

Chameleon

series

SIGNUM SYSTEMS CORPORATION

ETM Addendum to Chameleon Debugger

User Manual

SIGNUM
S Y S T E M S

COPYRIGHT NOTICE

Copyright (c) 2007 by Signum Systems Corporation. All rights are reserved worldwide. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual, or otherwise, without the prior written permission of Signum Systems.

DISCLAIMER

Signum Systems makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Also, Signum Systems reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Signum Systems to notify any person or organization of such revision or changes.

WARRANTY

Signum Systems warrants to the original purchaser that this product is free of defects in material and workmanship and performs to applicable published Signum Systems specifications for a period of SIX MONTHS from the date of shipment. If defective, the product must be returned to Signum Systems, prepaid, within the warranty period, and it will be repaired or replaced (at our option) at no charge. Equipment or parts which have been subject to misuse, abuse, alteration, neglect, accident, unauthorized installation or repair are not covered by warranty. This warranty is in lieu of any other warranty expressed or implied. IN NO EVENT SHALL SIGNUM SYSTEMS BE LIABLE FOR CONSEQUENTIAL DAMAGES OF ANY KIND. It is up to the purchaser to determine the reliability and suitability of this product for his particular application.

SIGNUM
S Y S T E M S
1211 FLYNN RD., UNIT #104
CAMARILLO, CA 93012, U.S.A
PHONE 805 • 383 • 3682
WWW.SIGNUM.COM

Table of Contents

C H A P T E R 1

ETM Trace Buffer	1
ETM Status Tab	2
General Control Tab	3
ETM Trigger/Settings Tab	4

C H A P T E R 2

Trace Display Window	6
Trace Window Items	7
Source View Window Items	7

C H A P T E R 3

Trace Window Operations	9
Built-In Keyboard Operations	9
Mouse Operations	9
Main Popup Menu	10
Popup Menu and Trace Window Toolbar	10
Field Format Selection Menu	12
Quick Tag & Filter Menu	13
Complex Tagging and Filtering	14
Customizing Displayed Fields	15
Saving Trace	16
Binary Saving	17
Text Saving	18

C H A P T E R 4

ETM Trace Commands	20
General Commands	20
Trace Capture Control	21
On-Chip ETM Control	22
Low Level Access to ETM Control Registers	22

C H A P T E R 5

Using JTAGjet-Trace with Signum EVB-LPC2138 Board	24
JTAGjet-Trace Hardware Setup	25
Chameleon Debugger Setup	27
Setup Verification	28
Creating Macro Buttons	29
Working with ETM	31
Data Tracing with Philips LPC	40

C H A P T E R 6

Using JTAGjet-Trace and Keil MCB2100 Board	46
JTAGjet-ETM Hardware Setup	47
Chameleon Debugger Setup	49
Chameleon Debugger Setup	50

S I G N U M S Y S T E M S

C H A P T E R 7

Using JTAGjet-Trace with OMAP5905	53
JTAGjet-ETM Hardware Setup	54
Chameleon Debugger Setup	55
Using ETM Trace – Short Tutorial	61
Data Tracing	65
Using Macros	68

C H A P T E R 8

Using JTAGjet-Trace-OMAP with OMAP5912	
“Larry Board”	70
JTAGjet-Trace Hardware Setup	71
Chameleon Debugger Setup	73
Setup Verification	74
Creating Macro Buttons	75
Working with ETM	77
Data Tracing with OMAP5912	87
TRACING VARIABLE RANGE	89
Changing Variables During Execution	91

C H A P T E R 9

Using JTAGjet-Trace with Raisonance REva	
STR912F Board	94
JTAGjet-Trace Hardware Setup	95
Chameleon Debugger Setup	95
Setup Verification	96

Creating Macro Buttons	97
Working with ETM	99
Data Tracing with Raisonance STR912F	109

Chapter

1

If you have already installed the software . . .

This addendum to Signum Chameleon Debugger contains reference material for the ETM trace usage along with short tutorials on how to use the JTAGjet-Trace emulator with the Philips LPC and TI OMAP devices.

If you have already installed the software and have the Philips LPC or OMAP target board available, you may want to refer directly to the tutorial section of your interest.

ETM Trace Buffer

To access the **ETM Control box**, open the Trace window using either the View menu or the TRC button on Chameleon Debugger main tool bar.

The ETM trace capture is controlled by the ETM Control dialog box. To open the box, press the Control button located on the far left of the Trace window tool bar. (Figure 1)

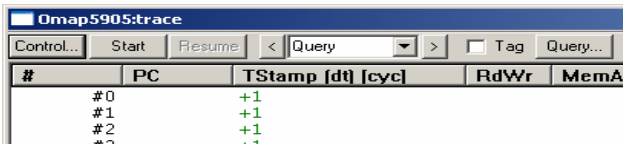


FIGURE 1 ETM Toolbar.

The ETM Control dialog box contains these three tabs:

T A B	P U R P O S E
Status	Displays the on-chip ETM resources.
General Control	Sets up the emulator trace resources.
ETM Trigger / Settings	Sets up the on-chip triggers, events and other ETM controls.

ETM Status Tab

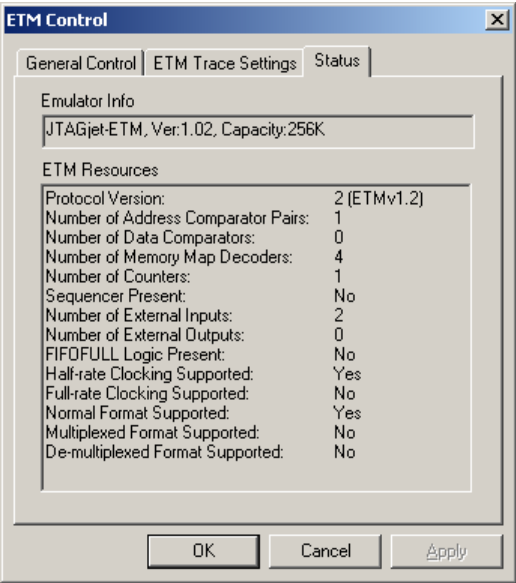


FIGURE 2 ETM Status tab

The Status tab displays all ETM resources available on the ARM silicon being debugged. These resources were fixed by the silicon vendor for that particular device and cannot be changed.

At the very top, the Status tab displays the version and trace depth capacity of the JTAGjet emulator. The trace frame width is always 36 bits, which allows for

capturing 4, 8 and 16-bit wide trace packets. The remainder of the trace frame is reserved for status and time stamp information.

The depth of the trace size (depth) is one of the following: 256K, 1M, 2M or 4M frames.

General Control Tab

This tab is used to enable and configure the ETM recording modes.

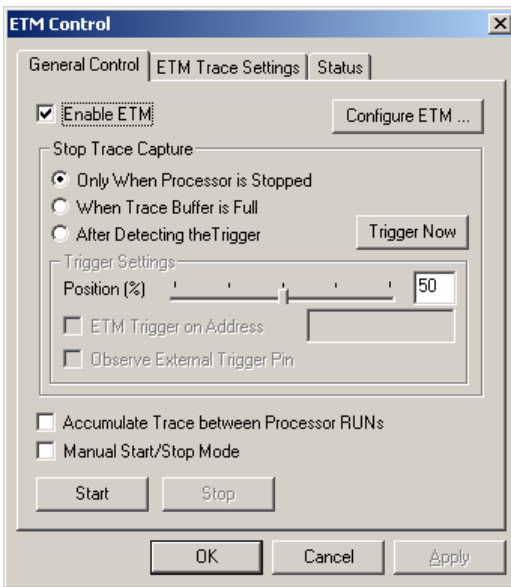


FIGURE 3 ETM Control tab.

Once the ETM Enable box is checked, the software configures the ETM for the best available trace resources so that the Configure ETM button does not need to be used. Still, the user can view and change the ETM control parameters by clicking on the Configure ETM button if necessary.

The General Control tab also allows to decide if the trace buffer is to be protected from being overwritten, is allowed to be overwritten until stopped, or set up to accumulate trace captures between experiments.

ETM Trigger/Settings Tab

This tab is used to control what the ARM device sends on the ETM trace bus pins. (Figure 4). It allows to set the most common debug trace modes without the need to read the Technical ETM Specifications. These modes include the following trace options:

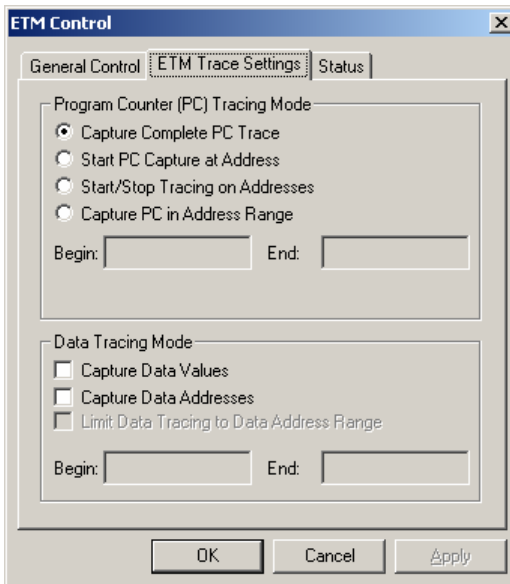


FIGURE 4 ETM Trace Settings tab.

1. **Capture complete PC trace.** This option will capture all PC addresses.
2. **Start PC Capture at Address.** The trace capture will not begin until the program executes the address specified in the **Begin** field.
3. **Start/Stop Tracing on Address.** This option will begin capturing trace when the **Begin** address is executed and stop capture when the **End** address is executed.

After the trace starts tracing the execution, any interrupts or functions called before the **End** address is executed will be visible in trace.

4. **Capture PC in Address Range.** This option will begin capturing trace when the PC in the address range given by **Begin** and **End** fields. In contrast to the option above, any functions or interrupts with PC addresses not in the range will not be captured in the trace memory.
5. **Capture Data Values.** When this option is set, in addition to the PC execution trace set above, any variables (or peripherals) read and written will be traced and their values stored in the trace memory.
6. **Capture Data Addresses.** This option allows to capture the addresses of variables (or peripherals) that were read or written during execution. variables addresses the data variables can be traced. The user may specify to capture data addresses, data values or both.
7. **Limit Data Tracing to Data Address Range.** This option shows only when Capture Data Addresses are selected and allows to limit variable tracing to a range of addresses. This option is recommended when using a 4-bit wide ETM port to avoid trace port FIFO overflows. On some ARM cores, this option will not be visible when Capture PC in Address Range is selected, due to limited silicon resources (only one address range available).

Note: Tracing variables with 4-bit wide ETM trace (like the one on Philips LPC2xxx devices) will overflow the ETM bandwidth quickly, so it must be used with caution (tracing only selected range of variables, etc.). Please see the LPC2xxx tutorials for detailed usage of the variable trace with 4-bit wide ETM port.

Chapter

2

Trace Display Window

The screenshot shows the Trace Display Window of a debugger. The window has a title bar with the text "Lpc2138:trace". Below the title bar is a toolbar with buttons for "Control...", "Start", "Resume", "Clear", "Query...", "Filter...", "Fields...", and "Save...". The main area displays a list of instructions with columns for "#", "PC", "Disas", "Source", and "TStamp [dt] [c]". The current sample is highlighted in yellow. The status bar at the bottom shows "Status: NotActive" and "Last Sample #262137 (100%)".

Title

Toolbar

Column headings

Current sample in Trace and in Source window

Trace filtered out markers

Status bar

#	PC	Disas	Source	TStamp [dt] [c]
#21	4000004C	LDR R1,[PC,#0x4c]	LDR r1,=[I...+3	
#22	40000050	LDR R3,[PC,#0x4c]	LDR r3,=[I...+1	
#23	40000054	CMP R0,R1	CMP r0,r1...+1	
#24	40000058	BEQ 0x4000006c	BEQ %F1...+5	
#27	4000006C	LDR R1,[PC,#0x34]	LDR r1,=[I...+1	
#2			MOV r2,#0...+1	
#2			CMP r3,r1...+1	
#3			STRCC r2,[r3...+1	
#3			BCC %B2...+1	
#3			C_Entry...+5	
#3			LR}	
#37	40000068	MOV R2,#0	GlobalStruct.Co	
#40	400000EC	LDR R0,[PC,#0xc8]	GlobalStruct.Pc	
#43	400000F8	BL 0x4000024c	led_init();	
#46	4000024C	MOV R1,#0	*pLPC_GPIO0_DI	
			R1,#0xff00	
			*pLPC_GPIO0_SE	
			PC,LR	
			R0,[PC,#0xb4]	
			for (..) {	
			led_set(0);	
			R0,#0	
#72	40000280	LDR R1,[PC,#0xc]	{	
#74	40000288	CMP R0,#0	if (v){	
#78	400002A4	MOV R2,#0xff00	else {	
			PC,LR	
			R4,#0	
			R0,#0x1	
			delay(1);	
#97	400001E4	SIMFD	SPI,{R4,LR}	
			{	

62: LDR r1,=[Image\$\$RW\$\$Base
63: LDR r3,=[Image\$\$ZI\$\$Base
64: CMP r0,r1
65: BEQ %F1
66: CMP r1,r3
67: LDRCC r2,[r0],#4
68: STRCC r2,[r1],#4
69: BCC %B0
70: 1 LDR r1,=[Image\$\$ZI\$\$Limit
71: MOV r2,#0
72: 2 CMP r3,r1
73: STRCC r2,[r3],#4
74: BCC %B2
75:
76:
77: ; Enter the C code
78: IMPORT C_Entry
79:

Status: NotActive Last Sample #262137 (100%)

For Help, press F1

FIGURE 5 Trace Display window.

Figure 5 shows Chameleon Debugger's Trace window along with the corresponding Source window. Right-clicking in the Trace window brings up a context popup menu.

Trace Window Items

ITEM	DESCRIPTION
Title	The window's title bar shows the name of the trace data file.
Toolbar	Contains buttons linked to frequently used options. (The right-click popup menu contains additional options.)
Column Headings	The column titles in the trace data display area. (The “#” column contains sample identifiers.)
Bookmark	The numerical bookmark identifying a trace sample. Toggled by double-clicking in the “#” column.
Current sample	The currently selected trace sample. Displayed in yellow. The selected trace sample will also show in Source Window.
Filtered-out marker	The red dotted lines marking the samples hidden as a result of post-filtering.
Tagged sample	The lines selected using the Tag option (popup menu). Highlighted in gray.
Status bar	Shows from left to right: activity, % utilization and currently measured ETM clock frequency.

TABLE 1 Trace window items. See Figure 5.

Source View Window Items

Source code can be displayed in multiple Source windows. The contents of the windows correspond to the current trace sample in the Trace window. Changing the current sample in the Trace window using the cursor or up and down arrow keys automatically updates the Source windows. Current samples are indicated by the yellow background. The title bars of the Source windows display the source code file names. The Source window highlights the current source code line in yellow. There can be only one current code line in each Source window. It

S I G N U M S Y S T E M S

corresponds to the current sample in the Trace windows. If a source file is displayed in multiple Source windows, it will be positioned identically in all windows.

Trace Window Operations

Built-In Keyboard Operations

OPERATION	DESCRIPTION
Keys 0 - 9	Go to bookmark #N (0 denotes bookmark #10).
Up/Down cursors	Move current sample position
Left/Right cursors	Navigate between bookmarks
Home/End	Beginning or end of trace buffer
PageUp/PageDn	Show previous/next page of samples.

TABLE 2 Active keys in the Trace window.

Mouse Operations

MOUSE OPERATION	GUI CONTEXT	FUNCTION
Left Click	In column header	Displays column customization menu
	In Timestamp column	Sets sample with time 0 in relative mode
	Other column	Move current sample and synchronize sources

MOUSE OPERATION	GUI CONTEXT	FUNCTION
Right click		Display popup menu
Double click	In # column	Toggle sample bookmark
	In Timestamp column	Change timestamp display mode
	Any other column	Display quick Tagging/Filtering menu

TABLE 3 Mouse operations in the Trace window.

Main Popup Menu

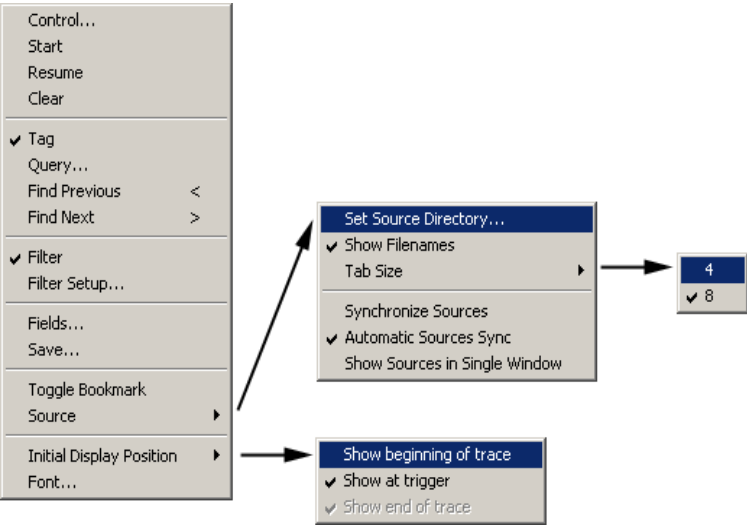


FIGURE 6 Trace popup menu

Popup Menu and Trace Window Toolbar

MENU OPTION	SUBMENU OPTION	DESCRIPTION
Control		Displays Trace Control window.

M E N U O P T I O N	S U B M E N U O P T I O N	D E S C R I P T I O N
Start/Stop		. Start/Stop acquisition. Stopped acquisition may be later resumed or started again. Start will start collecting from beginning of the trace buffer (old data will be lost).
Resume		. Resume trace acquisition (new trace data will be appended to existing trace data).
Clear		. Clears the entire Trace buffer (data is lost)
Tag		. Switch on/off tagging (all trace samples which meet criteria defined by search query will be highlighted). Lines with address 0xF20 are tagged on the above screen.
Query		. Display dialog box to define query condition (used for searching and tagging).
Find Previous/Next		. Search for samples or navigate through bookmarks or triggers.
Filter		. Switch filtering on/off.
Filter Setup		. Define condition for filtering.
Fields		. Customize list of displayed fields.
Save		. Save samples to a file (text or binary).
Toggle Bookmark		. Toggle bookmark on current sample.
Source		. Sub-menu to access source code
	Set Source Directory	. Sets directories used for source files searching
	Show Filenames	. Displays filenames if source cannot be located.
	Tab Size	. Size of TAB used in source display (4 or 8).
	Synchronize Sources	. Synchronizes sources with current sample. Useful if automatic synchronization is OFF.
	Automatic Sources Sync	. Synchronizes source display on every change of current sample position.
	Show Sources in Single Window	. Displays all source code in single window (unless source window is already displayed in it's own window).
Initial Display Position		. Sub-menu defining initial display position.

M E N U O P T I O N	S U B M E N U O P T I O N	D E S C R I P T I O N
	Show beginning of trace	Trace will be displayed at the beginning
	Show at trigger	Trace will be displayed at trigger point, if known.
	Show end of trace	Trace will be displayed at the end
Font		Standard windows menu to define font used in trace display.

TABLE 4 Trace Window Menu Option.

Field Format Selection Menu

This menu is displayed after clicking the column header (Figure 7).

The **Tag and Filter** sub-menu is described below. The second group of options, available for numeric fields only, allows the user to define the display format of the numeric field. The two remaining groups are for the for timestamp fields only. They enable the user to set the timestamp format to Absolute, Relative or Delta, and select the time unit.

The Header of the timestamp column displays the format in use and the wall-clock time, as shown in Table 5.

F O R M A T	D E S C R I P T I O N
[abs]	Absolute timestamp.
[rel]	Relative timestamp—positive or negative difference from base timestamp sample. Click on value in timestamp column to set base timestamp sample whose time is treated as time 0.
[dt]	Delta timestamp—distance in time to next sample.
[cyc]	CPU cycles (default)
[...]	wall-clock time (s/ms/us/ns)

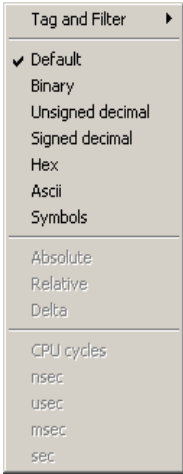


FIGURE 7 Trace FieldFormat Menu.

TABLE 5 Display formats of the Header and Timestamp column.

Quick Tag & Filter Menu

Clicking on a trace sample displays the Quick Tag & Filter menu. This menu allows you to quickly tag and filter trace samples based on the value of the currently selected sample (Figure 8).

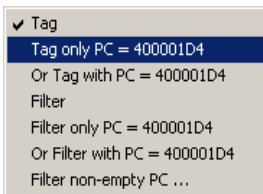


FIGURE 8 Tag & Filter Menu.

OPTION	DESCRIPTION
Keys 0 – 9	Go to bookmark #N (0 denotes bookmark #10).
Tag	Turn tagging on/off
Tag only NAME = VALUE	Tag only samples which have VALUE in field NAME
Or Tag with NAME = VALUE	Tag all currently tagged samples or samples which have VALUE in field NAME.
Filter	Turn filtering on or off.
Filter only NAME = VALUE	Filter only samples which have VALUE in field NAME
And Filter with NAME = VALUE	Filter all currently tagged samples or samples which have VALUE in field NAME.
Filter non-empty NAME	Filter only non-empty values of a field. Useful when filtering on Source field.

TABLE 6 Functionality of the Tag & Filter menu.

If the current condition already includes one of the fields, the **Or** prefix is displayed in front of the fields in the Tag & Filter menu. The condition becomes less

restrictive, causing more samples satisfy the new requirement. Conversely, if the current condition exists, but does not include any restrictions on the values in the Name field, the **And** prefix is displayed. The condition becomes more restrictive, since it needs to fulfill the requirement for more than one field.

Complex Tagging and Filtering

By clicking the **Query** button or the **Filter** button, or by selecting corresponding menu options, it is possible to define complex tagging and filtering conditions.

Numeric field options are shown in Figure 9.

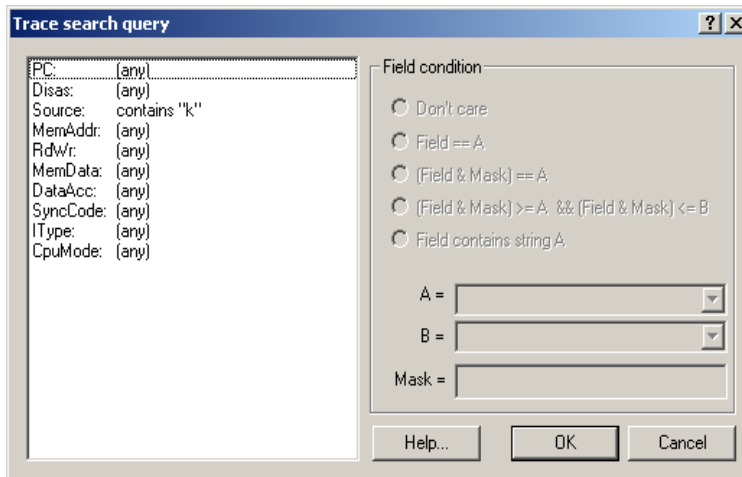


FIGURE 9 Trace Search Query dialog box.

The settings in Figure 9 are the result of selecting the **Tag Only PC = 0x20001234** option in the Tag & Filtering quick menu.

Values B and Mask allow to define ranges and mask conditions. Note that these options are not available from the quick menu.

Customizing Displayed Fields

Clicking the **Fields** button, or selecting the appropriate menu item, displays the Trace View Field Configuration dialog box (Figure 10). It allows you to customize the list of fields visible in the Trace window, as described in Table 7.

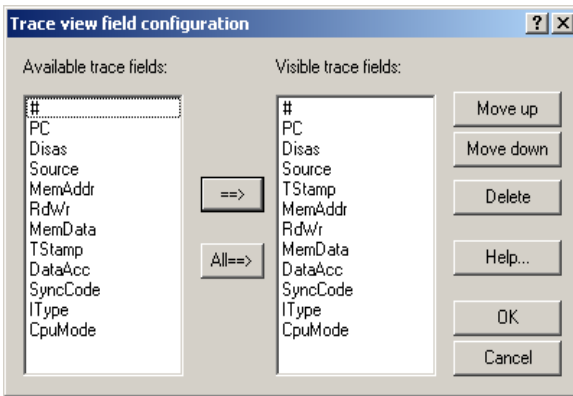


FIGURE 10 Trace Fields.

FIELD	DESCRIPTION
#	Frame sequence number – the smaller #, the older sample
PC	Program Counter
Disas	Disassembled code (in case there is no source file)
Source	Source code associated with the trace frame
MemAddr	Address of memory variable being read or written
RdWr	Data memory operation (Rd or Wr)
MemData	Variable value being read or written
TStamp	Time stamp (56-bits wide, will not overflow for years)
DataAcc	Type of memory access (Byte, Word, Half)

F I E L D	D E S C R I P T I O N
SyncCode	Trace status: <ul style="list-style-type: none">Start –start of trace after CPU stopSync –ETM sync framesOn –start of trace after each triggerFIFO – ETM FIFO had an overflow (loss of trace data)
CpuMode	CPU execution mode: <ul style="list-style-type: none">ARM – ARM mode executionThumb – Thumb mode executionNoExec – Instruction fetched but not executed

TABLE 7 Customizing Trace window fields.

Note: The Available trace fields may be different for some CPUs.

O P E R A T I O N	F U N C T I O N
= = >	Add selected field on end of visible fields
All = =>	Add all available fields as visible fields
Move up	Move selected field up (will be displayed more on left side)
Move down	Move selected field down (will be displayed on right side)
Delete	Delete selected field from list of visible fields

TABLE 8 Trace Fields Buttons.

It is possible to add the same field multiple times. The display format of each field may be different. The feature is especially useful for timestamp fields, when viewing time in delta or relative modes.

Saving Trace

The Save button, as well as the appropriate menu item, allows you to save the trace to a file in one of the formats listed in Table 9.

F O R M A T	D E S C R I P T I O N
= = >	Add selected field on end of visible fields
CSV	Can be loaded by Microsoft Excel or a text editor. Cannot be

FORMAT	DESCRIPTION
	loaded by the TraceDisplay module.
TTD	Text Trace Test Data file that is reloadable. May be opened by a text editor or used by external programs.
TDF	<p>Binary Trace Data File. Includes all binary trace data, display settings, filtering conditions, bookmarks and other data. After loading of this file, one will see same samples as seen in the moment when file was saved, since filtering conditions will be saved and automatically applied after loading of a file.</p> <p>Since the filtering conditions are saved and automatically applied after the file is loaded, the same set of samples shown at the time the file was saved will be displayed.</p>

TABLE 9 Recognized trace file formats.

Binary Saving

Saving the trace as a binary file prevents you from specifying the range of records or customizing the fields to be saved (Figure 11). All information is saved in one file that later can be copied to another computer for display and other purposes.

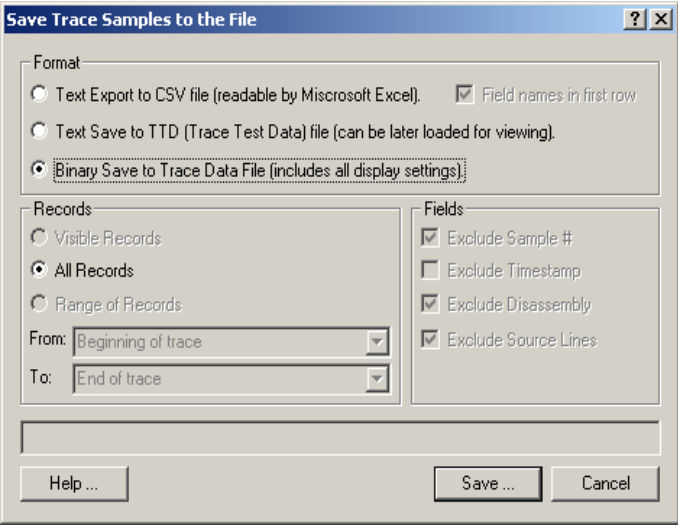


FIGURE 11 Saving the trace in binary form.

Text Saving

Saving the trace as a text file allows you to customize the range of records and the fields being saved (Figure 12).

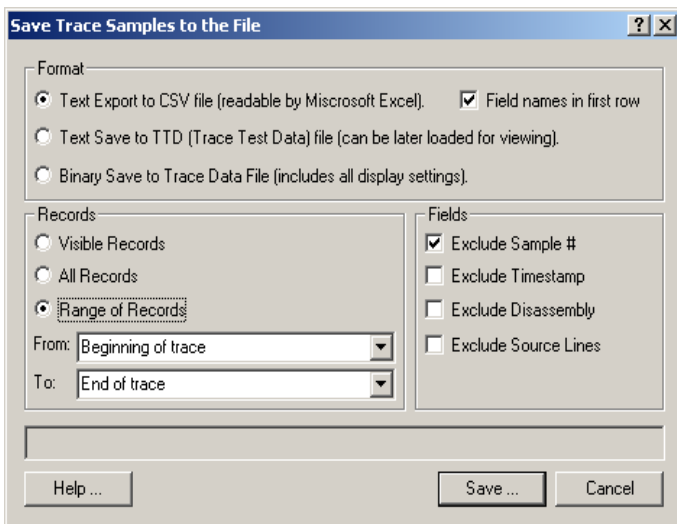


FIGURE 12 Save Trace in Range

Specifically, you can save

- **Visible records**—only samples shown on the screen are saved.
- **All records**—caution is advised: the option may create very large files.
- **Range of records**—saved are the samples between the selected starting and ending samples.

Saving in text form preserves only the fields actually displayed on the screen. The Field group allows you to exclude certain fields.

The save operation takes into account the current filtering settings; only those samples that meet the filtering conditions are saved.

ETM Trace Commands

The ETM Trace commands (Table 10 through Table 13) can be run in the Commands windows. They fall into one of the following categories:

- general commands
- trace capture commands
- on-chip ETM control
- Low Level Access to ETM Control Registers

To read the latest ETM trace documentation, type `help trace` in the Command window.

General Commands

C O M M A N D	D E S C R I P T I O N
trace	Display info, status and all settings
trace help	Help of trace commands (this description)
trace status	Display trace collection status
trace info	Display trace module info
trace disable	Disable trace module (as non-existed)
trace enable	Enable trace module
trace clear	Clear trace buffer
trace reset	Set trace subsystem
trace stop	Stop tracing now

COMMAND	DESCRIPTION
trace start	Start tracing (use new buffer if not in accumulate mode)
trace resume	Resume tracing (add to existing trace buffer)
trace trigger	Force trace trigger now
trace refresh	Force refresh of all trace windows and decode again
trace update	Update all trace settings (re-enable the trace)
trace calibrate = n	Allows entering trace calibration commands. Most Philips LPC21xx devices use n=3 or n=4. Other ARM devices do not need calibration.

TABLE 10 General ETM Trace commands.

Trace Capture Control

COMMAND	OPTION	DESCRIPTION
trace ctrl		List all trace control settings
trace ctrl	default	Set default control settings
trace ctrl	<name>=<value>	Set control parameter[s]
	stop=cputop	Stop trace collection on CPU stop
	stop=full	Stop trace collection on trace buffer full condition
	stop=exttrigger	Allows using external trigger as trace stop trigger
	stop=etmtrigger	Allows using ETM trigger frame as trace stop trigger
	stop=trigger	Allows using any trigger as trace stop trigger (external or ETM)
	triggerpos=NN	Sets trigger position (0-100 as % of trace buffer)
	accumulate=on off	Toggle control accumulate mode
	manual=on off	Sets manual start/stop option

TABLE 11 Trace capture control commands.

On-Chip ETM Control

C O M M A N D	O P T I O N	D E S C R I P T I O N
trace etm		List all ETM-specific settings
trace etm update		Update (re-program) on-chip ETMsettings
trace etm	<name>=<value> ... [/option]	Set ETM-specific parameters
	pc=start <addr>	Start ETM trace frames at PC=<addr>
	pc=stop <addr>	Stop ETM trace frames at PC=<addr>
	pc=from <addr>	Define PC range starting address
	pc=to <addr>	Define PC range ending address
	data=none	No data tracing
	data=addr	Trace only data addresses
	data=value	Trace only data values
	data=all	Trace data addresses and values
	data=from <addr>	Define PC range starting address
	data=to <addr>	Define data address range ending address
	portsize=4 8 16	ETM port size (4/8/16 bits)
	halfrate=on off	Half rate clocking option
	/force	Force settings (suppress performance warnings)
	/noupdate	Do not update settings now

TABLE 12 On-chip ETM control commands.

Low Level Access to ETM Control Registers

C O M M A N D	O P T I O N	D E S C R I P T I O N
trace etm reg		Display values of extra ETM registers being set
trace etm reg none		Do not set any extra ETM registers
trace etm reg	<regaddr>=<value> ...	Set additional trace register[s] to specified values
	<regaddr>	Register address (use 0x prefix for hex number)

COMMAND	OPTION	DESCRIPTION
	<value>	Register value (use 0x prefix for hex number)

TABLE 13 ETM Trace commands: low level access to ETM control registers.

These commands are for experienced users only. If certain registers are defined, these registers will be set to the specified values after all other trace commands (ETM Control GUI) are processed. Extra registers are set in order, in which they were defined.

Using JTAGjet-Trace with Signum EVB-LPC2138 Board

This chapter describes how to use JTAGjet-Trace with the Signum EVB-LPC2138 board. The board contains Philips LPC2138 ARM device which is based on the ARM7TDMI-S core.

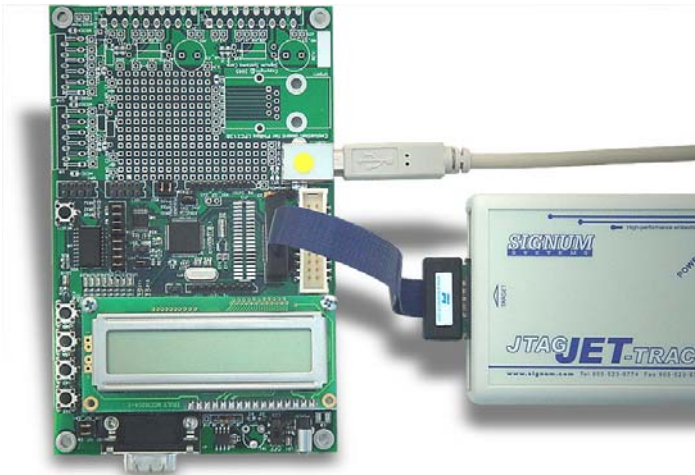


FIGURE 13 JTAGjet-Trace connected to the EVB-LPC2138 board.

Usually, all Signum ARM boards have the ETM connector installed on them. If not, you must do it yourself. The ETM connector is made by Tyco Electronics; the part number for a standard PCB (0.062 thickness) is 767054-1. These connectors are available from Signum Systems and from any Tyco distributor.

JTAGjet-Trace Hardware Setup

It is assumed that the JTAGjet-Trace software and USB drivers were already installed. If not, install them prior to connecting to the target board.

1. Verify that the following jumpers are installed:

USB Power = ON	(Enables the USB port to power the board – factory default)
JTAG_EN1 = ON	(Enables the JTAG port on Reset – factory default)
TRACE_EN1 = ON	(Enables the ETM port on Reset – factory default)
P0_8 to 15 = all ON	(Connects LED2 –LED9 to P0.8 – P0.15, respectively – factory default)

2. Make sure target board is not powered and connect the JTAGjet-Trace to the ETM connector on the board using the blue ETM cable.
3. Connect power-supply to the JTAGjet-Trace. The POWER LED indicator should come ON at this time.
4. Connect the USB cable between the PC and JTAGjet-Trace. The PC should automatically recognize the JTAGjet on the USB port. Make sure your PC is equipped with the USB 2.0 port.
5. Always power-up the target board last. The board comes with a USB cable that provides enough power to the board. Alternately, there is a 5V DC power jack on the board. If the Power LED does not come on, check the board's SW1 power switch, which is located near the POWER connector. The JTAG LED on emulator should turn ON at this time, indicating that target power has been detected and the hardware is ready to be used.

SIGNUM SYSTEMS

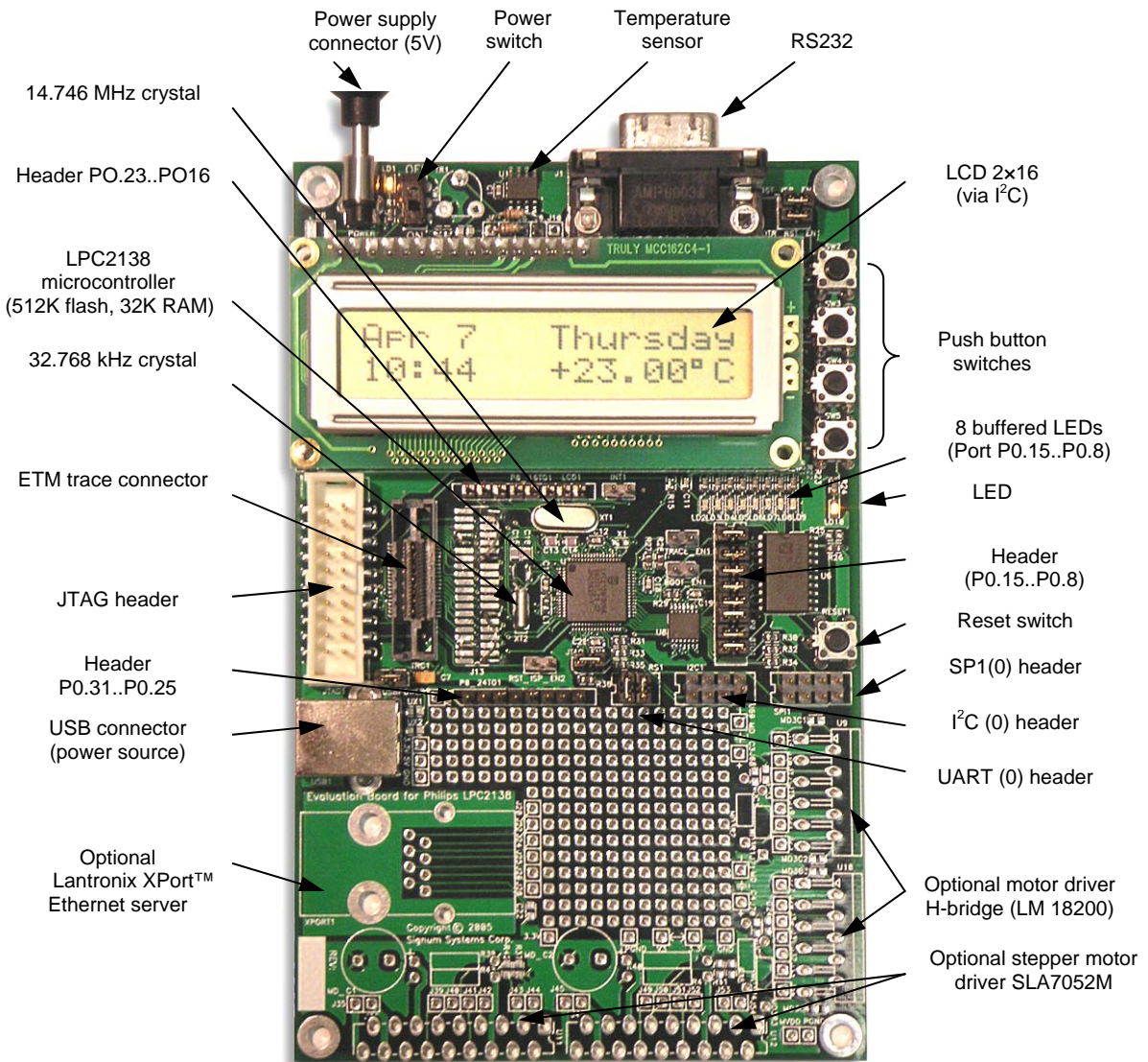


FIGURE 14 The front view of the EVB-LPC2138 board. Hidden under the LCD display are an audio amplifier and mini speaker, 3V lithium battery holder, as well as PC9555 I2C I/O port for servicing the LCD, LEDs and push buttons.

Note: For best performance, the use of a USB 2.0 port is strongly suggested. Complete trace buffer may consist of up to 18 MBytes of trace data, while high-speed transfer is essential to avoid delays when reading the trace. USB 1.1 is not recommended as it will work 40 times slower than the USB 2.0.

Chameleon Debugger Setup

1. Execute **Chameleon** debugger from the Windows Start menu. Open the **View-System Configuration** menu and click on the **Add Target** button to create a new target in the debugger. In the Target Selection dialog box select the **ARM** family and enter a more specific name in the **Target Name** field (e.g. EVB2138). Target names should describe boards, rather than devices—it is not uncommon to work over a period of time with various targets with the same CPU. Click **OK**. Next make sure that **Connect to an Emulator Automatically** is selected and click **Next**.

Note: in the Startup Configuration Selection window under Philips select the LPC2138 and click OK.

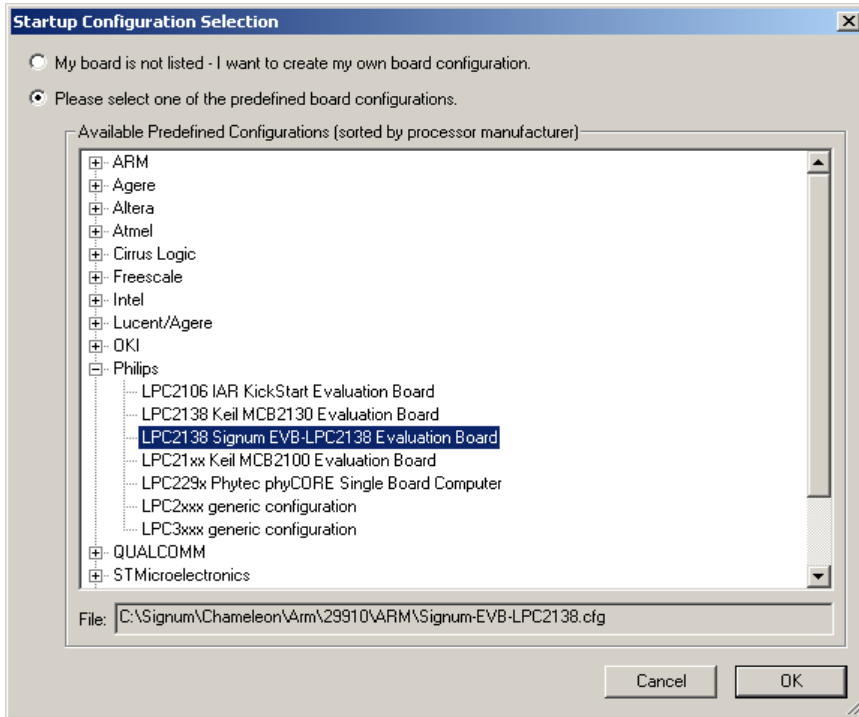


FIGURE 15 ARM core selection for the EVB-LPC2138 board (ARM7TDMI-S)

Next, in the Startup Configuration Selection window under Philips select the LPC2138 and click **OK**.

Setup Verification

1. The debugger will try to connect to the target board at this time. The **SYSLOG** window appears, showing the progress of the operation, and displaying the version of the JTAGjet-Trace along with the size of the trace buffer (Figure 16). Click **OK** to close the System Configuration setup window. The Syslog window displays now the capacity of the ETM trace buffer. Verify that it matches the size of trace you ordered. The trace sizes vary from **256K** to **4M** and signify the

trace depth. The trace width is fixed as **36-bit wide** so that a compressed time stamp can be added to each trace sample.

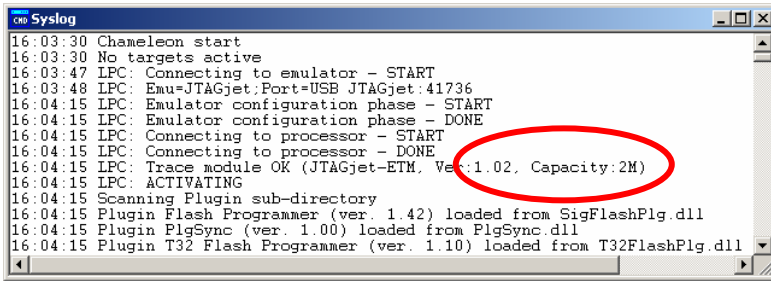


FIGURE 16 The Syslog window. “Ver: 1.02” denotes the revision of the emulator’s trace hardware, and “Capacity 2M” signifies a 2M deep (36-bits wide) buffer for a total of 9 MBytes of trace data).

- To finish the installation, bring up the CPU status window by clicking the CPU button on the toolbar.

It is a good idea to place this window in the upper right hand corner of the debugger screen or another unobtrusive location. The Status window contains all core registers, PC, Stack Pointer and the flags. It also shows if the CPU is running (pulsating green circle) or stopped (red STOP sign).

Creating Macro Buttons

Macros are script files written in the debugger’s command language to automate the debugging and testing processes. Macros may contain only a few commands that, for example, initialize the board to a known state and load the user’s code. More complex macros often comprise hundreds of commands that in turn may call other macros and

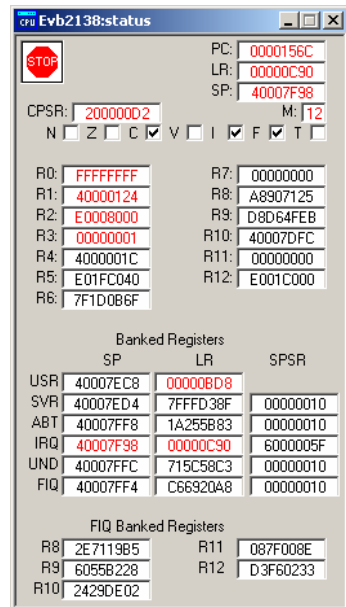


FIGURE 17 The Status window.

perform such functions as NAND Flash formatting or detailed testing of the target board.

Macros can be created off-line via editor, or live with the use of the Command Window. Existing macro files may be executed in the Command window. Additionally, it is possible to create a button (on the tool bar) that with a click of a mouse runs a specific macro.

To create a macro button, click the MACROS button and select the desired macro from the list and Click OK. A new macro button appears on the toolbar.

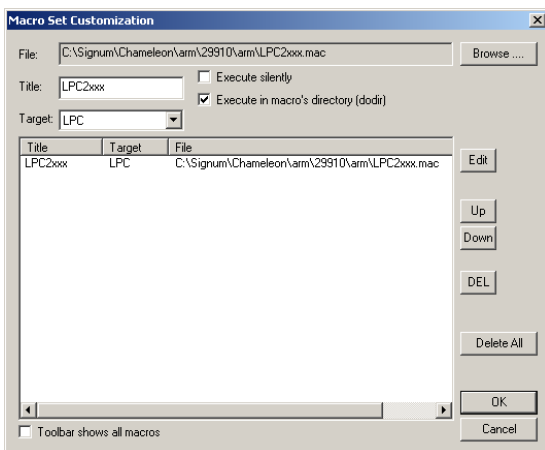


FIGURE 18 Attaching a macro to a Macros toolbar button.

The Macros toolbar should look like as shown in .

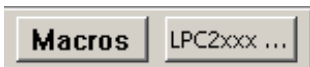


FIGURE 19 Completed Macros toolbar.

The LPC2xxx.mac is the macro used by the debugger to configure the board and enable the ETM trace. We will need to use this macro every time the CPU is Reset or powered-down.

The Set_PLL option in the LPC2xxx macro is a good example of how macro commands can be used interactively. When activated, it asks the user to type the

new CPU speed in MHz, after which it tries to adjust the LPC2138 PLL registers to get as close to the required frequency as possible.

Working with ETM

1. Normally, after starting the Chameleon debugger, the ETM trace is disabled by default, since many devices share, i.e. multiplex, the ETM pins with other I/O functions. However, the startup macro we selected contains the **trace enable** command that enables the ETM trace on startup. To verify this, open the **Trace** window from the View | Trace menu. The status is shown in the bottom left corner. Open ETM Control by clicking the **Control** button and the **Enable** check box if the status is “Disable.”

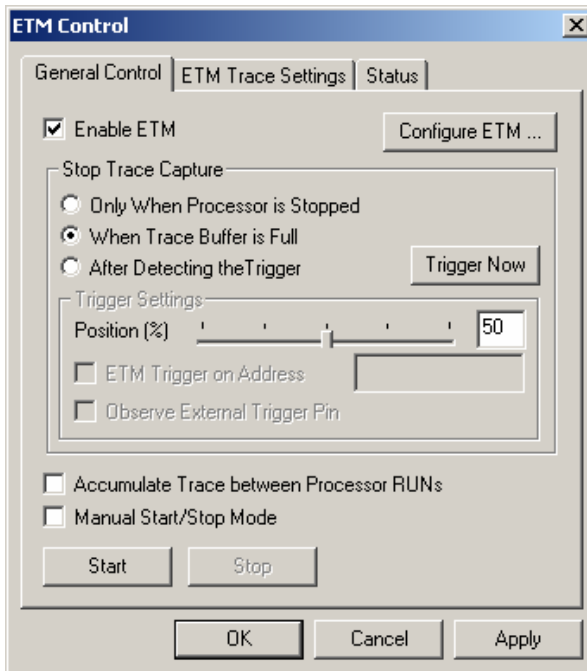


FIGURE 20 The ETM General Control tab.

2. Select the Status tab to display information related to the ETM resources embedded into the target ARM device. Please read the contents of the information dialog box and note the description of the ARM device's ETM resources (Figure 21). These resources can vary substantially from one ARM device family to another. Oftentimes, locating such information in the device's data sheet is cumbersome.

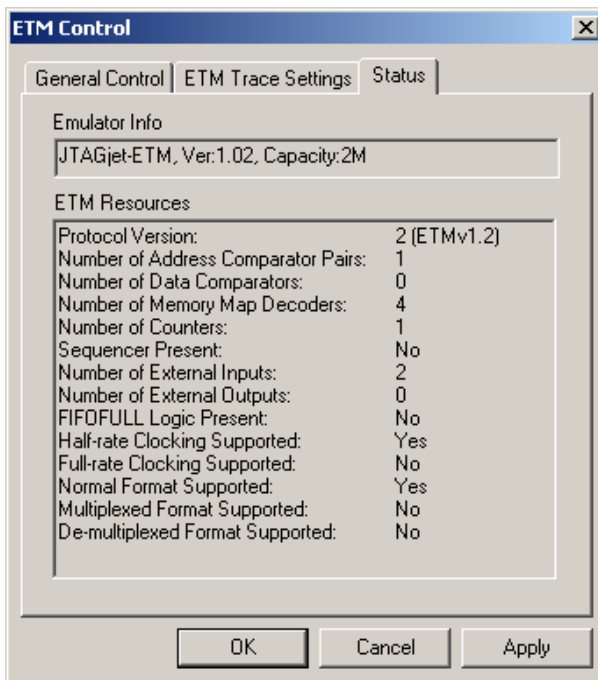


FIGURE 21 ETM Status tab showing the LPC2138 ETM resources

3. Select the **ETM Trace Settings** tab to make sure that the **Capture Complete PC Trace** is enabled and that the Data Tracing check boxes are cleared (Figure 22). Click **OK** to complete the trace configuration setup.

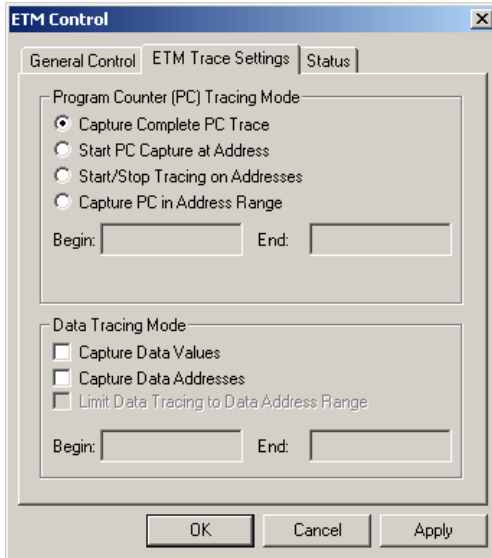


FIGURE 22 ETM Trace Settings tab.

Once the ETM is enabled, the Trace window will be cleared. The bottom status line shows now the current status of the trace; the Trace Clock frequency is approximately 29.49 MHz (Figure 23). The displayed frequency should be the actual CPU clock frequency set by the application residing in the Flash. If you see a 14.74 MHz it means the device is running at the default speed. This is possible when the flash is erased, or another application is programmed into the flash that does not change the PLL clock registers.

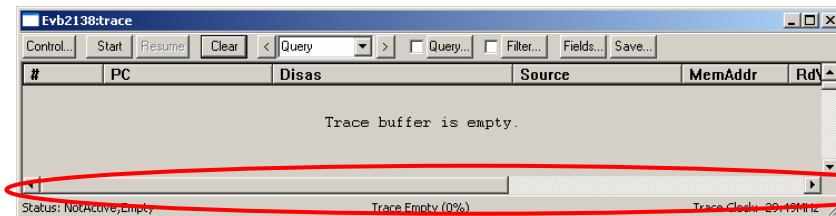


FIGURE 23 Trace Display Status bar showing activity, % usage and Trace Clock frequency

4. Another way to change the CPU speed is with the use of a macro. We have a button for the LPC2xxx macro, click on it now Select SetPLL from the menu. The Command Window appears, prompting you to enter the new CPU frequency in MHz. Once entered, the macro will calculate and set the needed CPU PLL registers to be as close to the entered frequency as possible.

If the Trace Clock frequency in the Status bar displays 0, there is a problem with the target board, or the ETM connector is not plugged in all the way, or the jumpers are not installed properly. Please make sure the frequency is correct before continuing with the tutorial.

Keep in mind that whenever the board is reset or powered-down, the ETM Trace will need to be re-enabled manually or by running the LPC2xxx.mac macro from the toolbar.

5. The Chameleon installation includes a small demo program for the Philips LPC238 devices. The program is written in C and performs simple functions and LED blinking. To load the demo program (Figure 24), select File | Load from the main menu and select the demo.elf file located in the folder:

C:\...\Chameleon\Arm\Demos\BoardSupport\Philips\Signum-LPC2138\Demo.

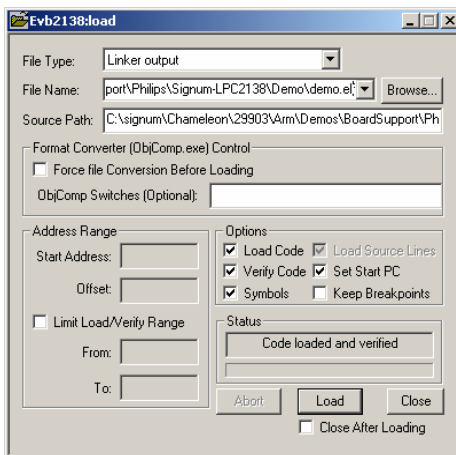


FIGURE 24 Load File dialog box

6. Select a file and click on the **Load** button. Verify the “**Code loaded and verified**” is displayed in the Status field. After the code is loaded, click **Close** to close the Load window. If the program did not load correctly, make sure you selected the demo.elf file from the correct directory. Open the Source window by clicking the **SRC** button. The Source Window now should be displaying the beginning of the startup code (PC = 0x40000000) written in assembly. To make the code comments visible, set the display mode to Source. You can change the Source Window display mode anytime by clicking the right mouse button in the Source Window and choosing the **Display Mode** menu.
7. To capture the execution of the startup code only, open the **Source** window, and position it right above the Trace window. Then scroll down to line 113 (BL main) and insert a breakpoint by clicking on the leftmost column in line 113. A red dot and a letter S will appear, signifying it is a SOFTWARE breakpoint. Now just click the green **GO** button on the tool bar to start program execution. When the break stops execution, the trace will display the end of the trace buffer, showing the last several executed instructions.

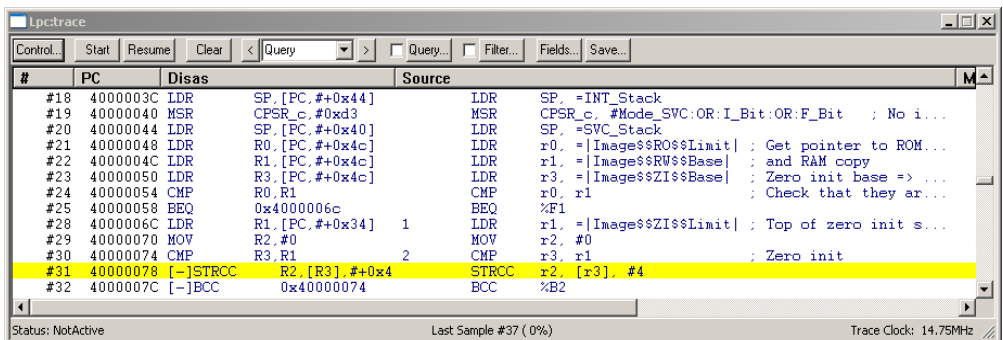


FIGURE 25 Trace Window showing the end of assembly startup code

8. **Synchronization with the Source Window** To see the trace buffer in the context of the source code click on the last sample in the trace (the yellow bar indicates the current sample) and automatically a similar yellow bar is placed in the Source window pointing to the line of code responsible for that trace frame. Using the arrow keys on your keyboard, you can scroll the yellow bar up the

trace buffer row by row and see the corresponding source code synchronized with the trace all the way to the beginning of the code.

To display symbolic addresses in the trace PC column, click on the PC column header and add checkmark next to the **Symbols**. All symbolic references are colored green for easy spotting.

9. To fill the entire trace buffer with trace history, click the trace **Control** button and make sure the **Stop Trace Capture When Trace Buffer is Full** option is selected. Continue running the program by selecting **GO** again. The trace buffer will accumulate a large amount of samples before it gets full. At that point, the trace window will stop acquiring more data and begin displaying the captured information. Press the red **STOP** button on the toolbar to stop the CPU. This will allow the debugger to update the open memory and register windows.
10. **Trace Filtering** The ability to hide unwanted information from the trace buffer to get a clearer picture cannot be overestimated. Trace filtering is easily accomplished by double-clicking in any column in the Trace window with the exception of the Time Stamp column. For example, to see only trace frames that contain C source lines, double-click inside any Source line column. In the menu that appears, select “**Filter non-empty Source**”

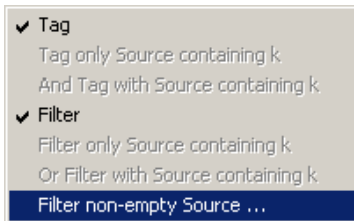


FIGURE 26 Filtering lines with non-empty source lines

The Trace window will filter out all frames without C source associated with them. With only lines of C code displayed, program execution will be easier to analyze.

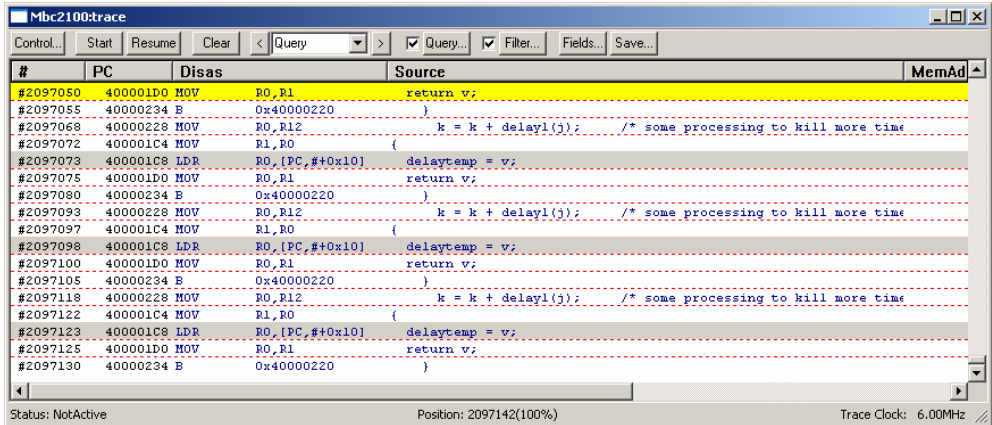


FIGURE 27 Filtering to show only the rows with C code

11. **Using Query** To return to the unfiltered view, uncheck the Filter check box on the toolbar. Click on the Query drop down list and change the query to look for the next **Module**. Keep pressing the forward [>] button until **processing (Buf1)** is found. Double click on the **PC** column of that trace line and from the menu select “**Filter only PC = main+0x6C**” (PC = 0x40000148). This will change the filter to show only calls to **processing**. This will allow us to measure the frequency of the calls and determine if the interval between calls is constant.
12. To find out how often the **processing** function is called, scroll the trace window to the right to see the **TStamp** column and double-clicking on it until it shows the time in delta mode and cycles [dt][cyc]. At the end of the process, we find that the call takes place every 32,708 CPU cycles. To convert this value to microseconds, click on the TStamp header and select the **usec** from the list.

Now double click anywhere in the time stamp column to change the display to [abs][us]. Scroll to the end of the buffer to see how much execution time is covered by the contents of the trace buffer. A 256K buffer, the smallest available, is capable of storing over 11 msec worth of a program running at 30 MHz. With the largest trace buffer of 4M, this capability is 16-fold larger.

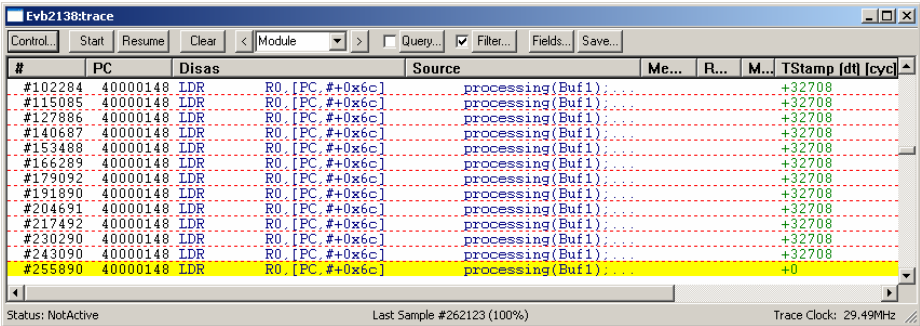


FIGURE 28 Filtered trace showing calls to only one function.

13. As you probably noticed, our experiment filled the trace buffer in an instant. This is because the buffer stored the samples indiscriminately without eliminating such uninteresting fragments of the application as loops waiting for ready bits or software delays. This can easily be amended by pre-filtering the data that comes into the trace buffer so that only the relevant information is retained. One way to do it is to set the trace to capture execution in PC address range that interests us.

Please uncheck the **Filter** to see all samples, and then click on the **CLEAR** button to erase the trace buffer. Now open the Control dialog box and select the **ETM Trace Settings** tab.

In the Settings tab, select the **Capture PC in Address Range** setting and enter the following addresses:

Begin: {"demo.c"}@29 End: {"demo.c"}@58

These settings will capture execution trace of the main.c program loop only, and any function calls outside of main will not be recorded. This is a great way to catch the profile of the application as you can see clearly the timing of the major system functions.

Click **OK** to save the settings. Click also the yellow **RST** (restart) button on the top toolbar. The RST button brings the application back to the initial point without resetting the CPU or reloading the application.

Click the **GO** button. Notice that the trace buffer fills much slower this time, as it needs to make selections in order to store only the lines in the main loop.

When the trace fills up and stops, stop the CPU and hit the Home button on your keyboard to see the beginning of the trace. The window should look similarly to that in Figure 29.

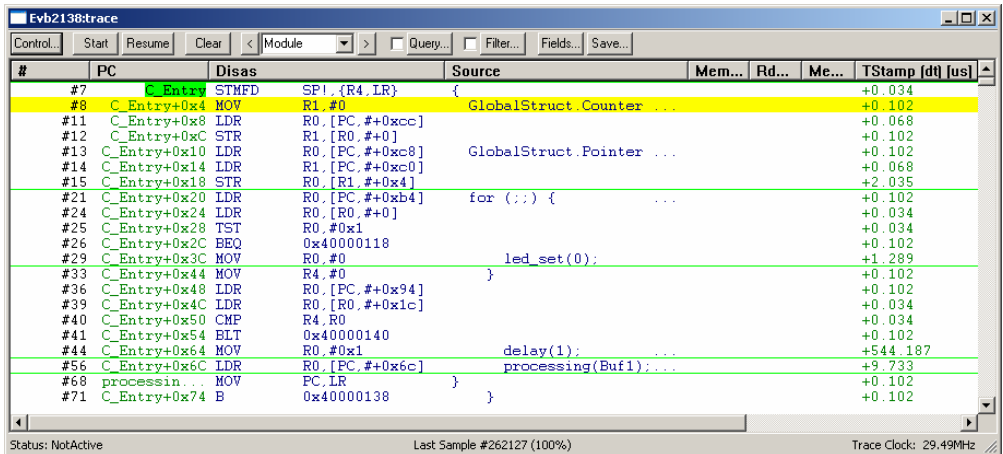


FIGURE 29 Filtering to show calls to one function only.

The green lines in the Trace window represent trace discontinuities. As expected, the trace started tracing from the first line of the main program (**main**) and stopped on C_Entry+18, which is there where the first function (**led_init**) was called. We find that this function stopped the trace for 2.035 μ s and started to capture again when it returned to the main program to execute the **for (; ;)** loop. The trace captured a few lines from main and stopped again when **led_set(0)** was called, which took 1.289 μ s. We can also see that the intended 1 ms **delay** function takes actually 544.187 μ s. If this timing is important, the developer may need to adjust this value.

- To eliminate any function from the trace, we must use the **Start/Stop Tracing on Address** feature. Also, it is necessary to enter the function's exit address in the trace **Begin** field and the function's entry point in the trace **End** field. Figure 31 shows an example with the delay functions filtered out, allowing the trace to capture over 622 msec of the run time.

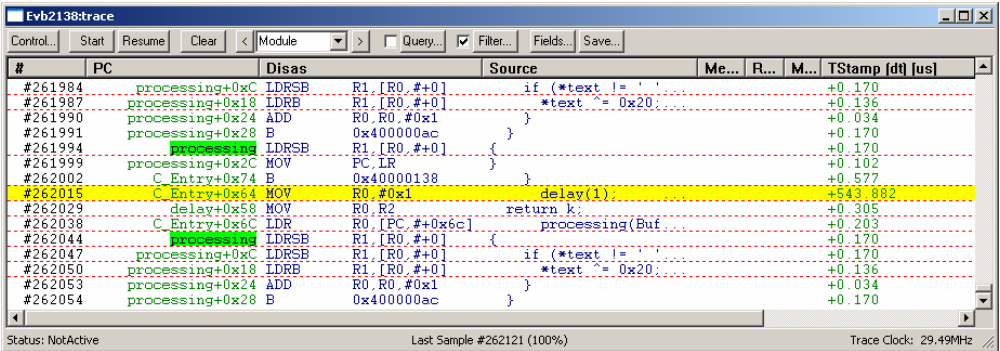


FIGURE 30 Trace with the delay function filtered out.

Data Tracing with Philips LPC

Tracing of code execution is an excellent method for detecting many programming errors. However, without an ability to visualize data transfers, it may be hard to see what the application is actually doing. In typical C code, most problematic places are prologue and epilogue of functions where several 32-bit CPU registers are transferred to or from the stack with the STMIA and LDMIA instructions.

The Philips LPC21xx data sheets do not mention the capability of data tracing over the ETM, which may suggest that it is not supported, most likely due to the lack of bandwidth on the 4-bit ETM.

Nevertheless, our experiments show that if data access is not very frequent, some data variables may be successfully traced even on a 4-bit ETM port. Frequent variable access will overflow the on-chip ETM FIFO, causing loss of PC information. Therefore tracing variables may require a certain amount of caution.

In the ETM Control dialog box, you can choose to trace the variable(s) address or data or both. Since each variable address is 32-bit long and most data contain 32-bit values, up to sixteen clock cycles may be used just to export each variable. Thus, whenever possible, using byte variables will save a lot of bandwidth when tracing.

To trace what is read and written to the `Buf1[]` array, click on the Control button, select the ETM Trace Settings tab and make sure the Capture Complete PC Range is selected (Figure 31).

In the **Data Tracing** section right below, check all three boxes and enter the addresses **&Buf1[0]** and **&Buf1[14]** in the Begin and End fields, respectively.

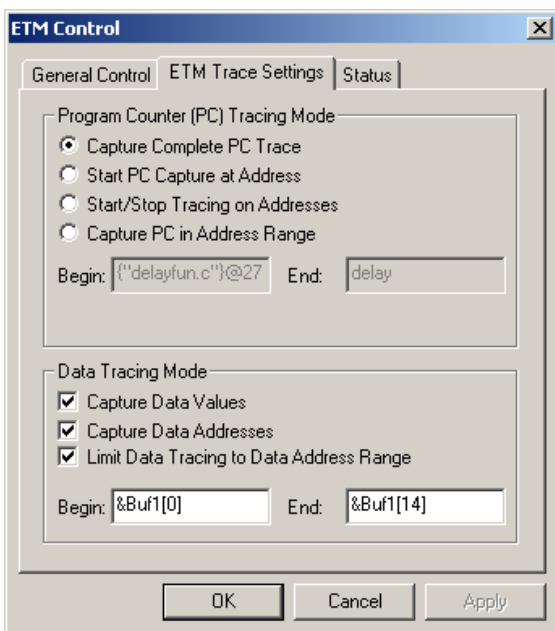


FIGURE 31 Adding tracing of variables to the ETM settings

Click **OK** to close the Control window. If the CPU is still running, stop it by pressing the **STOP** button.

Open the Command window (e.g., by pressing the CMD button on the tool bar) and type,

```
>delaycnt = 1
```

This instruction reduces the amount of the software delay to a minimum, so that the Buf1[] array is written more frequently and more of it will be included in the trace.

To restart trace capture when the CPU is running, press the **Start** button. If the CPU is stopped, press the **GO** button.

The trace will fill instantly and display new data. To find the data variables in the trace, double click in the MemAddr or MemData fields and select Filter non-empty MemAddr. This will only display the trace frames with valid MemAddr fields. In our exercise, only the Buf1[] writes and reads and the associated data values are shown (Figure 32).

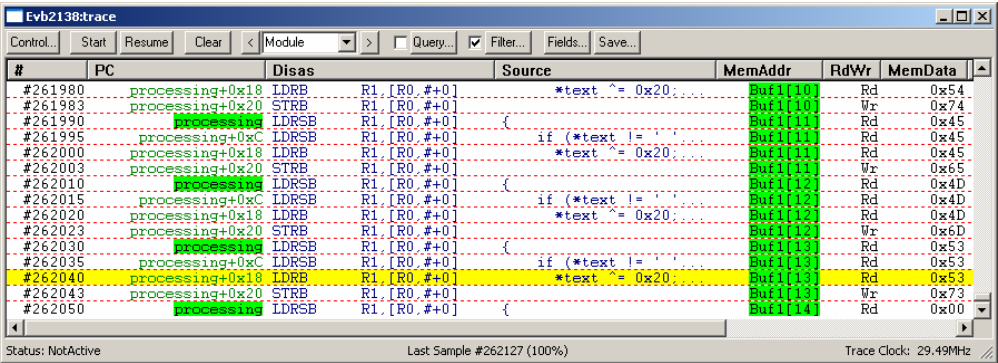


FIGURE 32 Filtered trace showing only read and write to the Buf1[] array operations.

To have the MemData column display in ASCII mode, click in the column header and check ASCII from the menu. When closed, the trace will show ASCII characters in place of HEX values, making reading the contents of the buffers containing text strings easier for a human reader.

In our example, switching to ASCII format reveals that the processing() function flips the case of the Buf1[] string (lower to upper or vice versa). See Figure 33.

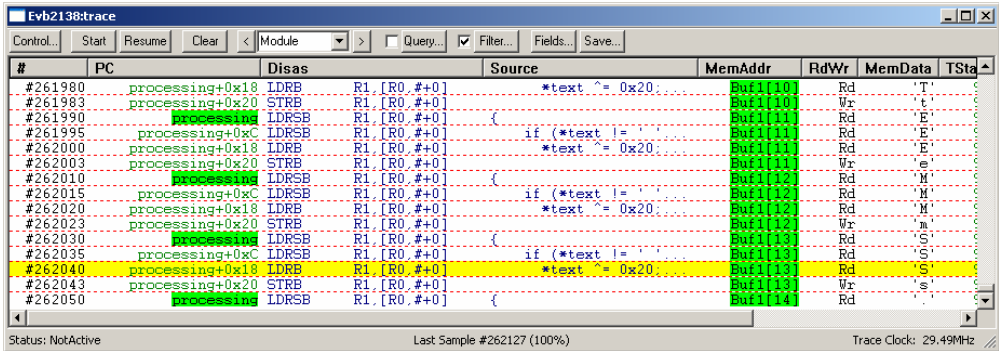


FIGURE 33 Filtered trace showing variable data in the ASCII format.

Changing Variables During Execution

Even though unrelated directly to the ETM trace, you may find this section helpful. The information contained herewith is not easily located in the debugger documentation.

If our application runs in real-time, the blinking of the LEDs will not be perceptible, because we set the value of the delaycnt variable from 1. Let us return to the original value of 500 while the application is running.

Close the Trace window and press the **GO** button. The application starts running. Open the **Watch** window by clicking on the **WCH** icon on the toolbar. An empty Watch window appears. Right-click inside the window, then select **Add** from the pop-up menu. In the dialog box that appears, type the name **delaycnt**.

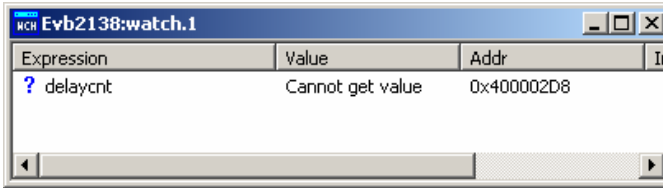
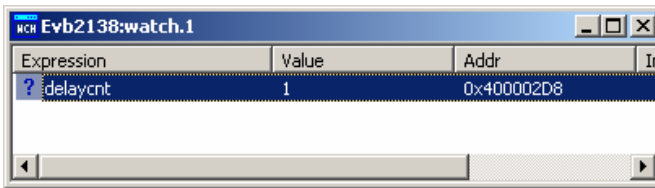


FIGURE 34 Filtered trace showing variable data in the ASCII format.

The variable `delaycnt` appears in the Watch window, but the value will not be read since the application is running (Figure 35). To force the reading of its value, select the variable in the Watch window and right-click again. Check the **Access in RUN mode** feature.

Now select the variable again with the mouse and press the Space bar on your keyboard. This will force the variable to be read showing the value of 1 (Figure 35).

FIGURE 35 Selecting the `delaycnt` variable in the Watch window.

To change the variable, double click on its value in the Watch window and enter 500 in the entry field. The blinking of the LEDs will now be visible to the eye.

Variables can also be drag-and-dropped to the Watch window. To try this capability, we need to stop the CPU. Right-click in the Source Window and select **View Module** in the pop-up menu. When a list of the modules appears, select **demo.c** and click OK.

The Source window will show the beginning of the `demo.c` file.

To be compatible with older Signum emulators, a double-click on any line in the Source window triggers the option **Execute to this line**. This default setting may be changed by opening the Source window options menu with a right-click, and then selecting **Options** from the list and un-checking **Double click == GO TILL**.

With this change, a double-click on a source line will select variables. Click on any GlobalStruct variable and drag it into the Watch window (Figure 36).

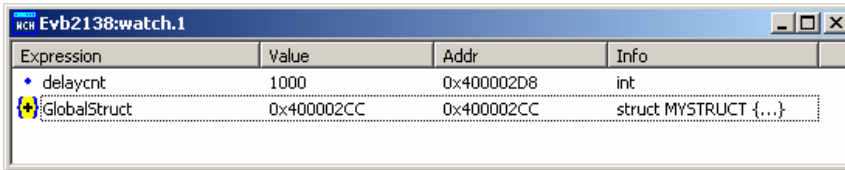


FIGURE 36 Selecting variables in the Watch window.

To expand the structure, click the yellow + next to the variable's name and make sure it is accessible in RUN mode, just as we did with the delaycnt variable.

Press **Go** to run the application again. You can now inspect the members of the structure while the application is running by selecting them in the Watch window and pressing the space bar (Figure 37).

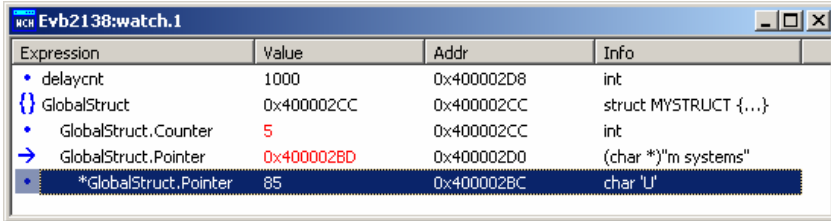


FIGURE 37 Selecting a member of a complex variable for inspection.

Using JTAGjet-Trace and Keil MCB2100 Board

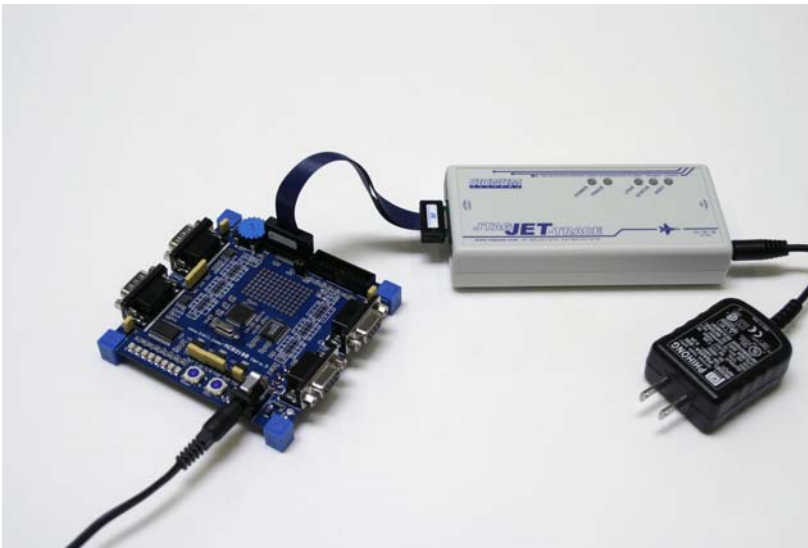


FIGURE 38 JTAGjet-Trace connected to the MCB2100 board.

This chapter describes how to use JTAGjet-Trace with the Keil MCB2100 board. The board contains Philips LPC2129 ARM device which is based on the ARM7TDMI-S core.

Usually, these boards do not have the ETM connector installed, so you must do it yourself. The ETM connector is made by **Tyco** and the correct part number for a standard PCB (0.062 thickness) is: **767054-1**. These connectors are available from Signum Systems and from any Tyco distributor.

Caution

- Keil MCB2100 Rev 2.0 (and possibly earlier versions) board has a wrong pinout of ETM connector, making it impossible to use the ETM trace module with this board. **DO NOT PLUG JTAGjet-Trace** to this ETM connector. This problem has been fixed with Rev 3.0 of the board.
- Keil MCB2100 board shares ETM pins with LEDs. If your application is blinking LEDs, these LEDs will not blink if ETM port is enabled.

JTAGjet-ETM Hardware Setup

It is assumed that the JTAGjet-Trace SOFTWARE and USB drivers were already installed. If not, install them before connecting to the target board.

15. Make sure you have the MCB2100 board revision 3.0 or later with ETM (P6) connector installed.
16. Verify that the following jumpers are installed:
JP9 = ON (enables the JTAG port on Reset – factory default)
JP8 = ON (enables the ETM port on Reset – not installed by factory)
17. Make sure target board is not powered. Connect the JTAGjet-Trace to the ETM connector on the target board using the blue ETM cable.
18. Connect power-supply to the JTAGjet-Trace. Power LED should come ON at this time.

19. Connect the USB cable to JTAGjet-Trace. The PC should automatically recognize the JTAGjet on the USB port.
20. Always power-up the target board last. The JTAG LED on JTAGjet-Trace should turn ON at this time, which indicates that target power has been detected and the H/W is ready to be used.

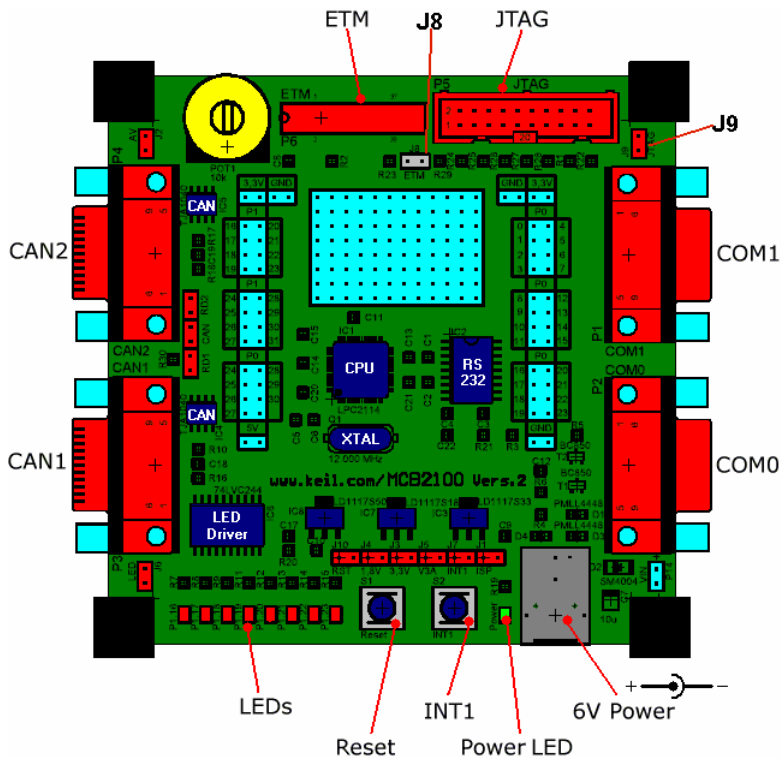


FIGURE 39

Note: It is strongly recommended to use USB 2.0 port. Complete trace buffer may consist of up to 18 MBytes of trace data and high-speed transfer is essential to avoid delays when reading the trace. USB 1.1 port will work 40 times slower than the USB 2.0.

Chameleon Debugger Setup

1. Execute **Chameleon** debugger from the Windows Start menu. Open the **View-System Configuration** menu and click on the **Add Target** button to create a new target in the debugger. In the Target Selection dialog box select the **ARM** family and enter a more specific name in the **Target Name** field (e.g. MCB2100). Target names should describe boards, rather than devices—it is not uncommon to work over a period of time with various targets with the same CPU. Click **OK**. Next make sure that **Connect to an Emulator Automatically** is selected and click **Next**.
2. In the Startup Configuration Selection window, under Philips, select the LPC21xx Keil MCB2100 Evaluation Board and click **OK** (Figure 40).

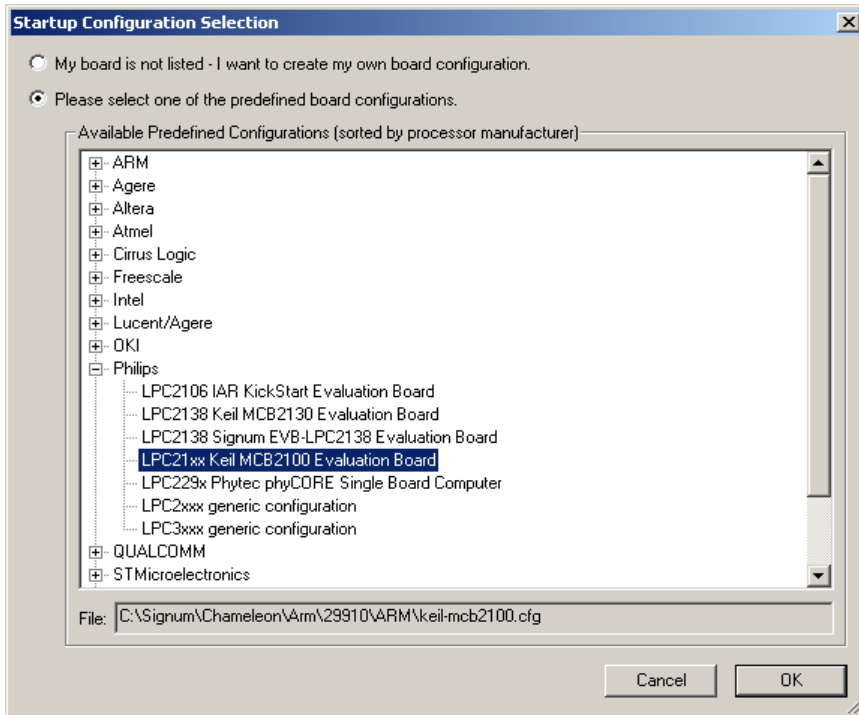


FIGURE 40

3. In the next JTAG Configuration window choose the **Only one processor connected** button and click **OK**.
4. In the Startup Configuration Options screen, scroll down and select **Philips LPC21xx Keil MCB2100 Evaluation Board** and click **OK**.

Chameleon Debugger Setup

1. Execute **Chameleon** debugger from the Windows Start menu. Open the **View-System Configuration** menu and click on the **Add Target** button to create a new target in the debugger. In the Target Selection dialog box select the **ARM** family and enter a more specific name in the **Target Name** field (e.g. MCB2100). Target names should

describe boards, rather than devices—it is not uncommon to work over a period of time with various targets with the same CPU. Click **OK**. Next make sure that **Connect to an Emulator Automatically** is selected and click **Next**.

2. Next, in the Startup Configuration Selection window under Philips select the LPC21xx Keil MCB2100 Evaluation Board and click **OK** (Figure 41).

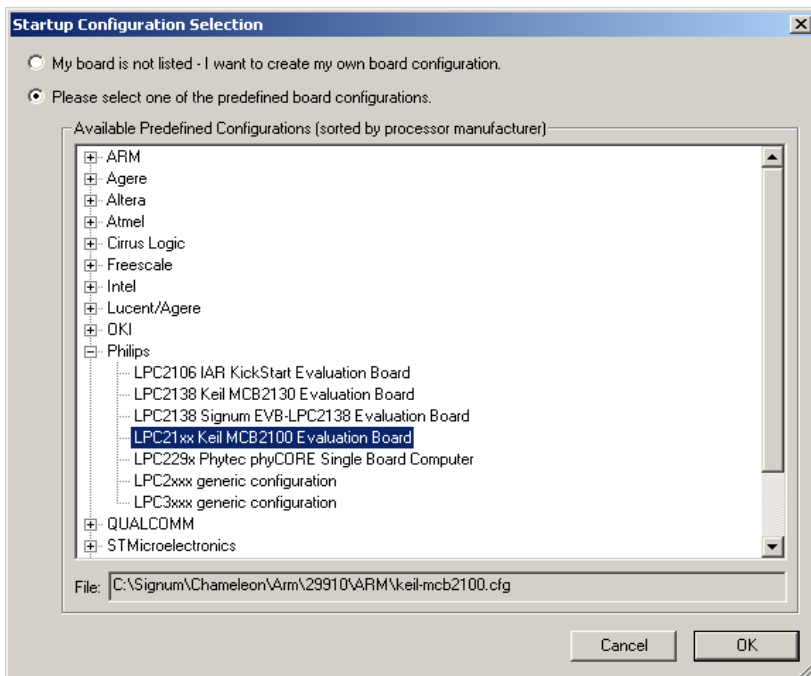


FIGURE 41 Selecting the Keil MCB2100 Evaluation Board startup configuration file

3. The debugger will now connect to the JTAGjet. The **SYSLOG** window appears, showing the progress of the operation, displaying the version of the JTAGjet-ETM and the size of the trace buffer. Click **OK** to end and close the System Configuration setup window.

To continue with the tutorial, please refer to Section *Setup Verification*, p. 28, and continue as if you were working with the Signum LPC2138 evaluation board.

Using JTAGjet-Trace with OMAP5905

This chapter describes how to use JTAGjet-Trace with the OMAP5905, but the instructions here may be applied directly to OMAP161x, DM730 and DM732. Other OMAP devices are only installed differently due to the differences in the ARM core and startup macro, but the trace usage applies directly to all OMAP devices.

In order to use the JTAGjet-Trace, you must have an OMAP board equipped with the ETM connector. The ETM connector that goes on the target board is made by **Tyco** and the correct part number for a standard PCB (0.062 thickness) is: **767054-1**. These connectors are available from Signum Systems and from any Tyco distributor.

Caution

Some OMAP target boards do not have the JTAG signals connected to the ETM connector. This is a mistake on the board manufacturer's side, and you have two options: either connect the missing JTAG lines to the ETM connector with thin wires, or use the ETM-JTAG splitter adapter and connect the JTAG cable to the JTAG header and the ETM splitter into the ETM connector (see picture below).

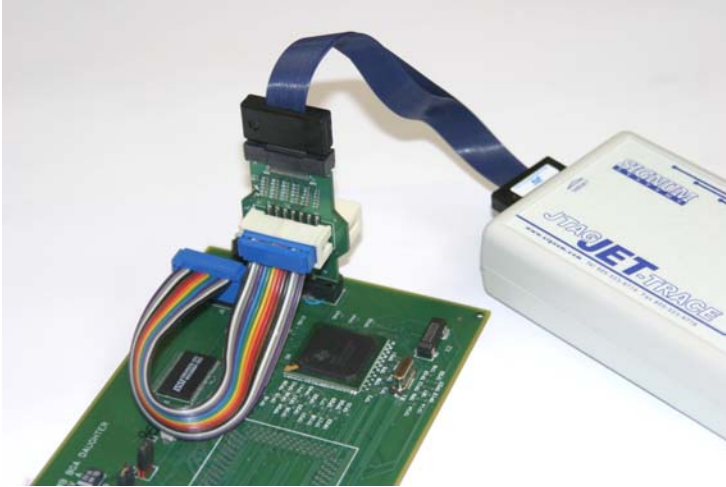


FIGURE 42 Connecting JTAGjet-Trace with separated ETM & JTAG signals.

JTAGjet-ETM Hardware Setup

1. It is assumed that the JTAGjet-Trace SOFTWARE and USB drivers were already installed. If not, install them before connecting to the target board.
2. Make sure target board is not powered, then connect the JTAGjet-Trace to the ETM connector on the board using the blue ETM cable (or using the ETM-JTAG splitter)
3. Connect power-supply to the JTAGjet-Trace. Power LED should come ON at this time.
4. Connect the USB cable to JTAGjet-Trace. The PC should automatically recognize the JTAGjet on the USB port.
5. Always power-up the target board last. The JTAG LED on JTAGjet-Trace should turn ON at this time, which indicates that target power has been detected and the H/W is ready to be used.

Important

The USB 2.0 port is strongly recommended. Complete trace buffer may consist of up to 18 MBytes of trace data and high-speed transfer is essential to avoid delays when reading the trace. USB 1.1 port will work 40 times slower than the USB 2.0.

Chameleon Debugger Setup

1. Execute **Chameleon** debugger from the Windows Start menu. Open the **View-System Configuration** menu and click on the **Add Target** button to create a new target in the debugger. In the Target Selection dialog box select the **ARM** family and enter a more specific name in the **Target Name** field (e.g. OMAP5905). Target names should describe boards, rather than devices—it is not uncommon to work over a period of time with various targets with the same CPU. Click **OK**. Next make sure that **Connect to an Emulator Automatically** is selected and click **Next**.
2. Next, in the Startup Configuration Selection window under Texas Instruments select the OMAP5905 and click **OK** (Figure 43).

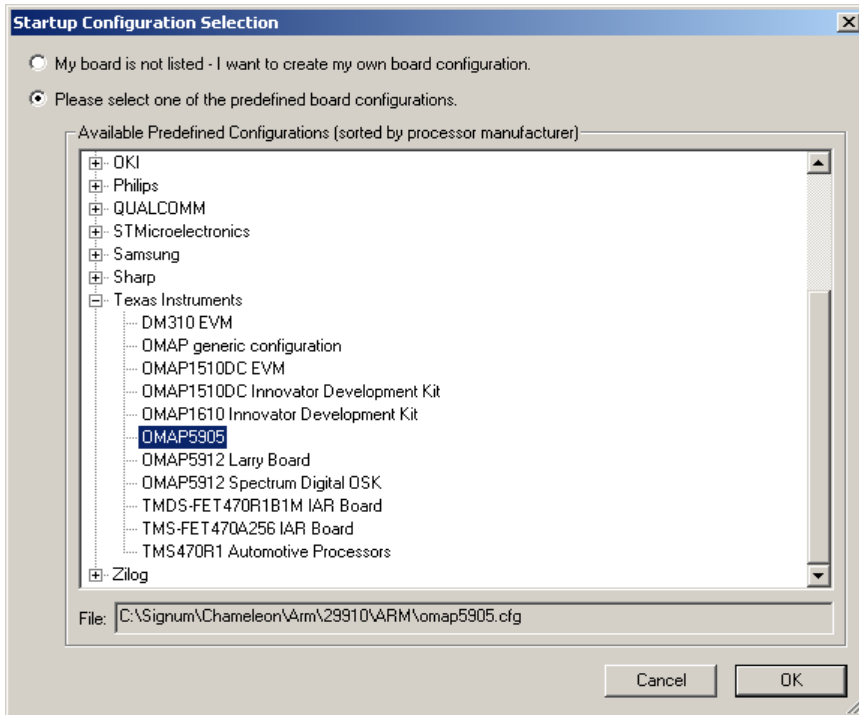


FIGURE 43

3. The debugger will now try to connect with the JTAGjet and the **SYSLOG** window will appear showing the progress as well as version of JTAGjet-ETM and the size of the trace buffer. Click **OK** to end and close the System Configuration setup window. The Syslog window will display the capacity of the ETM trace buffer so please verify that it shows the ordered amount of trace.

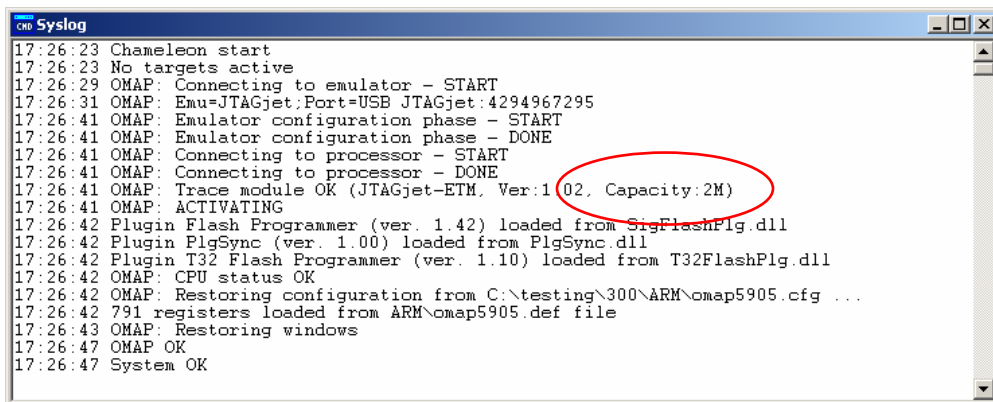


FIGURE 44

“Capacity 2M” means 2M frames deep (each frame is 36-bits wide) for a total of 9 MBytes of raw trace data.

4. After starting the Chameleon debugger, the ETM trace module will be disabled. To enable it, open the **Trace** window (in **View-Trace** menu) and click on the **Control** button. When the ETM Control dialog box comes up, check the **Enable ETM** box to enable the trace and make sure Stop Trace Capture selection is on When Trace buffer is full.

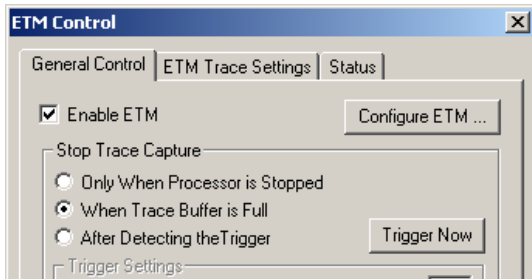


FIGURE 45

This will not allow the trace buffer to be overwritten without your knowledge.

Next click on the **Configure ETM ...** button and make sure the **8-bit** mode shows in Trace Port Size selection and **Normal, Half-rate clocking** is selected.

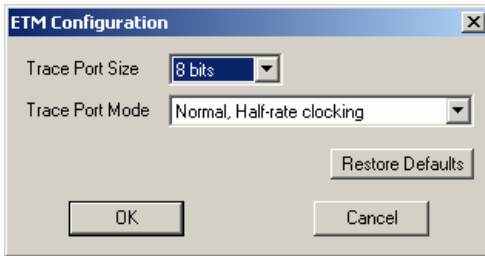


FIGURE 46

Click **OK** when done with ETM port configuration to return to the Control tab.

5. Next, select the **Status** tab, which will display information related to the ETM resources embedded into the ARM device being debugged. Please inspect the information screen as this information is hard to find in the device data sheet. It describes the ETM Resources of the device itself and these resources may differ substantially between different ARM device families.

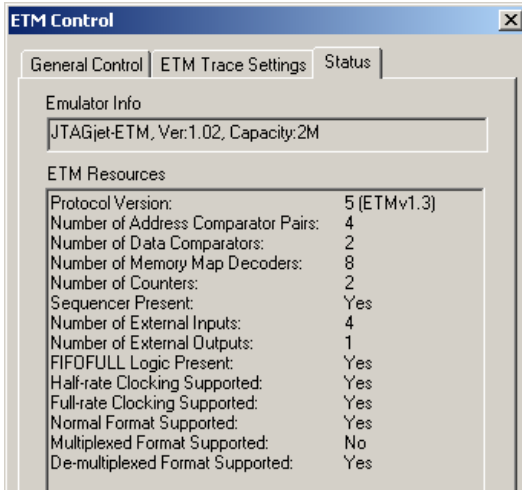


FIGURE 47

- Now select the **ETM Trace Settings** tab and make sure the **Capture Complete PC Trace** is enabled and click **OK** to finish the trace configuration setup.

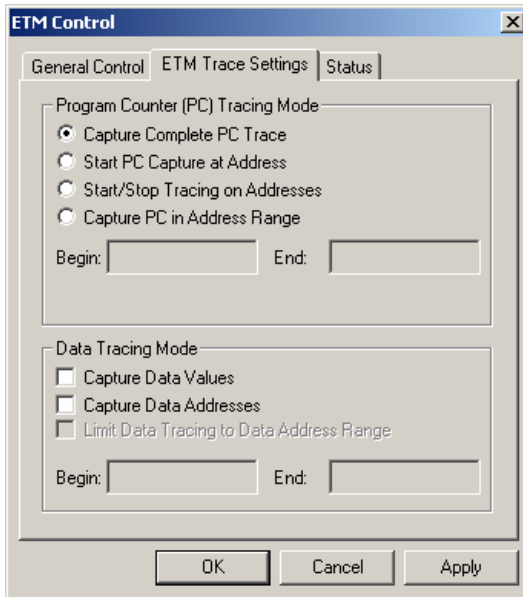


FIGURE 48

7. Once the ETM is enabled, the Trace window will be blank, the bottom status line will show the current status of the trace and the Trace Clock frequency will show **0 MHz**. This is OK because the ETM pins must be enabled on the OMAP device itself as outputs by writing into the PINCR2 register. We will do this now from the Command window and later on we will use a Macro file.

Open the **Command** window from the **View** menu or using the **CMD** icon from the toolbar and enter the following command followed by Enter:

```
>PINCR2 = PINCR2 | 0x3
```


When done, the Trace Clock frequency will show to be **48 MHz** or exactly $\frac{1}{2}$ of the **CPU clock** frequency. If the Trace Clock frequency shows 0, there is a problem with the target board or the ETM connector is not plugged in all the way.

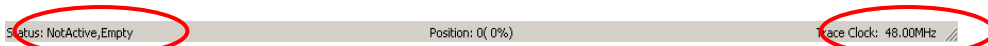


FIGURE 49

The ETM is now ready to be used. Please remember that the trace will be disabled after every exit/start of the debugger. It is possible to enable the trace via a **trace enable** command in the Command window or place the trace enable command in the startup macro file, so that trace will be automatically enabled on every debugger startup.

Using ETM Trace – Short Tutorial

8. The Chameleon installation includes a small demo program for the OMAP5905 devices. This program is written in C and performs simple text processing and writing to a LED variable, which can be easily changed to blink a real LED if available on a board..

To load the demo program, select **File-Load** from the main menu and browse for the **demo.elf** file located in the folder

...ARM\Demos\BoardSupport\TT\OMAP\Demo

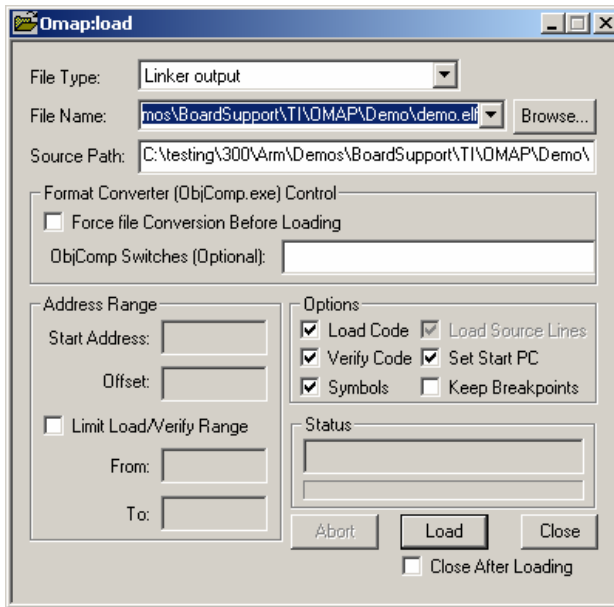


FIGURE 50

9. When the file is selected, click on the **Load** button and make sure the “**Code loaded and verified**” is displayed in the Status field. After the code is loaded, click **Close** to close the Load window. If the program did not load correctly, make sure you selected the correct demo.elf file.
10. To capture the execution of part of the startup code only, open the **Source** window, press function key **F4** to see a list of modules loaded and double click on the demo.c module. This will bring the demo.c module into the Source window, so scroll down to the beginning of the **main** loop and insert a **breakpoint** by clicking on the leftmost column in this line. A red dot and a letter **S** will appear, signifying it is a software breakpoint. Now just click on the green **GO** button on the tool bar to start program execution. When the breakpoint stops execution, the trace will display the end of the trace buffer, showing the last several executed instructions. The last entry in the Trace window should be the jump to main indicated by the MOV PC, R2 instruction.

#	PC	Disas	Source	MemAddr	RdWr	MemData	TStamp
#1202	20000920	MOVNR	PC, R12				7143939
#1203	20000924	LDMPD	SP!, {R4-R11, PC}				7144199
#1213	200003C4	LDMPD	SP!, {PC}				7144312
#1217	200002EC	LDR	R0, {PC, #0x368}				7144340
#1218	200002F0	LDR	R12, {PC, #0x368}				7144395
#1219	200002F4	CMP	R12, #0				7144409
#1220	200002F8	LDREQ	R12, {PC, #0x364}				7144437
#1221	200002FC	CMP	R12, #0				7144493
#1222	20000300	BLNE	__call_ip ; __call_ip				7144507
#1226	20000318	TST	R12, #0x1				7144594
#1227	2000031C	MOVHQ	PC, R12				7144607
#1231	2000024C	STR	LR, {SP, #-0x4}!				7144691
#1232	20000250	MOV	R2, R0				7144705
#1233	20000254	MOV	R0, #0				7144719
#1234	20000258	MOV	R1, #0				7144720

Status: NotActive Position: 1244(0%) Trace Clock: 48,00MHz

FIGURE 51

Now, to synchronize the Source with the C-level code, click on the **Step Into C Statement** button on the main debugger toolbar. This will bring up the main.c into the Source window and the green arrow will point to the next C line to be executed (delayloop = 100;). The Trace window will append one executed line at the end of the trace. The time stamp may show a large time stamp value which corresponds to the actual time elapsed between the last entry in trace and the single step operation.

11. **Synchronization with the Source Window** To see the trace buffer in the context of the source code click on any trace row (the yellow bar indicates the current sample) and automatically a similar yellow bar is placed in the Source window pointing to the line of code responsible for that trace frame. The best way to view it, is to put the Source window on top and the trace window just below it without overlapping each other. Using the arrow keys on your keyboard, you can scroll up/down the trace buffer line by line and see the corresponding source code synchronized with the trace. To continue with program execution, click on the **GO** button again and the trace buffer will capture hundreds of loops of the running program before it is full. When full, the trace window will stop acquiring more data and display the captured information again. You can change the trace capture setting to allow it to overwrite itself over and over again to catch as much as the last 4M samples before the CPU is stopped (depends on the trace depth).

- 12. Trace Filtering** To hide the unwanted information from the trace buffer to get a clearer picture of the execution is very important. The trace filtering can be easily done by double-clicking on any column in the trace window except Time Stamp.

For example, to see only the trace frames that contain C source lines, double-click inside any Source line and when the menu comes-up, select the “**Filter non-empty Source.**”

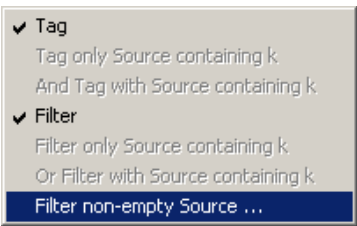


FIGURE 52

Almost instantly, the trace window will filter out all frames without the C source, so the program execution looks cleaner and more like the C source.

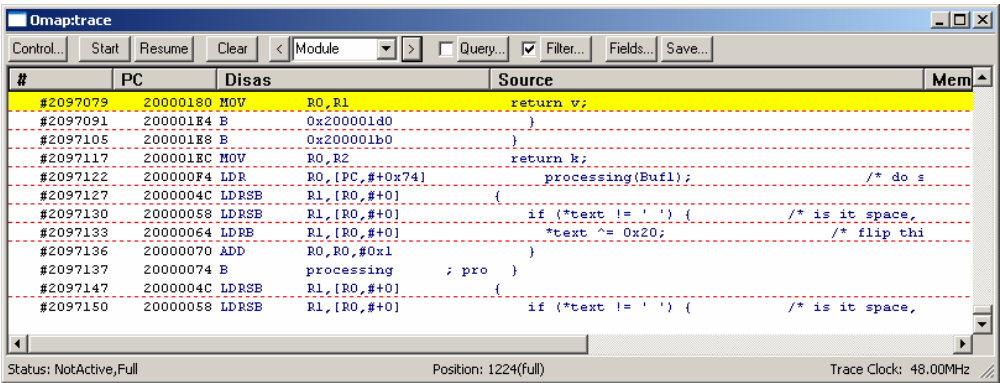


FIGURE 53

- 13. Using Query** Uncheck the **Filter** check box at the tool bar to return to the unfiltered trace and select **Module** in the Query field. To find a new module in

the trace, just click the down [$>$] or up [$<$] query buttons until **processing (Buf1)** is found. Double click on the **PC** column of that trace line and from the menu select “**Filter only PC = 0x20000F4**”. This will change the filter to show every call to the function **processing** only.

14. To find out how often this function is called, scroll the trace window to the right to see the **TStamp** column and double click on it until it shows the time in delta-cycle mode [**dt**][**cyc**]. When done, you will see that this function is called every 8,230 CPU cycles.

#	PC	Disas	Source	M...	R...	M...	TStamp (dt) [cyc]
#35308	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#35735	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#36171	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#36600	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#37036	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#37465	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#37901	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#38328	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#38764	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#39193	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#39629	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#40058	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#40494	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230
#40921	200000F4	LDR R0, [PC, #+0x74]	processing(Buf1);				+8230

Status: NotActive,Full Position: 1224(full) Trace Clock: 48.00MHz

FIGURE 54

You can convert the time stamp to microseconds by clicking on the **TStamp** column header and selecting the **usec** from in the list.

Data Tracing

Tracing of code execution is a great way to detect many programming errors, but without the visibility of data transfers it may be hard to see what the application is actually doing. In ‘normal’ C code, most ‘problematic’ places are prologue and epilogue of functions where several 32-bit CPU registers are transferred to/from the stack using **STMIA** and **LDMIA** instructions. Fortunately, OMAP devices come

with an 8-bit wide ETM port, which is usually enough to capture a complete data trace.

Our experiments with OMAP5905 show full visibility of all data memory reads and writes, however, there is a possibility of the ETM internal FIFO overflow if data accesses are very frequent or the memory WR/RD wait states are very short. Such overflows will cause a temporary loss of PC information, so please use caution when tracing variables on systems with fast memory cycles.

In the ETM Control dialog box you can choose to trace the variable(s) address or data or both. Since each variable address is 32-bit and most data contains a 32-bit value, about sixteen clock cycles will be used just to export each variable, so using byte variables (if possible) will save a lot of ETM bandwidth during tracing.

To trace what is written to the **Buf1** array, click on the **Control** button, select the **ETM Trace Settings** tab and check the first two boxes under the Data Tracing Mode.

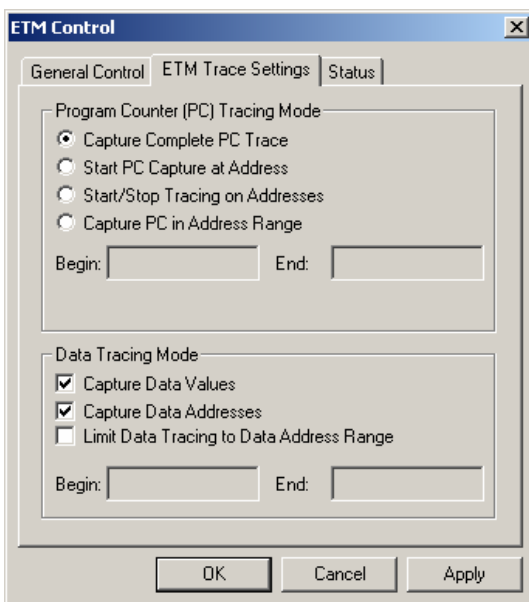


FIGURE 55

When done, click **OK** to close the Control window. If the CPU is still running, stop it by clicking on the **STOP** button. Open the Command window using the CMD icon on the tool bar and type in it:

```
>delaycnt = 10.
```

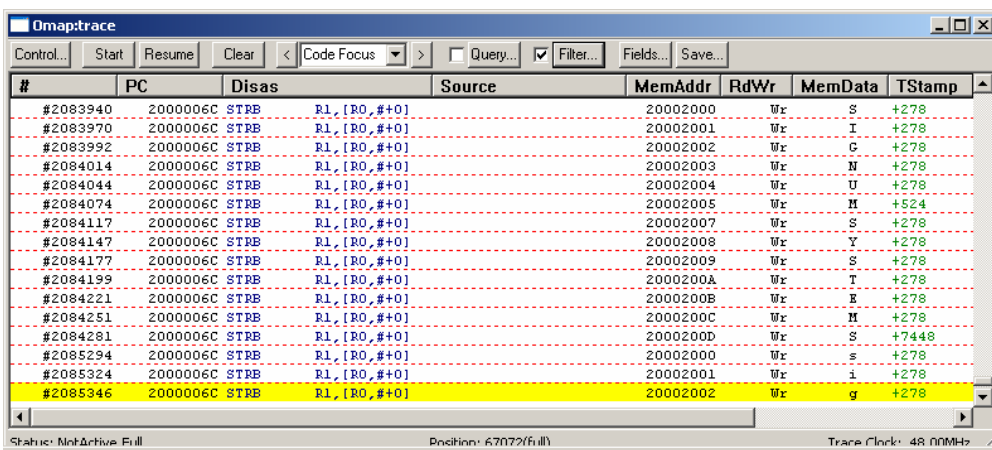
This reduces the amount of delay loops to 10 so that the LED is written more frequently so more of them will be shown in the trace.

To restart trace capturing when the CPU is running, just click on the **Start** button. If the CPU is stopped, clicking on the **GO** button will also start trace acquisition.

The trace will fill instantly and display new data with the MemAdr and MemData columns filled.

Our first filter will be to leave only the memory WR cycles in the trace, so double click on the **RdWr** column and select the **Filter only RdWr = Wr**.

Next, to see only the write operations produced by the STRB instruction at address 0x2000006C, double click on this address and select **And Filter with PC=0x2000006C**. This will AND the first filter setting with the second producing a trace that shows only what was written to the array.



The screenshot shows the Omaptrace application window. The main pane displays a list of trace events. The columns are: #, PC, Disas, Source, MemAddr, RdWr, MemData, and TStamp. The trace is filtered to show only write operations (Wr) to memory address 20002002. The events are listed in a table below.

#	PC	Disas	Source	MemAddr	RdWr	MemData	TStamp
#2083940	2000006C	STRE R1,[R0,#+0]		20002000	Wr	S	+278
#2083970	2000006C	STRE R1,[R0,#+0]		20002001	Wr	I	+278
#2083992	2000006C	STRE R1,[R0,#+0]		20002002	Wr	G	+278
#2084014	2000006C	STRE R1,[R0,#+0]		20002003	Wr	N	+278
#2084044	2000006C	STRE R1,[R0,#+0]		20002004	Wr	U	+278
#2084074	2000006C	STRE R1,[R0,#+0]		20002005	Wr	H	+524
#2084117	2000006C	STRE R1,[R0,#+0]		20002007	Wr	S	+278
#2084147	2000006C	STRE R1,[R0,#+0]		20002008	Wr	Y	+278
#2084177	2000006C	STRE R1,[R0,#+0]		20002009	Wr	S	+278
#2084199	2000006C	STRE R1,[R0,#+0]		2000200A	Wr	T	+278
#2084221	2000006C	STRE R1,[R0,#+0]		2000200B	Wr	F	+278
#2084251	2000006C	STRE R1,[R0,#+0]		2000200C	Wr	H	+278
#2084281	2000006C	STRE R1,[R0,#+0]		2000200D	Wr	S	+7448
#2085294	2000006C	STRE R1,[R0,#+0]		20002000	Wr	s	+278
#2085324	2000006C	STRE R1,[R0,#+0]		20002001	Wr	i	+278
#2085346	2000006C	STRE R1,[R0,#+0]		20002002	Wr	g	+278

The status bar at the bottom indicates: Status: NotActive Full, Position: 67072(Full), Trace Clock: 48.00MHz.

FIGURE 56

In order to save the ETM bandwidth and achieve the same results, we could limit the Data addresses to be traced to a range of addresses from 0x20002000 to 0x20002013 in the ETM Control dialog box.

Using Macros

To create a macro button on the tool bar, click the **MACROS** button and select the **OMAP5905** macro from the list of macros found in the Chameleon/ARM directory and click **OK**. A new macro button will appear on the toolbar.

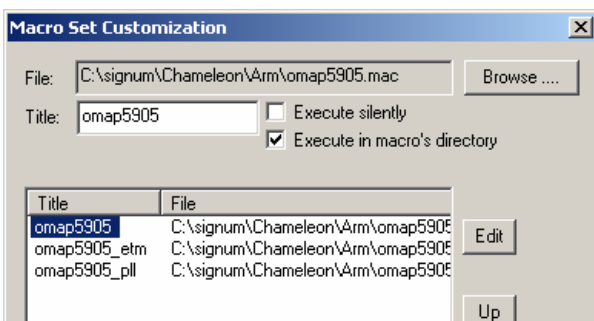


FIGURE 57

Do the same for the **OMAP5905_pll** and **OMAP5905_etm** macros and the toolbar will look like the one below:



FIGURE 58

To execute the **OMAP5905_pll** macro, just click on its button and the Command window will open asking to enter the new CPU frequency in MHz. Do not enter values larger than the maximum speed of the CPU in order not to damage the device (168 MHz in our example).

When the frequency is entered, the CPU speed will change and you should see a new **Trace Clock** speed being displayed at the bottom right of the trace window.

Please note that the Trace Clock speed is always $\frac{1}{2}$ of the CPU speed as the ETM captures data on both edges of the Trace clock in Half-Rate trace mode. Half-Rate trace mode is recommended, as it will allow the ARM devices to be clocked as high as 400 MHz and the Trace Clock will be only 200 MHz and still have the full bandwidth for trace capture.

The second macro **OMAP5905_etm** enables the device ETM port output, so it may be convenient to keep it on the tool bar to eliminate entering the command

```
>PINCR2 = PINCR2 | 0x3
```

after each device reset.

The **OMAP5905** macro is the macro needed to initialize the OMAP after power-up or device reset. It performs memory mapping for the debugger as well as disables the watchdog so it doesn't reset the CPU while debugging.

Using JTAGjet-Trace-OMAP with OMAP5912 “Larry Board”

*This chapter describes how to use JTAGjet-Trace emulator with the
OMAP5912 ECARUS board a.k.a. Larry Board. Rev 1.1.*

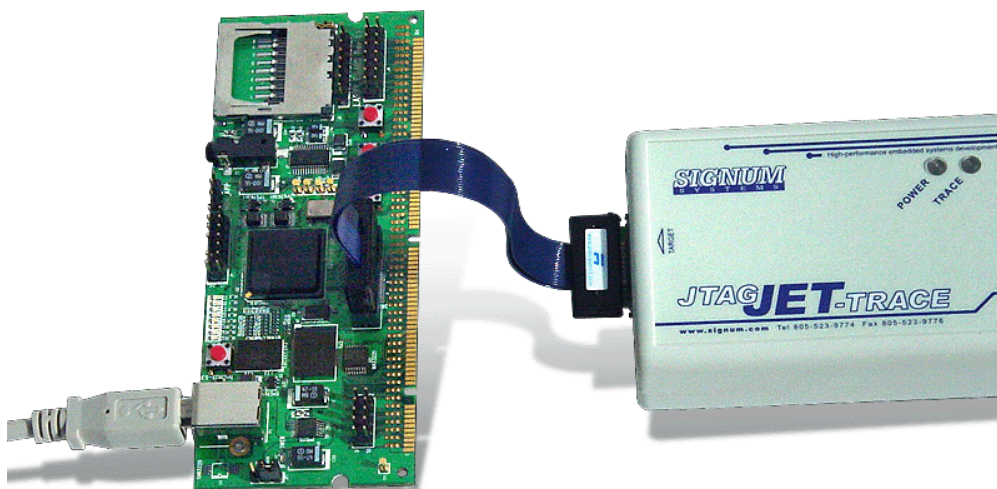


FIGURE 59 JTAGjet-Trace connected to the OMAP5912 “Larry Board” Rev 1.1

The ECARUS board should have the 38-pin ETM connector installed. Otherwise, it is up to the user to install it. The ETM connector is made by Tyco Electronics; the correct part number for a standard PCB (0.062 thickness) is 767054-1. These connectors are available from Signum Systems and from any Tyco distributor.

JTAGjet-Trace Hardware Setup

It is assumed that the JTAGjet-Trace software and USB drivers were already installed. If not, install them prior to connecting to the target board.

1. Verify that the following jumpers are installed:

JP1 [+5V] to [DC_MAIN] = SHORTED

This jumper enables the USB port to power the board and should be set as default by factory)

2. Make sure target board is not powered and connect the JTAGjet-Trace to the ETM connector on the board using the blue ETM cable.
3. Connect power-supply to the JTAGjet-Trace. The POWER LED indicator should come ON at this time.
4. Connect USB cable between the PC and JTAGjet-Trace. The PC should automatically recognize the JTAGjet on the USB port. Make sure your PC is equipped with the USB 2.0 port. The JTAGjet-Trace contains up to 18 Mbytes of trace memory which must be accessed quickly, so working with USB 1.1 ports is not recommended.
5. Always power-up the target board last. The board comes with a USB cable that provides power to the board. If the POWER LED does not come on, check the board's JP1 jumper – it should be installed between the **DC_MAIN** and **+5V** pins. The JTAG LED on emulator should turn ON at this time, indicating that target power has been detected and the hardware is ready to be used.

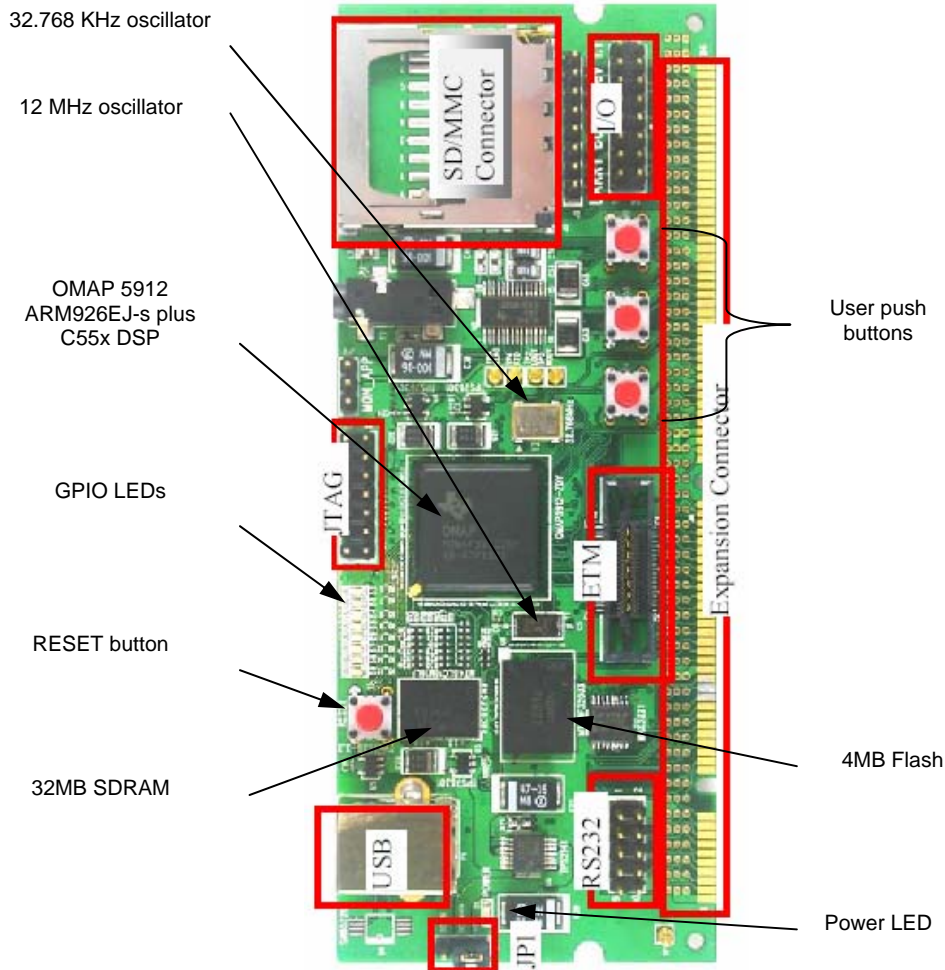


FIGURE 60 The front view of the Larry Board (Rev 1.1) showing connectors and major components.

Note: For best performance, the use of a USB 2.0 port is strongly suggested. Complete trace buffer may consist of up to 18 MBytes of trace data, and high-speed transfer is essential to avoid delays when reading the trace. USB 1.1 is not recommended as it will work 40 times slower than the USB 2.0.

Chameleon Debugger Setup

1. Execute **Chameleon** debugger from the Windows Start menu. Open the **View-System Configuration** menu and click on the **Add Target** button to create a new target in the debugger. In the Target Selection dialog box select the **ARM** family and enter a more specific name in the **Target Name** field (e.g., OMAP5912). Target names should describe boards, rather than devices—it is not uncommon to work over a period of time with various targets with the same CPU. Click **OK**. Next make sure that **Connect to an Emulator Automatically** is selected and click **Next**.
2. In the Startup Configuration Selection window, under Texas Instruments, select the OMAP5912 Larry Board and click **OK** (Figure 61).

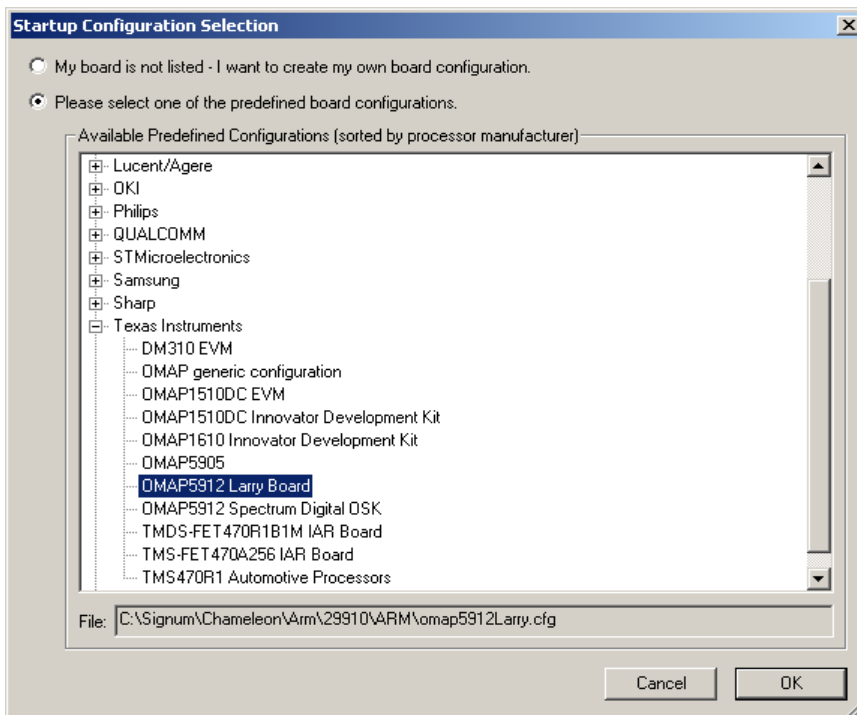


FIGURE 61 Selecting the TI OMAP5912 Larry board..

Setup Verification

1. The debugger will try to connect to the target board at this time. The **SYSLOG** window will appear, showing the progress of the operation, and displaying the version of the JTAGjet-Trace along with the size of the trace buffer. Please verify that it matches the size of trace you ordered. The trace sizes vary from **256K to 4M** and signify the trace depth. The trace width is fixed as **36-bit wide** so that a compressed time stamp can be added to each trace sample.

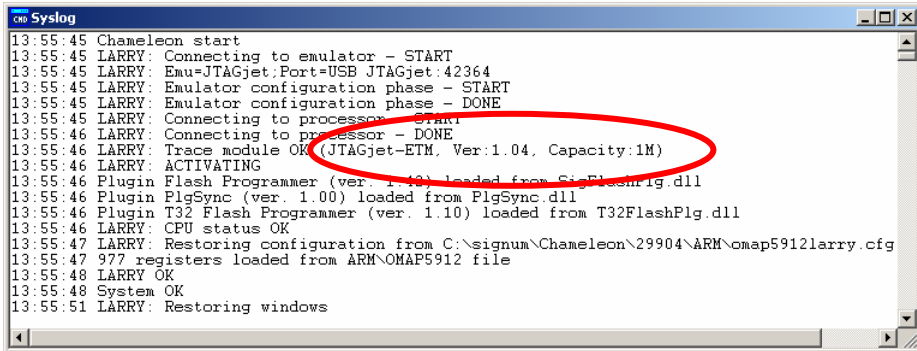


FIGURE 62 The Syslog window. “Ver: 1.02” denotes the revision of the emulator’s trace hardware, and “Capacity 2M” signifies a 2M deep (36-bits wide) buffer for a total of 9 MBytes of trace data).

- To finish the installation, bring up the CPU status window by clicking the CPU button on the toolbar.

It is a good idea to place this window in the upper right hand corner of the debugger’s screen or another unobtrusive location. The Status window contains all core registers, PC, Stack Pointer and the flags. It also shows if the CPU is running (pulsating green circle) or stopped (red STOP sign).

Creating Macro Buttons

Macros are script files written in the debugger’s command language to automate the debugging and testing processes. Macros may contain only a few commands that, for example, initialize the board to a known state and load the user’s code. More complex macros often comprise hundreds of commands that in turn may call other macros

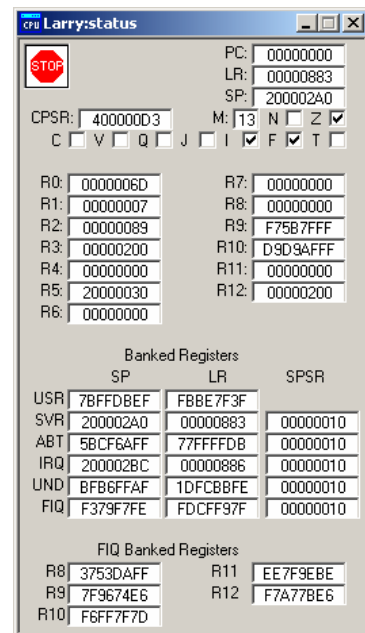


FIGURE 63 The Status window.

and perform such functions as NAND Flash formatting or detailed testing of the target board.

Macros can be created off-line via editor, or live with the use of the Command Window. Existing macro files may be executed in the Command window. Additionally, it is possible to create on the tool bar a button that with a click of a mouse runs a specific macro.

To create a macro button, click the **MACROS** button and select the **omap5912larry** macro from the list and click OK. A new macro button appears on the toolbar. Repeat the process, selecting the **omap5912_etm** and **omap5912_pll** macros.

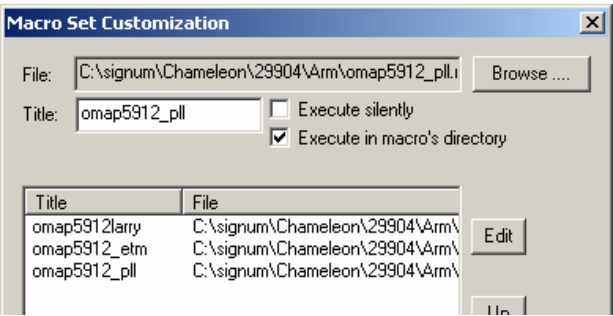


FIGURE 64 Attaching a macro to a Macros toolbar button.

The Macros toolbar can be docked to the main tool bar, left side of the screen, right side of the screen or allowed to float horizontally or vertically.



FIGURE 65 Completed Macros toolbar in horizontal form.

The macros were created to help the user in setting up the board and the ETM

The **omap5912larry.mac** macro is the same as the macro used by the debugger to configure the board on debugger startup. We will need to use this macro every time the CPU is Reset or powered-down to make sure the board is initialized for debugging.

The **omap5912_pll.mac** macro is a good example of how macro commands can be used interactively. When activated, it asks the user to type the new CPU speed in

MHz, after which it tries to adjust the PLL registers to get as close to the required operating frequency as possible.

The **omap5912_etm.mac** macro is needed to configure the OMAP5912 pins to output the ETM information to the ETM port. This should be normally done before the application starts, so that the startup code can be traced from the very beginning.

Working with ETM

1. Normally, after starting the Chameleon debugger, the ETM trace is disabled by default, since many devices share, i.e. multiplex, the ETM pins with other I/O functions. To enable the ETM all we need to do is to click on the **omap5912_etm** macro. To verify if the trace was enabled, open the **Trace** window from the **View | Trace** menu
2. Once the ETM is enabled, the Trace window will be cleared and the bottom line will show the status of the trace: **NotActive/Empty, Trace Empty (0%)** and **Trace Clock: 96.00MHz**. The displayed frequency should be the actual CPU clock frequency set by the **omap5912larry** macro on startup. If you see a 12MHz trace clock it means the device was reset or just powered-up and runs at the default clock speed.

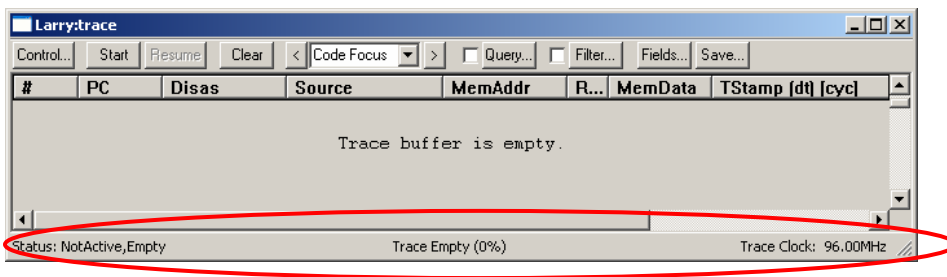


FIGURE 66 Trace Display Status bar showing activity, % usage and Trace Clock frequency

3. Another way to change the CPU speed is with the use of a macro. Since we have created a button for the **omap5912_pll** macro, click on it now. The Command Window will appear, prompting you to enter the new CPU

frequency in MHz. Once entered, the macro will calculate and set the needed CPU PLL registers to be as close to the entered frequency as possible in multiples of 12MHz.

4. Setting frequency higher than 200 MHz (the maximum CPU speed) should be avoided.

Keep in mind that whenever the board is reset or powered-down, the ETM Trace will need to be re-enabled by running the **omap5912larry** and **omap5912_etm** macros from the toolbar.

5. Once the ETM is enabled, click on the Select the **Control** button to bring-up the ETM setup window. Make sure the Enable ETM checkmark is on and that the trace is stopped **When Trace Buffer is Full**.

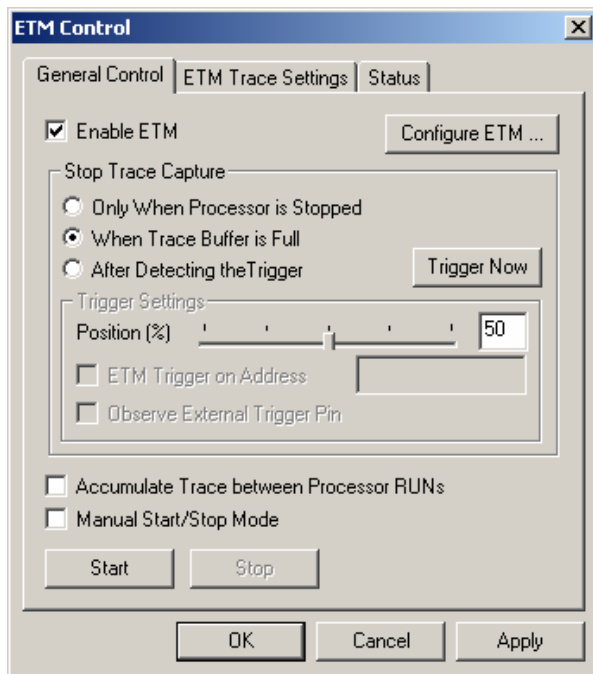


FIGURE 67 ETM General Control tab

- Next, click on the **Status** tab to display information related to the ETM resources embedded into the target ARM device. Please read the contents of the information dialog box and note the description of the ARM device's ETM resources. These resources can vary substantially from one ARM device family to another. Oftentimes, locating such information in the device's data sheet is cumbersome.

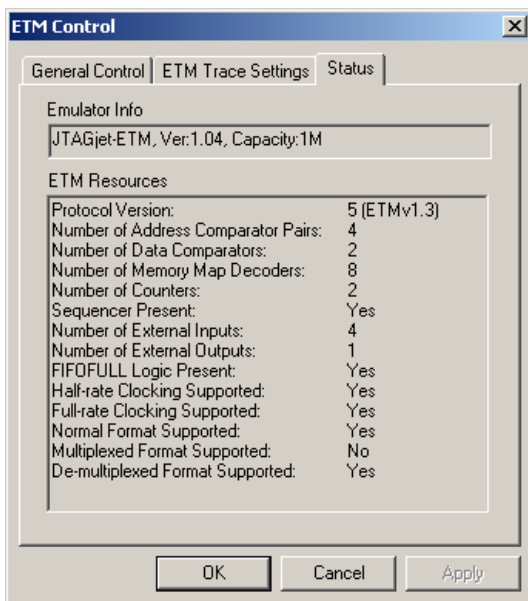


FIGURE 68 ETM Status tab showing the ETM resources

- Select the **ETM Trace Settings** tab to make sure that the **Capture Complete PC Trace** is enabled and that the Data Tracing check boxes are cleared. Click **OK** to complete the trace configuration setup.

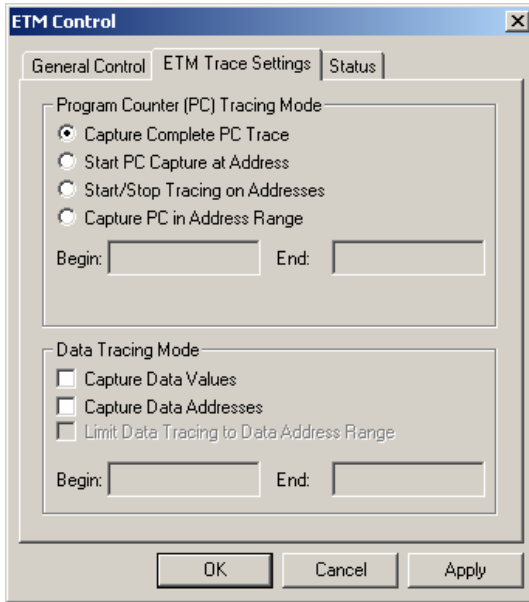


FIGURE 69 ETM Trace Settings tab.

8. The Chameleon installation includes a small demo program for the Larry Board.. The program is written in C and performs simple functions and LED blinking. To load the demo program select **File | Load** from the main menu and select the demo.elf file located in the folder

C:\...\Chameleon\Arm\Demos\BoardSupport\TT\Larry\Demo.

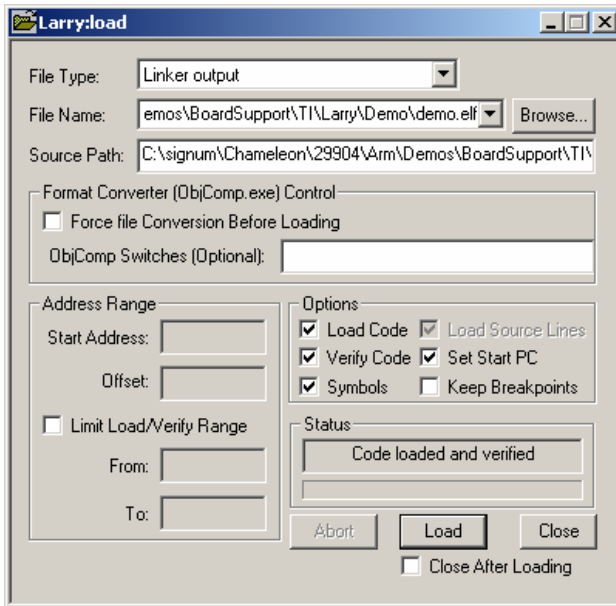


FIGURE 70 Load File dialog box

When the file is selected, click on the **Load** button. Verify that “**Code loaded and verified**” is displayed in the Status field. After the code is loaded, click **Close** to close the Load window. If the program did not load correctly, make sure you selected the demo.elf file from the correct directory.

9. Next, open the Source Window (View | Source) and verify that it displays the beginning of the startup code (**b startup**) written in assembly. To make the code comments visible, set the display mode to Source. You can change the Source Window display mode anytime by clicking the right mouse button in the Source Window and choosing the **Display Mode** menu.
10. To capture the execution of the startup code only, position the **Source** window, right above the Trace window. Then scroll down to **line 80 (BL C_Entry)** and insert a breakpoint by clicking on the leftmost column in line 80. A red dot and a letter **S** will appear, signifying it is a software breakpoint. Now just click the green **GO** button on the tool bar to start program execution. When the

break stops execution, the trace will display the end of the trace buffer, showing the last several executed instructions.

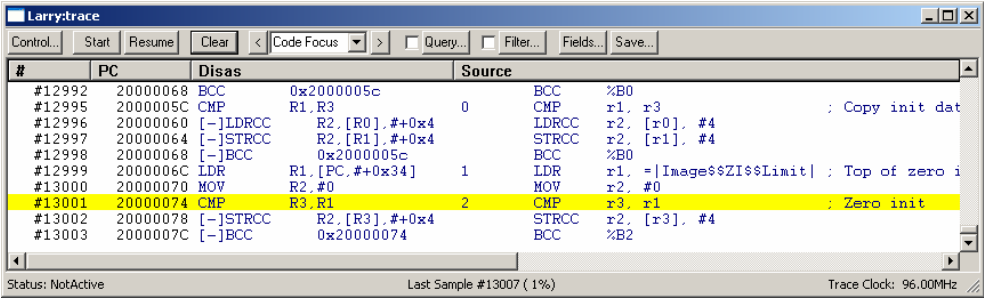


FIGURE 71 Trace Window showing the end of startup code

11. **Synchronization with the Source Window.** To see the trace buffer in the context of the source code click on the last sample in the trace (the yellow bar indicates the current sample) and automatically a similar yellow bar is placed in the Source window pointing to the line of code responsible for that trace frame. Using the arrow keys on your keyboard, you can scroll the yellow bar up the trace buffer row by row and see the corresponding source code synchronized with the trace all the way to the beginning of the code.
12. To fill the entire trace buffer with trace history, click the trace **Control** button and make sure the **Stop Trace Capture When Trace Buffer is Full** option is selected. Continue running the program by selecting **GO** again. The trace buffer will accumulate a large amount of samples before it gets full. At that point, the trace window will stop acquiring more data and begin displaying the captured information. Press the red **STOP** button on the toolbar to stop the CPU. This will allow the debugger to update the open memory and register windows.
13. To display symbolic addresses in the trace PC column, click on the PC column header and add checkmark next to the **Symbols**. All symbolic references are colored green for easy spotting.
14. **Trace Filtering.** The ability to hide unwanted information from the trace buffer to get a clearer picture cannot be overestimated. Trace filtering is easily accomplished by double-clicking in any column in the Trace window with the exception of the Time Stamp column. For example, to see only trace frames

that contain C source lines, double-click inside any Source line column. In the menu that appears, select “Filter non-empty Source”

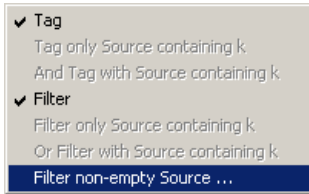


FIGURE 72 Filtering lines with non-empty source lines

The Trace window will filter out all frames without C source associated with them. With only lines of C code displayed, program execution will be easier to analyze.

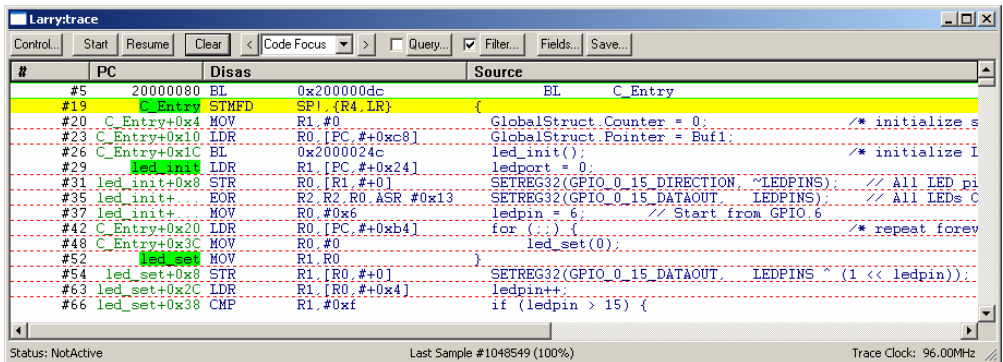


FIGURE 73 Filtering to show only the rows with C code

15. **Using Query.** To return to the unfiltered view, uncheck the Filter check box on the toolbar. Click on the Query drop down list and change the query to look for the next **Module**. Keep pressing the forward [>] button until **processing (Buf1)** is found. Double click on the **PC** column of that trace line and from the menu select “Filter only PC = C_Entry+0x6C”. This will change the filter to

show only calls to **processing**. This will allow us to measure the frequency of the calls and determine if the interval between calls is constant.

16. To find out how often the **processing** function is called, scroll the trace window to the right to see the **TStamp** column and double-clicking on it until it shows the time in delta mode and cycles **[dt][cyc]**. At the end of the process, we find that the call takes place every 4950 CPU cycles. To convert this value to microseconds, click on the TStamp header and select the **usec** from the list.

Now double click anywhere in the time stamp column to change the display to **[abs][us]**. Scroll to the end of the buffer to see how much execution time is covered by the contents of the trace buffer. A 1M buffer, is capable of storing over 91 msec worth of a program running at 96 MHz. With the 4M trace buffer the total captured program will span about 365 msec.

#	PC	Disas	Source	M..	R..	M..	TStamp [dt] [cyc]	D..
#244	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#615	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#988	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#1362	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#1733	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#2106	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#2480	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#2851	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#3224	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#3597	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#3968	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#4341	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#4712	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#5086	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	
#5459	20000148	LDR	R0, [PC, #+0x6c]				processing(Buf1)... +4950	

Status: NotActive Last Sample #1048549 (100%) Trace Clock: 96.00MHz

FIGURE 74 Filtered trace showing calls to only one function with time stamp in Delta Cycles mode

17. As you probably noticed, our experiment filled the trace buffer in an instant. This is because the buffer stored the samples indiscriminately without eliminating such uninteresting fragments of the application as loops waiting for ready bits or software delays. This can easily be amended by pre-filtering the data that comes into the trace buffer so that only the relevant information is retained. One way to do it is to set the trace to capture execution in PC address range that interests us.

Please uncheck the **Filter** to see all samples, and then click on the **CLEAR** button to erase the trace buffer. Now open the Control dialog box and select the **ETM Trace Settings** tab.

In the Settings tab, select the **Capture PC in Address Range** setting and enter the following addresses:

18. Begin: {"demo.c"}@30 End: {"demo.c"}@58

These settings will capture execution trace of the main.c program loop only, and any function calls outside of main will not be recorded. This is a great way to catch the profile of the application as you can see clearly the timing of the major system functions.

Click **OK** to save the settings. Click also the yellow **RST** (restart) button on the top toolbar. The RST button brings the application back to the initial point without resetting the CPU or reloading the application.

Click the **GO** button. Notice that the trace buffer fills much slower this time, as it stores only the lines executed from the main loop. When the trace fills up and stops, stop the CPU and click the Home button to see the beginning of the trace.

#	PC	Disas	Source	M.	R.	M.	TStamp [abs] [us]	Dat
#7	C_Entry	STMFD SP!, {R4, LR}	{				0.406	
#8	C_Entry+0x4	MOV R1, #0	GlobalStruct.Counter...				0.510	
#9	C_Entry+0x8	LDR R0, [PC, #+0xc0]					0.677	
#10	C_Entry+0xc	STR R1, [R0, #+0]					0.771	
#11	C_Entry+0x10	LDR R0, [PC, #+0xc8]	GlobalStruct.Pointer...				0.813	
#12	C_Entry+0x14	LDR R1, [PC, #+0xc0]					0.979	
#13	C_Entry+0x18	STR R0, [R1, #+0x4]					0.990	
#19	led_init+0x28	MOV PC, LR					3.188	
#22	C_Entry+0x20	LDR R0, [PC, #+0xb4]	for (;;) {				3.365	
#23	C_Entry+0x24	LDR R0, [R0, #+0]					3.448	
#24	C_Entry+0x28	TST R0, #0x1					3.615	
#25	C_Entry+0x2c	BEQ 0x20000118					3.625	
#28	C_Entry+0x3c	MOV R0, #0	led_set(0);				3.781	
#34	led_set+0x4c	MOVNE PC, LR					5.386	
#37	C_Entry+0x44	MOV R4, #0	}				5.563	
#38	C_Entry+0x48	LDR R0, [PC, #+0x94]					5.688	

FIGURE 75 Filtering to show capture in PC range

The green lines in the Trace window represent trace discontinuities. As expected, the trace started tracing from the first line of the main program (**C_Entry**) and stopped on **C_Entry+18**, which is where the first function (**led_init**) was called. We find that this function stopped the trace for 3.188 μ s and started to capture again when it returned to the main program to execute the **for (; ;)** loop. The trace captured a few lines from main and stopped again when **led_set(0)** was called, which took 5.386 μ s. If we scroll the trace window down, we can also see that the **delay** function takes 17.615 μ s. The end of the trace buffer will show that the application now run for about 735 msec before the trace buffer filled up.

19. To eliminate any function from the trace, we must use the **Start/Stop Tracing on Address** feature. It is necessary to enter the function's exit address (in the trace **Begin** field and the function's entry point in the trace **End** field.

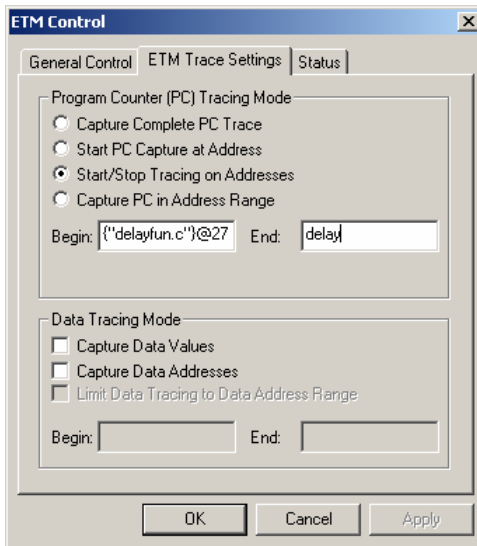
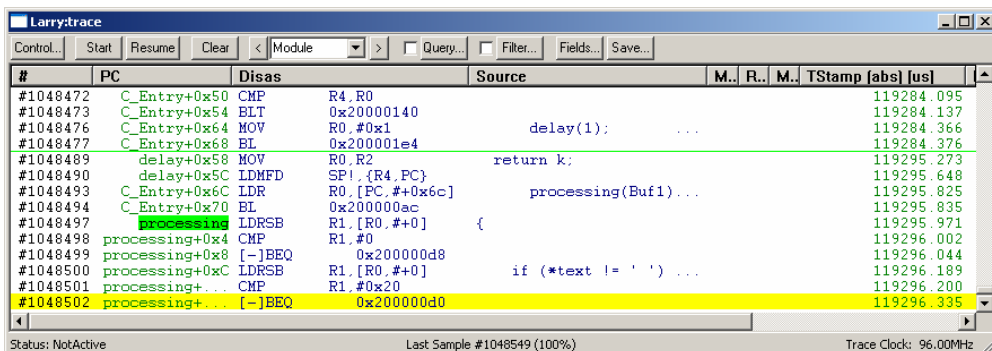


FIGURE 76 Trace with the delay function filtered out.

The figure below shows an example with the delay function filtered out, allowing the trace to capture over 119 msec of the application run time.



#	PC	Disas	Source	M.	R.	M.	TStamp [abs] [us]
#1048472	C_Entry+0x50	CMP	R4, R0				119284.095
#1048473	C_Entry+0x54	BLT	0x20000140				119284.137
#1048476	C_Entry+0x64	MOV	R0, #0x1				119284.366
#1048477	C_Entry+0x68	BL	0x200001e4				119284.376
#1048489	delay+0x58	MOV	R0, R2				119295.273
#1048490	delay+0x5C	LDHFD	SP!, {R4, PC}				119295.648
#1048493	C_Entry+0x6C	LDR	R0, [PC, #+0x6c]				119295.825
#1048494	C_Entry+0x70	BL	0x200000ac				119295.835
#1048497	processing	LDRSB	R1, [R0, #+0]				119295.971
#1048498	processing+0x4	CMP	R1, #0				119296.002
#1048499	processing+0x8	[−]BEQ	0x200000d8				119296.044
#1048500	processing+0xC	LDRSB	R1, [R0, #+0]				119296.189
#1048501	processing+...	CMP	R1, #0x20				119296.200
#1048502	processing+...	[−]BEQ	0x200000d0				119296.335

Status: NotActive Last Sample #1048549 (100%) Trace Clock: 96.00MHz

FIGURE 77 Trace with the delay function filtered out.

Data Tracing with OMAP5912

Tracing of code execution is an excellent method for detecting many programming errors. However, without an ability to visualize data transfers, it may be hard to see what the application is actually doing. In typical C code, most problematic places are prologue and epilogue of functions where several 32-bit CPU registers are transferred to or from the stack with the STMIA and LDMIA instructions.

The OMAP5912 offers full data tracing capability as it has an 8-bit wide ETM port which gives enough bandwidth for both PC and variable tracing. Only in certain frequent variable accesses the on-chip ETM FIFO may overflow, causing a temporary loss of PC information. When this occurs, the word FIFO will show in the SyncCode column. To eliminate frequent accesses to memory variables you may experiment with different levels of optimization. Please remember that setting the level of optimization too high may make the code hard to follow, so some experimenting will be necessary.

In the ETM Control dialog box, you can choose to trace the variable(s) address or data or both.. To trace all data variables, click on the **Control** button, select the **ETM Trace Settings** tab and make sure the **Capture Complete PC Range** is selected and Capture Data Values and Capture Data Addresses are selected.

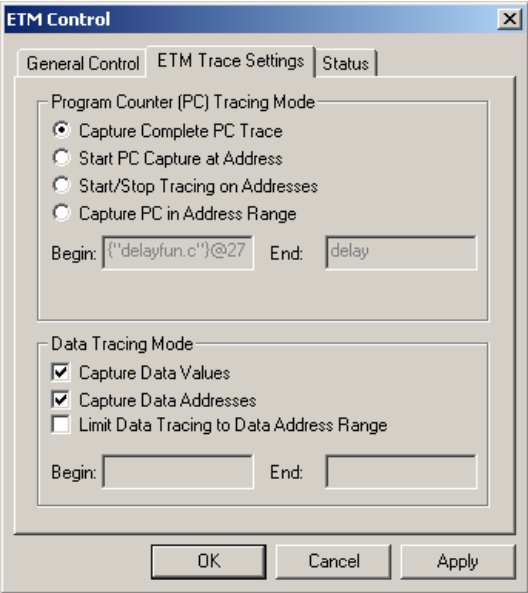


FIGURE 78 Tracing of variables

Click **OK** to close the Control window. If the CPU is still running, stop it by pressing the **STOP** button. To restart the application click on the yellow **RST** and **Go** buttons and the trace will now show both the PC and variables trace.

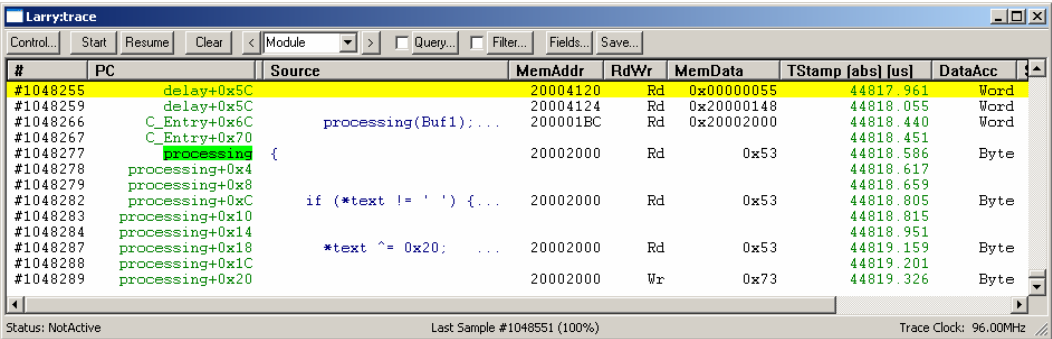


FIGURE 79 Tracing of variables

TRACING VARIABLE RANGE

To trace what is read and written to the `Buf1[]` array only, click on the **Control** button, select the **ETM Trace Settings** tab and make sure the **Capture Complete PC Range** is selected. In the **Data Tracing** section right below, check all three boxes and enter the addresses **&Buf1[0]** and **&Buf1[14]** in the Begin and End fields, respectively.

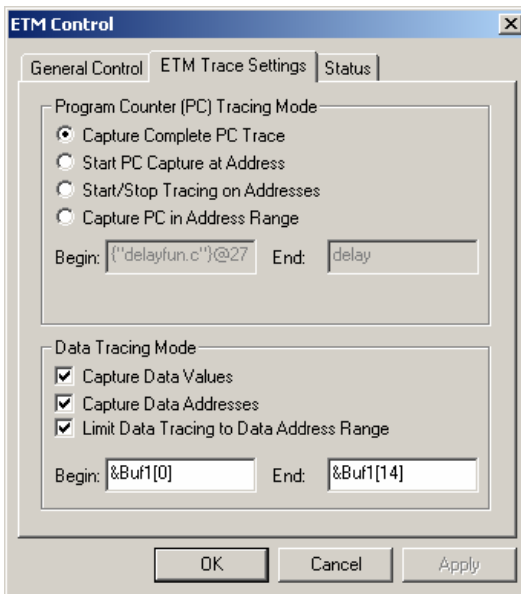


FIGURE 80 Adding tracing of variables to the ETM settings

Click **OK** to close the Control window. If the CPU is still running, stop it by pressing the **STOP** button.

To restart trace capture when the CPU is running, press the **Start** button in the trace window. If the CPU is stopped, press the **GO** button to start the target application.

The trace will fill instantly and display new data. To find the data variables in the trace, double click in the **MemAddr** or **MemData** fields and select **Filter non-empty MemAddr**. This will only display the trace frames with valid MemAddr

fields. In our exercise, only the Buf1[] writes and reads and the associated data values are shown.

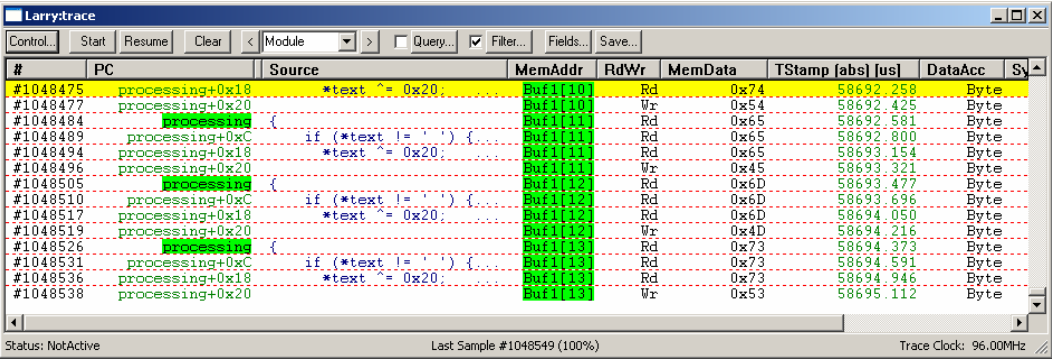


FIGURE 81 Filtered trace showing only read and write to the Buf1[] array operations.

To have the MemData column display in ASCII mode, click in the column header and check ASCII from the menu. When closed, the trace will show ASCII characters in place of HEX values, making reading the contents of the buffers containing text strings easier for a human reader.

In our example, switching to ASCII format reveals that the **processing()** function flips the case of the Buf1[] string (lower to upper or vice versa).

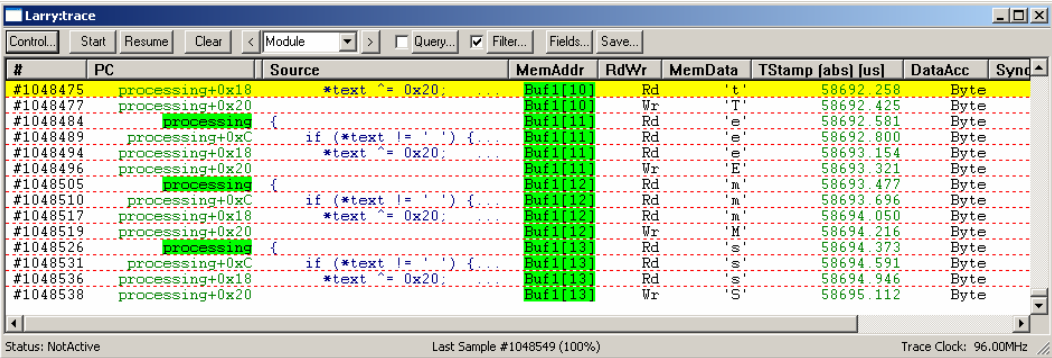


FIGURE 82 Filtered trace showing variable data in the ASCII format.

Changing Variables During Execution

Even though unrelated directly to the ETM trace, you may find this section pertinent and helpful.

If our application runs in real-time, the blinking of the LEDs will not be perceptible, because value of the delaycnt variable was set to 5 and the resulting delay is very short. We will learn how to change this value while the application is running.

Close the Trace window and press the **GO** button. The application starts running. Open the **Watch** window by clicking on the **WCH** icon on the toolbar. An empty Watch window appears. Right-click inside the window, then select **Add** from the pop-up menu. In the dialog box that appears, type the name **delaycnt**.

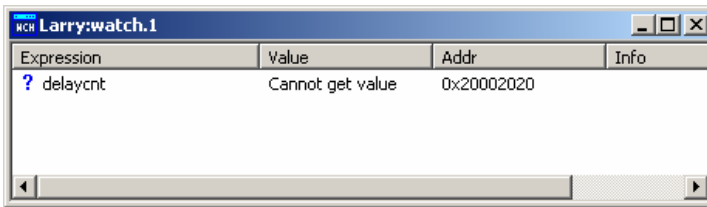


FIGURE 83 Filtered trace showing variable data in the ASCII format.

The variable delaycnt appears in the Watch window, but the value will not be read since the application is running. To force the reading of its value, select the variable in the Watch window and right-click on it and check the **Access in RUN mode** feature.

Now select the variable again with the mouse and press the Space bar on your keyboard. This will force the variable to be read showing the value of 5

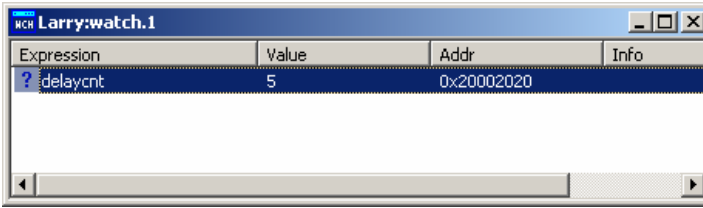


FIGURE 84 Selecting the delaycnt variable in the Watch window.

To change the variable, double click on its value in the Watch window and enter 500 in the entry field. The blinking of the LEDs will now be visible to the eye.

Variables can also be drag-and-dropped to the Watch window. To try this capability, we need to stop the CPU. Right-click in the Source Window and select **View Module** in the pop-up menu. When a list of the modules appears, select **demo.c** and click OK.

The Source window will show the beginning of the demo.c file.

To be compatible with older Signum emulators, a double-click on any line in the Source window triggers the option **Execute to this line**. This default setting may be changed by opening the Source window options menu with a right-click, and then selecting **Options** from the list and un-checking **Double click == GO TILL**. With this change, a double-click on a source line will select variables. Click on any GlobalStruct variable and drag it into the Watch window.

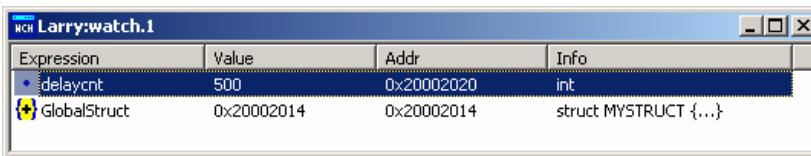


FIGURE 85 Selecting variables in the Watch window.

To expand the structure, click the yellow **{+}** next to the variable's name and make sure it is accessible in RUN mode, just as we did with the delaycnt variable.

Press **Go** to run the application again. You can now inspect the members of the structure while the application is running by selecting them in the Watch window and pressing the space bar.

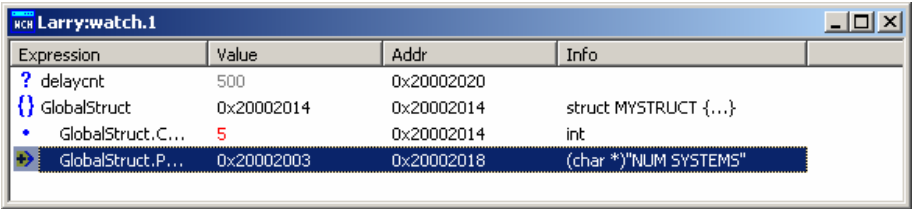


FIGURE 86 Selecting a member of a complex variable for inspection.

Using JTAGjet-Trace with Raisonance REva STR912F Board

This chapter describes how to use JTAGjet-Trace with the SRaisonance REva STR912F board. The board contains the STR912F ARM device which is based on the ARM9 core.

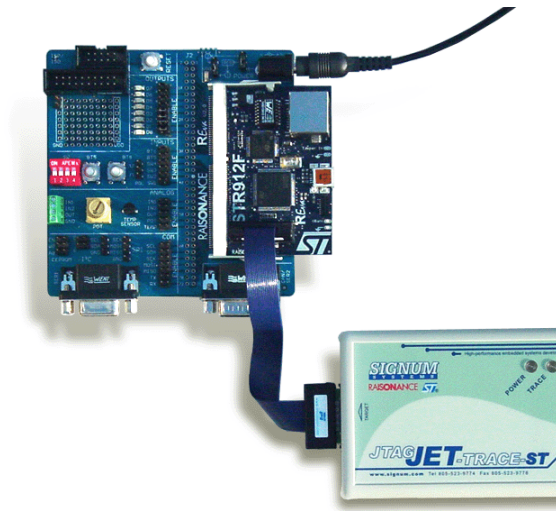


FIGURE 87 JTAGjet-Trace connected to the STR912F target board.

Usually, all Signum ARM boards have the ETM connector installed. If not, you must do it yourself. The ETM connector is made by Tyco Electronics; the correct part number for a standard PCB (0.062 thickness) is 767054-1. These

connectors are available from Signum Systems and from any Tyco distributor.

JTAGjet-Trace Hardware Setup

It is assumed that the JTAGjet-Trace software and USB drivers were already installed. If not, install them prior to connecting to the target board.

1. Make sure target board is not powered and connect the JTAGjet-Trace to the ETM connector on the board using the blue ETM cable.
2. Connect power-supply to the JTAGjet-Trace. The POWER LED indicator should come ON at this time.
3. Connect the USB cable between the PC and JTAGjet-Trace. The PC should automatically recognize the JTAGjet on the USB port. Make sure your PC is equipped with the USB 2.0 port.

Always power-up the target board last. The board comes with a USB cable that provides enough power to the board. Alternately, there is a 5V DC power jack on the board. If the Power LED does not come on, check the board's SW1 power switch, which is located near the POWER connector. The JTAG LED on emulator should turn ON at this time, indicating that target power has been detected and the hardware is ready to be used.

Note: For best performance, the use of a USB 2.0 port is strongly suggested. Complete trace buffer may consist of up to 18 MBytes of trace data, while high-speed transfer is essential to avoid delays when reading the trace. USB 1.1 is not recommended as it will work 40 times slower than the USB 2.0.

Chameleon Debugger Setup

1. Execute **Chameleon** debugger from the Windows Start menu. Open the **View-System Configuration** menu and click on the **Add Target** button to create a new target in the debugger. In the Target Selection dialog box select the **ARM** family and enter a more

specific name in the **Target Name** field (e.g. STR9-REva). Target names should describe boards, rather than devices—it is not uncommon to work over a period of time with various targets with the same CPU. Click **OK**. Next make sure that **Connect to an Emulator Automatically** is selected and click **Next**.

Next, in the Startup Configuration Selection window under STMicroelectronics select the STR912F Raisonance Reva board and click OK (Figure 88).

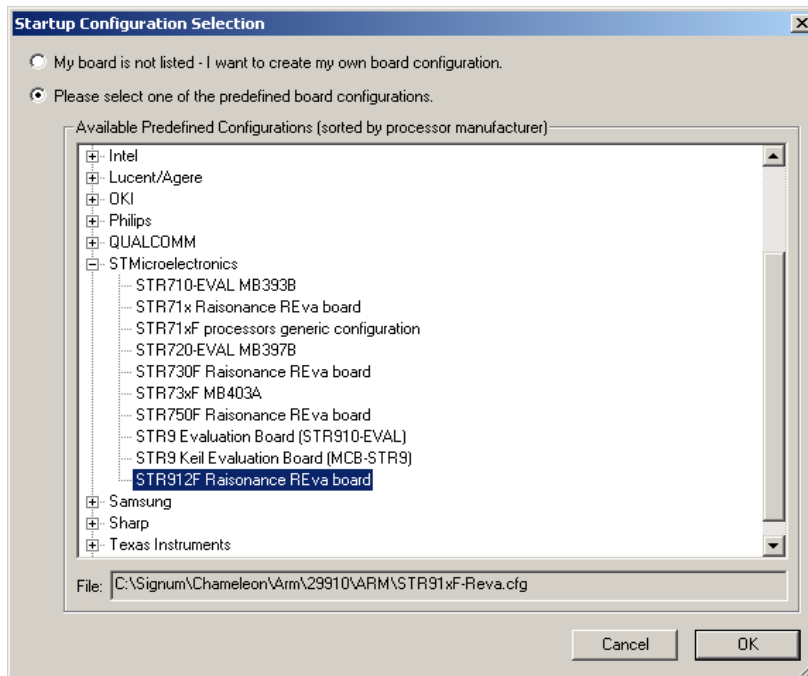


FIGURE 88 Selecting the STR912F Raisonance Reva board.

Setup Verification

1. The debugger will try to connect to the target board at this time. The **SYSLOG** window appears, showing the progress of the operation, and displaying the version of the JTAGjet-Trace along with the size of the trace buffer (FIGURE 89).

Click **OK** to close the System Configuration setup window. The Syslog window displays now the capacity of the ETM trace buffer. Verify that it matches the size of trace you ordered. The trace sizes vary from **256K to 4M** and signify the trace depth. The trace width is fixed as **36-bit wide** so that a compressed time stamp can be added to each trace sample.

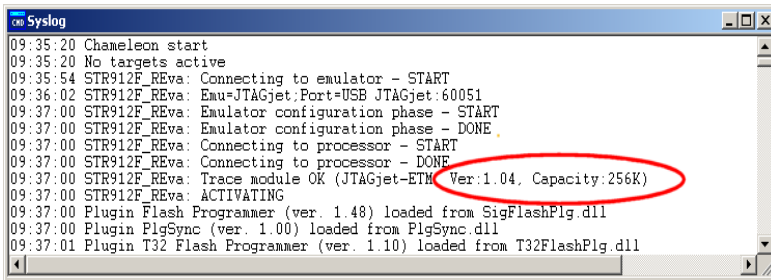


FIGURE 89 The Syslog window. “Ver:1.04” denotes the revision of the emulator’s trace hardware, and “Capacity 256K” signifies a 256K deep (36-bits wide) buffer for a total of 1.125 MBytes of trace data).

- To finish the installation, bring up the CPU status window by clicking the CPU button on the toolbar.

It is a good idea to place this window in the upper right hand corner of the debugger screen or another unobtrusive location. The Status window contains all core registers, PC, Stack Pointer and the flags. It also shows if the CPU is running (pulsating green circle) or stopped (red STOP sign).

Creating Macro Buttons

Macros are script files written in the debugger’s command language to automate the debugging and testing processes. Macros may contain only a few commands that, for example, initialize the board to a

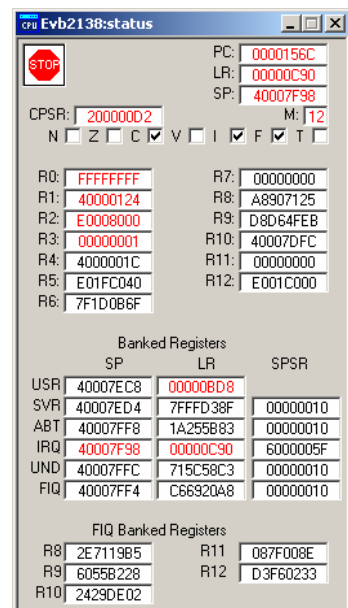


FIGURE 90 The Status window.

known state and load the user's code. More complex macros often comprise hundreds of commands that in turn may call other macros and perform such functions as NAND Flash formatting or detailed testing of the target board.

Macros can be created off-line via editor, or live with the use of the Command Window. Existing macro files may be executed in the Command window. Additionally, it is possible to create a button (on the tool bar) that with a click of a mouse runs a specific macro.

To create a macro button, click the MACROS button and select the desired macro from the list and Click OK. A new macro button appears on the toolbar.

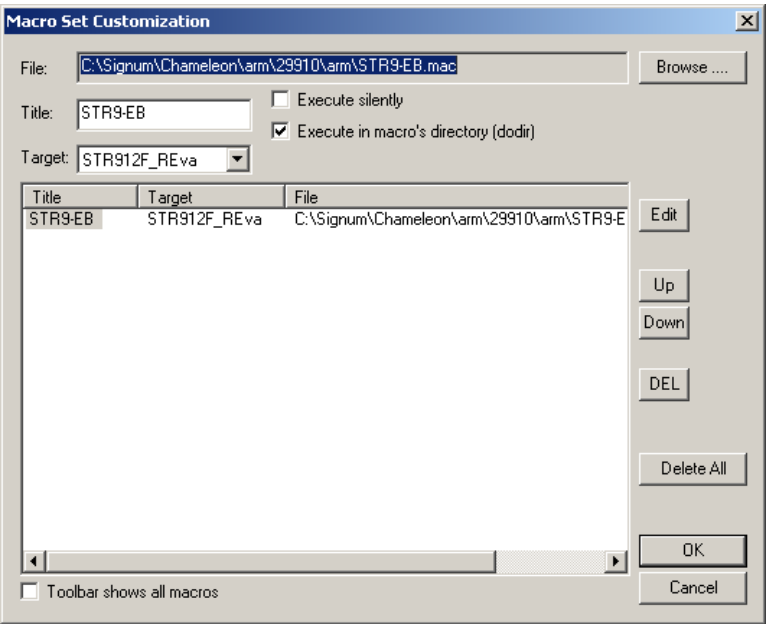


FIGURE 91 Attaching a macro to a Macros toolbar button.

The Macros toolbar should look like this (Figure 92).



FIGURE 92 A completed Macros toolbar.

The STR9-EB.mac is the macro used by the debugger to configure the board and enable the ETM trace. We will need to use this macro every time the CPU is Reset or powered-down.

The Set_PLL66 and Set_PLL96 options in the STR9-EB macro are good example of how macro commands can be used. When activated, each option adjusts the STR912F PLL registers to get as close to the respective CPU clock frequency as possible.

Working with ETM

1. Normally, after starting the Chameleon debugger, the ETM trace is disabled by default, since many devices share, i.e. multiplex, the ETM pins with other I/O functions. However, the startup macro we selected contains the **trace enable** command that enables the ETM trace on startup. To verify this, open the **Trace** window from the View | Trace menu. The status is shown in the bottom left corner. Open ETM Control by clicking the **Control** button and click the **Enable** check box if the status is “Disable.”

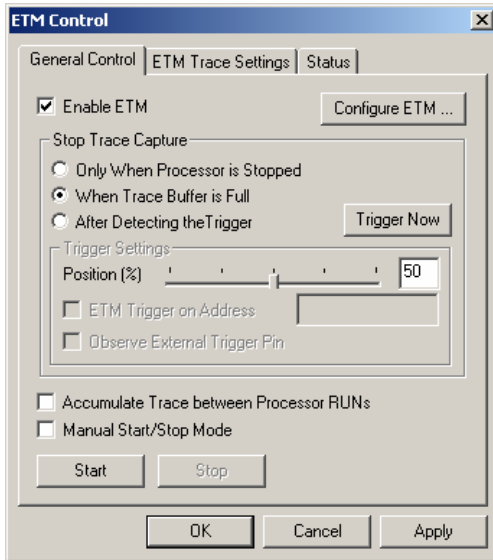


FIGURE 93 The ETM General Control tab

2. Select the **Status** tab to display information related to the ETM resources embedded into the target ARM device. Please read the contents of the information dialog box and note the description of the ARM device's ETM resources (Figure 94). These resources can vary substantially from one ARM device family to another. Oftentimes, locating such information in the device's data sheet is cumbersome.

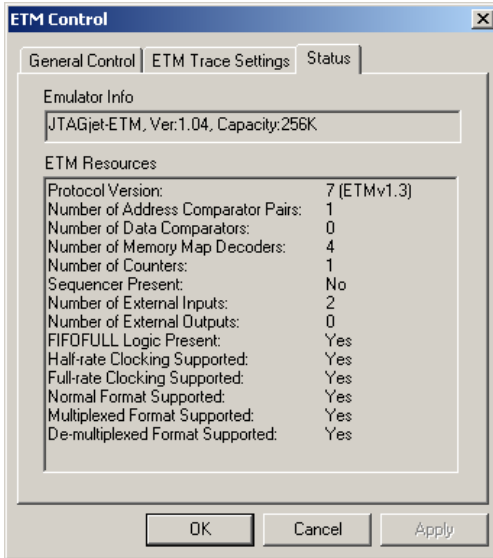


FIGURE 94 ETM Status tab showing the STR912F ETM resources

3. Select the **ETM Trace Settings** tab to make sure that the **Capture Complete PC Trace** is enabled and that the Data Tracing check boxes are cleared (Figure 95). Click **OK** to complete the trace configuration setup.

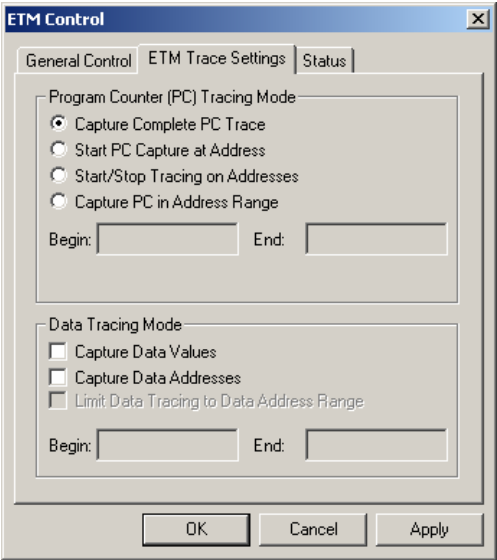


FIGURE 95 ETM Trace Settings tab.

Once the ETM is enabled, the Trace window will be cleared. The bottom status line shows now the current status of the trace; the Trace Clock frequency is approximately 96.00 MHz (FIGURE 96). The displayed frequency should be the actual CPU clock frequency set by the startup macro. If you see a 25.00 MHz it means the device is running at the default speed. This is possible if the startup macro is not run after turning on or resetting the board.

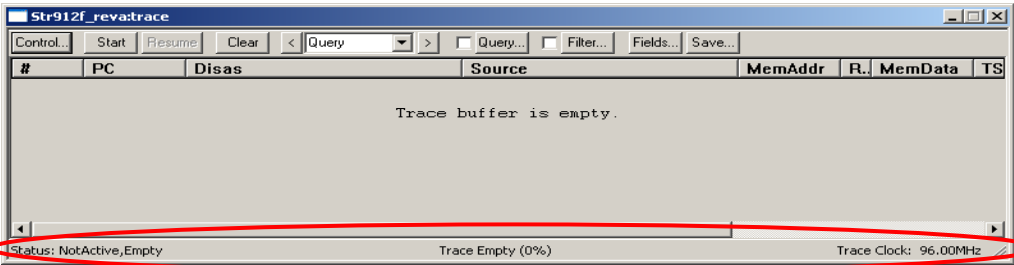


FIGURE 96 The Trace Display Status bar showing activity, % usage and Trace Clock frequency

4. Another way to change the CPU speed is with the use of a macro. We have a button for the STR9EB macro, click on it now Select SetPLL96 from the menu. This option in the macro sets the CPU clock to 96MHz

If the Trace Clock frequency in the Status bar displays 0, there is a problem with the target board, or the ETM connector is not plugged in all the way, or the jumpers are not installed properly. Please make sure the frequency is correct before continuing with the tutorial.

Keep in mind that whenever the board is reset or powered-down, the ETM Trace will need to be re-enabled manually or by running the STR9EB.mac macro from the toolbar.

5. The Chameleon installation includes a small demo program for the Rainsonance STR912F devices. The program is written in C and performs simple functions and LED blinking. To load the demo program (FIGURE 97), select File | Load from the main menu and select the demo.elf file located in the folder:

C:\...\Chameleon\Arm\Demos\BoardSupport\ST\STR91x\Demo.

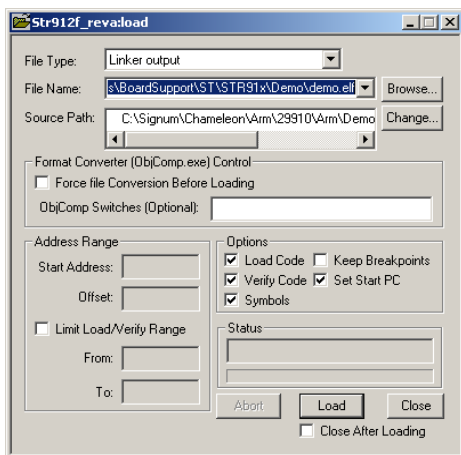


FIGURE 97 Load File dialog box

6. Select a file and click on the **Load** button. Verify the “**Code loaded and verified**” is displayed in the Status field. After the code is loaded, click **Close** to close the Load window. If the program did not load correctly, make sure you selected the demo.elf file from the correct directory. Open the Source window by clicking the **SRC** button. The Source Window now should be displaying the beginning of the startup code (PC = 0x40000000) written in assembly. To make the code comments visible, set the display mode to Source. You can change the Source Window display mode anytime by clicking the right mouse button in the Source Window and choosing the **Display Mode** menu.
7. To capture the execution of the startup code only, open the **Source** window, and position it right above the Trace window. Then scroll down to line 113 (BL main) and insert a breakpoint by clicking on the leftmost column in line 113. A red dot and a letter S will appear, signifying it is a SOFTWARE breakpoint. Now just click the green **GO** button on the tool bar to start program execution. When the break stops execution, the trace will display the end of the trace buffer, showing the last several executed instructions.

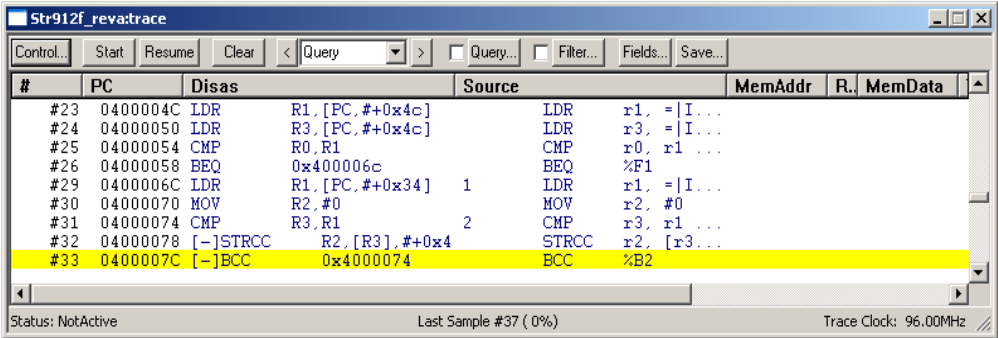


FIGURE 98 Trace Window showing the end of assembly startup code

8. **Synchronization with the Source Window** To see the trace buffer in the context of the source code click on the last sample in the trace (the yellow bar indicates the current sample) and automatically a similar yellow bar is placed in the Source window pointing to the line of code responsible for that trace frame. Using the arrow keys on your keyboard, you can scroll the yellow bar up the trace buffer

row by row and see the corresponding source code synchronized with the trace all the way to the beginning of the code.

To display symbolic addresses in the trace PC column, click on the PC column header and add checkmark next to the **Symbols**. All symbolic references are colored green for easy spotting.

9. To fill the entire trace buffer with trace history, click the trace **Control** button and make sure the **Stop Trace Capture When Trace Buffer is Full** option is selected. Continue running the program by selecting **GO** again. The trace buffer will accumulate a large amount of samples before it gets full. At that point, the trace window will stop acquiring more data and begin displaying the captured information. Press the red **STOP** button on the toolbar to stop the CPU. This will allow the debugger to update the open memory and register windows.
10. **Trace Filtering** The ability to hide unwanted information from the trace buffer to get a clearer picture cannot be overestimated. Trace filtering is easily accomplished by double-clicking in any column in the Trace window with the exception of the Time Stamp column. For example, to see only trace frames that contain C source lines, double-click inside any Source line column. In the menu that appears, select **Filter non-empty Source**.

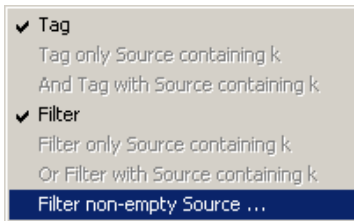


FIGURE 99 Filtering lines with non-empty source lines

The Trace window will filter out all frames without C source associated with them. With only lines of C code displayed, program execution will be easier to analyze.

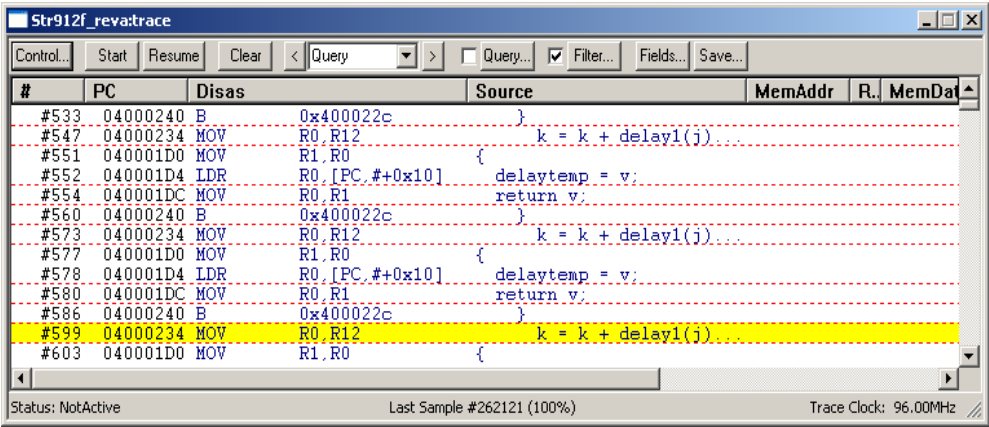


FIGURE 100 Filtering to show only the rows with C code

- 11. **Using Query** To return to the unfiltered view, uncheck the Filter check box on the toolbar. Click on the Query drop down list and change the query to look for the next **Module**. Keep pressing the forward [>] button until **processing (Buf1)** is found. Double click on the **PC** column of that trace line and from the menu select “**Filter only PC = main+0x6C**” (PC = 0x40000148). This will change the filter to show only calls to **processing**. This will allow us to measure the frequency of the calls and determine if the interval between calls is constant.
- 12. To find out how often the **processing** function is called, scroll the trace window to the right to see the **TStamp** column and double-clicking on it until it shows the time in delta mode and cycles [dt][cyc]. At the end of the process, we find that the call takes place every 32,708 CPU cycles. To convert this value to microseconds, click on the TStamp header and select the **usec** from the list.

Now double click anywhere in the time stamp column to change the display to [abs][us]. Scroll to the end of the buffer to see how much execution time is covered by the contents of the trace buffer. A 256K buffer, the smallest available, is capable of storing over 11 msec worth of a program running at 30 MHz. With the largest trace buffer of 4M, this capability is 16-fold larger.

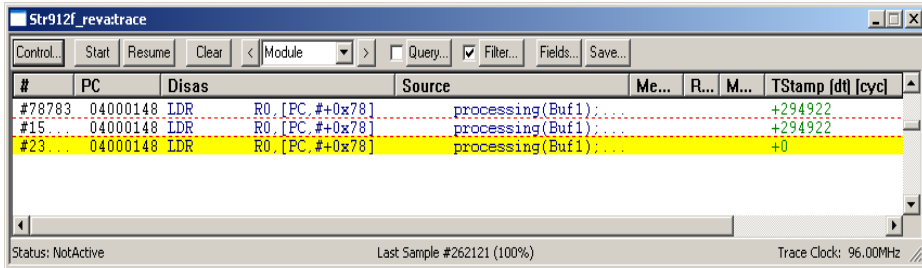


FIGURE 101 Filtered trace showing calls to only one function.

13. As you probably noticed, our experiment filled the trace buffer in an instant. This is because the buffer stored the samples indiscriminately without eliminating such uninteresting fragments of the application as loops waiting for ready bits or software delays. This can easily be amended by pre-filtering the data that comes into the trace buffer so that only the relevant information is retained. One way to do it is to set the trace to capture execution in PC address range that interests us.

Please uncheck the **Filter** to see all samples, and then click on the **CLEAR** button to erase the trace buffer. Now open the Control dialog box and select the **ETM Trace Settings** tab.

In the Settings tab, select the **Capture PC in Address Range** setting and enter the following addresses:

Begin: {"demo.c"}@29 End: {"demo.c"}@58

These settings will capture execution trace of the main.c program loop only, and any function calls outside of main will not be recorded. This is a great way to catch the profile of the application as you can see clearly the timing of the major system functions.

Click **OK** to save the settings. Click also the yellow **RST** (restart) button on the top toolbar. The RST button brings the application back to the initial point without resetting the CPU or reloading the application.

Click the **GO** button. Notice that the trace buffer fills much slower this time, as it needs to make selections in order to store only the lines in the main loop.

When the trace fills up and stops, stop the CPU and hit the Home button on your keyboard to see the beginning of the trace. The window should look similarly to that in FIGURE 102.

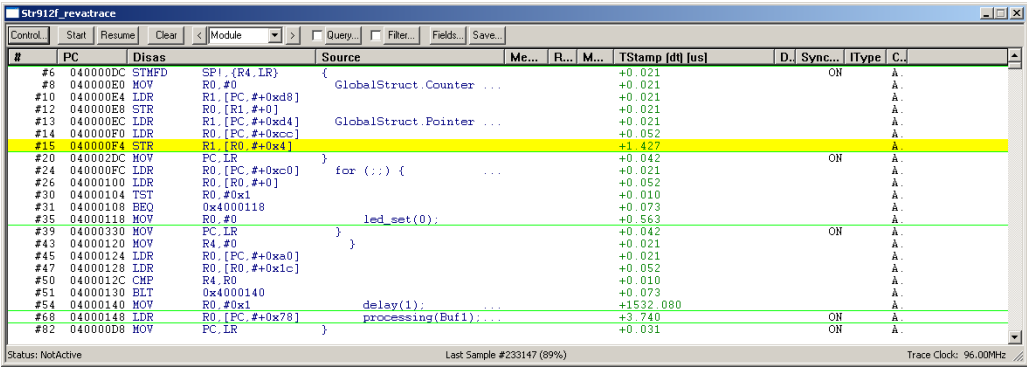


FIGURE 102 Filtering to show calls to one function only.

The green lines in the Trace window represent trace discontinuities. As expected, the trace started tracing from the first line of the main program (**main**) and stopped on main+18, which is there where the first function (**led_init**) was called. We find that this function stopped the trace for 1.427 μ s and started to capture again when it returned to the main program to execute the **for (; ;)** loop. The trace captured a few lines from main and stopped again when **led_set(0)** was called, which took .536 us. We can also see that the intended 1 ms **delay** function takes actually 1532.080 μ s. If this timing is important, the developer may need to adjust this value.

14. To eliminate any function from the trace, use the **Start/Stop Tracing on Address** feature. Also, it is necessary to enter the function's exit address in the trace **Begin** field and the function's entry point in the trace **End** field. Figure 103 shows an example with the delay functions filtered out, allowing the trace to capture over 622 msec of the run time.

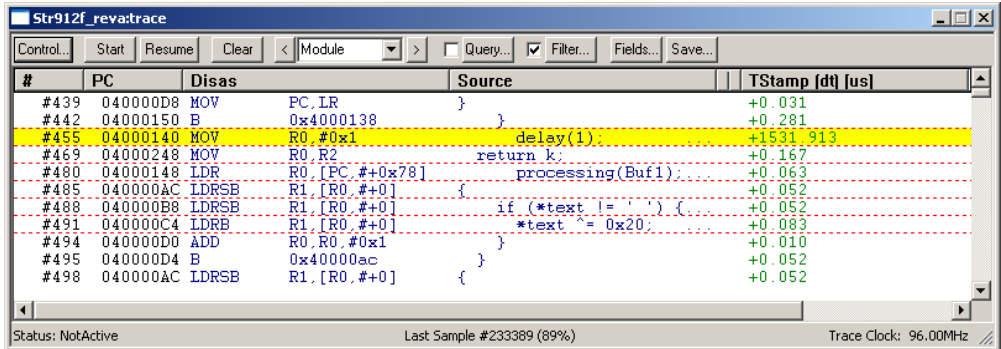


FIGURE 103 Trace with the delay function filtered out.

Data Tracing with Raisonance STR912F

Tracing of code execution is an excellent method for detecting many programming errors. However, without an ability to visualize data transfers, it may be hard to see what the application is actually doing. In typical C code, most problematic places are prologue and epilogue of functions where several 32-bit CPU registers are transferred to or from the stack with the STMIA and LDMIA instructions.

The Raisonance STR912F data sheets do not mention the capability of data tracing over the ETM, which may suggest that it is not supported, most likely due to the lack of bandwidth on the 4-bit ETM.

Nevertheless, our experiments show that if data access is not very frequent, some data variables may be successfully traced even on a 4-bit ETM port. Frequent variable access will overflow the on-chip ETM FIFO, causing loss of PC information. Therefore tracing variables may require a certain amount of caution.

In the ETM Control dialog box, you can choose to trace the variable(s) address or data or both. Since each variable address is 32-bit long and most data contain 32-bit values, up to sixteen clock cycles may be used just to export each variable. Thus, whenever possible, using byte variables will save a lot of bandwidth when tracing.

To trace what is read and written to the Buf1[] array, click on the **Control** button, select the **ETM Trace Settings** tab and make sure the **Capture Complete PC Range** is selected (FIGURE 104).

In the **Data Tracing** section right below, check all three boxes and enter the addresses **&Buf1[0]** and **&Buf1[14]** in the Begin and End fields, respectively.

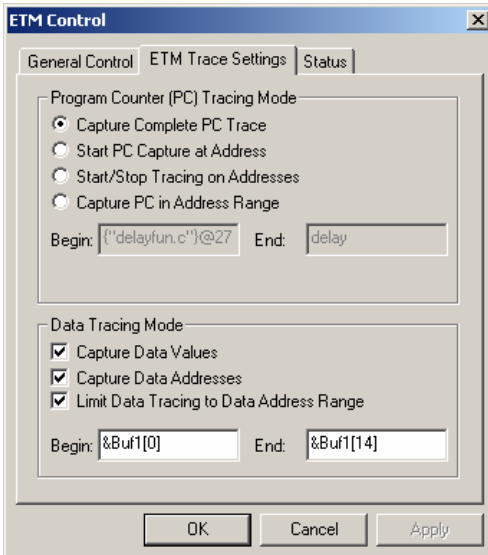


FIGURE 104 Adding tracing of variables to the ETM settings

Click **OK** to close the Control window. If the CPU is still running, stop it by pressing the **STOP** button.

Open the Command window (e.g., by pressing the CMD button on the tool bar) and type,

```
>delaycnt = 1
```

This instruction reduces the amount of the software delay to a minimum, so that the Buf1[] array is written more frequently and more of it will be included in the trace.

To restart trace capture when the CPU is running, press the **Start** button. If the CPU is stopped, press the **GO** button.

The trace will fill instantly and display new data. To find the data variables in the trace, double click in the **MemAddr** or **MemData** fields and select **Filter non-empty MemAddr**. This will only display the trace frames with valid MemAddr fields. In our exercise, only the Buf1[] writes and reads and the associated data values are shown (Figure 105).

#	PC	Disas	Source	MemAddr	RdWr	MemData
#319	processing+0xC	LDRSB R1, [R0, #+0]	if (*text != '...	Buf1[10]	Rd	0x54
#324	processing+0x18	LDRB R1, [R0, #+0]	*text ^= 0x20	Buf1[10]	Rd	0x54
#326	processing+0x20	STRB R1, [R0, #+0]		Buf1[10]	Wr	0x74
#336	processing	LDRSB R1, [R0, #+0]	{	Buf1[11]	Rd	0x45
#341	processing+0xC	LDRSB R1, [R0, #+0]	if (*text != '...	Buf1[11]	Rd	0x45
#346	processing+0x18	LDRB R1, [R0, #+0]	*text ^= 0x20	Buf1[11]	Rd	0x45
#348	processing+0x20	STRB R1, [R0, #+0]		Buf1[11]	Wr	0x65
#358	processing	LDRSB R1, [R0, #+0]	{	Buf1[12]	Rd	0x4D
#363	processing+0xC	LDRSB R1, [R0, #+0]	if (*text != '...	Buf1[12]	Rd	0x4D
#368	processing+0x18	LDRB R1, [R0, #+0]	*text ^= 0x20	Buf1[12]	Rd	0x4D
#370	processing+0x20	STRB R1, [R0, #+0]		Buf1[12]	Wr	0x6D
#380	processing	LDRSB R1, [R0, #+0]	{	Buf1[13]	Rd	0x53
#385	processing+0xC	LDRSB R1, [R0, #+0]	if (*text != '...	Buf1[13]	Rd	0x53
#390	processing+0x18	LDRB R1, [R0, #+0]	*text ^= 0x20	Buf1[13]	Rd	0x53
#392	processing+0x20	STRB R1, [R0, #+0]		Buf1[13]	Wr	0x73
#402	processing	LDRSB R1, [R0, #+0]	{	Buf1[14]	Rd	0x00

FIGURE 105 Filtered trace showing only read and write to the Buf1[] array operations.

To have the MemData column display in ASCII mode, click in the column header and check ASCII from the menu. When closed, the trace will show ASCII characters in place of HEX values, making reading the contents of the buffers containing text strings easier for a human reader.

In our example, switching to ASCII format reveals that the **processing()** function flips the case of the Buf1[] string (lower to upper or vice versa). See Figure 106.

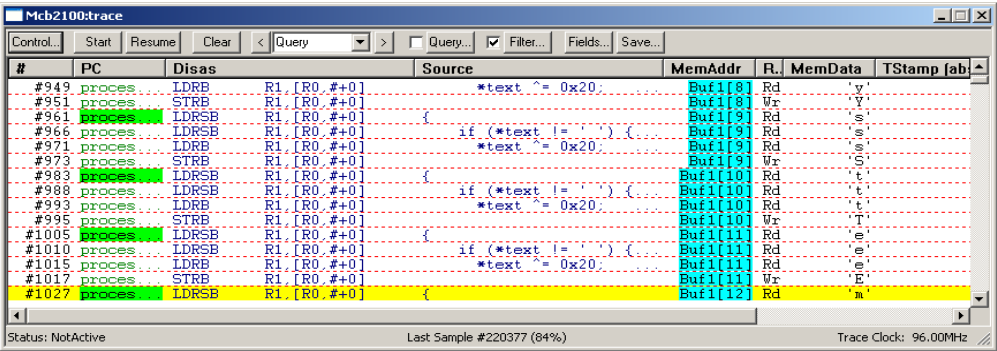


FIGURE 106 Filtered trace showing variable data in the ASCII format.

Changing Variables During Execution

Even though unrelated directly to the ETM trace, you may find this section pertinent and helpful. The information contained herewith is not easily located in the debugger documentation.

If our application runs in real-time, the blinking of the LEDs will not be perceptible, because we set the value of the delaycnt variable from 1. Let us return to the original value of 500 while the application is running.

Close the Trace window and press the **GO** button. The application starts running. Open the **Watch** window by clicking on the **WCH** icon on the toolbar. An empty Watch window appears. Right-click inside the window, then select **Add** from the pop-up menu. In the dialog box that appears, type the name **delaycnt**.

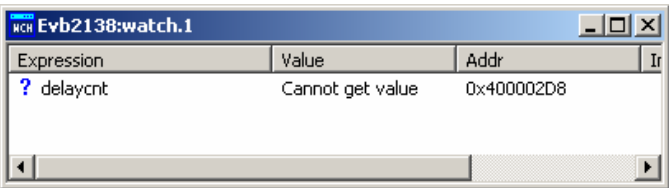


FIGURE 107 Filtered trace showing variable data in the ASCII format.

The variable `delaycnt` appears in the Watch window, but the value will not be read since the application is running (FIGURE 107). To force the reading of its value, select the variable in the Watch window and right-click again. Check the **Access in RUN mode** feature.

Now select the variable again with the mouse and press the Space bar on your keyboard. This will force the variable to be read showing the value of 1 (Figure 108).

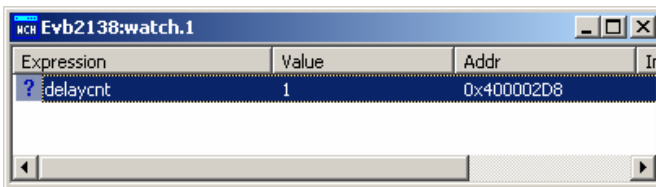


FIGURE 108 Selecting the `delaycnt` variable in the Watch window.

To change the variable, double click on its value in the Watch window and enter 500 in the entry field. The blinking of the LEDs will now be visible to the eye.

Variables can also be drag-and-dropped to the Watch window. To try this capability, we need to stop the CPU. Right-click in the Source Window and select **View Module** in the pop-up menu. When a list of the modules appears, select **demo.c** and click OK.

The Source window will show the beginning of the `demo.c` file.

To be compatible with older Signum emulators, a double-click on any line in the Source window triggers the option **Execute to this line**. This default setting may be changed by opening the Source window options menu with a right-click, and then selecting **Options** from the list and un-checking **Double click == GO TILL**. With this change, a double-click on a source line will select variables. Click on any GlobalStruct variable and drag it into the Watch window (Figure 109).

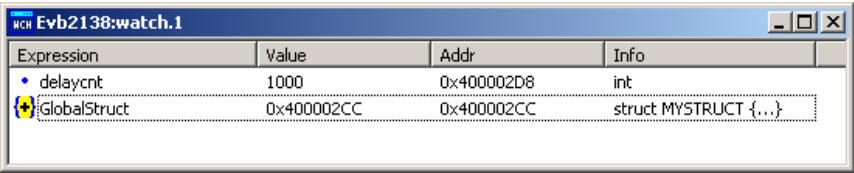


FIGURE 109 Selecting variables in the Watch window.

To expand the structure, click the yellow + next to the variable's name and make sure it is accessible in RUN mode, just as we did with the delaycnt variable.

Press **Go** to run the application again. You can now inspect the members of the structure while the application is running by selecting them in the Watch window and pressing the space bar (Figure 110).

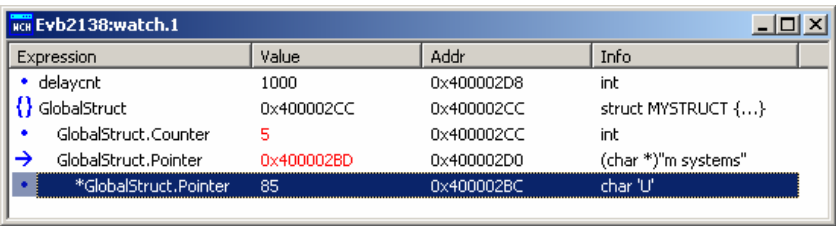


FIGURE 110 Selecting a member of a complex variable for inspection.

