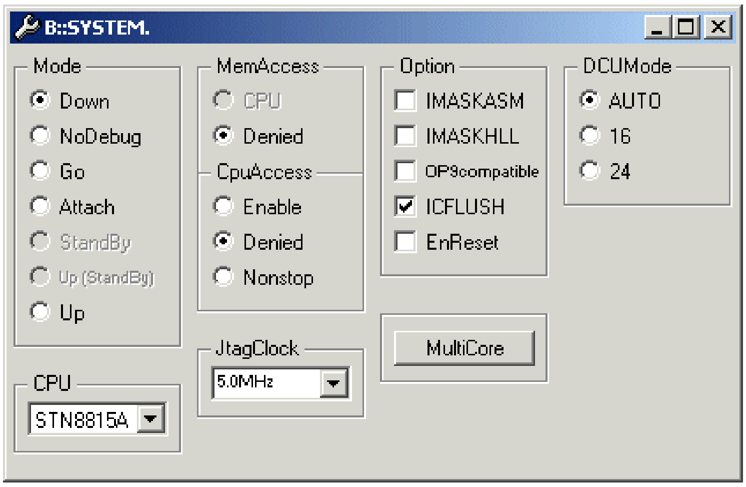
```
startt32 unchained mmdsp 8815 a
```

TRACE32 opens with STN8815A selected.

To connect to the target, do the following:

1. From the **CPU** menu select **System Settings…**, the **SYSTEM** dialog box opens, see *Figure 4.*
2. Select either:
   - **Mode: Up** to connect and reset the target core (this is the default value)
   - **Mode: Attach** to connect without resetting the core

Figure 4. Connect the target from the SYStem dialog box



B::SYSTEM.

Mode: ● Down ○ NoDebug ○ Go ○ Attach ○ StandBy ○ Up (StandBy) ○ Up

MemAccess: ○ CPU ● Denied

CpuAccess: ○ Enable ● Denied ○ Nonstop

Option: □ IMASKASM □ IMASKHLL □ OP9compatible ☑ ICFLUSH □ EnReset

DCUMode: ● AUTO ○ 16 ○ 24

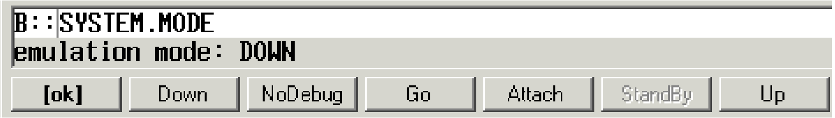
CPU: STN8815A

JtagClock: 5.0MHz

Alternately, use the softkeys:

1. Click the **SYStem** then **Mode** softkeys, the softkeys change to the available connection options, see *Figure 5.*
2. Click the softkey for the required connection and click **ok**.

Figure 5. Connecting to the target using the softkeys



```
B::SYSTEM.MODE
emulation mode: DOWN
```

2.4 Changing CPUs

The default connections are:

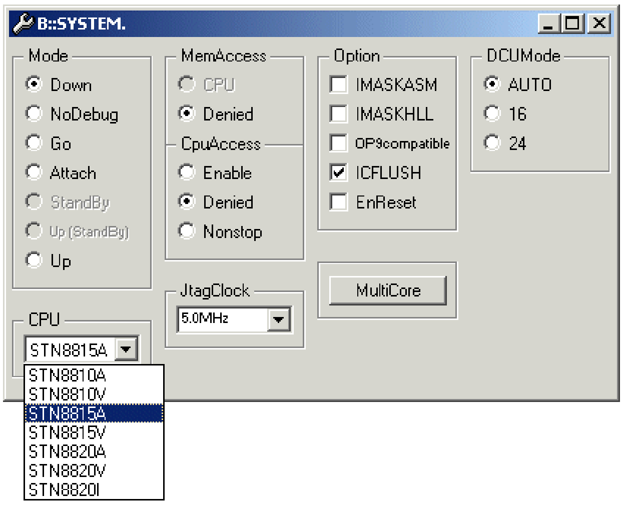
- For TRACE32-ARM, TRACE32 connects automatically to the ARM core.
- For TRACE32-MMDSP, the default CPUs are:

STn8810A (STn8810-SAA)	`8810a`
STn8810V (STn8810-SVA)	`8810v`
STn8815A (STn8815-SAA)	`8815a`
STn8815V (STn8815-SVA)	`8815v`
STn8820A (STn8820-SAA)	`8820a`
STn8820V (STn8820-SVA)	`8820v`
STn8820I (STn8820-SIA)	`8820i`

To change CPUs, do the following.

1. From the **CPU** menu select **System Settings…**, the **SYSTEM** dialog box opens, see *Figure 6.*
2. From the **CPU** drop-down list, select the CPU.

Figure 6. Changing the CPU from the SYStem dialog box



B::SYSTEM.

Mode: ● Down ○ NoDebug ○ Go ○ Attach ○ StandBy ○ Up (StandBy) ○ Up

MemAccess: ○ CPU ● Denied

CpuAccess: ○ Enable ● Denied ○ Nonstop

Option: □ IMASKASM □ IMASKHLL □ OP9compatible ☑ ICFLUSH □ EnReset

DCUMode: ● AUTO ○ 16 ○ 24

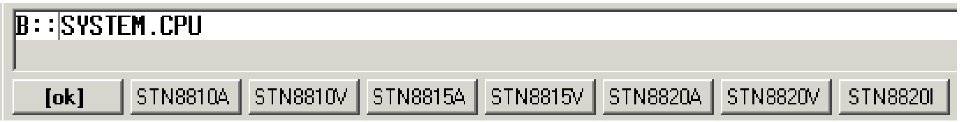
JtagClock: 5.0MHz

CPU: STN8815A

Alternately, use the softkeys:

1. Click the **SYStem** then **CPU** softkeys, the softkeys change to the available CPUs, see *Figure 7.*
2. Click the softkey for the required CPU and click **ok**.

Figure 7. Changing the CPU using the softkeys



```
B::SYSTEM.CPU
```

2.5 Loading an executable program

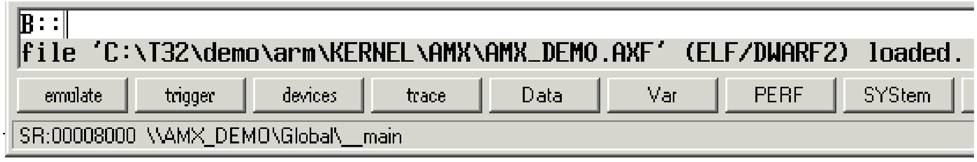
When TRACE32 connects to the selected target (see *Section 2.3: Connecting to a target core*), you are ready to debug.

If the connection to the target core is performed through the **system > up** softkeys or from the from the **CPU > system Settings > up** button, load the executable program or a binary file onto the target.

To load an executable or a binary file, do the following.

1. From the **File** menu select **Load…**, a file browser opens.
2. Locate the required executable and click **Open**. The program loads to TRACE32, the message line shows the path to the program, see *Figure 8.*

Figure 8. Loading an executable program

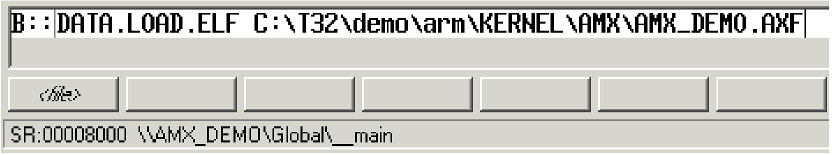


```
B::
file 'C:\T32\demo\arm\KERNEL\AMX\AMX_DEMO.AXF' (ELF/DWARF2) loaded.
SR:00008000 \\AMX_DEMO\Global\__main
```

Alternately, use the softkeys:

1. Click the **Data**, **Load** and **ELF** softkeys.
2. Type the path to the executable file and press the **Enter** key.

Figure 9. Loading an executable program using the softkeys



```
B::DATA.LOAD.ELF C:\T32\demo\arm\KERNEL\AMX\AMX_DEMO.AXF
SR:00008000 \\AMX_DEMO\Global\__main
```

If the connection to the target core is performed through the **system > mode > attach** softkeys, or from the **CPU > System settings…** menu, and if the target core memories are already loaded, there is no requirement to load any program or binary.

The debugger requires the debug segments of the program or the binary file. To load only the debug segment without loading the target core memories, use the following command.

```
data.load <file> /NOCODE
```

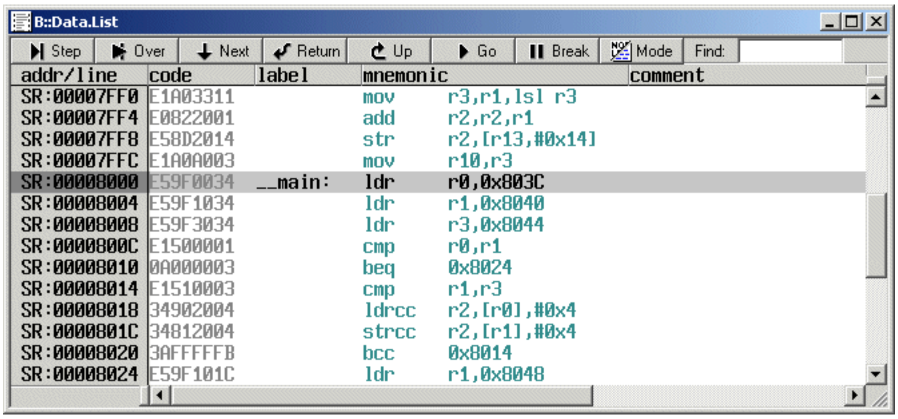
2.5.1 Viewing the executable code

There are three ways to view the executable code:

- from the **View** menu, select **List Source…**
- click the **Data**, **List** softkeys and click **ok**
- from the toolbar, click the button

The **Data.List** window opens, see *Figure 10.*

Figure 10. View the executable code



```
addr/line    code      label    mnemonic                        comment
SR:00007FF0 E1A03311           mov    r3,r1,lsl r3
SR:00007FF4 E0822001           add    r2,r2,r1
SR:00007FF8 E58D2014           str    r2,[r13,#0x14]
SR:00007FFC E1A0A003           mov    r10,r3
SR:00008000 E59F0034 __main:   ldr    r0,0x803C
SR:00008004 E59F1034           ldr    r1,0x8040
SR:00008008 E59F3034           ldr    r3,0x8044
SR:0000800C E1500001           cmp    r0,r1
SR:00008010 0A000003           beq    0x8024
SR:00008014 E1510003           cmp    r1,r3
SR:00008018 34902004           ldrcc  r2,[r0],#0x4
SR:0000801C 34812004           strcc  r2,[r1],#0x4
SR:00008020 3AFFFFFB           bcc    0x8014
SR:00008024 E59F101C           ldr    r1,0x8048
```

Executable code view modes

There are three viewing modes available:

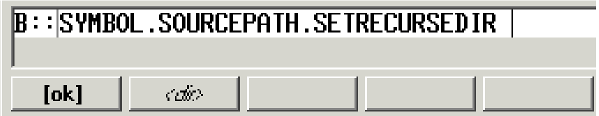
- ASM (Assembler)
- HLL (High Level Language)
- Mix.

To view the complete source code in HLL or Mix mode, TRACE32 must know the location of the source code.

To specify the source code location, do the following.

1. Click the **sYmbol**, **sourcePATH** then **SetRecurseDir** softkeys.
2. Click **ok**, see *Figure 11.*

Figure 11. Setting the executable source path using the softkeys



```
B::SYMBOL.SOURCEPATH.SETRECURSEDIR
```

There are three ways to toggle between the code view modes:

- from the **Run** menu, select **Mode**
- click the **Mode** softkeys and click **ok**
- from the toolbar, click the button

2.6 Running the executable program

TRACE32 has the ability to run, step, continue or stop the loaded executable.

Table 2. Program execution keys

Actions	Softkey	Button	Hot key
Do a single step (execute an ASM instruction)	step	⏭	F2
Step over function call or subroutines	step.over	⏭	F3
Go to the next HLL code line. Useful, for example, to leave loops	go.next	↓	F4
Go to the last instruction of a function	go.return	↙	F5
Return to the caller function	go.up	↱	F6
Run the program	go	▶	F7
Stop execution	break	⏸	F8

2.7 Breakpoints

There are two commonly used breakpoints:

- software program breakpoint

  A stop instruction replaces the break instruction to stop the real time execution. All code lines where there is a program breakpoint are marked with a small red bar.
- onchip breakpoint

  A hardware stop signal, stops the execution when it encounters the break condition. Onchip breakpoints are mainly used as breakpoints on data access, but program breakpoints are also available. The SxA cores support a maximum of two onchip program breakpoints and one onchip data breakpoint. If the number of active onchip breakpoints is beyond the hardware limit, TRACE32-MMDSP reports an error.

Breakpoints in the **Data.List** window appear as red lines next to the address line.

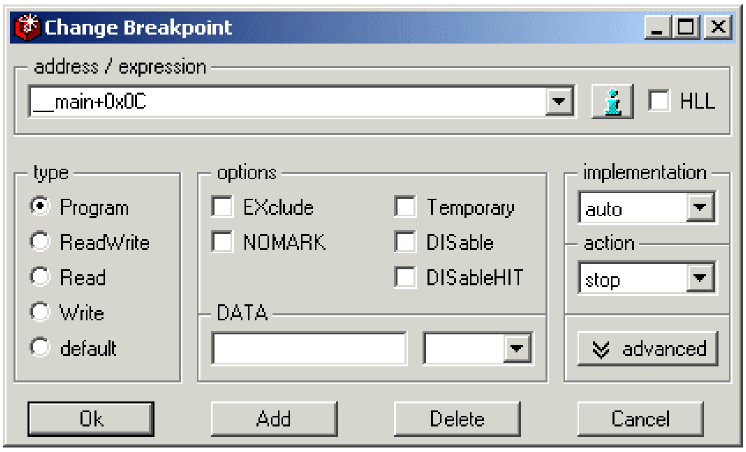
2.7.1 Setting breakpoints

There are four ways to set a breakpoint:

- from the **Data.List** window, double click on the code line (toggles on or off)
- for software breakpoints, click the **break.set <range> <address>** softkeys. For example, **BREAK.SET P:0x20**
- for onchip breakpoints, click the **break.set <range> <address> /onchip** softkeys. For example, **break.set P:0x20 /onchip**
- set breakpoints through the **Change Breakpoint** dialog box:
  - a) From the **Break** menu, select **Set…**, the change breakpoint dialog box opens, see *Figure 12.*.
  - b) Type the **address** of the breakpoint and click **Ok**.

The break condition can be configured for various variables: a data value, task magic number (for RTOS), count and so on.

Figure 12. Change Breakpoint dialog box



address / expression: __main+0x0C

☐ HLL

type: ● Program ○ ReadWrite ○ Read ○ Write ○ default

options: ☐ EXclude ☐ Temporary ☐ NOMARK ☐ DISable ☐ DISableHIT

implementation: auto

action: stop

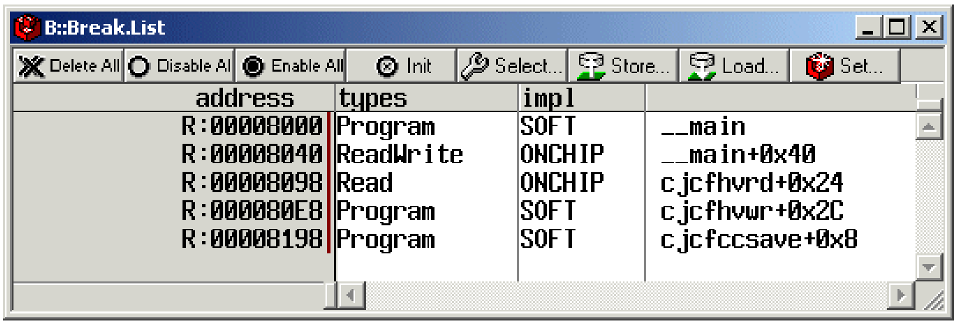
2.7.2 Viewing breakpoints

There are three ways to display information about all set breakpoints:

- from the **Break** menu select **List**.
- click the **Break**, **List** softkeys and click **ok**
- from the toolbar, click the button

The **Break.List** window opens, see *Figure 13*.

Figure 13. View breakpoints



address	types	impl	
R:00008000	Program	SOFT	__main
R:00008040	ReadWrite	ONCHIP	__main+0x40
R:00008098	Read	ONCHIP	cjcfhvrd+0x24
R:000080E8	Program	SOFT	cjcfhvwr+0x2C
R:00008198	Program	SOFT	cjcfccsave+0x8

*Note: To open a new **Data.List** window, double-click on a breakpoint.*

2.7.3 Changing breakpoints

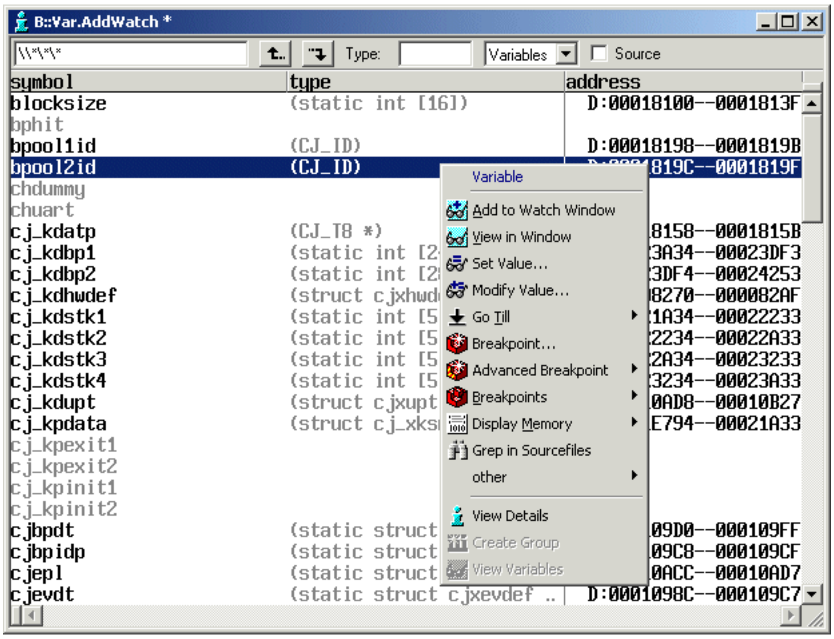
To modify or disable the breakpoint, do the following.

1. From the **Break.List** window (see *Section 2.7.2: Viewing breakpoints*), right-click on the breakpoint to open the context-sensitive popup menu and select **Change**. The **Change Breakpoint** dialog box opens, see *Figure 12: Change Breakpoint dialog box*
2. Update the setpoint to the required values.

2.8 Variables

To view the list of variables in an executable file, from the **Var** menu select **Watch**. The variable watch screen opens, see *Figure 14*.

Figure 14. View variables

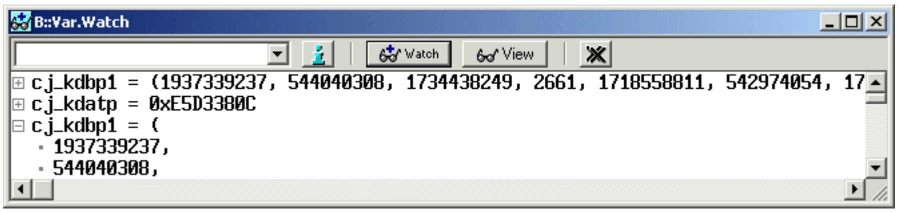


```
\\*\*\*
symbol          type                        address
blocksize       (static int [16])           D:00018100--0001813F
bphit
bpool1id        (CJ_ID)                     D:00018198--0001819B
bpool2id        (CJ_ID)                     D:0001819C--0001819F
chdummy
chuart
cj_kdatp        (CJ_T8 *)                   8158--0001815B
cj_kdbp1        (static int [2              3A34--00023DF3
cj_kdbp2        (static int [2              3DF4--00024253
cj_kdhwdef      (struct cjxhwd              8270--000082AF
cj_kdstk1       (static int [5              1A34--00022233
cj_kdstk2       (static int [5              2234--00022A33
cj_kdstk3       (static int [5              2A34--00023233
cj_kdstk4       (static int [5              3234--00023A33
cj_kdupt        (struct cjxupt              0AD8--00010B27
cj_kpdata       (struct cj_xks              E794--00021A33
cj_kpexit1
cj_kpexit2
cj_kpinit1
cj_kpinit2
cjbpdt          (static struct              09D0--000109FF
cjbpidp         (static struct              09C8--000109CF
cjepl           (static struct              0ACC--00010AD7
cjevdt          (static struct cjxevdef ..  D:0001098C--000109C7
```

There are three ways to add a variable to the watch window (see *Figure 15*.):

- double click on a variable (this closes the view variables window)
- right-click on the variable to open the context-sensitive popup menu and select **Add to Watch Window**
- open an existing watch window. Drag and drop the variable to the watch window

Figure 15. Watch window



```
⊞ cj_kdbp1 = (1937339237, 544040308, 1734438249, 2661, 1718558811, 542974054, 17
⊞ cj_kdatp = 0xE5D3380C
⊟ cj_kdbp1 = (
  · 1937339237,
  · 544040308,
```

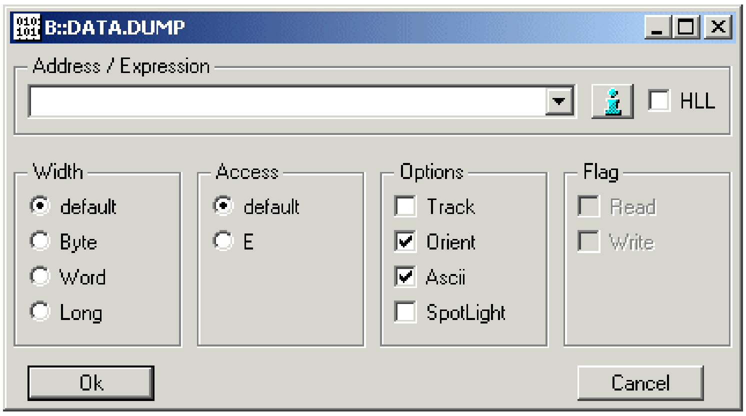
2.9 Memory

There are three ways to display the memory:

- from the **View** menu, select **Dump…**
- click the **data**, and **dump** softkeys and click **ok**
- from the toolbar, click the 0101 button

The **Data dump** dialog box opens, see *Figure 16*.

Figure 16. Memory dump dialog box



B::DATA.DUMP

Address / Expression: HLL □

Width: ● default ○ Byte ○ Word ○ Long

Access: ● default ○ E

Options: □ Track ☑ Orient ☑ Ascii □ SpotLight

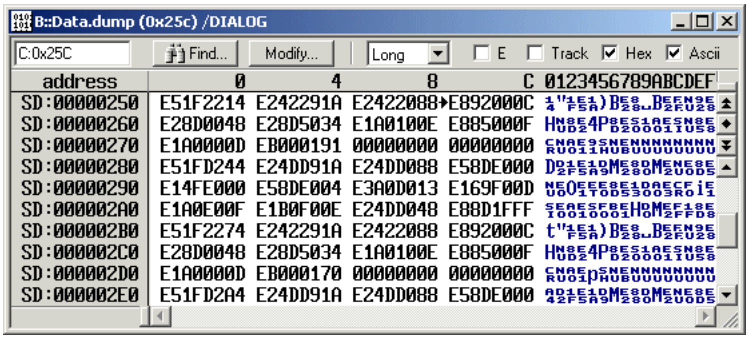
Flag: □ Read □ Write

Note: To open the symbol browser, click the button, see Section 2.10: Symbols.

To view the memory zone, do the following.

1. Type the address range.
2. If necessary, change the **Width**, **Access** and **Options**.
3. Click the **Ok** button. The memory zone containing the given address opens, see *Figure 17*.

Figure 17. Data dump window



B::Data.dump (0x25c) /DIALOG

C:0x25C | Long | □ E □ Track ☑ Hex ☑ Ascii

```
address            0        4        8        C 0123456789ABCDEF
SD:00000250 E51F2214 E242291A E2422088►E892000C
SD:00000260 E28D0048 E28D5034 E1A0100E E885000F
SD:00000270 E1A0000D EB000191 00000000 00000000
SD:00000280 E51FD244 E24DD91A E24DD088 E58DE000
SD:00000290 E14FE000 E58DE004 E3A0D013 E169F00D
SD:000002A0 E1A0E00F E1B0F00E E24DD048 E88D1FFF
SD:000002B0 E51F2274 E242291A E2422088 E892000C
SD:000002C0 E28D0048 E28D5034 E1A0100E E885000F
SD:000002D0 E1A0000D EB000170 00000000 00000000
SD:000002E0 E51FD2A4 E24DD91A E24DD088 E58DE000
```

Note: The title of the window reflects the calling parameters.

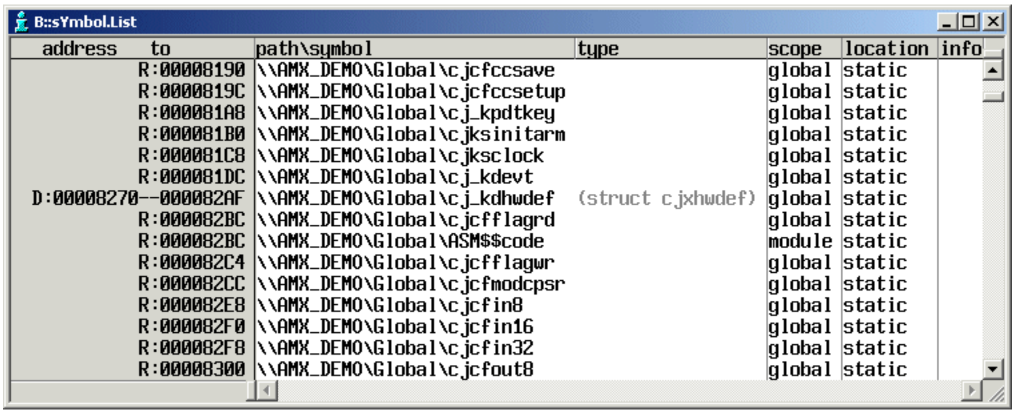
2.10 Symbols

There are two ways to view symbols such as functions, modules and types:

- from the **View** menu, select the **Symbols** and the required symbol from the menu
- click the **sYmbol** and the required symbol softkeys and click **ok**

Each view has its own display mode. For example, *Figure 18* shows the **Symbols by Type** window.

Figure 18. Symbols list



B::sYmbol.List

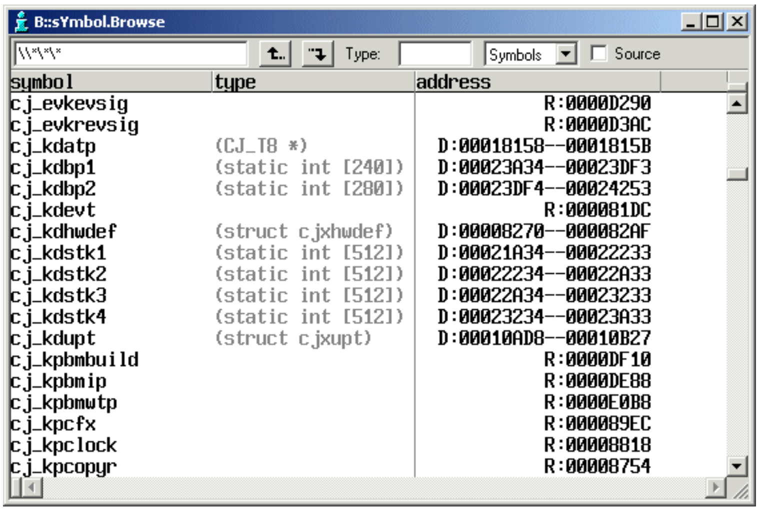
address	to	path\symbol	type	scope	location	info
	R:00008190	\\AMX_DEMO\Global\cjcfccsave		global	static	
	R:0000819C	\\AMX_DEMO\Global\cjcfccsetup		global	static	
	R:000081A8	\\AMX_DEMO\Global\cj_kpdtkey		global	static	
	R:000081B0	\\AMX_DEMO\Global\cjksinitarm		global	static	
	R:000081C8	\\AMX_DEMO\Global\cjksclock		global	static	
	R:000081DC	\\AMX_DEMO\Global\cj_kdevt		global	static	
D:00008270	--000082AF	\\AMX_DEMO\Global\cj_kdhwdef	(struct cjxhwdef)	global	static	
	R:000082BC	\\AMX_DEMO\Global\cjcfflagrd		global	static	
	R:000082BC	\\AMX_DEMO\Global\ASM$$code		module	static	
	R:000082C4	\\AMX_DEMO\Global\cjcfflagwr		global	static	
	R:000082CC	\\AMX_DEMO\Global\cjcfmodcpsr		global	static	
	R:000082E8	\\AMX_DEMO\Global\cjcfin8		global	static	
	R:000082F0	\\AMX_DEMO\Global\cjcfin16		global	static	
	R:000082F8	\\AMX_DEMO\Global\cjcfin32		global	static	
	R:00008300	\\AMX_DEMO\Global\cjcfout8		global	static	

Alternately, use the symbol browser, there are three ways to open the browser:

- from the **View** menu, select **Symbols** then **Browse**
- click the **sYmbol**, and the **Browse** softkeys and click **ok**
- from the toolbar, click the button

The **Symbol Browser** window opens, see *Figure 19*.

Figure 19. Symbols browser



B::sYmbol.Browse

*** Type: Symbols ☐ Source

symbol	type	address
cj_evkevsig		R:0000D290
cj_evkrevsig		R:0000D3AC
cj_kdatp	(CJ_T8 *)	D:00018158--0001815B
cj_kdbp1	(static int [240])	D:00023A34--00023DF3
cj_kdbp2	(static int [280])	D:00023DF4--00024253
cj_kdevt		R:000081DC
cj_kdhwdef	(struct cjxhwdef)	D:00008270--000082AF
cj_kdstk1	(static int [512])	D:00021A34--00022233
cj_kdstk2	(static int [512])	D:00022234--00022A33
cj_kdstk3	(static int [512])	D:00022A34--00023233
cj_kdstk4	(static int [512])	D:00023234--00023A33
cj_kdupt	(struct cjxupt)	D:00010AD8--00010B27
cj_kpbmbuild		R:0000DF10
cj_kpbmip		R:0000DE88
cj_kpbmwtp		R:0000E0B8
cj_kpcfx		R:000089EC
cj_kpclock		R:00008818
cj_kpcopyr		R:00008754

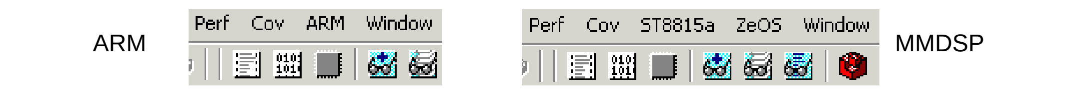
2.11 Target core’s peripherals view

Trace32 can display the hardware peripherals of the target core and make the configuration of writable peripherals possible.

After connecting to the target core, Trace32 adds an additional menu with the core name.

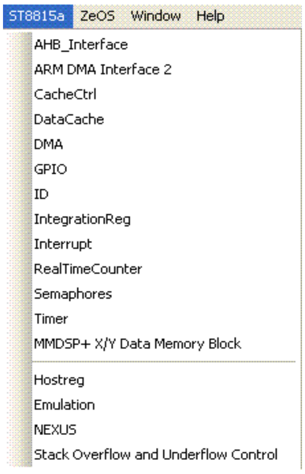
For ARM cores there is an ARM menu, for MMDSP the menu shows the core name, for example, ST8815a, see *Figure 20*.

Figure 20. peripheral menu with core name



The menu lists all the available peripherals. For example: STN8815A peripherals.

Figure 21. peripherals menu



ST8815a ZeOS Window Help

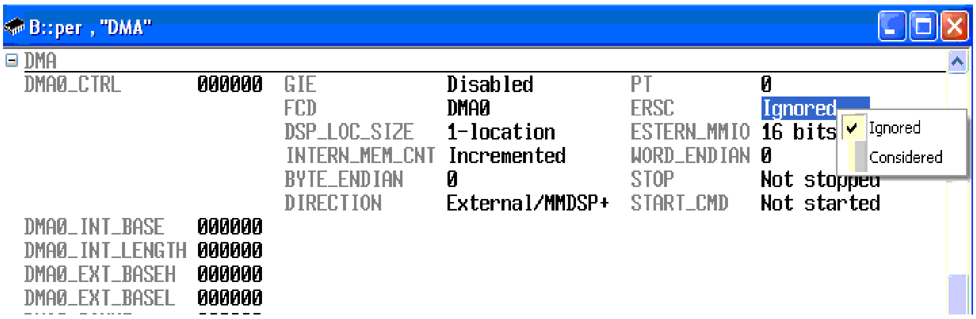
- AHB_Interface
- ARM DMA Interface 2
- CacheCtrl
- DataCache
- DMA
- GPIO
- ID
- IntegrationReg
- Interrupt
- RealTimeCounter
- Semaphores
- Timer
- MMDSP+ X/Y Data Memory Block
- Hostreg
- Emulation
- NEXUS
- Stack Overflow and Underflow Control

Each target core corresponds to a `.per` file that defines all the available peripherals and the read/write rules of fields.

To view pop-up information about a field, select and keep the cursor on it the field.

To change a value, right-click on the value to open the context-sensitive popup menu and select **set** or **modify**. For some values, there can be a pull-down menu that displays meaningful values.

Figure 22. pull-down popup menu



```
B::per , "DMA"
⊟ DMA
  DMA0_CTRL        000000  GIE              Disabled        PT            0
                           FCD              DMA0            ERSC          Ignored
                           DSP_LOC_SIZE     1-location      ESTERN_MMIO   16 bits
                           INTERN_MEM_CNT   Incremented     WORD_ENDIAN   0
                           BYTE_ENDIAN      0               STOP          Not stopped
                           DIRECTION        External/MMDSP+ START_CMD     Not started
  DMA0_INT_BASE    000000
  DMA0_INT_LENGTH  000000
  DMA0_EXT_BASEH   000000
  DMA0_EXT_BASEL   000000
```

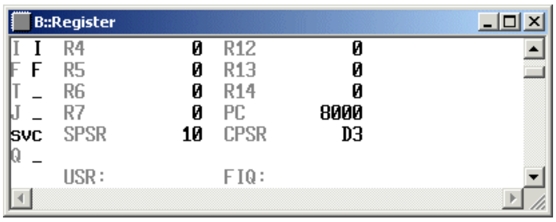
2.12 Registers

There are three ways to display the registers:

- from the **View** menu, select **Registers**
- click the **Register** softkey and click **ok**
- from the toolbar, click the button

The **Registers** window opens, see *Figure 23*.

Figure 23. Registers window



```
I  I   R4        0  R12        0
F  F   R5        0  R13        0
T  _   R6        0  R14        0
J  _   R7        0  PC      8000
svc    SPSR     10  CPSR      D3
Q  _
       USR:         FIQ:
```

To change the value of a register, do the following.

1. Double-click on the value to be changed. The register value opens in the command bar.
2. In the command bar, type the new value and click the **ok** softkey.

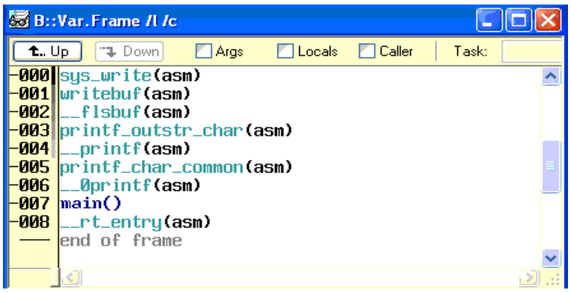
2.12.1 Stack

There are two ways to display the registers:

- from the **View** menu, select **Stackframe**
- from the toolbar, click the button

The **Stack** window opens, see *Figure 24*.

Figure 24. Stack window



```
-000 sys_write(asm)
-001 writebuf(asm)
-002 __flsbuf(asm)
-003 printf_outstr_char(asm)
-004 __printf(asm)
-005 printf_char_common(asm)
-006 __0printf(asm)
-007 main()
-008 __rt_entry(asm)
 ——  end of frame
```

To check the caller of the current function, click the **UP** button. Trace32 shows the caller function as the current function and the registers and variables and so on, are switched to be that of the callers too. Note that the current function, that is, pc register and stack content on the target core does not affected.

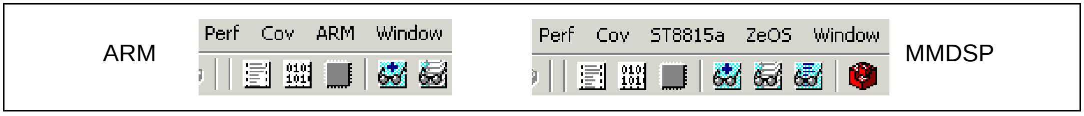
To return to the callee function, click on the **Down** button.

2.13 RTOS (TRACE32-MMDSP only)

For MMDSP targets, an additional ZeOS menu item opens, see *Figure 25*.

TRACE32 is ZeOS (SAA RTOS) aware. Additional features are available such as tasks (thread) information, statistics, stack coverage and task related debugging.

Figure 25. ZeOS menu



3 Performance analysis tools

TRACE32 has performance analysis tools that are based on up 6 methods. 3 methods are usable with ARM and SxA on the Nomadik:

- StopAndGo (default)

  The debugger stops the core, reads the PC, then resumes the execution. This is intrusive.

- Trace

  The debugger retrieves the PC history from Nexus trace in the case of SxA or ETM trace in the case of ARM. This is not intrusive.

  This method is available only if the target board has a Nexus or ETM trace connection, and the Nexus preprocessor or ETM preprocessor is connected to the trace port on the board. See *Appendix B: Target board configuration on page 55* and *Appendix D: Connections on page 59*.

- Snoop

  The debugger uses special target core features to get the PC while the target is running. This is not intrusive. This method is available for ARM and SxA in STn8820 and later.

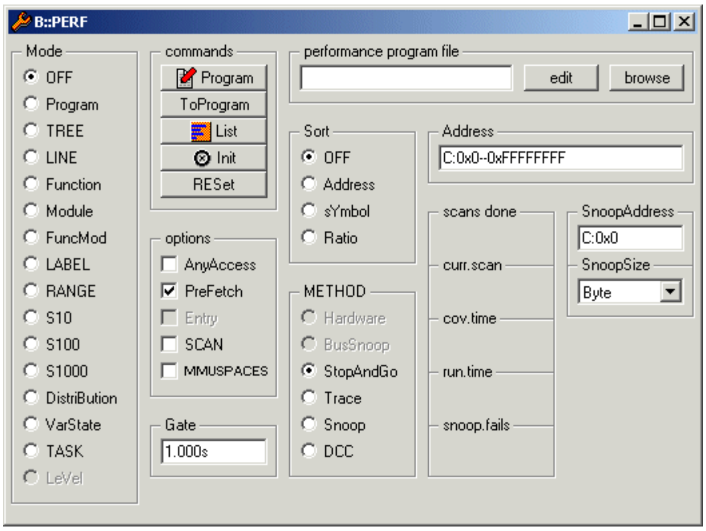
The performance analysis tools give information such as:

- minimum, maximum and average execution times for each function
- the number of times each function has been called
- details of code coverage
- other performance information

To select the items to monitor, do the following.

1. From the **Perf** menu, select **Perf Configuration…** the **PERF** dialog box opens, see *Figure 26*.
2. Select the items to monitor.

Figure 26. Performance configuration



B::PERF

Mode: ● OFF ○ Program ○ TREE ○ LINE ○ Function ○ Module ○ FuncMod ○ LABEL ○ RANGE ○ S10 ○ S100 ○ S1000 ○ DistriBution ○ VarState ○ TASK ○ LeVel

commands: Program, ToProgram, List, Init, RESet

options: □ AnyAccess ☑ PreFetch □ Entry □ SCAN □ MMUSPACES

Gate: 1.000s

performance program file: edit, browse

Sort: ● OFF ○ Address ○ sYmbol ○ Ratio

METHOD: ○ Hardware ○ BusSnoop ● StopAndGo ○ Trace ○ Snoop ○ DCC

Address: C:0x0--0xFFFFFFFF

scans done

curr.scan

cov.time

run.time

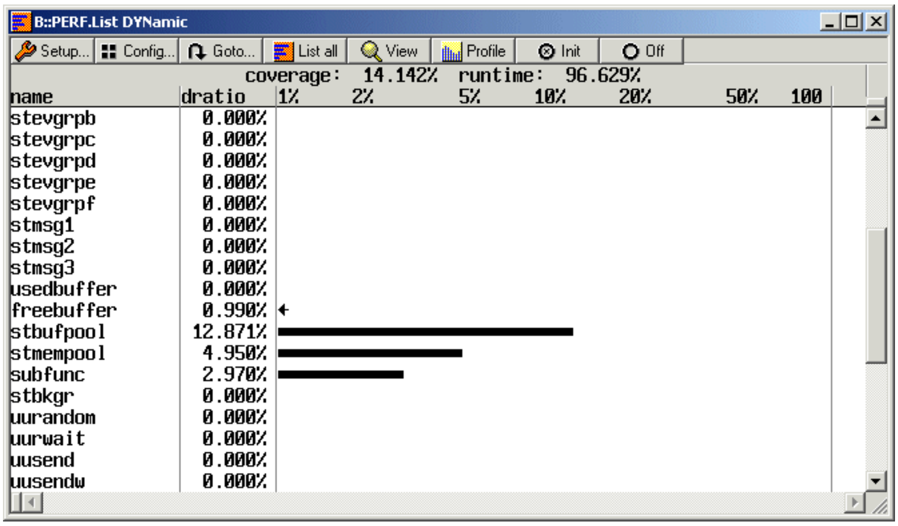
snoop.fails

SnoopAddress: C:0x0

SnoopSize: Byte

To view the results, for example function performance, from the **Perf** menu, select **Perf List…** the **Perf** dialog box opens, see *Figure 27*.

Figure 27. Perf.List.DYNamic



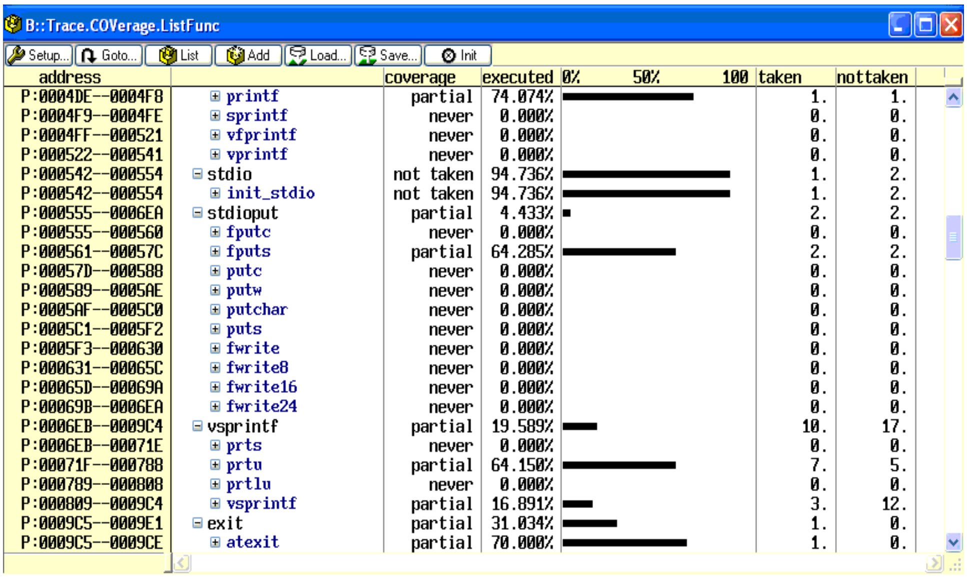
```
                    coverage:  14.142%  runtime:  96.629%
name           dratio  1%     2%       5%      10%      20%         50%   100
stevgrpb       0.000%
stevgrpc       0.000%
stevgrpd       0.000%
stevgrpe       0.000%
stevgrpf       0.000%
stmsg1         0.000%
stmsg2         0.000%
stmsg3         0.000%
usedbuffer     0.000%
freebuffer     0.990%
stbufpool     12.871%
stmempool      4.950%
subfunc        2.970%
stbkgr         0.000%
uurandom       0.000%
uurwait        0.000%
uusend         0.000%
uusendw        0.000%
```

3.1 Coverage statistics

The coverage analysis tool is based on the Nexus or ETM trace.

After a trace capture, but before the coverage analysis tools in TRACE32 can be activated, add the trace (Nexus or ETM) data to the coverage buffer, see *Section 4.2: Nexus trace capture on page 38*. From the **Cov menu,** select **Add Tracebuffer**.

Figure 28. Coverage statistics



```
address                                    coverage   executed  0%      50%      100  taken  nottaken
P:0004DE--0004F8      ⊞ printf             partial    74.074%                          1.       1.
P:0004F9--0004FE      ⊞ sprintf              never     0.000%                          0.       0.
P:0004FF--000521      ⊞ vfprintf             never     0.000%                          0.       0.
P:000522--000541      ⊞ vprintf              never     0.000%                          0.       0.
P:000542--000554    ⊟ stdio              not taken    94.736%                          1.       2.
P:000542--000554      ⊞ init_stdio       not taken    94.736%                          1.       2.
P:000555--0006EA    ⊟ stdioput             partial     4.433%                          2.       2.
P:000555--000560      ⊞ fputc                never     0.000%                          0.       0.
P:000561--00057C      ⊞ fputs              partial    64.285%                          2.       2.
P:00057D--000588      ⊞ putc                 never     0.000%                          0.       0.
P:000589--0005AE      ⊞ putw                 never     0.000%                          0.       0.
P:0005AF--0005C0      ⊞ putchar              never     0.000%                          0.       0.
P:0005C1--0005F2      ⊞ puts                 never     0.000%                          0.       0.
P:0005F3--000630      ⊞ fwrite               never     0.000%                          0.       0.
P:000631--00065C      ⊞ fwrite8              never     0.000%                          0.       0.
P:00065D--00069A      ⊞ fwrite16             never     0.000%                          0.       0.
P:00069B--0006EA      ⊞ fwrite24             never     0.000%                          0.       0.
P:0006EB--0009C4    ⊟ vsprintf             partial    19.589%                         10.      17.
P:0006EB--00071E      ⊞ prts                 never     0.000%                          0.       0.
P:00071F--000788      ⊞ prtu               partial    64.150%                          7.       5.
P:000789--000808      ⊞ prtlu                never     0.000%                          0.       0.
P:000809--0009C4      ⊞ vsprintf           partial    16.891%                          3.      12.
P:0009C5--0009E1    ⊟ exit                 partial    31.034%                          1.       0.
P:0009C5--0009CE      ⊞ atexit             partial    70.000%                          1.       0.
```

4 Using Nexus trace in TRACE32

The SxA target core generates Nexus trace data that is collected by the debug tools. TRACE32 gives:

- a visual representation of the execution of the embedded application
- monitoring of data memory accesses which occur during the execution

For information about the MMDSP+ implementation of the Nexus Trace unit, see the *HAMAC Audio/Video IP for Nomadik 8820 (ADCS 8044228)*.

TRACE32 makes MMDSP-Nexus trace available and associates it with the debugging sessions. It also performs advanced analysis of Nexus trace to provide valuable application statistics.

4.1 Configuring Nexus trace

The default configuration for Nexus tracing is as follows:

- Nexus maintains the entire history of the program counter from the beginning of execution to the stall state of the target core
- Nexus deactivates trace monitoring of data memory access

If specifically configured, Nexus filters the trace generation according to user defined criteria and generates a trace that monitors data memory accesses, see the *HAMAC Audio/Video IP for Nomadik 8820 (8044228)*.

Before carrying out a customized configuration, execute the command `do nexusconfig on`. This command ensures that TRACE32 abandons the default configuration. To subsequently return to the default configuration, execute the command `do nexusconfig off`.

To reconfigure the filters or watch points of Nexus trace, perform one of the following operations.

- In the menu ST881xx/Nexus, set the Nexus register values. This method is available for the Nexus in the MMDSP+s on the STn8810, STn8815 and later.
- use the commands listed below. This method is not available for the Nexus in the MMDSP+s on the STn8810.

```
do nexus_reset_8815
do nexus_fil_conf_8815 <fil_number> <FLTVAL0> <FLTVAL1> <FLTCTL>
do nexus_wth_config_8815 <wtch_number> <WTCHVAL0> <WTCHVAL1>
<WTCHVAL2> <WTCHCNT> <WTCHCTL>
do nexus_enable_8815 <NXSCTl>
```

For examples of configuration commands for the STn8815, see *Section 4.2: Nexus trace capture*

Note: All the above commands (as well as `nexusconfig on/off`) are `.cmm` script files that are not included in the TRACE32 initial installation package. These script files are provided to aid the Nexus configuration. They are delivered within Nomadik Toolset under the folder:

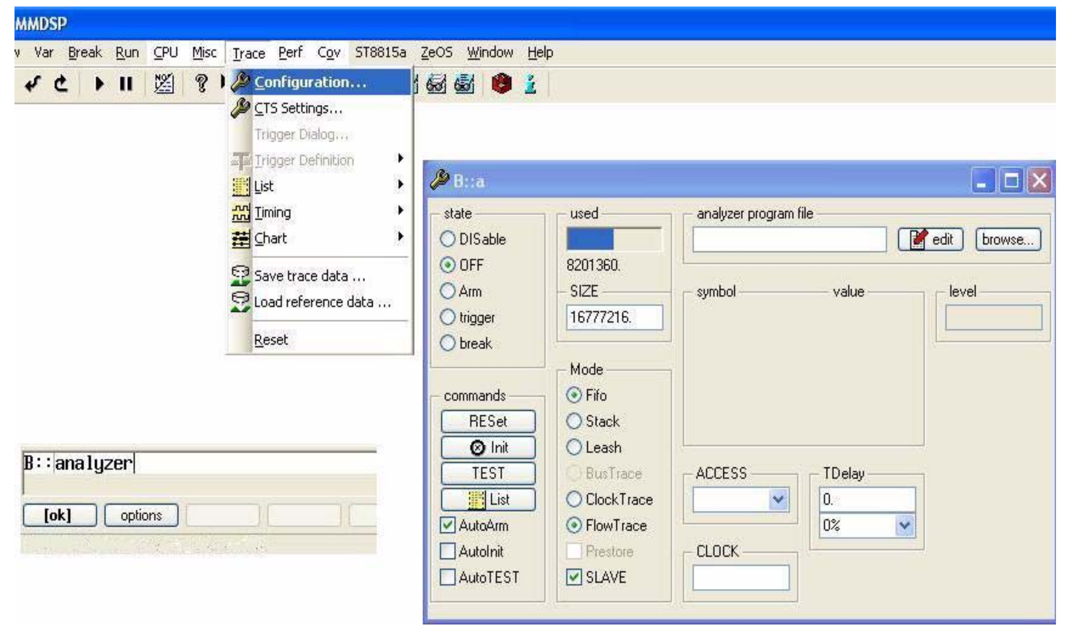
```
<NDKTOOLS_ROOT>\configuration\debugger\trace32
```

4.2 Nexus trace capture

Only PTn (see) provides the Nexus trace capture service. PTd is not suitable in this case. Connect the PTn to the target board with the help of the scheme PTn, PDd+PTn or PTd+PTn.

PTn is equipped with a 16 Mb buffer that temporarily stocks the trace to be read by the software interpreter module. TRACE32 enables selection of the buffering mode from **Fifo**, **Stack**, **leash**, and so on.

Figure 29. Nexus trace capture



B::analyzer

state: ○ DISable ● OFF ○ Arm ○ trigger ○ break

used: 8201360.

SIZE: 16777216.

Mode: ● Fifo ○ Stack ○ Leash ○ BusTrace ○ ClockTrace ● FlowTrace □ Prestore ☑ SLAVE

☑ AutoArm □ AutoInit □ AutoTEST

TDelay: 0. 0%

Click on the **Reset** button to delete collected messages if necessary.

Run the application program (after configuring the Nexus trace, if needed). The trace capture can be monitored by the **used** bar which shows the run-time occupation of the buffer.

4.3 View Program and data by Nexus trace

When the application stops, or is stopped by a **break** command, the execution can be viewed by clicking on the **list** button in the **analyzer** window.

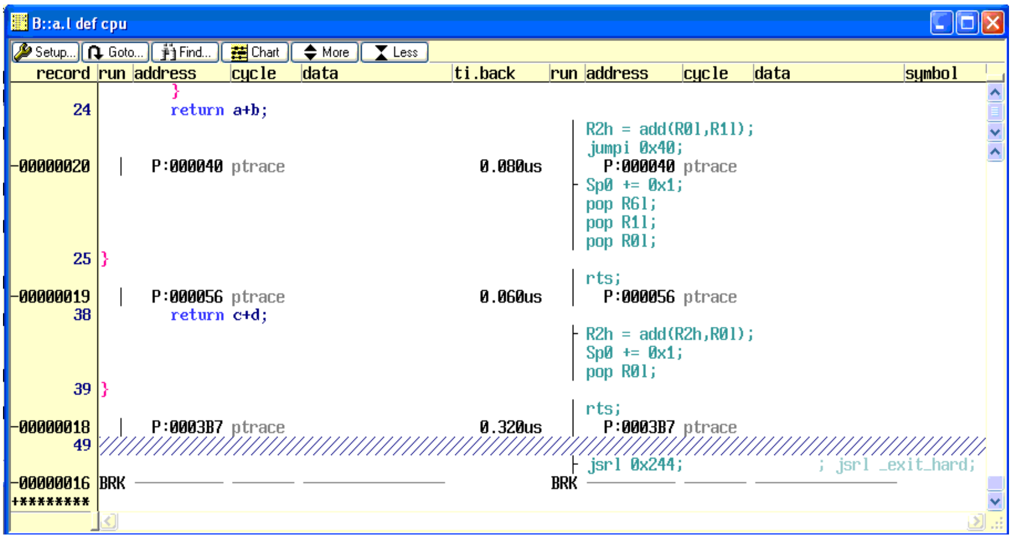
To help make the output easier to read, we recommend the use of the following command. The command separates the program addresses and HLL from disassembler code.

```
analyzer.list def cpu
```

By default, the analyzer is positioned at the end of the execution. To go to the beginning, use the **CTRL + Home** key combination.

To search the listing, use the **CTRL + f** combination key. This is useful, for example, when searching for a specific function.

Figure 30. List window



```
    record run address    cycle   data           ti.back    run address    cycle   data               symbol
                    }
        24          return a+b;
                                                               R2h = add(R01,R11);
                                                               jumpi 0x40;
-00000020   |    P:000040 ptrace                   0.080us       P:000040 ptrace
                                                               Sp0 += 0x1;
                                                               pop R61;
                                                               pop R11;
                                                               pop R01;
        25 }
                                                               rts;
-00000019   |    P:000056 ptrace                   0.060us       P:000056 ptrace
        38          return c+d;
                                                               R2h = add(R2h,R01);
                                                               Sp0 += 0x1;
                                                               pop R01;
        39 }
                                                               rts;
-00000018   |    P:0003B7 ptrace                   0.320us       P:0003B7 ptrace
        49 ////////////////////////////////////////////////////////////////////////////////////////////
                                                               jsrl 0x244;                 ; jsrl _exit_hard;
-00000016 BRK ──────── ───── ────────────               BRK ──────── ───── ────────────
+*********
```

4.4 Example of using Nexus in STn8815

This section gives a number of typical configurations for Nexus tracing.

Note: *Due to a hardware flaw, from STn8815 cut2 and later, the Nexus capture requires the running of the program* `go <program_address>` *or* `go <function>`*, where <*`p:program_address`*> and <*`function`*> is a server attained spot by the execution. For example:*

```
go 0xFFFFF
```

The following examples assume that the program is loaded in the zone `[0x0 -- 0xFFFF]`, view of MMDSP+.

Example one

To capture the execution history of program between 0x123 and 0xABC, use the following the filtering.

1. Configure the trace:
   ```
   system.option btm off
   do nexusconfig on
   do nexus_reset_8815
   do nexus_fil_conf_8815 0 0x123 0xABC 1
   do nexus_enable_8815 1
   ```
2. Run the program:
   ```
   go p:0xffff (or a never attained address or function)
   ```
3. Restore the `btm` configuration:
   ```
   sys.o btm on
   ```

To view the trace output:

```
analyzer.list def cpu
```

Example two

To capture the execution history of all instructions reading or writing the memory address 0x400000, and output data access value:

1. Configure the trace:
   ```
   system.option btm off
   do nexusconfig on
   do nexus_reset_8815
   do nexus_fil_conf_8815 0 0 0xffff 1
   do nexus_fil_conf_8815 1 0x400000 0x400000 0xF
   do nexus_wth_conf_8815 0 0x400000 0x400000 0 0 0x1B
   do nexus_enable_8815 0x11
   ```
2. Run the program:
   ```
   go p:0xffff (or a never attained address or function)
   ```
3. Restore the `btm` configuration:
   ```
   sys.o btm on
   ```

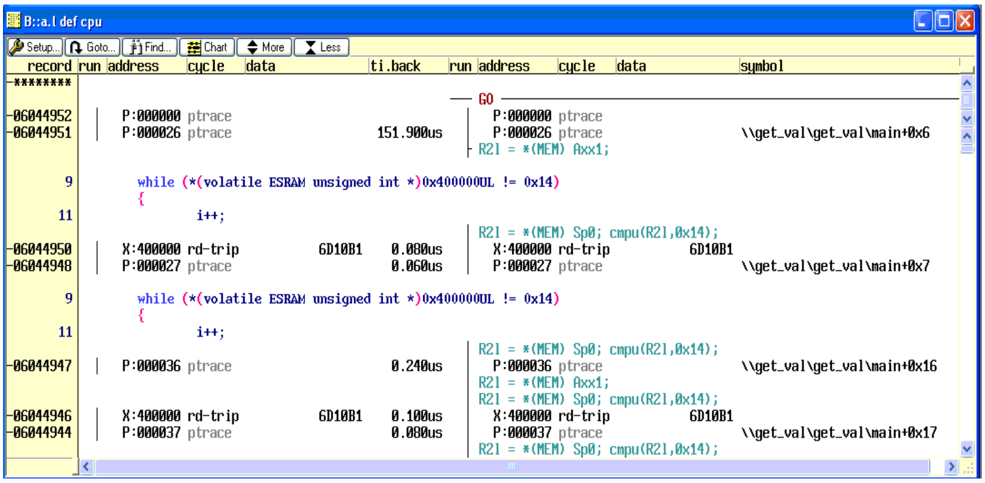
To view the trace output use the following command.

```
analyzer.list def cpu
```

Each `wr-trip` corresponds to a `P:addr` that is just above the `wr-trip`. The `P:addr` is the instruction that writes the defined data memory address (`0x400000`). To search each `wr-trip` and look at the `P:addr` above the `wr-trip`. The **data** column displays the written value.

Each `rd-trip` corresponds to a `P:addr` that is just above the `rd-trip`. The `P:addr` is the instruction that reads the defined data memory address (0x400000). To search each `rd-trip` and look at the `P:addr` above the `rd-trip`. The **data** column displays the read value.

Figure 31. Example output



```
   record run address    cycle   data           ti.back   run address    cycle   data            symbol
-********
                                                          ── GO ─────────────────────────────────────────────────
-06044952  |   P:000000 ptrace                              |  P:000000 ptrace
-06044951  |   P:000026 ptrace                  151.900us   |  P:000026 ptrace                   \\get_val\get_val\main+0x6
                                                            ├ R21 = *(MEM) Axx1;

        9         while (*(volatile ESRAM unsigned int *)0x400000UL != 0x14)
                  {
       11                 i++;
                                                            | R21 = *(MEM) Sp0; cmpu(R21,0x14);
-06044950  |   X:400000 rd-trip          6D10B1   0.080us   |  X:400000 rd-trip          6D10B1
-06044948  |   P:000027 ptrace                    0.060us   |  P:000027 ptrace                   \\get_val\get_val\main+0x7

        9         while (*(volatile ESRAM unsigned int *)0x400000UL != 0x14)
                  {
       11                 i++;
                                                            | R21 = *(MEM) Sp0; cmpu(R21,0x14);
-06044947  |   P:000036 ptrace                    0.240us   |  P:000036 ptrace                   \\get_val\get_val\main+0x16
                                                            | R21 = *(MEM) Axx1;
                                                            | R21 = *(MEM) Sp0; cmpu(R21,0x14);
-06044946  |   X:400000 rd-trip          6D10B1   0.100us   |  X:400000 rd-trip          6D10B1
-06044944  |   P:000037 ptrace                    0.080us   |  P:000037 ptrace                   \\get_val\get_val\main+0x17
                                                            | R21 = *(MEM) Sp0; cmpu(R21,0x14);
```

Example three

To capture the history of each execution of the program between 0x123 and 0xABC and sub-function calls. A watch point is required, that is, the trace generation and output is enabled when `PC== 0x123` and is disabled at the moment that `PC== 0xABC`.

1. Configure the trace:
   ```
   system.option btm off
   do nexusconfig on
   do nexus_reset_8815
   do nexus_fil_conf_8815 0 0 0xFFFF 1
   do nexus_wth_conf_8815 0 0x123 0xABC 0 0 0xC1
   do nexus_enable_8815 0x11
   ```
2. Run the program:
   ```
   go p:0xffff (or a never attained address or function)
   ```
3. Restore the `btm` configuration:
   ```
   sys.o btm on
   ```

To view the trace output:

```
analyzer.list def cpu
```

Example four

To capture the history of every third time execution of the program between 0x123 and 0xABC, and, in addition, the sub function calls:

1. Configure the trace:
   ```
   system.option btm off
   do nexusconfig on
   do nexus_reset_8815
   do nexus_fil_conf_8815 0 0 0xFFFF 1
   do nexus_wth_conf_8815 0 0x123 0xABC 0 2 0xC1
   do nexus_enable_8815 0x11
   ```
2. Run the program:
   ```
   go p:0xffff (or a never attained address or function)
   ```
3. Restore the `btm` configuration:
   ```
   sys.o btm on
   ```

To view the trace output:

```
analyzer.list def cpu
```

Example five

To capture the history of the read and write access to the stack (from memory address 0x123 to 0xABC):

1. Configure the trace:
```
system.option btm off
do nexusconfig on
do nexus_reset_8815
do nexus_fil_conf_8815 0 0x123 0xABC 0xF
do nexus_enable_8815 0x1
```
2. Run the program:
```
go p:0xffff (or a never attained address or function)
```
3. Restore the `btm` configuration:
```
sys.o btm on
```

To view the trace output:

```
analyzer.list def cpu
```

Example six

To capture the execution history of the program from 0x123 to 0xABC (without subfunctions calls) and to monitor all data access performed by this section of program:

1. Configure the trace:
```
system.option btm off
do nexusconfig on
do nexus_reset_8815
do nexus_fil_conf_8815 0 0x123 0xABC 0x11
do nexus_fil_conf_8815 1 0x0 0xFFFFFF 0xF
do nexus_enable_8815 0x1
```
2. Run the program:
```
go p:0xffff (or a never attained address or function)
```
3. Restore the `btm` configuration:
```
sys.o btm on
```

To view the trace output:

```
analyzer.list def cpu
```

Example 7

To capture the execution history only of the program between 0x123 and 0xABC (without subfunctions calls), a filter is required. To also monitor the data access from the address 0x235 to 0x434, performed by this section of program:

1. Configure the trace:
```
system.option btm off
do nexusconfig on
do nexus_reset_8815
do nexus_fil_conf_8815 0 0x123 0xABC 0x11
do nexus_fil_conf_8815 1 0x235 0x434 0xF
do nexus_enable_8815 0x1
```
2. Run the program:
```
go p:0xffff (or a never attained address or function)
```
3. Restore the `btm` configuration:
```
sys.o btm on
```

To view the trace output:

```
analyzer.list def cpu
```

Example 8

To capture the history of each execution of the program from 0x123 to 0xABC as well as the sub-function calls. To also monitor the data access from the address 0x235 to 0x434, performed by this section of program:

1. Configure the trace:
```
system.option btm off
do nexusconfig on
do nexus_reset_8815
do nexus_fil_conf_8815 0 0 0xFFFF 1
do nexus_fil_conf_8815 1 0x235 0x434 0xF
do nexus_wth_conf_8815 0 0x123 0xABC 0 0 0xC1
do nexus_enable_8815 0x10411
```
2. Run the program:
```
go p:0xffff (or a never attained address or function)
```
3. Restore the `btm` configuration:
```
sys.o btm on
```

To view the trace output:

```
analyzer.list def cpu
```

5 Cross debugging

Nomadik (STn8810, STn8815 and STn8820) has the ability to perform cross debugging. It is possible to propagate a `stop` or `stop at breakpoint` event from one target core to another. For example, if cross debugging is established between the ARM and SAA, when the ARM stops (either at a breakpoint or otherwise), the SAA stops simultaneously.

The `mmxdbg.cmm` script helps to configure the cross debugging. It is delivered with the Nomadik Toolset in the following folder.

```
<NDKTOOLS_ROOT>\configuration\debugger\trace32
```

See *Section 1.1.1: Installing the run environments on page 10*.

The parameters for cross debugging are:

```
do mmxdbg <break_src> <break_dest> <and_or>
```

Where:

`<break_src>` The source of the BREAK signal. It can be `ARM`, `SAA`, `SVA`, `SIA` (8820 only), `EXTBRK`, `RESET` or `SBAG` (8820 only).

If *`<break_src>`* is `RESET`, the script resets the cross debugging configuration so that there is no source or destination of the `break` signal.

`<break_dest>` The destination of the `break` signal. It can be `ARM`, `SAA`, `SVA`, `SIA` (8820 only).

`<and_or>` Can be either `AND` or `OR`.

When `AND`, the condition that `break_src` is stopped or not is logically `AND` combined with other conditions to decide if `dest_src` is to be stopped.

When `OR`, the condition is logically `OR` combined with other conditions.

To enable cross debugging, in the command line, type `do mmxdbg`.

Note: 1 In Trace32-ARM, use the `do mmxdbg` command.

*2 If `<break_dest>` is SAA, SVA or SIA, and the Trace32-MMDSP requires connection to the target core by using the **up** mode, use the `do mmxdbg` command after the connection. This is because the **up** mode resets the target core and clears the configuration performed by `do mmxdbg`.*

Example one

In Trace32-ARM, to configure the ARM and SAA for cross debugging, use the following commands.

```
do mmxdbg RESET
do mmxdbg ARM SAA OR
```

Outcome When the ARM core stops (for example, by a breakpoint), the SAA core stops simultaneously.

Condition "ARM is stopped or not" is an OR condition that determines if SAA is to be stopped.

Example two

In Trace32-ARM, executing the following commands configures the ARM, SAA, SVA for cross debugging:

```
do mmxdbg RESET
do mmxdbg ARM SAA OR
do mmxdbg SVA SAA OR
```

Outcome When either the ARM or the SVA core stops (for example, by a breakpoint), the SAA stops simultaneously.

Condition "ARM is stopped or not" is an `OR` condition that determines if the SAA is to be stopped.

Condition "SVA is stopped or not" is an `OR` condition that determines if the SAA is to be stopped.

6 Miscellaneous and tips

This chapter describes miscellaneous tips when configuring and connecting to target boards.

6.1 Terminal interface for I/O operations (MMDSP only)

TRACE32 uses the `mmterm.cmm` script from `t32config.zip` to activate the terminal window for TRACE32-MMDSP, see *Section 1.1.1: Installing the run environments on page 10*.

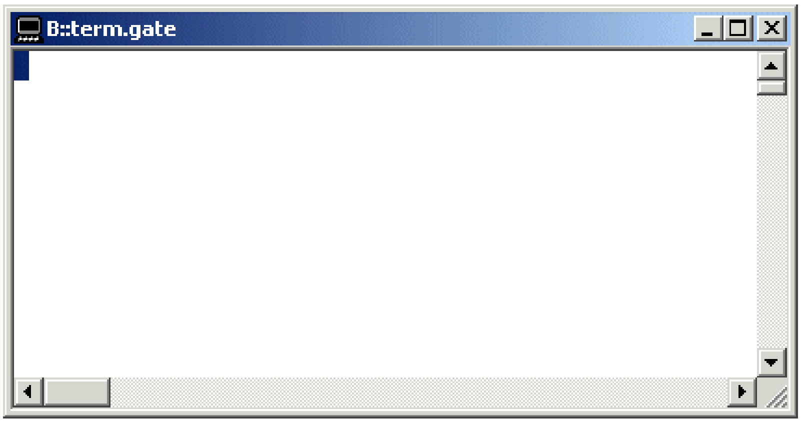
In TRACE32-MMDSP, the terminal window serves as the stdin/stdout device. For example, for standard C I/O operations such as `printf()`.

The terminal window opens automatically when TRACE32 starts. The TRACE32 boot file `xy-t32.cmm` activates the I/O terminal in TRACE32-ARM by default, see *Section 1.2.3: Connecting the Trace32 Devices to the target board on page 12*.

To open the terminal window, do the following.

1. Ensure that the `mmterm.cmm` script from the `t32config.zip` is present in the default folder, see *Section 1.1.1: Installing the run environments on page 10*.
2. In the command line, type `do.mmterm`, see *Figure 32.*

Figure 32. The terminal window



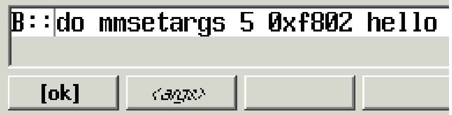
6.2 Setting arguments for an MMDSP program

The application running on the MMDSP possibly needs to set some of the `main (argc, argv)` function arguments.

To set the arguments, do the following.

1. Ensure that the `mmsetargs.cmm` script from the `t32config.zip` is present in the default folder, see *Section 1.2.3: Connecting the Trace32 Devices to the target board on page 12*.
2. In the command line, type `do mmsetargs <arg1> <arg2>` and so on, see *Figure 33.*

Figure 33. Setting the main function arguments example



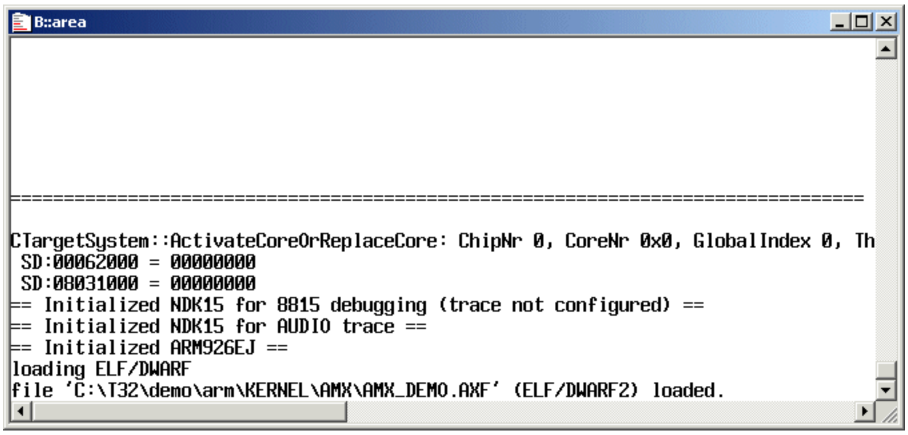
```
B::do mmsetargs 5 0xf802 hello
```

6.3 Displaying TRACE32 messages

The **area** window shows TRACE32 runtime messages, error messages and so on. It opens automatically when TRACE32 starts.

To open the **area** window, in the command line, type `area`, see *Figure 34.*

Figure 34. The area window



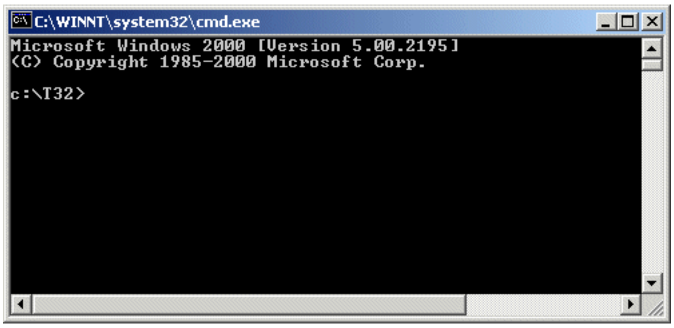
```
==================================================================================

CTargetSystem::ActivateCoreOrReplaceCore: ChipNr 0, CoreNr 0x0, GlobalIndex 0, Th
 SD:00062000 = 00000000
 SD:08031000 = 00000000
== Initialized NDK15 for 8815 debugging (trace not configured) ==
== Initialized NDK15 for AUDIO trace ==
== Initialized ARM926EJ ==
loading ELF/DWARF
file 'C:\T32\demo\arm\KERNEL\AMX\AMX_DEMO.AXF' (ELF/DWARF2) loaded.
```

6.4 Executing commands on the host shell

The `OS` command executes a command on the host shell.

To open a host shell, in the command line, type either `os` or `os.screen`, see *Figure 35.*

Figure 35. Host shell



C:\WINNT\system32\cmd.exe

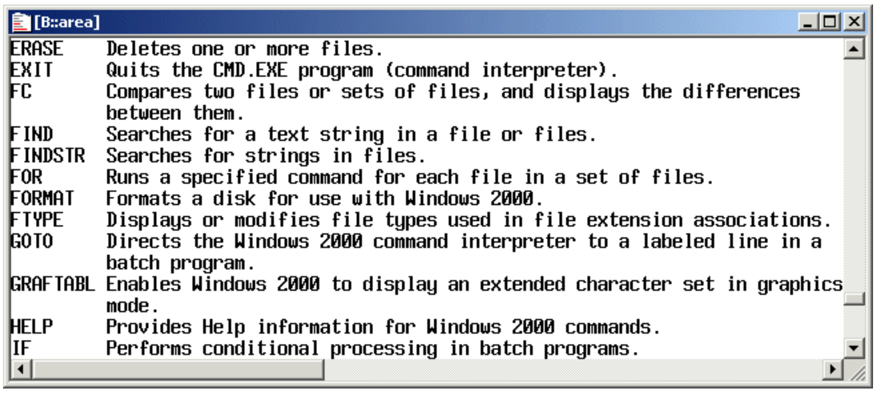
```
Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-2000 Microsoft Corp.

c:\T32>
```

A command can run on the host operating system level and the output re-routed to the **area** window.

To execute a host shell command and display it in the area window, do the following.

1. Open an area window, in the command line, type **area** an area window opens.
2. In the command line, type **os.area <commandline>**, for example **os.area help**, see *Figure 36.*

Figure 36. Area window



[B::area]

```
ERASE    Deletes one or more files.
EXIT     Quits the CMD.EXE program (command interpreter).
FC       Compares two files or sets of files, and displays the differences
         between them.
FIND     Searches for a text string in a file or files.
FINDSTR  Searches for strings in files.
FOR      Runs a specified command for each file in a set of files.
FORMAT   Formats a disk for use with Windows 2000.
FTYPE    Displays or modifies file types used in file extension associations.
GOTO     Directs the Windows 2000 command interpreter to a labeled line in a
         batch program.
GRAFTABL Enables Windows 2000 to display an extended character set in graphics
         mode.
HELP     Provides Help information for Windows 2000 commands.
IF       Performs conditional processing in batch programs.
```

6.5 Printing

You can print the contents of the active window to:

- a printer
- the clipboard
- to file

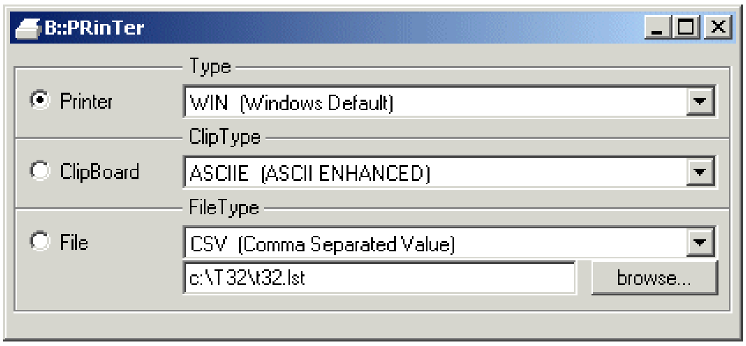
6.5.1 Print preferences

There are two ways to set the print preferences:

- from the **File** menu, select **Print** then **Printer selection…**
- click the **Printer** softkey and click **ok**

The printer dialog box opens, see *Figure 36.*

Figure 37. Print preferences



B::PRinTer

● Printer — Type: WIN (Windows Default)

○ ClipBoard — ClipType: ASCIIE (ASCII ENHANCED)

○ File — FileType: CSV (Comma Separated Value)

c:\T32\t32.lst

6.5.2 Print the contents of a window

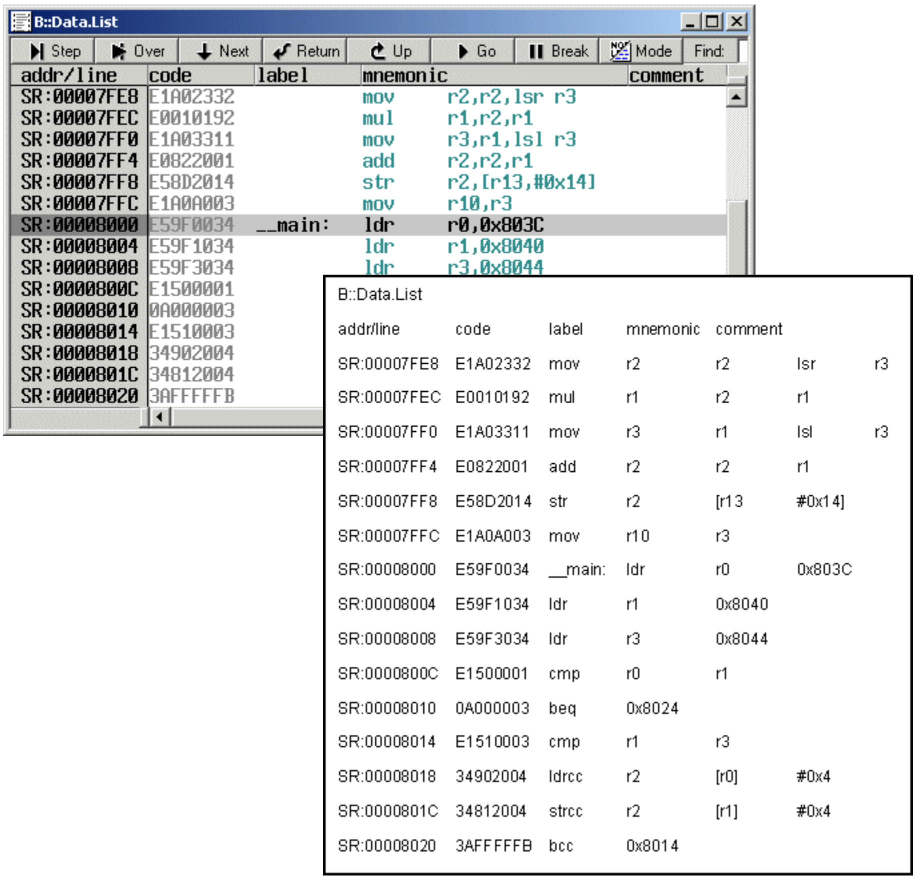
There are two ways to print the contents of a window:

- from the **File** menu, select **Print** then **Hardcopy**
- click the **PRinTer** and **Hardcopy** softkeys and click **ok**

The contents of the window prints to the printer, clipboard or file, see *Figure 38.*

Note: Only the visible part of the window prints.

Figure 38. Window contents to be printed



```
addr/line      code      label     mnemonic             comment
SR:00007FE8    E1A02332            mov      r2,r2,lsr r3
SR:00007FEC    E0010192            mul      r1,r2,r1
SR:00007FF0    E1A03311            mov      r3,r1,lsl r3
SR:00007FF4    E0822001            add      r2,r2,r1
SR:00007FF8    E58D2014            str      r2,[r13,#0x14]
SR:00007FFC    E1A0A003            mov      r10,r3
SR:00008000    E59F0034  __main:   ldr      r0,0x803C
SR:00008004    E59F1034            ldr      r1,0x8040
SR:00008008    E59F3034            ldr      r3,0x8044
SR:0000800C    E1500001
SR:00008010    0A000003
SR:00008014    E1510003
SR:00008018    34902004
SR:0000801C    34812004
SR:00008020    3AFFFFFB
```

```
B::Data.List

addr/line      code       label    mnemonic  comment

SR:00007FE8    E1A02332   mov      r2        r2      lsr     r3
SR:00007FEC    E0010192   mul      r1        r2      r1
SR:00007FF0    E1A03311   mov      r3        r1      lsl     r3
SR:00007FF4    E0822001   add      r2        r2      r1
SR:00007FF8    E58D2014   str      r2        [r13    #0x14]
SR:00007FFC    E1A0A003   mov      r10       r3
SR:00008000    E59F0034   __main:  ldr       r0      0x803C
SR:00008004    E59F1034   ldr      r1        0x8040
SR:00008008    E59F3034   ldr      r3        0x8044
SR:0000800C    E1500001   cmp      r0        r1
SR:00008010    0A000003   beq      0x8024
SR:00008014    E1510003   cmp      r1        r3
SR:00008018    34902004   ldrcc    r2        [r0]    #0x4
SR:0000801C    34812004   strcc    r2        [r1]    #0x4
SR:00008020    3AFFFFFB   bcc      0x8014
```

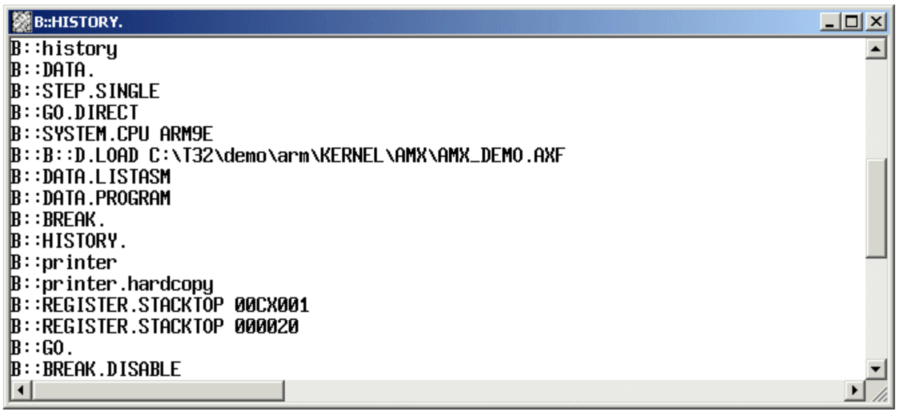
6.6 Commands history

TRACE32 remembers every successfully completed command line entry.

To display a complete list of the successful commands, do the following.

1. Click the **HISTory** softkey.
2. Click **ok**. The History window opens, see *Figure 39.*

Figure 39. Command history



```
B::history
B::DATA.
B::STEP.SINGLE
B::GO.DIRECT
B::SYSTEM.CPU ARM9E
B::B::D.LOAD C:\T32\demo\arm\KERNEL\AMX\AMX_DEMO.AXF
B::DATA.LISTASM
B::DATA.PROGRAM
B::BREAK.
B::HISTORY.
B::printer
B::printer.hardcopy
B::REGISTER.STACKTOP 00CX001
B::REGISTER.STACKTOP 000020
B::GO.
B::BREAK.DISABLE
```

Click on any of the commands to send it to the command line. To execute the command, click **ok**.

6.7 Save and reuse settings

TRACE32 can save all the current settings to an executable script file. The setting can later be re-applied for a subsequent session. The settings include:

- TRACE32's configuration system setting
- performance analysis tools setting
- interface set-up such as windows size and layout

To store the settings, do the following.

1. Click the **STOre** softkey.
2. Type the path and name of the executable script file.
3. Click **ok**.

For example, store c:\mylayout.cmm, stores the current layout to an executable script file named `mylayout.cmm`.

To reapply a configuration do the following.

1. In the command line, type **do c:\mylayout.cmm**.
2. Click **ok**. TRACE32 applies the windows settings defined in the file.

6.8 Logging

Logging records all commands to a file until a complementary close command is given.

To start logging, do the following.

1. Click the **LOG** softkey.
2. Type the path and name of the log file, For example `C:\mylog`.
3. Click **ok**. TRACE32 writes all commands to the log file.

To view the current log, click the **LOG** softkey and click **ok**.

To close the current log, click the **LOG** and **CLOSE** softkeys and click **ok**.

Note: TRACE32 does not write to the log file until it is closed.

The following factors apply to log files:

- the **log** command records not only the executed softkey commands but also the operations activated by a mouse

  Mouse commands are recorded as their line-oriented softkey command equivalents.
- every **log.open <file>** command generates a new file

  If a file with the same name already exists, it is overwritten.
- the size of the file is unlimited
- when the log has been activated, command execution slows down due to the recording
- name the log file with the `.cmm` suffix

  This enables the commands recorded in the log file to be re-executed by entering **do <file.cmm>**.

6.9 TRACE32 help

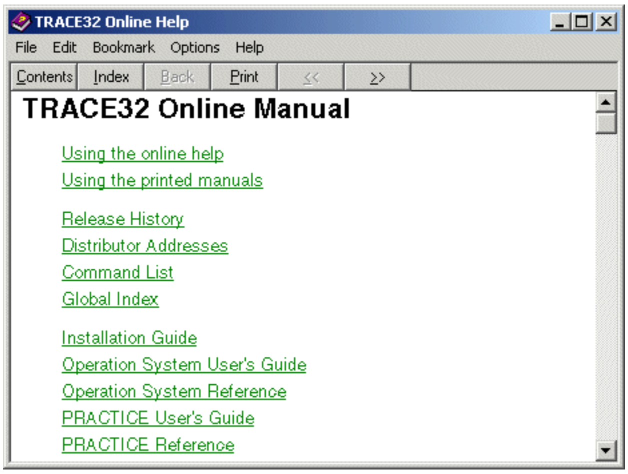
We strongly recommend that you consult the TRACE32 help system. It is especially helpful for checking the softkey command syntax and usages.

There are three ways to open the help:

- from the **Help** menu, select **Help topics…**
- press F1
- from the toolbar, click the ? button

The help window opens, see *Figure 40.*

Figure 40. Trace32 help



TRACE32 Online Help
File Edit Bookmark Options Help
Contents Index Back Print << >>

TRACE32 Online Manual

Using the online help
Using the printed manuals

Release History
Distributor Addresses
Command List
Global Index

Installation Guide
Operation System User's Guide
Operation System Reference
PRACTICE User's Guide
PRACTICE Reference

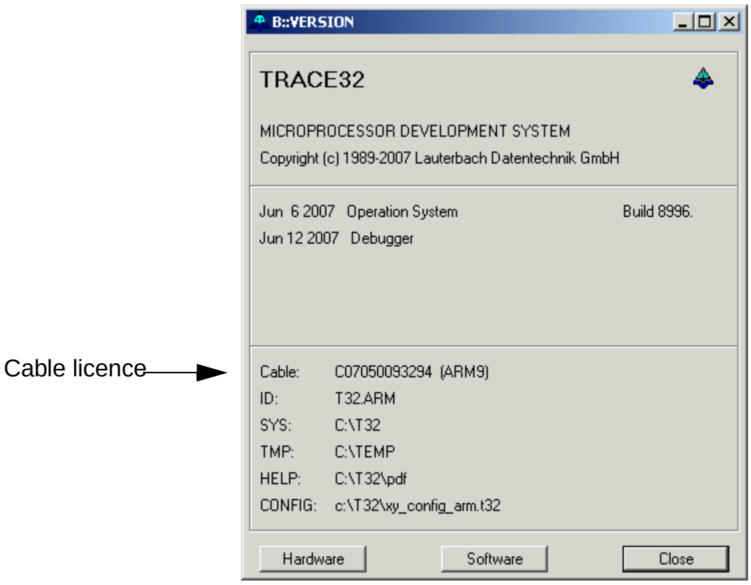
Appendix A License and guarantee

A.1 License

A TRACE32 license key is associated with the serial number of a hardware device such as a debug cable or a Nexus adapter (also known as a Nexus preprocessor).

To obtain the serial number of a debug cable or a Nexus adapter, from the Help menu, select **About TRACE32…**

Figure 41. About TRACE32 window



B::VERSION

TRACE32

MICROPROCESSOR DEVELOPMENT SYSTEM
Copyright (c) 1989-2007 Lauterbach Datentechnik GmbH

```
Jun  6 2007   Operation System                        Build 8996.
Jun 12 2007   Debugger
```

```
Cable:    C07050093294  (ARM9)
ID:       T32.ARM
SYS:      C:\T32
TMP:      C:\TEMP
HELP:     C:\T32\pdf
CONFIG:   c:\T32\xy_config_arm.t32
```

The license key is valid for 12 months. After 12 months, contact TRACE32 purchase support to update the licence[a]. Without this update, TRACE32 runs in a demo mode. In demo mode TRACE32 is accessible and usable with full functionality, but utilization is limited to 60 minutes. When this time has expired, TRACE32 must be closed and restarted to get another 60-minute session.

Each time a TRACE32 software update or patch is received, make sure that the license keys associated with the hardware devices are still valid.

Note: *The beta-test license for TRACE32-MMDSP has a valid period of 6 months instead of 12 months.*

A.2 Guarantee

Lauterbach gives a guarantee of 12 months for software and 3 years for hardware. Within the guarantee periods, free software upgrades and free hardware repair are provided.

[a] An update fee may be charged.

Appendix B Target board configuration

This appendix describes the target board configurations for the NDK15 and COB20 boards.

B.1 NDK15 boards

B.1.1 Switches configurations for debug and Nexus trace

Table 3 shows how to configure Nomadik Development Kit boards for debugging and Nexus trace.

Table 3. NDK boards configurations for debugging and Nexus trace

	NDK-15-REVB	NDK-15-REVC	NDK-15 rev1.0

Debug ARM and SxA both through MAIN JTAG port.
(ARM Debug unit and SxA Debug unit are in the “chained” mode.)
	SW1-2 = ON
	SW4-1 = ON
	SW4-2 = OFF
	SW4-4 = OFF
	SW8-2 = ON
	SW2-2 = OFF
		W1-2 = ON
		SW4-1 = ON
		SW4-2 = OFF
		SW4-4 = OFF
		SW8-2 = ON
		SW2-2 = OFF[(1)]
			SW1-2 = ON
			SW4-1 = ON
			SW4-2 = OFF
			SW8-2 = ON
			SW2-2 = OFF

Debug ARM MAIN JTAG port.
Debug SxA through NEXUS mictor.
(ARM Debug Unit and SxA Debug Unit are in the ”unchained” mode.)
	SW1-2 = ON
	SW4-1 = ON
	SW4-2 = OFF
	SW4-4 = OFF
	SW8-2 = ON
	SW2-2 = ON
	SW9-1 = OFF
	SW9-2 = ON
	SW5-4 = OFF(CPLD not available)
		SW1-2 = ON
		SW4-1 = ON
		SW4-2 = OFF
		SW4-4 = OFF
		SW8-2 = ON
		SW2-2 = ON
		SW9-1 = OFF
		SW9-2 = ON
		SW5-4, R54, R56[(2)]
		All dips of SW11, SW12 are at OFF position.
			SW1-2 = ON
			SW4-1 = ON
			SW4-2 = OFF
			SW8-2 = ON
			SW2-2 = ON
			SW5-4 = OFF (CPLD not available)
			JP1~4= pos2-3 or not connected.

Table 3. NDK boards configurations for debugging and Nexus trace (continued)

	NDK-15-REVB	NDK-15-REVC	NDK-15 rev1.0

Debug ARM through MAIN JTAG port,
Debug SxA through NEXUS mictor,
Nexus trace through NEXUS mictor.
(ARM Debug Unit and SxA Debug Unit are in the "unchained" mode.)
	SW1-2 = ON
	SW4-1 = ON
	SW4-2 = OFF
	SW4-4 = OFF
	SW8-2 = ON
	SW2-2 = ON
	SW9-1 = OFF
	SW9-2 = ON
	SW5-4 = OFF(CPLD not available)
		SW1-2 = ON
		SW4-1 = ON
		SW4-2 = OFF
		SW4-4 = OFF
		SW8-2 = ON
		SW2-2 = ON
		SW9-1 = OFF
		SW9-2 = ON
		SW5-4, R54, R56[2]
		Disconnect R59-R66[3]
		All dips of SW11, SW12 are at OFF position.
			SW1-2 = ON
			SW4-1 = ON
			SW4-2 = OFF
			SW8-2 = ON
			SW2-2 = ON
			SW5-4 = OFF (CPLD not available)
			JP1~4= pos2-3

Debug ARM through MAIN JTAG port,
Debug SxA through NEXUS mictor,
Nexus trace through NEXUS mictor,
(ARM Debug unit and SxA Debug unit are in the "chained" mode.)
	Not supported by the NDK-15 rev2	SW1-2 = ON
		SW4-1 = ON
		SW4-2 = OFF
		SW4-4 = OFF
		SW8-2 = ON
		SW2-2 = OFF
		SW9-1 = OFF
		SW9-2 = OFF
		SW5-4, R54, R56[2]
		Disconnect R59-R66[1]
		All dips of SW11, SW12 are at ON position.
			SW1-2 = ON
			SW4-1 = ON
			SW4-2 = OFF
			SW8-2 = ON
			SW2-2 = OFF
			SW5-4 = OFF (CPLD not available)
			JP1~4= pos1-2

1. To make SW2-2 effective, SW10-1, SW10-2 should be both at OFF at power-on/reset phase.
2. SxA debug shares `GPIOs[33-37]` with peripherals such as keyboard, MSP1 and LCD.
3. Keyboard is not available if R59-R66 are disconnected.

To make GPIOs available for SxA debug, select the appropriate configuration.

- Disconnect R54, R56 so that SxA debug and peripherals are both available.
- Disconnect only R56, SW5-4 = OFF so that SxA debug is available and peripherals are NOT available.

B.1.2 Distinguish the debug modes

Table 4. How to distinguish the debug modes:

	NDK-15-REVB	NDK-15-REVC	NDK-15 rev1.0
Chained	SW2-2 = OFF	SW2-2 = OFF	SW2-2 = OFF
Unchained	SW2-2 = ON	SW2-2 = ON	SW2-2 = ON

B.2 COB20 boards

B.2.1 Switch configurations for debug and Nexus trace

Switch positions:

OFF: Away from the Nomadik chip.

ON: Towards the Nomadik chip.

Table 5. switches configuration for COB20 boards for debugging

Description	COB20-A
ARM debug or ARM+SxA debug in chained debug mode through MAIN JTAG port	TAPSEL is OFF SCANEN is ON
SxA debug (with or without Nexus trace) in unchained debug mode through NEXUS mictor	NEXUS is ON[1] VPIP is OFF

[1] Remove resistors R715 to R718, R727, R729, R700. If there is poor Nexus trace quality, the following resistors possibly need to be removed: R701 ~ R706, R713, R714.

B.2.2 Debug mode change in STn8820

By default, the COB20 board is in chained debug mode.

To change to the “unchained” mode, run the following command in Trace32-ARM.

```
do jtagmode unchained
```

There is no need to reconnect the debugger to the target core. By default, after setting the unchained mode on the STn8820, the `jtagmode` command updates the debug configuration and reconnects the debugger.

Appendix C TRACE32 devices

Table 6. TRACE32 devices

	Without Nexus trace			With Nexus trace	
	Debug ARM standalone	**Debug SxA standalone**	**Debug ARM + SxA**	**Debug SxA standalone**	**Debug ARM + SxA**
Separated ARM and SxA JTAGs in Nomadik[1]	PDd[2]	PTn	PDd[2] + PTn	PTn	PDd[2]+ PTn
Chained ARM and SxA JTAGs in Nomadik[3]	PDd[2]	PDd[2]	PDd[2]	PTn	PDd[2] + PTn
				(not supported by NDK-15 rev2 boards)	(not supported by NDK-15 rev2 boards)

1. Default configuration on COB-10(B) and NDK-15 boards
2. A PDd can be substituted by a PTd
3. For details on setting up chained ARM and SxA JTAG TAPs, see *Appendix B: Target board configuration*

Table 7. Legend used in *Table 6*

Device code	Description	Connection
PDd	PowerDebug (LA-7705) with a debug cable (LA-7742)	The debug cable is connected to the debug cable port on the PowerDebug (see *Figure 42*)
PTd	PowerTrace (LA-7707) with a debug cable (LA-7742)	The debug cable is connected to the debug cable port on the PowerTrace (see *Figure 43*)
PTn	PowerTrace (LA-7707) with a Nexus preprocessor dongle (LA-7625)	The Nexus preprocessor is connected to the C connector on the PowerTrace (see *Figure 44*)
+	Connected by the PODBUS	PODBUS OUT port is connected to the PODBUS IN port

Appendix D Connections

Nexus connector: labelled NEXUS and P3

JTAG interface: labelled J6. It is also marked by JTAG on the COB-10(B) boards and by MAIN JTAG on NDK-15 boards.

Figure 42. PDd

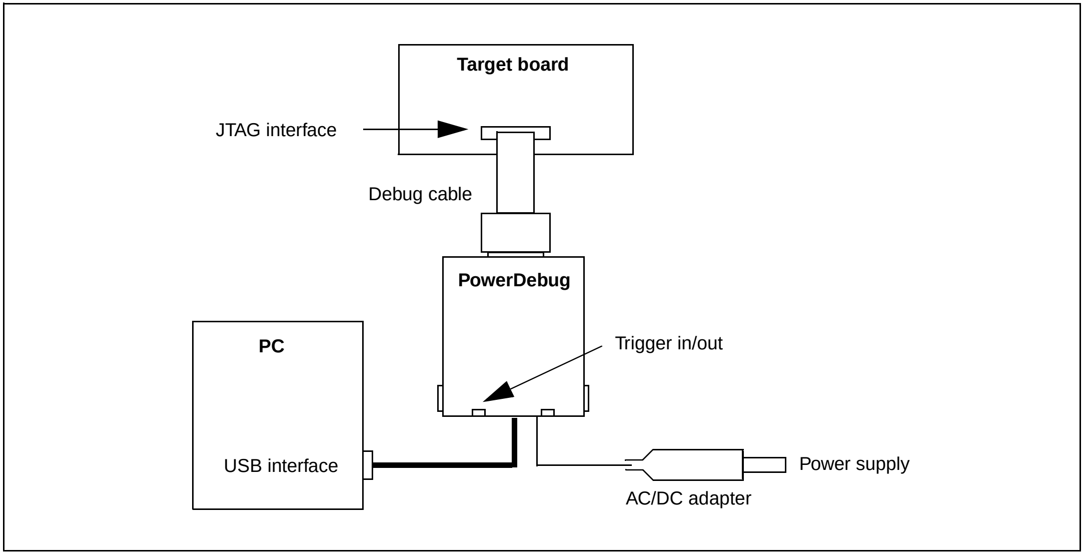


Figure 43. PTd

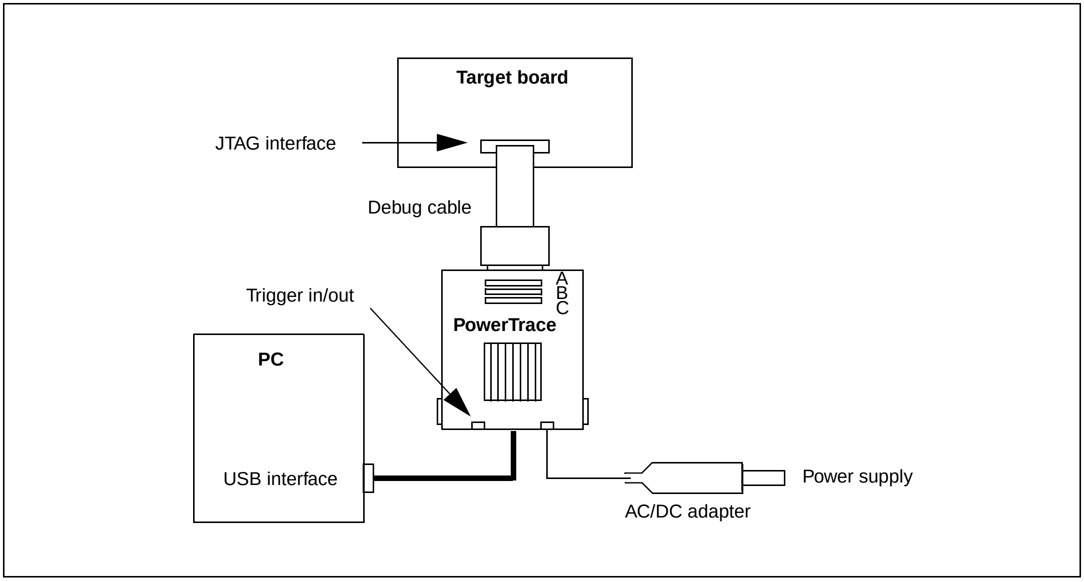


Figure 44. PTn

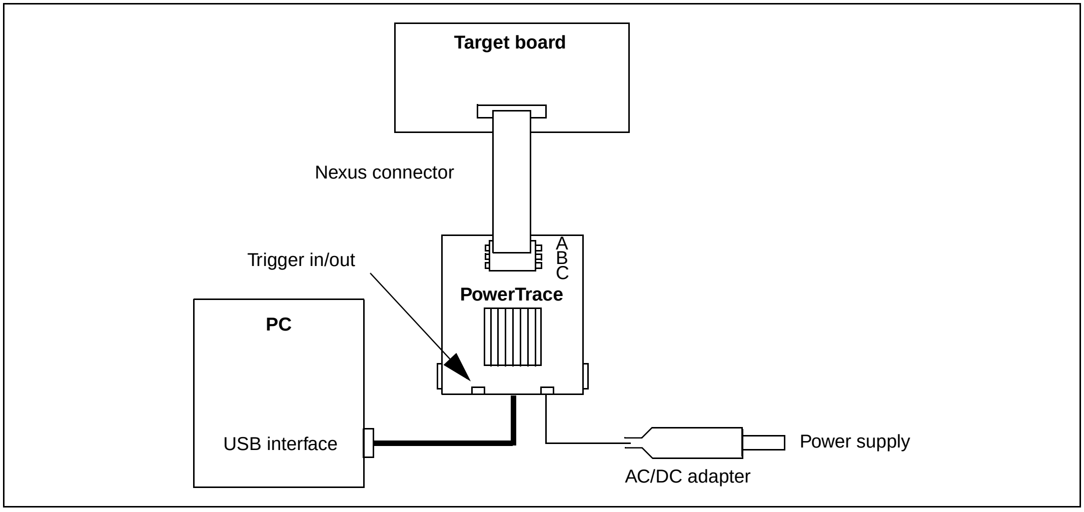


Figure 45. PDd + PTn

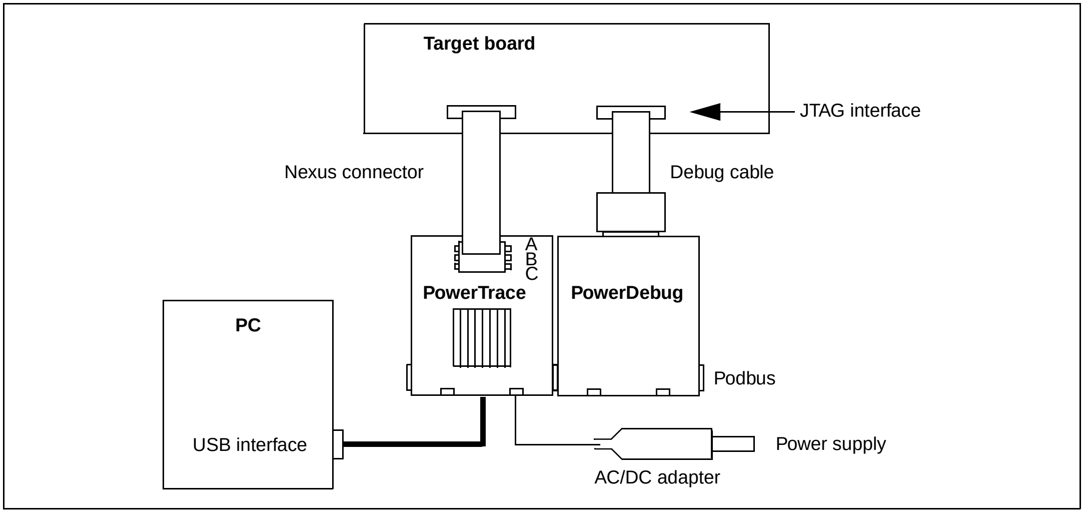
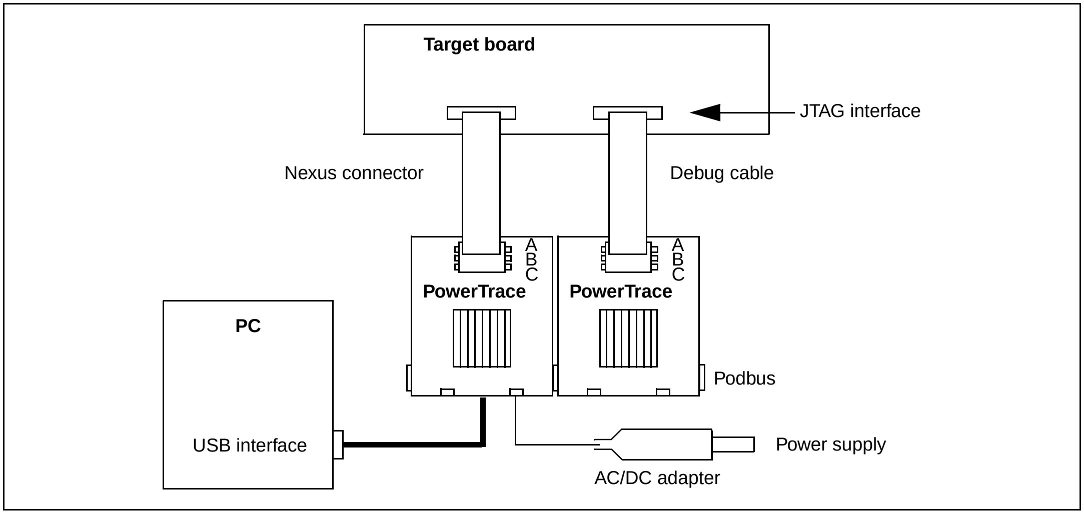


Figure 46. PTd + PTn



Appendix E Glossary

TRACE32-MMDSP	TRACE32 software adapted for MMDSP (SxA)
TRACE32-ARM	TRACE32 software for ARM processors family
PODBUS	High speed serial bus which connects TRACE32 debuggers to the host system
HLL	High level language
JTAG	Joint test action group. A 0standard for test access port and boundary-scan architecture for printed circuit board

Revision history

Table 8. Document revision history

Date	Revision	Changes
10-Jan-2008	A	Initial release.

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.

UNLESS EXPRESSLY APPROVED IN WRITING BY AN AUTHORIZE REPRESENTATIVE OF ST, ST PRODUCTS ARE NOT DESIGNED, AUTHORIZED OR WARRANTED FOR USE IN MILITARY, AIR CRAFT, SPACE, LIFE SAVING, OR LIFE SUSTAINING APPLICATIONS, NOR IN PRODUCTS OR SYSTEMS, WHERE FAILURE OR MALFUNCTION MAY RESULT IN PERSONAL INJURY, DEATH, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE.

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.

Information in this document supersedes and replaces all information previously supplied.

STMicroelectronics group of companies

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

www.st.com