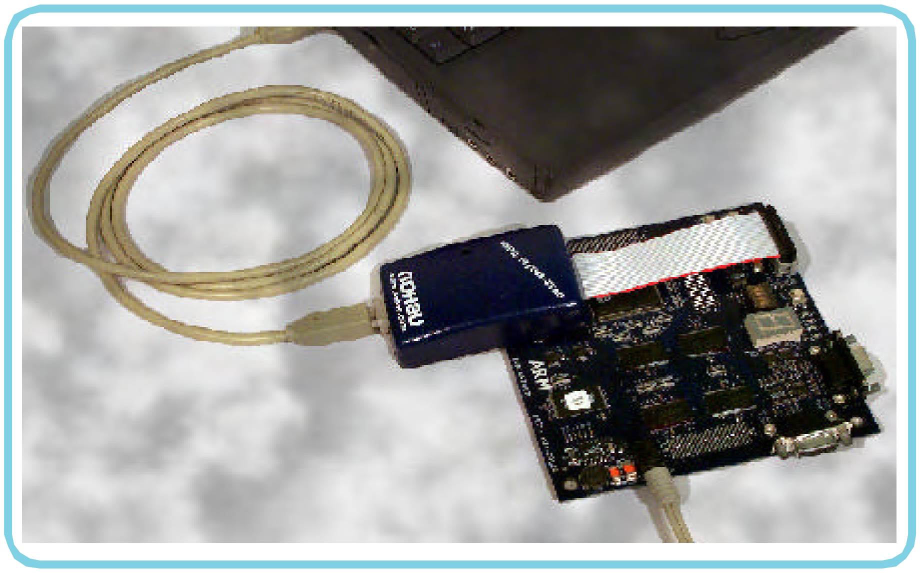
ICE TECHNOLOGY nohau®

EMUEMUL-ARM
Getting Started Guide



EMUL–ARM™

Getting Started Guide

© ICE Technology. All rights reserved worldwide.

Table of Contents

Warranty Information

The debugger hardware is sold with a six-month warranty starting from the date of purchase. Any defective component under warranty will either be repaired or replaced at ICE Technology's discretion.

Seehau software is sold with no warranty, however upgrades can currently be obtained from the ICE Technology site: http://www.icetech.com.

ICE Techology makes no other warranties, express or implied, including, but not limited to the implied warranties of merchantability and fitness for a particular purpose. In no event will ICE Technology be liable for consequential damages.

Warnings

- To avoid damage to the debugger hardware or to your target, do not connect the debugger hardware to your target when the target power is on.
- Do not apply power to your system unless you are sure the target connector is correctly oriented. Failing to do so can cause damage to your target.

About This Guide

The EMUL–ARM is a PC-based hardware debugger for the ARM™ Core, available for the ARM7TDMI, ARM710T, ARM720T, ARM922, ARM940T, ARM9ES and ARM9x6. This guide helps you to get started with the basics of setting up, configuring, and running the Seehau software and debugger.

The *EMUL–ARM Getting Started Guide* is intended for both novice and advanced users. This guide introduces the following tasks:

- Installing and configuring the Seehau software
- Installing the debugger hardware
- Starting the hardware and Seehau software
- Shutting down Seehau

To open an electronic version of this guide, do the following:

1. From the Start menu, select **Programs** and then select **Documentation**

2. Click on **Getting Started** on the left

1 OVERVIEW OF THE EMUL–ARM EMULATOR SYSTEM

Software

The debugger is configured and operated by the SeehauARM user interface. Seehau is a high-level language user interface that allows you to perform many useful tasks, for example:

- Editing code in the Source Window and building projects using the Nohau Project Manager.
- Loading code, running, and stepping programs. Additional documentation for programming the ARM core is available from ARM. Contact support@icetech.com Technical Support if you have questions.
- Modifying and viewing memory contents including general-purpose registers.
- Setting multiple breakpoints.
- Displaying program flow with the optional trace module.

System Requirements

CAUTION

Like all Windows applications, the Seehau software requires a minimum amount of free operating system resources. The recommended amount is at least 40%. (This is only a guideline. This percentage might vary depending on your PC.) If your resources are dangerously low, Seehau might become slow, unresponsive or even unstable. If you encounter any of these conditions, check your free resources. If they are below 40%, reboot and limit the number of concurrently running applications. If you are unable to free at least 40% of your operating system resources, contact your system administrator or Nohau Technical Support at support@icetech.com.

The following are minimum system requirements:

- Pentium 200 (Pentium II or faster is recommended)
- USB Port
- Windows 98 / ME, or 2000 PRO / XP
- Random Access Memory (RAM)
  - For Windows 98: 64 MB
  - For Windows 2000 / ME / XP: 128 MB

Hardware

The basic hardware for the EMUL–ARM debugger system is the EMUL–PC/ USB–JTAG. At present, EMUL-ARM supports the ARM7 and ARM9 families.

Refer to Chapter 3, “Installing the Hardware” for a detailed hardware information overview.

European CE Requirements

We have included the following information in order to comply with European CE requirements.

User Responsibility

The in-circuit debugger application, as well as all other unprotected circuits, need special mitigation to ensure Electromagnetic Compatibility (EMC).

The user has the responsibility to take required measures in the environment to prevent other activities from disturbances to the debugger application according to the user and installation manual.

If the debugger is used in a harsh environment (field service applications for example), it is the user’s responsibility to control other activities so that there is no risk for personal hazard/injuries.

Special Measures for Electromagnetic Emission Requirements

To reduce the disturbances to meet conducted emission requirements, it is necessary to place a ground plane on the surface under the JTAG cable and the connected processor board. The ground plane shall have a low impedance ground connection to the host computer frame. The insulation sheet between the ground plane and circuit boards shall not exceed 1mm of thickness.

Documentation

Electronic copies of the documentation are available. Go to the Start menu and select Programs, then Documentation. This will open an HTML page, which is a guide to the documentation.

You can also access electronic copies of the documentation with Windows Explorer by browsing to the directory: <*install drive*>\Nohau\SeehauARM\Documents.

2 INSTALLING THE SEEHAU SOFTWARE

Installing the Seehau Software From the CD

To install the Seehau software, do the following:

1. Insert the Seehau CD into your CD ROM drive. The installation process will start automatically.

2. Click **Install Seehau Interface for EMUL-ARM** and follow the instructions that appear on your screen.

If the installation does not start automatically, you might have your Windows Autorun feature disabled. You will then need to do one of three things:

- Use Windows Explorer and navigate to the CD root directory. Double-click **Autorun.exe.** The Windows Install Shield will start the installation process.
- Right-click on the CD ROM symbol while running Windows Explorer and select **AutoPlay** to start the installation process.
- From the taskbar, select **Start**, then **Settings**. Click on **Control Panel,** then **Add/Remove Programs,** and then **Install.** The installation process will start when you select the correct path to the CD.

Downloading and Installing the Seehau Software From the Internet

1. Go to the Nohau web site (http://www.icetech.com). Click **Downloads**. The Nohau Software Downloads page opens.
2. Click **Current Seehau Software**. The Seehau Software Status page opens.
3. Locate the EMUL–ARM product listing. There will be two listings: one for documentation and one for software.
4. Click **Information and Download (Seehau).**
5. Review the information on the page.
6. Click **Yes I Want to Download.** A Customer Information Form page opens. Complete this form, then click **Proceed**. (You have the option to download more than one product.) A verification page opens with the information you have just entered. If all information is correct, select **SEND** at the bottom of the page. A message will open that verifies your information has been sent.
7. Click **Go to Download**. The **Available Download Areas** page opens.
8. Click either option for a download site. The Nohau Software Updates page opens.
9. Click the EMUL–ARM link.
10. Click the ARM.exe link. The application will start downloading. Make a note of the directory for this downloaded file.
11. Following the download, go to the directory which has the downloaded file. Click the ARM.exe file and follow the installation instructions.

After installing the Seehau software, the **Setup Complete** dialog box appears, allowing you to view the Readme.txt file and/or launch the SeehauARM configuration.

Note

You must launch the SeehauARM configuration before running the Seehau software.

3 INSTALLING THE HARDWARE

USB Driver

When installing the USB device, you must install the Seehau software first, before connecting the Nohau hardware (refer to Chapter 2, “Installing the Seehau Software”). This allows the computer to recognize the proper driver for the hardware.

Note:

Windows 95 or NT does not support the USB option.

The USB drivers are loaded as part of the Seehau software installation. The driver is located in the root directory of the installation CD. After installation, the driver is also located in the SeehauARM subdirectory on your hard drive (C:\Nohau\SeehauARM). The system software should find and load the USB driver without your intervention.

Power Supply

The EMUL–PC/USB–JTAG uses power supplied by the USB cable and the target board. The amount of power used by the USB-JTAG interface from the target board is less than 100 µA. The power supplied by the target board drives the signal from the buffer on the pod through the target connector to the target board. The 20-pin JTAG pod draws power from the target board through Vsupply, which is regulated to the voltage of VTref (see the diagram on the following page). Vsupply and VTref must be between 2.3V and 5.5V. The 14-pin pod behaves similarly.

For Trace, however, the Compact Trace Module (CTM) does not use any power from the target. Also, the CTM operates on 3.3V but can be modified for other voltages.

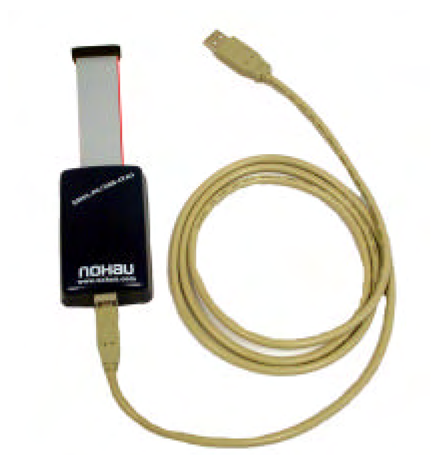
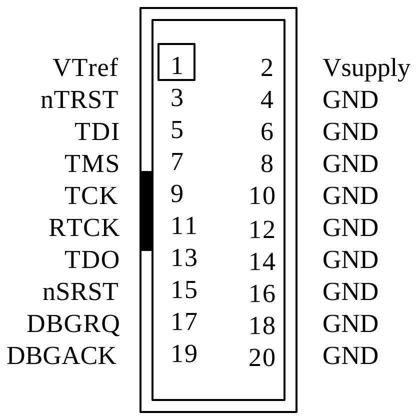


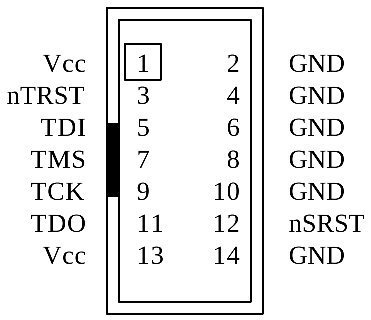
Figure 1: EMUL–PC/USB–JTAG

Note

The EMUL–PC/USB–JTAG is powered from the target. This allows the hardware to operate with a wide range of target voltages.

The following diagram shows the signal layout of the target connector. Pin one is denoted by a square around the pin in the diagram. For detailed descriptions including trace connections, please see the document "ARM_Connections.pdf" which is included on the Seehau CD and is installed into the SeehauARM Documentation folder.





Refer to Figure 2 below while following the installation instructions.

1. Make sure your target is powered off.

2. Plug the host end (larger) of the USB cable into the computer.

3. Plug the 20 or 14-pin EMUL–PC/USB–JTAG serial debug connector onto your target's header. This connector supplies the signals needed to communicate with the EMUL–PC/USB–JTAG.

4. Plug the device side (smaller) of the USB cable into the EMUL-PC/USB-JTAG connector.

5. Power on your target.

Note

Exit Seehau and shut down the power to the target when the target is not in use.

The red wire indicates pin one on the debug connector.

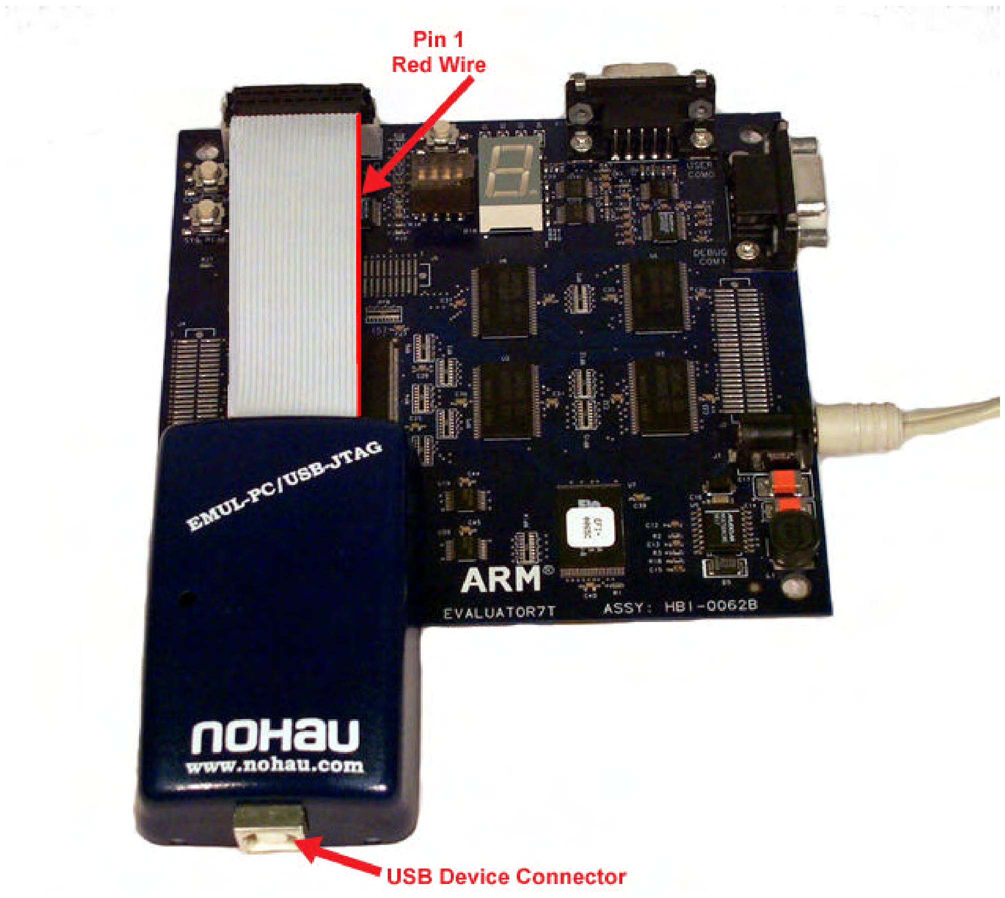


Figure 2: Connections for the USB Cable and the Red Wire Pin 1

Trace Connection

To connect the trace to the target board, make sure the Mictor 38 connector is oriented in the correct direction. The connector is keyed to prevent incorrect insertion. On the Mictor 38 target board receptacle, there are two chamfered corners to allow only the correct connector orientation; the Mictor 38 male connector connects into that receptacle. The same procedure is followed when connecting the other end of the Mictor 38 extension cable to the bottom of the CTM. Refer to Figure 3 for a photo of the correct way to plug the Mictor 38 connector into the target board.

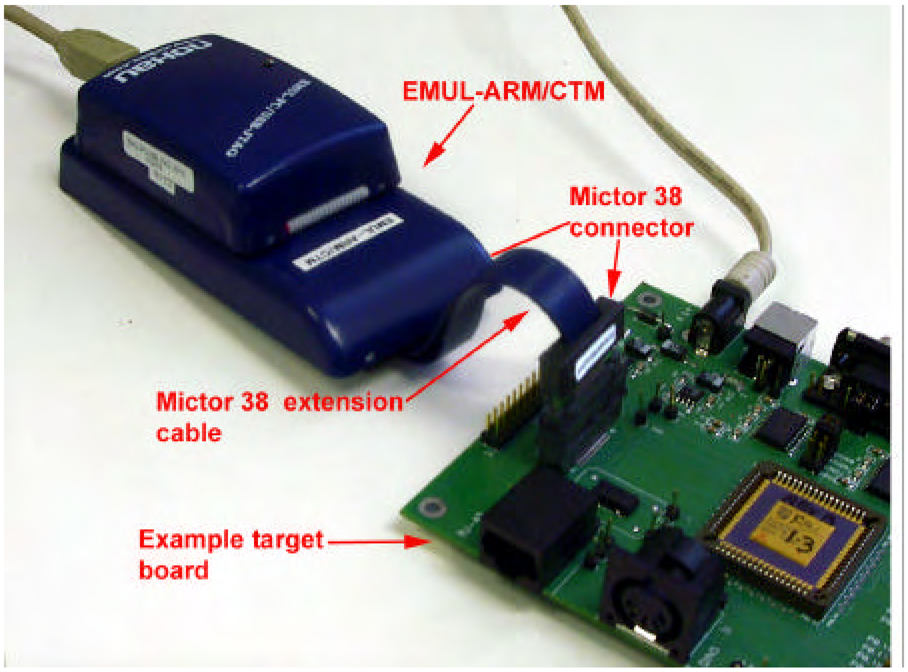


Figure 3: ARM with Trace Connected to the Target Board.

4 CONFIGURING THE SEEHAU SOFTWARE

Choosing to Automatically Start the Seehau Configuration Program

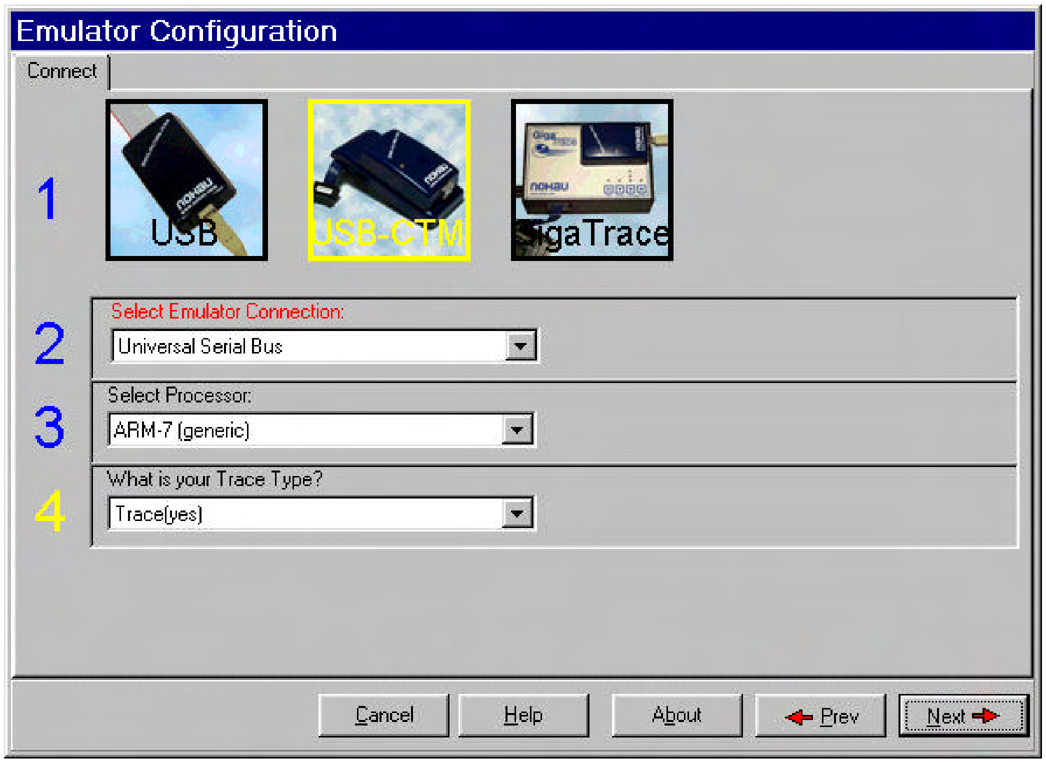
After installing Seehau, it is recommended that you automatically start the Seehau Configuration program. Do the following steps before starting Seehau:

1. From the **Setup Complete** dialog box, select **Launch SeehauARM Configuration**.
2. Click **Finish**.

If you do not choose to automatically start the Seehau Configuration Program, do the following:

From the **Start** menu, select **Programs.**

Select **SeehauARM**. then click **Config** to open the **Emulator Configuration** window displaying the **Connect** tab (Figure 4). Please note that it is not necessary to be connected at this time.



Select Emulator Connection: Universal Serial Bus

Select Processor: ARM-7 (generic)

What is your Trace Type? Trace(yes)

Figure 4: Emulator Configuration Window Displaying the Connect Tab

Configuring the Communications Interface

Connect Tab

The graphical user interface for this tab is divided into four regions:

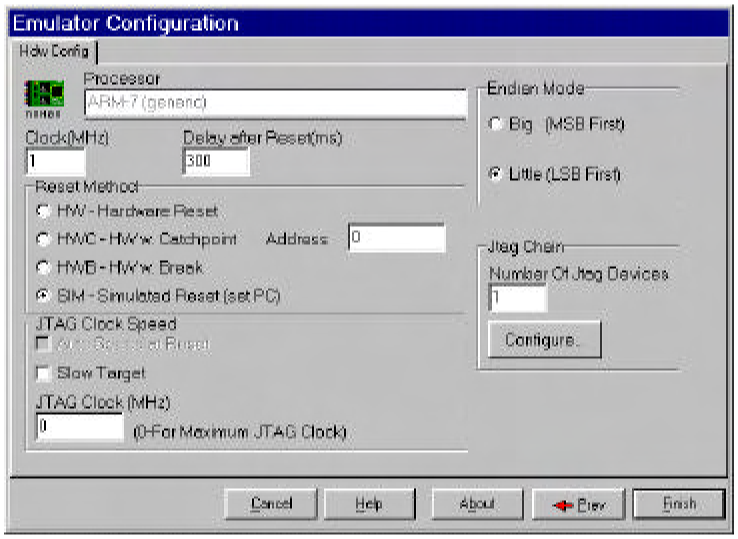
1. Region 1 – **Communications Interface:**
   Displays the three communications interface options for the ARM. The USB is for the pod only. If you have the optional Compact Trace Module (CTM), choose the USB-CTM (in the middle). The third box is for the GigaTrace option.

2. Region 2 – **Select Emulator Connection**:
   No action required. Default is **Universal Serial Bus.**

3. Region 3 – **Select Processor**:
   Default **is ARM7 (generic).** Click on the arrow to view the drop down list for the other devices. For **ARM9,** select **ARM9 (generic).**

4. Region 4 – **What is your Trace Type?:**
   The default is **Yes** if trace is supported (based on earlier selections).

5. Click **Next**. The **Hdw Config** tab opens (Figure 5).



Processor: ARM-7 (generic)

Clock(MHz): 1

Delay after Reset(ms): 300

Reset Method
- HW - Hardware Reset
- HWC - HW w. Catchpoint   Address: 0
- HWB - HW w. Break
- [✓ SIM - Simulated Reset (set PC)]

JTAG Clock Speed
- ☐ Auto Speed at Reset
- ☐ Slow Target

JTAG Clock (MHz): 0 (0-For Maximum JTAG Clock)

Endian Mode
- Big (MSB First)
- [✓ Little (LSB First)]

Jtag Chain
Number Of Jtag Devices: 1
Configure..

Figure 5: Emulator Configuration Window Displaying the Hdw Config Tab

Hardware Configuration Tab

The initial configuration settings are read from the Startup.bas file when the debugger initializes. After the initial startup and configuration of Seehau, the configuration parameters are saved in the Startup.bas file (located in the Macro subdirectory). The next time the emulator is started, the configuration parameters are read from the Startup.bas file and compared with the default parameters on the chip.

- **Processor:**
  Shown for reference only. This is the selected processor. If you need to change the processor, click **Prev**.

- **Clock(MHz):**
  This is used for the trace timestamp only. Enables you to set the clock speed.

- **Delay after Reset (ms):**
  Used to set the time to wait after reset before accessing the hardware. This allows the target to complete reset operations such as loading the logic into an FPGA.

- **Remap:**
  This is used in the ARM chip for remapping the starting address of RAM. When this is selected, the remap bit will be set at reset. On devices that do not provide this, it is not set and will therefore be grayed out.

- **Reset Method**
  - **HW – Hardware Reset:**
    Sets a hardware break at address 0 and asserts reset.
  - **HWC – HW w Catchpoint:**
    Inserts a jump to itself at the address specified.
  - **HWB – HW wBreak:**
    This reset method asserts reset, waits for the set time as determined by the Delay after Reset, then stops execution.
  - **SIM – Simulated Reset (set PC):**
    This reset method sets the Program Counter to 0 and does not assert hardware reset.

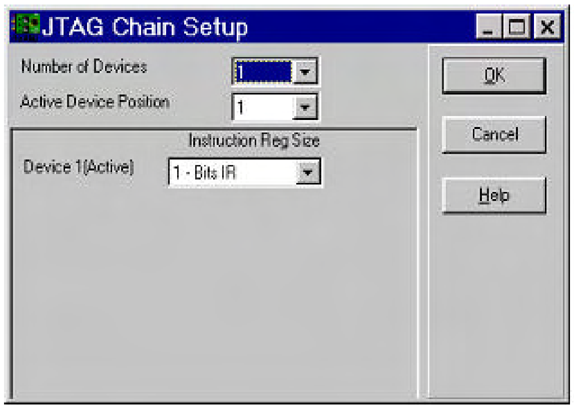
Note

For a more detailed description of the Reset Method, refer to the Reset Method document called ARM_Reset located in the document folder.

- **JTAG Clock Speed**
  - **Auto Speed at Reset:**
    Allows EMUL-ARM to bring the target up to full speed automatically. Not implemented for all MCUs.
  - **Slow Target:**
    A special slow target communication mode is entered. Do not use this unless really necessary, since it will slow down target communication significantly. This does not affect the JTAG clock.

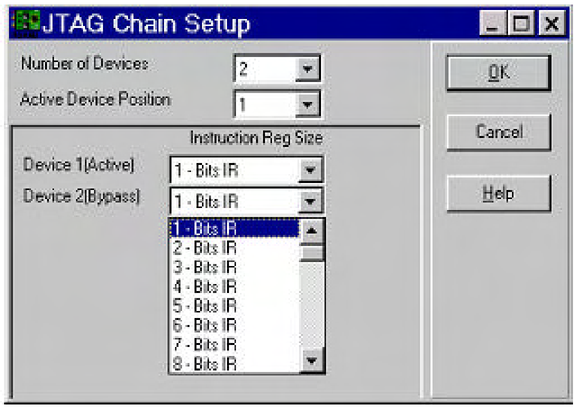
– **JTAG Clock (KHz):**
Sets the JTAG clock speed.

- **Endian Mode:** This is used to configure the endianess -- byte ordering -- for your chip.
  - **Big (MSB First):**
    Select this if your target has the Most Significant Byte (MSB) first.
  - **Little (LSB First):**
    Select this if your target has the Least Significant Byte (LSB) first.



Number of Devices: 1
Active Device Position: 1
Device 1(Active): 1 - Bits IR

Figure 6: JTAG Chain Setup Screen.



Number of Devices: 2
Active Device Position: 1
Device 1(Active): 1 - Bits IR
Device 2(Bypass): 1 - Bits IR

1 - Bits IR
2 - Bits IR
3 - Bits IR
4 - Bits IR
5 - Bits IR
6 - Bits IR
7 - Bits IR
8 - Bits IR

Figure 7: Setup Screen with Two Devices

- **JTAG Chain** configuration is used if the ARM device is not the only chip connected to the JTAG, or if the chip has multiple JTAG controllers (Figure 5).
  - **Number of JTAG devices:**
    This is the total number of devices connected to the JTAG Scan Chain. If only one device is present, nothing needs to be configured.
  - **Configure:**
    This button is used to configure the number of devices and the Instruction Length for each device. This is only available during the intial configuration. If you click the **Configure** button, Figure 5 or Figure 6 appears depending on the number of devices that were entered.
  - **Number of Devices:**
    This is used to set the total number of devices in the JTAG Scan Chain (Figure 6).
  - **Active Device Position**:
    This is used to set which device in the ARM chip the JTAG will connect to.
- **Instruction Reg Size:** There will be a drop down list for each of the devices. If the number of devices is set to more than one, there will be a drop down list for each device (Figure 6).
  - **Device1 (Active):**
    The Instruction Register size for the active device, which is automatically configured. (ARM7 = 4, ARM9 = 5)
  - **Device2(Bypass):**
    This is where the Instruction Register Length is set for the device in the chain that will be bypassed.

When you click **Finish**, Seehau starts to load. For more information about starting Seehau, see the "Starting Seehau" section at the end of this chapter.

Configuring the Emulator Options From Within Seehau

In Seehau, open the Emulator Configuration window. From the **Config** menu select **Emulator**. The Emulator Configuration window opens displaying the **Hdw Config** tab (Figure 8).

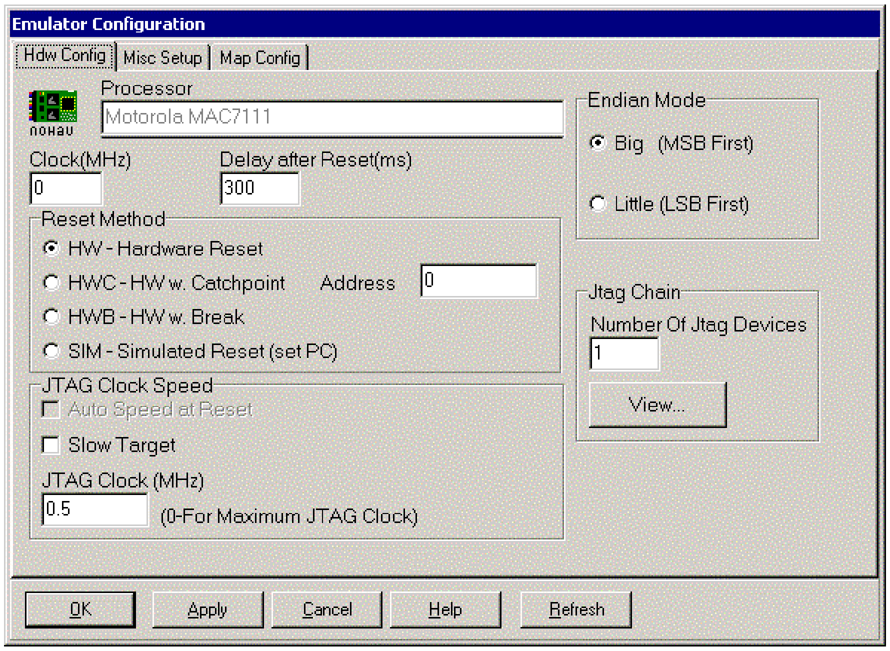
Only the differences from the initial configuration will be discussed here.

The **Emulator Configuration** window contains three tabs. When selected, each tab allows you to set the following options:

Hdw Cfg:	Set up emulator hardware options.
Misc Setup:	Select reset options.
Map Config	Used to manually set the address ranges where Thumb code is stored.

Buttons Common to All Tabs

- **OK:**
  Saves the settings for the tab and exits the dialog box.
- **Apply:**
  Saves the settings for the tab.
- **Cancel:**
  Exits without saving the settings for the dialog box.
- **Help:**
  Displays the Seehau Help file.
- **Refresh:**
  Allows you to retrieve and view the current emulator hardware configuration settings.



Processor: Motorola MAC7111
Clock(MHz): 0
Delay after Reset(ms): 300
Reset Method:
[✓ HW - Hardware Reset]
HWC - HW w. Catchpoint   Address: 0
HWB - HW w. Break
SIM - Simulated Reset (set PC)
JTAG Clock Speed:
☐ Auto Speed at Reset
☐ Slow Target
JTAG Clock (MHz): 0.5 (0-For Maximum JTAG Clock)
Endian Mode:
[✓ Big (MSB First)]
Little (LSB First)
Jtag Chain:
Number Of Jtag Devices: 1

Figure 8: Emulator Configuration Window Displaying the Hdw Config Tab

Hardware Configuration Tab

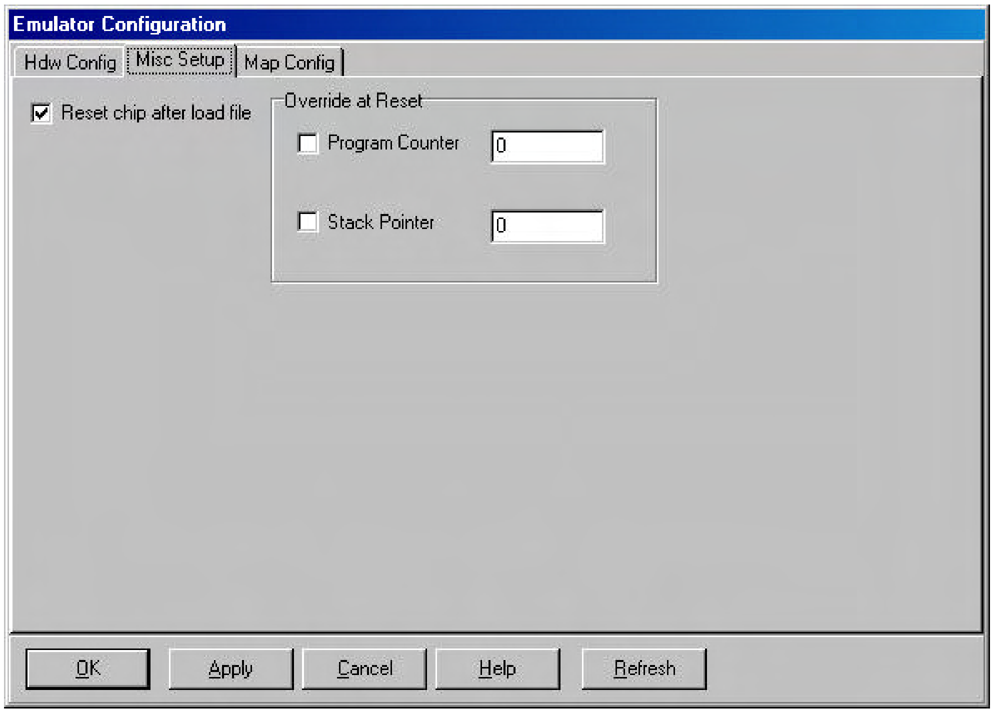
- **JTAG Chain**
  - The **View** button is used to check the current configuration. The configuration can only be changed during the initial configuration.

Miscellaneous Setup Tab

The **Misc Setup** tab (Figure 9) is accessible only after the initial software configuration.

- **Reset chip after load file:**
  Sets the ARM core to issue a reset after the code is loaded.

- **Override at Reset**
  - The **Program Counter** option selects the value that the program counter will be set to after the reset sequence has completed. Enter the program counter value in the box.
  - The **Stack Pointer** option selects the value that the stack pointer will be set to after a reset. Enter the stack pointer value in the box.



☑ Reset chip after load file

Override at Reset

☐ Program Counter 0

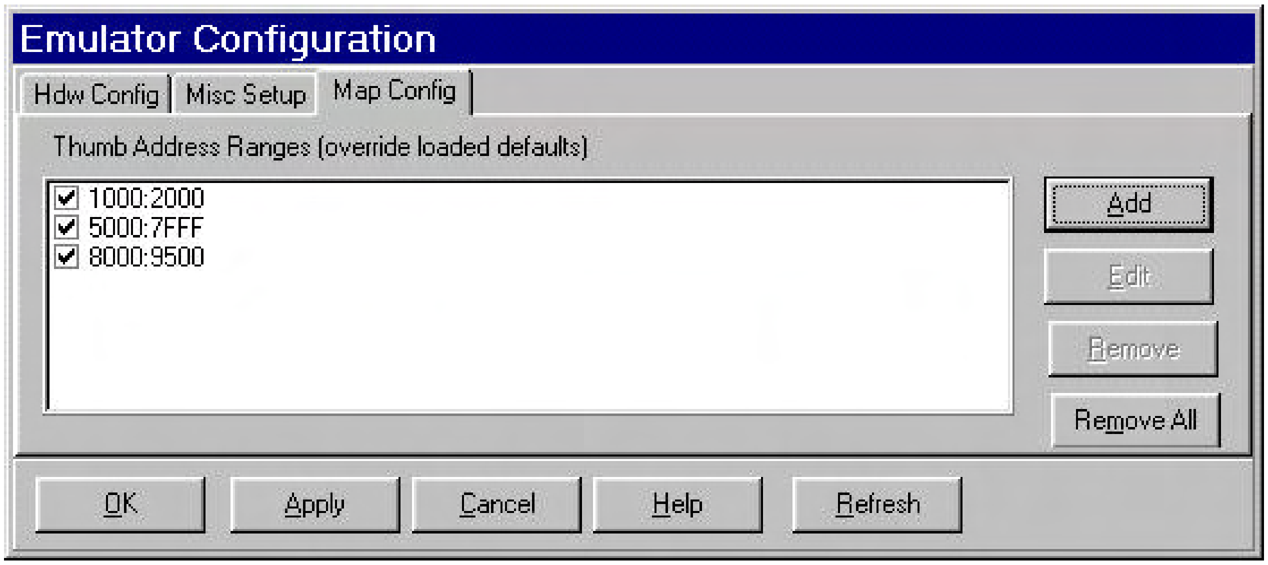
☐ Stack Pointer 0

Figure 9: Emulator Configuration Window Displaying the Misc Setup Tab

MAP CONFIG TAB

Figure 10 is accessible only after the initial software configuration. The file loader will automatically map THUMB regions defined in the output file. This mapping window provides a manual override for THUMB regions. The large open area displays the address ranges that contain THUMB code.

- The **Add** button is used to add a range of memory that is used for THUMB code.
- The **Edit** button is available only when a range is selected. The range values may be changed.
- The **Remove** button is used to delete one memory range from the table.
- The **Remove All** button is used to delete all the memory ranges from the table.



Thumb Address Ranges (override loaded defaults)

☑ 1000:2000
☑ 5000:7FFF
☑ 8000:9500

Figure 10: Emulator Configuration Window Displaying the Map Config Tab

Starting Seehau

After you configure the Seehau software, you have the option of starting Seehau. The Non-Demo Mode assumes you have an emulator and target connected to your PC. The Demo Mode is not a simulator, but allows a tour of the Seehau user interface.

Non-Demo Mode

To start Seehau in Non-Demo mode, do the following:

1. Double-click the SeehauARM icon. The Seehau main window opens (Figure 11). Seehau will load its configuration from the Startup.bas file. Notice that the word "macro" is displayed in red at the bottom of the main window while Startup.bas is running.

2. When the software startup is complete, you can position and resize the main window to your preference. At this time, you will need to load code.

3. To open new windows, select the window you want to open from the **New** menu.

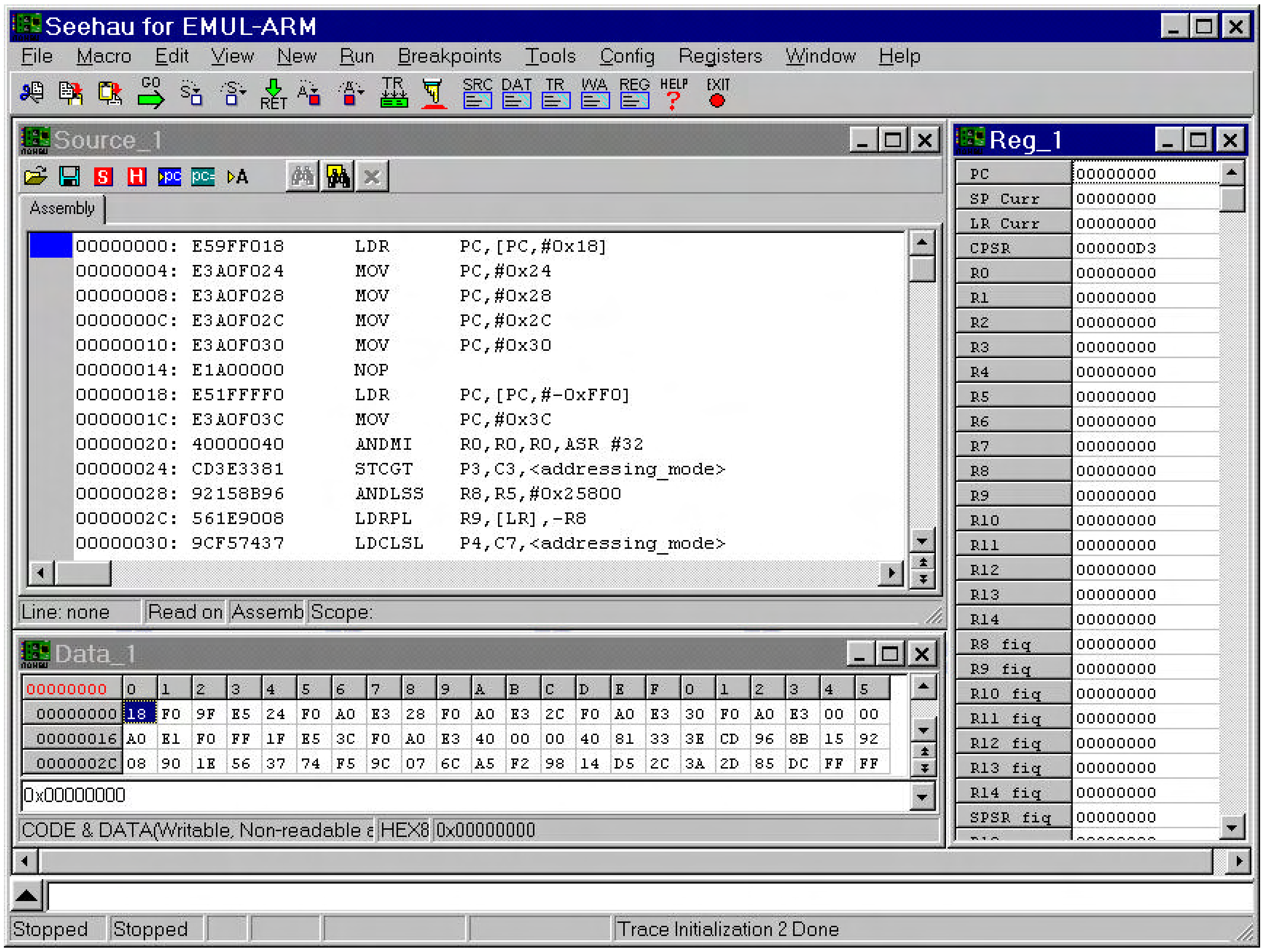
Demo Mode

To start Seehau in Demo mode, do the following:

1. Double-click the Demo icon. After Seehau is loaded, a box with the message "**This is a Demo version of Seehau. This Demo is not a simulator"** opens. The Seehau main window opens (Figure 11). Seehau will load its configuration from the Startup.bas file. Notice that the word "macro" is displayed in red at the bottom of the main window while Startup.bas is running.

2. When the software startup is complete, you can position and resize the main window to your preference. At this time, you will need to load code.

3. To open new windows, select the window you want to open from the **New** menu.


Seehau for EMUL-ARM
File Macro Edit View New Run Breakpoints Tools Config Registers Window Help
GO RET TR SRC DAT TR WA REG HELP EXIT
Source_1
pc pc= A
Assembly
Data_1
Reg_1

```
00000000: E59FF018     LDR      PC,[PC,#0x18]
00000004: E3A0F024     MOV      PC,#0x24
00000008: E3A0F028     MOV      PC,#0x28
0000000C: E3A0F02C     MOV      PC,#0x2C
00000010: E3A0F030     MOV      PC,#0x30
00000014: E1A00000     NOP
00000018: E51FFFF0     LDR      PC,[PC,#-0xFF0]
0000001C: E3A0F03C     MOV      PC,#0x3C
00000020: 40000040     ANDMI    R0,R0,R0,ASR #32
00000024: CD3E3381     STCGT    P3,C3,<addressing_mode>
00000028: 92158B96     ANDLSS   R8,R5,#0x25800
0000002C: 561E9008     LDRPL    R9,[LR],-R8
00000030: 9CF57437     LDCLSL   P4,C7,<addressing_mode>
```

Line: none | Read on | Assemb | Scope:

```
00000000  0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F  0  1  2  3  4  5
00000000  18 F0 9F E5 24 F0 A0 E3 28 F0 A0 E3 2C F0 A0 E3 30 F0 A0 E3 00 00
00000016  A0 E1 F0 FF 1F E5 3C F0 A0 E3 40 00 00 40 81 33 3E CD 96 8B 15 92
0000002C  08 90 1E 56 37 74 F5 9C 07 6C A5 F2 98 14 D5 2C 3A 2D 85 DC FF FF
```

0x00000000

CODE & DATA(Writable, Non-readable a | HEX8 | 0x00000000

PC	00000000
SP Curr	00000000
LR Curr	00000000
CPSR	000000D3
R0	00000000
R1	00000000
R2	00000000
R3	00000000
R4	00000000
R5	00000000
R6	00000000
R7	00000000
R8	00000000
R9	00000000
R10	00000000
R11	00000000
R12	00000000
R13	00000000
R14	00000000
R8 fiq	00000000
R9 fiq	00000000
R10 fiq	00000000
R11 fiq	00000000
R12 fiq	00000000
R13 fiq	00000000
R14 fiq	00000000
SPSR fiq	00000000

Stopped | Stopped | Trace Initialization 2 Done

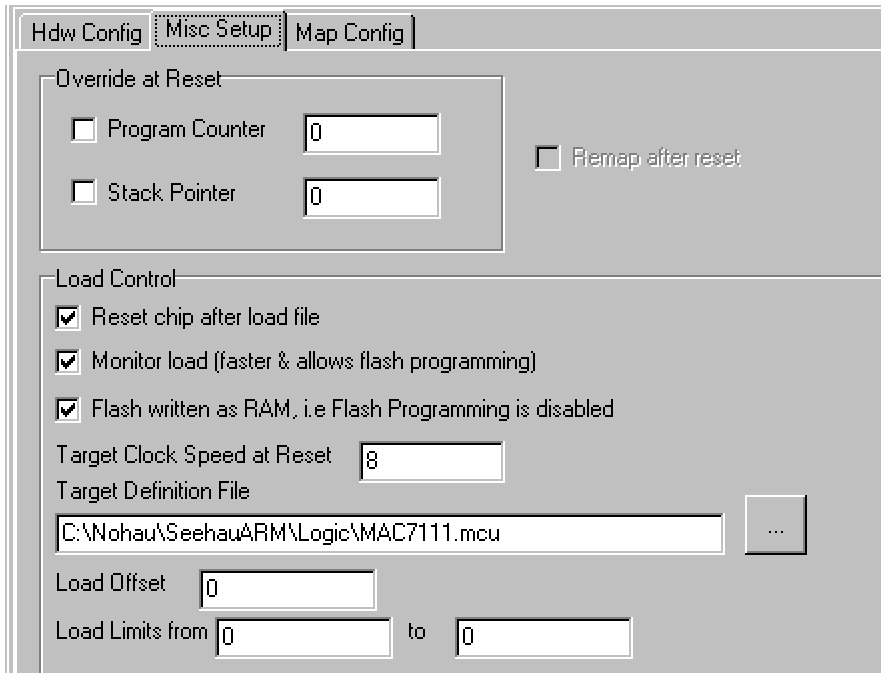
Figure 11: Seehau for EMUL–ARM

Flash Programming / Load Control

EMUL-ARM supports loading and executing programs (and data) into Flash. Conceptually, this is handled as any normal load (into RAM). However, there are some special considerations; most importantly how to define the target. It is currently not possible to write flash from a Data Window.

For many MCUs with internal memory and some popular evaluation boards, support for Flash Programming is built-in. For other boards, you will have to do some work to enable Flash Programming (fully documented in “ARM_Flash.pdf”).

By default, Flash Programming is disabled to prevent erasing the Flash by mistake. It can be enabled and configured using the menu: **Config | Emulator** and the **Misc Setup** tab which has the following (partial) dialog:



- [ ] Program Counter: 0
- [ ] Stack Pointer: 0
- [ ] Remap after reset
- [✓] Reset chip after load file
- [✓] Monitor load (faster & allows flash programming)
- [✓] Flash written as RAM, i.e Flash Programming is disabled

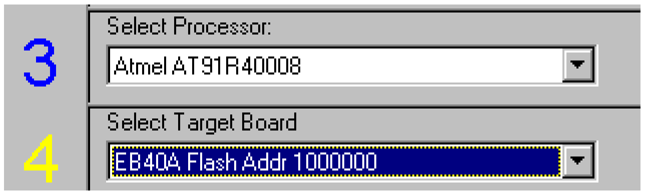
Target Clock Speed at Reset: 8
Target Definition File: C:\Nohau\SeehauARM\Logic\MAC7111.mcu
Load Offset: 0
Load Limits from: 0 to: 0

How to Quickly Enable Flash programming:

- Make sure there already is a “Target Definition File”. Otherwise, continue as follows:
- Uncheck “Flash Written as RAM”.
- Load a file using **File | Load** – this will cause the Flash to be erased and written to.

The **Load Control** section controls Flash Programming and other issues related to loading.

First, check the section showing the "Target Definition File" – without a file name, there can be no Flash Programming. For supported MCUs with internal Flash, it is automatically set up; for selected target boards, it is set up during the initial configuration after selecting the processor.



Select Processor: Atmel AT91R40008

Select Target Board: EB40A Flash Addr 1000000

If the "Target Definition File" is not automatically set up, you will need to consult the document entitled ARM_Flash.pdf for additional information.

- **Reset chip after load file**
  When checked, the debugger will issue a reset after loading a file. Note that when using "Simulated Reset", the reset signal will not be asserted. When this option is not checked, the PC keeps it value from before the load.

- **Monitor load**
  When checked, EMUL-ARM will use a monitor that executes in the target during load to increase download speed and implement Flash Programming.

- **Flash written as RAM**
  When checked, Flash Programming is disabled. This option allows for two things:

  – Disabling Flash Programming in general;

  – Switch back and forth between using and not using Flash Programming (for boards that have switches for one or the other, and when using special debug hardware with RAM where there normally would be Flash, without having to change much configuration).

- **Target Clock Speed at Reset**
  For some MCUs with internal flash, it is required that the MCU clock frequency is known (currently Philips LPC21xx and MAC71xx series). Enter the frequency in MHz here.

- **Target Definition File**
  This file defines where the target has Flash and RAM, and what sort of Flash memories are used.

- **Load Offset**
  A positive or negative offset that is added to the address of data to be loaded. Does not affect addresses of symbols.

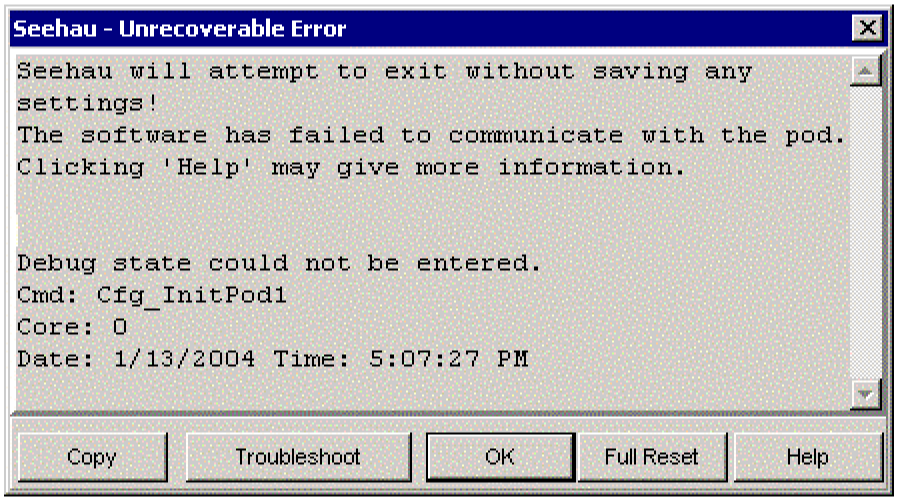
- **Load Limits**
  A low and high address defining limits for what is loaded. Data/code outside the limits is not loaded.

Erasing the Flash

During a load, Flash sectors are erased as needed. For example, if we are loading into a sector, it will be erased automatically. Additionally, Seehau allows erasing Flash manually, using the menu **Tools | Flash Programming**.

Troubleshooting

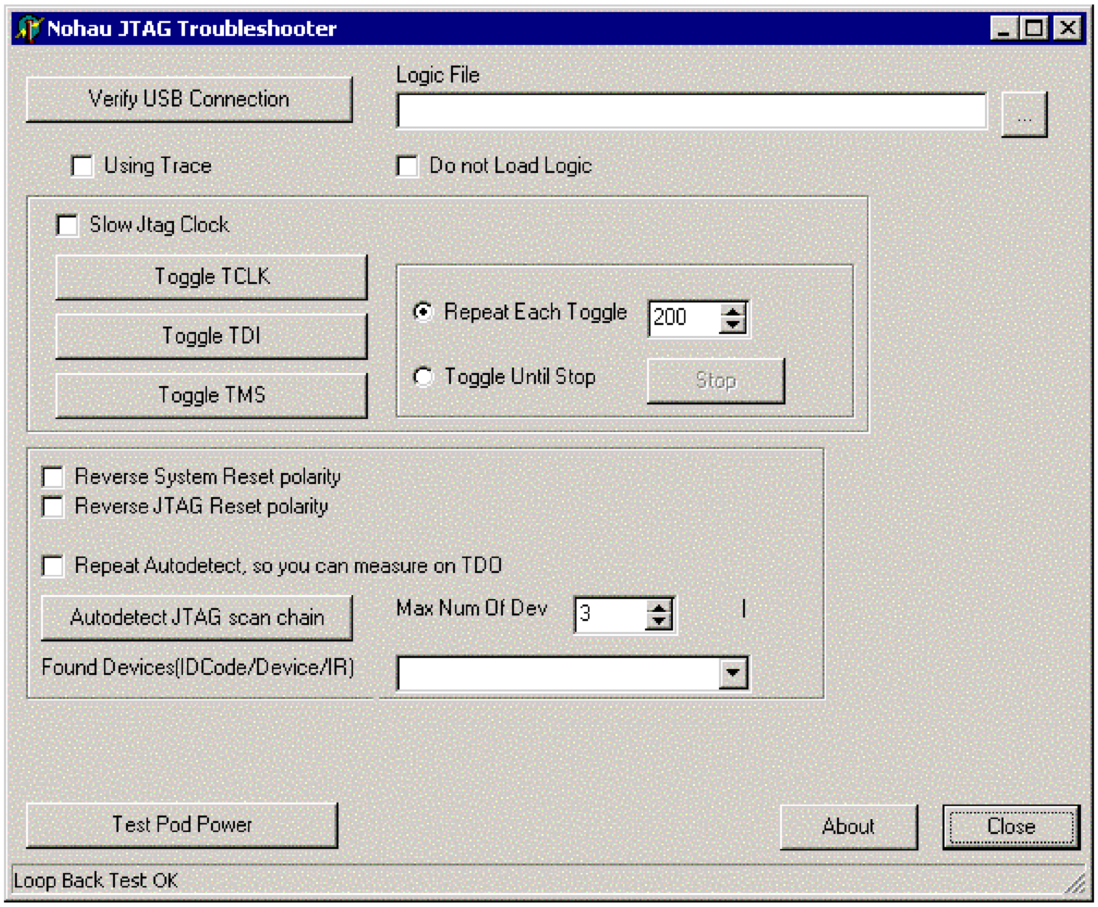
If there is a problem with target communication, an error message like this one will appear.



```
Seehau will attempt to exit without saving any
settings!
The software has failed to communicate with the pod.
Clicking 'Help' may give more information.


Debug state could not be entered.
Cmd: Cfg_InitPod1
Core: 0
Date: 1/13/2004 Time: 5:07:27 PM
```

Selecting **OK** will close the application; **Full Reset** will reset the target and restart communication. To troubleshoot low-level target communication, press **Troubleshoot** to bring up the window below. (Normally, this does not happen if you have been successfully debugging in the past.)



[✓ Repeat Each Toggle] 200

Max Num Of Dev 3

Loop Back Test OK

5 RUNNING PROGRAM EXAMPLES

We provide several C program examples located at: **C:\Nohau\SeehauARM\Examples\**

Start Seehau by following the instructions in the "Starting Seehau" section in Chapter 4. Then do the following to run the test program:

Loading an Example Program

1. Resize the windows on your screen.

2. From the **File** menu select **Load Code**. The **Open** dialog box opens (Figure 12).

3. Select one of the program example files. For the ARM EVALUATOR7T open the folder called 7Segment_Eval7 and load 7segment.elf.

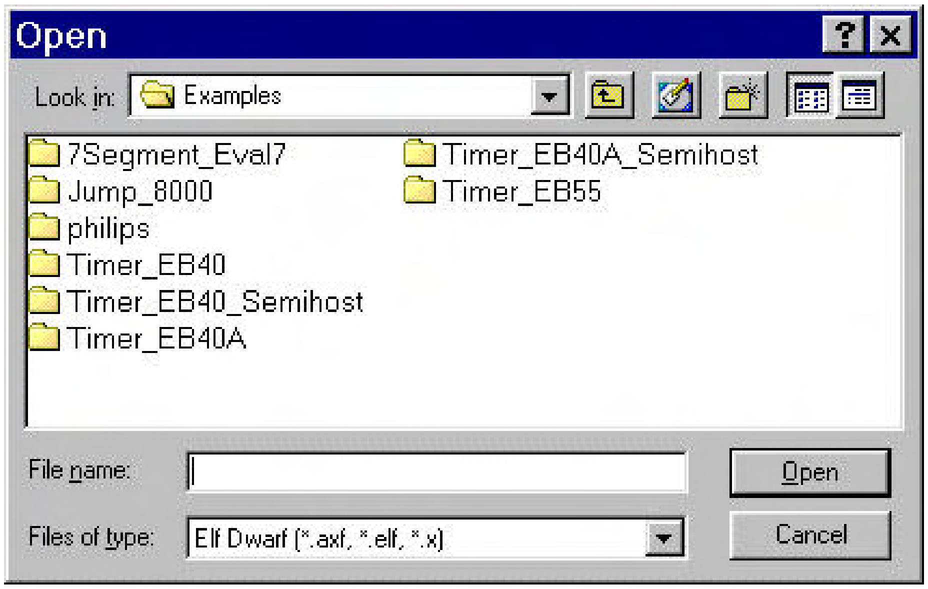
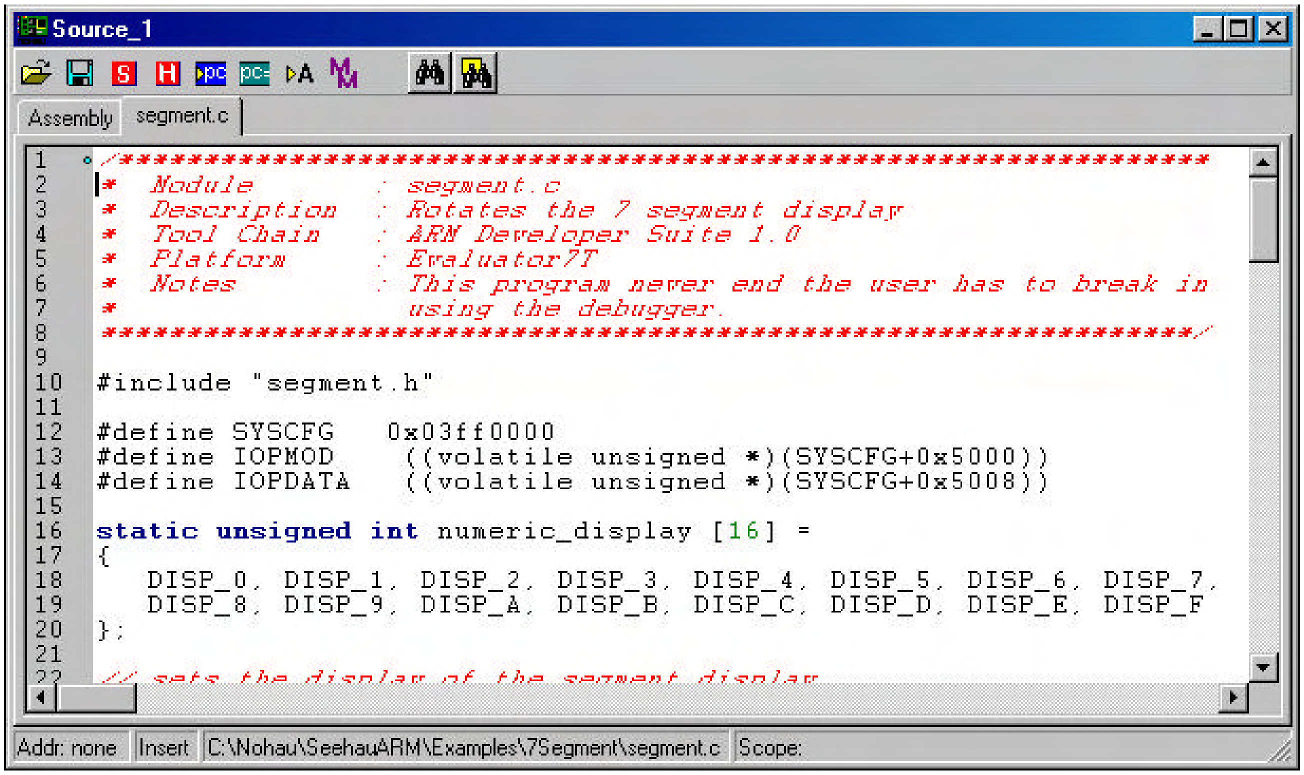


Figure 12: Open Dialog Box Displaying the Program Example Directories



```
/**********************************************************************
*  Module        : segment.c
*  Description   : Rotates the 7 segment display
*  Tool Chain    : ARM Developer Suite 1.0
*  Platform      : Evaluator7T
*  Notes         : This program never end the user has to break in
*                  using the debugger.
**********************************************************************/

#include "segment.h"

#define SYSCFG    0x03ff0000
#define IOPMOD     ((volatile unsigned *)(SYSCFG+0x5000))
#define IOPDATA    ((volatile unsigned *)(SYSCFG+0x5008))

static unsigned int numeric_display [16] =
{
    DISP_0, DISP_1, DISP_2, DISP_3, DISP_4, DISP_5, DISP_6, DISP_7,
    DISP_8, DISP_9, DISP_A, DISP_B, DISP_C, DISP_D, DISP_E, DISP_F
};

// sets the display of the segment display
```

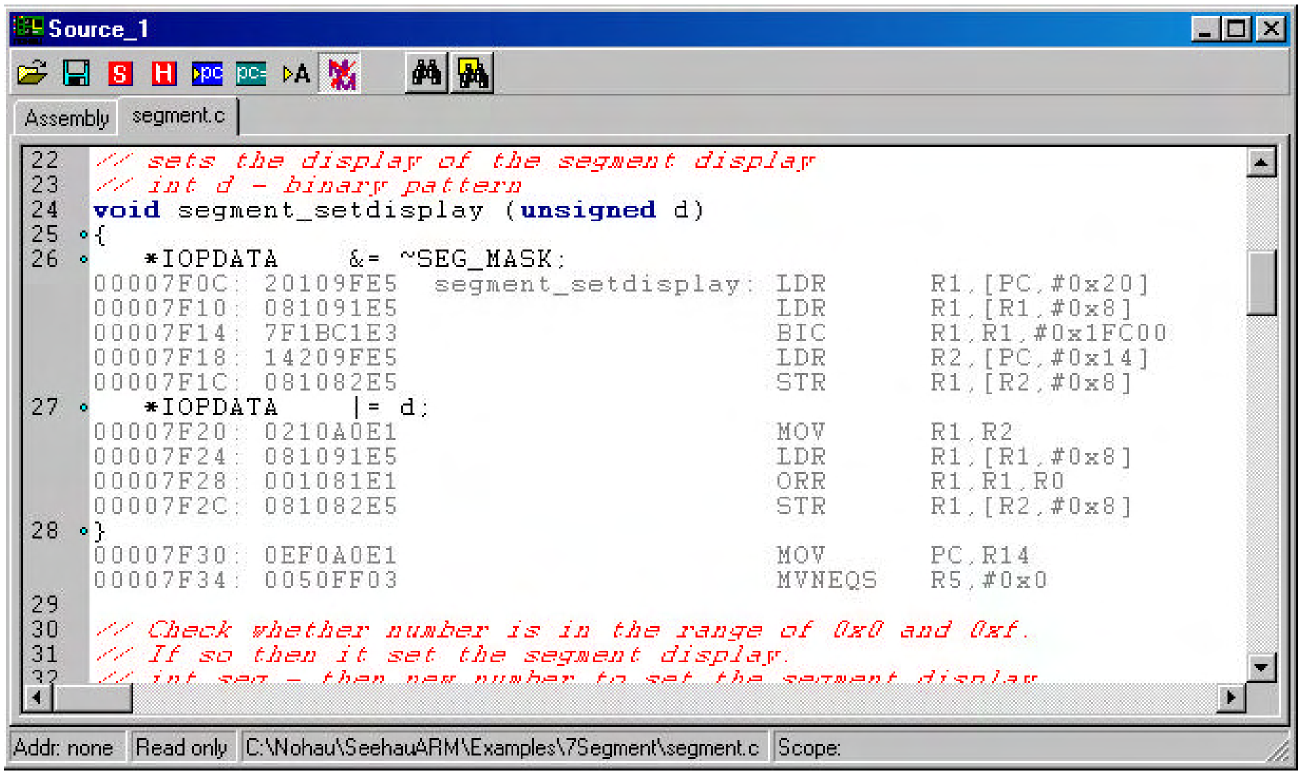
```
Addr: none | Insert | C:\Nohau\SeehauARM\Examples\7Segment\segment.c | Scope:
```

Figure 13: Program Example in Source Only

4. Highlight the program example file and click **Open**. The source file automatically displays the program example in source only (Figure 13).

5. To single-step in mixed mode, click the **Assembly Step Into** or **Step Over** button. You will see the assembly code mixed in with the associated source lines (Figure 14).

6. To single-step in source only mode, first click the **Source** window. Select the **7segment.c** tab. Right-click to verify that mixed mode is cleared. Then point to the **Setting** sub-menu item, and select **Lock Tabs**.



```
// sets the display of the segment display
// int d - binary pattern
void segment_setdisplay (unsigned d)
{
    *IOPDATA     &= ~SEG_MASK;
00007F0C: 20109FE5  segment_setdisplay: LDR      R1,[PC,#0x20]
00007F10: 081091E5                      LDR      R1,[R1,#0x8]
00007F14: 7F1BC1E3                      BIC      R1,R1,#0x1FC00
00007F18: 14209FE5                      LDR      R2,[PC,#0x14]
00007F1C: 081082E5                      STR      R1,[R2,#0x8]
    *IOPDATA     |= d;
00007F20: 0210A0E1                      MOV      R1,R2
00007F24: 081091E5                      LDR      R1,[R1,#0x8]
00007F28: 001081E1                      ORR      R1,R1,R0
00007F2C: 081082E5                      STR      R1,[R2,#0x8]
}
00007F30: 0EF0A0E1                      MOV      PC,R14
00007F34: 0050FF03                      MVNEQS   R5,#0x0

// Check whether number is in the range of 0x0 and 0xf.
// If so then it set the segment display.
// int seg - then new number to set the segment display
```

Figure 14: Program Example in Mixed Mode

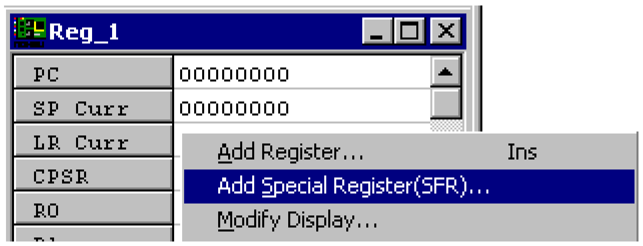
Using SFRs

SFR is an acronym for Special Function Register. For some microprocessor families, this has a precise meaning spelled out in the microprocessor documentation. For other families, it is used very imprecisely, but always refers to registers in the system being emulated.

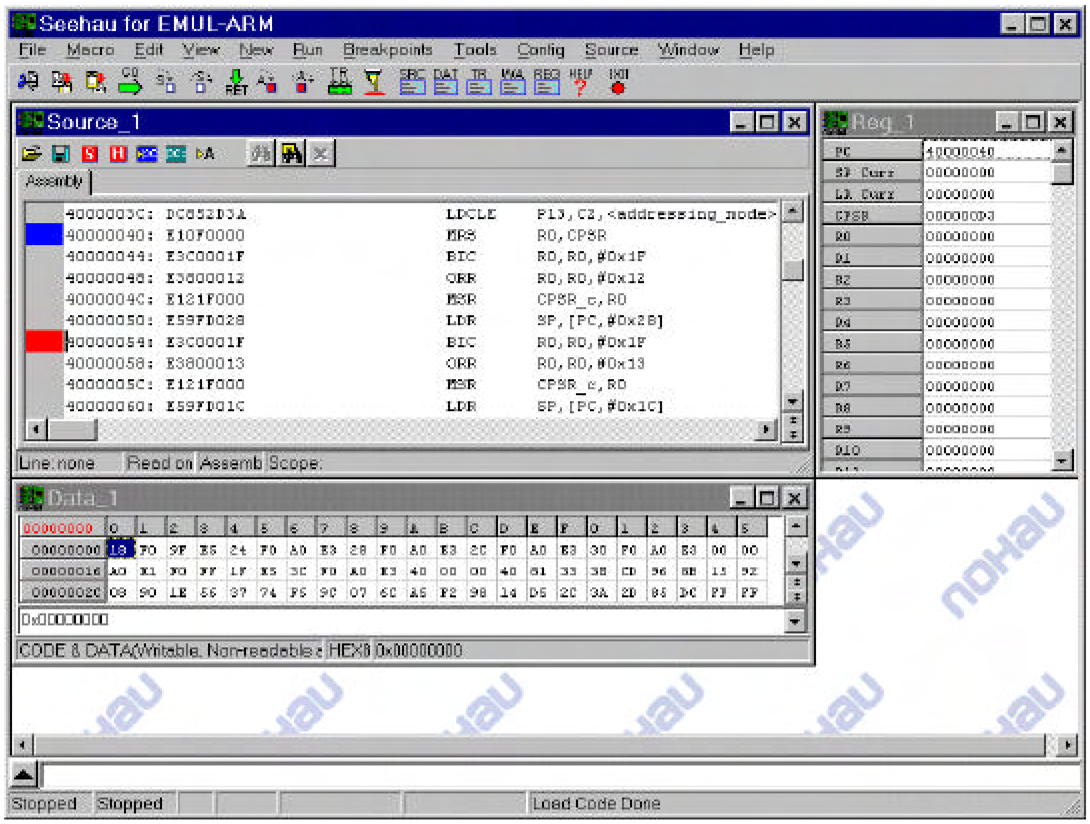
All directly supported MCUs (e.g. selectable during the initial configuration) have built in support for SFRs. For other MCUs, when you configure for “ARM-7 (Generic)”, you need to define SFRs yourself (see “Seehau_RegisterFiles.pdf”). It is also possible to define SFRs for your board.

The **Seehau Register** Window can be modified to show SFRs by:

- Right click the **Register** window, and select **Add SFR**. This will add the SFR to the list of registers in Seehau.
- To make it visible, right click again. This time select **Modify Display** to remove/add registers to show.



Trace Example



```
4000003C: DC852D3A        LDCLE   P13,C2,<addressing_mode>
40000040: E10F0000        MRS     R0,CPSR
40000044: E3C0001F        BIC     R0,R0,#0x1F
40000048: E3800012        ORR     R0,R0,#0x12
4000004C: E121F000        MSR     CPSR_c,R0
40000050: E59FD028        LDR     SP,[PC,#0x28]
40000054: E3C0001F        BIC     R0,R0,#0x1F
40000058: E3800013        ORR     R0,R0,#0x13
4000005C: E121F000        MSR     CPSR_c,R0
40000060: E59FD01C        LDR     SP,[PC,#0x1C]
```

Line:none | Read on | Assemb | Scope:

PC	40000040
SB Curr	00000000
LR Curr	00000000
CPSR	000000D3
R0	00000000
R1	00000000
R2	00000000
R3	00000000
R4	00000000
R5	00000000
R6	00000000
R7	00000000
R8	00000000
R9	00000000
R10	00000000

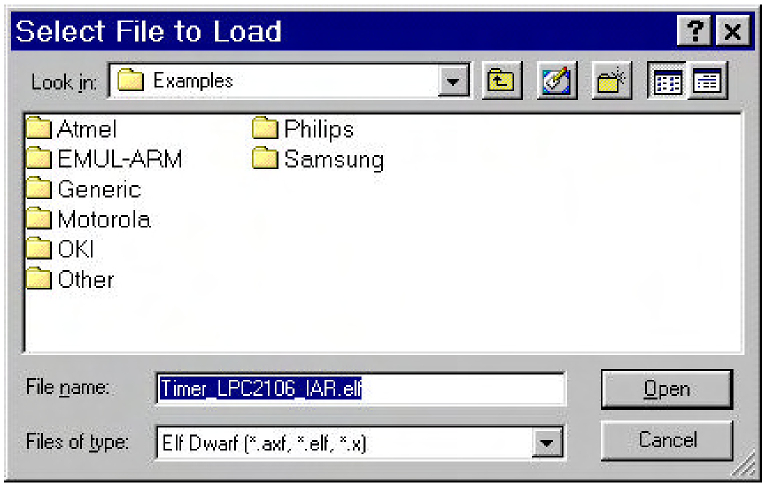
00000000	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	0	1	2	3	4	5
00000000	18	F0	9F	E5	24	F0	A0	E3	28	F0	A0	E3	2C	F0	A0	E3	30	F0	A0	E3	00	00
00000016	A0	E1	F0	FF	1F	E5	3C	F0	A0	E3	40	00	00	40	81	33	38	CD	56	8B	15	92
0000002C	08	90	1E	56	37	74	F6	9C	07	6C	A5	F2	98	14	D5	2C	3A	2D	84	DC	FF	FF

0x00000000

CODE & DATA(Writable, Non-readable | HEX8 | 0x00000000

Figure 15: EMUL-ARM Screen with Trace

To run the trace on the EMUL-ARM, make sure to have the emulator and trace set up and running. See Figure 16 as an example of the screen. For illustration purposes, we have chosen to run the Philips. Go to the **File** menu, click on **Load Code….** In the **Select File to Load** window, click on your trace type. See Figure 16 below. After opening the correct folder, click on the **Timer** folder. Then double click on the correct .elf that matches your trace. We clicked on Timer LPC2106_IAR.elf. You will see the code is now loaded.



Look in: Examples

Atmel
EMUL-ARM
Generic
Motorola
OKI
Other
Philips
Samsung

File name: Timer_LPC2106_IAR.elf

Files of type: Elf Dwarf (*.axf, *.elf, *.x)

Figure 16: Selecting Philips as an Example

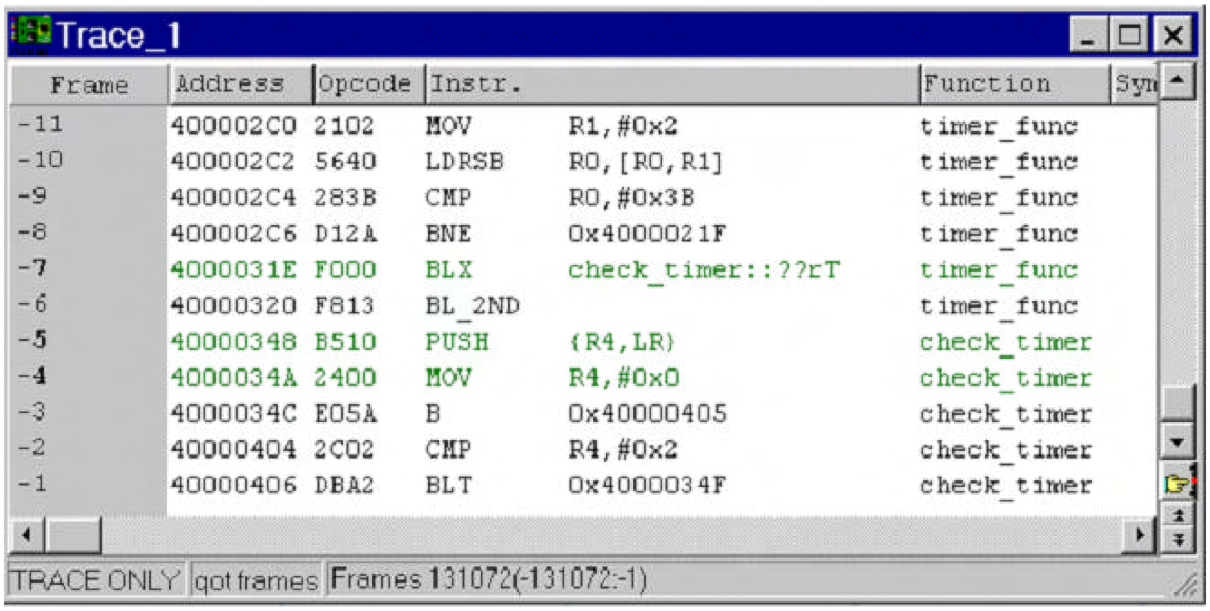
The trace will automatically start when the **GO** button is pressed.

To run the trace without a break, do the following:

Open the **Trace** window by clicking on the **TR** in the main menu bar with the blue window below it. This will open the **Trace_1** window. Next click on the **GO** button on the main menu bar. The user application should now be running. To stop the trace and see all of the recorded instructions, click on the **TR** with the green box below it. This will stop the trace. All of the instructions that were recorded will show up in the **Trace** window.

To run the trace with a break, do the following:

In the **Source_1** window, you will see a grayed out bar on the far left, there will also be a blue box somewhere in the grayed out bar. Click further down on the same grayed out bar (below the blue box) and you will see a red box where you clicked. The points in between the blue and red box are what will be recorded. Next, go to the menu bar and click on the **TR** with the blue window below it to open up the **Trace** window. Click the **GO** button on the menu bar. After a short while, you will see all of the information that was recorded in the **Trace** window. See Figure 17 below.



Frame	Address	Opcode	Instr.		Function	Sym
-11	400002C0	2102	MOV	R1,#0x2	timer_func	
-10	400002C2	5640	LDRSB	R0,[R0,R1]	timer_func	
-9	400002C4	283B	CMP	R0,#0x3B	timer_func	
-8	400002C6	D12A	BNE	0x4000021F	timer_func	
-7	4000031E	F000	BLX	check_timer::??rT	timer_func	
-6	40000320	F813	BL_2ND		timer_func	
-5	40000348	B510	PUSH	{R4,LR}	check_timer	
-4	4000034A	2400	MOV	R4,#0x0	check_timer	
-3	4000034C	E05A	B	0x40000405	check_timer	
-2	40000404	2C02	CMP	R4,#0x2	check_timer	
-1	40000406	DBA2	BLT	0x4000034F	check_timer	

TRACE ONLY | got frames | Frames 131072(-131072:-1)

Figure 17: Setting a Break

6 ARM TRACE

Trace Overview

EMUL-ARM has support for the ARM ETM (Embedded Trace Macrocell).

This document describes the CTM interface.

ARM ETM Trace vs. Nexus Trace

There are currently two different specifications / implementations for the ARM Trace:

- ETM – Embedded Trace Macrocell developed by ARM
- Nexus – A general standard for debug

EMUL-ARM currently supports ARM ETM Trace and NEXUS Trace, with support for ARM Nexus Trace under development.

JTAG Trace

Traditionally, JTAG debuggers have not had the capability to support trace. This is not a design choice by the debugger designer, but instead is determined by the MCU design. Historically, Trace support capability was only available with In-Circuit Emulators which rely on the availability of "bond-out" chips, however there are no bond-out chips for ARM at this time.

To facilitate tracing capabilities, ARM has created the ETM, which adds a logic block inside the MCU and a number of pins (4, 8 or 16 data bits + 4 status bits) that send out the information. The more pins, the more information can be transferred. With fewer pins, and while trying to trace too much information, there will be an overflow internally in the MCU. Regardless of the implementation, the ETM is always capable of tracing program flow if all other options are disabled.

An MCU manufacturer that wants to use the ETM needs to license it from ARM, and then design it in their MCU. This results in licensing fees, additional silicon area and pins. Philips NXP has released their LPC210x MCU with ETM, Atmel has announced a new ARM9-based device with ETM, and there are more devices that are not yet announced.

About Trace Functionality

A trace is an optional part of an emulator system that supplies advanced debugging capabilities that include:

- Storing the execution of instructions so that the history of execution that led up to an error situation can be analyzed.
- Storing only a small portion of the program execution, which can be applied on address range, among others. This function is called **filtering** which allows the user to display only the information that is of interest at any given point (for example, to allow solving a specific software bug in a specific function).
- Setting complex conditions that identify very specific events caused by the microcontroller and which then can stop the trace (triggering).
- Storing timestamp information for every recorded "frame", allowing you to measure how long it takes to execute certain functions.

General Features

The trace size is 128K frames – which translates to approximately the same number of instructions – available in the execution history buffer.

Using Trace

Activating Trace

The Trace functionality is activated by choosing the menu **New | Trace**, or clicking the **Trace** Window button (marked **TR**) on the toolbar. It is possible to open more than one Trace Window.

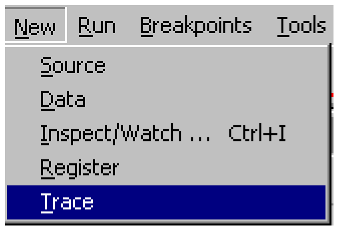


Figure 18: Activating Trace Functionality

Starting / Stopping Trace

Trace is started and stopped when execution is started and stopped. But it can also be started and stopped independently. This is done by choosing the menu **Run | Trace Start/Stop,** or the corresponding button on the toolbar.

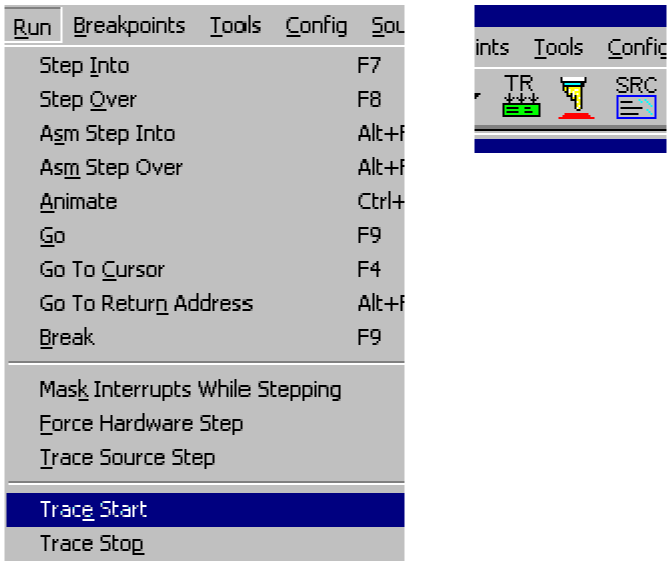
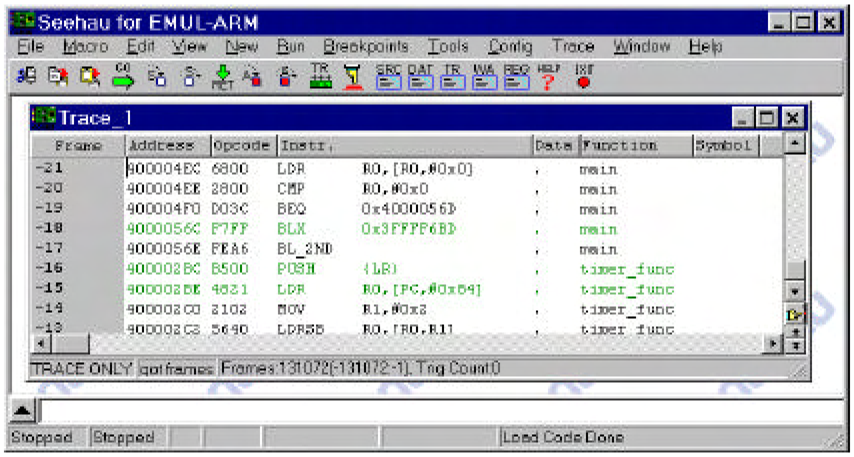


Figure 19: Trace Start and Stop

Trace Window


Seehau for EMUL-ARM
File Macro Edit View New Run Breakpoints Tools Config Trace Window Help
GO SRC DAT TR WA REG HELP INT
Trace_1

```
 Frame      Address   Opcode  Instr.                       Data  Function      Symbol
-21         400004EC  6800    LDR       R0,[R0,#0x0]       ,     main
-20         400004EE  2800    CMP       R0,#0x0            ,     main
-19         400004F0  D03C    BEQ       0x4000056D         ,     main
-18         4000056C  F7FF    BLX       0x3FFFF6BD         ,     main
-17         4000056E  FEA6    BL_2ND                       ,     main
-16         400002BC  B500    PUSH      {LR}               ,     timer_func
-15         400002BE  4821    LDR       R0,[PC,#0x84]      ,     timer_func
-14         400002C0  2102    MOV       R1,#0x2            ,     timer_func
-13         400002C2  5640    LDRSB     R0,[R0,R1]         ,     timer_func
TRACE ONLY  got frames  Frames:131072(-131072-1), Trig Count:0
Stopped  Stopped  Load Code Done
```

Figure 20: Trace Window

The picture above shows the **Trace** Window with all following columns visible:

- **Frame**
  Frame Number. Frame 0 always represents the trigger frame. If there is no trigger, frame -1 is the last frame in the buffer.
- **Time**
  Absolute or Relative time stamp (see below)
- **Address**
  Address of the bus cycle
- **Status**
  Traced instruction execution status
- **Opcode**
  Machine language instruction
- **Instr**
  Disassembled instruction
- **Data**
  Data accesses
- **Function**
  Function to which the instruction belongs
- **Symbol**
  Symbol associated with the trace line

Time Stamp displays one of the following, based on what you select in the **Trace** Menu:

- **Relative Time**
  Amount of elapsed time for the current instruction
- **Absolute Time**
  Amount of elapsed time since the beginning of execution

- **Relative Cycle**
  Number of CPU cycles for the current instruction

- **Absolute Cycle**
  Total number of CPU cycles since the beginning of execution

Symbols are shown with markers as follows:

- **Branch**
  Shown with an arrow in front, for instance “==> OSTaskIdleHook”

- **Symbol**
  Shown with a ‘:’ after it, for instance “init :”
- **Data Item Address**
  Shown within brackets, for instance [OSTaskID]

Trace Window Colors

Frame number colors indicate status; for an explanation of the color-coding, make the **Status** field visi-ble.

Trace information in green indicates that there is a source line for that address.

Trace Menu

The **Trace** Menu can be accessed when the **Trace** Window is active, achieved by right-clicking anywhere in the **Trace** Window.

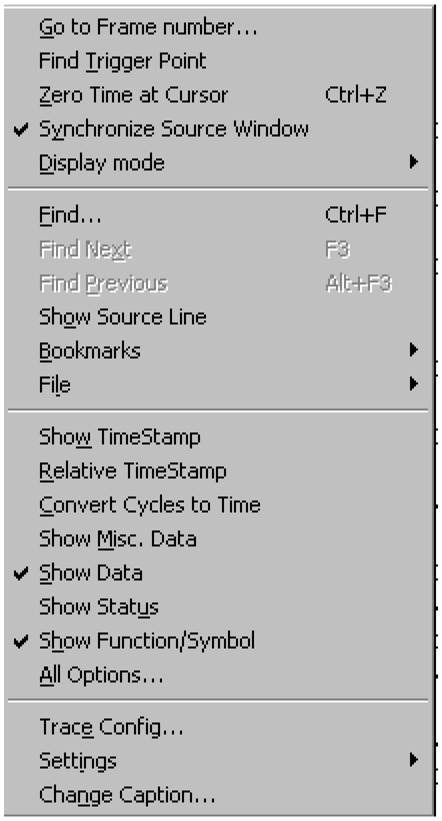


Figure 21: Trace Menu

- **Go to Frame number**
  Opens a dialog box where a specific frame number for display can be entered

- **Find Trigger Point**
  Displays the trigger point (frame zero)

- **Zero Time at Cursor**
  Changes the timestamp at the selected frame to zero and makes all other timestamps relative to the selected frame

- **Synchronize Source Window**
  Automatically aligns the display of code in the **Source** window as you scroll through the Assembly code in the trace buffer. You must use the up/down arrow keys in conjunction with this feature.

- **Display Mode**
  Opens a submenu that allows you to select which code to display:

  – Trace Only: Displays Assembly code only in the trace buffer

  - Mixed: Displays both C and Assembly code in the trace buffer
  - Source Only: Displays C source code only in the trace buffer

- **Find**
  Opens the **Find Address** dialog box where you can search the trace buffer for an address or a range of addresses and a cycle type
- **Find Next**
  Click to find the next frame match
- **Find Previous**
  Click to find the previous frame match
- **Show Source Line**
  Displays the associated source code for the current frame. The frame address must match a source line address.
- **Bookmarks**
  Operates with bookmarks for easy navigation through out the trace buffer. The bookmarks appear as blue rectangles on the left margin. Clicking on a frame number will highlight it and allow quick return. All the functionality is doubled by using the bookmark buttons on the tool bar. The tool bar will be visible when the first bookmark is set. All bookmarks will be lost when a new trace buffer is recorded. The following are sub-items:
  - **Toggle Bookmark** (Ctrl-F2)
    Toggles the bookmark on the selected trace frame. It can also be done by left mouse-clicking on the margin.
  - **Toggle Named Bookmark**
    Toggles the bookmark and allows you to set a specific name for it. Unnamed bookmarks are named by their frame number. This can also be done by holding down the Alt key and left mouse-clicking on the left margin.
  - **Next Bookmark** (F2)
    Locates the next bookmark down the trace buffer.
  - **Previous Bookmark** (Shift F2)
    Locates the previous bookmark on the trace buffer.
  - **Go To Bookmark** (Shift-Ctrl-F2)
    Lists all the existing bookmarks and can jump to the one specified.
  - **Delete All Bookmarks** (Shift-Ctrl-F2):
    Deletes all bookmarks.
- **File | Save to File**
  Opens the **Save Trace To File** dialog box where you can save the contents of the trace buffer as text to a file.

- **File | Print**
  Allows you to send the trace buffer to a printer

- **Show TimeStamp**
  Displays the timestamp which represents the number of machine cycles that have elapsed since the beginning of program execution

- **Relative TimeStamp**
  Displays the timestamp as the number of machine cycles that have elapsed since the execution of the previous instruction

- **Convert Cycles to Time**
  Converts the timestamp from machine cycles to actual time based on the MCU clock

- **Show Data**
  Data accesses to/from memory

- **Show Status**
  Traced instruction execution status

  IE__ – Instruction Executed – ARM mode.
  IE_T – Instruction Executed – Thumb mode.
  IN__ – Instruction NOT Executed – ARM mode.
  IN_T – Instruction NOT Executed – Thumb mode.
  BE__ - Branch executed.
  BE_T - Branch executed (THUMB mode).
  ID__ - Instruction executed with Data.
  ID_T - Instruction executed with Data (THUMB mode).
  BD__ - Branch executed with Data.
  BD_T - Branch executed with Data (THUMB mode).
  TR – Trigger occurred on this clock cycle.
  -TD- Trace Disabled (should have been filtered out).
  WT__ Wait, i.e. data only frame (should have been filtered out).

- **Show Function/Symbol**
  Function to which the instruction belongs

- **All Options**
  Opens the **Display Options** window where you can select or clear trace options in a single update

- **Trace Config**
  Opens the **Trace Configuration** dialog box.

- **Settings**
  Opens a submenu that allows you to set up **Trace** window attributes

- **Change Caption**
  Allows you to change the **Trace** window caption in the title bar

Synchronize Source Window

The output in the Trace can be synchronized with the **Source** window, which allows following the flow of execution when scrolling in the **Trace** Window. This feature is enabled/disabled on the **Trace** Menu.

Display Mode

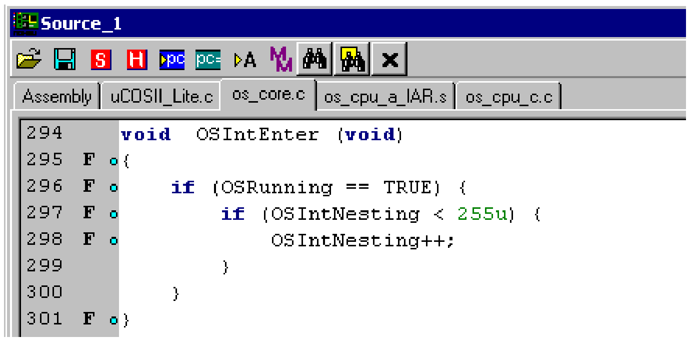
The **Trace** Window can show Trace Only (disassembly), Mixed Mode or Source Level only. This is controlled by the **Trace** Menu, or clicking in the lower left corner of the **Trace** Window (where it says "TRACE ONLY").

Trace Filtering

Filtering means to store only a small portion of the program execution, which can be applied on address range, among others. Seehau has the ability to configure one address range to filter using the GUI.

To use it, position the cursor where the filtering should start, right-click in the **Source** window and select **Filter | Begin Here**. Repeat for **Filter End**. Alternatively, Trace Filtering can be set up using the Trace Configuring menu, **Config | Trace**.

This results in a filter being set up. The picture below shows what this might look like. The "F" indicates Filter.



```
     void  OSIntEnter (void)
  F  {
  F      if (OSRunning == TRUE) {
  F          if (OSIntNesting < 255u) {
  F              OSIntNesting++;
             }
         }
  F  }
```

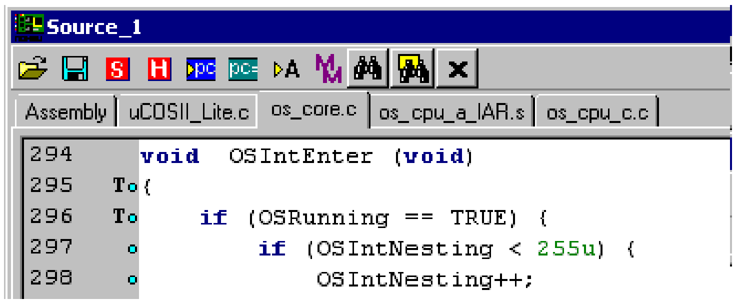
Figure 22: Trace Filtering

Trace Triggering

Trace triggering is similar to breakpoints, but will only stop the trace by default. (It can optionally stop execution.) This is useful, for instance, when the system under debug does not lend itself to breaking execution. Seehau has the ability to configure one trigger range to filter using the GUI.

To use it, position the cursor where the trigger should occur. Right-click in the **Source** window and select **Trigger | Toggle Here**. Alternatively, Trace Filtering can be set up using the Trace Configuring menu, **Config | Trace**.

This results in a trigger being set up. The picture below shows what this might look like. The "T" indicates Trigger.



```
294     void  OSIntEnter (void)
295   To{
296   To    if (OSRunning == TRUE) {
297    o        if (OSIntNesting < 255u) {
298    o            OSIntNesting++;
```

Figure 23: Trace Triggering

Advanced Features

Trace Chart (EMUL-ARM-PRO only)

A graphical representation of function calling captured in the trace memory is available from the **View – Trace Chart** menu item. This window can be scaled and markers are available for measuring times within this window. This window shows graphically the function calling pattern of a program and gives a compact display of the general pattern of execution of the program under test.

This window is a graphical overview of the trace memory data. The same data is displayed in detail in the **Trace** window.

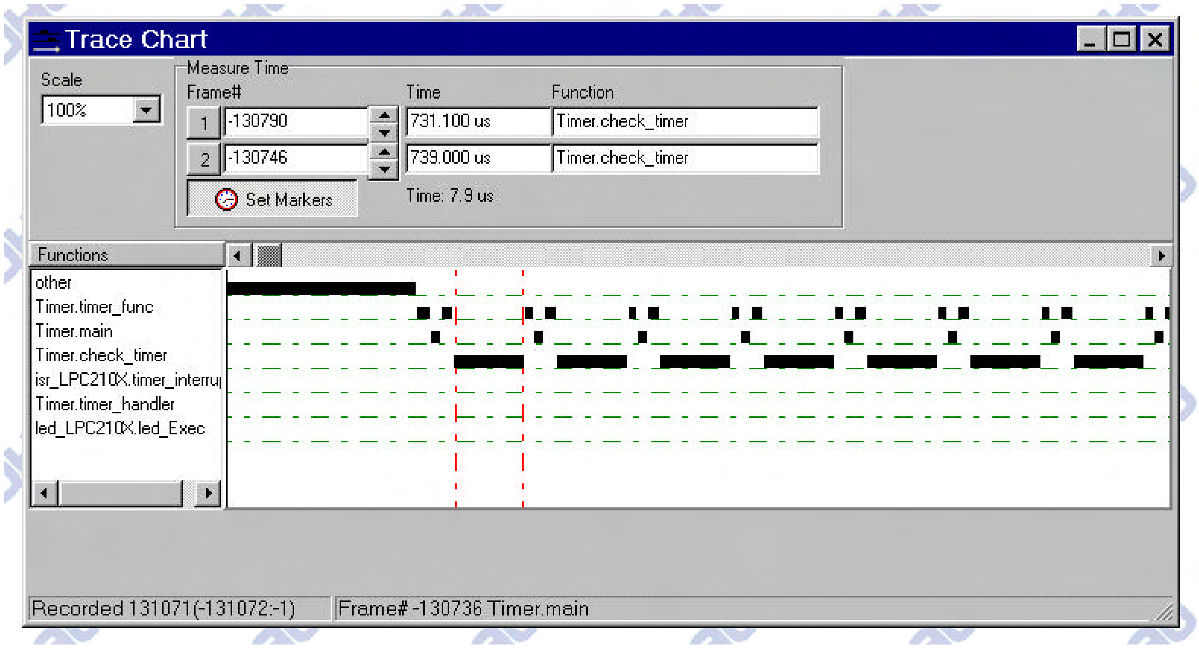
The function column shows the names of the C functions that have executions recorded in the graph window. The function execution order is displayed from left to right. A heavy line shows the function being executed.

The **Scale** field allows selection of the time scale factor, and the horizontal scroll bar at the top of the graph section of the window allows scrolling the window.

After clicking the **Set Markers** button, the first marker is set at the trace frame corresponding to the location in the graph window of the next mouse click. The ending marker is set by the next mouse click after that in the graph window.

As each marker is set, the frame number, time and function for the marker are displayed. The markers can be moved by using the increment and decrement buttons in the Frame# field.

When both markers are set, the Time field to the right of the **Set Markers** button displays the time difference between the markers.



100%

Frame#	Time	Function
1 -130790	731.100 us	Timer.check_timer
2 -130746	739.000 us	Timer.check_timer

Time: 7.9 us

other
Timer.timer_func
Timer.main
Timer.check_timer
isr_LPC210X.timer_interru
Timer.timer_handler
led_LPC210X.led_Exec

Recorded 131071(-131072:-1)   Frame# -130736 Timer.main

Figure 24: Trace Chart (EMUL-ARM-PRO only)

Trace Configuration

About Trace Configuration

We have already shown how to use the GUI-based configuration (for instance, how to set a filter using the **Source** Window).

However, for more advanced Trace Configuration, we offer a **Trace Configuration** dialog box which is opened using the **Config | Trace** menu.

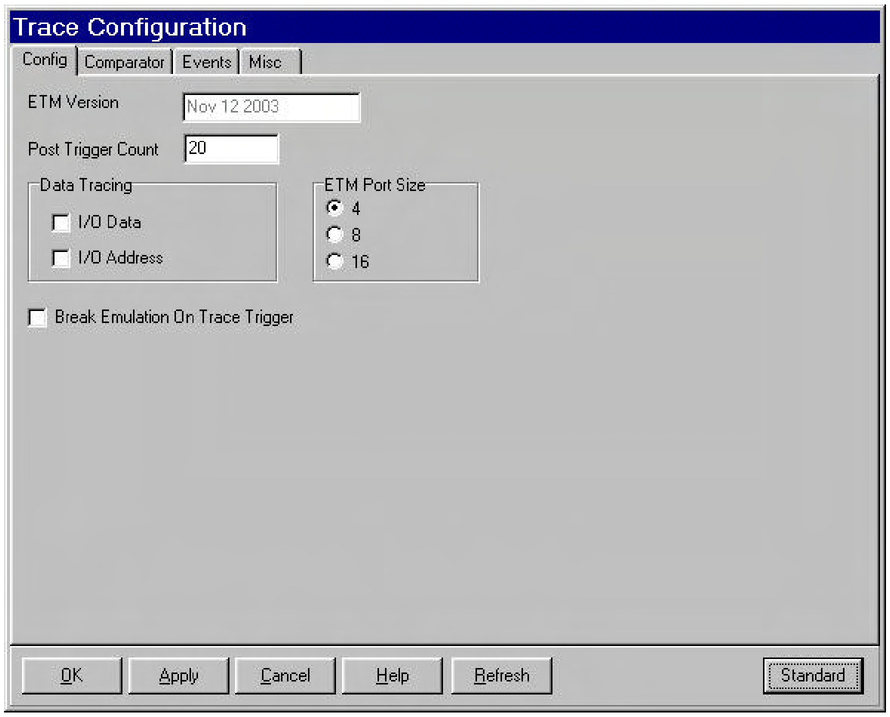
Standard / Advanced Mode

Trace Configuration operates in two different modes (not available for ARM Nexus Trace yet):

- Standard Mode – with a minimum of details
- Advanced Mode – allows all available configuration options

The example below shows the trace configuration in Advanced Mode. Change Mode by pressing the **Standard** button located to the bottom right in the dialog.

The picture below is for the ETM trace, but the principle for Standard and Advanced Mode is the same for both ETM and Nexus Trace.



ETM Version: Nov 12 2003

Post Trigger Count: 20

Data Tracing
- [ ] I/O Data
- [ ] I/O Address

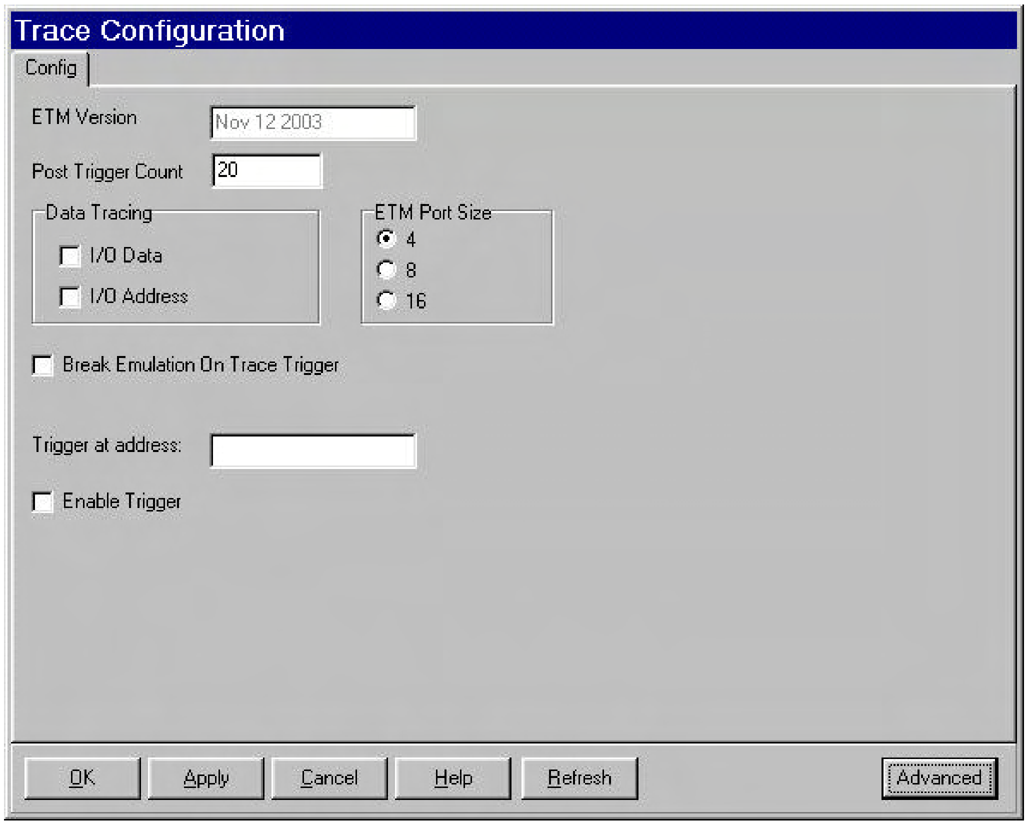
ETM Port Size
- [✓ 4]
- 8
- 16

- [ ] Break Emulation On Trace Trigger

Figure 25: Advanced Trace Configuration

ARM ETM Trace Configuration

Config Tab – ETM Standard Mode


Trace Configuration

Config

ETM Version: Nov 12 2003

Post Trigger Count: 20

Data Tracing
- [ ] I/O Data
- [ ] I/O Address

ETM Port Size: [✓ 4] 8 16

- [ ] Break Emulation On Trace Trigger

Trigger at address:

- [ ] Enable Trigger

Figure 26: ETM Trace Configuration

- **ETM Version**
  Shows the ETM version reported by the MCU.

- **Post Trigger Count**
  Number of frames to record after trigger occurred

- **ETM Port Size**
  The ETM uses 4, 8 or 16 bits to transfer information for each clock cycle. Seehau will default to the highest number of bits for any given MCU. If your target board is designed with less than maximum, you will need to configure accordingly.

- **Data Tracing**
  Select to Trace data writes, address and/or data. Note that not all ETM's allow data tracing, and that the buffers internally in the MCU may overflow if data tracing is enabled.

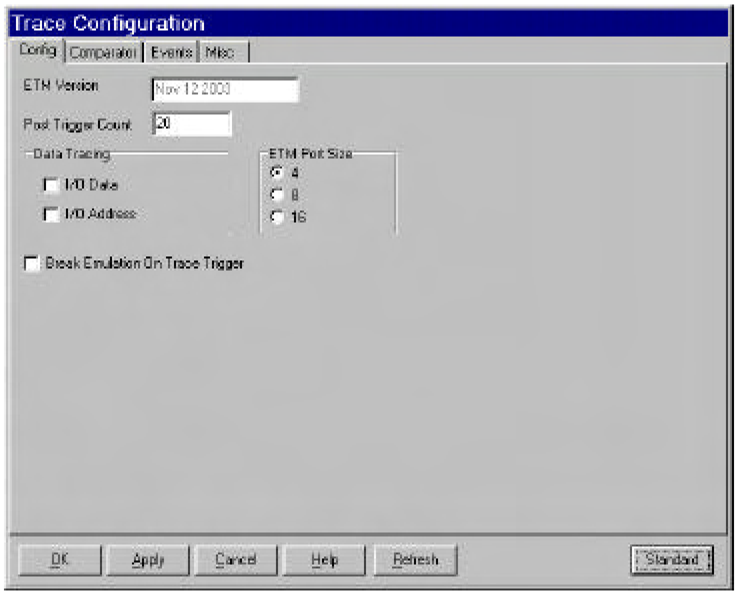
- **Break Emulation On Trace Trigger**
  When enabled, execution will stop at trigger. Otherwise, execution will continue as if nothing happened.

- **Trigger at Address**
  Select an address at which the trigger shall be active

- **Enable Trigger**
  Chhecck to enable triggering

Config Tab – ETM Advanced Mode

See the **Standard Mode Config** Tab.


Trace Configuration

Config | Comparator | Events | Misc

ETM Version: Nov 12 2003

Post Trigger Count: 20

Data Tracing
- [ ] I/O Data
- [ ] I/O Address

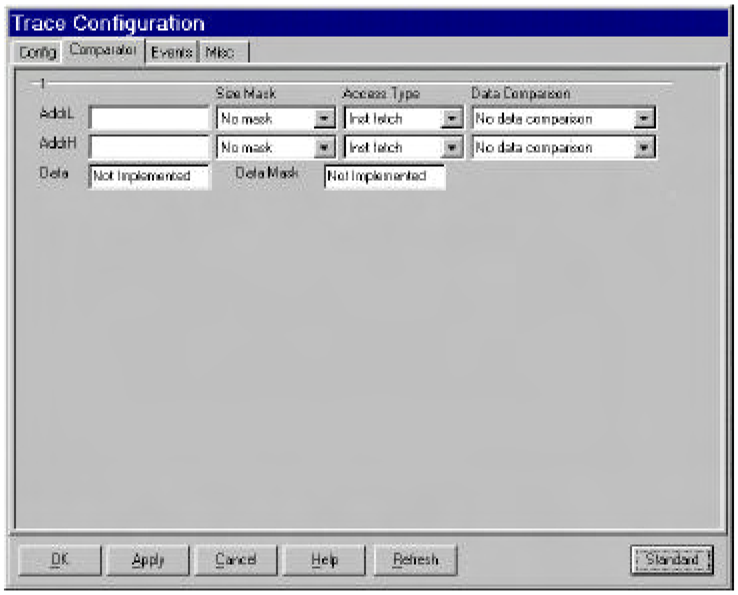
ETM Port Size: [✓ 4] 8 16

- [ ] Break Emulation On Trace Trigger

Figure 27: ETM Advanced Mode

Comparator Tab – ETM Advanced Mode

There are from 1 to 8 comparator pairs in the ETM depending on the processor chip configuration. Only comparators that are available for a specific processor will be displayed. A comparator pair consists of a low address, high address and data comparator, if supported by the processor. Unsupported data comparators will have “not implemented” in the data and data mask fields. The low and high addresses specify a range that can be used to trigger an event, e.g. trace recording enable, trace trigger, etc. The low and high address comparators can be used for single address comparison if not required for range. The size mask, access type and data comparison fields must match for range comparison, but need not match for single address comparison.



1

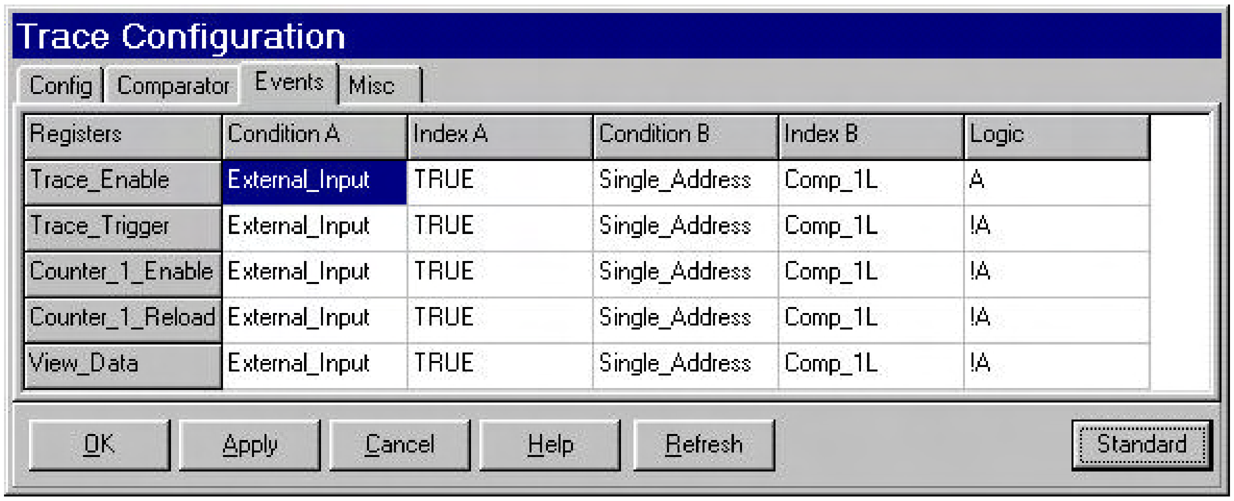
		Size Mask	Access Type	Data Comparison
AddrL		No mask	Inst fetch	No data comparison
AddrH		No mask	Inst fetch	No data comparison
Data	Not Implemented	Data Mask	Not Implemented	

Figure 28: ETM Advanced Mode Comparator Tab

Events Tab – ETM Advanced Mode

An event is a combination of conditions that when met will cause something to occur in the trace output. An example is a trace trigger when an instruction at a specific address is executed. An event register contains two conditions to be evaluated and selectable logic field to combine the two conditions and cause the event to occur. A condition can be an address comparison, watchpoint, counter has reached zero, external input, map decode, sequencer state, or external input. The condition index selects the specific source of the condition to be evaluated. Only available conditions will be displayed.

The logic field is used to evaluate the two conditions for the event to occur. The logic field can select condition **A** only or a combination of **A and B**. The boolean expression of the two condition fields is selected for a condition true (E.G. A) or False (E.G.! A). For an event to be programmed to always occur, a special combination of external input with index of True is selected with the logic field set to A, an event to always be false would be !A.



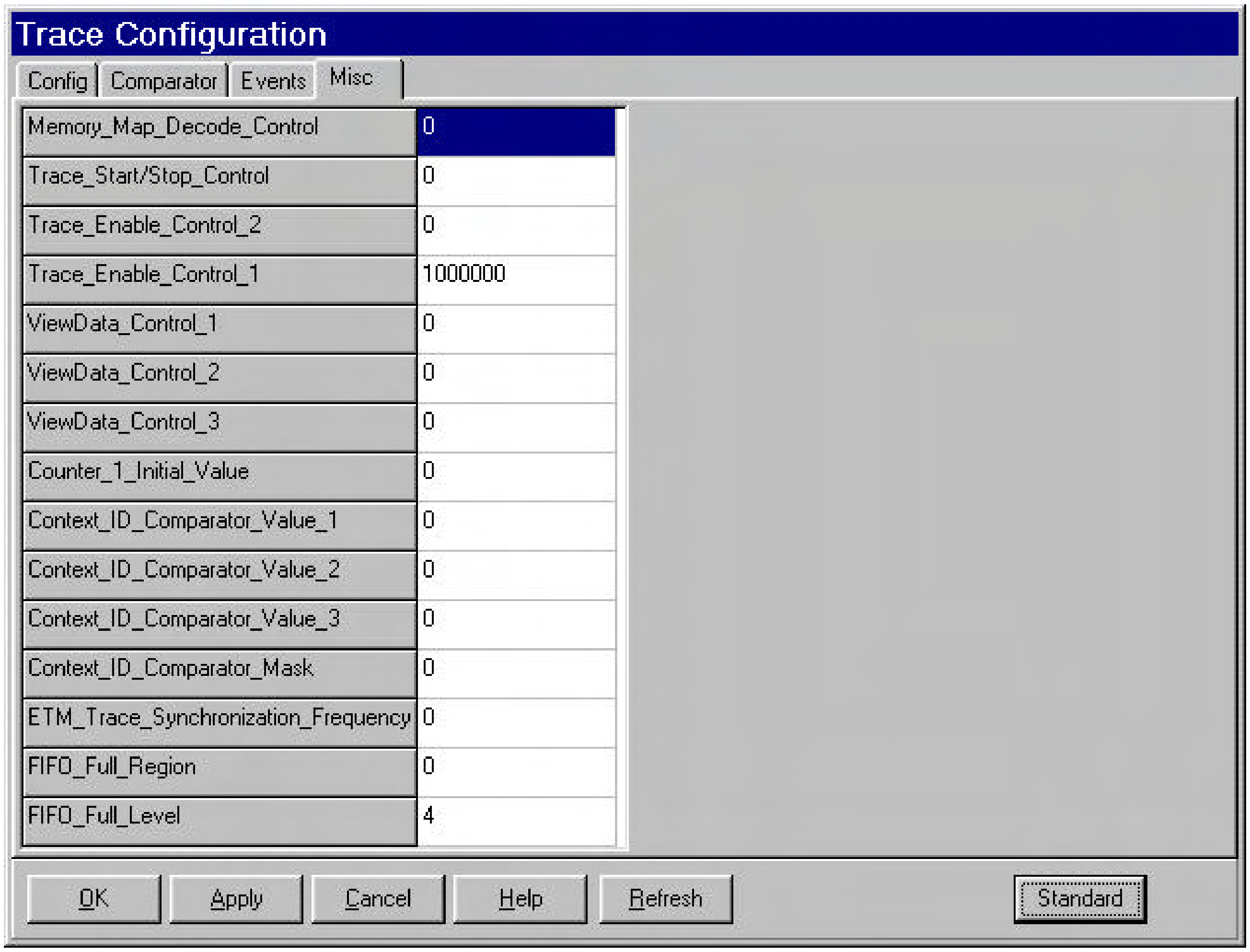
Registers	Condition A	Index A	Condition B	Index B	Logic
Trace_Enable	External_Input	TRUE	Single_Address	Comp_1L	A
Trace_Trigger	External_Input	TRUE	Single_Address	Comp_1L	!A
Counter_1_Enable	External_Input	TRUE	Single_Address	Comp_1L	!A
Counter_1_Reload	External_Input	TRUE	Single_Address	Comp_1L	!A
View_Data	External_Input	TRUE	Single_Address	Comp_1L	!A

Figure 29: ETM Advanced Mode Events Tab

Misc Tab – ETM Advanced Mode

The **Misc** Tab shows advanced topics. These registers are used by the trace configuration software. The only register of interest to the user is the counter initial value registers. The initial value is loaded into the specific counter on trace start or when the specific counter reload event occurs.

Used for the ETM Counter and the ETM Context Comparator.

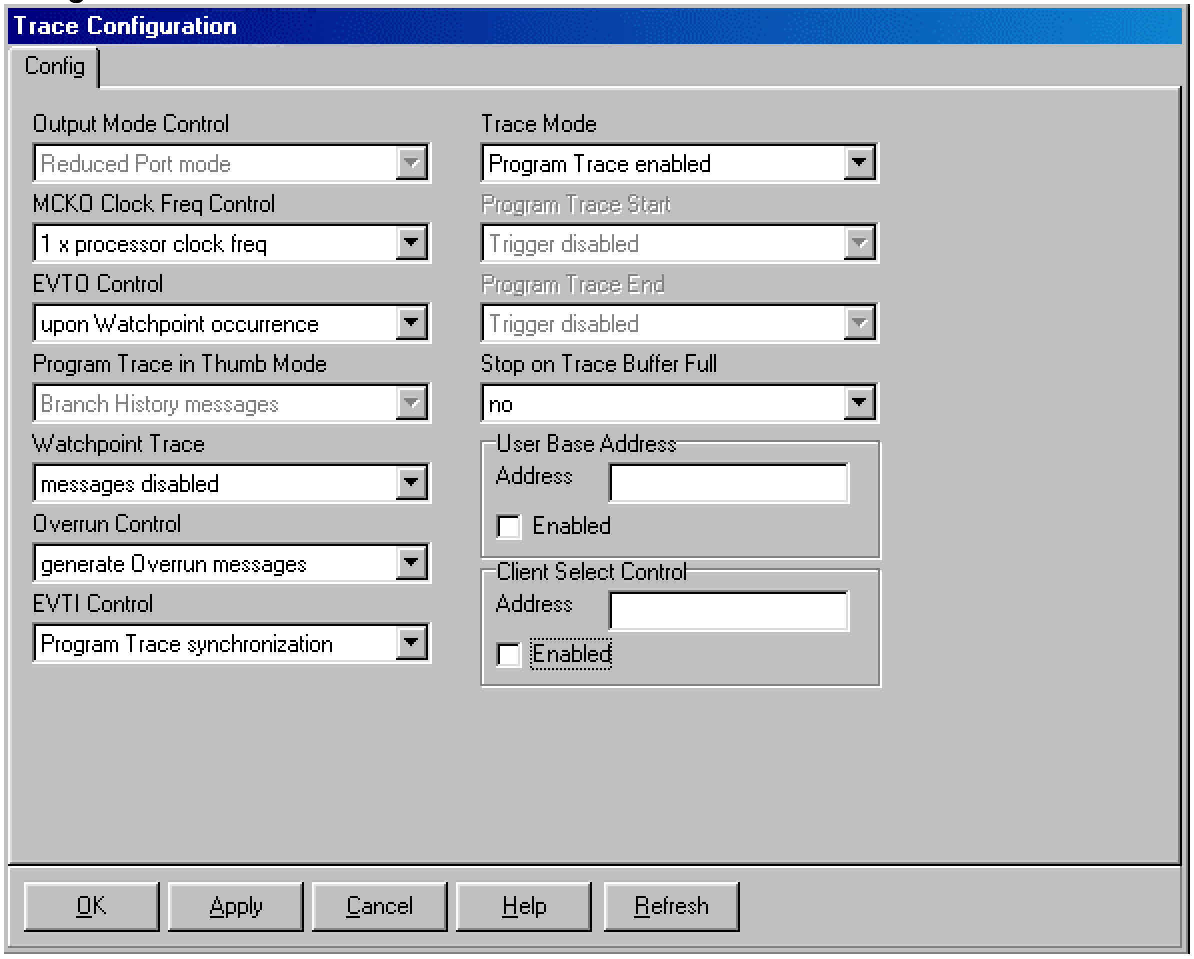


Memory_Map_Decode_Control	0
Trace_Start/Stop_Control	0
Trace_Enable_Control_2	0
Trace_Enable_Control_1	1000000
ViewData_Control_1	0
ViewData_Control_2	0
ViewData_Control_3	0
Counter_1_Initial_Value	0
Context_ID_Comparator_Value_1	0
Context_ID_Comparator_Value_2	0
Context_ID_Comparator_Value_3	0
Context_ID_Comparator_Mask	0
ETM_Trace_Synchronization_Frequency	0
FIFO_Full_Region	0
FIFO_Full_Level	4

Figure 32: ETM Advanced Mode Miscellaneous Tab

ARM Nexus Trace Configuration

Config Tab – Nexus



Output Mode Control: Reduced Port mode
MCKO Clock Freq Control: 1 x processor clock freq
EVTO Control: upon Watchpoint occurrence
Program Trace in Thumb Mode: Branch History messages
Watchpoint Trace: messages disabled
Overrun Control: generate Overrun messages
EVTI Control: Program Trace synchronization
Trace Mode: Program Trace enabled
Program Trace Start: Trigger disabled
Program Trace End: Trigger disabled
Stop on Trace Buffer Full: no
User Base Address — Address: ; Enabled: ☐
Client Select Control — Address: ; Enabled: ☐

Figure 33: Nexus ARM Trace Configuration

- **Output Mode Control**
  Based on the micro's abilities you will have one or two options:
  - Reduced Port mode - 2 pins are used
  - Full Port mode - 8 pins are used (only on devices that support this operation.)
- **MCKO Clock Freq Control**
  This is the clocking control for the trace. This means the trace receives data at the speed you have selected in reference to the MCU's clock.
- **EVTO Control** (Event Output)
  Two options are available here for the event output pin to send a signal to some device like a Logic Analyzer or scope.
  - upon Watchpoint occurrence

  - upon Debug mode entry

- **Program Trace in Thumb Mode**
  Only one mode is currently supported: “Branch History messages”.

- **Watchpoint Trace**
  This tells the micro to either generate or not to generate the event message when a watchpoint has been hit.

- **Overrun Control**
  This feature can either generate a message that an overrun has occurred in the message queue in the microcontroller, or halt the MCU until the message queue can accept more messages.

- **EVTI Control** (Event Input)
  Three options are available here for the event-input pin to supply the trace with one of two events, or you can also disable this feature.

  - Program Trace synchronization
  - Debug mode request (The current revision of the micro does not support this feature.)
  - Disabled

- **Trace Mode**
  Three options are available to control how the trace will function.

  - Program Trace enabled - Trace program execution flow
  - Ownership Trace enabled - Trace based on the Ownership control of the microcontroller.

  Note: Must be used with User Base Address (UBA) so that the system can track the access to the ownership trace.

- **Program Trace Start**
  This is used in conduction with the Ownership Trace, and tells the MCU to start generating messages for the program flow at this point.

  - The available options are to use either watchpoint 1 or 2.

- **Program Trace Stop**
  This is used in conduction with the Ownership Trace, and tells the MCU to stop generating messages for the program flow at this point.

  - The available options are to use either watchpoint 1 or 2.

- **Stop on Trace Buffer Full**
  This feature will stop the trace recording when the trace buffer is completely filled with data.

- **User Base Address** (UBA)
  Ownership Trace Messaging is implemented using the Nexus defined User Base Address Register. The User Base Address Register defines the memory mapped base address for the Ownership Trace Register (OTR). The operating system writes the ID for the current task/process in the OTR.

- **Client Select Control**
  This tells the debug system which client in the debug chain is under development.