LAUTERBACH
DEVELOPMENT TOOLS

Arm ETM Trace

MANUAL

Arm ETM Trace

Arm ETM Trace

Version 06-Jun-2024

History

08-Jul-22 New commands: ETM.CycleCountTickEnable and ETM.CycleCountTickRate.

Installation

Software Installation

The TRACE32 software for the ARM debugger includes support for the ETM trace. No extra software installation for the ARM-ETM trace is required.

Recommendation for Starting the Software

- Disconnect the debug cable from the target while the target power is off.
- Connect the host system, the TRACE32 hardware and the debug cable.
- Start the TRACE32 software.
- If possible connect the debug cable directly to the target. If there is no appropriate jack on your target, you can also connect it to the preprocessor.
- Connect the preprocessor to your target's trace port by using the mictor flex extension delivered with your preprocessor. For port sizes greater 16 bit you need to connect port "Trace B" as well, using a second mictor flex extension (LA-7991, LA-7992 and LA-7923 only).

  NOTE: The second flex extension has to be ordered additionally.
- Switch the target power ON.
- Run your start-up script.
- If supported by your preprocessor execute **Analyzer.AutoFocus**

Recommendation for Power Down

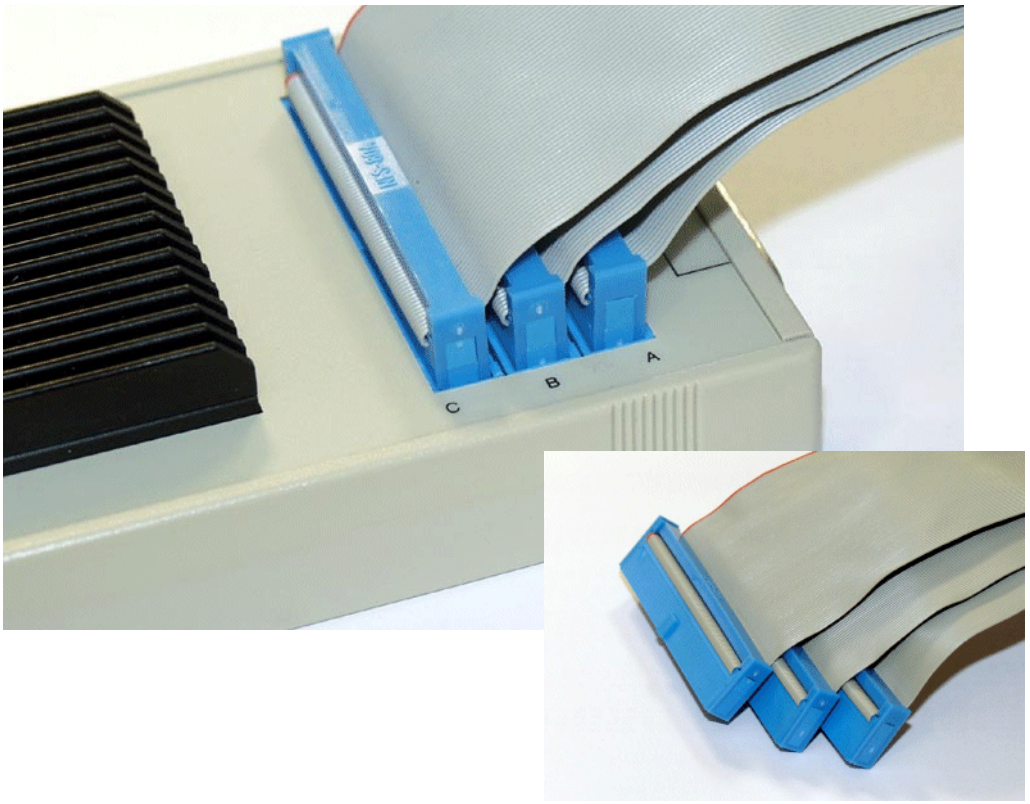
- Switch off the target power.
- Disconnect the debug cable and mictor flex extension from the target.

Hardware Installation

If a RISC TRACE module is used, please connect the PODBUS IN connector of the RISC TRACE module to the PODBUS OUT connector of the (POWER) DEBUG INTERFACE.

If a POWER TRACE PX or POWER TRACE II or POWER TRACE III is used please connect it to a POWER DEBUG PRO or POWER DEBUG II via the "PODBUS EXPRESS" connectors.

The preprocessor (small PBC / probe) has to be connected to **RISC TRACE**, **POWER TRACE**, **POWER TRACE PX**, **POWER TRACE II**, or **POWER TRACE III.** The three flat cables have different length and need to be connected without crossing:



The shortest cable needs to be connected to plug A, the middle to plug B and the longest to plug C.

You can identify the preprocessor version by typing **VERSION.HARDWARE** into the TRACE32 command line or compare your preprocessor with the pictures below. Preprocessor versions and a description of the main differences are described in the following:

Product Number	LA-7889	LA-7921	LA-7923	LA-7990
TRACE32 ID **Full rate** **Half rate** **DSP mode**	40	43 39 3A	46	58 57 59
Delivery year	2000-2008	since 2001	2001-2008	2004-2009
Serial number	none	since 04/2004	none	yes
Supported target voltage range [V]	2.5 … 3.3	1.8 … 3.3	2.5 … 3.3	1.8 … 3.3
Casing	none	since 04/2004	none	yes
Number of flat cables	3	3	3	2
Supported ETM port sizes	4/8/16	4/8/16	4/8/16/32	4/8
Supported ETM modes	Normal - Demux 4bit Full rate	Normal Mux Demux 4bit Full/Half rate	Normal Mux Demux Full/Half rate	Normal Mux - Full/Half rate
Maximum channel data rate	120 Mbit/s	200 Mbit/s	120 Mbit/s	270 Mbit/s
Input delay resolution	-	-	-	-
Termination	none	47 Ω Thevenin	none	47 Ω Thevenin
Threshold level	fixed	programmable	fixed	both, fixed and programmable
Input signals	2.5 V LVTTL 3.3 V LVTTL compatible	>0.5 V	2.5 V LVTTL 3.3 V LVTTL compatible	1.8 V LVTTL 2.5 V/3.3 V STL compatible

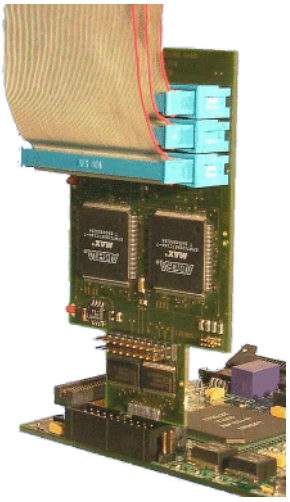
Product Number	LA-7991	LA-7991	LA-7991	LA-7992
TRACE32 ID	70 (OTP)	70	70	71
Delivery year	11/2004-12/2005	06-09/2005	since 08/2005	since 2006
Serial number	yes	yes	yes	yes
Supported target voltage range [V]	1.8 … 5	1.8 … 5	1.8 … 3.3	1.2 … 3.3
Casing	yes	yes	yes	yes (with fan)
Number of ribbon cables	3	3	3	3
Supported ETM port sizes [bit]	4/8/16/32	4/8/16/32	4/8/16/32	4/8/16/32
Supported ETM modes	Normal Mux Demux Full/Half rate	Normal Mux Demux Full/Half rate	Normal Mux Demux Full/Half rate	Normal Mux Demux Full/Half rate Continuous
Maximum line data rate	300 Mbit/s	350 Mbit/s	350 Mbit/s	600 Mbit/s
Input delay resolution	-	-	480 ps	78 ps
Termination	pluggable	pluggable	47 Ω Thevenin	47 Ω Thevenin
Threshold level	programmable	programmable	programmable	programmable
Input signals	> 0.5 V	>0.5V	>0.5 V	>0.5 V

Product Number	LA-7993	LA-7988
TRACE32 ID	72	ETM-HSSTP (74)
Delivery year	since 2010	since 05/2008
Serial number	yes	yes
Supported target voltage range [V]	1.2 … 3.3	0.1-0.7
Casing	yes (with fan)	yes (with fan)
Number of ribbon cables	3	3
Supported ETM port modes	1-40bit	1-4 lanes
Supported ETM modes	Normal*) Mux *) Demux *) Full/Half rate*) Continuous	Normal Continuous Bypass
Maximum line datarate	600 Mbit/s	6250Mbit/s
Coupling	DC	AC
Input delay resolution	78ps	-
Termination	47 Ω Thevenin	-
Threshold level	programmable	-
Connectorisation	QTH, 60pin	ERF8, 40pin
Input signals	> 0.5 V	> 0.5 V

*) For ETMv1 modes please contact support@lauterbach.com

The target hardware has to be equipped with a 38 pin mictor connector in order to connect the **Preprocessor for ARM-ETM 120**. For dimensions and target connector pinout of the preprocessor refer the chapter **Technical Data**.

All trace signals are connected after plugging the preprocessor into the target´s trace connector.



If it is not possible to directly plug the preprocessor to the trace target connector, a **Mictor Flex Extension (LA-1370)** can be used.

The debug cable has also to be connected to the hardware. Use one of the following connectors:

- the JTAG connector of your target
- the JTAG connector of the preprocessor

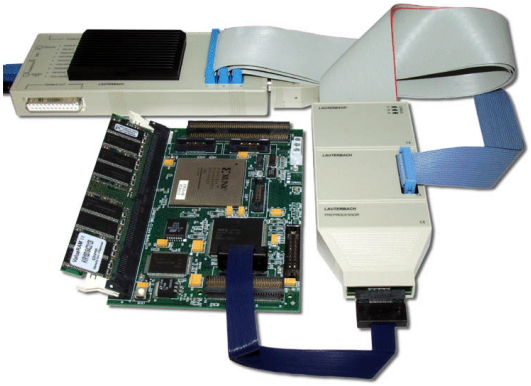
The JTAG connector on the **Preprocessor for ARM-ETM 120** is a 20 pin connector. The connector is located close to the trace target connector. If you are using a 14 pin debug cable you need to use a **JTAG ARM Converter 14-20 (LA-7747).**

If you power up the TRACE32 equipment and the **CONNECT ERROR** LED of the **RISC TRACE** module is glowing, please check all flat cables again.

If you power up the TRACE32 equipment and the **CON ERROR** LED of the **PowerTrace** module is glowing, please check the correct connection of all flat cables again.

The target hardware has to be equipped with a 38 pin mictor connector in order to connect the **Preprocessor for ARM-ETM 200**. For dimensions and target connector pinout of the preprocessor refer to the chapter **Technical Data**.

All trace signals are connected after plugging the preprocessor into the target´s trace connector.



If it is not possible to directly plug the preprocessor to the target´s trace connector, the Mictor Flex Extension can be used.

The debug cable has also to be connected to the hardware. Use one of the following connectors:

- the JTAG connector on your target
- the JTAG connector on the preprocessor

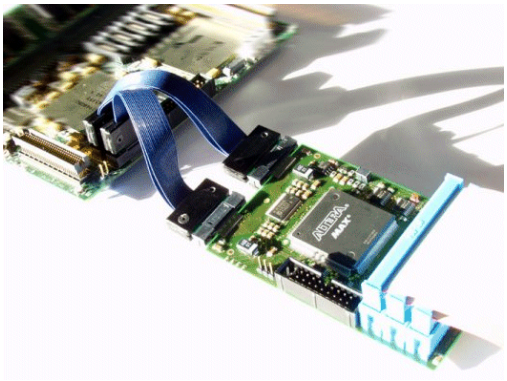
The JTAG connector on the **Preprocessor for ARM-ETM 200** is a 20 pin connector. The connector is located close to the blue flat cable connectors. If you are using a 14 pin debug cable you need to use a **JTAG ARM Converter 14-20 (LA-7747)**.

If you power up the TRACE32 equipment and the **CONNECT ERROR** LED of the **RISC TRACE** module is glowing, please check the flat cables again.

If you power up the TRACE32 equipment and the **CON ERROR** LED of the **PowerTrace** module is glowing, please check the correct connection of all flat cables again.

The target hardware has to be equipped with one or two 38 pin mictor connectors in order to connect this Preprocessor for ARM-ETM. For dimensions and target connector pinout of the preprocessor see the chapter **Technical Data**.

All trace signals are connected after plugging the preprocessor into the target´s trace connector.

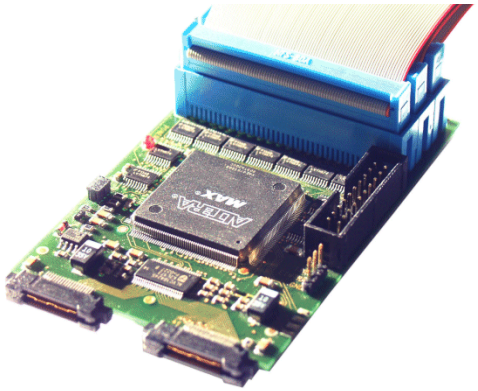


If it is not possible to plug the preprocessor to the trace target connector directly, use a Mictor Flex Extension. Let the second connector unused, if the target does not support 32 bit ETM modes.

Connecting the debug cable two ways are possible:

- the JTAG connector of your target
- the JTAG connector of the preprocessor

The JTAG connector of the preprocessor is a 20 pin connector. The connector is located close to the blue flat cable connectors (DEBUG). If you are using a 14 pin debug cable you need to use a **JTAG ARM Converter 14-20 (LA-7747)**.



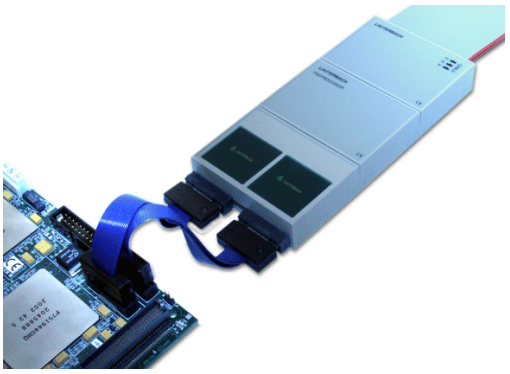
If you power up the TRACE32 equipment and the **CONNECT ERROR** LED of the **RISC TRACE** is glowing, please check the flat cables.

If you power up the TRACE32 equipment and the **CON ERROR** LED of the **PowerTrace** is glowing, please check the flat cables again.

The target hardware has to be equipped with one or two 38 pin mictor connectors in order to connect this Preprocessor for ARM-ETM. For dimensions and target connector pinout of the preprocessor refer to the chapter **Technical Data**

All trace signals are connected after plugging the preprocessor (Trace A) into the target´s trace connector.

If it is not possible to plug the preprocessor to the trace target connector directly, use a Mictor Flex Extension. Let the second connector (Trace B) unused, if the target does not support >16bit ETM modes.



Connecting the debug cable two ways are possible:

- the JTAG connector of your target
- the JTAG connector of the preprocessor

The JTAG connector of the preprocessor is a 20 pin connector. The connector is located under the blue flat cable connectors. If you are using a 14 pin debug cable you need to use a **JTAG ARM Converter 14-20 (LA-7747)**.

There are two types of LA-7991 that can be distinguished with **VERSION.HARDWARE**.

```
B::VERSION.HARDWARE
PowerDebug/USB
   USB-Flash V6.0
   USECORE: 1
   Debug Cable V2
   ETM-AF(OTP) (70)
```

The LA-7991 OTP is based on a one-time-programmable FPGA that became obsolete in 2005. In the **VERSION.HARDWARE** window it is marked '(OTP)'.

The LA-7991 OTP is succeeded by a re-programmable version.

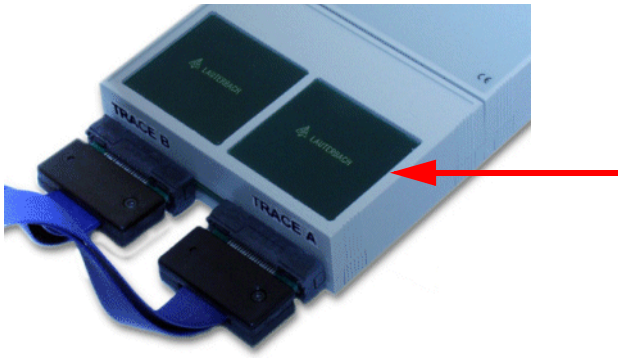
Both types have a similar performance, but there is a difference in the time resolution when it comes to adjustment of sampling points. You might notice this in the **Trace.ShowFocus** window. However this should not impact the actual trace result.

NOTE: The OTP version supports only ETM v1-3 pinouts, CTOOLs pinouts that follow the ETM v1-3 specification are supported (e.g. OMAP2420). However some CTOOLs pinouts are limited in their trace capabilities (e.g. OMAP1030): only simple tracing without trace compression is possible. Contact sales@lauterbach.com, if your preprocessor is OTP and you require an unsupported CTOOLs pinout.

Before 2006 both the OTP as well as its re-programmable successor were delivered. Starting 2006 only the re-programmable Preprocessor with integrated termination is delivered.

	Preprocessors delivered before 2006 might be marked "(OTP)" in the **VERSION.HARDWARE** window indicating that they are one-time-programmable. They support only ETM v1-3 pinouts (ARM7/9/10/11). Some CTOOLs pinouts do not follow the ETM v1-3 specification (e.g. OMAP1030). As a consequence only simple tracing without trace compression is possible. Contact sales@lauterbach.com, if your preprocessor is OTP and you require an unsupported CTOOLs pinout.

Most **Preprocessors for ARM-ETM with AUTOFOCUS** delivered in 2005 came with two pairs of Termination Daughter PCBs. One pair labeled '1.5 … 5 V", the other labeled '1.5 … 3.3 V' or '1.5 … 2.5 V' for early versions of the Preprocessor.



How to choose the proper termination PCB:

- Complete range version (1.5 … 5 V)

  The complete range version cuts the signal amplitude roughly in half. Hence it is save to use, even for 5 V targets, but it might not be optimal for target voltages below 2.5 V.

- Low voltage version (1.5 … 3.3 V or 1.5 … 2.5 V)

  The low voltage version does not affect the signal amplitude significantly. This module is usually showing better results in terms of data eye width, especially for target voltages below 2.5 V. For early versions of this termination module the signal amplitude after the termination PCB was conservatively specified for a maximum of 2.5 ,V which is why these modules were labeled "1.5 … 2.5 V". As more data became known, the maximum voltage could be increased to 3.3 V, so this module is now labeled "1.5 … 3.3 V".

  You must not use the low voltage termination for target voltages above 3.3 V!

- Integrated low voltage termination (1.5 … 3.3 V)

  The low voltage termination is now integrated in the main PCB. For target voltages greater 3.3 V a voltage converter (LA-7922) has to be used. However this voltage converter might reduce the maximum trace frequency. You should always contact support@lauterbach.com to discuss solutions for target voltages outside the specified range of 1.8 … 3.3 V or if you require a customized termination module.

Not all termination PCBs are compatible with all **Preprocessors for ARM-ETM with AUTOFOCUS**, so it is best to only use the termination PCBs that were delivered together with your preprocessor. Please refer to the table below to find out PCB ID combinations that are compatible. There is an ongoing effort to optimize the termination module for even higher frequencies, especially for the low voltage targets. In the table below bold Termination PCB IDs are indicating that the termination PCB contains the latest improvements. If you are unable to trace your target application at its maximum operating frequency and you do not have the latest available termination module, contact sales@lauterbach.com for delivery arrangements. Use the **Diagnosis Tool** to find out your preprocessors PCB IDs.

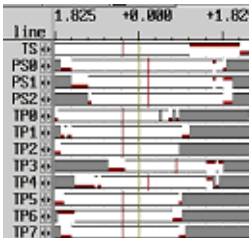
LA-7991 PCB ID	Termination PCB ID for 1.8 … 3.3 V	Termination PCB ID for 1.8 … 5 V
0x0 (OTP)	0x4	0x1
0x7 (OTP)	0x4	0x1
0xE (OTP)	0x6, 0xE, **0xF**	0x2
0x0	0xF	0x2
0x1	0xF (integrated)	not applicable

	Not all termination PCBs are compatible with all **Preprocessors for ARM-ETM with AUTOFOCUS**, so it best to only use the termination PCBs that were delivered together with your preprocessor. You must not use the low voltage termination for target voltages above 3.3 V

In case your **Preprocessors for ARM-ETM with AUTOFOCUS** came with Termination Daughter PCBs, you may wish to find out, which of the two termination PCB types best suits your needs. You can print out some data eye statistics on the area window by pressing the "Info" button of the **Diagnosis Tool** (after executing **Analyzer.AutoFocus**). Here is an example for a 1.8 V target:

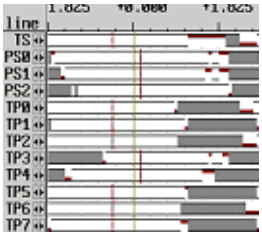
- Complete range version (1.5 … 5 V)

```
Active channels              : 12
Max. possible sampling points: 1812
Possible setup violations    :
BUS0 (rising edge)           : 674 sampling points
BUS0 (falling edge)          : 658 sampling points

36 percent of implemented time window has possible setup violations
```



- Low voltage version (1.5 … 2.5 V targets)

```
Active channels              : 12
Max. possible sampling points: 1812
Possible setup violations    :
BUS0 (rising edge)           : 573 sampling points
BUS0 (falling edge)          : 542 sampling points

30 percent of implemented time window has possible setup violations
```



The low voltage version has less setup violations, so the data eyes are broader and easier to sample, hence it is expected to be able to handle higher frequencies than the complete range version for that particular target.

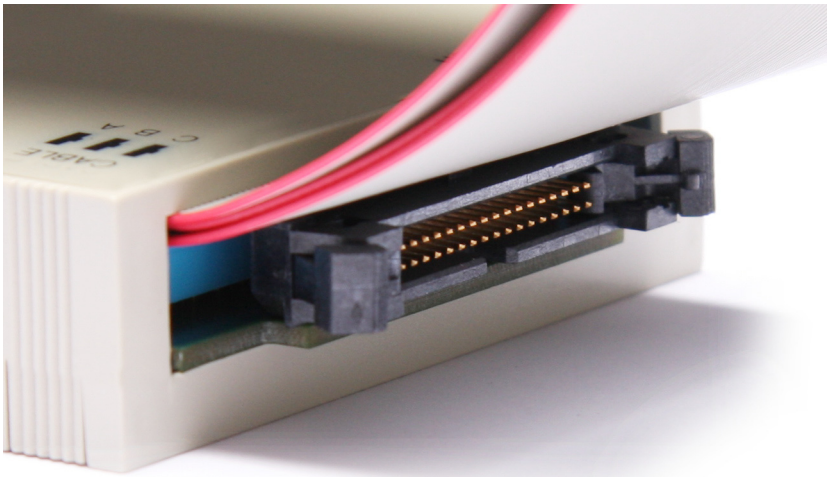
Preprocessor for ARM-ETM Autofocus II (LA-7992)

The Preprocessor for ARM-ETM Autofocus 2 is the next generation of Autofocus preprocessors. Its handling is similar to **ARM-ETM Autofocus (LA-7991)**

The TRACE32 online help provides a **"AutoFocus User's Guide"** (autofocus_user.pdf), please refer to this manual if you are interested in details about Preprocessor AutoFocus II .

The Preprocessor for ARM-ETM Autofocus MIPI is the next generation of Autofocus preprocessors. Its handling is similar to **ARM-ETM Autofocus (LA-7991)**

The JTAG connector of the preprocessor is a 34 pin connector. The connector is located under the blue flat cable connectors. A adapter is required, if you are using a debug cable with a non-MIPI connector (e.g. **ARM Converter ARM-20 to MIPI-34 (LA-3770))**.

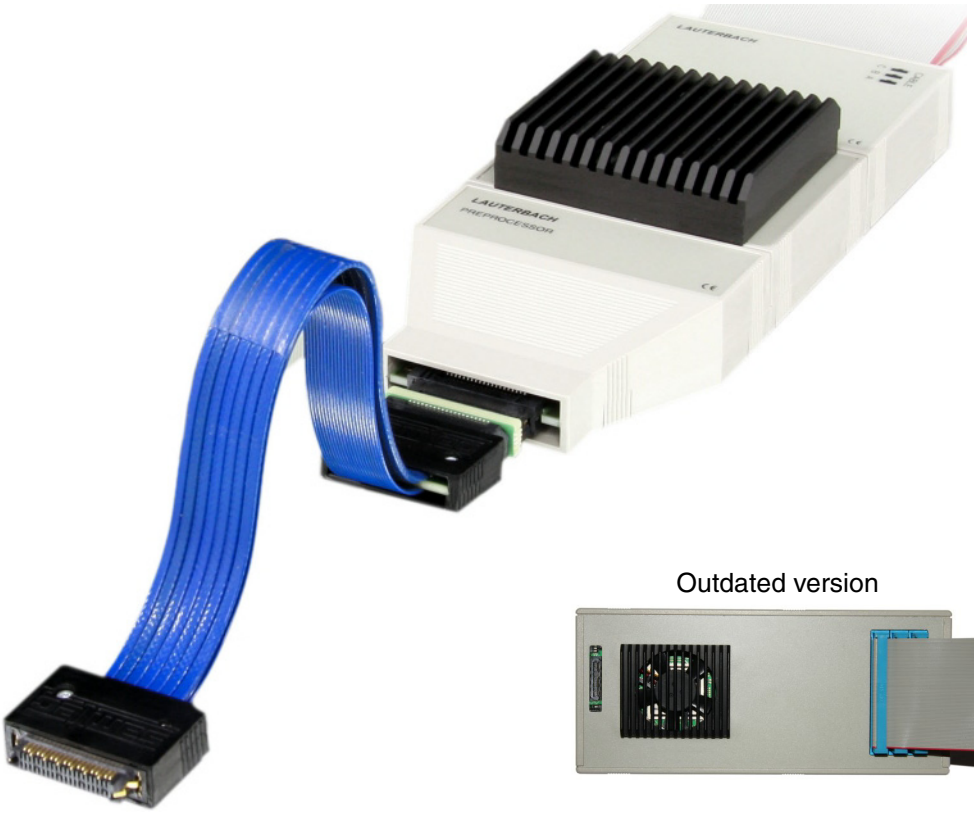


The TRACE32 online help provides a **"AutoFocus User's Guide"** (autofocus_user.pdf), please refer to this manual if you are interested in details about Preprocessor AutoFocus II .

Preprocessor for ARM-ETM HSSTP (LA-7988)

The HSSTP (High Speed Serial Trace Port) is different to the common parallel trace ports as ETM use. The Preprocessor for ARM-ETM HSSTP opens the way to receive trace data on a serial way at higher data rates.

The target hardware has to be equipped with a 40 pin ERM8 connector in order to connect this preprocessor for ARM-ETM HSSTP. For dimensions and target connector pinout of the preprocessor see the chapter **Technical Data**.



Outdated version

The outdated version is no longer available, but still supported.

All trace signals are connected after plugging the preprocessor into the target´s trace connector.

In case of no separate JTAG connector on the target, the debug cable can be conneted with the preprocessors JTAG connector on the back side.

PowerTrace Serial 4 GigaByte for ARM-ETM HSSTP and PCIe

Please refer to **“PowerTrace Serial User’s Guide”** (serialtrace_user.pdf).

Startup Script

Example ETMv1

The following ETM settings are required:

- Define the width of the trace port with the command **ETM.PortSize**.
- Define the mode of the trace port with the command **ETM.PortMode**.
- Define if the ETM works in HalfRate mode or not with the command **ETM.HalfRate**.
- Turn on the ETM with the command **ETM.ON**.

Further the target must be configured:

- Setup I/O-ports
- Setup board (buffers, jumpers, etc.)
- Set operating frequency

Finally the preprocessor needs to be set up correctly:

- Setup AUTOFOCUS hardware with **Analyzer.AutoFocus**
- Also check the trace channel with **Analyzer.TestFocus** (included in Analyzer.AutoFocus)

This example is made for an ARM9 target (e.g. CM966E-S by ARM):

```
; JTAG DEBUGGER SETUP
SYStem.RESet                              ; Initialize system
SYStem.JtagClock RTCK                     ; Select JTAG clock
SYStem.CPU ARM966E                        ; Select CPU type
SYStem.Up                                 ; Start debugger

; TARGET SETUP
Data.Set SD:0x10000014 %LE %L 0a05f       ; Unlock target registers
Data.Set SD:0x10000008 %LE %LONG 20       ; Set target frequency
SYStem.Option.BigEndian OFF               ; Set endianism

; PROGRAM SETUP
Data.LOAD.ELF armle.axf /SPATH /LPATH     ; Load example PRACTICE script
Register.Set PC main                      ; Set program counter to program
                                          ; start
Register.Set R13 0x1000                   ; Initialize stack pointer
```

```
; ETM SETUP
ETM.PortSize 16                         ; Set the trace port width to 16

ETM.PortMode Normal                     ; Set the trace mode to Normal
                                        ; mode

ETM.HalfRate OFF                        ; Set full rate mode for ETM

ETM.DataTrace Both                      ; Trace Address and Data
ETM.ON                                  ; Turn ETM on

; Configure Preprocessor                ; set threshold to 50% of the
Analyzer.THreshold VCC                  ; voltage level of pin12 of the
                                        ; target connector
Analyzer.TERMination ON                 ; connect termination voltage
                                        ; during trace
Analyzer.AutoFocus                      ; Set threshold and sampling
                                        ; points automatically

; Test trace port                       ; Load, execute and trace test
Analyzer.TestFocus                      ; program and report errors

; END OF SCRIPT
ENDDO                                   ; End of script
```

Don’t forget to check the ETM port with **Analyzer.TestFocus**. The check must always finish with success.

Example HSSTP

The following ETM settings have to be done:

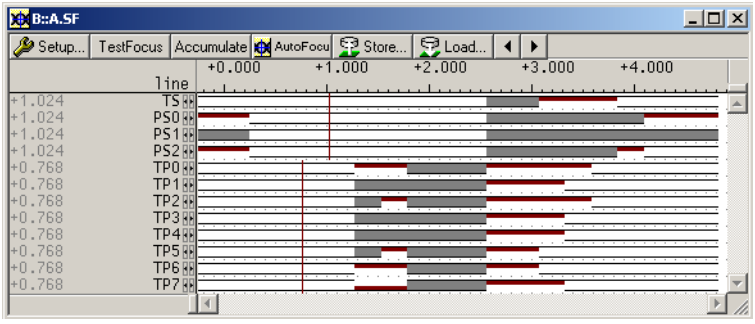
- Define the width of the trace port with the command **ETM.PortSize**.
- Define the mode of the trace port with the command **ETM.PortMode**.
- Define TPIU ETM register base
- Check HSSTP registers (PHY/Config)
- Turn on the ETM with the command **ETM.ON**.
- Finally check the ETM port with **Analyzer.TestFocus**

This example is made for an Cortex-R4 target:

```
; JTAG DEBUGGER SETUP
SYStem.RESet                                    ; Initialize system
SYStem.JtagClock RTCK                           ; Select JTAG clock
SYStem.CPU CORTEXR4                             ; Select CPU type

SYStem.CONFIG MEMORYACCESSPORT 0
SYStem.CONFIG DEBUGACCESSPORT  1
SYStem.CONFIG COREBASE APB:0x8000A000

SYStem.CONFIG ETMBASE APB:0x80006000            ; Define
SYStem.CONFIG FUNNELBASE APB:0x80004000
SYStem.CONFIG ETMFUNNELPORT 0
SYStem.CONFIG TPIUBASE  APB:0x80003000

SYStem.Up                                       ; Start debugger

; PROGRAM SETUP
Data.LOAD.ELF armle.axf /SPATH /LPATH           ; Load example PRACTICE
Register.Set PC main                            script
Register.Set R13 0x1000                         ; Set program counter to
                                                ; main
                                                ; Initialize stack
                                                ; pointer

; ETM SETUP
ETM.PortType HSSTP
ETM.PortSize 3LANE                              ; 3 lanes
ETM.PortMode 6000Mbps                           ; 6 Gbps
ETM.DataTrace Both                              ; Trace Address and Data
ETM.ON                                          ; Turn ETM on

; HSSTP CHANNEL TRAINING
Data.Set APB:0x8000D000 %LE %LONG 0xc           ; reset STP
Data.Set APB:0x8000D000 %LE %LONG 0xd           ; enable init sequence

IF Analyzer.ISCHANNELUP()                       ; Channel up?
(
 Data.Set EAPB:0x8000D000 %LE %LONG 0xf         ; enable transmission
 PRINT "Channel is up"
)
ELSE
(
 PRINT "Channel up failed!"
)
```

For the Preprocessors ARM-ETM **without** AUTOFOCUS at the most two settings need to be stored that enable restoring the previous configuration of the Preprocessor: use **Analyzer.TERMination ON | OFF** and **Trace.THreshold <value>** to restore settings from previous sessions.

For Preprocessors for ARM-ETM **with** AUTOFOCUS you can use the **Store** and **Load** buttons in the **Trace.ShowFocus** window to store settings of the current session or restore settings from a previous session.



Trace.ShowFocus as it appears for a re-programmable LA-7991

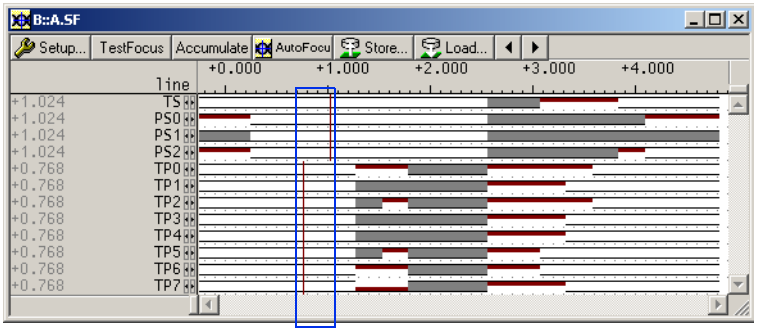
Pressing the **Store** button will call **STOre** *<file>* **AnalyzerFocus** and generate a PRACTICE script similar to this:

```
B::

IF ANALYZER()
(
     ANALYZER.TERMINATION ON             ; connect termination voltage
                                         ; during trace
     ANALYZER.THRESHOLD  1.19 0.99       ; clock reference voltage
                                         ; = 1.19 V
                                         ; data reference voltage
                                         ; = 0.99 V

      ANALYZER.SAMPLE    TS     -0.219    ; Store trace channel sampling
      ANALYZER.SAMPLE    PS0    -0.219    ; points
      ANALYZER.SAMPLE    PS1    +0.365
      ANALYZER.SAMPLE    PS2    +0.365
      ANALYZER.SAMPLE    TP0    -1.387
      ANALYZER.SAMPLE    TP1    -1.387
      ANALYZER.SAMPLE    TP2    -1.387
      ANALYZER.SAMPLE    TP3    -1.387
      ANALYZER.SAMPLE    TP4    -0.803
      ANALYZER.SAMPLE    TP5    -1.387
      ANALYZER.SAMPLE    TP6    -1.387
      ANALYZER.SAMPLE    TP7    -1.387
  )

ENDDO                                    ; End of script
```

In following sessions the settings can be restored either by using the **Load…** button or simply by including the PRACTICE script in your regular setup script.

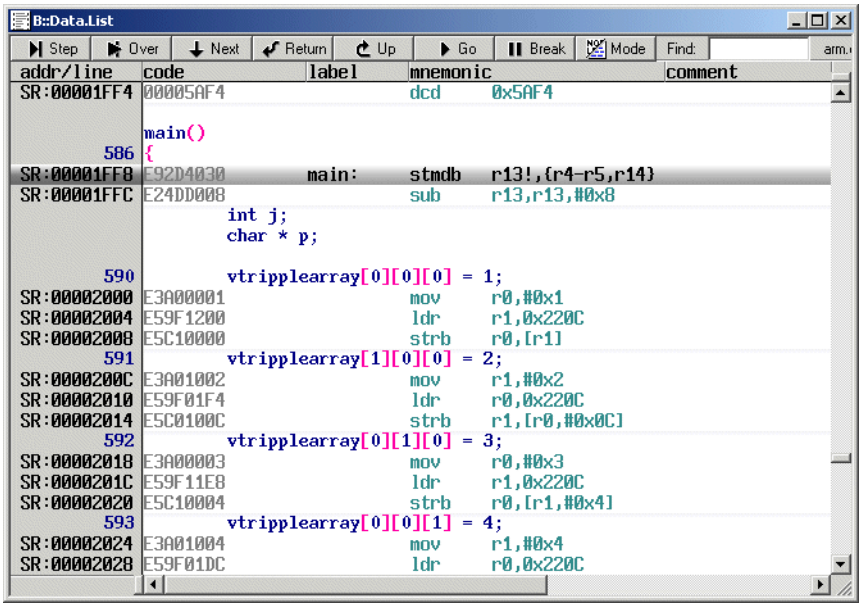
It is not recommended to manually edit the data related to the sampling points, instead the **Trace.ShowFocus** window should be used:

- Use the left / right arrows to adjust the clock delay (all sampling points will be moved globally)
- You may move individual channel sampling points to the left or right by double-clicking a position within the rectangle you can think of being drawn around all sampling points in the **Trace.ShowFocus** window (the blue rectangle in the picture below). In the example below you could move TS and/or PS[2:0] one position to the left and/or TP[7:0] one position to the right.

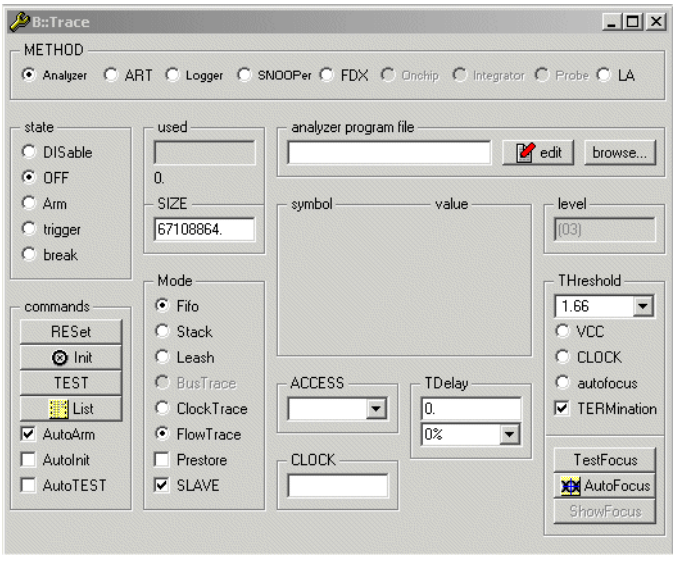


Displaying Trace Results

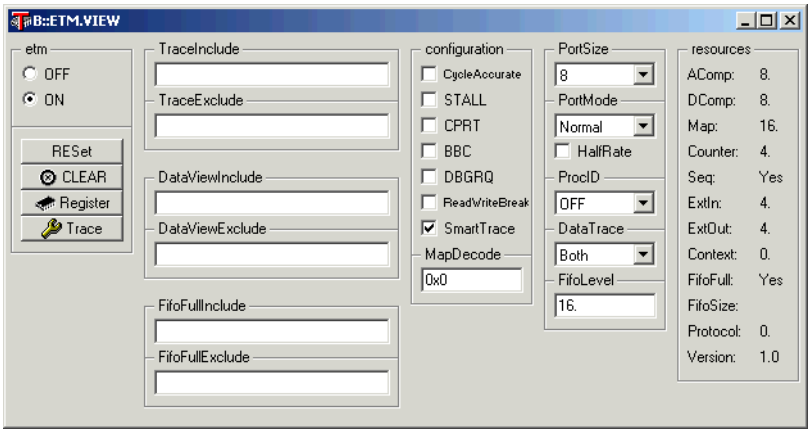
After the PRACTICE script was started by using **Run Script** in the **File** menu or by entering the command **DO** *<file>*, display the source listing by using **List Source** from the **View** menu or by entering the command **Data.List**.



Open the Trace setup window by using **Configuration** from the **Trace** menu or by entering the command **Trace.state**.



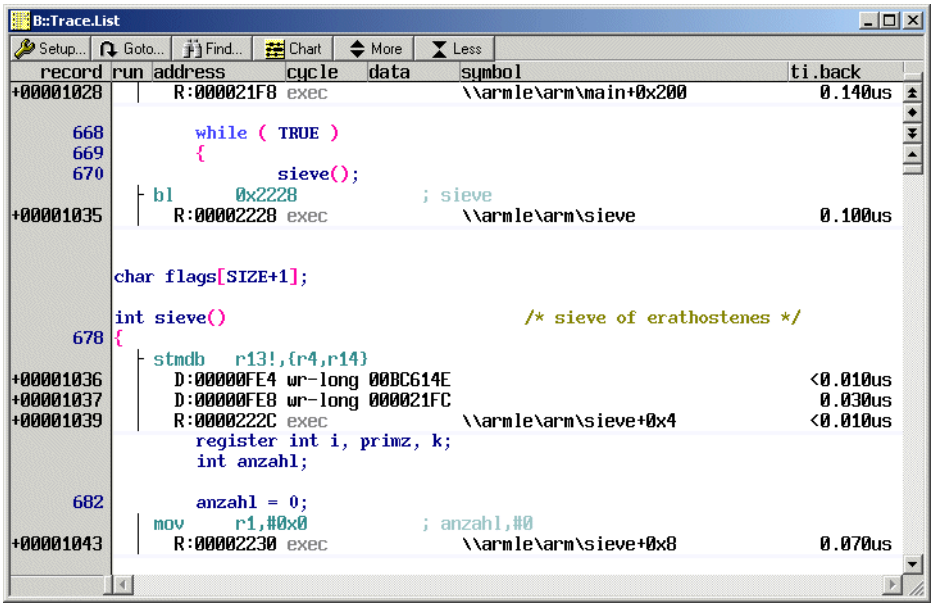
Open the ETM setup window by using **ETM Settings** in the **Trace** menu or by entering the command **ETM.state**.



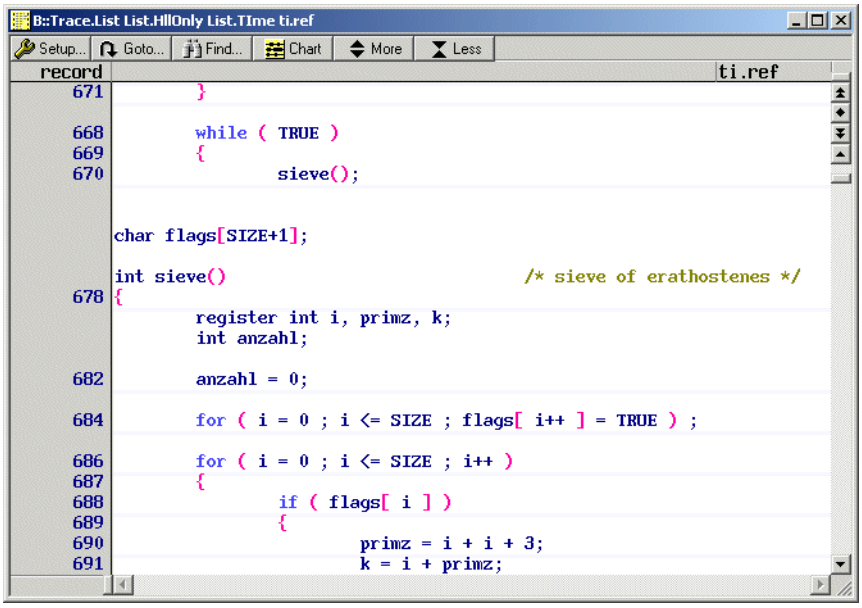
Type **Go** **`sieve`** into the command line and the CPU will run until the entry of the function sieve and the ***used*** field of the ***Trace.state*** window shows the amount of records that were sampled into the trace buffer.

If **AutoInit** is ON in the **commands** field of the **Trace.state** window the trace contents is cleared at every program start. Enable this feature by clicking to the check box **AutoInit** in the **Trace.state** window. Type **Go** **`sieve`** again and the function sieve will be executed once. The trace is filled with the program flow of the function sieve only.

To display the trace content use **List->Default** in the **Trace** menu or enter the command **Trace.List**.



For a pure HLL trace use **List->HLL Source Only** in the **Trace** menu.

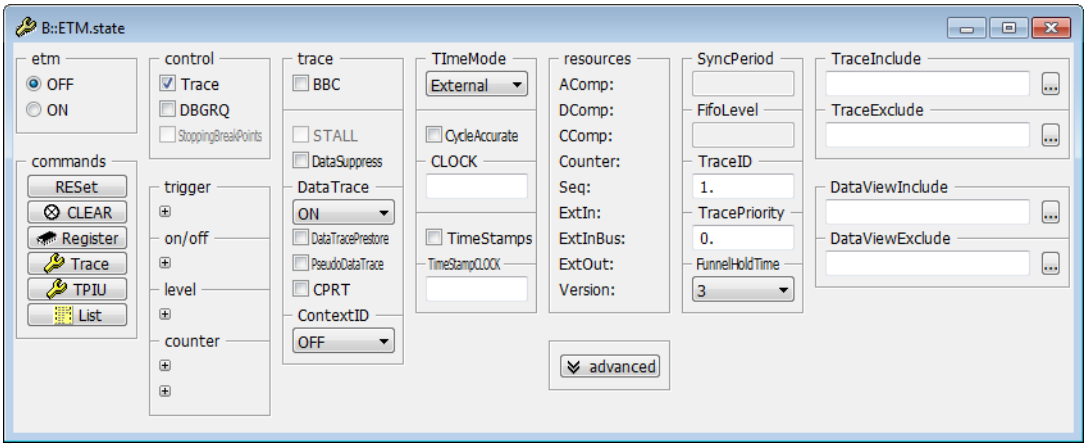


If undefinable errors occur in the trace display refer to the commands:

- **Analyzer.THreshold**
- **Analyzer.TERMination**

Supported Features

The features of the ARM-ETM trace mainly depend on the implementation of the Embedded Trace Macrocell (ETM). All trigger and filter features are provided by the ETM. To get information about the available resources of the ETM it is possible to read out the configuration register. Use **ETM Settings** in the **Trace** menu or enter the command **ETM.state** to open the **ETM.state** window.

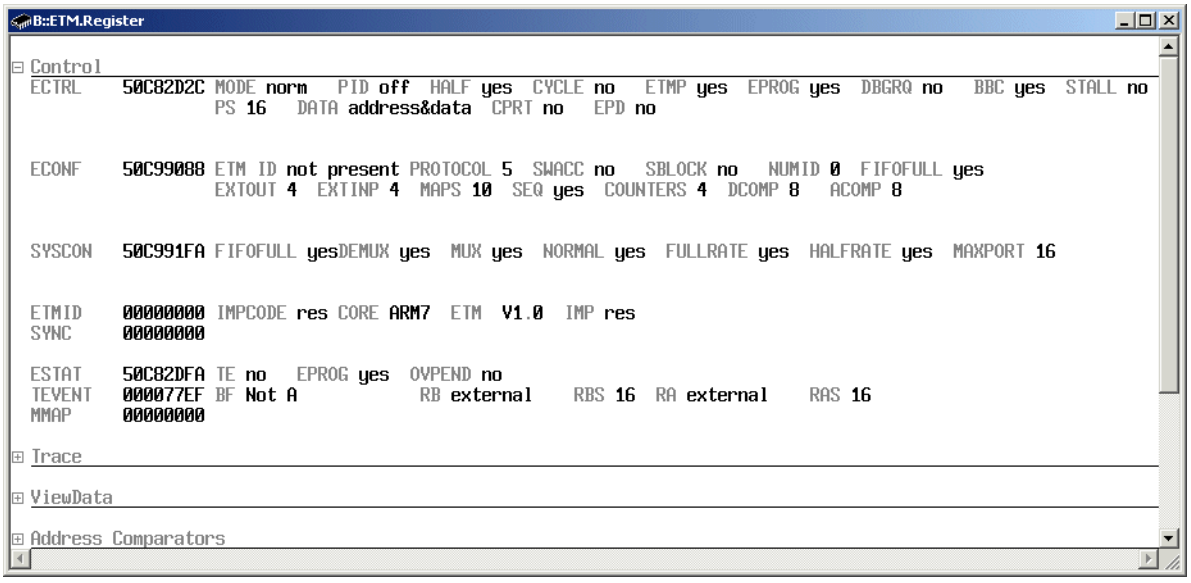


The right side of the window shows the list of all **resources** of the ETM:

AComp	Number of pairs of address comparators
DComp	Number of data comparators
CComp	Number of Context ID comparators
Map	Number of memory map decoders
Counter	Number of counters
Seq	Sequencer available
ExtIn	Number of external inputs
ExtIntBus	Extended external bus
ExtOut	Number of external outputs
FifoFull	FIFOFULL Logic of ETM available or not
Fifosize	Number of bytes of ETM FIFO
Protocol	Protocol version
Version	ETM version

Register Encoding	Function	Description
0000 0000	ETM control	Controls the general operation of the ETM
0000 0001	ETM config code	Holds the number of each resource
0000 0010	Trigger event	Holds controlling event
0000 0011	Memory map decode control	Configures the map decoder
0000 0100	ETM status	Holds pending overflow status bit
0000 0101	Reserved	
0000 0110	Reserved	
0000 0111	Reserved	
0000 1000 0000 1001	TraceEnable event TraceEnable region	Holds enabling event Holds include/exclude region
0000 1010 0000 1011	FifoFull region FifoFull level	Holds include/exclude region Holds the level below which the FIFO is considered full
0000 1100 0000 1101 0000 1110 0000 1111	ViewData event ViewData control 1 ViewData control 2 ViewData control 3	Holds the enabling event Holds include/exclude region Holds include/exclude region Holds include/exclude region
0001 xxxx 0010 xxxx	Address comparator 1-16 Address access type 1-16	Holds the address of comparison Holds the type of access
0011 xxxx 0100 xxxx	Data comparator values Data comparator masks	Holds the data to be compared Holds the mask for the data access
0101 00xx 0101 01xx 0101 10xx 0101 11xx	Initial counter value 1-4 Counter enable 1-4 Counter reload 1-4 Counter value 1-4	Holds initial value of the counter Holds counter clock enable/event Holds counter reload event Holds current counter value
0110 0xxx	Sequencer state/control	Holds the next state triggering events
0110 10xx	External output 1-4	Holds controlling event for each output
0110 11xx	Reserved	
0111 0xxx	Implementation specific	8 implementation specific register
0111 1xxx	Reserved	

The **ETM registers** can be displayed by pushing the **Register** button in the **ETM.state** window or by using the command **ETM.Register**:



The window shows a tree display of all control register groups. Details about a special group can be displayed by clicking to the small + sign beside the group name.

A modification of each register is possible by a simple double click on the value. The following command is automatically generated in the command line:

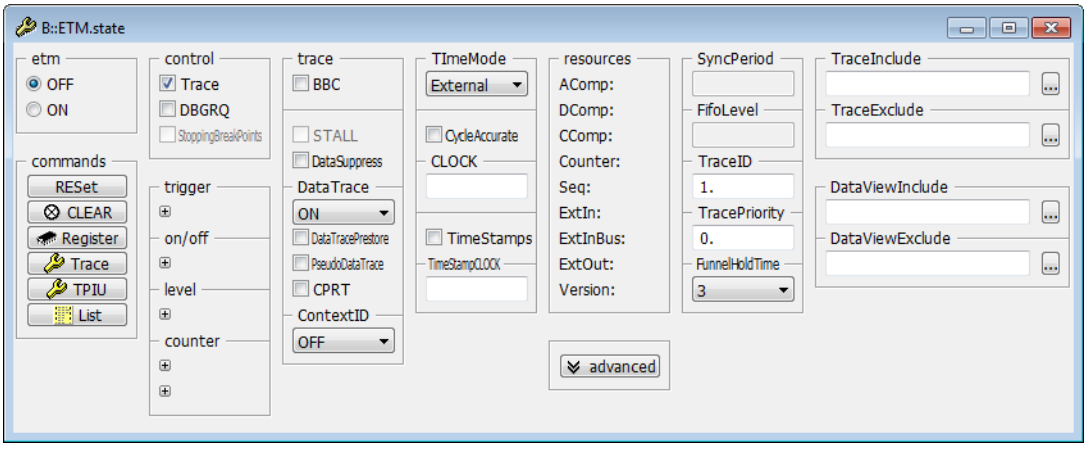
PER.Set EETM:<*register_address*> **%Long** <*value*>

ETM registers can be read and modified while the program execution is running.

It is also possible to use the **ETM.Set** command to modify the ETM registers. A full description of all available commands is in chapter **Commands**.

ETM — Embedded Trace Macrocell (ETM)

For configuration, use the TRACE32 command line, a PRACTICE script (*.cmm), or the **ETM.state** window.



The following TRACE32 commands are available to configure the ETM:

See also

- ETM.AbsoluteTimestamp
- ETM.BBC
- ETM.CLOCK
- ETM.CPRT
- ETM.CycleCountTickRate
- ETM.DataViewExclude
- ETM.FifoFullInclude
- ETM.INSTP0
- ETM.OFF
- ETM.PortMode
- ETM.PseudoDataTrace
- ETM.RefClock
- ETM.ReturnStack
- ETM.state
- ETM.TimeStampCLOCK
- ETM.TraceCORE
- ETM.TraceID
- ETM.TracePriority
- ETM()
- ETM.PROTOCOL()
- ETM.AddressMunging
- ETM.BBCExclude
- ETM.COND
- ETM.CycleAccurate
- ETM.DataSuppress
- ETM.DataViewInclude
- ETM.FifoLevel
- ETM.LPOVERRIDE
- ETM.ON
- ETM.PortRoute
- ETM.QE
- ETM.Register
- ETM.Set
- ETM.StoppingBreakPoints
- ETM.TimeStamps
- ETM.TraceDataPriority
- ETM.TraceInclude
- ETM.TraceRESet
- ETM.ADDRCOMP()
- ETM.ATBTrigger
- ETM.BBCInclude
- ETM.ContextID
- ETM.CycleCountThreshold
- ETM.DataTrace
- ETM.DBGRQ
- ETM.FunnelHoldTime
- ETM.MapDecode
- ETM.PortDisable
- ETM.PortSize
- ETM.QTraceExclude
- ETM.ReserveContextID
- ETM.SmartTrace
- ETM.SyncPeriod
- ETM.TimeStampsTrace
- ETM.TraceERRor
- ETM.TraceNoPCREL
- ETM.TRCIDR
- ETM.ADDRCOMPTOTAL()
- ETM.AUXCTLR
- ETM.CLEAR
- ETM.CORE
- ETM.CycleCountTickEnable
- ETM.DataTracePrestore
- ETM.FifoFullExclude
- ETM.HalfRate
- ETM.NoOverflow
- ETM.PortDisableOnchip
- ETM.PowerUpRequest
- ETM.QTraceInclude
- ETM.RESet
- ETM.STALL
- ETM.TImeMode
- ETM.Trace
- ETM.TraceExclude
- ETM.TraceNoSPREL
- ETM.VMID
- ETM.FIFOFULL()

- ▲ 'ETM Functions' in 'General Function Reference'
- ▲ 'Release Information' in 'Legacy Release History'

ETM.AbsoluteTimestamp — Absolute cyclecount pakets

Format: **ETM.AbsoluteTimestamp** [**ON** | **OFF**]

OFF Cycle counts in cycle accurate tracing mode are relative (default).

ON Cycle counts in cycle accurate tracing mode are absolute. This is the behavior of some (non ARM) ETM units.

See also

- ETM
- ETM.state

ETM.AddressMunging — Dig endian address munging

Format: **ETM.AddressMunging** [**ON** | **OFF**]

Default: OFF.

This command handles big endian address munging in ETM.

See also

- ETM
- ETM.state

Format: **ETM.ATBTrigger** [**ON** | **OFF**]

Configures the ETMv4 to drive an ATB trigger on event 0. This means that a trace trigger occurring in the ETM, is transported to the trace sink (TPIU, ETB, ETF, or ETR) via the CoreSight Advanced Trace Bus (ATB). You need to configure this option manually only for advanced operations. (See below.)

- On ARM chips with CoreSight ETMv3 or PTM (e.g. Cortex-A9) **ETM.ATBTrigger** is not available (or has no effect). Thus (except for Cortex-M) you have to configure the Cross Trigger Interfaces (CTI) manually to transport a trace trigger from the ETM to the trace port (TPIU) or onchip trace buffer (ETB/ETF/ETR) via the CoreSight Cross Trigger Matrix (CTM).
- On ARM chips without CoreSight debug infrastructure (ARM9 / ARM11) this option is not required.
- On ARM chips with ETMv4 (e.g. Cortex-R7/R8/R52, Cortex-A3x/A5x/A7x) setting **ETM.ATBTrigger** to **ON** configures the ETM to transport a trace trigger via the CoreSight Advanced Trace Bus (ATB).

NOTE: While normal trace data is usually buffered before it is emitted by the trace port, the ATB trigger is normally not buffered. This means that a trace trigger might be received before the associated trace data is received.

You can configured an event causing a trace trigger in the ETM by using the either the command **Break.Set /TraceTrigger** or the advanced command **ETM.Set Trigger**. Both commands set automatically **ETM.ATBTrigger** to **ON**.

When configuring an ETM trigger with **ETM.Set Trigger** you may use **ETM.ATBTrigger OFF**, to disable trigger propagation via ATB. This makes sense, if you prefer to transport you trigger through the cross trigger system (CTI & CTM) e.g. to stop a core directly when an ETM trigger occurs.

Example 1: Create a trace trigger when an instruction is fetched at address 0x6300100 on Cortex-R7.

```
;Trace trigger is generated on Cortex-R7 when an instruction is fetched at
;0x6300100. This command sets "ETM.ATBTrigger ON" automatically.
Break.Set 0x6300100 /Program /TraceTrigger

;Configures the trace-sink to stop the recording 2400 trace records after
;the trace trigger was received
Trace.TDelay 2400.
```

Be aware that the ETMv4 of a Cortex-R and Cortex-M have (unfortunately) a "visible speculation depth" at its output. Thus, for these cores it's recommended to generate a trace-trigger via a SingleShot comparators. (See example below.)

Example 2: Generate a trace trigger when the execution of an instruction was confirmed on Cortex-R7.

```
;Clear previous ETM.Set settings
ETM.CLEAR

;Configure 1st address comparator to raise an event on "execution" of
;program address 0x6300100
ETM.Set Address 1  Execute  0x6300100

;Configure 1st SingleShot comparator to confirm the execution
ETM.Set SingleShot 1  Address 1

;Generate an ETM trace trigger when 1st SingleShot comparator fires.
;Automatically sets "ETM.ATBTrigger ON".
ETM.Set Trigger SingleShot 1

;Configures the trace-sink to stop the recording 2400 trace records
;after the trace trigger was received
Trace.TDelay 2400.
```

On Cortex-A you won't need the SingleShot comparators and both examples would generated the trigger only when the instruction at address 0x6300100 was really executed. This is because the ETMv4 has (luckily) no "visible speculation depth" on Cortex-A.

Example 3: Stop Cortex-R7 core when any instructions inside address range 0x1000++0xfff was executed.

```
;Clear previous ETM.Set settings
ETM.CLEAR

;Configure 1st address range comparator
ETM.Set Range 1  Execute 0x1000++0xfff

;Use SingleShot to confirm the execution
ETM.Set SingleShot 1  Range 1

;Send ETM trigger when SingleShot fires.
ETM.Set Trigger SingleShot 1

;Don't propagate trigger via trace bus
ETM.ATBTrigger OFF

;Enable CTI of core and ETM (0x80918000 is here the base address of the
;CTI, this address differs from chip to chip)
Data.Set EDAP:0x80918000 %Long 1

;Send ETM trigger (which uses CTITRIGIN[2] on Cortex-R7) to CTM channel 2
Data.Set EDAP:0x80918028 %Long 4

;Receive CTM channel 2 on CTITRIGOUT[0], which is connected to the core's
;EDBGRQ signal (whish stops the core)
Data.Set EDAP:0x809180A0 %Long 4
```

See also

- ETM
- ETM.state

ETM.AUXCTLR Set ETMv4 implementation-specific auxiliary control register

Format: **ETM.AUXCTLR** *<value>* (ETM≥4.0)

Sets the value of the ETMv4 auxiliary control register "TRCAUXCTLR".

The function of this register is not defined by the ETMv4 specification, but can be used by any implementation of the ETMv4 for implementation-specific purposes.
E.g.: Bit 0 and 1 are defined for Cortex-R7 (see "CoreSight ETM-R7 Technical Reference Manual")

TRACE32 will only write to the ETMv4 register TRCAUXCTLR, if you have specified a value with **ETM.AUXCTLR** (and leave it untouched otherwise). After resetting the target chip, TRCAUXCTLR contains 0.

See also

- ETM
- ETM.state

ETM.BBC Branch address broadcast

Format: **ETM.BBC** [**ON** | **OFF**]

Enable or disable branch-broadcasting globally.

When branch-broadcasting is active, the ETM broadcasts the address information for all branches. This consumes more trace memory and trace port bandwidth. It is usually not required.

OFF The ETM broadcasts only the address information when the processor branches to a location that cannot be directly inferred from the source code (default).

ON The ETM broadcasts the address information for all branches.
This option has to be ON, if hardware based code coverage with ETMv1 is used.

See also

- ETM
- ETM.state

ETM.BBCExclude Exclude address ranges from branch-broadcasting

Format: **ETM.BBCExclude** *<access>* … *<address_range>* (ETM≥4.0)

<access>: **Execute** | **Read** | **Write** | **ReadWrite**

While command **ETM.BBC OFF** disables branch-broadcasting globally, this commands disables branch-broadcasting only for certain address ranges (while it is enabled elsewhere).

When branch-broadcasting is active, the ETM broadcasts the address information for all branches. This consumes more trace memory and trace port bandwidth. It is usually not required.

The commands **ETM.BBCInclude** and **ETM.BBCExclude** are mutually exclusive.

See also

- ETM
- ETM.state

ETM.BBCInclude Enable branch-broadcasting for dedicated address ranges

Format: **ETM.BBCInclude** *<access>* … *<address_range>* (ETM≥4.0)

<access>: **Execute** | **Read** | **Write** | **ReadWrite**

While command **ETM.BBC ON** enables branch-broadcasting globally, this commands enables branch-broadcasting only for certain address ranges (while it is disabled elsewhere).

When branch-broadcasting is active, the ETM broadcasts the address information for all branches. This consumes more trace memory and trace port bandwidth. It is usually not required.

The commands **ETM.BBCInclude** and **ETM.BBCExclude** are mutually exclusive.

See also

- ETM
- ETM.state

Format: **ETM.CLEAR**

Switches the ETM ON, clears the trace and clears all setting for the sequencer respectively clears all setting done by the command **ETM.Set**.

See also

■ ETM ■ ETM.Set ■ ETM.state

ETM.CLOCK Set core clock frequency for timing measurements

Format: **ETM.CLOCK** *<frequency>*
(alias for **<trace>.CLOCK**)

Tells the debugger the core clock frequency of the traced ARM core.

- If the timing information is based on core clock cycles (**ETM.TImeMode CycleAccurate**), this setting is used to calculate the elapsed time in seconds from the elapsed clock cycles.
- If the timing information is based on external timestamps or (**ETM.TImeMode External** or **ETM.TImeMode ExternalInterpolate**), this setting is used to calculate the elapsed clock cycles from the elapsed time in seconds.
- If the timing information is based on synchronous internal timestamps (**ETM.TImeMode SyncTimeStamp**), this setting is used to calculate the elapsed clock cycles from the elapsed time in seconds.
- If the timing information is based on asynchronous internal timestamps (**ETM.TImeMode AsyncTimeStamp**), this setting is used together with **ETM.TimeStampCLOCK** to calculate the elapsed clock cycles from the elapsed time in seconds.
- For timing modes which combine timestamps with cycle count information, this setting is not required.

See also

■ ETM ■ ETM.state

Format: **ETM.CORE** *<core_id>*

Selects the core to be traced when the ETM unit can be connected to multiple cores.

See also

- ETM
- ETM.state

ETM.CPRT Monitor coprocessor register transfers

Format: **ETM.CPRT** [**ON** | **OFF**] (ETM≤3.5)

Monitor Coprocessor Register Transfers are traced if **ETM.CPRT** is set to ON. Default is OFF.

See also

- ETM
- ETM.state

Format: **ETM.COND OFF | Loads | Stores | LoadsAndStores | ALL** (ETM≥4.0)

Configures the ETM if information about the execution of conditional non-branch instructions should be included in the trace stream. This gets implicitly enabled if a data trace for loads and/or stores is enabled.

The execution of a conditional branch instruction is always traced. This is a configuration option for ETMv4 on ARMv7-R, ARMv8-R, ARMv7-M and ARMv8-M architecture.

The command is also needed at the time of trace decoding if the Trace Configuration Register (TRCCONFIGR) of the ETMv4 cannot be read (e.g. trace post processing in the TRACE32 simulator).

OFF	Conditional instruction tracing is disabled.
Loads	Conditional load instruction are traced.
Stores	Conditional store instructions are traced.
LoadsAndStores	Conditional load and store instructions are traced.
ALL	All conditional instructions are traced.

See also

- ETM
- ETM.state

ETM.ContextID Select the width of the "ContextID" register

Format: **ETM.ContextID 8 | 16 | 32 | OFF**

Select the width of the Context ID register. By setting **ETM.ContextID** to any value (except **OFF**), the ETM will emit a trace message containing the value written to the Context ID register.

When tracing a CPU with a target operating system, the trace recording should include information about the active tasks and/or threads. This can be either achieved by using data-trace or by using **ETM.ContextID** (especially if data-trace is not available (e.g. Cortex-A)). If you are using **ETM.ContextID** you have to ensure that your target operating systems writes to the Context ID register whenever a context switch (task/thread switched) occurs.

See also

- ETM
- ETM.state

Format: **ETM.CycleAccurate** [**ON** | **OFF**]

Enables cycle accurate tracing if ON. Default is OFF.

Cycle accurate tracing can be used to observe the exact number of cycles that a particular code sequence takes to execute. When **ETM.CycleAccurate** is OFF then the timestamp information from the TRACE32 hardware will be used. These timestamps are generated when the tracepaket is recorded in the tracebuffer. As the packets may be buffered in FIFOs on the chip the packets may get a variable delay between the generation from the ETM and the time the packets is seen on the external trace port. This results in small errors in the timestamps. For most measurements these errors can be discarded when the time taken from the trace is large compared with the error (e.g. taking the time of a larger function). However the errors will become relevant when looking for the time of very small functions or even single instructions. This is the use case for cycle accurate tracing. Cycle accurate tracing must also be used to get any time information from onchip trace buffers. Cycle accurate tracing has two disadvantages: it requires more trace port bandwidth and it takes more time to display the trace. Timestamps are generated based on the CPU clock if the CPU clock is specified with the command **Trace.CLOCK** *<cpu_clock>*. It is recommended to reduce the data trace information if cycle accurate tracing is used, because cycle accurate tracing generates extra load on the trace port (not for ETMv1).

Here is a summary of pros and cons for cycle accurate tracing:

+ exact timestamps for small code pieces (or even single instructions)
+ timestamps for onchip trace buffers
+ trace can show number of clocks even when core clock changes dynamically
+ exact time correlation with other cores (when global timestamps are available)

- requires more traceport bandwidth (about four times more) (not ETMv1)
- reduced tracing time (more trace packets generated)
- longer trace processing time (needs to process whole trace to get timestamp of last record) (not ETMv1)
- no time correlation with other cores (except when global timestamps are available)
- no time correlation with other trace hardware

See also

■ ETM ■ ETM.DataTrace ■ ETM.state ■ <trace>.CLOCK

ETM.CycleCountThreshold Set granularity for cycle accurate timing info

Format: **ETM.CycleCountThreshold** *<threshold>* (ETM≥4.0)

Configure the granularity of cycle accurate time information for the ETMv4. (See also command **ETM.TImeMode** for details about cycle accurate timing information.)

By default the **ETM.CycleCountThreshold** is set to a low number which ensures that cycle information is provided with (almost) every program trace cycle.

<threshold> The threshold value sets the minimum number of core clock cycles that need to elapse before a cycle information is emitted on the trace port. Larger numbers reduce the required bandwidth and required trace memory, but make the time information less accurate.

See also

- ETM
- ETM.state

ETM.CycleCountTickEnable ETMv4 cycle counter overflows

[build 149103 - DVD 09/2022]

Format: **ETM.CycleCountTickEnable** [**ON** | **OFF**]

Enables workaround for cycle counter overflows in ETMv4.

See also

- ETM
- ETM.state

ETM.CycleCountTickRate ETMv4 cycle counter rate

[build 149103 - DVD 09/2022]

Format: **ETM.CycleCountTickRate** *<rate>*

Default: 100.

Defines the rate at which ticks are generated when **ETM.CycleCountTickEnable** and **ETM.CycleAccurate** are enabled.

The ticks are a workaround for limitations of the ETMv4 cycle counter trace protocol.

The number of clocks reported for one trace message (one block of instructions between branches) is limited. If this limit is reached the ETMv4 will just report that an “unknown” number of clocks has elapsed. The trace will display an appropriate flow error in this case and the execution time will be taken as zero.

With enabled ticks the trace decoder has a way to determine the time from the number of ticks seen in the trace. The accuracy of this coarse time stamping depends on the tick rate.

Ticks are traced as ETM events. This means that the trace will generate a continuous stream of messages at the tick rate.

See also

- ETM
- ETM.state

ETM.DataSuppress — Suppress data flow to prevent FIFO overflow

Format: **ETM.DataSuppress** [**ON** | **OFF**]

Allow the ETM to suppress the data flow information if a FIFO overflow is likely to happen.

Example:

```
ETM.FifoLevel 16.                    ; Select a FifoLevel

ETM.DataSuppress ON
```

See also

- ETM
- ETM.state

Format: **ETM.DataTrace** *<def>*

<def>:
ON
Read
Write
Address
ReadAddress
WriteAddress
Data
ReadData
WriteData
Only (ETM≥3.0)
OnlyAddress (ETM≥3.0)
Only Data (ETM≥3.0)
OFF

Configures which elements are included in the data trace:

ETM.DataTrace	Trace of **Program Flow**	Trace **Data Values** of **Read** Accesses	Trace **Data Values** of **Write** Accesses	Trace **Addresses** of **Read** Accesses	Trace **Addresses** of **Write** Accesses
ON	■	■	■	■	■
Read	■	■		■	
Write	■		■		■
Address	■			■	■
ReadAddress	■			■	
WriteAddress	■				■
Data	■	■	■		
ReadData	■	■			
WriteData	■		■		
Only		■	■	■	■
OnlyAddress				■	■
OnlyData		■	■		
OFF	■				

See also

■ ETM ■ ETM.CycleAccurate ■ ETM.Set ■ ETM.state
■ ETM.TimeStampsTrace

▲ 'Release Information' in 'Legacy Release History'

Format: **ETM.DataTracePrestore** [**ON** | **OFF**] (ETM≤3.5)

This command is an alias for the deprecated command **Analyzer.Mode.Prestore** - but supports also onchip trace.

ETM.DataTracePrestore configures the ETM to generate an extra program trace cycle (ptrace) for every traced data cycle in **Trace.List**. Thus, for every traced data access you get also the address of the command which caused the data access. This is especially useful if you are not tracing the complete program flow e.g. by using command:

```
Break.Set /TraceEnable <data_address>
```

- **ETM.DataTracePrestore** is mainly related to the ETMv3 (e.g. ARM11, Cortex-R4/R5, Cortex-A5/A7/A8), where this command controls if additional trace packets are generated or not.
- The ETMv1 (e.g. ARM9) always reports program trace cycles for all data accesses. Thus, the command **ETM.DataTracePrestore** just enables or disables the display of the program trace cycle associated with a data cycle, if you have disabled the complete program trace.
- For ETMv4 (e.g. Cortex-R7) this command has no effect.

See also

■ ETM ■ ETM.state

Format:	**ETM.DataViewExclude** *<access>* … *<address_range>* | *<address>* (ETM≤3.5) **ETM.DataViewExclude** *<access>* … *<address_range>* (ETM≥4.0)
<access>:	**Execute** | **Read** | **Write** | **ReadWrite** (all) **Fetch** | **ExecutePass** | **ExecuteFail** | **MAP** (ETM≤3.5)

This command can be used:

- to exclude the specified *<address_range>* from broadcasting of data accesses
- to exclude a small *<address_range>* or a single *<address>* from an include range

Example:

```
; broadcast address and data
ETM.DataTrace Both

; exclude the accesses to the address range 0x6000++0xfff from
; broadcasting
ETM.DataViewExclude ReadWrite 0x6000++0xfff
```

See also

- ETM
- ETM.Set
- ETM.state

Format: **ETM.DataViewInclude** *<access>* … *<address_range>* | *<address>* (ETM≤3.5)
ETM.DataViewInclude *<access>* … *<address_range>* (ETM≥4.0)

<access>: **Execute** | **Read** | **Write** | **ReadWrite**
Fetch | **ExecutePass** | **ExecuteFail** | **MAP** (ETM≤3.5)

Defines the *<address_range>* | *<address>* for which data accesses are broadcast.

Example:

```
; broadcast address and data for data accesses
ETM.DataTrace Both

; restrict the broadcasting to accesses to the address range
; 0x6000++0xfff
ETM.DataViewInclude Access 0x6000++0xfff

; broadcast address and data for data accesses
ETM.DataTrace Both

; restrict the broadcasting to write accesses to the
; variable flags
ETM.DataViewInclude Write V.RANGE(flags)

; broadcast address and data for data accesses
ETM.DataTrace Both

; restrict the broadcasting to write accesses to the
; memory selected by the memory map decoder 3
ETM.DataViewInclude Write MAP 3.
```

See also

- ETM
- ETM.Set
- ETM.state

Format: **ETM.DBGRQ** [**ON** | **OFF**] (ETM≤3.5 or PTM)

Set debug request control. When set to ON and a trigger occurs the ARM processor can be forced to enter the debug state.

Use this to make a trigger stop the tracing plus the program execution.

See also

■ ETM ■ ETM.state

ETM.FifoFullExclude No activation of FIFOFULL in range

Format: **ETM.FifoFullExclude** [*<access>* …] *<address_range>*

<access>: **Execute** | **Read** | **Write** | **ReadWrite** (all)
Fetch | **ExecutePass** | **ExecuteFail** | **MAP** (ETM≤3.5 or PTM)

Defines the *<address_range>* where FIFOFULL will not be generated in the case of a FIFO overflow, so that the processor is not stalled in critical code.

The commands **ETM.FifoFullInclude** or **ETM.FifoFullExclude** are mutually exclusive.

Example:

```
; do not generate FIFOFULL in the defined address range
ETM.FifoFullExclude 0x1f20--0x1ff7

; do not generate FIFOFULL for the memory selected by memory map decoder
; 3
ETM.FifoFullExclude MAP 3.
```

See also

■ ETM ■ ETM.Set ■ ETM.state

Format: **ETM.FifoFullExclude** [*<access>* *<address_value>* …]

Defines the *<address_range>* where FIFOFULL is generated in the case of a FIFO overflow.

The commands **ETM.FifoFullInclude** or **ETM.FifoFullExclude** are mutually exclusive.

Example:

```
; generate FIFOFULL in the defined address range
ETM.FifoFullInclude 0x10000++0xffff

; generate FIFOFULL for the memory selected by memory map decoder 3
ETM.FifoFullInclude MAP 3.
```

See also

- ETM
- ETM.Set
- ETM.state

ETM.FifoLevel Define FIFO level for FIFOFULL

Format: **ETM.FifoLevel** *<value>*

<value>: **1** | **2** … *n*

Defines the FIFO level. If the FIFO has less then *<value>* number of bytes of space available FIFOFULL is generated if enabled by **ETM.FifoFullInclude** or **ETM.FifoFullExclude**.

See also

- ETM
- ETM.Set
- ETM.state

ETM.FunnelHoldTime — Define minimum funnel hold time

Format: **ETM.FunnelHoldTime** *<value>*

<value>: **1** | **2** … **7**

Define the minimum Hold Time of all Coresight Funnels that are involved to the ETM trace data.

The formatting scheme of the trace data stream can easily become inefficient if fast switching occurs so where possible this should be minimized. If a source has nothing to transmit then another source will be selected irrespective of the minimum no. of cycles.

See also

- ETM
- ETM.state

ETM.HalfRate — Halfrate mode

Format: **ETM.HalfRate** [**ON** | **OFF**]

ETM.HalfRate has to be ON if the ETM works in half rate mode, which means that trace data should be captured on both rising and falling edge of the trace clock (aka. "Double Data Rate").

This configuration option is only available for ETMv1. All further ETM versions (including PTM/PFT) operate always in HalfRate mode.

See also

- ETM
- ETM.state

ETM.LPOVERRIDE — Prohibit lower power mode

Format: **ETM.LPOVERRIDE** [**ON** | **OFF**]

ETM.LPOVERRIDE ON configures the ETMv4 not to enter low-power state when the ARM cores enters low-power state.

See also

- ETM
- ETM.state

Format: **ETM.INSTP0 Branches | Loads | Stores | LoadsAndStores**

Configures the ETMv4 if load and store instructions are included in the program flow trace. This gets implicitly enabled if a data trace for loads and/or stores is enabled.

Branches are always traced. This is a configuration option for ETMv4 on ARMv7-R, ARMv8-R, ARMv7-M and ARMv8-M architecture.

The command is also needed at the time of trace decoding if the Trace Configuration Register (TRCCONFIGR) of the ETMv4 cannot be read (e.g. trace post processing in the TRACE32 simulator).

Branches	Do not trace load and store instructions as P0 instructions.
Loads	Trace load instructions as P0 instructions.
Stores	Trace store instructions as P0 instructions.
LoadsAndStores	Trace load and store instructions as P0 instructions.

See also

- ETM
- ETM.state

ETM.MapDecode Memory map decode control

Format: **ETM.MapDecode** *<code>*

Sets the memory map decode control register to *<code>*.

See also

- ETM
- ETM.state

ETM.NoOverflow Enable ETMv4 feature to prevent target FiFo overflows

Format: **ETM.NoOverflow** [**ON** | **OFF**]

Enables (or disables) a mechanism of the ETMv4 to prevent overflows (if supported). Similar to **ETM.STALL**. Enabling the feature might have a significant performance impact.

See ARM ETMv4 architecture specification for details.

See also

- ETM
- ETM.state

ETM.ON Switch ETM on

Format: **ETM.ON**

Enables ETM functionality.

See also

- ETM
- ETM.state

ETM.OFF Switch ETM off

Format: **ETM.OFF**

Disables ETM functionality.

See also

- ETM
- ETM.state

Format: **ETM.PortDisable** [**ON** | **OFF**]

Default: OFF

Setting **ETM.PortDisable** to **ON** forces the ETMEN signal to 0 in the ETM main control register. This usually disables the trace output from the ETM. Thus, you should normally not set **ETM.PortDisable** to **ON**.

On an ARM chip the ETMEN signal can be used to enable the trace port pins, which are shared with other functions like e.g. GPIO. On some chips driving the ETMEN signal by the ETM has some fatal consequences. In this rare case you can force ETMEN signal to 0 with this command.

This setting is mainly for ARM cores without CoreSight debug & trace infrastructure (ARM9 / ARM11). It is considered by the ETMv1, ETMv3 and PTM, but has no effect for ETMv4.

ETM.ON **ETM.OFF**	**ETM.Trace**	**ETM.PortDisable**	**ETM.PortDisableOnchip**	**ETM.PortRoute**	**ETMEN** (port enable)
ETM.OFF	x	x	x	x	**0**
ETM.ON	OFF	x	x	x	**0**
ETM.ON	ON	ON	x	x	**0**
ETM.ON	ON	OFF	ON	Onchip	**0**
ETM.ON	ON	OFF	ON	(C)Analyzer	**1**
ETM.ON	ON	OFF	OFF	x	**1**

See also the ARM Embedded Trace Macrocell Architecture Specification for ETMv1.0 to ETMv3.5.

See also

- ETM
- ETM.state

Format: **ETM.PortDisableOnchip** [**ON** | **OFF**]

Default: OFF

Setting **ETM.PortDisable** to **ON** forces the ETMEN signal to 0 in the ETM main control register *when using onchip trace (ETB)*. This usually disables the trace output from the ETM. Most (older) ETMs require the trace port to be enabled even when tracing just to ETB. Thus, you should normally not set **ETM.PortDisable** to **ON**.

On an ARM chip the ETMEN signal can be used to enable the trace port pins, which are shared with other functions like e.g. GPIO. On some chips you have to set **ETM.PortDisable** to **ON** to use the onchip trace (ETB) while using the physical pins of the trace-port for other purposes. However most ETMs require the trace port to be enabled (according to the ETM main control register) even when just using onchip trace.

This setting is mainly for ARM cores without CoreSight debug & trace infrastructure (ARM9 / ARM11). It is considered by the ETMv1, ETMv3 and PTM, but has no effect for ETMv4.

ETM.ON ETM.OFF	ETM.Trace	ETM.PortDisable	ETM.PortDisableOnchip	ETM.PortRoute	ETMEN
ETM.OFF	x	x	x	x	**0**
ETM.ON	OFF	x	x	x	**0**
ETM.ON	ON	ON	x	x	**0**
ETM.ON	ON	OFF	ON	Onchip	**0**
ETM.ON	ON	OFF	ON	(C)Analyzer	**1**
ETM.ON	ON	OFF	OFF	x	**1**

See also

- ETM
- ETM.state

Format:	**ETM.PortMode Normal** | **Muxed** | **Demuxed** | **Demuxed2** (ETMv1.x) **ETM.PortMode Dynamic** | **Custom** | **2/1** | **1/1** | **1/2** | **1/3** | **1/4** (ETMv3.x, ARM11) **ETM.PortMode Bypass** | **Wrapped** | **Continuous** (CoreSight) (deprecated) **ETM.PortMode 1500Mbps** | **2000Mbps** | **2500Mbps** (serial ETM) (deprecated) **ETM.PortMode 3000Mbps** | **3125Mbps** | **4250Mbps** (serial ETM) (deprecated) **ETM.PortMode 5000Mbps** | **6000Mbps** | **6250Mbps** (serial ETM) (deprecated)

Select the ETM mode or port bandwidth.

CoreSight (deprecated)

Use **TPIU.PortMode** instead.

The TPIU/ETB merges the trace information generated by the various trace sources within the multicore chip to a single trace data stream. A trace source ID (e.g **ETM.TraceID**) allows to maintain the assignment between trace information and its generating trace source. The task of the Formatter within the TPIU/ETB is to embed the trace source ID within the trace information to create this single trace stream.

ETM.PortMode specifies the Formatter operation mode.

Bypass	There is only one trace source, so no trace source IDs is needed. In this operation mode the trace port interface needs to provide the TRACECTL signal.
Wrapped	The Formatter embeds the trace source IDs. The TRACECTL signal is used to indicate valid trace information.
Continuous	The Formatter embeds the trace source IDs. Idles are generated to indicate invalid trace information. The TRACE32 preprocessor filters these idles in order to record only valid trace information into the trace memory.

See also

- ETM
- ETM.state

Format: **ETM.PortRoute** [**AUTO** | **Analyzer** | **CAnalyzer** | **Onchip**]

Default: AUTO

Prepares the selected trace hardware for ETM trace capture.

AUTO	Automatic detection
Analyzer	PowerTrace (via TPIU)
CAnalyzer	Compact-Analyzer: CombiProbe or µTrace (MicroTrace)
Onchip	Onchip trace buffer (ETB, ETF or ETR)

See also

■ ETM ■ ETM.state

ETM.PortSize Define trace port width

Format: **ETM.PortSize 4** | **8** | **16** (ETMv1.x)
ETM.PortSize 1 | **2** | **4** | **8** | **16** | **24** | **32** (ETMv3.x and higher)
ETM.PortSize 1lane | **2lane** | **3lane** | **4lane** (serial ETM)

Defines the width of the ETM trace port.

See also

■ ETM ■ ETM.state

▲ 'Release Information' in 'Legacy Release History'

ETM.PowerUpRequest Power-up request for the ETM by the debugger

Format: **ETM.PowerUpRequest** [**ON** | **OFF**]

Default: ON

When **ETM.PowerUpRequest** is set to **ON**, the debugger sets the power-up request bit in the ETM configuration register TRCPDCR, when the ETM is used, This enables the power domain of the onchip trace unit.

See also

■ ETM ■ ETM.state

ETM.PseudoDataTrace Enable pseudo data trace detection

Format: **ETM.PseudoDataTrace** [**ON** | **OFF**]

Allows to generate "artificial" data cycles in the trace based on a program trace. This can be useful for ETMs/PTMs that don't implement data value tracing (e.g. Cortex-A8, Cortex-A9). It requires special code in the target that executes a sequence of branch instructions to transmit the data information.

An example for the special code can be found in ~~/demo/arm/etc/tracedata (where ~~ stands for the TRACE32 installation directory).

OFF Detection of data cycles disabled (default).

ON Detection of data cycles enabled.

See also

■ ETM ■ ETM.state

▲ 'Release Information' in 'Legacy Release History'

ETM.QE Enable Q elements

Format: **ETM.QE ON** | **Counted** | **OFF**

Controls if the ETMv4 trace stream may include Q elements. This is a configuration option for the ETMv4 on an ARMv8-A CPU. See the ARM ETMv4 architecture specification for details.

The command is also needed at the time of trace decoding if the Trace Configuration Register (TRCCONFIGR) of the ETMv4 can not be read (e.g. trace post processing in the TRACE32 simulator).

OFF	Q elements are disabled.
Counted	Q elements with instruction counts are enabled. Q elements without instruction counts are disabled.
ON	Q elements with and without instruction counts are enabled.

See also

- ETM
- ETM.state

ETM.QTraceExclude Prohibit Q trace elements in given address range

Format: **ETM.QTraceExclude** *<access> <address_range>* (ETM≥4.0)

<access>: **Execute** | **Read** | **Write** | **ReadWrite**

While command **ETM.QE ON** allows the usage of Q elements in the trace stream globally, this commands enables Q elements only for certain address ranges (while they are forbidden elsewhere).

The commands **ETM.QTraceInclude** and **ETM.QTraceExclude** are mutually exclusive.

See also

- ETM
- ETM.state

ETM.QTraceInclude Allow Q trace elements in given address range

Format: **ETM.QTraceInclude** *<access> <address_range>* (ETM≥4.0)

<access>: **Execute** | **Read** | **Write** | **ReadWrite**

While command **ETM.QE ON** allows the usage of Q elements in the trace stream globally, this commands enables Q elements only for certain address ranges (while they are forbidden elsewhere).

The commands **ETM.QTraceInclude** and **ETM.QTraceExclude** are mutually exclusive.

See also

- ETM
- ETM.state

Format: **ETM.RefClock** [**ON** | **OFF**]

OFF (default) Disable reference clock.

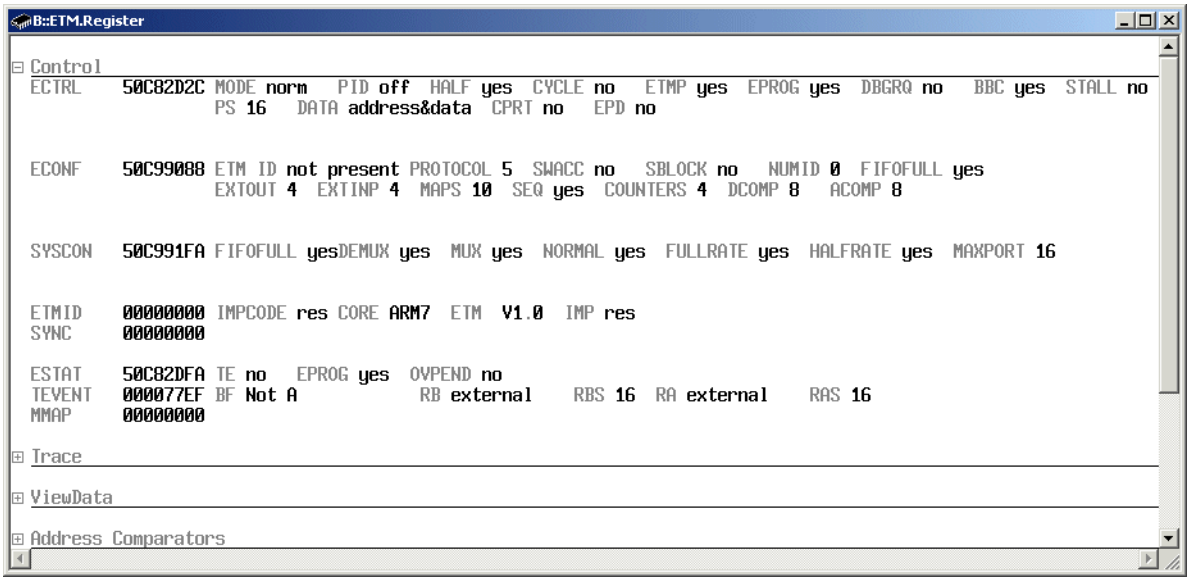
ON The STP preprocessor broadcasts a high frequency reference clock signal to the target. The frequency is half the bitrate (**ETM.PortMode**) in MHz.
This option is required to support configuration C as specified in the ARM HSSTP architecture specification and should be disabled in other cases.

See also

- ETM
- ETM.state

Format:	**ETM.Register** [*<file>* **/***<option>*]
<option>:	**SpotLight** | **DualPort** | **Track** | **AlternatingBackGround** **CORE** *<core_number>*

Displays the ETM registers. The contents will vary with the ETM version.



<option>	For a description of the options, see **PER.view**.

Examples:

```
ETM.Register , /SpotLight

ETM.Register , /DualPort

ETM.Register , /CORE 1.
```

See also

- ETM
- ETM.state

▲ 'Release Information' in 'Legacy Release History'

Format: **ETM.RESet**

Reset of setting of the **ETM.state** window to default.

See also

- ETM
- ETM.state

ETM.ReserveContextID Reserve special values used with context ID

Format: **ETM.ReserveContextID** *<range>*

Reserves a range of special values used at the ContextID register for special messages. These special values are not interpreted for task switch or memory space switch detection.

In the trace display the cycle type `task` is assigned to Linux task switches, while the cycle type `info` is assigned to the special values.

See also

- ETM
- ETM.state

Format: **ETM.ReturnStack** [**ON** | **OFF**]

OFF Regular tracing of return instructions. Each return instruction generates indirect branch packets. (default)

ON Return instructions that hit the return stack may be traced as direct branches (PTM and ETMv4 only). This reduces the required trace port bandwidth and can reduce the number of trace port FIFO overflows.

See also

■ ETM ■ ETM.state

▲ 'Release Information' in 'Legacy Release History'

Format:
ETM.Set *<complex_action>* [**NOT**] *<trg.src.A>* [**AND** | **OR** [**NOT**] *<trg.src.B>*]
ETM.Set *<simple_action>* *<addr.comp.1>* [*<addr.comp.2>*] … [*<addr.comp.3>*]
ETM.Set *<conf.trig.src>*

<complex_action>: **Complex Trigger Actions**
Trigger
TraceEnable
ViewData
External *<1..4>*
CountReload *<1..4>* | **CountEnable** *<1..4>*
TimeStamp (ETM≥3.5 or PTM)
SEQ1TO2 | **SEQ1TO3** | **SEQ2TO1** | **SEQ2TO3** | **SEQ3TO1** | **SEQ3TO2**
SEQTO1 (ETM≥4.0)
SingleShot (ETM≥4.0)

<simple_action>: **Simple Trigger Actions**
TraceON | **TraceOFF** (1.2≤ETM or PTM)
TraceAddressInclude | **TraceAddressExclude** (1.2≤ETM≤3.5 or PTM)
TraceRangeInclude | **TraceRangeExclude**
TraceMapInclude | **TraceMapExclude** (ETM≤3.5 or PTM)
ViewDataIAddressInclude | **ViewDataIAddressExclude**
ViewDataIRangeInclude | **ViewDataIRangeExclude**
ViewDataIMapInclude | **ViewDataIMapExclude** (ETM≤3.5)
FifoFullAddressInclude | **FifoFullAddressExclude** (ETM≤3.5 or PTM)
FifoFullRangeInclude | **FifoFullRangeExclude** (ETM≤3.5 or PTM)
BBCInclude | **BBCExclude** (ETM≥4.0)

<conf.trig.src>: **Configuration of Trigger Resources**
Address *<id: 1…16.>* *<access_type>* *<marker>*|*<address_value>*
Range *<id: 1…8.>* *<access_type>* *<marker>*|*<address_range>*
Data *<id: 1…16.>* *<data_value>*
ContextID *<id: 1…3.>* *<comp._value>* (ETM≥2.0 or PTM)
Count *<id: 1…4>* *<count_value>*
ExtendedExternal *<id: 1…4>* *<input_selector>* (ETM≥3.1 or PTM)

<trg.src.A>, *<trg.src.b>*: **Trigger Resources**	**Address** *<id: 1…16.>*	
	Range *<id: 1…8>*	
	ContextID *<id: 1…3>*	(ETM≥2.0 or PTM)
	VMID	(ETM≥3.5 or PTM)
	Count *<id: 1…4>*	
	Seq *<id: 1…3>*	
	Instrumentation *<id: 1…4>*	(ETM≥3.3 or PTM)
	External *<id: 1…4>*	
	ExtendedExternal *<id: 1…4>*	(ETM≥3.1 or PTM)
	EmbeddedICE *<id: 1…8>*	(ETM≤3.5 or PTM)
	MAP *<id: 1…16.>*	(ETM≤3.5 or PTM)
	TraceON	(ETM≥2.0 or PTM)
	NONSECURE	
	TraceProhibited	
	True	
	PROCESSOR *<id: 1…16>*	(ETM≥4.0)
	SingleShot *<id: 1…8>*	(ETM≥4.0)
<access_type>:	**Execute** | **Read** | **Write** | **ReadWrite**	
	Fetch | **ExecutePass** | **ExecuteFail**	(ETM≤3.5)
<marker>:	**Alpha** | **Beta** | **Charly** | **Delta** | **Echo**	
<count_value>:	*<decimal_value>* | *<hex_value>* | *<binary_value>*	
<data_value>, *<comp._value>*:	*<decimal_value>* | *<hex_value>* | *<binary_value>* | *<bitmask>*	
<id>, *<addr.comp.>*:	1,2, … , *n*	(with n≤16.)

ETM.Set allows a precise controlling and programming of the ETM event resources and the actions caused by these triggers.

You cannot use **ETM.Set** for Cortex-M.

In general the ETM allows to trigger several actions (like toggling an external pin) based on the occurrence of some events (e.g. a certain value was read by the CPU from a special address) detected by an event resource (e.g. an address comparator).

There are basically the following components:

- **Resources** which can detect events. Some of them can be configured.
- **Actions** which can be triggered by the ETM.
- Registers which control which sources cause which action.

For detailed information of the available trigger resources of the ETM see the *"ARM Embedded Trace Macrocell™ Architecture Specification"* (IHI 0014Q) at **http://infocenter.arm.com**

Trigger Resource	Description	IDs	Configuration	ETM
Address or **Range**	Address comparators trigger on a CPU access to a certain address. For each comparator you can configure an *address*, a *data value* and the *access type* (read, write, read-write, fetch, execute, execute-pass, execute-fail). If you are using **Range,** two single address comparators are combined to an address range comparator,	1-16	ETM.Set Address ETM.Set Range ETM.Set Data	
ContextID	Context ID comparators	1-3	ETM.Set ContextID	≥ 2.0
MAP	Memory map decoders	1-16	*implementation specific*	
Count	Down counting counters. Trigger is active when counter value is zero.	1-4	ETM.Set Count	
Seq	Active when the "ETM three-state sequencer" is in the specified state.	1-3		
Instrumentation	Events controlled by software instructions	1-4		≥ 3.3
External	External inputs	1-4		
ExtendedExternal	Extended external inputs	1-4	ETM.Set ExtendedExternal	≥ 3.1
EmbeddedICE	EmbeddedICE™ module watchpoint comparators	1-8	TrOnchip Break.Set	
TraceON	Active when ETM trace is active	-		≥ 2.0
VMID	Virtual Machine ID comparator	-		≥ 3.5
NONSECURE	Active when CPU in non-secure state	-		
TraceProhibited	Active	-		
True	This resource is always active.	-		

Complex Trigger Actions

The complex trigger actions are based one or two **Trigger Resources**. You can combine two sources with an logical AND or and OR. Optional the output from each resource can be inverted.

All complex trigger actions are programmed the following, with *<trg.src.A>* and *<trg.src.>* indicating any **Trigger Resource** mentioned above (e.g. Address, ContextID, Seq...)

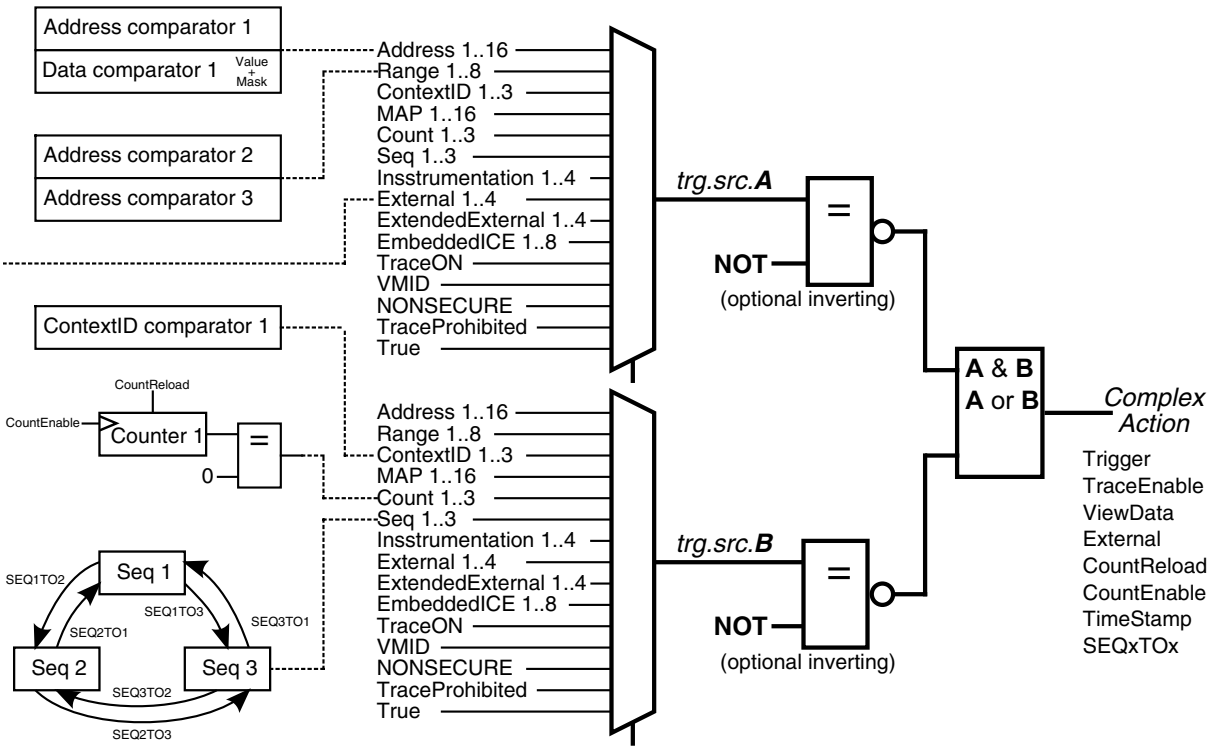
ETM.Set *<complex_action>* [**NOT**] *<trg.src.A>* [**AND** | **OR** [**NOT**] *<trg.src.B>*]

There are the following complex trigger actions:.

Trigger	Generate a trace trigger
TraceEnable	Enable the trace recording. ANDed by some simple trigger actions.
ViewData	Enable the data trace. ANDed by some simple trigger actions.
External *<1..4>*	Drive external output high. (Simple command would be **Break.Set** *<addr.>* **/BusTrigger**)
CountEnable *<1..4>*	Decrement 16-bit counter
CountReload *<1..4>*	Load 16-bit counter with value specified with **ETM.Set Count**
SEQ1TO2	Change sequencer state from 1 to 2
SEQ2TO1	Change sequencer state from 2 to 1
SEQ2TO3	Change sequencer state from 2 to 3
SEQ3TO1	Change sequencer state from 3 to 1
SEQ3TO2	Change sequencer state from 3 to 2
SEQ1TO3	Change sequencer state from 1 to 3

Counter and Sequencer are controlled by complex trigger actions but are used as trigger resources.

.



The simple trigger actions are based on several address ranges. You can combine all address comparators with a logical OR as a trigger source for a simple action. However the only **Trigger Resources** you can use for simple triggers are the **Address**, **Range** and **MAP** sources.

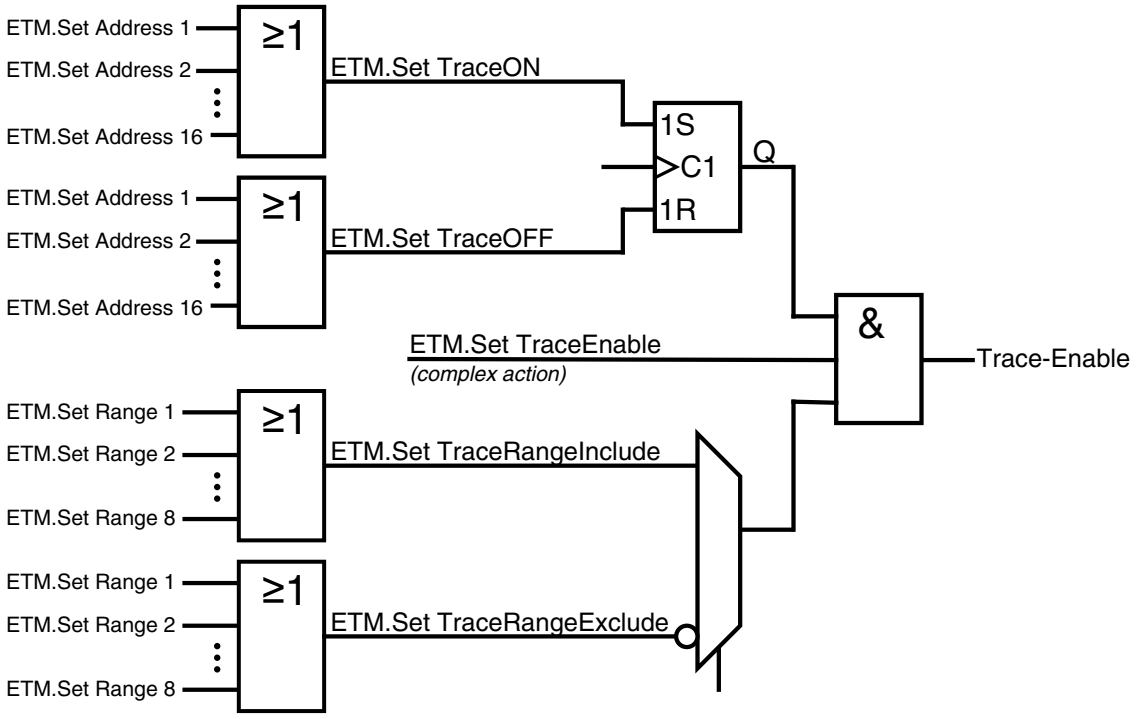
All simple trigger actions are programmed the following, with <id> specifying IDs of **Address**, **Range** or **MAP** sources depending on the used simple-action:

ETM.Set *<simple_action>* *<id.1>* [*<id.2>*] … [*<id.16>*]

The following simple actions **enable or disable the trace recording**. From the include/exclude triggers you can use either some trace-includes or some of the trace-excludes. But you can't mix them.

TraceON	Select single address(es) where tracing gets enabled.
TraceOFF	Select single address(es) where tracing gets disabled.
TraceAddressInclude	Select single address(es) where tracing is done
	Select address range(s) where tracing is done
TraceRangeInclude	Select memory map decoder regions where tracing is done
TraceMapInclude	
TraceAddressExclude	Select single address(es) where tracing is disabled
	Select address range(s) where tracing is disabled
TraceRangeExclude	Select memory map decoder regions where tracing is disabled
TraceMapExclude	

The trace is enabled when it should be enabled according to **TraceON/TraceOFF**, according to **TraceAddress/Range/MapInclude** (or **TraceAddress/Range/MapExclude**) and also according to the complex action trigger **ETM.Set TraceEnable**. If you don't configure one resource it fires an enable by default.



Configuring theses simple actions (to enable or disable the trace) via **ETM.Set** makes normally only sense, if you use them together with a complex trigger action **ETM.Set TraceEnable**. Otherwise it is recommended to use the commands **Break.Set /TraceON**, **Break.Set /TraceOFF** and **Break.Set /TraceEnable** (or **ETM.TraceExclude** / **ETM.TraceInclude**).

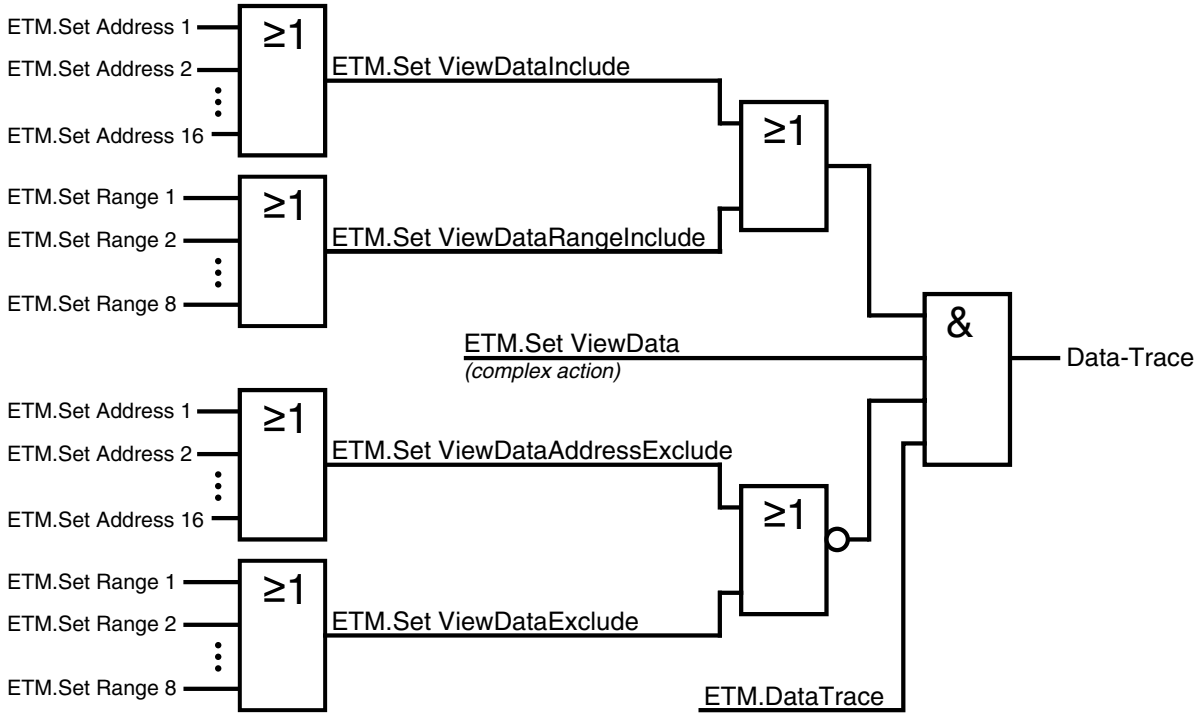
Example: Enable recording after entering the main routine.

```
ETM.CLEAR                          // Clear previous ETM.Set settings
ETM.Set Range 1 Execute _start     // Configure 1st single addr comparator
ETM.Set Range 2 Execute main       // Configure 2nd single addr comparator
ETM.Set TraceOFF 1                 // Disable trace on 1st address
ETM.Set TraceON  2                 // Enable trace on 2nd address
```

The following simple actions **enable or disable the data tracing**. You may use both data-include and data-exclude actions at the same time.

ViewDataInclude	Select single address(es) where data tracing is done
ViewDataRangeInclude	Select address range(s) where data tracing is done
ViewDataMapInclude	Select memory map decoder regions where data tracing is done

ViewDataExclude	Select single address(es) where data tracing is disabled.
ViewDataRangeExclude	Select address range(s) where data tracing is disabled.
ViewDataMapExclude	Select memory map decoder regions where data tracing is disabled



Configuring theses simple actions (to enable or disable the data trace) via **ETM.Set** makes normally only sense, if you use them together with a complex trigger action **ETM.Set ViewData**. Otherwise it is recommended to use the commands **Break.Set /TraceData** or **ETM.DataViewInclude** and **ETM.DataViewExclude**.

Setting **ETM.DataTrace** to **OFF** will globally disable the data trace ignoring any filter programming.

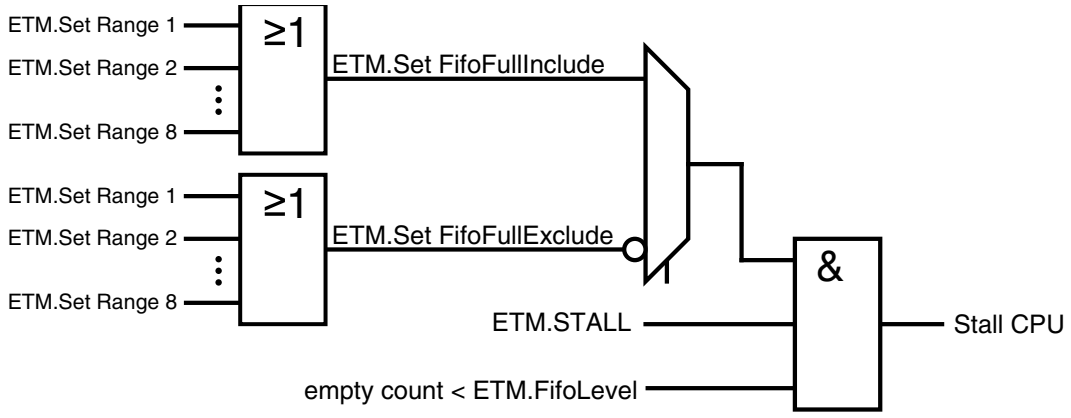
Example: Exclude call-stack and heap from data tracing.

```
ETM.CLEAR                                    // Clear previous ETM.Set settings
ETM.Set Range 1 ReadWrite 0x8f000--0x8ffff   // Configure 1st addr. range comparator
ETM.Set Range 2 ReadWrite Var.RANGE("heap")  // Configure 2nd addr. range comparator
ETM.Set ViewDataMapExclude 1 2               // Enable stalling for 1st and 2nd range
```

The following simple actions **enable or disable the stalling of the CPU** when the ETM output Fifo buffer inside your chip is almost full. The Fifo is considered as "almost full"when there is less empty space in the ETM output Fifo than configured with **ETM.FifoLevel**.

FifoFullInclude	Select address range(s) where CPU may be stalled when output Fifo is almost full.
FifoFullMapInclude	Select memory map decoder regions where CPU may be stalled when output Fifo is almost full.
FifoFullExclude	Select address range(s) where CPU may be stalled when output Fifo is almost full.
FifoFullMapExclude	Select memory map decoder regions where CPU may be stalled when output Fifo is almost full.

The actions have only an effect if stalling of the CPU is possible with your implementation of the ETM and stalling was globally enabled with **ETM.STALL** ist set to **ON**. Usually only ETMv1.x supports stalling. The actions have *no influence on the Data Suppression* of the ETMv3 (see **ETM.DataSuppress**)
You can use either some Fifo-include or some of the Fifo-exclude actions. But you can't mix them.



In most cases it is easier to use the commands **ETM.FifoFullInclude** and **ETM.FifoFullExclude** instead of **ETM.Set FifoFullInclude/Exclude** to allow or forbid stalling for some memory regions.

Example: Allow CPU stalls by the ETM globally but forbid them for time-critical functions like interrupt service routines.

```
ETM.CLEAR                                   // Clear previous ETM.Set settings
ETM.Set Range 1 Execute 0x1000--0x1fff      // Configure 1st addr. range comparator
ETM.Set Range 2 Execute Var.RANGE("isr2")   // Configure 2nd addr. range comparator
ETM.Set FifoFullExclude 1 2                 // Enable stalling for 1st and 2nd range
ETM.FifoLevel 16.                           // Set level at which Fifo is considered as
                                               almost full
ETM.STALL ON                                // Enable CPU stalling via the ETM
```

See also

- ETM.STALL
- ETM
- ETM.CLEAR
- ETM.DataTrace
- ETM.DataViewExclude
- ETM.DataViewInclude
- ETM.FifoFullExclude
- ETM.FifoFullInclude
- ETM.FifoLevel
- ETM.state
- ETM.TraceExclude
- ETM.TraceInclude

▲ 'Release Information' in 'Legacy Release History'

Format: **ETM.SmartTrace** [**ON** | **OFF**] (deprecated)

See also

- ETM
- ETM.state

ETM.STALL Stall processor to prevent FIFO overflow

Format: **ETM.STALL** [**ON** | **OFF**]

Allows the ETM to stall the processor to prevent a output FIFO overflow. If enabled, the trace will be no longer real time.

This feature is only supported by some CPU cores. If it is not supported it might be still possible to set this option, but then it won't have an effect.
In general CPU cores with ETMv1 and ETMv4 likely support **ETM.STALL**, while with ETMv3 and PTM/PFT it's likely not supported.

The following CPU cores are known for supporting **ETM.STALL**:
Cortex-R7, ARM926EJ-S, ARM946E-S, ARM966E-S.

The following CPU cores are known for **not supporting ETM.STALL**:
ARM7TDMI, ARM720T, ARM920T, ARM922T

Example:

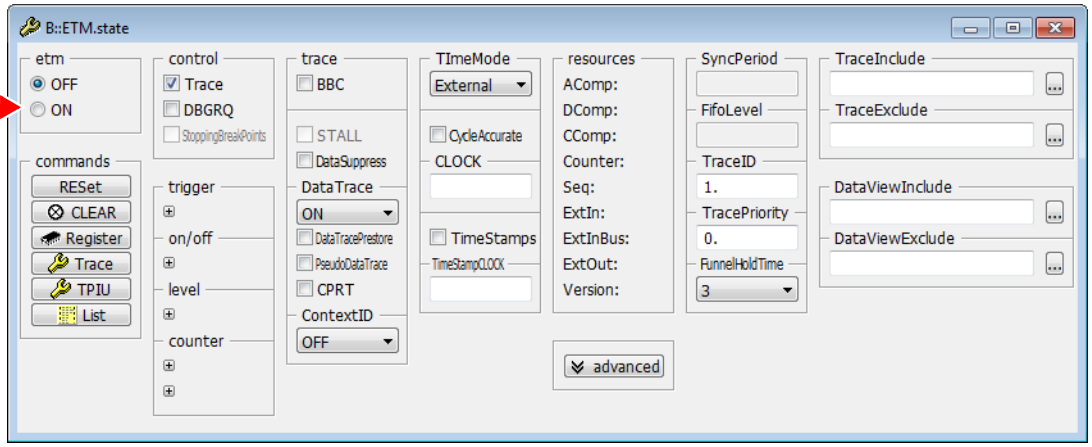
```
; Check if a FifoFull logic is available on your ETM
; FifoFull Yes

ETM.FifoLevel 16.                    ; Select a FifoLevel

ETM.STALL ON
```

See also

- ETM.Set
- ETM
- ETM.state

Format: **ETM.state**

Shows the ETM configuration window.



A For descriptions of the commands in the **ETM.state** window, please refer to the **ETM.*** commands in this chapter. **Example**: For information about **ON**, see **ETM.ON**.

Exceptions:

- The **Trace** button opens the main trace control window (**Trace.state**).
- The **TPIU** button opens the **TPIU.state** window.
- The **List** button opens the main trace list window (**Trace.List**).

See also

- ETM
- ETM.AUXCTLR
- ETM.CLEAR
- ETM.CORE
- ETM.CycleCountTickEnable
- ETM.DataTracePrestore
- ETM.FifoFullExclude
- ETM.HalfRate
- ETM.NoOverflow
- ETM.PortDisableOnchip
- ETM.PowerUpRequest
- ETM.QTraceInclude
- ETM.RESet
- ETM.STALL
- ETM.TimeStampCLOCK
- ETM.TraceCORE
- ETM.TraceID
- ETM.TracePriority
- ETM()
- ETM.PROTOCOL()
- ETM.AbsoluteTimestamp
- ETM.BBC
- ETM.CLOCK
- ETM.CPRT
- ETM.CycleCountTickRate
- ETM.DataViewExclude
- ETM.FifoFullInclude
- ETM.INSTP0
- ETM.OFF
- ETM.PortMode
- ETM.PseudoDataTrace
- ETM.RefClock
- ETM.ReturnStack
- ETM.StoppingBreakPoints
- ETM.TimeStamps
- ETM.TraceDataPriority
- ETM.TraceInclude
- ETM.TraceRESet
- ETM.ADDRCOMP()
- ETM.AddressMunging
- ETM.BBCExclude
- ETM.COND
- ETM.CycleAccurate
- ETM.DataSuppress
- ETM.DataViewInclude
- ETM.FifoLevel
- ETM.LPOVERRIDE
- ETM.ON
- ETM.PortRoute
- ETM.QE
- ETM.Register
- ETM.Set
- ETM.SyncPeriod
- ETM.TimeStampsTrace
- ETM.TraceERRor
- ETM.TraceNoPCREL
- ETM.TRCIDR
- ETM.ADDRCOMPTOTAL()
- ETM.ATBTrigger
- ETM.BBCInclude
- ETM.ContextID
- ETM.CycleCountThreshold
- ETM.DataTrace
- ETM.DBGRQ
- ETM.FunnelHoldTime
- ETM.MapDecode
- ETM.PortDisable
- ETM.PortSize
- ETM.QTraceExclude
- ETM.ReserveContextID
- ETM.SmartTrace
- ETM.TImeMode
- ETM.Trace
- ETM.TraceExclude
- ETM.TraceNoSPREL
- ETM.VMID
- ETM.FIFOFULL()

Format: **ETM.StoppingBreakPoints ON | OFF**
ETM.ReadWriteBreak ON | OFF (deprecated)

The command **ETM.StoppingBreakPoints ON** allows the debugger to use

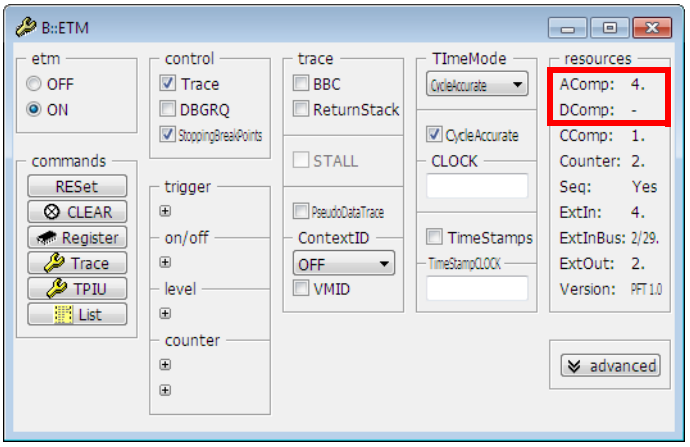
- ETM/PTM Address Comparators as program range breakpoints.
- ETM/PTM Address/Data Comparators as Read/Write breakpoints with data value.

Use cases

Cortex-A9

Debugger resources: 6 single address breakpoints for instructions

PTM (Version PFT 1.0) resources

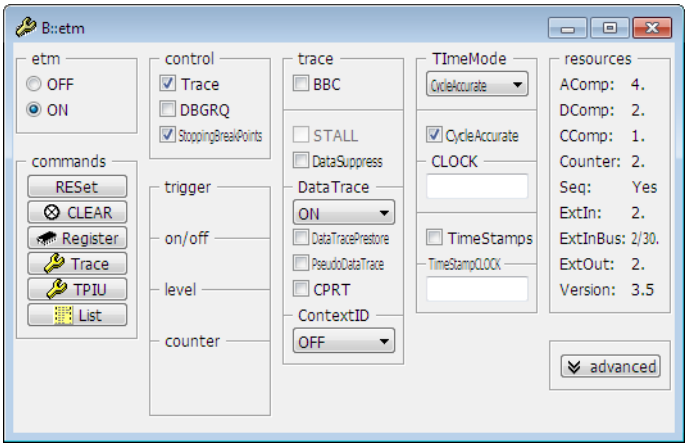


4. address comparator pairs on program addresses
- no address comparators for load/store addresses

```
ETM.StoppingBreakPoints ON              ; allows up to two address range
                                        ; breakpoints for program addresses

Var.Break.Set func10                    ; set address range breakpoint on
                                        ; the address range of function
                                        ; func10
```

Debugger resources: 3 single address breakpoints for instructions
4 bitmasks for load/store breakpoints, break before make
no data value possible

ETM (Version 3.5) resources



- 4 address comparators pairs
- 2 data value comparators

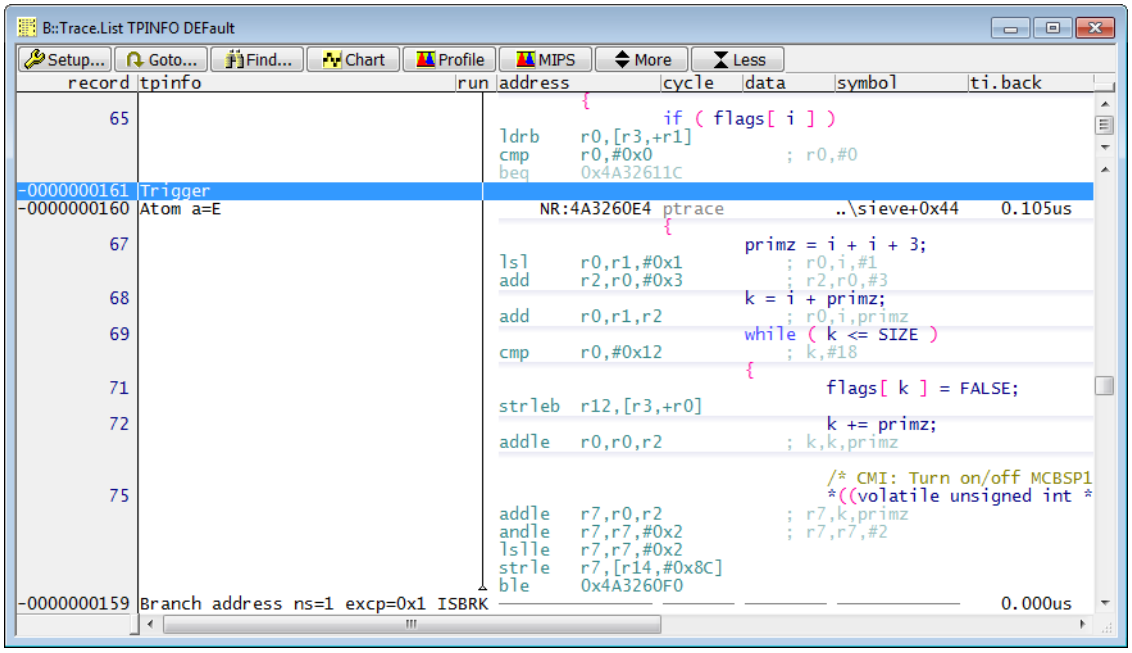
```
ETM.StoppingBreakPoints ON                ; allows up to two address
                                          ; range breakpoints for
                                          ; program addresses
                                          ; or
                                          ; up to two read/write
                                          ; breakpoints with data values

                                          ; address range breakpoints
                                          ; have priority

Var.Break.Set func10                      ; set address range breakpoint
                                          ; on the address range of
                                          ; function func10

Var.Break.Set flags /Write /DATA 0x0      ; set write breakpoint on
                                          ; address range of variable
                                          ; flags plus data value 0x0
```

Notes

1. Since the configuration of the ETM/PTM is done via the JTAG interface, no Preprocessor for ARM-ETM has to be connected, in order to use the ETM Address/Data Comparators, but the ETM has to be enabled by **ETM.ON**.

2. Please be aware that these so-called ETM breakpoints are asynchronous. The trigger is usually visible with the next branch and then it takes some time until the program stops.

```
Trace.List TPINFO DEFault                ; use the More button in the
                                         ; Trace.List window to see
                                         ; the Trigger info
```



3. The so-called ETM breakpoints are implemented via the ETM event logic setting (TEVENT) "A or B". That is the reason why there are only two ETM breakpoints possible for the ETMv1/ETMv3/PTM.

4. As long as **ETM.StoppingBreakPoints** is **ON**, it is not possible to use the option /TraceTrigger for a breakpoint.

See also

- ETM
- ETM.state

Format: **ETM.SyncPeriod** [*<period>*]

Synchronisation and timestamp packets are generated periodically. The default frequency is chosen depending on the trace method.

The command **ETM.SyncPeriod** allows to change this value. Please refer to your ETM/PTM manuals for details (ETMSYNCFR register).

See also

- ETM
- ETM.state

Format: **ETM.TImeMode** *<mode>*

<mode>: **OFF** | **External** | **ExternalInterpolated** |
SyncTimeStamps | **AsyncTimeStamps** |
CycleAccurate
CycleAccurate+External
CycleAccurate+ExternalTrack
CycleAccurate+SyncTimeStamps
CycleAccurate+AsyncTimeStamps

There are three methods to timestamp trace information generated by an ETM/PTM:

External (off-chip trace)

The trace information is timestamped with the global TRACE32 timestamp when it is stored into the trace memory of the TRACE32 trace tool.

- **Pros**

  No extra bandwidth is required for the timestamp.

  ETM/PTM trace information can be correlated with the trace information exported by other trace sources on the chip.

  ETM/PTM trace information can be correlated with the trace information recorded with other TRACE32 tools e.g. the logic analyzer PowerIntegrator.

- **Cons**

  Timestamp might be imprecise due to delays caused by the trace infrastructure of the chip and due to delays caused by the TRACE32 recording technology.

  On ARM Cortex-A/R CPUs external timestamps might be wrong when only parts of the program flow gets traced (e.g. by using trace filters TraceEnable, TraceON or TraceOFF), which is caused by the CoreSight trace infrastructure on the chip.

CycleAccurate (on-chip and off-chip trace)

The ETM/PTM protocol provides the number of core clock cycles that were needed by an instruction or an instruction range.

- **Pros**

  Provides accurate core clock cycle information.

  Accurate time information can be calculated, if the core clock was constant while recording the trace information.

- **Cons**

  Cycle accurate tracing requires up to 4 times more bandwidth.

  ETM/PTM trace information can not be correlated with any other trace information.

  Trace information has to be processed always from the start of the trace memory. "Tracking" indicates that the display of the trace information might need some time.

TimeStamps (on-chip and off-chip trace)

Timestamp Packets are exported by the ETM/PTM.

- **Pros**

  ETM/PTM trace information can be correlated with the trace information exported by other trace sources on the chip.

- **Cons**

  Timestamp Packets are not generated too often. Thus, only a rather inaccurate time information is possible.

  Timestamp Packets require some bandwidth.

TRACE32 PowerView provides various **TImeMode** based on these three methods:

OFF	Timestamping is disabled (default setting for on-chip trace).
External	The trace information is timestamped with the global TRACE32 timestamp when it is stored into the trace memory of the TRACE32 trace tool (off-chip trace only, default setting for off-chip trace).
ExternalInterpolated	Tries to improve the precision of the time information for wrapped multicore traces (**ETM.PortMode Wrapped/Continuous**) by linear interpolation of trace sections. Section borders are trace packets that have a precise timestamp (off-chip trace only).
CycleAccurate	Cycle accurate tracing (on-chip and off-chip trace). Related commands: **ETM.CycleAccurate ON** (automatically set when **CycleAccurate** is selected as **TImeMode**) **Trace.CLOCK** *<core_clock>* specifies the clock that is used to calculate time information out of the cycle count information. This requires that the core clock was constant while recording the trace information.
SyncTimeStamps	Time information is generated by linear interpolation of trace sections. Section borders are the Timestamp Packets. The clock of the global timestamp is the same as the core clock. This setting is recommended for onchip trace if a time correlation with trace information generated by other trace sources on the chip is required. Related commands: **ETM.TimeStamps ON** (automatically set when **SynchTimeStamps** is selected as **TImeMode**) It is recommended to inform TRACE32 PowerView about the core clock by using the command **Trace.CLOCK** *<core_clock>.* **ETM.SyncPeriod** *<frequency>* allows to increase the Timestamp Packet rate.

AsyncTimeStamps	Time information is generated by linear interpolation of trace sections. Section borders are the Timestamp Packets. The clock of the global timestamp is different from the core clock. This setting is recommended for onchip trace if a time correlation with trace information generated by other trace sources on the chip is required. Related commands: **ETM.TimeStamps ON** (automatically set when **AsynchTimeStamps** is selected as **TImeMode**) **ETM.TimeStampCLOCK** *<timestamp_clock>* specifies the frequency of the global timestamp. **ETM.SyncPeriod** *<frequency>* allows to increase the Timestamp Packet rate.
CycleAccu-rate+External	Combines external timestamps with cycle accurate information to provide very exact time information (off-chip trace only). Recommended if the core clock changes occasionally. It solves three issues of **CycleAccurate** tracing: - Core clock does not have to be constant while recording the trace information. - Correlation with other trace information exported by other trace sources on the chip is possible. - Correlation with the trace information recorded with other TRACE32 tools e.g. the logic analyzer PowerIntegrator is possible. Calculates time information by linear interpolation of trace sections. Section borders are trace packets that have a precise external timestamp. Related commands: **ETM.CycleAccurate ON** (automatically set when **CycleAccurate+External** is selected as **TImeMode**)
CycleAccurate+ ExternalTrack	Similar to **CycleAccurate+External** but smaller trace sections are used for the interpolation. Recommended if the core clock changes frequently. Related commands: **ETM.CycleAccurate ON** (automatically set when **CycleAccurate+ExternalTrac**k is selected as **TImeMode**)

CycleAccurate+ SyncTimeStamps	Combines Timestamp Packets with cycle accurate information to provide very exact time information (mainly for on-chip traces). Provides more precise time information then **SyncTimeStamps** mode and allows a time correlation with trace information generated by other trace sources on the chip. Related commands: **ETM.CycleAccurate ON** (automatically set when **CycleAccurate+SynchTimeStamps** is selected as **TImeMode**) **ETM.TimeStamps ON** (automatically set when **CycleAccurate+SynchTimeStamps** is selected as **TImeMode**) It is recommended to inform TRACE32 PowerView about the core clock by using the command **Trace.CLOCK** *<core_clock>*. **ETM.SyncPeriod** *<frequency>* allows to increase the Timestamp Packet rate.
CycleAccurate+ AsyncTimeStamps	Combines Timestamp Packets with cycle accurate information to provide very exact time information (mainly for on-chip traces). Provides more precise time information then **AsyncTimeStamps** mode and allows a time correlation with trace information generated by other trace sources on the chip. Related commands: **ETM.CycleAccurate ON** (automatically set when **CycleAccurate+AsyncTimeStamps** is selected as **TImeMode**) **ETM.TimeStamps ON** (automatically set when **CycleAccurate+AsyncTimeStamps** is selected as **TImeMode**) **ETM.TimeStampCLOCK** *<timestamp_clock>* specifies the frequency of the global timestamp. **ETM.SyncPeriod** *<frequency>* allows to increase the Timestamp Packet rate.

See also

- ETM
- ETM.state

ETM.TimeStampCLOCK Specify frequency of the global timestamp

Format: **ETM.TimeStampCLOCK** *<frequency>*

If the trace infrastructure of the SoC provides a global timestamp and if the clock of the global timestamp is different from the core clock, TRACE32 needs to know the frequency of this timestamp.

See also

- ETM
- ETM.state

ETM.TimeStamps Control for global timestamp packets

Format: **ETM.TimeStamps** [**ON** | **OFF**]

Requires that the frequency of the timestamp is specified if the clock of the global timestamp is different from the core clock (**ETM.TimeStampCLOCK**).

OFF (default)	ETM does not generate Timestamp Packets.
ON	ETM generates Timestamp Packets.

See also

- ETM
- ETM.state

Format: **ETM.TimeStampsTrace** [**ON** | **OFF**] (ETM≥4.0)

This command specifies how TRACE32 assigns the data address/data value information of the data trace stream to the appropriate load/store instruction.

OFF (default)	TRACE32 uses the P0, P1, P2 keys.
ON	TRACE32 uses the timestamp. This method requires a timestamp unit.

See also

■ ETM ■ ETM.DataTrace ■ ETM.state

▲ 'Release Information' in 'Legacy Release History'

ETM.Trace Control generation of trace information

Format: **ETM.Trace** [**ON** | **OFF**]

ON (default)	Trace information is generated and triggers/external signals are activated as programmed.
OFF	No trace information is generated. Only triggers/external signals are activated as programmed.

See also

■ ETM ■ ETM.state

Format: **ETM.TraceCORE** *<cores>*

Allows core specific default tracing.

See also

■ ETM ■ ETM.state

▲ 'Release Information' in 'Legacy Release History'

ETM.TraceDataPriority Define data trace priority

Format: **ETM.TraceDataPriority** *<priorities>* (ETM≥4.0)

Defines data trace priority.

See also

■ ETM ■ ETM.state

▲ 'Release Information' in 'Legacy Release History'

Format:	**ETM.TraceERRor** [**ON** | **OFF**] (ETM≥4.0)

When **ETM.TraceERRor** is set to **ON** the ETMv4 is forced to emit all system error exceptions. Consider this setting if you are using trace filters for a selective trace instead of a complete program flow trace..

ON	The ETMv4 always emits any system error exception, regardless of the configured trace-filter.
OFF (default)	The ETMv4 emits only a system error exception if it has also traced the exception or instruction immediately prior to the system error exception.

System error exception are:

- asynchronous Data Abort exceptions (only ARMv7 (Cortex-R7/R8, Cortex-M7/M33))
- SError interrupt (only ARMv8-A (Cortex-A3x/A5x/7x) and ARMv8-R (Cortex-R52)).
- MemManage exceptions (only Cortex-M)
- BusFault exceptions (only Cortex-M)
- HardFault exceptions (only Cortex-M)
- Lockup exceptions (only Cortex-M)
- SecureFault exceptions (only Cortex-M)

See "TRCVICTLR.TRCERR" in ETMv4 architecture specification for details.

See also

■ ETM ■ ETM.state

Format:	**ETM.TraceExclude** *<access>* … *<address_range>* | *<address>* (1.2≤ETM≤3.5) **ETM.TraceExclude** *<access>* … *<address_range>* (PTM or ETM≥4.0)
<access>:	**Execute** | **Read** | **Write** | **ReadWrite** (all) **Fetch** | **ExecutePass** | **ExecuteFail** | **MAP** (PTM or ETM≤3.5)

Configures the ETM to generate a program trace for all addresses except for the specified address range(s).

The commands **ETM.TraceInclude** or **ETM.TraceExclude** are mutually exclusive.

Example:

```
; exclude program flow in the specified address range from broadcasting
ETM.TraceExclude 0x1f20--0x1ff7

; exclude program flow for the memory selected by memory map decoder 3
; from broadcasting
ETM.TraceExclude MAP 3.
```

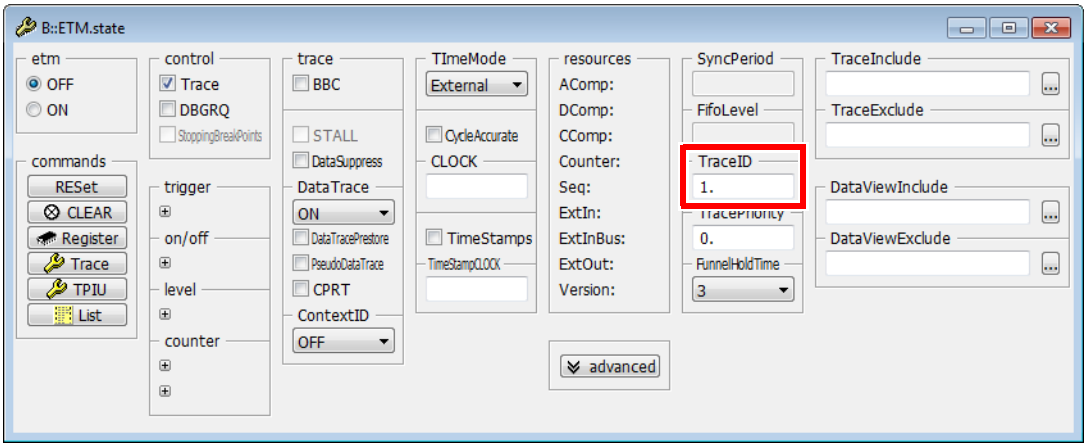
See also

- ETM
- ETM.Set
- ETM.state

Format: **ETM.TraceID** *<id>*

By default TRACE32 automatically assigns a trace source ID to all cores with a CoreSight ETM, the first ITM, and the first HTM. The command **ETM.TraceID** allows to assign an ID to a trace source overriding the defaults.

The current trace source ID is displayed in the **ETM.state** window.



See also

- ETM
- ETM.state

ETM.TraceInclude Restrict program trace to specified address range

Format: **ETM.TraceInclude** *<access>* … *<address_range>* | *<address>* (1.2≤ETM≤3.5)
ETM.TraceInclude *<access>* … *<address_range>* (PTM or ETM≥4.0)

<access>: **Execute** | **Read** | **Write** | **ReadWrite** (all)
Fetch | **ExecutePass** | **ExecuteFail** | **MAP** (PTM or ETM≤3.5)

Configures the ETM to generate a program trace only for the specified address range(s).

The commands **ETM.TraceInclude** or **ETM.TraceExclude** are mutually exclusive.

Example:

```
ETM.TraceInclude 0x1f20--0x1ff7       ; broadcast only program flow in
                                      ; the specified address range

ETM.TraceInclude MAP 3.               ; broadcast only program flow for
                                      ; memory selected by memory map
                                      ; decoder 3
```

See also

■ ETM ■ ETM.Set ■ ETM.state

ETM.TraceNoPCREL No data trace for accesses relative to program counter

Format: **ETM.TraceNoPCREL** [**ON** | **OFF**] (ETM≥4.0)

If enabled, the ETMv4 does not provide a data trace for read/write accesses relative to the program counter.

See also

■ ETM ■ ETM.state

ETM.TraceNoSPREL No data trace for accesses relative to stack pointer

Format: **ETM.TraceNoSPREL** [**ON** | **OFF**] (ETM≥4.0)

If enabled, the ETMv4 does not provide a data trace for read/write accesses relative to the stack pointer.

See also

■ ETM ■ ETM.state

Format: **ETM.TracePriority** *<priority>*

The CoreSight Trace Funnel combines 2 to 8 ATB input ports to a single ATB output. An arbiter determines the priority of the ATB input port. Port 0 has the highest priority (0) and port 7 the lowest priority (7) by default.

The command **ETM.TracePriority** allows to change the default priority of an ATB input port.

See also

- ETM
- ETM.state

ETM.TraceRESet Forces the ETM to emit all core resets

Format: **ETM.TraceRESet** [**ON** | **OFF**] (ETM≥4.0)

When **ETM.TraceRESet** is set to **ON** the ETMv4 is forced to emit all reset exceptions. Consider this setting if you are using trace filters for a selective trace instead of a complete program flow trace..

ON	The ETMv4 always emits any reset exception, regardless of the configured trace-filter.
OFF (default)	The ETMv4 emits only a reset exception if it has also traced the exception or instruction immediately prior to the reset error exception.

See "TRCVICTLR.TRCRESET" in ARM ETMv4 architecture specification for details.

See also

- ETM
- ETM.state

Format: **ETM.TRCIDR** *<spec_max> <p0_key_max> <p1_key_max> <p1_special_key_max> <cond_key_max> <cond_special_key_max> <id0> <id1> <id2>*
ETM.TRCIDR *<trcidr8> <trcidr9> <trcidr10> <trcidr11> <trcidr12> <trcidr13> <trcidr0> <trcidr1> <trcidr2>*

Sets the values of the registers TRCIDR8, TRCIDR9, TRCIDR10, TRCIDR11, TRCIDR12, TRCIDR13, TRCIDR0 in the simulator.

They are needed for offline analysis of traces with ETMv4 of custom cores (with **LA.IMPORT** command).

See also

- ETM
- ETM.state

ETM.VMID Virtual machine ID tracing

Format: **ETM.VMID** [**ON** | **OFF**]

The command **ETM.VMID** configures, if the ETM should emit Virtual Machine IDs to the trace stream. This command has an effect on Cortex-A (32- or 64-bit) target systems supporting virtualization (hypervisor), e.g. Cortex-A15 or Cortex-A57.

ON Enables Virtual Machine ID tracing. The VMID option shall be switched on when the hypervisor is used to switch between different guest operating systems on the target. Then special trace packages will be generated whenever the Virtual Machine ID (VMID) changes. This ensures the trace analysis tool knows about the complete context before decompressing any instructions.

OFF (default) Disables Virtual Machine ID tracing.

See also

- ETM
- ETM.state

OOO	Out-of-order data packet
CORECLOCK	

Examples for Trace Controlling

Tracing of a Specified Address Range

```
ETM.CLEAR                                       ; reset ETM

ETM.SET RANGE 0x1 EXECUTE P:0x9400--0x9500      ; define range1 for tracing
                                                ; execution cycles

ETM.SET TRACEENABLE RANGE 0x1                   ; trace only if in range 1

ETM.SET TRIGGER NOT TRUE                        ; switch trigger off
```

Tracing of Specified Data

```
ETM.CLEAR                                       ; reset ETM

ETM.SET RANGE 0x1 ACCESS v.range(flags)         ; define range 1 as variable
                                                ; flags for READ or WRITE
                                                ; access cycles

ETM.SET TRACEENABLE TRUE                        ; trace all instructions
ETM.SET VIEWDATA RANGE 0x1                      ; data trace only for range
                                                ; 1

ETM.SET TRIGGER NOT TRUE                        ; switch trigger off
```

Trigger at Address Access

```
ETM.CLEAR                                       ; reset ETM

ETM.SET RANGE 0x1 ACCESS v.range(flags)         ; define range 1 as variable
                                                ; flags for READ or WRITE
                                                ; access cycles

ETM.SET TRIGGER RANGE 0x1                       ; trigger on first access to
                                                ; range 1
```

Tracing of a Defined Amount of Cycles

```
ETM.CLEAR                                          ; reset ETM

ETM.SET RANGE 0x1 EXECUTE sieve--(sieve+30.)       ; define range 1 as
                                                   ; first 30 lines of
                                                   ; sieve

ETM.SET COUNT 1 0x10                               ; define count max value
                                                   ; to 10h

ETM.SET TRACEENABLE ! COUNT 1                      ; trace from count
                                                   ; reload until 0

ETM.SET COUNTENABLE 1 RANGE 0x1                    ; decement count 1 if
                                                   ; range 1

ETM.SET TRIGGER NOT TRUE                           ; switch trigger off
```

Runtime Measurement of a Function

```
ETM.CLEAR                                       ; reset ETM

ETM.SET RANGE 0x1 EXECUTE sieve--               ; range 1 begin of sieve
(sieve+1.)                                      ; range 2 end of sieve
ETM.SET RANGE 0x2 EXECUTE (sieve+0x98)--
(sieve+0xa0)

ETM.SET TRACEENABLE RANGE 0x1 || RANGE 0x2      ; trace if range1 or range2

ETM.SET TRIGGER NOT TRUE                        ; switch trigger off
```

Basics

Data trace is required by tracing real time operating systems (e.g. LINUX). This would be done by setting the data trace option of the ETM to BOTH (**ETM.DataTrace** or **ETM.state**).

Full data trace increases the need for bandwidth. At this point the trace port could reach its limit (**Fifofull**). Here it is required to reduce the amount of traced data with filters. Depending on the ETM revision two ways are possible: using ProcID or simple data filtering.

Trace Setup for LINUX

```
; reset ETM
ETM.CLEAR

; define range 1 as process data for READ or WRITE access cycles
ETM.SET RANGE 0x1 ACCESS TASK.CONFIG(MAGIC)++3

; trace all instructions
ETM.SET TRACEENABLE TRUE

; data trace only for range 1
ETM.SET VIEWDATA RANGE 0x1

; switch trigger off
ETM.SET TRIGGER NOT TRUE
```

FAQ

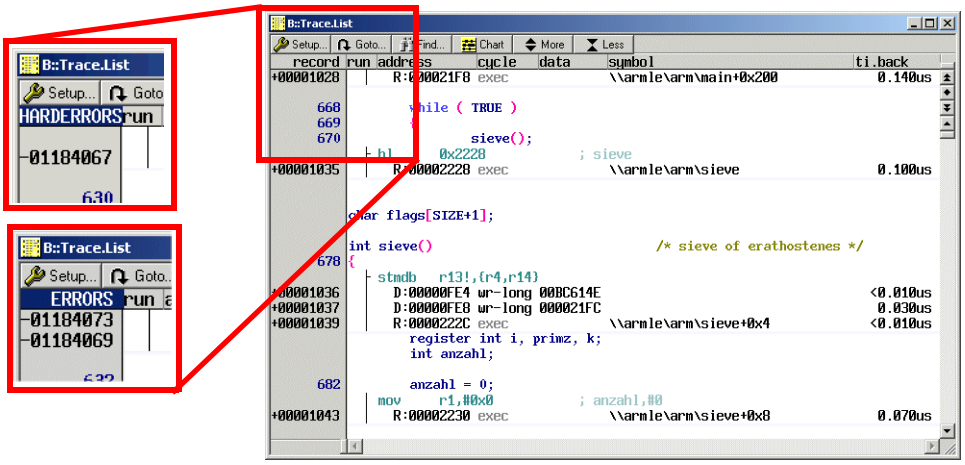
Please refer to https://support.lauterbach.com/kb.

Error Diagnosis

	Advanced trace analysis commands like **Trace.STATistic.Func**, **Trace.STATistic.TASK** or **PERF.List** display only accurate results if the trace recording works error free.

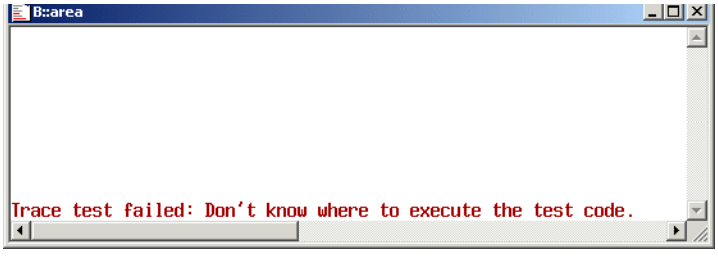
Error messages are displayed:

- In the upper left corner of the **Trace.List** window:



- In the message line:

- In the **Area.view** window:

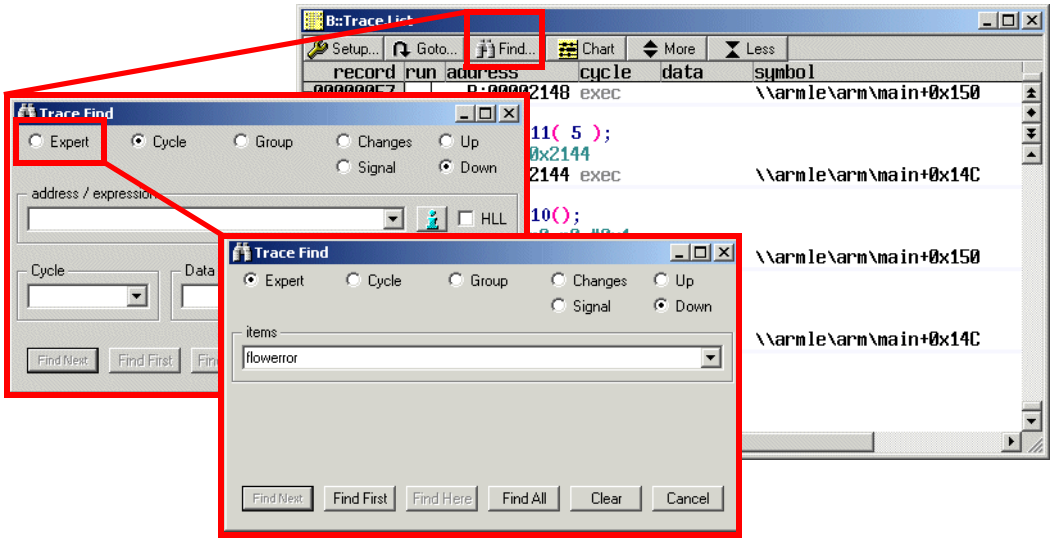


TRACE32 uploads only the requested trace information to the host to provide a quick display of the trace information. Due to this errors might be out of the uploaded range and therefore not visible immediately.

There are several ways to search for errors within the trace, all of them will force TRACE32 to upload the complete trace information to the host:

1. The *Trace Find* window

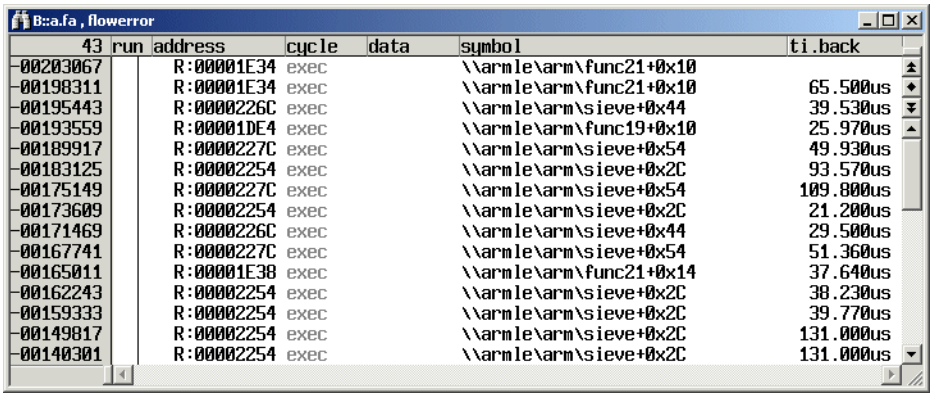
   Pushing the **Find** button of the **Trace.List** window a special search window opens:



   Select **Expert** and enter "flowerror" in the item field. The item entry is not case sensitive. Use the **Find First/Find Next** button to jump to the next flowerror within the trace. **Find All** will open a separate window with a list of all flowerrors.

2. Using command **Trace.FindAll , FLOWERROR**

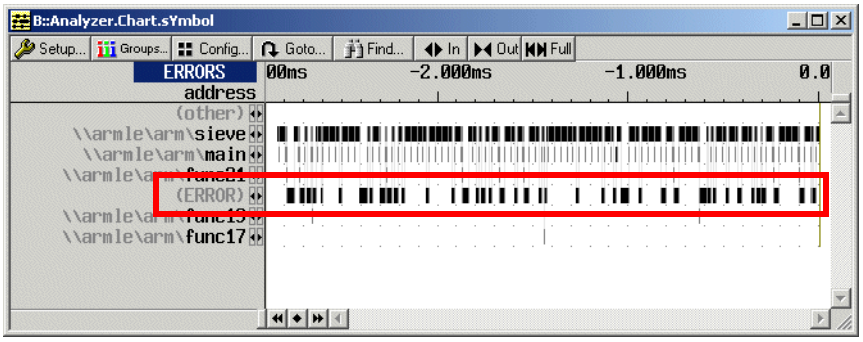
   This command will search within the entire trace buffer for **errors**. The records will be listed within a separate window. This command corresponds to the **FindAll** option described above.



43	run	address	cycle	data	symbol	ti.back
-00203067		R:00001E34	exec		\\armle\arm\func21+0x10	
-00198311		R:00001E34	exec		\\armle\arm\func21+0x10	65.500us
-00195443		R:0000226C	exec		\\armle\arm\sieve+0x44	39.530us
-00193559		R:00001DE4	exec		\\armle\arm\func19+0x10	25.970us
-00189917		R:0000227C	exec		\\armle\arm\sieve+0x54	49.930us
-00183125		R:00002254	exec		\\armle\arm\sieve+0x2C	93.570us
-00175149		R:0000227C	exec		\\armle\arm\sieve+0x54	109.800us
-00173609		R:00002254	exec		\\armle\arm\sieve+0x2C	21.200us
-00171469		R:0000226C	exec		\\armle\arm\sieve+0x44	29.500us
-00167741		R:0000227C	exec		\\armle\arm\sieve+0x54	51.360us
-00165011		R:00001E38	exec		\\armle\arm\func21+0x14	37.640us
-00162243		R:00002254	exec		\\armle\arm\sieve+0x2C	38.230us
-00159333		R:00002254	exec		\\armle\arm\sieve+0x2C	39.770us
-00149817		R:00002254	exec		\\armle\arm\sieve+0x2C	131.000us
-00140301		R:00002254	exec		\\armle\arm\sieve+0x2C	131.000us

3. Using command **Trace.Chart.sYmbol**

This command will start a statistic analysis. An additional symbol (ERROR) is shown if errors are found.



	The search could take a long time depending on the used memory size of the trace module and the type of host interface. Check the status to estimate the time.

One of the following 3 errors may occur:

- **HARDERROR**
- **FLOWERROR**
- **FIFOFULL**

Please see **Diagnosis Check List**.

HARDERRORS

There are no valid ETM data. Possible reasons are:

- ETM port multiplexed with other IO functions (no valid trace data)
- Trace signal capturing failed (setup/hold time violations)
- Wrong version of PowerTrace module
- Target frequency too high

Please see **Diagnosis Check List**.

FLOWERRORS

The traced data are not consistent with the code in the target memories. Possible reasons are:

- Memory contents have changed (e.g. self modifying code).
- Wrong trace data (as result of **HARDERRORS**)

Please see **Diagnosis Check List**.

The ETM output FIFO overflowed. The amount of trace data generated by the ETM was greater than the ETM port band width. To reduce the risk of a FIFO overflow:

- Increase the port size if possible (**ETM.PortSize**).
- Restrict the **DataTrace to read cycles** (write accesses can be reconstructed via **CTS**).
- Restrict the **DataTrace to write cycles**, a FIFO overflow becomes less likely.
- Reduce amount of trace data by using filters: use the filter **TraceEnable** or **TraceData**
- STALL the CPU if a FIFO overflow is likely to happen - ETMv1.x (**ETM.STALL**).
- Suppress the output of the data flow information if a FIFO overflow is likely to happen - ETMv3.x (**ETM.DataSuppress**).

The Embedded Trace Macrocell is not always able to prevent overflows of the internal FIFO. Even when STALL is enabled overflows may occur.

Trace Test Failed Messages

Trace.TestFocus supports a built in trace test. This command loads a short test program up to the target memory (RAM) and traces its execution. Afterwards the recorded program flow and data pattern will be checked for any errors.

"Analyzer data capture o.k." will be shown if the test was successful.

Test failures might be caused by a variety of reasons, usually error messages such as **"Trace test failed: not enough samples in the trace"** will give a clue to what might have caused the failure. Refer to **"Error Message Emulator"** in **"Error Messages"** (error.pdf) to find explanations on these messages.

Basic Checks

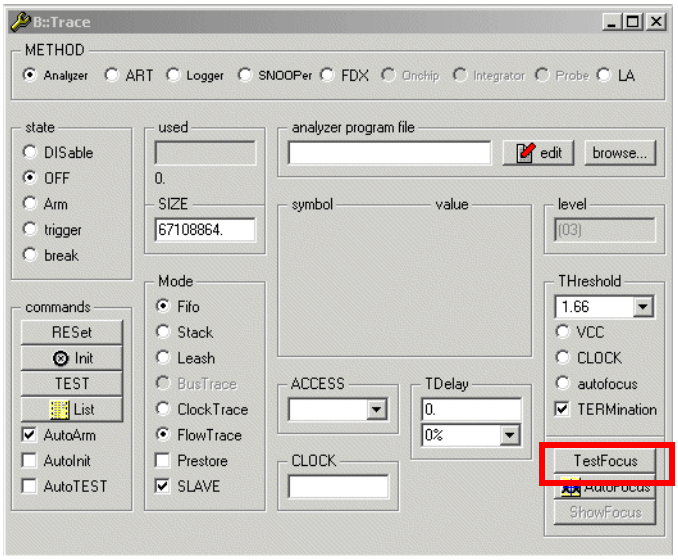
1. Did your debugger remain control over the target at all times when attempting to capture a trace? Error messages such as "**emulation debug port fail**" indicate that the debugger lost control over the target.

   In case your debugger lost control over the target:

   Is there a separate JTAG connector on your target?
   If available, connect the debug cable directly to this connector

   Double check your targets supply voltage. Is it stable when the trace port is active?

2. TRACE32 supports a build in ETM trace port check. It can simply be performed by clicking on the **"TestFocus"** button of the Trace window:



   This command loads a short test program up to the target memory (RAM) and traces its execution. Afterwards the recorded program flow and data pattern will be checked for any errors.

   "Analyzer data capture o.k." will be shown if the test was successful.

   Test failures might be caused by a variety of reasons, usually error messages such as **"Trace test failed: not enough samples in the trace"** will give a clue to what might have caused the failure. Refer to **"Error Message Emulator"** in **"Error Messages"** (error.pdf) to find explanations on these messages.

In case of the Preprocessor for ARM-ETM with AUTOFOCUS execute **Trace.AutoFocus** instead. Both an automatic hardware configuration as well as the trace test described above will be executed.

```
; Example: extended trace test for LA-7991

; 1.) Hardware configuration and trace test
Trace.AutoFocus

; 2.) Extended trace test
; Accumulate data eye information (10 runs)

&i = 0

WHILE (&i<10.)
(
     Trace.TestFocus /Accumulate
     &i=&i+1
)

; 3.) Show results
Trace.ShowFocus
```

3. Some cpu types do not have dedicated trace port pins. Instead trace signals are muxed with other signals. A special port pin setup may be required to get trace functionality. Example:

```
PER.Set SD:0x111D640 %Word 0x9AA0     ; Enable ETM functionality on
PER.Set SD:0x111D6A4 %Word 0x2901     ; GPIO's

PER.Set 0x111600D %Long %LE 0x01E     ; Enable CLK
```

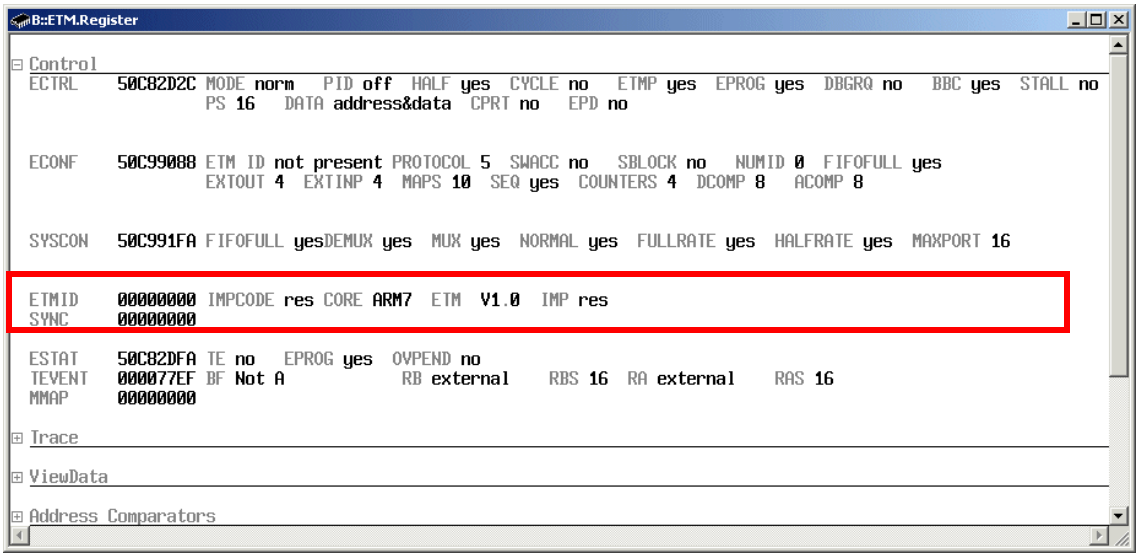
Check your cpu manual for the correct port pin configuration.

Advanced target applications could use more than one initialisation procedure or the setup might change during run time again. Make sure that the ETM port is actually enabled when attempting to trace.

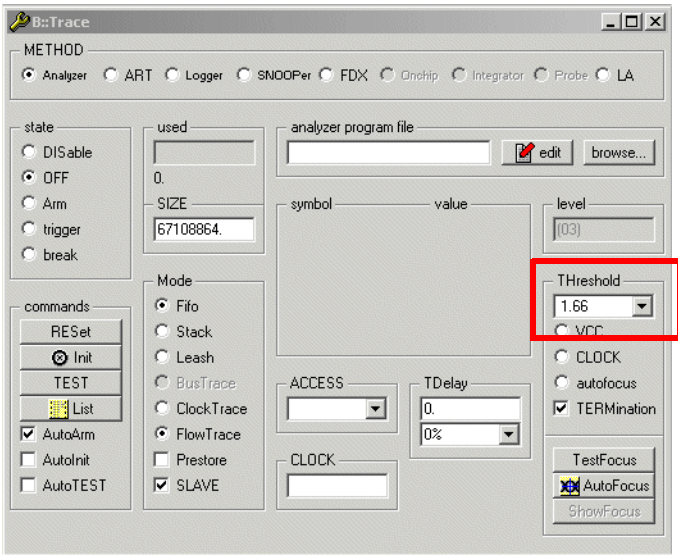
4. In case of shared ETM pins additional buffers may be used on the target hardware. Make sure that these buffers are enabled.

5. Check your CPU manual for correct ETM settings such as **ETM.PortSize**, **ETM.PortMode**, **ETM.HalfRate**. Wrong settings result in errors.

   Additional help offers the ETM register called SYSCON. It is available for ETM versions 1.2 or higher by executing **ETM.Register**:



6. Most of the ETM trace hardware is set up automatically. Special care has to be given to the threshold level (**Trace configuration**):



Should be close to 50% of the ETM signal voltage level.

The threshold value(s) for clock and/or data signals is/are set automatically at software start, depending on the voltage level of pin 12 of the target connector. However, if your target is turned off or not connected during software start you may need to execute **Trace.THreshold VCC** or manually set it to the proper value (e.g. 0.9 V , 1.25 V or 1.65 V).

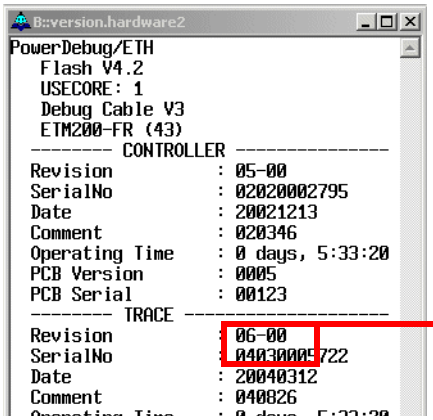
7. Check trace port datarate. Depending on the ETM version the data rate of the trace port is coupled on the core clock frequency or not:

   ETMv1/2: Bw=fcore * 1bit/ch

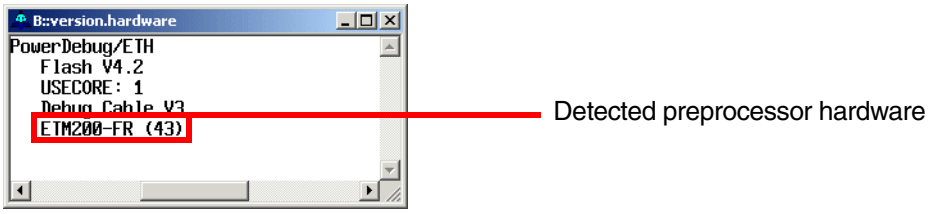
   ETMv3: Bw=fport * 2bit/ch

Product Number	Datarate (Bw)
PowerTrace LA-7690, LA-7707	<320Mbit/ch
PowerTrace II LA-7692, LA-7693, LA-7694	800Mbit/ch

8. Check the version of your PowerTrace 1 module. Version 6 is required for targets running faster than 200 MHz (**VERSION.HARDWARE2**).



9. Check the preprocessor seen by TRACE32 software. **VERSION.HARDWARE** shows all scanned hardware. The different ID's are explained below.



```
ETM120              LA-7889

ETM-Full            LA-7923 without ETMv3 support
ETM-Full V2         LA-7923 with full support

ETM200-HR           LA-7921 HalfRate
ETM200-FR           LA-7921 FullRate
ETM200-cTools       LA-7921 for cTools

ETM270-HR 1.8V      LA-7990
ETM270-FR 1.8V      1.8V/3.3 V depends on the threshold level. Values
ETM270-HR 3.3V      higher than 1 V are marked as 3.3 V, levels lower
ETM270-FR 3.3V      than 1 V are marked as 1.8 V

ETM-AF              LA-7991(re-programmable)
ETM-AF(OTP)         LA-7991(one-time-programmable)

ETM-HSSTP           LA-7988
```

For ARM-ETM200 and ARM-ETM270 also refer to **configuration test** for more details.

10. In case of preprocessors with more than one Mictor connector double-check that the mictor connectors are connected properly to your target. For ETM trace port sizes below 16 bit TRACE B remains unconnected. For further information on the correct connection of TRACE A / TRACE B refer to **Dimensions**.

11. Check trace port pinout?

Did your target board work with other PowerTrace/Preprocessor combinations (or TPAs from other vendors)? If your target worked with other PowerTrace/Preprocessor combinations your trace port pinout can be assumed to be correct. This is not always true for TPAs from other vendors. LAUTERBACH tools assume that trace port pinout follows exactly the ETM specification (please refer to **Connector Layout**).

Did the PowerTrace/Preprocessor combination work for other targets? If yes, what has changed on your new target board? Often messages such as **Trace test failed: not enough samples in the trace** or **Trace test failed: pin connection error** might indicate the source of error.

```
Error message                          Possible reason

Trace test failed:                     -trace port not enabled
not enough samples in the trace        -buffers not enabled
                                       -threshold out of signal range

Trace test failed:                     -short-circuit between pins
pin connection error                   -wrong connector pin out
                                       -unsupported ETM mode
                                       -trace port not enabled
```

Also check the voltage level of the reference voltage on pin 12. It is used as a reference for all ETM signals. It should correspond to the amplitude of your trace signals. For some targets this might differ from the JTAG signals. Pin 14 is used as reference for all JTAG signals in case the JTAG debugger is connected via the trace preprocessor probe.

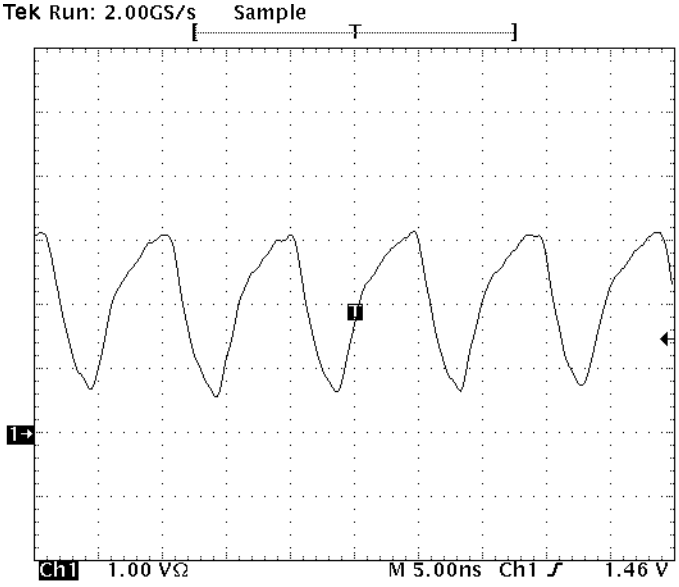
One way of testing the voltage on pin 12 is executing **Trace.THreshold VCC** and double-checking that the voltage displayed in **Trace Configuration** window is approximately 50% of your target voltage.

12. If supported by your Preprocessor try both settings for the termination voltage: .

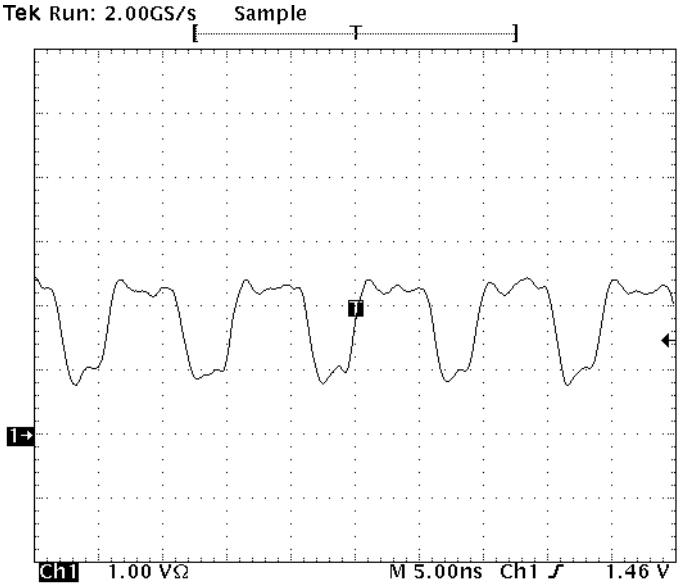
```
Trace.TERMination ON            ; Connect termination voltage during
                                ; trace capture

Trace.TERMination OFF           ; Always disconnect termination voltage
```

In most cases it is best to have the termination voltage connected during trace capture. However for lower frequencies and/or weak output drivers it can be helpfull to disable the termination. This feature is supported by ARM-ETM200 and ARM-ETM270 as wells as ARM-ETM with AUTOFOCUS.

Usually an unterminated signal will result in slower rise and fall times and it might have have over- and undershots. For Preprocessors ARM-ETM with AUTOFOCUS you will notice the slower rise and fall times by a reduced data eye size (white areas in the **Trace.ShowFocus** window):



The terminated signal however will have a reduced amplitude, "gravitating" towards 50% of the reference voltage on pin 12 of the target connector (see **Connector Layout**), as well as faster rise and fall times:



13. Is the TRACE32 software up-to-date?

```
VERSION.SOFTWARE
```

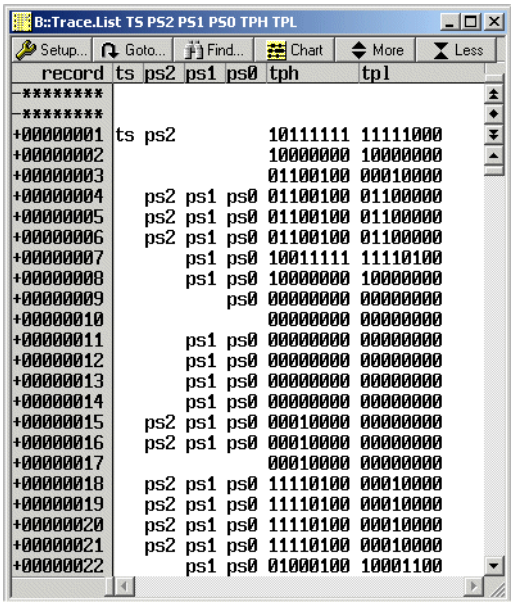
Check **www.lauterbach.com/download_trace32.html**or ask your local support team for an update or contact support@lauterbach.com.

If the basic check was completed and the trace results are still not faultless the following procedure might help you to determine the source of error. By manually disassembling the captured trace data (for a single step) you might notice swapped pins or data channels that might have timing issues.

	Electrical measurements must be done by trained personal only. Special care has to be taken concerning electrostatic discharge.

	The target must run and the trace has to be armed during electrical measurements to ensure trace port activity.

1. Checking the ETM trace signals can be done with special options within the **Trace.List** window. Use "**Trace.List TS PS2 PS1 PS0 TPH TPL**" to display the captured trace data in a format that is easily human readable:



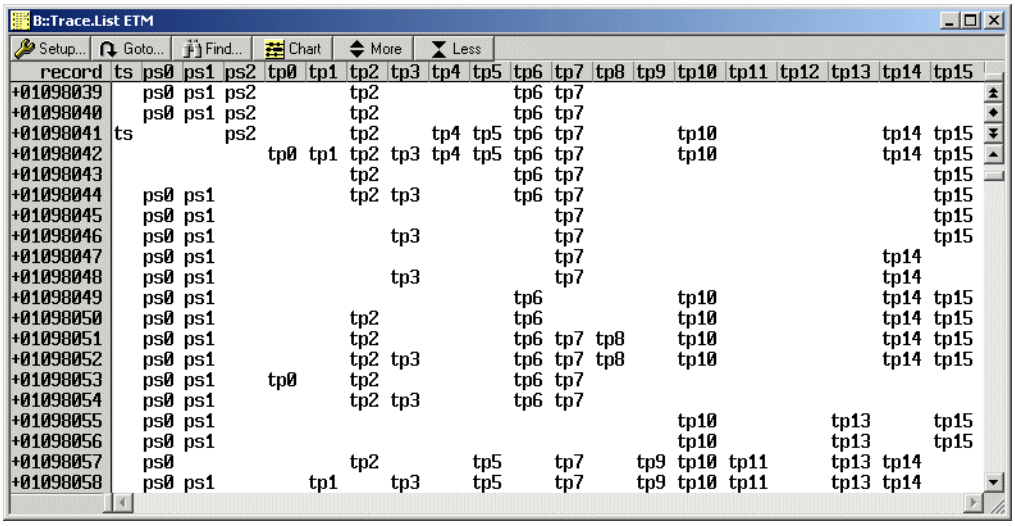
record	ts	ps2	ps1	ps0	tph	tpl
-********						
-********						
+00000001	ts	ps2			10111111	11111000
+00000002					10000000	10000000
+00000003					01100100	00010000
+00000004		ps2	ps1	ps0	01100100	01100000
+00000005		ps2	ps1	ps0	01100100	01100000
+00000006		ps2	ps1	ps0	01100100	01100000
+00000007			ps1	ps0	10011111	11110100
+00000008			ps1	ps0	10000000	10000000
+00000009				ps0	00000000	00000000
+00000010					00000000	00000000
+00000011			ps1	ps0	00000000	00000000
+00000012			ps1	ps0	00000000	00000000
+00000013			ps1	ps0	00000000	00000000
+00000014			ps1	ps0	00000000	00000000
+00000015		ps2	ps1	ps0	00010000	00000000
+00000016		ps2	ps1	ps0	00010000	00000000
+00000017					00010000	00000000
+00000018		ps2	ps1	ps0	11110100	00010000
+00000019		ps2	ps1	ps0	11110100	00010000
+00000020		ps2	ps1	ps0	11110100	00010000
+00000021		ps2	ps1	ps0	11110100	00010000
+00000022			ps1	ps0	01000100	10001100

Example: 16bit ETMv1.x port

When using the option **ETM** (**Trace.List ETM**) all trace signals independent of the ETM version are shown. Although the format is slightly different than what we have seen previously.

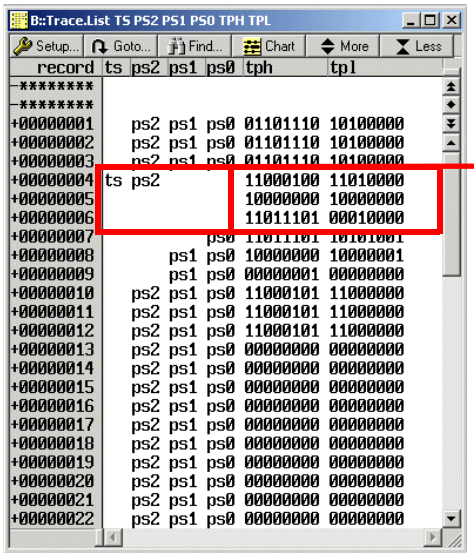
ETMv1 signal	Description
TS	Trace sync signals are synchronization points for the TRACE32 software. Invalid states will cause Harderrors and Flowerrors
PS2, PS1, PS0	Trace status signals correspond to pipline states at execution time. Invalid states will cause Harderrors and could freeze TRACE32 for a while.
TP (TPH, TPL)	Trace packet signals hold information about data, address and program counter. TPL correspond to TP[7..0] and TPH correspond to TP[15..8]. Invalid values will cause Flowerrors

This what our 16bit ETMv1.x port would look like, if we use "Trace.List ETM":



record	ts	ps0	ps1	ps2	tp0	tp1	tp2	tp3	tp4	tp5	tp6	tp7	tp8	tp9	tp10	tp11	tp12	tp13	tp14	tp15
+01098039		ps0	ps1	ps2			tp2				tp6	tp7								
+01098040		ps0	ps1	ps2			tp2				tp6	tp7								
+01098041	ts			ps2			tp2		tp4	tp5	tp6	tp7			tp10				tp14	tp15
+01098042					tp0	tp1	tp2	tp3	tp4	tp5	tp6	tp7			tp10				tp14	tp15
+01098043							tp2				tp6	tp7								tp15
+01098044		ps0	ps1				tp2	tp3			tp6	tp7								tp15
+01098045		ps0	ps1									tp7								tp15
+01098046		ps0	ps1					tp3				tp7								tp15
+01098047		ps0	ps1									tp7							tp14	
+01098048		ps0	ps1					tp3				tp7							tp14	
+01098049		ps0	ps1								tp6				tp10				tp14	tp15
+01098050		ps0	ps1				tp2				tp6				tp10				tp14	tp15
+01098051		ps0	ps1				tp2				tp6	tp7	tp8		tp10				tp14	tp15
+01098052		ps0	ps1				tp2	tp3			tp6	tp7	tp8		tp10				tp14	tp15
+01098053		ps0	ps1		tp0		tp2				tp6	tp7								
+01098054		ps0	ps1				tp2	tp3			tp6	tp7								
+01098055		ps0	ps1												tp10			tp13		tp15
+01098056		ps0	ps1												tp10			tp13		tp15
+01098057		ps0					tp2			tp5		tp7		tp9	tp10	tp11		tp13	tp14	
+01098058		ps0	ps1			tp1		tp3		tp5		tp7		tp9	tp10	tp11		tp13	tp14	

2. Enable the AutoInit mode (**Analyzer.AutoInit ON**) and do one single step to get a trace capture. The trace list window for ETMv1.x will look similar to this:



record	ts	ps2	ps1	ps0	tph	tpl
-********						
-********						
+00000001		ps2	ps1	ps0	01101110	10100000
+00000002		ps2	ps1	ps0	01101110	10100000
+00000003		ps2	ps1	ps0	01101110	10100000
+00000004	ts	ps2			11000100	11010000
+00000005					10000000	10000000
+00000006					11011101	00010000
+00000007				ps0	11011101	10101001
+00000008			ps1	ps0	10000000	10000001
+00000009			ps1	ps0	00000001	00000000
+00000010		ps2	ps1	ps0	11000101	11000000
+00000011		ps2	ps1	ps0	11000101	11000000
+00000012		ps2	ps1	ps0	11000101	11000000
+00000013		ps2	ps1	ps0	00000000	00000000
+00000014		ps2	ps1	ps0	00000000	00000000
+00000015		ps2	ps1	ps0	00000000	00000000
+00000016		ps2	ps1	ps0	00000000	00000000
+00000017		ps2	ps1	ps0	00000000	00000000
+00000018		ps2	ps1	ps0	00000000	00000000
+00000019		ps2	ps1	ps0	00000000	00000000
+00000020		ps2	ps1	ps0	00000000	00000000
+00000021		ps2	ps1	ps0	00000000	00000000
+00000022		ps2	ps1	ps0	00000000	00000000

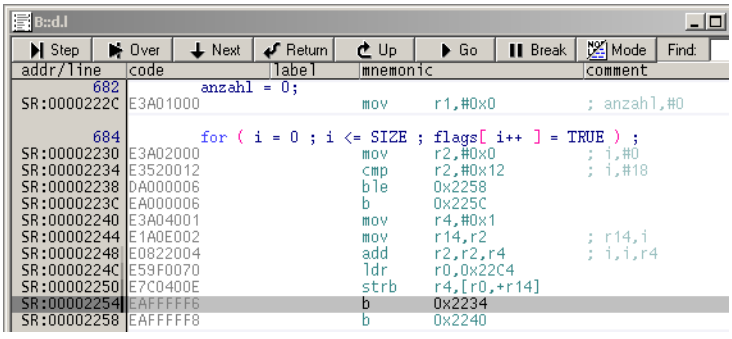
This trace syncronisation (ts) is similar for all single steps.

When looking for sources of error (e.g. pin swapping in your trace port pinout or timing issues) you may want to disassemble the captured data manually. Generally this requires some knowledge of the trace protocol (for more information refer to the ETM architecture specification). However below is an example of how to reconstruct the execution address from the traced data without knowing too much

about the trace protocol. Note that 5 bytes are needed to transmitt a 32-bit address on the trace port:

1 1000100 1 1010000 1 0000000 1 0000000 0 0010000	Trace packet bits can be structured as shown to the left. Each eighth bit is set to one if a packet follows and is cleared for the last packet.
1 1010000 1 1000100 1 0000000 1 0000000 0 0010000	Trace packet bits can be sorted as shown to the left.
101.0000 10.0010.0 0.0000.00 0000.000 001.0000	Dividing the row into groups of four bits (nibble) gives the address in binary format. Since the address is transmitted LSB first, this is done from right to left and top to bottom.
0101.0000 0010.0010 0000.0000 0000.0000 001	Now we write two nibbles per line again from right to left and top to bottom. Bits[6:4] of the fifth packet of a full branch address contain a reason code (here: 0x1 = "tracing has been enabled").
5 0 2 2 0 0 0 0 001	Finally the conversion to the hexadecimal format gives the full address 0x00002250.

Now compare this address to the one you would expect by looking at the **Data.List** window. Since you executed a single step you would expect the address before the current PC location to be the one transmitted on the trace port:



The grey cursor indicated the PC location

3. You may need to perform a couple of single steps to see, if there is a logical error (pairs of trace channels are swapped) indicating an erroneous pinout or if the errors seem to be related to certain channels being wrong every now and than. For the later timing violations could be the problem. All Preprocessors but the ETM-AF Preprocessor (LA-7991) have no or a rather limited ability to adjust the sampling point(s) so they will work only, if their **timing requirements** are satiesfied. In addition the maximum operation frequency has to be considered.

If the basic check was completed and the trace results are still not faultless the target connector pinout should be checked again (**ETMv3**).

If the basic check was completed and the trace results are still not faultless the following procedure might help you to determine the source of error. By manually disassembling the captured trace data (for a single step) you might notice swapped pins or data channels that might have timing issues.

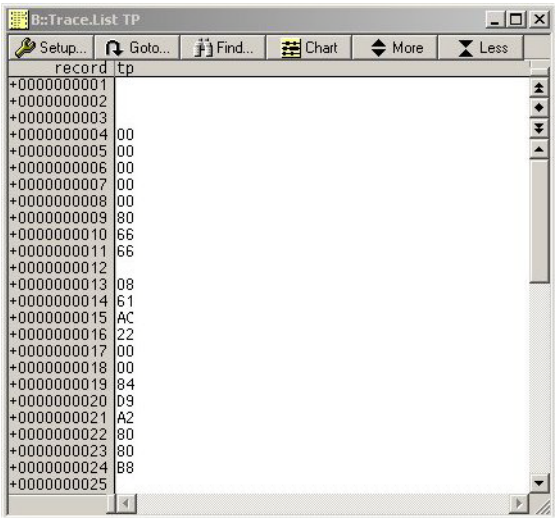
Pin 34,36 and 38 of the connector are handled by the trace hardware as normal data signals. That means this signals will also be affected by the termination circuitry. .

It is required to connect pin 34 directly to VCC and pin 30 and 32 directly to GND.

Electrical measurements must be done by trained personal only. Special care has to be taken concerning electrostatic discharge.

The target must run and the trace has to be armed during electrical measurements to ensure trace port activity.

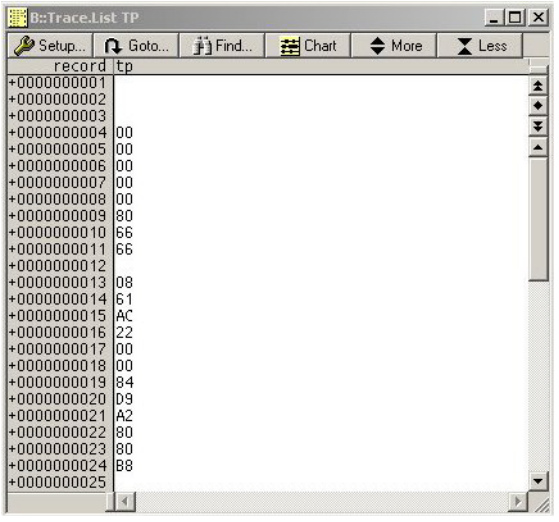
1. Checking the ETM trace signals can be done with special options within the **Trace.List** window. Use "**Trace.List TP**" to display the captured trace data in a format that is human readable:



Example: 8bit ETMv3.x port

ETMv3 signal	Description
TCTL	Trace control signal defines valid TP packets. The signal is low active. Invalid states will cause Harderrors and Flowerrors
TP	Trace packet signals hold information about data, address and program counter. Invalid values will cause Flowerrors

2. Enable the AutoInit mode (**Analyzer.AutoInit ON**) and do one single step to get a trace capture. The trace list window will look similar to this:

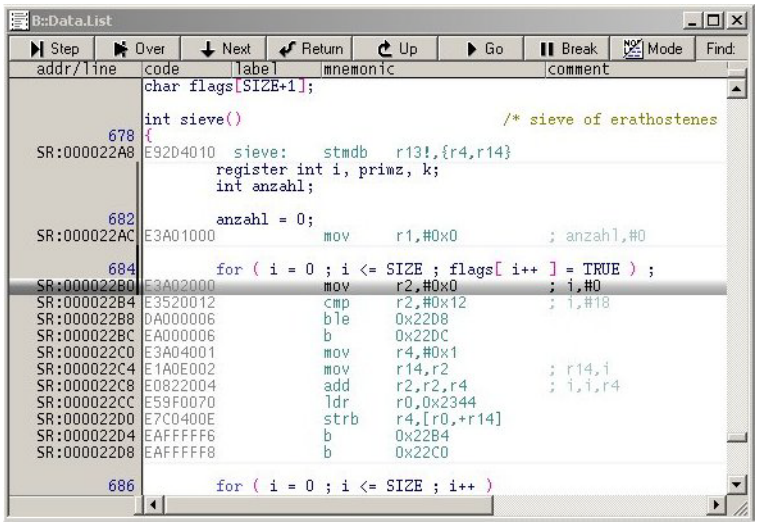


When looking for sources of error (e.g. pin swapping in your trace port pinout or timing issues) you may want to disassemble the captured data manually. Generally this requires some knowledge of the trace protocol (for more information refer to the ETM architecture specification). However below an example of how to reconstruct the execution address from the traced data without knowing too much

about the trace protocol.:

00 00 00 00 00 80	syncronization sequence (ISYNC)
66 66	idle sequence (IDLE)
08 61	address syncronization sequence (ASYNC)
AC 22 00 00	address (0x000022AC)
84 D9 A2 80 80 B8	program execution information sequence (PHEADER)

3. Now compare this address to the one you would expect by looking at the **Data.List** window. Since you executed a single step you would expect the address before the current PC location to be the one transmitted on the trace port:

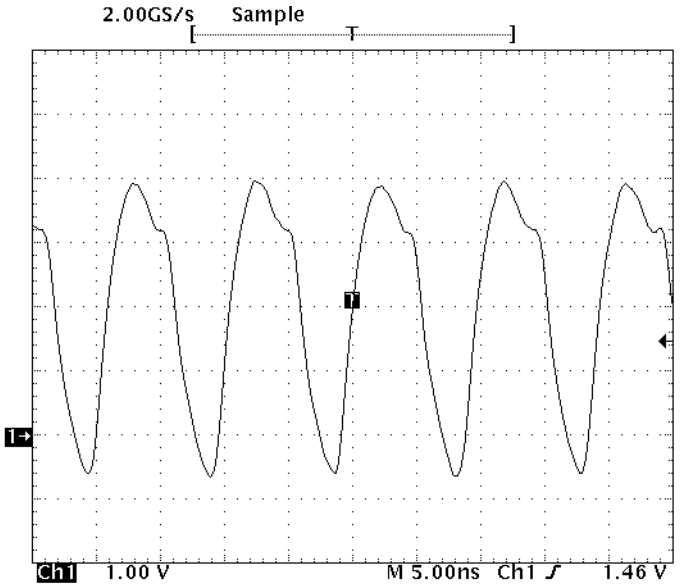


The grey cursor indicated the PC location

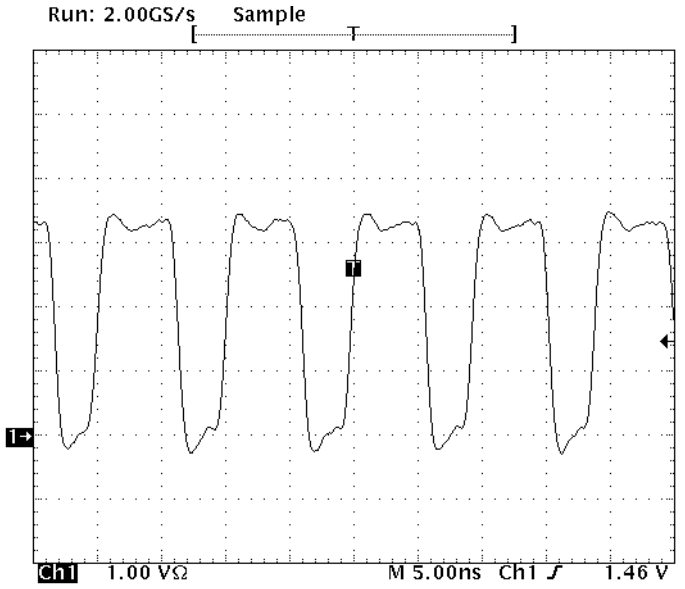
4. You may need to perform a couple of single steps to see, if there is a logical error (pairs of trace channels are swapped) indicating an erroneous pinout or if the errors seem to be related to certain channels being wrong every now and than. For the later timing violations could be the problem. All Preprocessors but the ETM-AF Preprocessor (LA-7991) have no or a rather limited ability to adjust the sampling point(s) so they will work only, if their **timing requirements** are satiesfied. In addition the maximum operation frequency has to be considered.

If you suspect timing issues to be the source of error, you may need to take a closer look at the setup and hold times of your trace port channels. An oscilloscope (2 channel, bandwidth >500 MHz, >5 GS/s, probe <5pF) is required for the following measurements.

Make sure that you use short ground connections. The following two screen shots show the influence of probe and ground connection length:

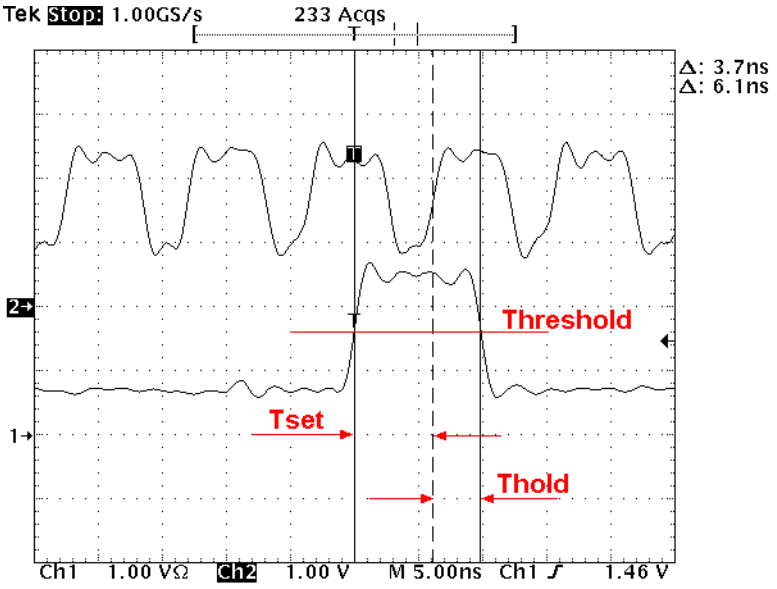


Long GND connection, Probe: 500 MHz, 8 pF, 10 MΩ, x10



Short GND connection, Probe: 750 MHz, < 2 pF, 1 MΩ, x10

Measure the setup and hold times for all data channels, which will also give you the maximum channel to channel skew. No trace probe can handle data skew, except the **Preprocessor for ARM-ETM with AUTOFOCUS** (LA-7991). So for all other probes setup and hold times have to be fullfilled for all channels. The following picture shows how to measure setup/hold times.



The threshold level should be set to the middle of the trace signal. The following table shows setup and hold time requirements for all ETM trace probes.

ARM ETM120 LA-7889, LA-7923	Ts: 3 ns Th: 2 ns
ARM ETM200 LA-7921	Ts: 3 ns Th: 1 ns
ARM ETM270 LA-7990	Ts: 0.9 ns Th: 1.1 ns
ARM ETM AUTOFOCUS LA-7991	Ts + Th >= 1.2 ns
ARM ETM AUTOFOCUS 2 LA-7992	Ts + Th >= 0.6 ns

The Preprocessor for ARM-ETM with AUTOFOCUS is very powerful, if it comes to configuring itself for different target trace ports. Usually you do not need to worry about setup and hold times for data rates as high as 200 Mbit/s, although it is always a good idea to keep the channel to channel skew low and the clock duty cycle close to 50/50. Depending on your trace port data rate the maximum channel to channel skew has to stay below:

ARM ETM AUTOFOCUS 1/2 LA-7991/LA-7992	ch2ch_skew <= t_period - (Ts + Th)

For instance a trace port with a data rate of 300 Mbit/s should have no more than (1/300 MHz - 0.6 ns) = 1.33 ns channel to channel skew. In addition the maximum operation frequency has to be considered.

Configuration Test

The extensions HR and FR specify the selected clocking mode: halfrate or fullrate.

The preprocessors ARM-ETM200 and ARM-ETM270 become firmware reconfigured in case of changing clocking modes and/or changing threshold levels (ARM-ETM270). A proceeding reconfiguration is shown in the status line by a moving bar after the target was started.

The preprocessor ID will be changed in case of success. Check **VERSION.HARDWARE**. The test should be done as follows:

- C*h*ange halfrate option (ETM.HalfRate ON->OFF or vice versa)
- Start target program and break (Go, Break)

TRACE32 should reprogram the preprocessor.

- Check VERSION.HARDWARE

The ID should have changed as expected. Especially check the HR/FR index.

- Change halfrate option (ETM.HalfRate back)
- Start target program and break again (Go, Break)

TRACE32 should reprogram the preprocessor again.

- Check VERSION.HARDWARE

	The preprocessor hardware of ETM200 and ETM270 consist of two levels which can be updated. The high level part will be configured in case of changing the clocking mode (halfrate/fullrate) and/or threshold level. If design updates are available they are included in every software update. Reconfigurations as described above will activate these design updates. However the low-level part is not software updatable.

Access the Diagnosis Tool

Lauterbach provides a diagnosis tool for the **Preprocessor with AUTOFOCUS**. It is recommended that you add the **AF Diagnosis** button to the TRACE32 toolbar for quick access to the diagnosis tool.

To access the diagnosis tool via a toolbar button:

1. Add this script line to the PRACTICE script file system-settings.cmm in your TRACE32 system directory:

```
DO ~~/demo/etc/diagnosis/autofocus/add_afdiag_button.cmm
```

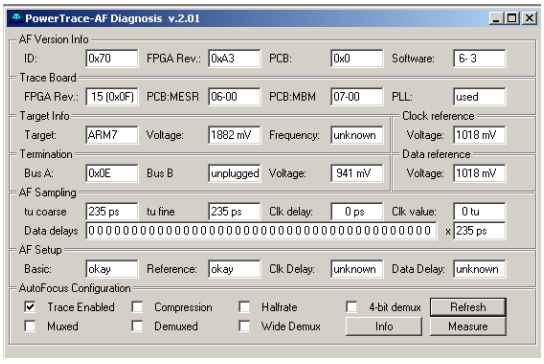
2. Re-start TRACE32.

   The **AF Diagnosis** button is now available on the TRACE32 toolbar.

To just start the diagnosis tool, execute the following command:

```
DO ~~/demo/etc/diagnosis/autofocus/afdiagnosis.cmm
```

If there is no **Preprocessor for ARM-ETM with AUTOFOCUS** attached executing this script will generate an error message, otherwise it displays some diagnosis results in the following window:



The diagnosis tool is just a way of communicating the current state of the **Preprocessor for ARM-ETM with AUTOFOCUS** (or at least what the low-level driver software thinks the current state should be). Do not feel discouraged, if some of the diagnosis results displayed in the window are meaningless to you. Instead follow the steps under **Diagnosis Check List**.

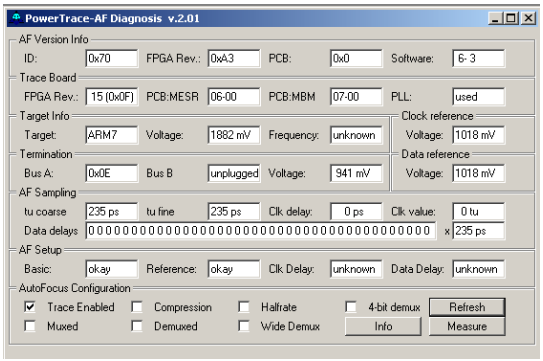
Diagnosis Check List

1. Make sure that you completed the **basic check** list, especially do not forget to execute **Trace.AutoFocus** after setting up your trace port with the frequency you wish to trace and before attempting to trace signals. Do you get any error or warning messages? (for very high frequencies you may need to repeat this command until the hardware configuration is successful).

2. What is your target voltage? For Preprocessors with pluggable termination module: is the proper termination PCB plugged in? For further information refer to **The Termination PCB**.

3. Use the diagnosis tool for **Preprocessor for ARM-ETM with AUTOFOCUS** to read in the current state of the driver software / Preprocessor. After executing commands that might change the state of the Preprocessor (e.g. **Trace.AutoFocus**) use the **Refresh** button to load current values

from the low-level driver into the diagnosis window. Press the "Measure" button, if you want to repeat measurements like the target voltage, frequency etc.. The Info button will display even more detailed information and also generate a log file.



Check the following:

Target Info - Voltage
Target Info - Frequency (displays the ETM frequency)

	The ETM clock frequency does not necessarily equal the CPU clock frequency. E.g. an ETMv1 or ETMv2 operating at HalfRate results in an ETM clock frequency that is half the CPU clock frequency, an ETMv3 operating with PortMode 1/2 results in an ETM frequency that is a quarter of the CPU clock frequency. The frequency measurement might not be very accurate for frequencies below 50 MHz.

Clock and Data reference voltage should normally be somewhere close to 50% of the target voltage

Termination - Bus A / Bus B (unplugged indicates that no termination is connected)
Check the stability of your target voltage by pressing the Measure button a couple of times and viewing the minimum and maximum values that were measured for that session by pressing the Info button:

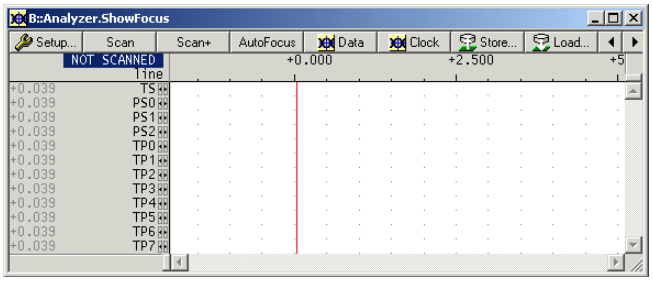
```
Target Voltage            : 1882 mV
 Minimum measured         : 1882 mV (minimum of 3 measurements)
 Maximum measured         : 1882 mV (maximum of 3 measurements)
```

4. The trace data are sampled at clock edges, on rising only or rising & falling. They must be stable around the clock edge over a short time. This time window of stable data is reflected in so-called data eyes.

	The data seen by the TRACE32 preprocessor depend highly from the clock threshold setup. Depending on the signal integrity the data eyes are more or less wide open.

TRACE32 offers a powerful feature called **Trace.ShowFocus** to analyse the signal integrity of the trace port. The functionality is similar to a sampling scope

TRACE32 differs two types of data eyes: analog and digital. The analog view is available for clock and data separately. **Analyzer.ShowFocus** opens the digital showfocus window:



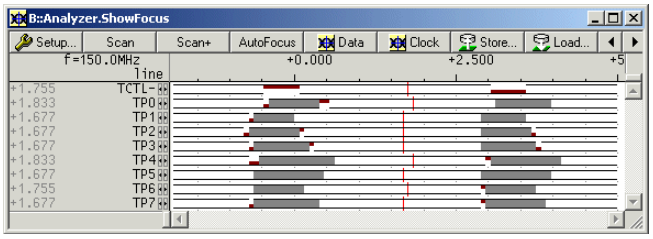
The horizontal axis reflects time line in nano seconds. On the left hand side the current delay is shown for each trace signal. The red line shows the sampling point. It can be different for each signal. Data lines are delayed if values smaller than zero are set or not all sampling points are equal. In case of values larger than zero the clock line is delayed.

	The data delay is used to eliminate data skew. Clock delay is used to correct setup and hold times. Both delays types can be used at the same time.

The horizontal axis shows all enabled signals listed. That means the number of signals depends on the trace port setup. The window needs to be re-opened after the trace port setup has changed.

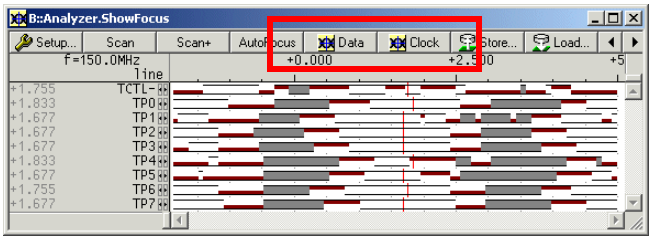
	**Analyzer.ShowFocus** is a kind of digital view of the data eyes. White areas mean stable data. Grey areas mean instable data (setup violations) on rising & falling clock edges. Red lines mean instable data on rising edges (upper line) or falling edges (lower line).

Pressing the **SCAN** button will execute **Analyzer.TestFocus** to update the window. In best case it should look like the following:



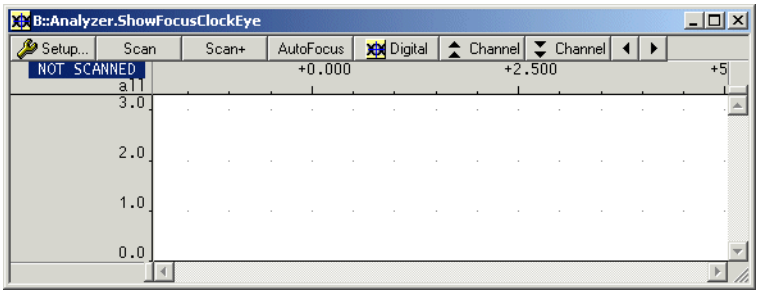
In this case no timing issue are expected. The sampling point should be placed in the middle of the white areas.

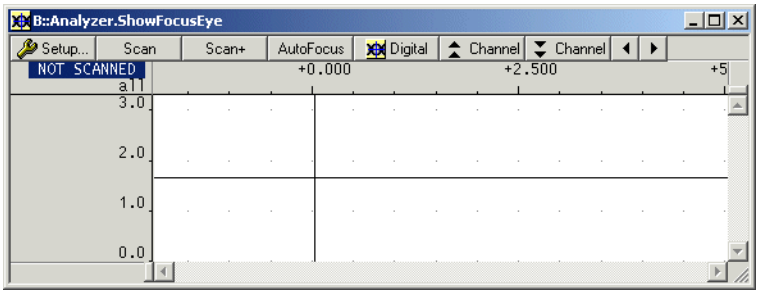
In comparison to the following scan results, where more investigations are required:



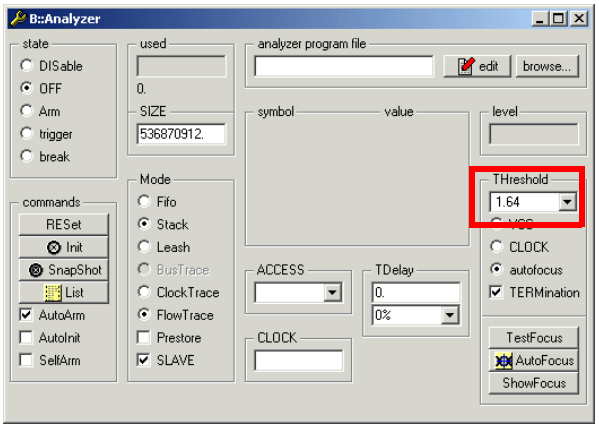
Red lines mark setup violations on single edges (red top: rising edge, red bottom: falling edge), grey areas mark setup violation on both, rising and falling edges.

Use the bottons DATA and CLOCK to open **Analyzer.ShowFocusEye** and **Analyzer.ShowFocusClockEye**. The vertical axis now shows the voltage [V], the horizontal axis still is the time [ns] axis:



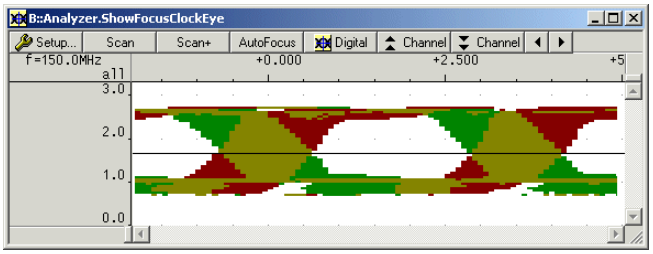


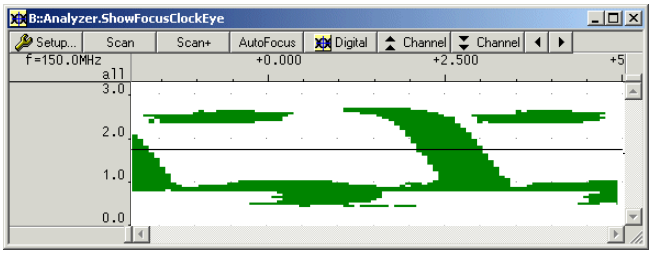
At the beginning the message NOT SCANNED is shown. The data threshold should be set to the middle of the trace port I/O voltage, before pressing the SCAN-button:

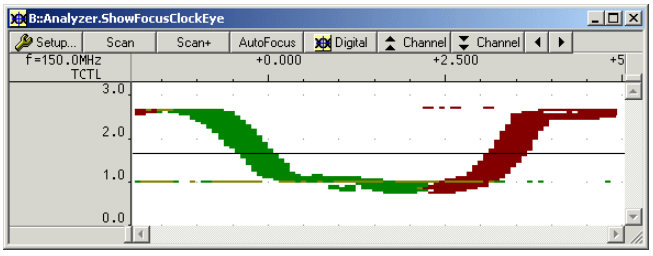


The ShowFocusClockEye window must be observed first. The integrity of the clock signal is important

for any further data eye analysis, because it is used for sampling. The result may look like the following figures:



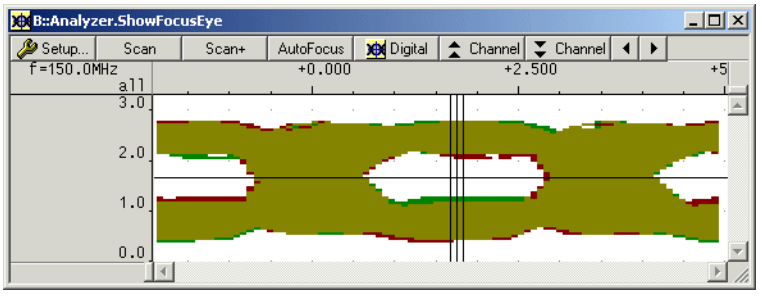




The graph does NOT show data eyes, but can be interpreted in a similar way (**How to understand**).

	The colored parts should define a white area, which can be enclosed or open on the upper/lower side. The clock threshold should be set to the middle of this area (as shown above).

After the clock threshold is set, press the **SCAN** button of the **Analyzer.ShowFocusEye** window. The result may look like the following:



	The graph shown in Analyzer.ShowFocusEye is similar to a scope measurement. It reflects the trace signal seen by the preprocessor. The given values can not be guaranteed, but give an suitable image of the signal integrity.

How to understand A.ShowFocusEye and A.ShowFocusClockEye

The **Analyzer.ShowFocusClockEye** window is frequently confused with **Analyzer.ShowFocusEye**. Both windows show similar graphs, but contain totally different information.

	The dimensions of both, the data eyes and the clock waveform, do highly depend on the clock threshold level.

The following figures show the resulting sample clock signal in dependency of the clock threshold level setup:

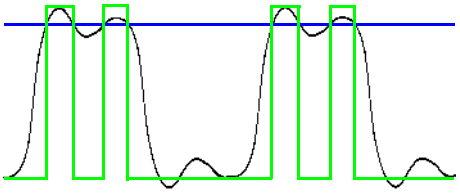
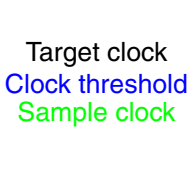


Fig.A



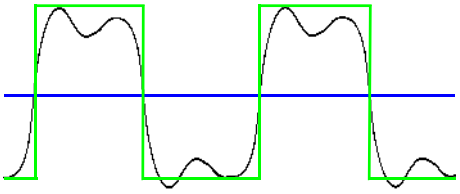
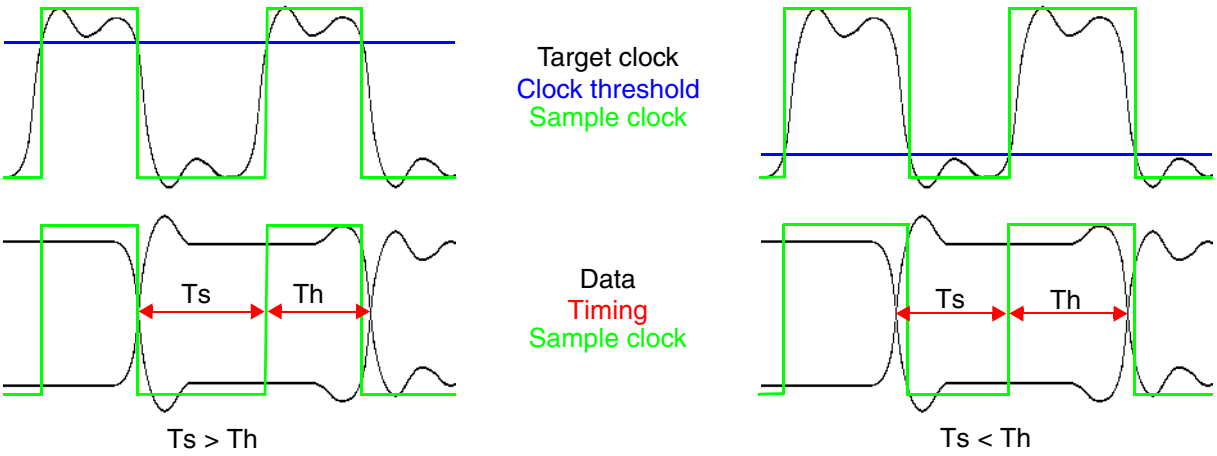


Fig.B

The threshold can be set manually using the command **Analyzer.ThresHold <value>** or automatically with the command **Analyzer.ThresHold Clock**. The algorithm attempts to find a threshold level close to a duty cycle of 50/50 as seen in the figure B.

Important to know: The clock threshold level influences the data setup (Ts) and hold (Th) times in dependency of the clock waveform:

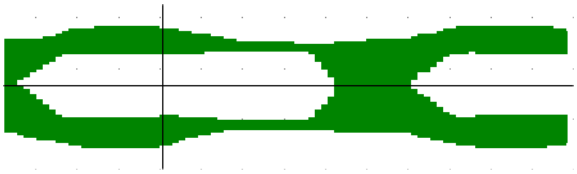


This is more important to care of as higher the trace port data rates are.

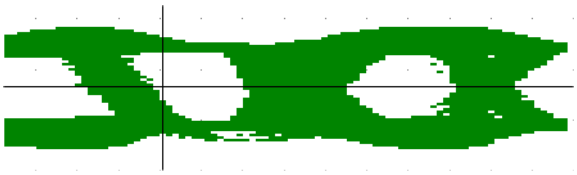
	Even if it is possible to change the data setup&hold times by the clock threshold level, this is not the right way to compensate data or clock skew. Anyway, sometimes it is the last way to get a suitable trace listing for non-Autofocus preprocessors.

With the given background it is easy to understand, why data eyes change with the clock threshold level. The following scans where made with different clock threshold levels. The given graphs show the same data signal.

Clock threshold at 0.8V

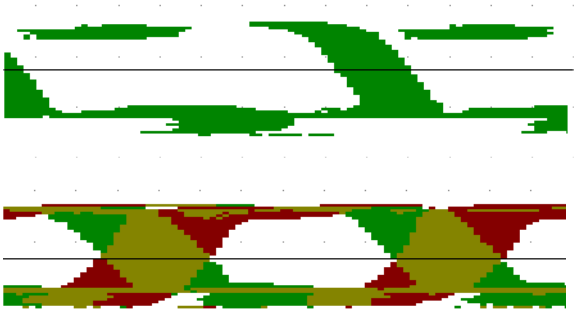


Clock threshold at 1.7V
This setup would give suitable trace results.



Clock threshold at 2.6V

The number of different data eye scans lead us to the clock eye window (**Analyzer.ShowFocusClockEye**) which may look like the following:

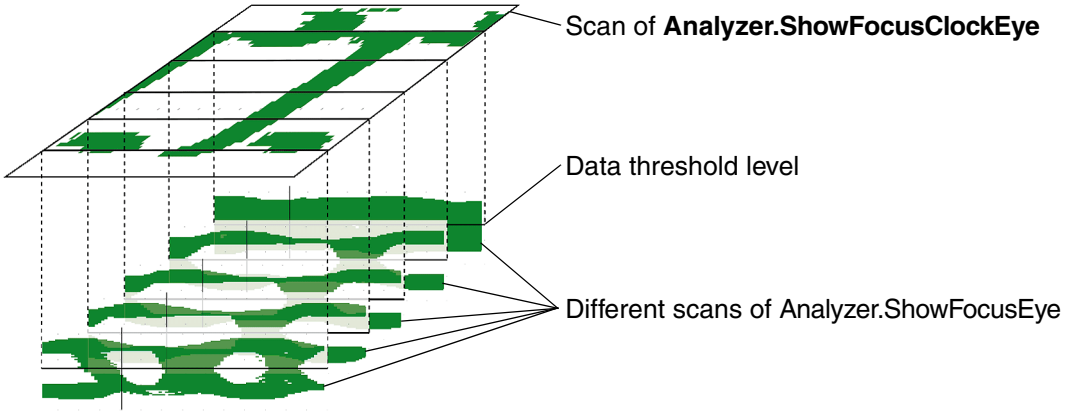


Sampling on rising clock edges
(e.g. ETMv1 normal mode)

Sampling on both clock edges
(e.g. ETMv3)

Green color means sampling on rising clock edge, Red color means sampling on falling clock edge.

The correlation between **Analyzer.ShowFocusEye** and **Analyzer.ShowFocusClockEye** is shown with the following 3D-figure:



On the lower part of the figure different data eye scans are cascaded. The data threshold level and the sampling point are marked by black lines. The data threshold line also define the level where the clock eye picture would be normally placed.
Basically, the clock eye scan shows a third dimension of the data eye scan..

	The scan of **Analyzer.ShowFocusEye** keeps the clock threshold constant during the data threshold is changed. The scan of **Analyzer.ShowFocusClockEye** keeps the data threshold constant during the clock threshold is changed.

If you can not detect any problem on your side, please send the following information to support@lauterbach.com:

- used start-up/configuration scripts
- the complete text of the error messages you get (**AREA.view**)
- a screenshot of **VERSION.SOFTWARE**
- a screenshot of **VERSION.HARDWARE**
- a screenshot of **VERSION.HARDWARE2**
- a screenshot of the **ETM** window
- JTAG voltage level
- JTAG frequency
- ETM voltage level
- cpu core frequency
- ETM connector pinout (board schematics and layout)
- Is the problem frequency dependent?
- Does another trace hardware behave different?

For ARM-ETM with AUTOFOCUS also do the following:
Please execute **Trace.AutoFocus** once more and add the following information to your request:

- a screenshot of **Trace.ShowFocus** (press SCAN before)
- a screenshot of **Trace.ShowFocusEye** (press SCAN before)
- a screenshot of **Trace.ShowFocusClockEye** (set threshold to VCC and press SCAN)
- the log file generated with Support.cmm (menu Help/About/Support) script

- Place the trace MICTOR connector close to the target processor. Make sure that the MICTOR ground Pins are connected to your target's ground plane.

  AMP part numbers for MICTOR connectors (designed for 50 Ω systems) can be found on:

  www.lauterbach.com/adetmmictor.html

  For information on MICTOR connectors please refer to: **www.amp.com**

  Additional information on MICTOR connectors and the MICTOR flex extension can be obtained by contacting support@lauterbach.com.

- Consider setup and hold time requirements. Especially for Preprocessors without AutoFocus it is important to meet their **timing requirements**.

- Match your traces to reduce channel to channel skew: same length, same PCB layer and same amount of vias for all traces. Avoid stubs.

  Ideally traces should have a 50 Ω impedance

- Avoid parallel routing of the trace bus. The closer the tracks the more cross talk will influence the signal integrity. Special routing (meandering, e.g. used for length equalization) should be used to prevent large areas of parallel track placment.
  Even at higher data rates this is an important point.

- Sufficient bypass capacitors are crucial to keep the supply voltage stable when the trace port is driven by your application. This is esp. important for portsizes greater 8 bit at high frequencies. If your supply voltages are not stable, the target processor might assume illegal JTAG tap states and the TPA might loose control over the target (typically this might result in messages such as "emulation debug port fail").

  If you have to safe cost, just allocate additional footprints for bypass capacitors in your PCB layout. That way you can mount additional bypass capacitors for the development PCBs that have to drive TPAs and omitt them for the production PCBs.

- Use Power and ground planes:

  Capacitor vias should never be shared. Each capacitor requires at least 2 vias: one for connecting to ground and one for its vcc connection. Vias should descend directly to the power and ground planes (traces should not be used to connect bypass capacitors to the power pins they are servicing).

- A series termination on the target is usually not required. All trace channels are terminated on the Preprocessor to 50% of the voltage level of pin12 of the trace mictor.

  If desirable, footprints for a series termination on the target can be implemented and mounted with O Ω. If necessary, sophisticated users can implement a series-parallel combination. Contact support@lauterbach.com for ECDs of the Preprocessor's termination.

- Pin 12 (VTREF) of the trace port mictor has to be on the same voltage level than the amplitude of the trace port's clock and data channels. Pin 14 has to be on the same voltage level than the amplitude of the JTAG signals (usually the same as Pin 12). Again: the termination voltage of the TPA will be 50% of VTREF.

- The target voltage has to be within the **specified range** specified range. For other voltage levels contact support@lauterbach.com.

  NOTE: for very high data rates (> 300 Mbit/s) 1.8 V signals are usually easier to support by the target than higher voltage levels.

- Recommendations for output drivers:

  If possible keep output drive strength and slew rate programmable (e.g. 8, 12, 16 mA for both low and high slew).

  NOTE: For very high data rates (> 300 Mbit/s) 12-16mA drivers should be used to get sharp edges and sufficient data eye opening .

If you cannot support high drive strength, at least make an investment in the clock signal. Usually the clock signal is the bottle neck, so the clock signal should have at least 6 mA, better 8 … 12 mA, drive strength. Also you should use HalfRate, whenever possible to divide the clock frequency by two.

Keep in mind that the drive strength of the output buffers has to be supported by proper PCB layout and sufficient bypass capacitors for all frequencies ranges!

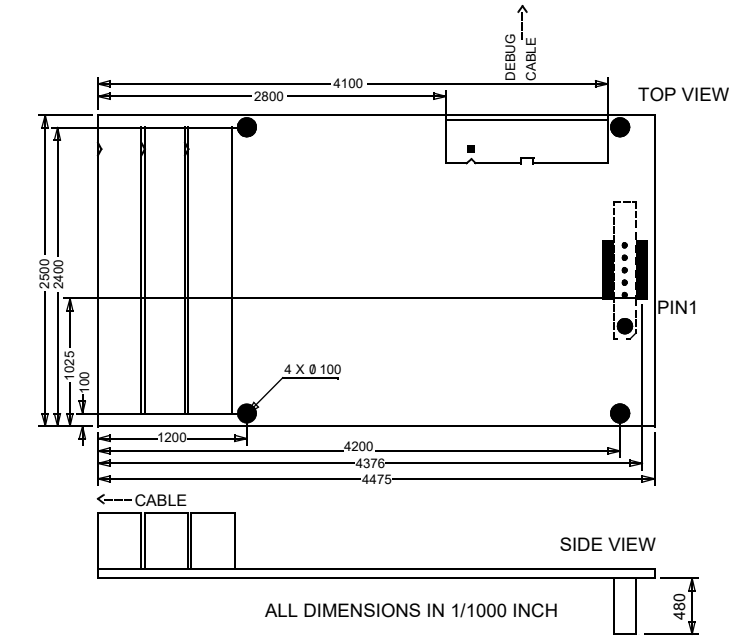
Your trace port supply voltage should not show any significant ripple, even if signals are traced by the TPA (contact support@lauterbach.com for ECDs of the TPA termination).

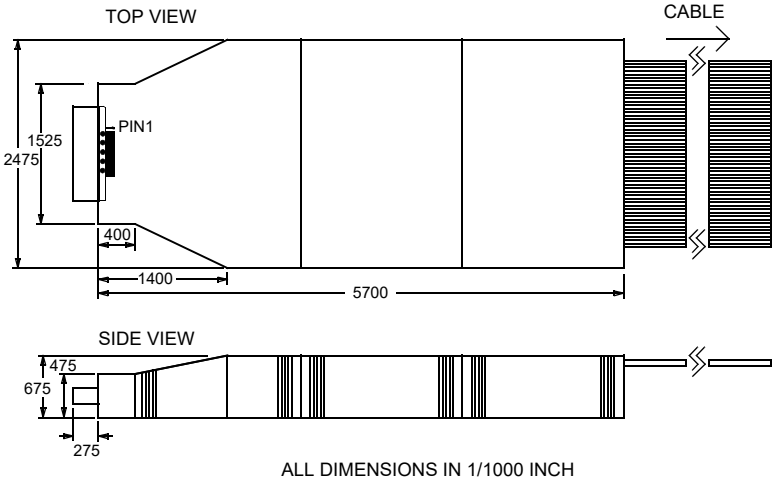
Operation Voltage

Adapter	OrderNo	Voltage Range
Preprocessor for ARM-ETM 120	LA-7889	2.5 .. 3.6 V
Preprocessor for ETM 2-MICTOR 2 flex cables	LA-7923	2.5 .. 3.6 V
Preprocessor for ARM-ETM with AUTOFOCUS Flex	LA-7991	1.8 .. 3.3 V
Preproc. for ARM-ETM/AUTOFOCUS II 600 Flex	LA-7992	1.2 .. 3.3 V
Preproc. for ARM-ETM/AUTOFOCUS 600 MIPI	LA-7993	1.2 .. 3.3 V

Dimension

LA-7921 PP-ARM-ETM/200

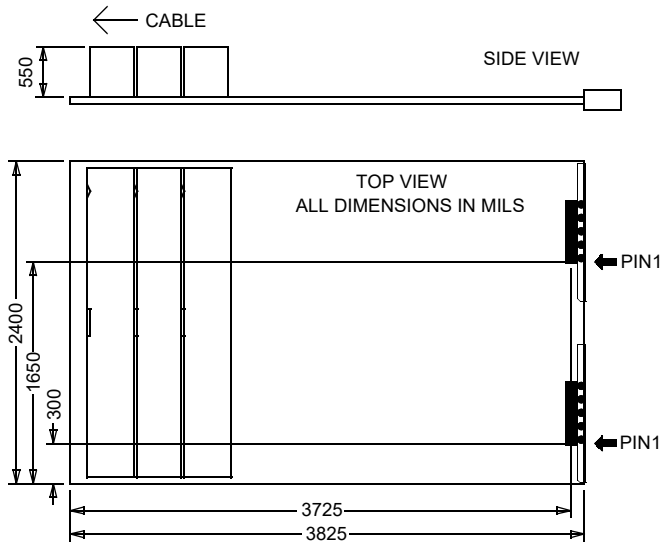




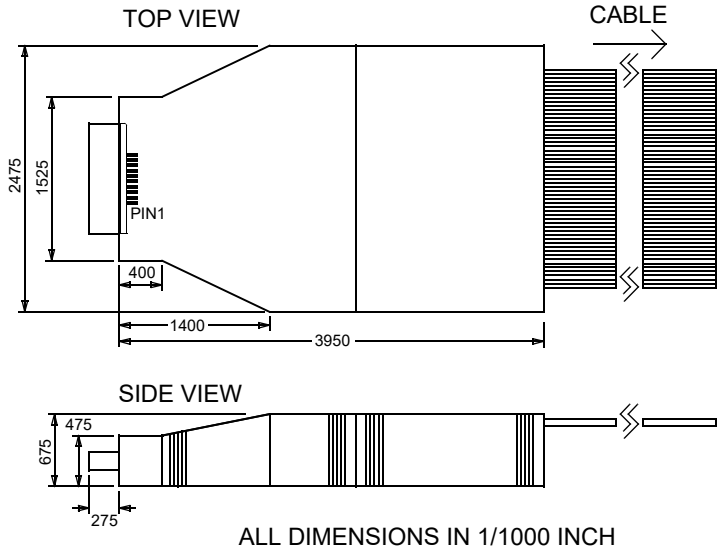
new case, delivery from december 2003

Dimension

LA-7923 PP-ARM-ETM-TWO

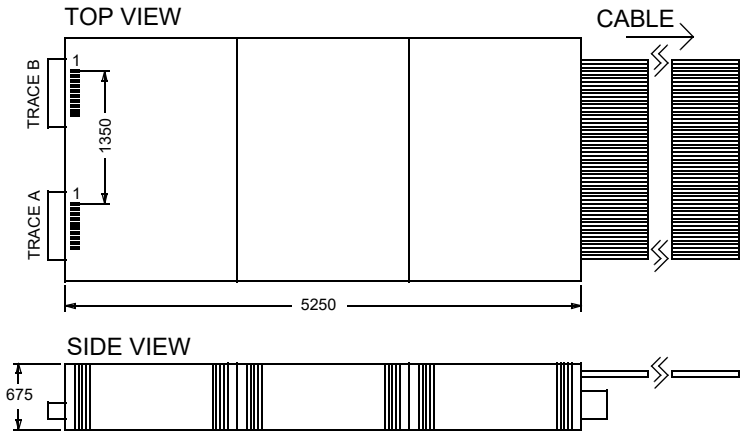


LA-7990 PP-ARM-ETM/270



Dimension

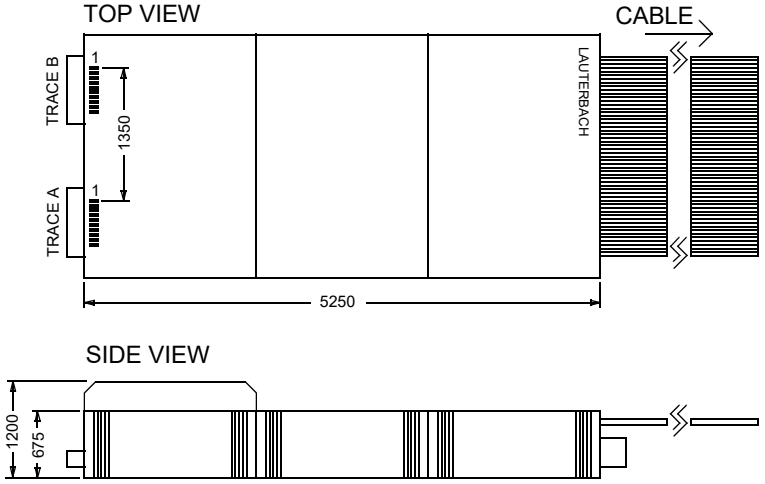
LA-7991 PP-ARM-ETM-AF



ALL DIMENSIONS IN 1/1000 INCH

Note: TRACE B is only used for Demux 2 (ARM7-10) or PortSize >16 (ARM11)

LA-7992 PP-ARM-ETM-AF-2

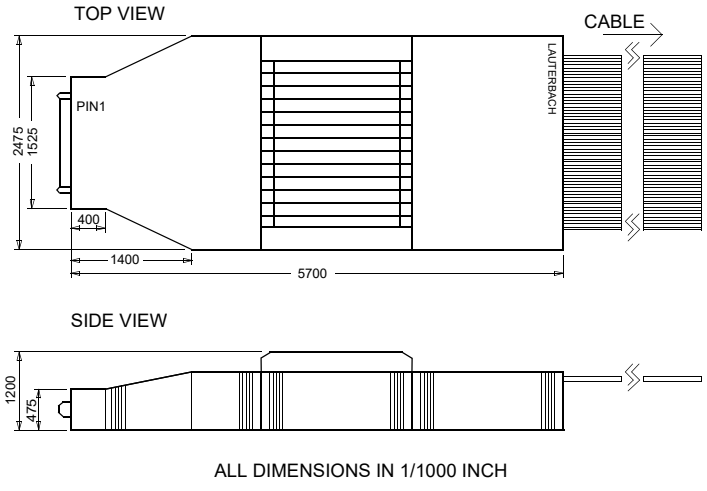


ALL DIMENSIONS IN 1/1000 INCH

Note: TRACE B is only used for Demux 2 (ARM7-10) or PortSize >16 (ARM11)

Dimension

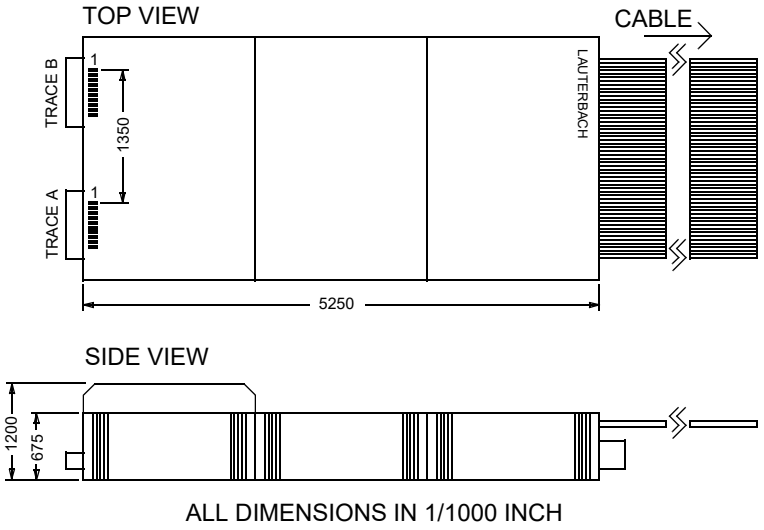
LA-7993 PP-ARM-ETM-AF-MIPI



ALL DIMENSIONS IN 1/1000 INCH

Dimension

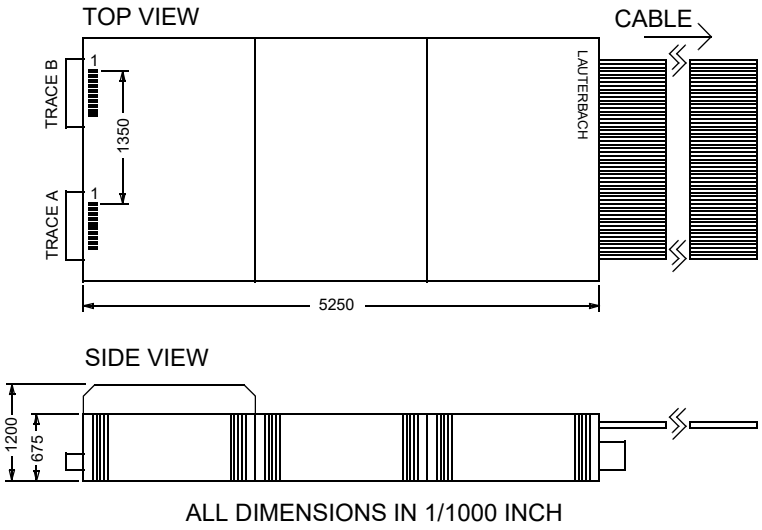
LA-7995 PP-C5500-AF-2
LA-3300 TRACE-RISCV-AFII-P
LA-3308 TRACE-ARM-ETM-AF-2
LA-3917 PP-NIOS-AF-2
LA-3918 PP-RH850-AFII
LA-3920 PP-STRED-AF-2
LA-3921 PP-XTENSA-AF-2
LA-3927 PP-RISCV-AF-2



ALL DIMENSIONS IN 1/1000 INCH

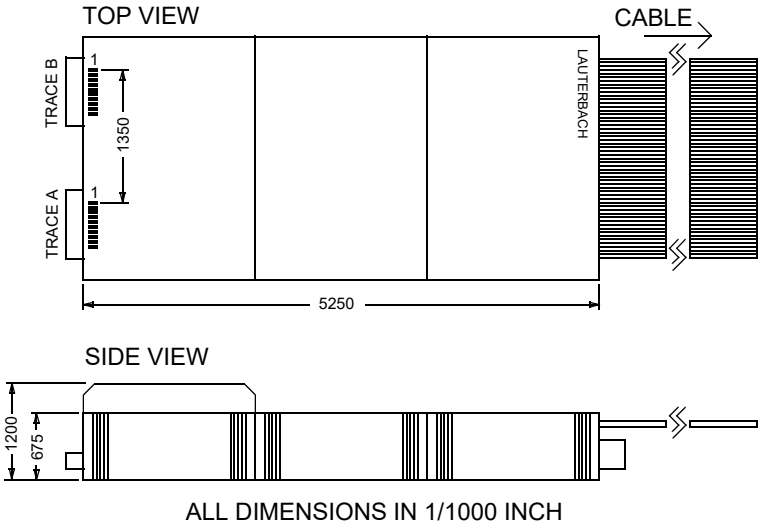
Dimension

LA-7996 PP-CEVA-AF-2



ALL DIMENSIONS IN 1/1000 INCH

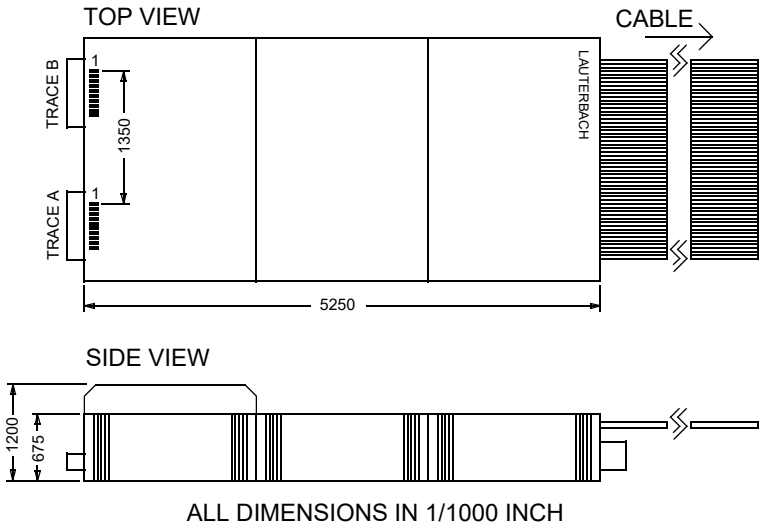
LA-7998 PP-HEXAGON-AF-2



ALL DIMENSIONS IN 1/1000 INCH

Dimension

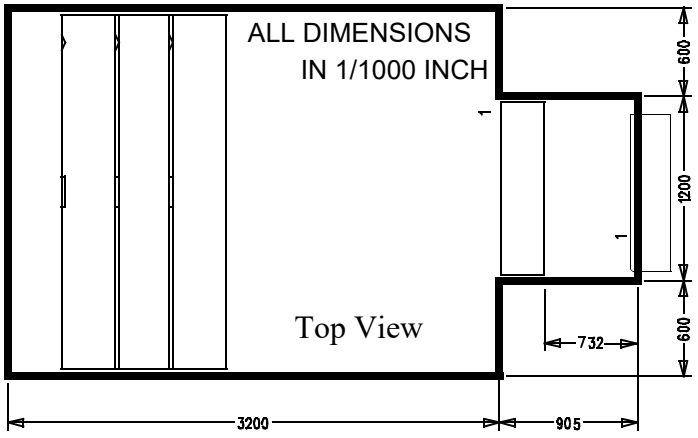
LA-7999 PP-STARCORE-AF-2

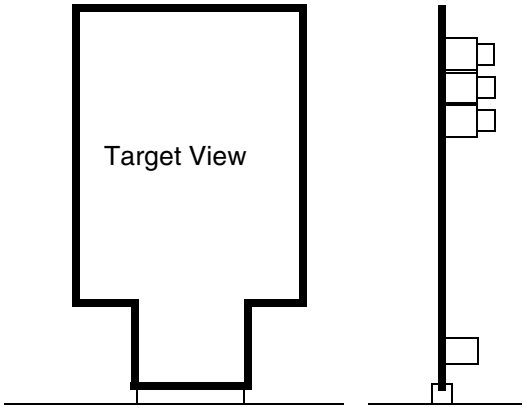


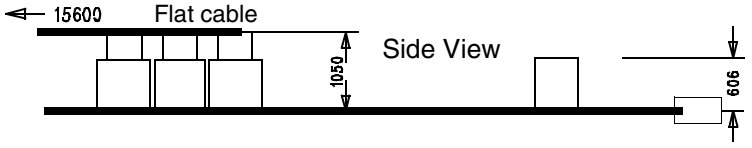
ALL DIMENSIONS IN 1/1000 INCH

Dimension

LA-7889 PP-ARM-ETM/120

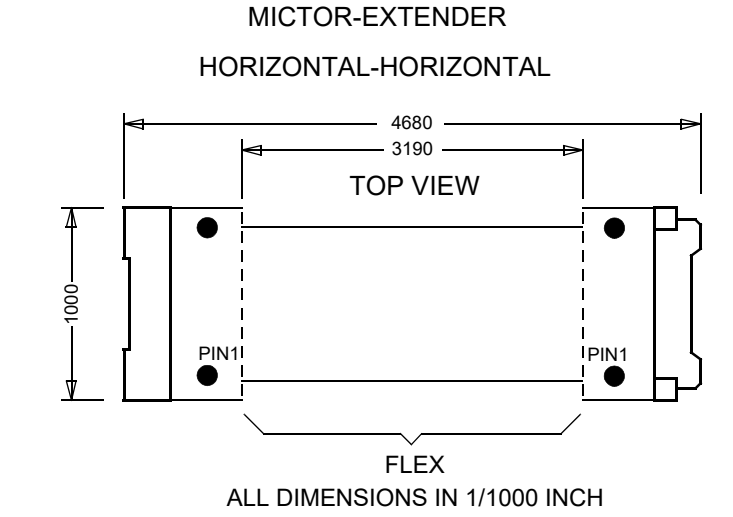


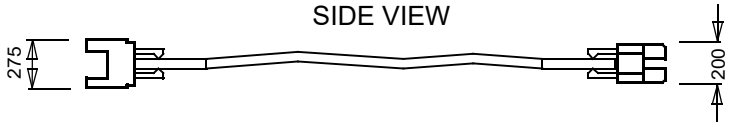




Dimension

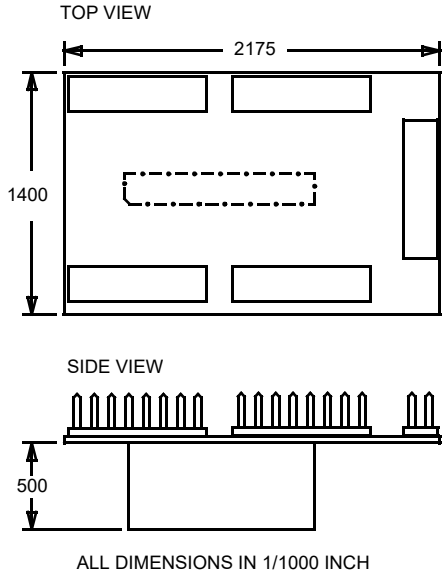
LA-1370 MICTOR-FLEXEXT





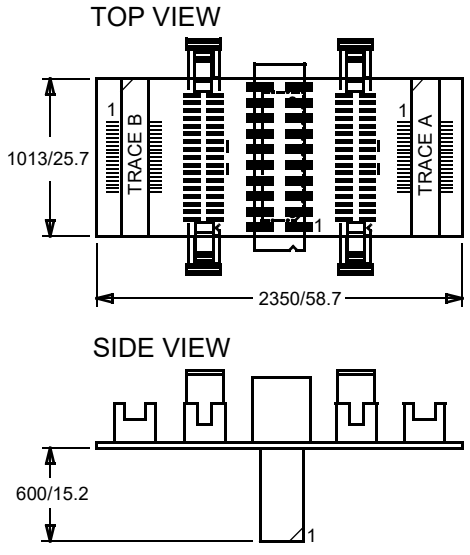
Dimension

LA-7649 CONV-MIC38-2.54MM



ALL DIMENSIONS IN 1/1000 INCH

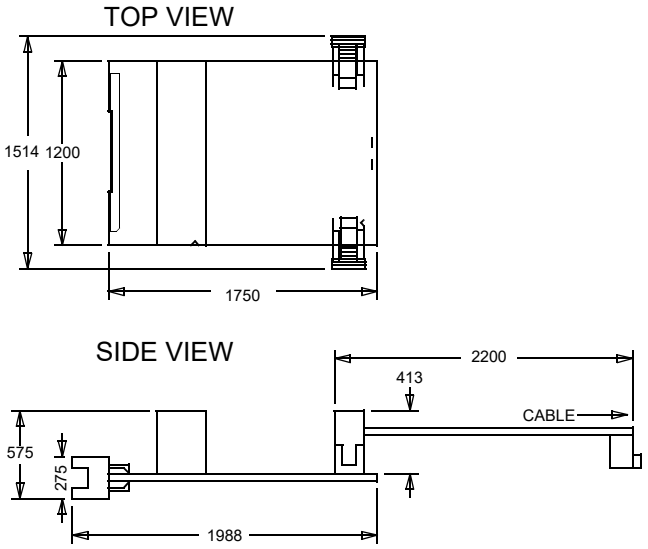
LA-3808 CONV-L8540-MIPI



ALL DIMENSIONS IN 1/1000 INCH / mm

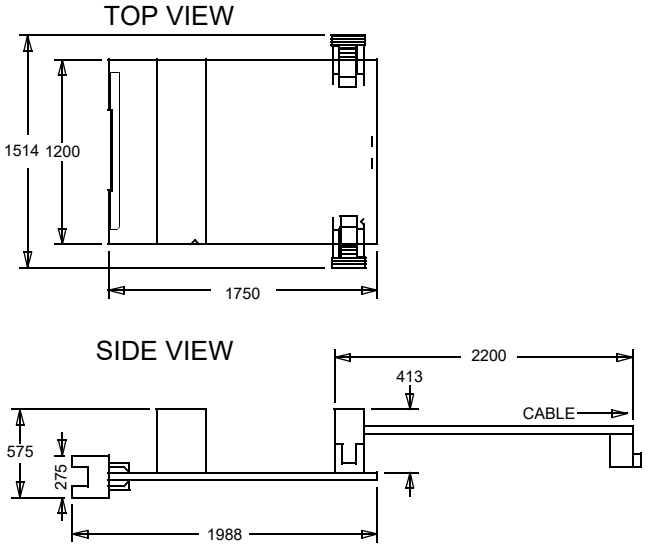
Dimension

LA-3809 CONV-IDC20A-MIC/MIPI



ALL DIMENSIONS IN 1/1000 INCH

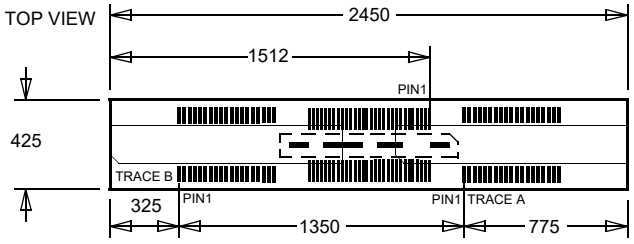
LA-3842 CONV-IDC20A-MIPI34

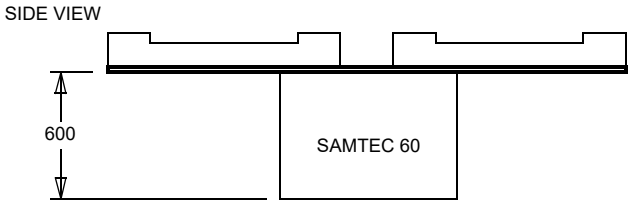


ALL DIMENSIONS IN 1/1000 INCH

Dimension

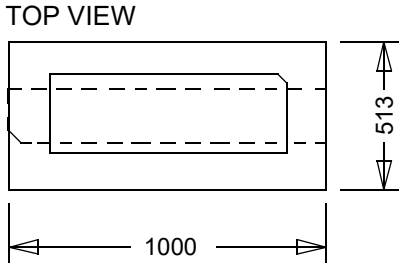
LA-3816 CON-2XMICTOR-MIPI60

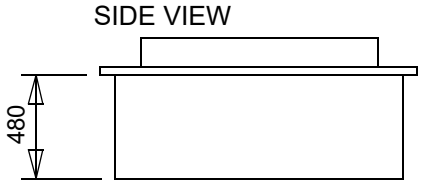




ALL DIMENSIONS IN 1/1000 INCH

LA-3817 CON-MIPI60-MICTOR38

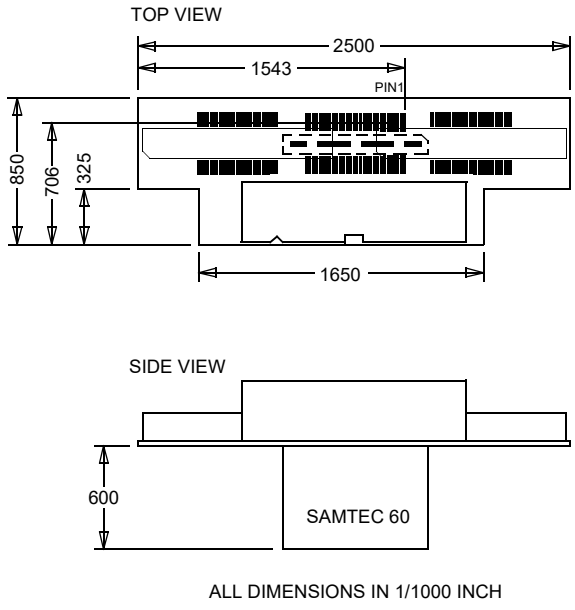




ALL DIMENSIONS IN 1/1000 INCH

Dimension

LA-3818 CONV-AFMIC38-MIPI60



Adapters

Not necessary.

Signal Description

ETMv1/2 signals

Signal	Description	Direction from target	Compliance
VTREF	”Voltage Reference” is the target reference voltage. It indicates if the target power is applied, it is used to create the logic-level reference (VTREF/2) for the trace tool input comparators and it auto adjusts the termination voltage level. It shall be directly connected to the power supply of the processors IO pins. Decoupling capacitors are optional (100nF). If the signal is not connected then trace recording will fail in most cases.	output	required
EXTTRIG	The function of “External Trigger” is not exactly defined by the ARM-ETM specification. It can be input or output. TRACE32 can use this signal to trigger (high-active) the trace hardware. This signal is pulled down via a >10kOhm resistor on tool side. It is recommended to pull-down the signal via 10kOhm if not used. This allows easy access in future if necessary. Alternatively connect to GND or let it not connected. Tracing is not limited if the signal is not available.	input/ output	not used/ optional
TRACECLK	“Trace Clock” is used as sampling clock for all trace signals. If the signal is not connected then no trace recording will be possible.	output	required
TRACEPKT TRACEPKTA TRACEPKTB	“Trace Packets” transmit main trace information. Unused pins must be directly connected to GND. If the signals is are connected then no trace recording will be possible.	output	required

Signal	Description	Direction from target	Compliance
TRACESYNC	“Trace Sync” marks trace packets as valid or invalid. If the signal is not connected then no trace recording will be possible. If the signal is connected to GND no trace packet filtering will happen. Tracing is basically possible, but trace decoding needs longer and could report errors due to too many idle patterns.	output	required
PIPESTAT	“Pipeline Status” transmits the CPU pipeline information like “instruction executed”. If the signal is not connected then no trace recording will be possible.	output	required
GND	”Ground”. All pins (including the metal bar in the middle of the connector) must be connected to minimize noise pickup. If no sufficient ground connection is given then no trace recording will be possible due to instable data eyes.		required

Signal	Description	Direction from target	Compliance
VREF-TRACE	”Voltage Reference” is the target reference voltage. It indicates if the target power is applied, it is used to create the logic-level reference (VTREF/2) for the trace tool input comparators and it auto adjusts the termination voltage level. It shall be directly connected to the power supply of the processors IO pins.	output	required
VCC	VCC on pin 34 is handled as trace signal. It must have the same level as VREF-TRACE (pin 12). The signal will be terminated like the other trace signals (e.g. TRACECLK). It must be directly connected to VTREF-TRACE. If the signal is not connected then some trace tools will fail to record the trace data correctly.	output	required
EXTTRIG	The function of “External Trigger” is not exactly defined by the ARM-ETM specification. It can be input or output. TRACE32 can use this signal to trigger (high-active) the trace hardware. This signal is pulled down via a >10kOhm resistor on tool side. It is recommended to pull-down the signal via 10kOhm if not used. This allows easy access in future if necessary. Alternatively connect to GND or let it not connected. Tracing is not limited if the signal is not available.	input/ output	not used/ optional
TRACECLK	“Trace Clock” is used as sampling clock for all trace signals. If the signal is not connected then no trace recording will be possible.	output	required
TRACEDATA	“Trace Data” transmit main trace information. Unused pins must be directly connected to GND. If the signals are not connected then no trace recording will be possible.	output	required

Signal	Description	Direction from target	Compliance
TRACECTL	“Trace Control” marks trace packets as valid or invalid. Continuous-mode operation: The signal is not required/available. The pin should be connected directly to GND. Other modes: If the signal is not connected then no trace recording will be possible. If the signal is connected to GND then no trace packet filtering will happen. Tracing is basically possible, but trace decoding needs longer and could report errors due to too many idle patterns.	output	optional required
GND	”Ground”. All pins must be connected to minimize noise pickup.		required

Signal	Description	Direction from target	Compliance
VREF-DEBUG	¨Voltage Reference¨ of the debug signals. It indicates if the target power is applied, it is used to create the logic-level reference (VTREF/2) for the debug signal input comparators and it auto-adjusts the voltage levels of the debug signal output driver. It shall be directly connected to the debug signal reference voltage. Furthermore, it might have a series resistor, although this is not recommended. It has to be strong enough to overdrive the 100 kOhms pull-down resistor of the debug cable.	output	required
TDI	¨Test Data In¨ is the data signal from debugger to processor. You can place a pull-up or pull-down resistor (1 kOhms - 47 kOhms) on this line to ensure a defined state even when the line is not driven by the debugger.	input	required
TDO	¨Test Data Out¨ is the data signal from processor to debugger. You can place a 33 Ohms series resistor close to the processor for series termination. You can place a pull-up or pull-down resistor (1 kOhms - 47 kOhms) on this line.	output	required
TMS	¨Test Mode Select¨ is the control signal for the TAP controller. You can place a pull-up or pull-down resistor (1 kOhms - 47 kOhms) on this line in order to give it a defined state even when the line is not driven by the debugger.	input	required
TCK	¨Test Clock¨ is the clock signal from debugger to processor. You should place a pull-up or pull-down resistor (1 kOhms - 47 kOhms) on this line in order to give it a defined state even when the line is not driven by the debugger.	input	required
RTCK	”Return Test Clock” can be used to synchronize the JTAG signals to internal clocks. For CoreSight/Cortex systems it is not needed and in this case we recommend not to connect it or to connect it to GND. Do not directly connect it to TCK on the target. This causes a stub on TCK and reflections and often causes the JTAG communication to fail.	output	optional, not recommend ed

Signal	Description	Direction from target	Compliance
TRST-	¨Test Reset¨ (low active) is used for an asynchronous reset of the JTAG Test Access Port (TAP). It resets the TAP state machine. The debugger drives it by a push-pull driver. From the debugger point of view it is optional, because it resets the TAP also by a certain JTAG sequence. Leave it open if the target does not have this signal. You should place a pull-down resistor (1 kOhms - 47 kOhms) on this signal on target side, although this is not JTAG conform. It ensures the on-chip debug logic is inactive when the debugger is not connected.	input	optional, connect if available
RESET-	“System Reset¨ (low active) is used to reset the target system. The signal is also used by the debugger to detect if the processor is held in reset. There is no need to provide this indication, but if a reset condition is not signalized by this line it should be high (= no reset). The debugger drives it open-drain. A 47 kOhms pull-up is within the debug cable. There might be the need to place a pull-up (1 kOhms - 47 kOhms) on target side to avoid unintentional resets when the debugger is not connected and probably to strengthen the weak 47 kOhms pull-up in the debug cable. If the signal is not available, leave it open or place a pull-up (1 kOhms - 47 kOhms).	input	optional, recommend ed
DBGACK	”Debug Acknowledge” (high active) is an input of the debugger to sense the processors halt status. This signal is typically not used anymore on CoreSight/Cortex systems. We recommend not to connect it or to connect it to GND.	output	optional, not recommend ed
DBGRQ	¨Debug Request¨ (high active) is an output of the debugger to cause the processor to enter debug mode (to halt the processor). This signal is typically not used anymore on CoreSight/Cortex systems. We recommend not to connect it. If this signal is provided by the processor you should place a pull-down resistor (1 kOhms - 47 kOhms) on target side for the case the debugger is not connected.	input	optional, not recommend ed

The pinout of the Mictor connector optionally includes the debug signals which needs to go to the Debug Cable (20-pin standard header). The Preprocessors have a 20-pin connector where the user can plug in the Debug Cable. Alternatively the Debug Cable can be plugged on a separate 20-pin connector on the target. In this case there is no need to connect the debug signals to the Mictor. For better signal quality it is even better not to connect them. The debug signals are the signals listed in the table above: VREF-DEBUG, TDI, TDO, TMS, TCK, RTCK, TRST-, RESET-, DBGACK, DBGRQ. Just leave them all not connected.

ETMv1/2

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
N/C	5	6	TRACECLK
DBGRQ	7	8	DBGACK
RESET-	9	10	EXTRIG
TDO	11	12	VREF-TRACE
RTCK	13	14	VREF-DEBUG
TCK	15	16	TRACEPKT7
TMS	17	18	TRACEPKT6
TDI	19	20	TRACEPKT5
TRST-	21	22	TRACEPKT4
TRACEPKT15	23	24	TRACEPKT3
TRACEPKT14	25	26	TRACEPKT2
TRACEPKT13	27	28	TRACEPKT1
TRACEPKT12	29	30	TRACEPKT0
TRACEPKT11	31	32	TRACESYNC
TRACEPKT10	33	34	PIPESTAT2
TRACEPKT9	35	36	PIPESTAT1
TRACEPKT8	37	38	PIPESTAT0

ETMv1/2 with Multiplexed Mode

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
N/C	5	6	TRACECLK
DBGRQ	7	8	DBGACK
SRST-	9	10	EXTRIG
TDO	11	12	VTREF
RTCK	13	14	VCC
TCK	15	16	N/C
TMS	17	18	N/C
TDI	19	20	TRACEPKT1415
TRST-	21	22	TRACEPKT1213
N/C	23	24	TRACEPKT1011
N/C	25	26	TRACEPKT0809
N/C	27	28	TRACEPKT0607
N/C	29	30	TRACEPKT0405
N/C	31	32	TRACEPKT0003
N/C	33	34	PS02TPKT02
N/C	35	36	PS01TPKT01
N/C	37	38	PS00TSYNC

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
N/C	5	6	TRACECLKA
DBGRQ	7	8	DBGACK
SRST-	9	10	EXTRIG
TDO	11	12	VTREF
RTCK	13	14	VCC
TCK	15	16	N/C
TMS	17	18	N/C
TDI	19	20	N/C
TRST-	21	22	N/C
TRACEPKTB3	23	24	TRACEPKTA3
TRACEPKTB2	25	26	TRACEPKTA2
TRACEPKTB1	27	28	TRACEPKTA1
TRACEPKTB0	29	30	TRACEPKTA0
TRACESYNCB	31	32	TRACESYNCA
PIPESTATB2	33	34	PIPESTATA2
PIPESTATB1	35	36	PIPESTATA1
PIPESTATB0	37	38	PIPESTATA0

Connector 1:

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
N/C	5	6	TRACECLK
DBGRQ	7	8	DBGACK
SRST-	9	10	EXTRIG
TDO	11	12	VTREF
RTCK	13	14	VCC
TCK	15	16	TRACEPKTA7
TMS	17	18	TRACEPKTA6
TDI	19	20	TRACEPKTA5
TRST-	21	22	TRACEPKTA4
TRACEPKTA15	23	24	TRACEPKTA3
TRACEPKTA14	25	26	TRACEPKTA2
TRACEPKTA13	27	28	TRACEPKTA1
TRACEPKTA12	29	30	TRACEPKTA0
TRACEPKTA11	31	32	TRACESYNCA
TRACEPKTA10	33	34	PIPESTATA2
TRACEPKTA9	35	36	PIPESTATA1
TRACEPKTA8	37	38	PIPESTATA0

Connector 2:

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
N/C	5	6	N/C
N/C	7	8	N/C
N/C	9	10	N/C
N/C	11	12	N/C
N/C	13	14	N/C
N/C	15	16	TRACEPKTB7
N/C	17	18	TRACEPKTB6
N/C	19	20	TRACEPKTB5
N/C	21	22	TRACEPKTB4
TRACEPKTB15	23	24	TRACEPKTB3
TRACEPKTB14	25	26	TRACEPKTB2
TRACEPKTB13	27	28	TRACEPKTB1
TRACEPKTB12	29	30	TRACEPKTB0
TRACEPKTB11	31	32	TRACESYNCB
TRACEPKTB10	33	34	PIPESTATB2
TRACEPKTB9	35	36	PIPESTATB1
TRACEPKTB8	37	38	PIPESTATB0

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
TRACECLKB	5	6	TRACECLKA
DBGRQ	7	8	DBGACK
SRST-	9	10	EXTRIG
TDO	11	12	VTREF
RTCK	13	14	VCC
TCK	15	16	ATRACEPKT7
TMS	17	18	ATRACEPKT6
TDI	19	20	ATRACEPKT5
TRST-	21	22	ATRACEPKT4
BTRACEPKT3	23	24	ATRACEPKT3
BTRACEPKT2	25	26	ATRACEPKT2
BTRACEPKT1	27	28	ATRACEPKT1
BTRACEPKT0	29	30	ATRACEPKT0
BTRACESYNC	31	32	ATRACESYNC
BPIPESTAT2	33	34	APIPESTAT2
BPIPESTAT1	35	36	APIPESTAT1
BPIPESTAT0	37	38	APIPESTAT0

Connector 1:

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
GND	5	6	TRACECLK
DBGRQ	7	8	DBGACK
RESET-	9	10	EXTRIG
TDOI-ISWO	11	12	VREF-TRACE
RTCK	13	14	VREF-DEBUG
TCK|TCKC|SWCLK	15	16	TRACEDATA[7]
TMS|TMSC|SWDIO	17	18	TRACEDATA[6]
TDI	19	20	TRACEDATA[5]
TRST-	21	22	TRACEDATA[4]
TRACEDATA[15]	23	24	TRACEDATA[3]
TRACEDATA[14]	25	26	TRACEDATA[2]
TRACEDATA[13]	27	28	TRACEDATA[1]
TRACEDATA[12]	29	30	GND
TRACEDATA[11]	31	32	GND
TRACEDATA[10]	33	34	VCC
TRACEDATA[9]	35	36	TRACECTL
TRACEDATA[8]	37	38	TRACEDATA[0]

Connector 2:

Signal	Pin	Pin	Signal
N/C	1	2	N/C
N/C	3	4	N/C
GND	5	6	N/C
N/C	7	8	N/C
N/C	9	10	N/C
N/C	11	12	N/C
N/C	13	14	N/C
N/C	15	16	TRACEDATA[23]
N/C	17	18	TRACEDATA[22]
N/C	19	20	TRACEDATA[21]
N/C	21	22	TRACEDATA[20]
TRACEDATA[31]	23	24	TRACEDATA[19]
TRACEDATA[30]	25	26	TRACEDATA[18]
TRACEDATA[29]	27	28	TRACEDATA[17]
TRACEDATA[28]	29	30	GND
TRACEDATA[27]	31	32	GND
TRACEDATA[26]	33	34	VCC
TRACEDATA[25]	35	36	GND
TRACEDATA[24]	37	38	TRACEDATA[16]

MIPI-60

ETMv3 / ETMv4 / PFTv1

Signal	Pin	Pin	Signal
VREF-DEBUG	1	2	TMS|TMSC|SWDIO
TCK|TCKC|SWCLK	3	4	TDO|-|SWO
TDI	5	6	RESET-
RTCK	7	8	TRST- PULLDOWN
TRST-	9	10	DBGRQ TRIGIN
DBGACK TRIGOUT	11	12	VREF-TRACE
TRACECLK	13	14	GND
GND	15	16	GND
TRACECTL	17	18	TRACEDATA[19]
TRACEDATA[0]	19	20	TRACEDATA[20]
TRACEDATA[1]	21	22	TRACEDATA[21]
TRACEDATA[2]	23	24	TRACEDATA[22]
TRACEDATA[3]	25	26	TRACEDATA[23]
TRACEDATA[4]	27	28	TRACEDATA[24]
TRACEDATA[5]	29	30	TRACEDATA[25]
TRACEDATA[6]	31	32	TRACEDATA[26]
TRACEDATA[7]	33	34	TRACEDATA[27]
TRACEDATA[8]	35	36	TRACEDATA[28]
TRACEDATA[9]	37	38	TRACEDATA[29]
TRACEDATA[10]	39	40	TRACEDATA[30]
TRACEDATA[11]	41	42	TRACEDATA[31]
TRACEDATA[12]	43	44	GND
TRACEDATA[13]	45	46	GND
TRACEDATA[14]	47	48	GND
TRACEDATA[15]	49	50	GND
TRACEDATA[16]	51	52	GND
TRACEDATA[17]	53	54	GND
TRACEDATA[18]	55	56	GND
GND	57	58	GND
GND	59	60	GND

Signal	Pin	Pin	Signal
VREF DEBUG	1	2	TMS
TCK	3	4	TDO
TDI	5	6	RESET-
RTCK	7	8	TRST- PULLDOWN
TRST-	9	10	DBGRQ TRIGIN
DBGACK TRIGOUT	11	12	VREF TRACE
TRACECLK	13	14	GND
GND	15	16	GND
PIPESTAT[0]	17	18	GND
PIPESTAT[1]	19	20	GND
PIPESTAT[2]	21	22	GND
TRACESYNC	23	24	GND
TRACEPKT[0]	25	26	GND
TRACEPKT[1]	27	28	GND
TRACEPKT[2]	29	30	GND
TRACEPKT[3]	31	32	GND
TRACEPKT[4]	33	34	GND
TRACEPKT[5]	35	36	GND
TRACEPKT[6]	37	38	GND
TRACEPKT[7]	39	40	GND
TRACEPKT[8]	41	42	GND
TRACEPKT[9]	43	44	GND
TRACEPKT[10]	45	46	GND
TRACEPKT[11]	47	48	GND
TRACEPKT[12]	49	50	GND
TRACEPKT[13]	51	52	GND
TRACEPKT[14]	53	54	GND
TRACEPKT[15]	55	56	GND
GND	57	58	GND
GND	59	60	GND