LAUTERBACH
DEVELOPMENT TOOLS

General Commands Reference Guide E

MANUAL

General Commands Reference Guide E

General Commands Reference Guide E

Version 06-Jun-2024

ELA

Embedded logic analyzer (ELA)

ARM

See also

- ELA.ATBTrigger
- ELA.CLEAR
- ELA.CLOCK
- ELA.OFF
- ELA.ON
- ELA.PortRoute
- ELA.Register
- ELA.RESet
- ELA.SELect
- ELA.Set
- ELA.state
- ELA.SyncPeriod
- ELA.TimeStampCLOCK
- ELA.TimeStamps
- ELA.TimeStampThreshold
- ELA.Trace
- ELA.TraceID
- ELA.TracePREDICT
- ELA.TracePriority
- SYStem.CONFIG.ELA
- ELABASE()

Overview ELA

ARM

The Embedded Logic Analyzer (ELA) is an ARM CoreSight component which chip designers can add on their devices to provide visibility to on-chip signals.

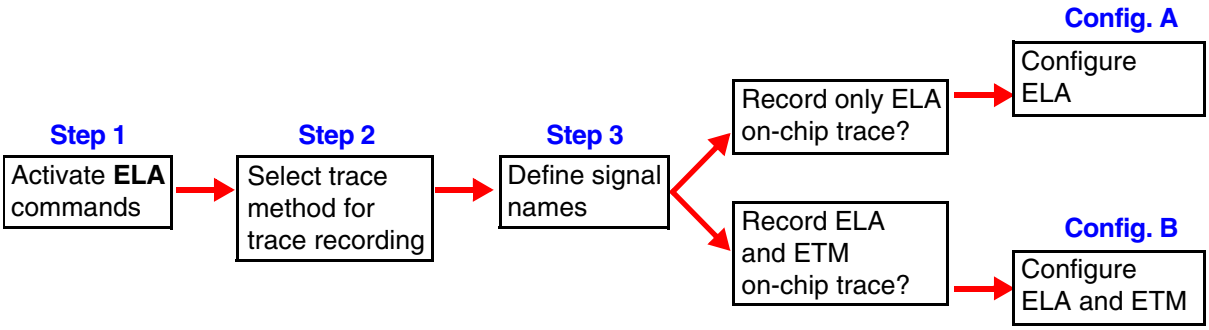
Today's IP/SoC complexity increases and requires more insight into the system. Especially for use cases like the following ones, the Embedded Logic Analyzer (ELA) might be the last chance to track down an issue:

- Processor dead locks
- An unsupported access hangs the system
- Suspected processor bugs

In addition, you could use the Embedded Logic Analyzer (ELA) to see processor load/stores, speculative fetches, cache activity or to visualize outstanding transactions in the interconnect.

Assuming there are ELA-500 components on the chip, you can use the **ELA** command group to trigger on on-chip signal events, to record processor-internal signals, or to halt the processor on an event for further investigations. The **ELA** command group represents the ELA features you can get by programming the ELA control register block, which you typically find on an APB bus like other CoreSight components. Therefore we recommend that you read ARM's ELA-500 documentation before you use this component.

The following diagram provides an overview of the major preparatory and configuration steps in TRACE32. For details, click the blue hyperlinks.



Step 1: The **ELA** command group is available after ELA-500 IP has been declared.

```
; declaring the ELA-500 IP activates the ELA commands
SYStem.CONFIG.ELA.Base <address>
```

Step 2: For TRACE32 to be able to record on-chip trace data, the trace method has to be set to **Onchip**.

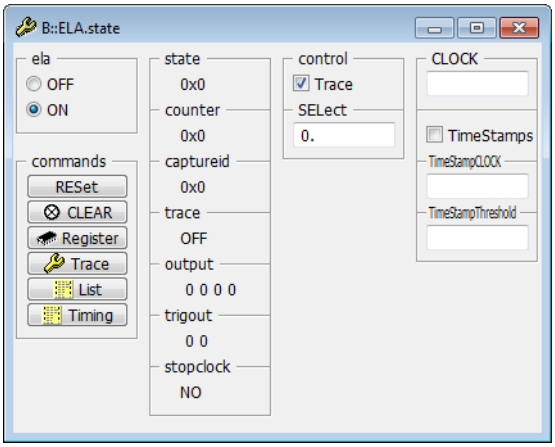
```
Trace.METHOD Onchip
```

Step 3: Using the **NAME** command group, you can assign user-defined names to the ELA signals you want to record (max. 128 signal names).

```
;         signal-idx    user-defined names
NAME.SET    Node.45     Node.RD_ELEM_SIZE_0
NAME.SET    Node.46     Node.RD_ELEM_SIZE_1
NAME.SET    Node.47     Node.RD_ELEM_SIZE_2
NAME.SET    Node.48     Node.RD_CACHE_ATTR_0
NAME.SET    Node.49     Node.RD_CACHE_ATTR_1
NAME.SET    Node.50     Node.RD_CACHE_ATTR_2
NAME.SET    Node.51     Node.RD_AARCH64

;          user-defined word name    signals forming the word
NAME.WORD   Word.RD_CACHE_ATTR       Node.RD_ELEM_SIZE_0 RD_ELEM_SIZE_1 \
                                     Node.RD_ELEM_SIZE_2
```

Configuration A: For ELA configuration, use the TRACE32 command line, a PRACTICE script (*.cmm), or the **ELA.state** window.



Configuration B: Example configuration for using both ELA and ETM program flow trace recorded in onchip trace buffers:

```
; Configure ELA trace source, creating trace of internal signals:
ELA.CLOCK 10.Mhz
ELA.TimeStampCLOCK 1.mhz
ELA.ON
ELA.state     ; Optionally open the ELA configuration window

; Connect ELA with trace sink "Onchip2":
Onchip2.TRACECONNECT ELA
Onchip2.state  ; Optionally open the "Onchip2" recording control window



; Configure ETM trace source, creating program flow trace:
ETM.CLOCK 50.MHz
ETM.TImeMode CycleAccurate
ETM.ON
ETM.state     ; Optionally open the ETM configuration window


; Connect ETM with trace sink "Onchip":
Onchip.TRACECONNECT ETM Onchip.state
Onchip.state  ; Optionally open the "Onchip" recording control window
```

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

You need to configure this option manually only for advanced operations. See **ETM.ATBTrigger** for more information.

See also

- ELA
- ELA.state

ELA.CLEAR Clear ELA.Set settings

ARM

Format: **ELA.CLEAR**

Clears all settings made with **ELA.Set**.

See also

- ELA
- ELA.state

ELA.CLOCK ELA sample rate

ARM

Format: **ELA.CLOCK** *<frequency>*

Informs the debugger about the sample rate used by the Embedded Logic Analyzer.

Example:

```
ELA.CLOCK 10.MHz
```

See also

- ELA
- ELA.state

ELA.OFF — Switch ELA off

ARM

Format: **ELA.OFF**

Switches the “Logic Analyzer” CoreSight component off.

See also

- ELA
- ELA.state

ELA.ON — Switch ELA on

ARM

Format: **ELA.ON**

Switches the “Logic Analyzer” CoreSight component on.

See also

- ELA
- ELA.state

ELA.PortRoute — Set up trace hardware

ARM

Format: **ELA.PortRoute** [**Analyzer** | **CAnalyzer** | **Onchip** | **Onchip2**]

Prepares the selected trace hardware for ELA trace capture.

Analyzer	PowerTrace (via TPIU)
CAnalyzer	Compact-Analyzer: CombiProbe or µTrace (MicroTrace)
Onchip **Onchip2**	Onchip trace buffer (ETB, ETF or ETR)

See also

- ELA
- ELA.state

Format: **ELA.Register** [*/<option>*]

<option>: **SpotLight** | **DualPort** | **Track** | **AlternatingBackGround**
CORE *<core_number>*

Displays the configuration register of the CoreSight Embedded Logic Analyzer.

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

See also

- ELA
- ELA.state

ELA.RESet Reset ELA settings

ARM

Format: **ELA.RESet**

Resets all ELA settings to default.

See also

- ELA
- ELA.state

Format:	**ELA.SELect** *<multiplexer>*
<multiplexer>:	**0**…**11**

Selects the signal group you want to record by default, e.g. the signal group **11**. Each signal group consists of 128 signals.

Using the **ELA.Set** commands, you can select different groups dynamically on events.

See also

- ELA
- ELA.state

ELA.Set Set ELA registers

ARM

Format:	**ELA.Set** *<state>* *<state_config>* **ELA.Set ALTernate** *<alternate_state_config>* **ELA.Set PreACTion** *<preaction_config>*
<state>:	**TS0** | **TS1** | **TS2** | **TS3** | **TS4**
<state_config>:	*<comparator_config>* | *<counter_config>* | *<misc_config>*

Configures the advanced trigger resources of the ELA-500 logic analyzer. As an alternative, you can use the **ELA.Register** window to configure the ELA registers manually.

<comparator_config>	See commands below to configure the signal comparator (Formats 1 to 5).
<counter_config>	See commands below to configure the event counter (Formats 6 to 9).
<misc_config>	See miscellaneous commands below for a given trigger state (Formats 10 to 12).

<preaction_config>	See command below at “Configure Output-State Before the First Action” (Format 13).
<alternate_state_ config>	See commands below at “Alternate Trace Enable” (Formats 14 to 21).

Formats 1 to 5 - Configuration of the Signal Comparator

Format 1:	**ELA.Set** *<state>* **COMParator.Signals** *<comparison>* [{**AND** *<comparison>*}]
<comparison>:	*<signals>* [**&** *<mask>*] *<operator>* *<reference>*
<signals>:	**Word.***<wordname>* | **Group.***<group_name>* | **Node.***<index>* | **CaptureID** | **NONE**
<reference>:	*<value>* | *<bitmask>* | **CaptureID**
<state>:	**TS0** | **TS1** | **TS2** | **TS3** | **TS4**
<operator>:	**>** | **>=** | **==** | **<=** | **<** | **!=**

CaptureID	**CaptureID** only possible if capturing of the current ID is disabled (**CaptureID NEVER** is the default). **CaptureID** can only be used on the left or right hand side of the comparison.
<operators>	The compare operators **==** and **!=** are always available. The compare operators **> < >= <=** are only available if the signals within the chosen *<word>* or *<group>* are in order.
AND	Further comparisons with **AND** are only possible if neither **CaptureID** nor compare operators **> < >= <=** are used in the first comparison.
<word>, *<group>*	If the chosen *<word>* or *<group>* has more than 64 bits, a second *<value>* or *<bitmask>* for the upper 64 bits can be specified.

Format 2:	**ELA.Set** *<state>* **COMParator.CtiTrigIn NONE** | *<value>* | *<bitmask>*
Format 3:	**ELA.Set** *<state>* **COMParator.ExtTrigIn NONE** | *<value>* | *<bitmask>*

NONE	Default for both Format 2 and 3.

Format 4:	**ELA.Set** *<state>* **COMParator.ACTion.hit** *<trace_action>* *<opt_actions>*
Format 5:	**ELA.Set** *<state>* **COMParator.ACTion.MISS** *<trace_action>* *<opt_actions>*
<action>:	**NONE** | **TraceON** | **TraceOFF** | **TraceBreak**
<opt_ actions>:	[**GOTO** *<state>*] [**OUTPUT** *<value>*] [**CtiTrigOut** *<value>*] [**CLOCKSTOP**]

NONE	For **ELA.Set** *<state>* **COMParator.ACTion.hit** (Format 4) the action **NONE** is only possible if the counter has been enabled via **ELA.Set***<state>* **COUNTer.ENable COMParatorHIT** or **ELA.Set***<state>* **COUNTer.Enable ALWAYS.** So you have to enable the counter first to disable an action triggered by a match of the comparator. **NONE** is the default action for **ELA.Set** *<state>* **COMParator.ACTion.MISS** (Format 5)
TraceON, **TraceOFF**	**TraceON** and **TraceOFF** are only possible in case of one of the following settings: • **COUNTer.ENable NEVER** (default) • Only Format 4: **COMParator.ACTion.MISS NONE** (default) • Only Format 5: **COUNTer.ENable COMParatorHIT** *and* **COMParator.ACTION.hit NEVER** **TraceON** is the default action for **ELA.Set** *<state>* **COMParator.ACTion.hit** (Format 4)

Format 6:	**ELA.Set** *<state>* **COUNTer.THreshold** *<value>*

Format 6 sets the value which must be reached by the counter to fire an event and trigger its action.

Format 7:	**ELA.Set** *<state>* **COUNTer.ENable NEVER** | **ALWAYS** | **COMParatorHIT**

ALWAYS	Every clock tick increments the counter.
NEVER (default)	**NEVER** only possible if **COMParator.ACTION.hit** is set to **TraceON** or **TraceOFF** (which is the default).
COMParatorHIT	The counter is incremented on a match of the signal comparator.

Format 8:	**ELA.Set** *<state>* **COUNTer.RESet NEVER** | **NewState** | **COMParatorHIT**

Format 8 defines which event resets the counter to zero.

NewState	Default for Format 8

Format 9:	**ELA.Set** *<state>* **COUNTer.ACTion.MISS** *<trace_action>* *<opt_actions>*
<action>:	**NONE** | **TraceON** | **TraceOFF** | **TraceBreak**
<opt_ actions>:	[**GOTO** *<state>*] [**OUTPUT** *<value>*] [**CtiTrigOut** *<value>*] [**CLOCKSTOP**]

Format 9 defines what should happen when the counter reaches the value configured with **ELA.Set** *<state>* **COUNTer.THreshold**.

Format 10:	**ELA.Set** *<state>* **SELect** *<ela_signal_group>*

Format 10 selects a group of 128-bit signals connected to the ELA block for the given triggers state.

Format 11:	**ELA.Set** *<state>* **TraceEnable** *<event>*
<event>:	**ALWAYS** | **COMParatorHIT** | **COUNTerEVent**

Format 11 defines events which enable the trace recording (inside the given state).

ALWAYS	Default for Format 11
COMParatorHIT	**COMParatorHIT** only possible if **COMParator.ACTion.MISS** is disabled (set to **NONE**, which is the default).

Format 12:	**ELA.Set** *<state>* **CaptureID** *<event>*
<event>:	**NEVER** | **COMParatorHIT**

Format 12 defines an event which loads the capture ID register (inside the given state).

NEVER	Default for Format 12
COMParatorHIT	**COMParatorHIT** only possible if **COMParator.SIGnals** is not comparing against the (previous) **CaptureID**.

Format 13 - Configure Output-State Before the First Action

Format 13:	**ELA.Set PreACTion** *<action> <opt_actions>*
<action>:	**NONE** | **TraceON** | **TraceOFF**
<opt_actions>:	[**OUTPUT** *<value>*] [**CtiTrigOut** *<value>*] [**CLOCKSTOP**]

Format 14 to 21 - Alternate Trace Enable

Format 14:	**ELA.Set ALTernate COMParator.Signals** *<comparison>* [{**AND** *<comparison>*}]
<comparison>:	*<signals>* [**&** *<mask>*] *<operator> <reference>*
<signals>:	**Word.***<wordname>* | **Group.***<group_name>* | **Node.***<index>* | **NONE**
<reference>:	*<value>* | *<bitmask>*
<operator>:	**>** | **>=** | **==** | **<=** | **<** | **!=**

Format 15:	**ELA.Set ALTernate COMParator.CtiTrigIn NONE** | *<value>* | *<bitmask>*
Format 16:	**ELA.Set ALTernate COMParator.ExtTrigIn NONE** | *<value>* | *<bitmask>*

Format 17:	**ELA.Set ALTernate COUNTer.THreshold** *<value>*
Format 18:	**ELA.Set ALTernate COUNTer.Enable NEVER** | **ALWAYS** | **COMParatorHIT**
Format 19:	**ELA.Set ALTernate COUNTer.RESet NEVER** | **COMParatorHIT**

Format 20:	**ELA.Set ALTernate SELect** *<ela_signal_group>*
Format 21:	**ELA.Set ALTernate TraceEnable** *<event>*
<event>:	**NEVER** | **COMParatorHIT** | **COUNTerEVent**

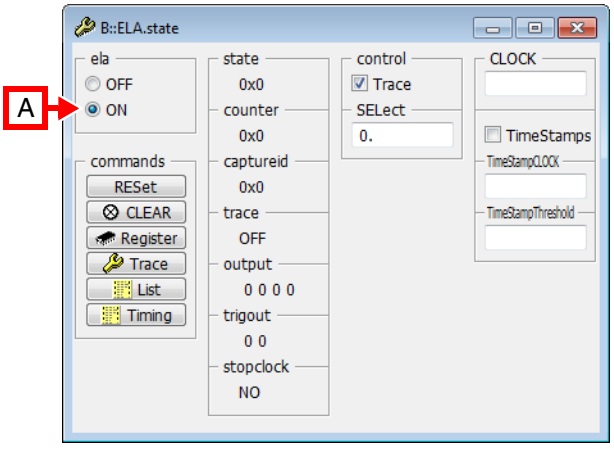
NEVER Setting to **NEVER** re-enables TS4 and disables the **ALTernate** mode.

See also

- ELA
- ELA.state

Format: **ELA.state**

Displays the ELA configuration window for the ELA-500 logic analyzer.



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

Exceptions:
- The **Trace** button opens the **Onchip.state** window.
- The **List** button opens the **Onchip.List** window.
- The **Timing** button opens the **Onchip.Timing** window.

See also

- ELA
- ELA.ATBTrigger
- ELA.CLEAR
- ELA.CLOCK
- ELA.OFF
- ELA.ON
- ELA.PortRoute
- ELA.Register
- ELA.RESet
- ELA.SELect
- ELA.Set
- ELA.SyncPeriod
- ELA.TimeStampCLOCK
- ELA.TimeStamps
- ELA.TimeStampThreshold
- ELA.Trace
- ELA.TraceID
- ELA.TracePREDICT
- ELA.TracePriority

ELA.SyncPeriod Set synchronization frequency

ARM

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

Synchronisation and timestamp packets are generated periodically. The default frequency is 1024 clock or 1024 bytes.

The command **ELA.SyncPeriod** allows to change this value.

See also

- ELA
- ELA.state

ELA.TimeStampCLOCK External clock frequency

ARM

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

Informs the debugger about the frequency of the CoreSight global timestamp generator.

See also

- ELA
- ELA.state

ELA.TimeStamps Emit global timestamp packets

ARM

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

Emits global timestamp packets.

ON	Inserts global timestamps from the CoreSight global timestamp generator to the captured trace records.
OFF (default)	Do not generate global timestamp packets.

See also

- ELA
- ELA.state

ELA.TimeStampThreshold — Set granularity for occurrence of timestamps

ARM

Format: **ELA.TimeStampThreshold** *<threshold>*

Default: 4

Configures the granularity of the timestamp information in the trace recording. The default value results in a timestamp package when the timestamp counter has incremented by 4. The default value gives optimal timing accuracy, since two bits of timing information are stored in each data packet.

See also

- ELA
- ELA.state

ELA.Trace — Control generation of trace information

ARM

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

Enables the recording of CoreSight internal signals without the need to meet a certain trigger condition.

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

See also

- ELA
- ELA.state

ELA.TraceID — Change the default ID for an ELA trace source

ARM

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

Allows to assign an ID to a trace source overriding the defaults.

See also

- ELA
- ELA.state

ELA.TracePREDICT — Enable/disable prediction

ARM

Format: **ELA.TracePREDICT** [**ON** | **OFF**]

Refer to the ARM ELA Reference Manual for more information.

See also

- ELA
- ELA.state

ELA.TracePriority — Define priority of ELA

ARM

Format: **ELA.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 **ELA.TracePriority** allows to change the default priority of an ATB input port.

See also

- ELA
- ELA.state

ELA<trace> - Trace Data Analysis

Using the **ELA<trace>** command group, you can configure the trace recording as well as analyze and display the recorded ELA trace data. The command groups consist of the name of the trace source, here **ELA**, plus the TRACE32 trace method you have chosen for recording the ELA trace data.

For more information about the TRACE32 convention of combining *<trace_source>* and *<trace_method>* to a *<trace>* command group that is aimed at a specific trace source, see **"Replacing <trace> with Trace Source and Trace Method - Examples"** (general_ref_t.pdf).

Not any arbitrary combination of *<trace_source>* and *<trace_method>* is possible. For an overview of the available command groups **"Related Trace Command Groups"** (general_ref_t.pdf).

ELAAnalyzer Analyze ELA information recorded by TRACE32 PowerTrace

Format:	**ELAAnalyzer.***<sub_cmd>*

The **ELAAnalyzer** command group allows to display and analyze the information emitted by the CoreSight Embedded Logic Analyzer (**ELA**).

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **ELAAnalyzer.List** refer to **<trace>.List**

ELACAnalyzer Analyze ELA information recorded by Compact Analyzer

Format:	**ELACAnalyzer.***<sub_cmd>*

The **ELACAnalyzer** command group allows to display and analyze the information emitted by the CoreSight Embedded Logic Analyzer (**ELA**).

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **ELACAnalyzer.List** refer to **<trace>.List**

ELAHAnalyzer Analyze ELA information captured by the host analyzer

Format:	**ELAHAnalyzer.***<sub_cmd>*

The **ELAHAnalyzer** command group allows to display and analyze the information emitted by the CoreSight Embedded Logic Analyzer (**ELA**). Trace data is transferred off-chip via the USB port and is recorded in the trace memory of the TRACE32 host analyzer.

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **ELAHAnalyzer.List** refer to **<trace>.List**

ELALA Analyze ELA information from binary source

Format:	**ELALA.***<sub_cmd>*

The **ELALAnalyzer** command group allows to display and analyze the information emitted by the CoreSight Embedded Logic Analyzer (**ELA**). Trace data is collected from Lauterbach's Logic Analyzer or from a binary file.

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **ELALAnalyzer.List** refer to **<trace>.List**

ELAOnchip Analyze ELA information captured in target onchip memory

Format:	**ELAOnchip.***<sub_cmd>*

The **ITMOnchip** command group allows to display and analyze the information emitted by the CoreSight Embedded Logic Analyzer (**ELA**).

The ELA information emitted to the target's onchip memory is read by TRACE32 via debug cable (JTAG).

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **ELAOnchip.List** refer to **<trace>.List**

ELATrace — Method-independent analysis of ELA trace data

Format: **ELATrace.***<sub_cmd>*

The **ELATrace** command group can be used as a generic replacement for the above **ELA<trace>** command groups.

<sub_cmd>	For descriptions of the subcommands, please refer to the general *<trace>* command descriptions in **"General Commands Reference Guide T"** (general_ref_t.pdf). **Example**: For a description of **ELATrace.List** refer to **<trace>.List**

ETA Energy test analysis for energy profiling

See also

- ETA.DRAW
- ETA.List
- ETA.ListNesting
- ETA.PROfileChart
- ETA.PROfileSTATistic
- ETA.RESet
- ETA.SELect
- ETA.state
- ETA.STATistic

Overview ETA

This chapter describes the **ETA** command group (**E**nergy **T**est **A**nalysis) for energy profiling. The **ETA** command group provides the means to perform detailed analyses of the energy consumption of embedded systems.

To save energy in general and to reduce battery drain in particular it is more important to focus on energy consumption (*energy* = *power* x *time*) rather than power consumption (*power* = *voltage* x *current*).

Each of these energy parameters can be influenced by the control software to extend the battery lives of portable and mobile devices, personal IT and other battery-operated products. Therefore, software developers have to constantly try to find the optimal settings for these three parameters in the respective operating mode of the application.

Simply reducing the number of source-code lines of a function that is called rarely does not significantly reduce energy consumption.

NOTE:	TRACE32 *simultaneously records* and *automatically correlates* the measurement of the program flow with the measurement of current and voltage.

Using the **ETA** commands, you can then see at a glance how often functions are called, how long they take, and how much energy they consume. For example, energy-hungry functions that are called frequently are ideal candidates for source code optimization.

ETA.state	Allows you to make global settings and provides links to the most common analysis commands.
ETA.SELect	Select the power channels (p0, p1, p3) to be analyzed.
ETA.List	Lists the ETA contents.
ETA.ListNesting	Displays function call nesting.
ETA.PROfileChart.*	Plots charts displaying power consumption by function as function of time.
ETA.PROfileSTATistic.*	Shows the results of numerical interval analysis in tabular format.
ETA.STATistic.*	Performs a statistical analysis of the energy consumption.

Getting Started

Prerequisite: Use one of the three hardware configuration options shown in the Lauterbach whitepaper "Optimization of Embedded Software's Energy Consumption".
See **https://www.lauterbach.com/doc/optimization_of_embedded_software.pdf**.

The following step-by-step procedure is just a suggestion. It is intended to give you an idea of how to perform an energy analysis using the ETA commands.

NOTE: The suggested approach for an energy analysis is very similar to a run-time analysis.
- In a run-time analysis, the central question is: How long does a particular function take?
- In an energy analysis, the central question is: How much energy does a particular function consume?

1. (Optional step) Make an assumption about the expected energy consumption of the target system.
2. Extend the start-up script of your application with the setup for the Analog Probe.
   - Example: Setup for Analog Probe and IProbe
   - Example: Setup for Analog Probe and CombiProbe
3. Start TRACE32 and run your extended start-up script.
   - TRACE32 records and automatically correlates the measurement of the program flow with the measurement of current and voltage.
   - Symbol information is stored in the TRACE32 symbol database.

4. Identify energy-hungry functions, for example, by opening a statistic, profile statistic, and profile chart window:

```
ETA.STATistic.sYmbol Ratio BAR.log Total Count ALL /Track \
                                                    /Sort Ratio

ETA.PROfileSTATistic.sYmbol /Track /InterVal 100.ms /Sort Ratio

ETA.PROfileChart.sYmbol /Track /ZoomTrack /Sort Ratio
```

5. Analyze the reasons for the energy consumption.
6. Optimize the source code of your application based on the result of your analysis.
7. Verify your source code optimization by repeating the above steps (3. and 4.) and comparing energy consumption before and after source code optimization.

Example: Setup for Analog Probe and IProbe

```
...
; Setup for hardware configuration option involving
; Analog Probe and IProbe

POD.ADC IP V0 ON 8/1                ;Activate the voltage channels V0
POD.ADC IP V1 ON 8/1                ;to V3 and set the compression rate
POD.ADC IP V2 ON 8/1                ;to 8/1
POD.ADC IP V3 ON 8/1

POD.ADC IP I0 ON 8/1 1.000          ;Activate the current channels I0
POD.ADC IP I1 ON 8/1 1.000          ;to I2, set the compression rate to
POD.ADC IP I2 ON 8/1 1.000          ;8/1. Set the shunt to 1.0 Ohm.

POD.ADC IP P0 ON 8/1 3.300          ;Activate the power channels P0 to
POD.ADC IP P1 ON 8/1 3.300          ;P2, set the compression rate to
POD.ADC IP P2 ON 8/1 3.300          ;8/1. Set the voltage to 3.3.

IProbe.OFF
IProbe.AutoInit ON
...
```

```
...
; Setup for hardware configuration option involving
; Analog Probe and CombiProbe

CIProbe.CONFIG.CHANNEL CIP.V0 ON      ;Activate the voltage channels V0
CIProbe.CONFIG.CHANNEL CIP.V1 ON      ;to V3.
CIProbe.CONFIG.CHANNEL CIP.V2 ON
CIProbe.CONFIG.CHANNEL CIP.V3 ON

CIProbe.CONFIG.CHANNEL CIP.I0 1.0     ;For the current channels I0 to I2,
CIProbe.CONFIG.CHANNEL CIP.I1 1.0     ;set the shunt to 1.0 Ohm.
CIProbe.CONFIG.CHANNEL CIP.I2 1.0

CIProbe.CONFIG.CHANNEL CIP.I0 ON      ;Activate the current channels I0
CIProbe.CONFIG.CHANNEL CIP.I1 ON      ;to I2.
CIProbe.CONFIG.CHANNEL CIP.I2 ON

CIProbe.CONFIG.CHANNEL CIP.P0 3.3     ;For the power channels P0 to P1,
CIProbe.CONFIG.CHANNEL CIP.P1 3.3     ;set the voltage to 3.3 volts.
CIProbe.CONFIG.CHANNEL CIP.P0 ON      ;Activate the power channels P0
CIProbe.CONFIG.CHANNEL CIP.P1 ON      ;to P1.

CIProbe.OFF
CIProbe.AutoInit On
...
```

The recording times vary depending on the used hardware configuration (see “**Getting Started**”). If you would like to increase the recording time, observe the following tips and tricks.

Analog Probe and IProbe

At maximum sampling rate, the built-in memory of the IProbe allows a recording time of 1.5 seconds.

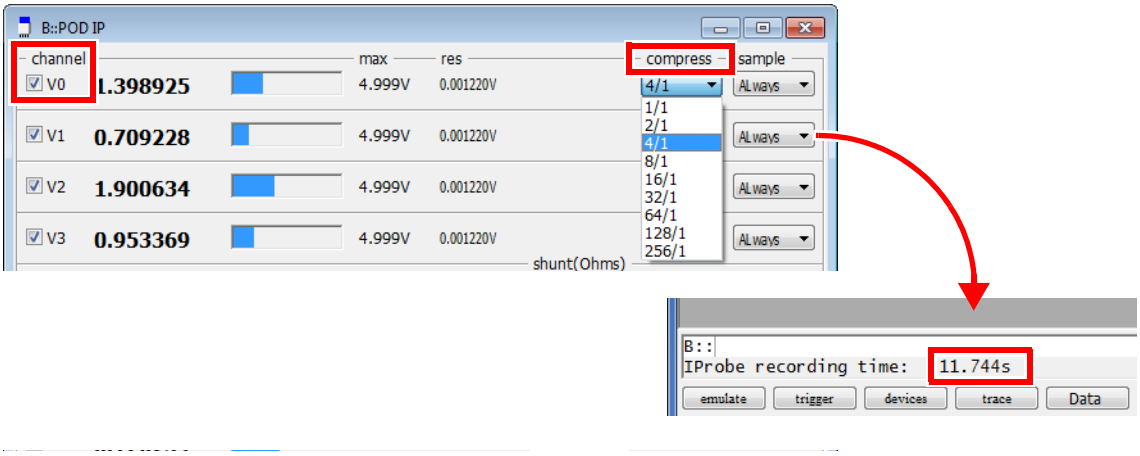
To increase the recording time:

1. In TRACE32, open the **POD IP** window by typing at the command line:

```
POD IP
```

2. For each channel, select the desired compression factor from the **compress** drop-down list.

   TRACE32 displays the resulting recording time below the command line.



Alternatively, you can include this setting in your start-up script:

```
...
; IProbe setup
POD.ADC IP V0 ON 4/1 ; Configuration for channel V0, see above screenshot
...
```

PowerIntegrator

In TRACE32, choose **ProbeADC** menu > **Recording Time** to configure the recording time.

Analog Probe and CombiProbe

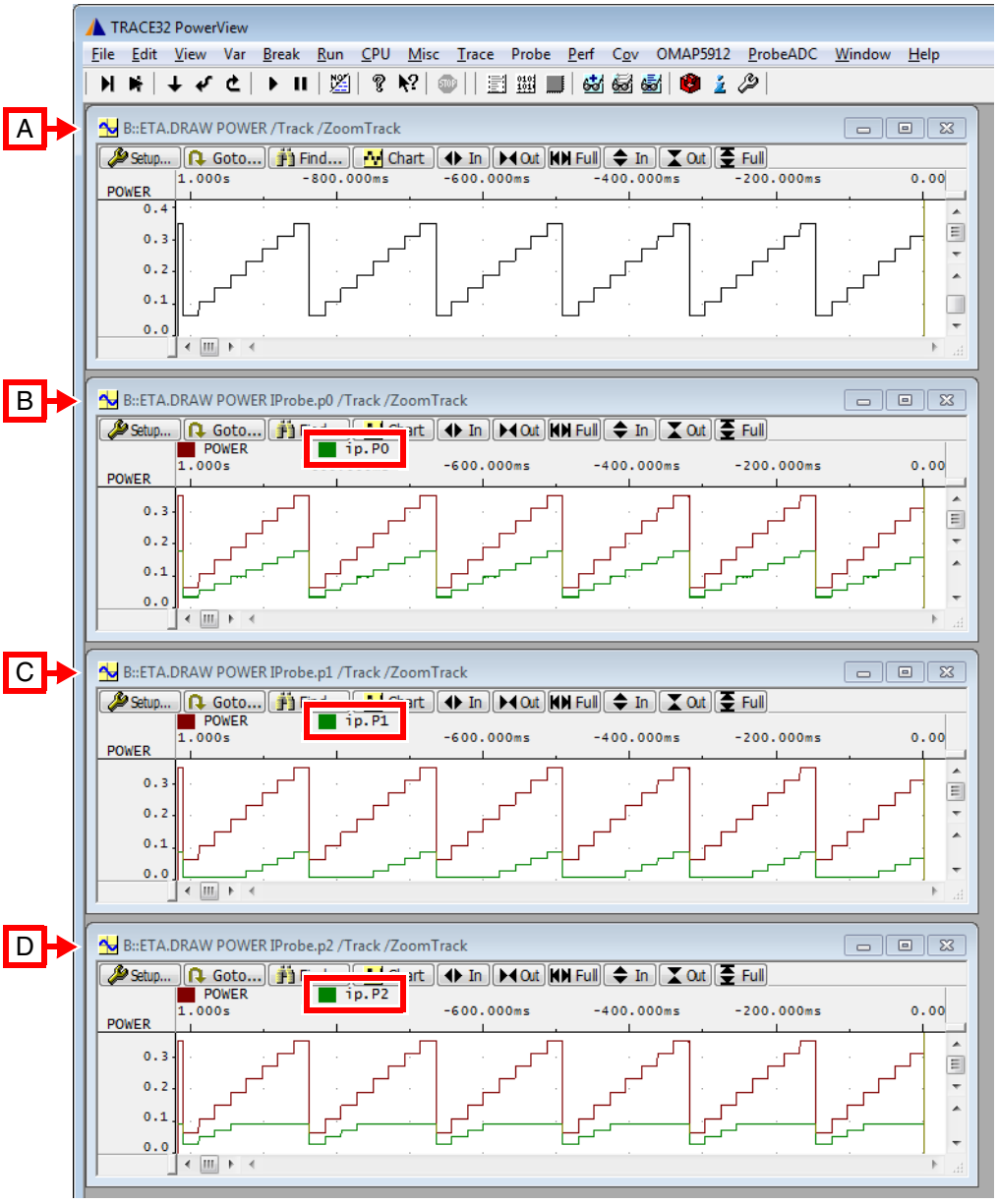
In streaming mode, the recording time is only limited by the capacity of your hard disk drive.

Format:	**ETA.DRAW** [<*record_range*>] [**%**<*format*>] [<*items …*>] [**/**<*options*>]
<*record_range*>:	<*start_record*>**--**<*end_record*>
<*format*>:	**Decimal.**[<*width*>] **DecimalU.**[<*width*>] **Hex.**[<*width*>] **Float.**[**Ieee** | **IeeeDbl** | **IeeeeXt** | **IeeeMFFP** | …]
<*width*>:	**DEFault** | **Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte**
<*items*>:	**DEFault** | **ALL** | <*cpu*> | <*signals*> | **IProbe**[**.**<*subitem*>] | **MARK**[**.**<*marker*>] | **ENERGY.Abs** | **POWER**[**.OFF**] | **SAMPLE**[**.OFF**] | **SPARE**[**.OFF**]
<*option*>:	**FILE** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** <*draw_options*> | **MinMax** | **Color** | **LOG** | **FIRST** <*address*> | **Filter** <*filter_items*> **Track** **ZoomTrack**
<*draw_options*>:	**Points** | **Vector** | **MarkedVector** | **Steps** | **Impulses**

Plots analog trace data as a line chart. Each voltage, current, or power channel can be displayed separately or together in a single window.

Example:

- The first **DRAW** window displays a line chart for all three power channels, plotted as one graph.
- The other three **DRAW** windows show power consumption in comparison to the power channels 0, 1, and 2.


TRACE32 PowerView
File Edit View Var Break Run CPU Misc Trace Probe Perf Cov OMAP5912 ProbeADC Window Help
A
B::ETA.DRAW POWER /Track /ZoomTrack
Setup... Goto... Find... Chart In Out Full In Out Full
POWER
1.000s -800.000ms -600.000ms -400.000ms -200.000ms 0.00
0.4 0.3 0.2 0.1 0.0
B
B::ETA.DRAW POWER IProbe.p0 /Track /ZoomTrack
ip.P0
C
B::ETA.DRAW POWER IProbe.p1 /Track /ZoomTrack
ip.P1
D
B::ETA.DRAW POWER IProbe.p2 /Track /ZoomTrack
ip.P2

Script for the above or similar screenshots - in this example, the IProbe is used.

```
WinPOS 0% 0% 96. 11.  ;Size and position of the first DRAW window.
ETA.DRAW POWER /Track /ZoomTrack            ;<- [A] See screenshot above

WinPOS 0% 22% 96. 11.
ETA.DRAW POWER IProbe.p0 /Track /ZoomTrack  ;<- [B] See screenshot above

WinPOS 0% 44% 96. 11.
ETA.DRAW POWER IProbe.p1 /Track /ZoomTrack  ;<- [C] See screenshot above

WinPOS 0% 66% 96. 11.
ETA.DRAW POWER IProbe.p2 /Track /ZoomTrack  ;<- [D] See screenshot above
```

See also

- ETA
- ETA.state

Format:	**ETA.List** [**%***<format>*] [*<record>* | *<record_range>*] [*<list_items>* …] [**/***<options>*]
<format>:	**DEFault** | **LEN** *<size>* **Timing** | **HighLow** | **01** | **Hex** | **Decimal** | **BINary** | **Ascii** | **Signed** | **Unsigned** | **TimeAuto** | **TimeFixed**
<list_items>:	**%***<format>* **DEFault** | **ALL** | *<cpu>* | *<signals>* | **List**[**.***<subitem>*] | **Port**[**.***<subitem>*] | **TIme**[**.***<subitem>*] | **CLOCKS**[**.***<subitem>*] | **TCLOCKS**[**.***<subitem>*] | **MARK**[**.***<marker>*] | **ENERGY**[**.***<subitem>*] | **POWER**[**.OFF**] | **SPARE**[**.OFF**]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **Track** **Mark**

Opens a window showing the recorded trace data starting at the specified *<record>* or for a range of trace records *<record_range>* (e.g. `(-10000.)--(-2000.)`).

The columns of the window can be defined using the *<list_items>*. The order of the columns in the window is according to the order of the *<list_item>* parameters given.

NOTE: Note that the default columns are hidden, when you manually specify the columns to display. They can be added using the option **DEFault** e.g.
`ETA.List List.address DEFault`

See also

- ETA
- ETA.state

Format:	**ETA.ListNesting** [**%***<format>*] [*<record>* | *<record_range>*] [*<list_items>* …] [***/***<options>*]
<format>:	**DEFault** | **LEN** *<size>* **Timing** | **HighLow** | **01** | **Hex** | **Decimal** | **BINary** | **Ascii** | **Signed** | **Unsigned** | **TimeAuto** | **TimeFixed**
<list_items>:	**%***<format>* **DEFault** | **ALL** | *<cpu>* | *<signals>* | **List**[**.***<subitem>*] | **Port**[**.***<subitem>*] | **TIme**[**.***<subitem>*] | **CLOCKS**[**.***<subitem>*] | **TCLOCKS**[**.***<subitem>*] | **MARK**[**.***<marker>*] | **ENERGY**[**.***<subitem>*] | **POWER**[**.OFF**] | **SPARE**[**.OFF**]
<option>:	**FILE** | **FlowTrace** | **BusTrace** **Track** **Mark**

Displays the recorded trace data in a way that the nesting of function calls is outlined together with the energy consumption per function per call.

See also

- ETA
- ETA.state

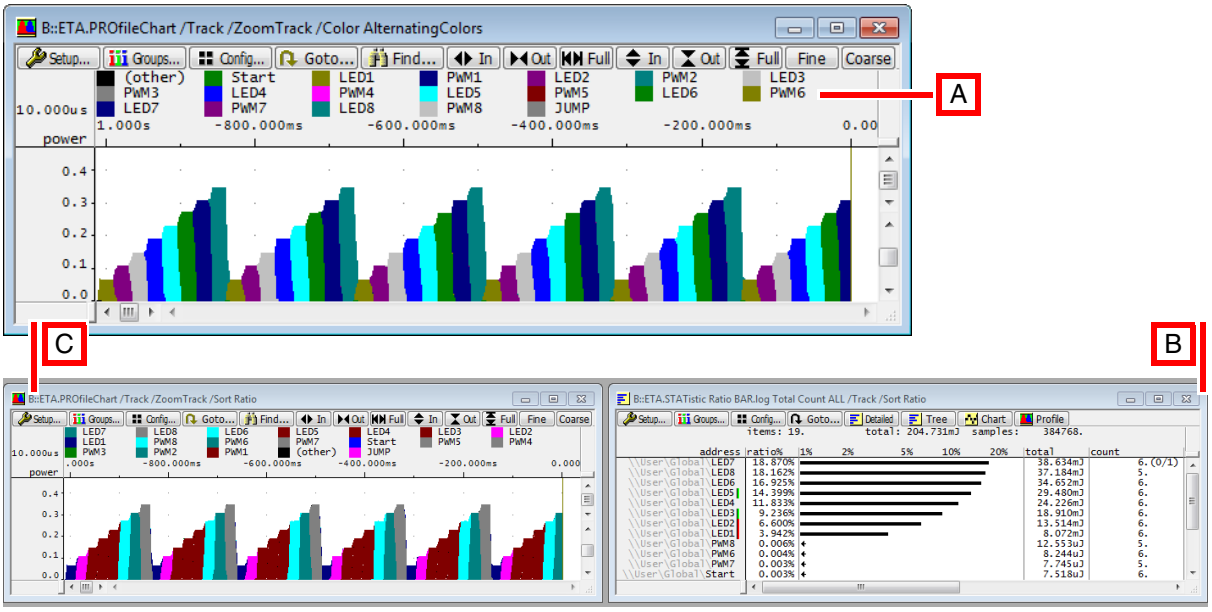
Format: **ETA.PROfileChart** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Shows power consumption by function as function of time.

<trace_area> Refer to **Trace.PROfileChart**.
<option>

Example:




```
;[A] Legend of the PROfileChart window is sorted versus time.
ETA.PROfileChart /Track /ZoomTrack /Color AlternatingColors

;[B] Allows you to compare the above PROfileChart window with
;    the total values in a STATistic window.
ETA.STATistic Ratio BAR.log Total Count ALL /Track /Sort Ratio

;[C] Legend of the PROfileChart window is sorted by energy consumption.
ETA.PROfileChart /Track /ZoomTrack /Sort Ratio
```

See also

- <trace>.PROfileChart.TASKVSINTERRUPT
- ETA.PROfileChart.DatasYmbol
- ETA.PROfileChart.GROUP
- ETA.PROfileChart.MODULE
- ETA.PROfileChart.PROGRAM
- ETA.PROfileChart.TASK
- ETA.PROfileChart.TASKINTR
- ETA.PROfileChart.TASKORINTERRUPT
- ETA.PROfileChart.TASKVSINTR
- ETA.state
- ETA.PROfileChart.AddressGROUP
- ETA.PROfileChart.DistriB
- ETA.PROfileChart.Line
- ETA.PROfileChart.POWER
- ETA.PROfileChart.sYmbol
- ETA.PROfileChart.TASKINFO
- ETA.PROfileChart.TASKKernel
- ETA.PROfileChart.TASKSRV
- ETA

▲ 'Release Information' in 'Legacy Release History'

ETA.PROfileChart.AddressGROUP Energy per GROUP graphically

Format: **ETA.PROfileChart.AddressGROUP** [[*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Plots a chart visualizing the energy consumption of the items pooled by the **GROUP.Create** command. The results include groups for both program and data.

<trace_area> Refer to **Trace.PROfileChart.AddressGROUP**.
<option>

See also

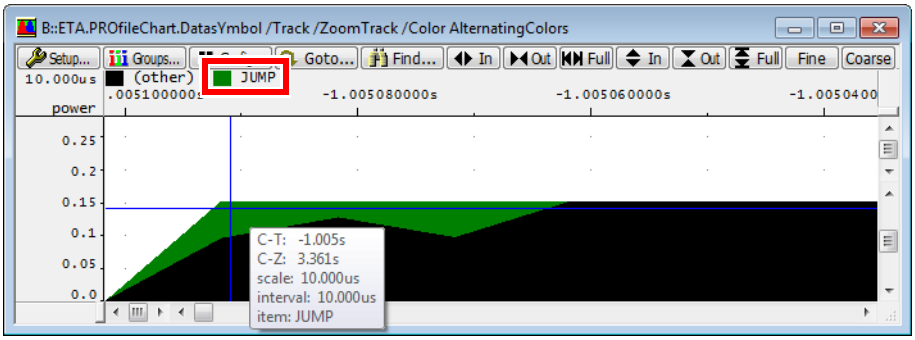
- ETA.PROfileChart
- ETA.PROfileChart.GROUP

Format:	**ETA.PROfileChart.DatasYmbol** [<trace_area>] [/<option>]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Analyzes data symbols and plots the energy consumption as a chart.

<trace_area>	Refer to **Trace.PROfileChart.DatasYmbol**.
<option>	

Example: The screenshot below visualizes the energy consumption of the data symbol JUMP.




```
ETA.PROfileChart.DatasYmbol /Track /ZoomTrack /Color AlternatingColors
```

See also

- ETA.PROfileChart

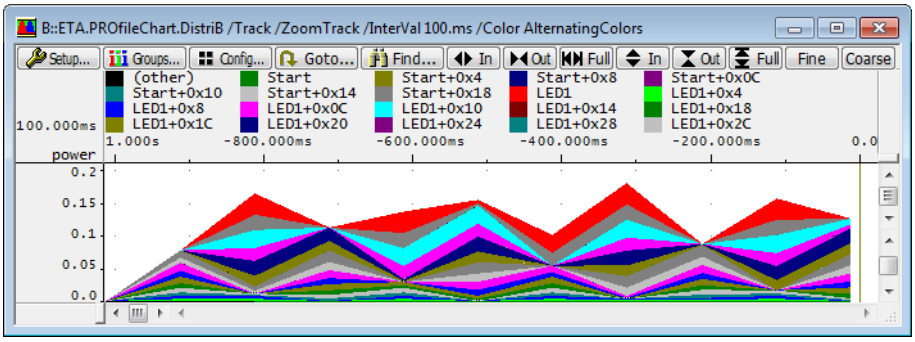
Format:	**ETA.PROfileChart.DistriB** [**%***<format>*] [*<trace_area>*] [*<items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** *<size>* **Timing** | **HighLow** | **01** | **Hex** | **Decimal** | **BINary** | **Ascii** | **Signed** | **Unsigned** | **TimeAuto** | **TimeFixed**
<items>:	**DEFault** | **ALL** | *<cpu>* | *<signals>* | **FLAGs** | **MARK** | **ENERGY.Abs** | **POWER**[**.OFF**] | **SAMPLE**[**.OFF**] | **SPARE**[**.OFF**]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The distribution of any data is displayed if *<item>* is specified. Without *<item>* the chart is based on the symbolic addresses, as shown in the screenshot below. If a filter is specified for a variable, this command visualizes how much energy the variable consumes in high-power and low-power state.

<trace_area> *<option>*	Refer to **Trace.PROfileChart.DistriB**.

Example:

```
ETA.PROfileChart.DistriB /Track /ZoomTrack /InterVal 100.ms /Color AlternatingColors
```



See also

- ETA.PROfileChart

Format:	**ETA.PROfileChart.Group** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Plots a chart visualizing the energy consumption of the items pooled by the **GROUP.Create** command. The results only include groups within the program range. Groups for data addresses are not included.

<trace_area> Refer to **Trace.PROfileChart.GROUP**.
<option>

Example: The screenshot below shows two groups: the group *LED1+2* and the group *LED3+5*. Non-grouped items make up the default group *(other)*. The **/Sort Ratio** parameter does not to sort the chart, but the chart legend by energy consumption in descending order.

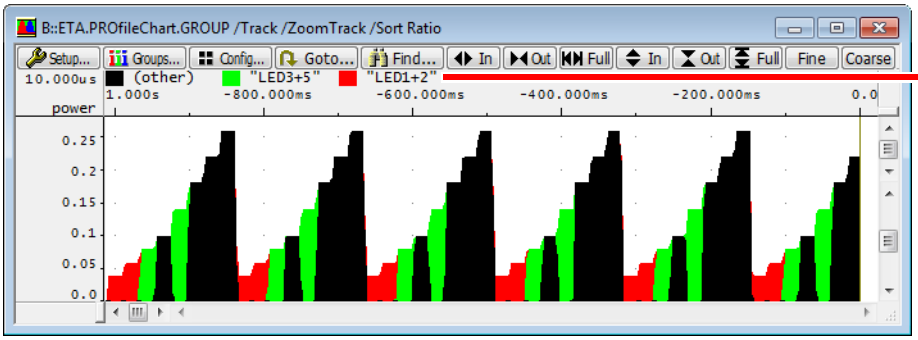


Chart legend sorted by energy consumption

Script for the above or similar screenshot:

```
; Create a group with two members: functions LED1 and LED2
; and assign color RED to the group named LED1+2.
Group.Create "LED1+2" Var.RANGE(LED1) Var.RANGE(LED2) /RED

; Second group.
Group.Create "LED3+5" Var.RANGE(LED3) Var.RANGE(LED5) /LIME

; Open the analysis window.
ETA.PROfileChart.GROUP /Track /ZoomTrack /Sort Ratio

; Lists all groups and allows you to edit them on the fly.
GROUP.List
```

See also

- ETA.PROfileChart
- ETA.PROfileChart.AddressGROUP

ETA.PROfileChart.Line Energy per high-level language line graphically

Format: **ETA.PROfileChart.Line** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Plots an energy chart for all executed high-level code lines.

<trace_area> Refer to **Trace.PROfileChart.Line**.
<option>

See also

- ETA.PROfileChart

ETA.PROfileChart.MODULE Energy per module graphically

Format: **ETA.PROfileChart.MODULE** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Plots an energy chart for the code execution broken down by symbol modules. The list of loaded modules can be displayed with **sYmbol.List.Module**.

<trace_area> Refer to **Trace.PROfileChart.MODULE**.
<option>

See also

- ETA.PROfileChart

Format:	**ETA.PROfileChart.POWER** [**%**<*format*>] [<*trace_area*>] [<*list_items*> …] [**/**<*option*>]
<*format*>:	**ZeroUp.**[<*width*>] | **Up.**[<*width*>] | **Down.**[<*width*>] | **Frequency.**[<*width*>] | **POWER.**[<*width*>]
<*width*>:	**DEFault** | **Byte** | **Word** | **Long** | **Quad** | **TByte** | **HByte**
<*trace_area*>:	<*trace_bookmark*> | <*record*> | <*record_range*> | <*time*> | <*time_range*> [<*time_scale*>]
<*items*>:	**DEFault** | **ALL** | <*cpu*> | <*signals*> | **IProbe**[**.**<*subitem*>] | **MARK**[**.**<*marker*>] | **ENERGY.Abs** | **POWER**[**.OFF**] | **SAMPLE**[**.OFF**] | **SPARE**[**.OFF**]
<*option*>:	**FILE** | **FlowTrace** | **BusTrace** **RecScale** | **TimeScale** | **TimeZero** | **TimeREF** <*draw_options*> **InterVal** <*time*> | **Ratio** **Filter** <*filter_item*> **Sort** <*item*> **Track** **ZoomTrack**
<*draw_options*>:	**Vector** | **Steps** | **Color**

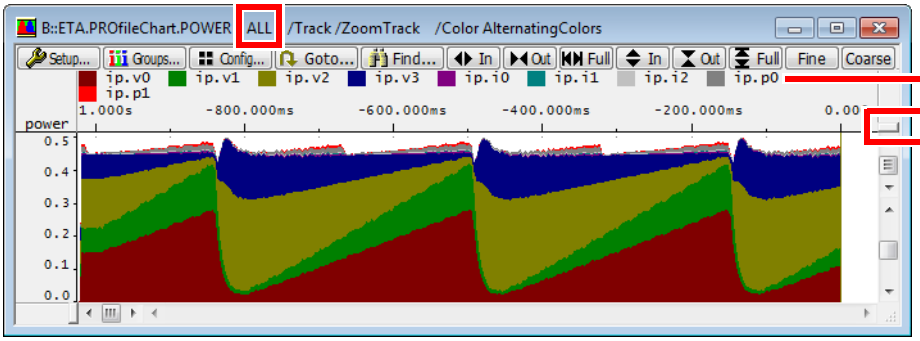
Plots a stacked area chart showing the energy consumption for each channel: 4 voltage channels (v0 to v3), 3 current channels (i0 to i2), and 3 virtual power channels (p0 to p2).

Example: In the screenshots below, **/ALL** is used to plot the energy consumption across all channels. It is also possible to plot the energy consumption only for selected channels. In this case, the syntax differs slightly depending on the hardware used, e.g. IProbe or CombiProbe.

- If the IProbe is used, click the **IProbe** softkey in TRACE32, and then select the channels you want to plot, for example, `ip.p0 ip.p1`, as shown in the screenshots below.
- If the CombiProbe is used, click the **CIProbe** softkey in TRACE32, and then select the channels you want to plot, for example, `cip.p0 cip.p1`, as shown in the screenshots below.

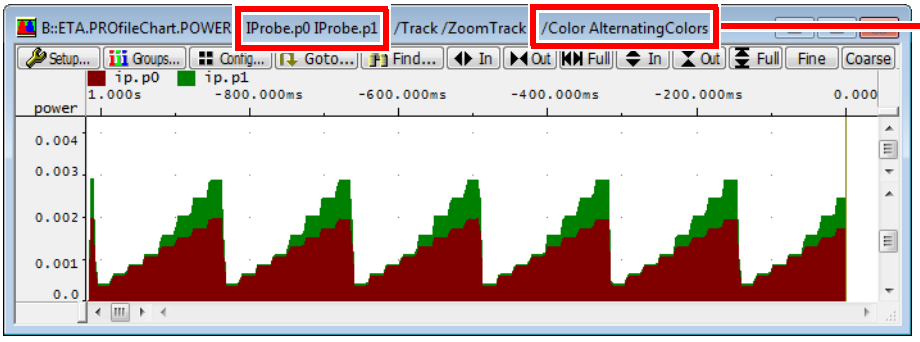
In the chart legend, the channel prefix indicates which hardware was used for recording the trace data: ip for the IProbe, cip for the CombiProbe.

Screenshots 1 and 2 - here the IProbe was used:



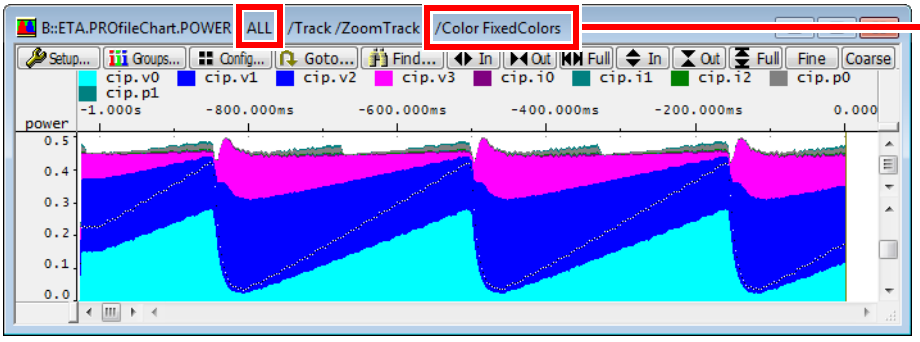
Legend showing the voltage, current, and power channels.

To display the legend, drag the handle down.

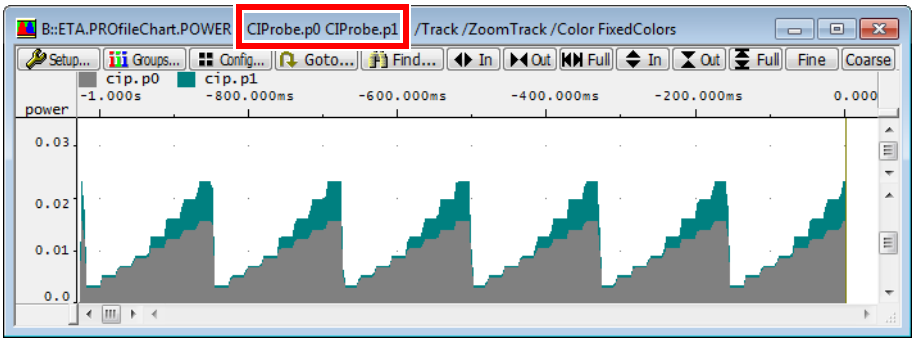


Coloring option 1

Screenshots 3 and 4 - here the CombiProbe was used:



Coloring option 2



Script for the above or similar screenshots:

```
; Prerequisite: trace data has been recorded.
; As a result, symbolic information is now stored in the symbol database.
; Allows you view and browse the symbol database:
sYmbol.Browse.Function

; Specifies the size and position of the PROfileChart window.
WinPOS 33% 0% 96. 15.

; Opens the profile chart window, plotting all voltage, current,
; and power channels.
ETA.PROfileChart.POWER ALL /Track /ZoomTrack

; Returns TRUE if the debugger is running on a TRACE32-IPROBE hardware.
IF IProbe()
(
   WinPOS 33% 28% 96. 15.
   ; Opens a second PROfileChart window, plotting just a subset
   ; of the available power channels: p0 and p1.
   ETA.PROfileChart.POWER IProbe.p0 IProbe.p1 /Track /ZoomTrack \
   /Color AlternatingColor
)

; If a CombiProbe is used:
IF CAnalyzer()
(
   WinPOS 33% 28% 96. 15.
   ETA.PROfileChart.POWER CIProbe.p0 CIProbe.p1 /Track /ZoomTrack \
   /Color FixedColors
)
```

NOTE:	The backslash **** can be used as a line continuation character in PRACTICE script files (*.cmm). No white space permitted after the backslash.

See also

- ETA.PROfileChart

Format: **ETA.PROfileChart.PROGRAM** [<trace_area>] [/<option>]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Plots an energy chart for loaded object file programs. The loaded programs can be displayed with the command **sYmbol.Browse ***.

<trace_area> Refer to **Trace.PROfileChart.PROGRAM**.
<option>

See also

- ETA.PROfileChart

ETA.PROfileChart.sYmbol Energy for all program symbols graphically

Format: **ETA.PROfileChart.sYmbol** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

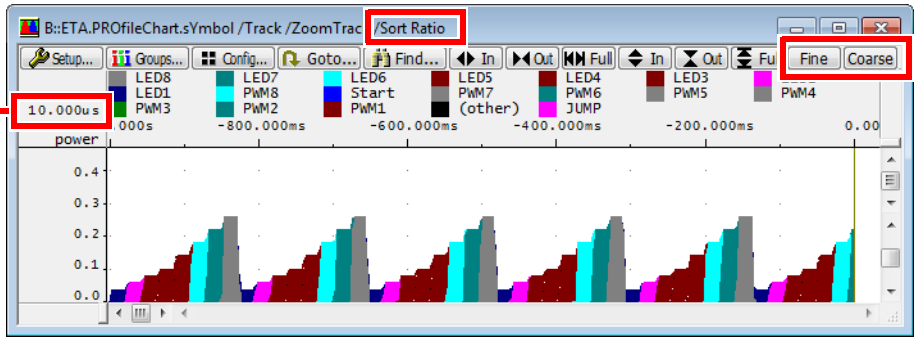
Draws an energy chart for all program symbols. The recorded trace data is divided into time intervals. The default interval is 10.us.

<trace_area> Refer to **Trace.PROfileChart.sYmbol**.
<option>

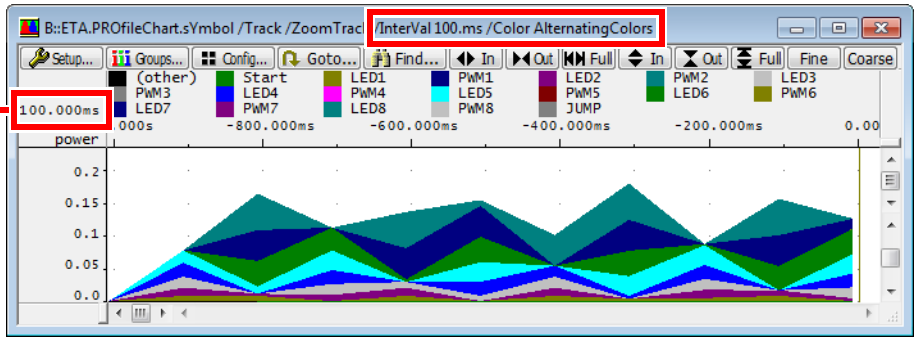
Example: The two screenshots below are based on the same trace data, but different parameters are applied to the charts.

For the first one, the default chart coloring and the default interval are used. The **/Sort Ratio** sorts the chart legend by energy consumption in descending order. For the second screenshot, a user-defined interval and alternating colors are used. Clicking the **Coarse** button increases the interval size; the current interval size is displayed in the top-left corner of the chart window.

Default interval



User-defined interval



Script for the above or similar screenshots:

```
; Prerequisite: trace data has been recorded.
; As a result, symbolic information is now stored in the symbol database.
; Allows you view and browse the symbol database:
sYmbol.Browse.Function

; Specifies the size and position of the first PROfileChart window.
WinPOS 33% 0% 96. 15.

; Opens the first profile chart window.
ETA.PROfileChart.sYmbol /Track /ZoomTrack /Sort Ratio

; Position of second window
WinPOS 33% 28% 96. 15.

; Opens the second PROfileChart window.
ETA.PROfileChart.sYmbol /Track /ZoomTrack /InterVal 100.ms /Color AlternatingColors
```

See also

- ETA.PROfileChart

Format: **ETA.PROfileChart.TASK** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Analyzes the energy consumption of all tasks and displays the result as a chart. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileChart.TASK**.
<option>

See also

- ETA.PROfileChart

ETA.PROfileChart.TASKINFO Energy per data trace message via context ID

Format: **ETA.PROfileChart.TASKINFO** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays a color chart of the energy consumption for special values written to the Context ID register for ETM trace. Please refer to **<trace>.PROfileChart.TASKINFO** for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKINFO**.
<option>

See also

- ETA.PROfileChart

Format: **ETA.PROfileChart.TASKINTR** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Analyzes the energy consumption of ORTI based ISR2 and displays the result as a chart. This feature can only be used if ISR2 can be traced based on the information provided by the ORTI file. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKINTR**.
<option>

See also

- ETA.PROfileChart

ETA.PROfileChart.TASKKernel Energy consumption per ISR2 graphically

Format: **ETA.PROfileChart.TASKKernel** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Refines the command **ETA.PROfileChart.TASK** for RTOS systems that don't assign a task ID to the kernel. Refer to **Trace.STATistic.TASKKernel** for more information. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileChart.TASKKernel**.
<option>

See also

- ETA.PROfileChart

Format: **ETA.PROfileChart.TASKORINTERRUPT** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Analyzes the energy consumption of all tasks and interrupts and displays the result as a time chart. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileChart.TASKORINTERRUPT**.
<option>

See also

- ETA.PROfileChart

ETA.PROfileChart.TASKSRV Energy consumption per service routine

Format: **ETA.PROfileChart.TASKSRV** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Performs an energy consumption analysis of the service routines and displays the result as a chart. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKSRV**.
<option>

See also

- ETA.PROfileChart

Format: **ETA.PROfileChart.TASKVSINTR** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Analyzes the energy consumption of all task-related interrupts and displays the result as a chart. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKINTR**.
<option>

See also

- ETA.PROfileChart

See also

- <trace>.PROfileSTATistic.TASKVSINTERRUPT
- ETA.PROfileSTATistic.Address
- ETA.PROfileSTATistic.AddressGROUP
- ETA.PROfileSTATistic.DatasYmbol
- ETA.PROfileSTATistic.DistriB
- ETA.PROfileSTATistic.GROUP
- ETA.PROfileSTATistic.INTERRUPT
- ETA.PROfileSTATistic.Line
- ETA.PROfileSTATistic.MODULE
- ETA.PROfileSTATistic.PROGRAM
- ETA.PROfileSTATistic.RUNNABLE
- ETA.PROfileSTATistic.sYmbol
- ETA.PROfileSTATistic.TASK
- ETA.PROfileSTATistic.TASKINFO
- ETA.PROfileSTATistic.TASKINTR
- ETA.PROfileSTATistic.TASKKernel
- ETA.PROfileSTATistic.TASKORINTERRUPT
- ETA.PROfileSTATistic.TASKSRV
- ETA
- ETA.state

ETA.PROfileSTATistic.Address Statistics about addresses

Format: **ETA.PROfileSTATistic.Address** [*<trace_area>*] *<address1> <address2> ...* [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Provides statistical information about the energy consumption of code addresses in tabular form.

<trace_area> Refer to **Trace.PROfileSTATistic.Address**.

<option>

See also

- ETA.PROfileSTATistic

Format: **ETA.PROfileSTATistic.AddressGROUP** [[*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays the energy consumption as a table for the items pooled by the **GROUP.Create** command. The results includes groups for both program and data.

<trace_area> Refer to **Trace.PROfileChart.AddressGROUP**.
<option>

See also

- ETA.PROfileSTATistic
- ETA.PROfileSTATistic.GROUP

ETA.PROfileSTATistic.DatasYmbol Statistics about data symbols

[Example]

Format: **ETA.PROfileSTATistic.DatasYmbol** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Provides statistical information about the energy consumption of data symbols in tabular form.

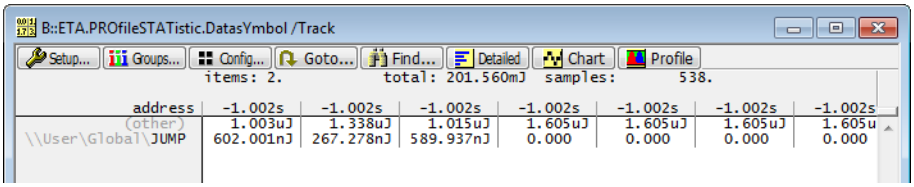
<trace_area> Refer to **Trace.PROfileSTATistic.DatasYmbol**.
<option>

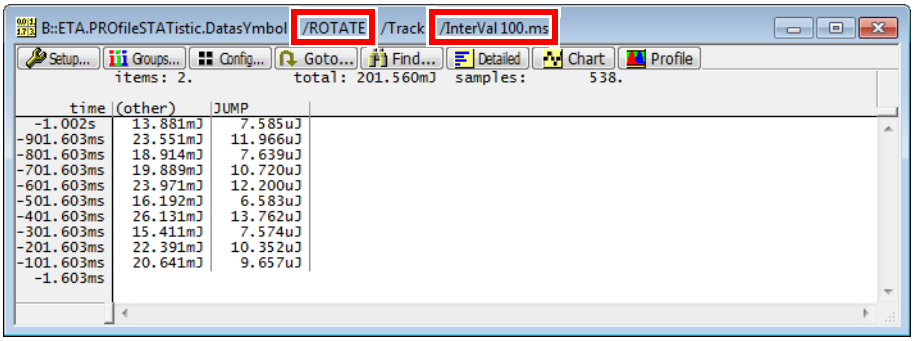
Example:

```
ETA.PROfileSTATistic.DatasYmbol /Track

ETA.PROfileSTATistic.DatasYmbol /ROTATE /Track /InterVal 100.ms
```

- In screenshot 1, the data symbol is given along the y-axis, the time intervals along the x-axis.
- In screenshot 2, the **/ROTATE** parameter is used to transpose columns and rows. As a result, the data symbol now appears on the x-axis, and the time intervals are listed along the y-axis.
  The **/InterVal** parameter calculates the energy consumption within the specified time interval.





See also

- ETA.PROfileSTATistic

Format: **ETA.PROfileSTATistic.DistriB** [**%**<*format*>] [<*trace_area*>] [<*list_items*> …] [/<*option*>]

<*trace_area*>: <*trace_bookmark*> | <*record*> | <*record_range*> | <*time*> | <*time_range*> [<*time_scale*>]

Displays the energy consumption in tabular format for the statistical distribution of a selected item or based on the symbolic addresses if no item is selected.

<*trace_area*> Refer to **Trace.PROfileSTATistic.DistriB**.
<*option*>

See also

- ETA.PROfileSTATistic

ETA.PROfileSTATistic.GROUP Energy per GROUP as a table

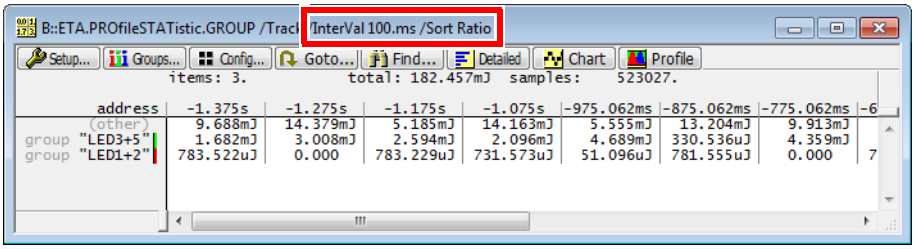
Format: **ETA.PROfileSTATistic.GROUP** [<*trace_area*>] [**/**<*option*>]

<*trace_area*>: <*trace_bookmark*> | <*record*> | <*record_range*> | <*time*> | <*time_range*> [<*time_scale*>]

Displays the energy consumption as a table for the items pooled by the **GROUP.Create** command. The results only include groups within the program range. Groups for data addresses are not included.

<*trace_area*> Refer to **Trace.PROfileSTATistic.GROUP**.
<*option*>

Example: The screenshot below shows two groups: the group *LED1+2* and the group *LED3+5*. Non-grouped items make up the default group *(other)*. The **InterVal** parameter is used to show the result in intervals of 100ms. The **/Sort Ratio** parameter sorts the groups by energy consumption in descending order.




```
; Prerequisite: trace data has been recorded.
; As a result, symbolic information is now stored in the symbol database.
; Allows you view and browse the symbol database:
sYmbol.Browse.Function

; Create a group with two members: functions LED1 and LED2
; and assign color RED to the group named LED1+2.
Group.Create "LED1+2" Var.RANGE(LED1) Var.RANGE(LED2) /RED

; Second group.
Group.Create "LED3+5" Var.RANGE(LED3) Var.RANGE(LED5) /LIME

; Open the analysis window.
ETA.PROfileSTATistic.GROUP /Track /InterVal 100.ms /Sort Ratio

; Add a third group.
Group.Create "LED7+8" Var.RANGE(LED7) Var.RANGE(LED8) /PURPLE

; The analysis window updates automatically to show the result.
```

See also

- ETA.PROfileSTATistic
- EVENTS.PROfileSTATistic.AddressGROUP
- ETA.PROfileSTATistic.AddressGROUP
- MIPS.PROfileSTATistic.AddressGROUP

Format: **ETA.PROfileSTATistic.INTERRUPT** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Analyzes the energy consumption of all interrupts and displays the result as a table with fixed time intervals. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.INTERRUPT**.
<option>

See also

- ETA.PROfileSTATistic

ETA.PROfileSTATistic.Line Energy per high-level language line as a table

Format: **ETA.PROfileSTATistic.Line** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays the energy consumption as a table for high-level code lines.

<trace_area> Refer to **Trace.PROfileSTATistic.Line**.
<option>

See also

- ETA.PROfileSTATistic

Format: **ETA.PROfileSTATistic.MODULE** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays the energy consumption as a table for symbol modules. The list of loaded modules can be displayed with **sYmbol.List.Module**.

<trace_area> Refer to **Trace.PROfileSTATistic.MODULE**.
<option>

See also

- ETA.PROfileSTATistic

ETA.PROfileSTATistic.PROGRAM Energy per program as a table

Format: **ETA.PROfileSTATistic.PROGRAM** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays the energy consumption as a table for loaded object file programs. The loaded programs can be displayed with the command **sYmbol.Browse ***.

<trace_area> Refer to **Trace.PROfileSTATistic.PROGRAM**.
<option>

See also

- ETA.PROfileSTATistic

Format: **ETA.PROfileSTATistic.RUNNABLE** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays the energy consumption as a table for AUTOSAR Runnables. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.RUNNABLE**.
<option>

See also

- ETA.PROfileSTATistic

ETA.PROfileSTATistic.sYmbol Energy for all program symbols as a table

Format: **ETA.PROfileSTATistic.sYmbol** [*<trace_area>*] [*/<option>*]

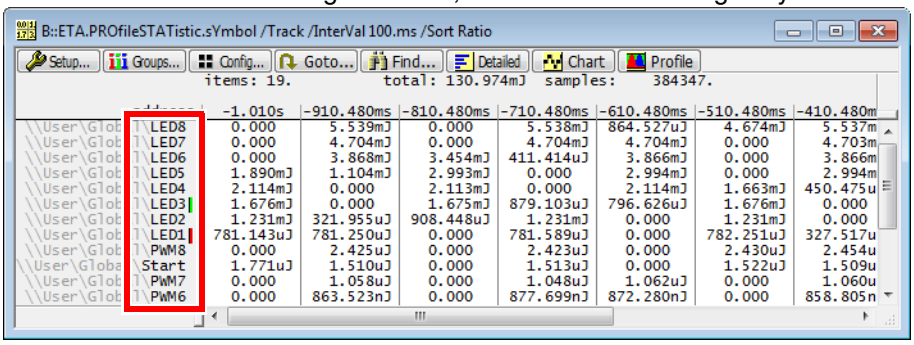
<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

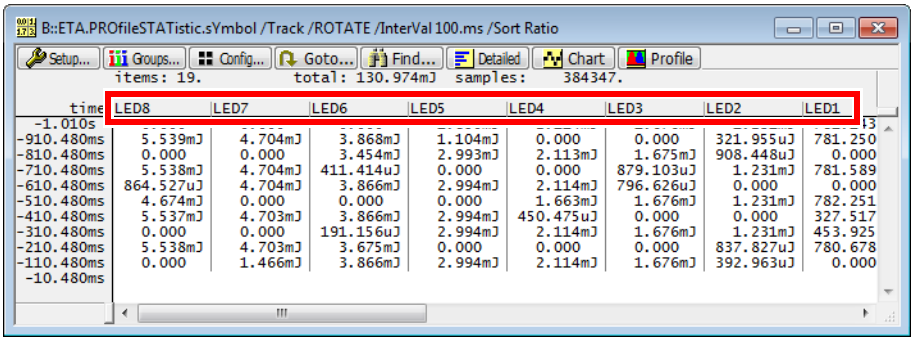
Lists the energy consumption per symbol in tabular form. The analysis can be displayed versus time, record or index number. Using the **/Ratio** option, the analysis can also be expressed as percentage of time. Do not confuse **/Ratio** with **/Sort Ratio** explained below.

<trace_area> Refer to **Trace.PROfileSTATistic.sYmbol**.
<option>

Example: In both screenshots, **/Sort Ratio** sorts the functions by energy consumption in descending order:

- In screenshot 1, the functions are listed along the y-axis, the time intervals along the x-axis.
- In screenshot 2, the **/ROTATE** parameter is used to transpose columns and rows. As a result, the functions are now listed along the x-axis, the time intervals along the y-axis.






```
ETA.PROfileSTATistic.sYmbol /Track /InterVal 100.ms /Sort Ratio
ETA.PROfileSTATistic.sYmbol /Track /ROTATE /InterVal 100.ms /Sort Ratio
```

See also

- ETA.PROfileSTATistic

ETA.PROfileSTATistic.TASK Energy consumption per TASK as a table

Format: **ETA.PROfileSTATistic.TASK** [<trace_area>] [/<option>]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Analyzes the energy consumption of all tasks and displays the result as a table with fixed time intervals. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.TASK**.
<option>

See also

- ETA.PROfileSTATistic

ETA.PROfileSTATistic.TASKINFO Energy per data trace via context ID

Format: **ETA.PROfileSTATistic.TASKINFO** [<trace_area>] [/<option>]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays a profile statistic table of the energy consumption for special values written to the Context ID register for ETM trace. Please refer to **<trace>.PROfileSTATistic.TASKINFO** for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKINFO**.
<option>

See also

- ETA.PROfileSTATistic

Format: **ETA.PROfileSTATistic.TASKINTR** [<trace_area>] [/<option>]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Performs a numerical interval analysis of the energy consumed by interrupt service routines (ISR2) and displays the result as a table with fixed time intervals. This feature can only be used if ISR2 can be traced based on the information provided by the ORTI file. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKINTR**.
<option>

See also

- ETA.PROfileSTATistic

ETA.PROfileSTATistic.TASKKernel Energy consumption as a table

Format: **ETA.PROfileSTATistic.TASKKernel** [<trace_area>] [/<option>]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Performs a numerical interval analysis of the energy consumed by tasks and kernel run-times and displays the result as a table with fixed time intervals. Kernel entry and exit must be marked with **sYmbol.MARKER.Create** commands. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKKernel**.
<option>

See also

- ETA.PROfileSTATistic

Format: **ETA.PROfileSTATistic.TASKORINTERRUPT** [<trace_area>] [/<option>]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Analyzes the energy consumption of all tasks and interrupts and displays the result as a table with fixed time intervals. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKORINTERRUPT**.
<option>

See also

- ETA.PROfileSTATistic

ETA.PROfileSTATistic.TASKSRV Energy analysis of service routines

Format: **ETA.PROfileSTATistic.TASKSRV** [<trace_area>] [/<option>]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Performs an energy consumption analysis of the task service routines. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKSRV**.
<option>

See also

- ETA.PROfileSTATistic

Format: **ETA.RESet**

Resets the power channels that have been selected using **ETA.SELect**.

See also

- ETA
- ETA.state

Format:	**ETA.SELect** [*<items>* …]
<items>:	**IProbe**[**.***<subitems>*] | **CIProbe**[**.***<subitems>*] | **Integrator**[**.***<subitems>*]

Allows you to select all three or just a subset of the three power channels (p0, p1, and p2) for analysis in a **ETA.PROfileChart** or **ETA.DRAW** window. Selecting just a subset is useful if you want to temporarily exempt certain power channels from the analysis.

The setting made via **ETA.SELect** is a global setting for the **ETA.PROfileChart** and **ETA.DRAW** windows.

Let's illustrate the **ETA.SELect** command by way of two examples.

Prerequisites for the two Examples

- Using the IProbe, we have recorded trace data for all channels (4 voltage channels, 3 current channels, and 3 virtual power channels).
- It is useful to assign mnemonic names to the power channels (to make them easy to remember).

```
; Assign mnemonic names to the power channels ip.p0, ip.p1, ip.p2
; The prefix ip indicates that the IProbe was used for recording
; the trace data.
NAME.SET IP.P0 IP.P-CPU
NAME.SET IP.P1 IP.P-RAM
NAME.SET IP.P2 IP.P-PERIPH
NAME.list ; Displays the result.
```

NOTE:	It is your responsibility to know what the power channels (p0, p1, and p2) stand for *in your case*, and assign mnemonic names accordingly. TRACE32 cannot "know" this.

Example 1

In this example, the 3 power channels stand for the CPU, the RAM, and the peripherals. Let's assume that we want to analyze just the power of the peripherals. To accomplish this, we simply exclude the power channels of the CPU and the RAM from ETA.SELect and include just the power channel of the peripherals.

To select just a subset of the power channels:

1. Open a **PROfileChart** window and an **ETA.state** window:

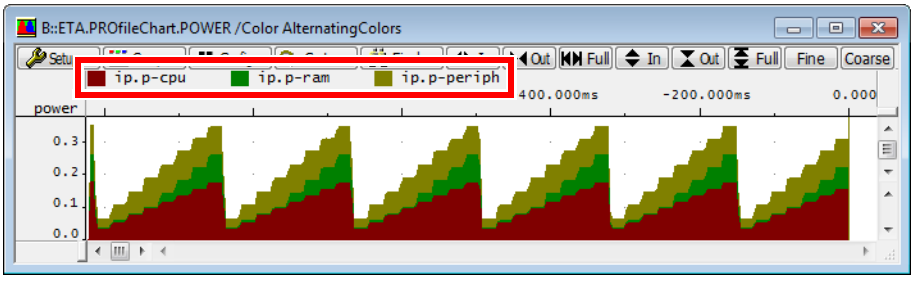
```
ETA.PROfileChart.POWER /Color AlternatingColors
ETA.state
```

2. In the **SELect** box, type: `ip.p-periph`
3. Update the **PROfileChart** window as follows: right-click the window caption, and then press the **Enter** key.
4. To display all power channels again, click **RESet**, or in the command line type:

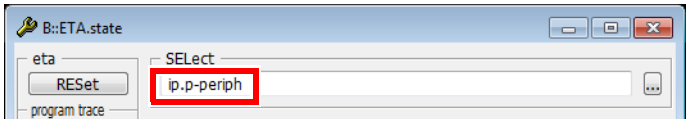
```
ETA.RESet
```

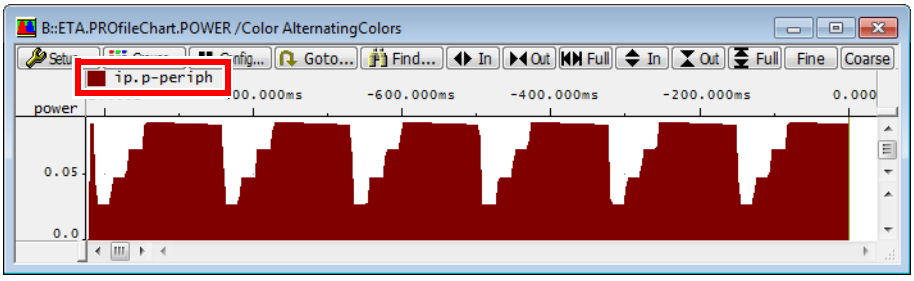
5. Update the **PROfileChart** window again.

Before



After





Example 2

The power channels selected via **ETA.SELect** are a global setting for **ETA.PROfileChart** and **ETA.DRAW** windows. You can manually override the global setting by explicitly specifying the power channels like this:

```
; Display the previously excluded power channels in another window.
ETA.PROfileChart.POWER IProbe.P-CPU IProbe.P-RAM /Color AlternatingColors
```

The complete script for example 2:

```
; Assigns mnemonic names to the power channels ip.p0, ip.p1, ip.p2
; The prefix ip indicates that the IProbe was used for recording
; the trace data.
NAME.SET IP.P0 IP.P-CPU
NAME.SET IP.P1 IP.P-RAM
NAME.SET IP.P2 IP.P-PERIPH
NAME.list ; Displays the result.

; Opens the ETA.state window.
ETA.state

; Global setting for ETA.PROfileChart and ETA.DRAW windows.
ETA.SELect ip.p-periph

; Plots the area chart based on the global setting.
ETA.PROfileChart.POWER /ZoomTrack /Color AlternatingColors

; Plots the line chart based on the global setting.
ETA.DRAW POWER /ZoomTrack

; Plots the area chart for the previously excluded power channels
; in another window without changing the global setting.
ETA.PROfileChart.POWER IProbe.P-CPU IProbe.P-RAM /ZoomTrack \
/Color AlternatingColors
```

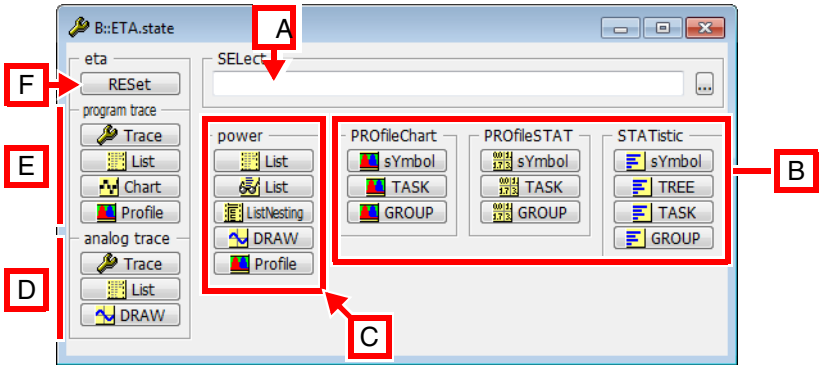
NOTE: The backslash **** can be used as a line continuation character in PRACTICE script files (*.cmm). No white space permitted after the backslash.

See also

- ETA
- ETA.state

Format: **ETA.state**

Opens the ETA configuration window, providing links to the most common ETA commands for analyzing the energy consumption of a target. In addition, the window provides access to commands for recording time-correlated program trace information and analog trace information about power consumption.



A Enter the power channels (p1, p2, p3) for which you want to record trace data.

B ETA commands for analyzing energy consumption.

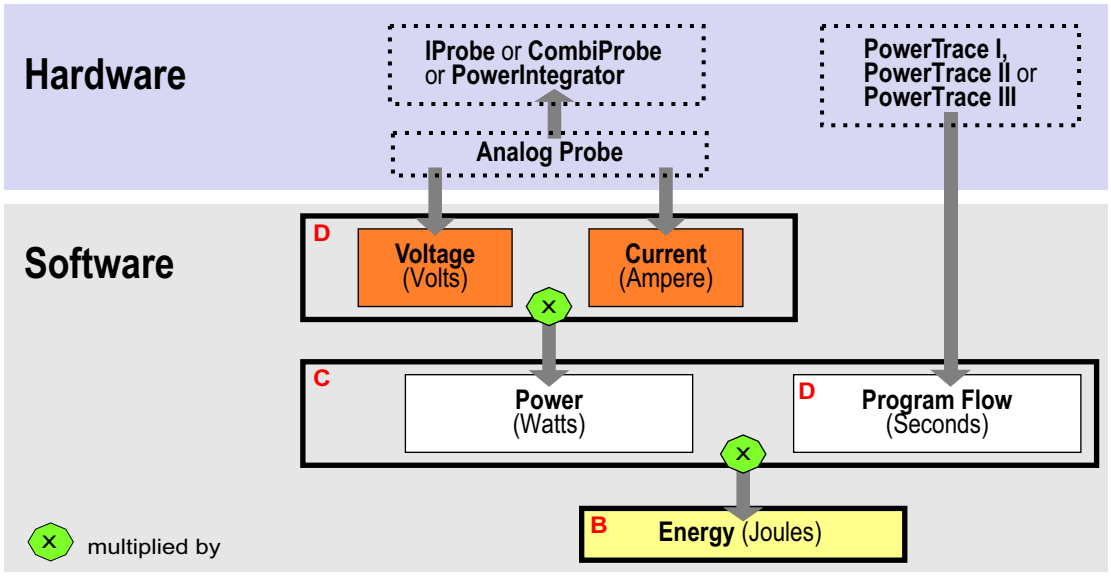
C ETA and trace commands for analyzing power consumption.

D Commands for recording power (**Trace** button) and analysis (**List**, **DRAW** buttons).

E Commands for recording the program flow (**Trace** button) and analysis (**List**, **Chart**, **Profile** buttons).

F Resets the trace buffer.

The callouts in the above **ETA.state** window ([**B**], [**C**], [**D**], and [**E**]) refer to the same callouts in the figure below. The formulas for power and energy illustrate the technical background of the **ETA.state** window



See also

- ETA
- ETA.DRAW
- ETA.List
- ETA.ListNesting
- ETA.PROfileChart
- ETA.PROfileSTATistic
- ETA.RESet
- ETA.SELect
- ETA.STATistic

Format: **ETA.STATistic** [%*<format>*] [*<list_items>* …] [*/<option>*]

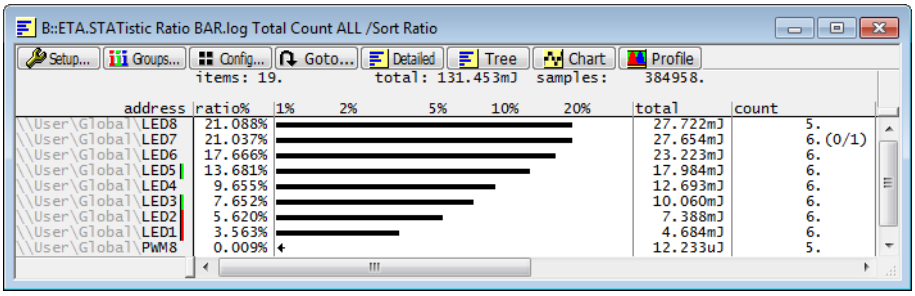
<format>: **DEFault** | **LEN** *<size>*
TimeAuto | **TimeFixed**

<list_items> **DEFault** | **ALL**
NAME
CORE
Total | **TotalMIN** | **TotalMAX**
Ratio | **RatioMIN** | **RatioMAX** | **BAR**[**.log** | **.LINear**]
Count | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX**
MIN | **MAX**
AVeRage

<option>: **FILE** | **FlowTrace** | **BusTrace**
BEFORE | **AFTER**
CountChange | **CountFirst** | **CountALL**
InterVal *<time>*
Filter *<filter_item>*
Accumulate
INCremental | **FULL**
CLOCKS
Sort *<item>*
Track

The **ETA.STATistic** commands can be used for a statistical energy consumption analysis based on the information sampled to the trace buffer.

Example: The screenshot below displays a statistical analysis of the program symbols. The **Total** and **Ratio** parameters show the exact amount of energy per symbol and as percentage values. **BAR** illustrates the percentage values as bar chart, and **Count** displays the number of calls for each symbol.




```
ETA.STATistic Ratio BAR.log Total Count ALL /Sort Ratio
```

See also

- ETA
- ETA.state

Format: **ETA.STATistic.ChildTREE** *<address>* [**%***<format>*] [*<list_items>* …] [**/***<option>*]

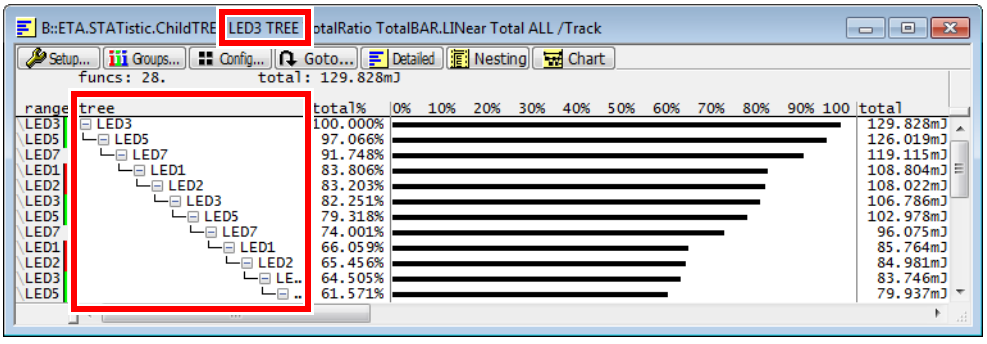
<format>: **DEFault** | **LEN** *<size>*
TimeAuto | **TimeFixed**

<list_items> **DEFault** | **ALL**
TREE
TASK
Total | **TotalRatio** | **TotalBAR**[**.log** | **.LINear**]
Count | **MIN** | **MAX**
AVeRage
Internal | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio**
InternalBAR[**.log** | **.LINear**]
External | **EAVeRage** | **EMIN** | **EMAX**
ExternalINTR | **ExternalINTRMAX**
INTRCount
ExternalTASK | **ExternalTASKMAX** | **TASKCount**

<option>: **FILE** | **FlowTrace** | **BusTrace**
TASK | **SplitTASK** | **MergeTASK**
IncludeOwn | **IncludeTASK** | **IncludeINTR**
INTRROOT | **INTRTASK**
Accumulate
INCremental | **FULL**
CLOCKS
NoMerge
Sort *<item>*
Track

Shows all children of a function as a tree. The screenshot below shows all children of the LED3 function.

Example:




```
ETA.STATistic.ChildTREE LED3 TREE TotalRatio TotalBAR.LINear Total ALL /Track
```

Format: **ETA.STATistic.DistriB** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

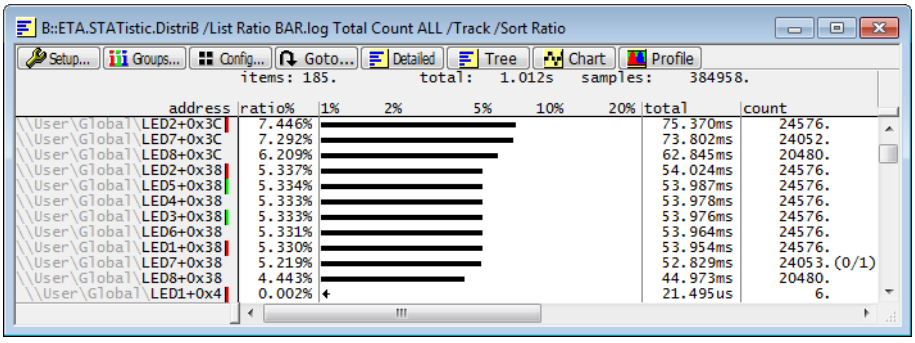
<format>: **DEFault** | **LEN** *<size>*
Timing | **HighLow** | **01** | **Hex** | **Decimal** | **BINary** | **Ascii** | **Signed** | **Unsigned** | **TimeAuto** | **TimeFixed**

<list_items>: **DEFault** | **ALL**
| *<cpu>* | *<signals>* | **Port**[**.***<subitem>* | **MARK**[**.***<marker>*] | **ENERGY.Abs** | **POWER**[**.OFF**] | **SAMPLE**[**.OFF**] | **SPARE**[**.OFF**]

<option>: **FILE** | **FlowTrace** | **BusTrace**
BEFORE | **AFTER**
CountChange | **CountFirst** | **CountALL**
List *<list_item>*
InterVal *<time>*
Filter *<filter_item>*
Accumulate
INCremental | **FULL**
CLOCKS
Sort *<item>*
Track

The statistic distribution of any data is displayed if *<list_tem>* is specified. The number of occurrences and the time after the events are displayed, i.e. the time an event is assumed to be valid. Without *<list_item>* the statistic is based on the symbolic addresses.

Example:




```
ETA.STATistic.DistriB /List Ratio BAR.log Total Count ALL /Track /Sort Ratio
```

Format: **ETA.STATistic.Func** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

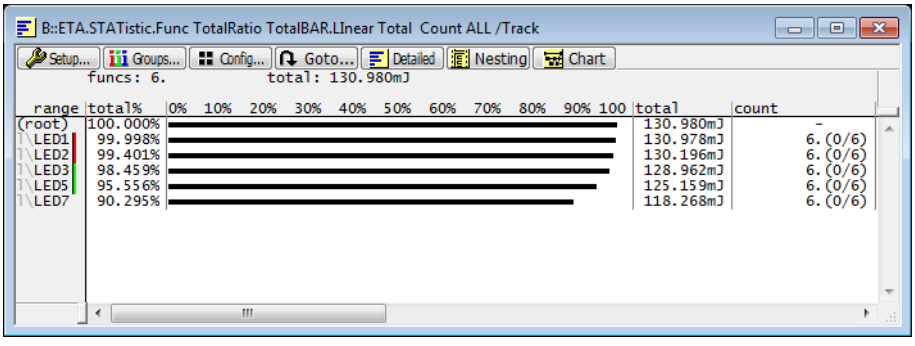
<format>: **DEFault** | **LEN** *<size>*
TimeAuto | **TimeFixed**

<list_items>: **DEFault** | **ALL**
NAME | **TASK**
Total | **TotalRatio** | **TotalBAR**[**.log** | **.LINear**]
Count | **MIN** | **MAX**
AVeRage
Internal | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio**
InternalBAR[**.log** | **.LINear**]
External | **EAVeRage** | **EMIN** | **EMAX**
ExternalINTR | **ExternalINTRMAX**
ExternalTASK | **ExternalTASKMAX** | **INTRCount** | **TASKCount**

<option>: **FILE** | **FlowTrace** | **BusTrace**
TASK | **SplitTASK** | **MergeTASK**
IncludeOwn | **IncludeTASK** | **IncludeINTR**
INTRROOT | **INTRTASK**
Accumulate
INCremental | **FULL**
CLOCKS
NoMerge
Sort *<item>*
Track

Analyzes the function nesting and calculates the energy consumed by functions and the number of function calls.

Example:




```
ETA.STATistic.Func TotalRatio TotalBAR.LINear Total Count ALL /Track
```

Format: **ETA.STATistic.GROUP** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

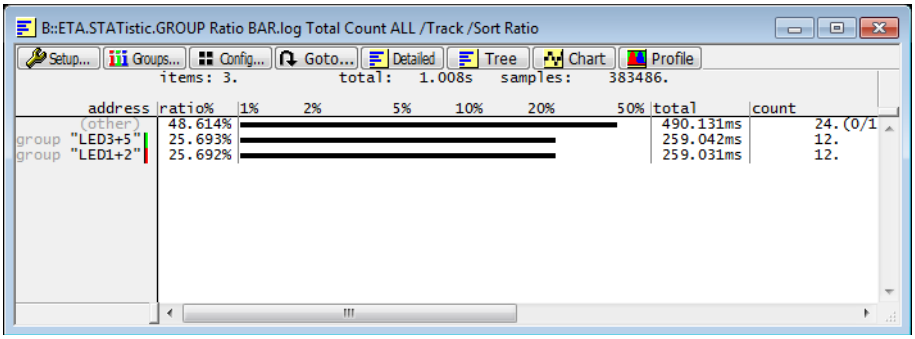
<format>: **DEFault** | **LEN** *<size>*
TimeAuto | **TimeFixed**

<list_items>: **DEFault** | **ALL**
NAME
CORE
Total | **TotalMIN** | **TotalMAX**
Ratio | **RatioMIN** | **RatioMAX**
BAR[**.log** | **.LINear**]
Count | **CountRatio** | **CountBAR**[**.log** | **.LINear**] | **CountMIN** | **CountMAX**
MIN | **MAX** | **AVeRage**

<option>: **FILE** | **FlowTrace** | **BusTrace**
BEFORE | **AFTER**
CountChange | **CountFirst** | **CountALL**
InterVal *<time>*
Filter *<filter_item>*
Accumulate
INCremental | **FULL**
CLOCKS
Sort *<item>*
Track

Measures the energy consumption per group and the number of calls. The results only include groups within the program range. Groups for data addresses are not included.

Example: The screenshot below shows two groups: the group *LED1+2* and the group *LED3+5*. Non-grouped items make up the default group *(other)*. The **Total** and **Ratio** parameters show the exact energy consumption per group and as percentage values. **BAR** illustrates the percentage values as bar chart, and **Count** displays the number of calls for each group. **/Sort Ratio** sorts the groups by their energy consumption in descending order.



```
; Prerequisite: trace data has been recorded.
; As a result, symbolic information is now stored in the symbol database.
; Allows you view and browse the symbol database:
sYmbol.Browse.Function

; Creates one group with two members: functions LED1 and LED2
; and assign color RED to the group named LED1+2.
Group.Create "LED1+2" Var.RANGE(LED1) Var.RANGE(LED2) /RED

; Second group.
Group.Create "LED3+5" Var.RANGE(LED3) Var.RANGE(LED5) /LIME

; Opens the analysis window.
ETA.STATistic.GROUP Ratio BAR.log Total Count ALL /Track /Sort Ratio

; Adds a third group.
Group.Create "LED7+8" Var.RANGE(LED7) Var.RANGE(LED8) /PURPLE

; The analysis window updates automatically to show the result.
```

Format: **ETA.STATistic.LINKage** *<address>* [*<list_items>* …] [**/***<option>*]

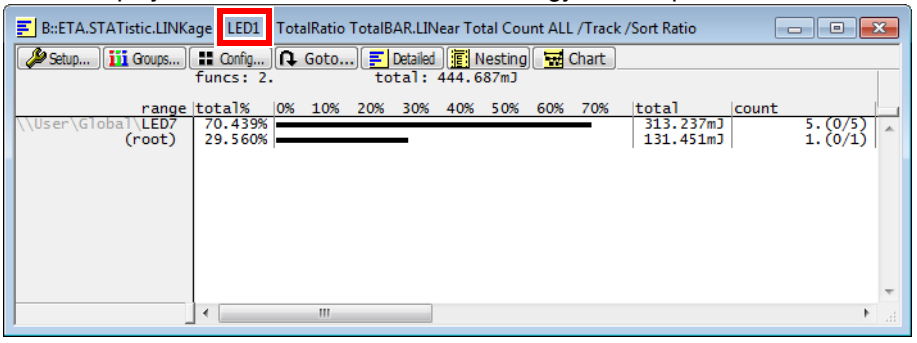
<format>: **DEFault** | **LEN** *<size>*
TimeAuto | **TimeFixed**

<list_items>: **DEFault** | **ALL**
NAME | **TASK**
Total | **TotalRatio** | **TotalBAR**[**.log** | **.LINear**]
Count | **MIN** | **MAX**
AVeRage
Internal | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio**
InternalBAR[**.log** | **.LINear**]
External | **EAVeRage** | **EMIN** | **EMAX**
ExternalINTR | **ExternalINTRMAX**
ExternalTASK | **ExternalTASKMAX** | **INTRCount** | **TASKCount**

<option>: **FILE** | **FlowTrace** | **BusTrace**
TASK | **SplitTASK** | **MergeTASK**
IncludeOwn | **IncludeTASK** | **IncludeINTR**
INTRROOT | **INTRTASK**
Accumulate
INCremental | **FULL**
CLOCKS
NoMerge
Sort *<item>*
Track

The call statements to one function from all other functions and the energy consumption are analyzed.

Example: The screenshot below shows that the function *LED1* has been called from function *LED7*, and **Count** displays the number of calls and the energy consumption.




```
ETA.STATistic.LINKage LED1 TotalRatio TotalBAR.LINear Total Count ALL /Sort Ratio
```

Format:	**ETA.STATistic.MODULE** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** *<size>* **TimeAuto** | **TimeFixed**
<list_items>:	**DEFault** | **ALL** **NAME** **CORE** **Total** | **TotalMIN** | **TotalMAX** **Ratio** | **RatioMIN** | **RatioMAX** **BAR**[**.log** | **.LINear**] **Count** | **CountRatio** | **CountBAR**[**.log** | **.LINear**] | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountALL** **InterVal** *<time>* **Filter** *<filter_item>* **Accumulate** **INCremental** | **FULL** **CLOCKS** **Sort** *<item>* **Track**

Displays the energy consumption of symbol modules. The list of loaded modules can be displayed with **sYmbol.List.Module**.

Format: **ETA.STATistic.ParentTREE** *<address>* [**%***<format>*] [*<list_items>* …] [**/***<option>*]

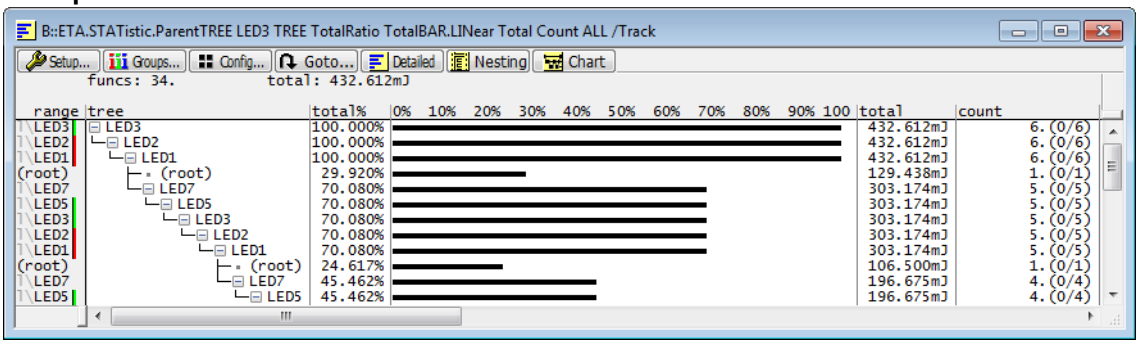
<format>: **DEFault** | **LEN** *<size>*
TimeAuto | **TimeFixed**

<list_items> **DEFault** | **ALL**
TREE
TASK
Total | **TotalRatio** | **TotalBAR**[**.log** | **.LINear**]
Count | **MIN** | **MAX**
AVeRage
Internal | **IAVeRage** | **IMIN** | **IMAX** | **InternalRatio**
InternalBAR[**.log** | **.LINear**]
External | **EAVeRage** | **EMIN** | **EMAX**
ExternalINTR | **ExternalINTRMAX**
INTRCount
ExternalTASK | **ExternalTASKMAX** | **TASKCount**

<option>: **FILE** | **FlowTrace** | **BusTrace**
TASK | **SplitTASK** | **MergeTASK**
IncludeOwn | **IncludeTASK** | **IncludeINTR**
INTRROOT | **INTRTASK**
Accumulate
INCremental | **FULL**
CLOCKS
NoMerge
Sort *<item>*
Track

Displays the energy consumption of a function dependant on the caller.

Example:




```
ETA.STATistic.ParentTREE LED3 TREE TotalRatio TotalBAR.LINear Total Count ALL /Track
```

Format:	**ETA.STATistic.PROGRAMs** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** *<size>* **TimeAuto** | **TimeFixed**
<list_items>:	**DEFault** | **ALL** **NAME** **CORE** **Total** | **TotalMIN** | **TotalMAX** **Ratio** | **RatioMIN** | **RatioMAX** **BAR**[**.log** | **.LINear**] **Count** | **CountRatio** | **CountBAR**[**.log** | **.LINear**] | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **BEFORE** | **AFTER** **CountChange** | **CountFirst** | **CountALL** **InterVal** *<time>* **Filter** *<filter_item>* **Accumulate** **INCremental** | **FULL** **CLOCKS** **Sort** *<item>* **Track**

Displays the energy consumption of loaded object file programs. The loaded programs can be displayed with the command **sYmbol.Browse ***.

Format: **ETA.STATistic.sYmbol** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

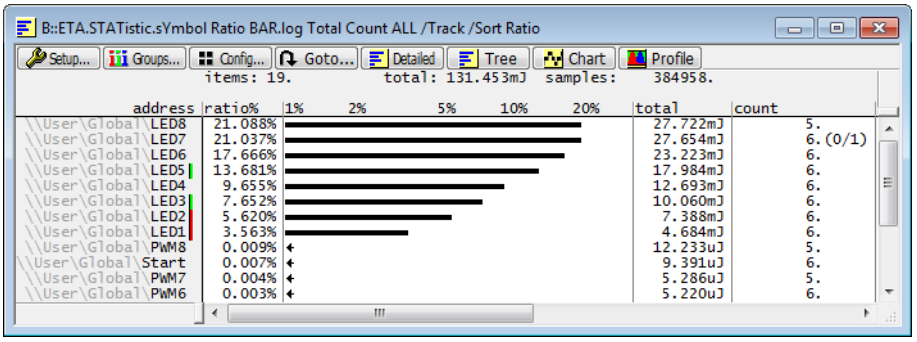
<format>: **DEFault** | **LEN** *<size>*
TimeAuto | **TimeFixed**

<list_items>: **DEFault** | **ALL**
NAME
CORE
Total | **TotalMIN** | **TotalMAX**
Ratio | **RatioMIN** | **RatioMAX**
BAR[**.log** | **.LINear**]
Count | **CountRatio** | **CountBAR**[**.log** | **.LINear**] | **CountMIN** | **CountMAX**
MIN | **MAX** | **AVeRage**

<option>: **FILE** | **FlowTrace** | **BusTrace**
BEFORE | **AFTER**
CountChange | **CountFirst** | **CountALL**
InterVal *<time>*
Filter *<filter_item>*
Accumulate
INCremental | **FULL**
CLOCKS
Sort *<item>*
Track

Displays a statistical analysis of the energy consumption for each function.

Example: In this screenshot, the functions are sorted by energy consumption in descending order. **BAR** illustrates the percentage values as bar chart, and **Count** displays the number of calls for each group. **/Sort Ratio** sorts the symbols by their energy consumption in descending order.




```
ETA.STATistic.sYmbol Ratio BAR.log Total Count ALL /Track /Sort Ratio
```

Displays a statistical analysis of the energy consumption for nesting functions as tree.

Format:	**ETA.STATistic.TASK** [**%**<*format*>] [<*list_items*> …] [**/**<*option*>]
<*format*>:	**DEFault** | **LEN** <*size*> **TimeAuto** | **TimeFixed**
<*list_items*>:	**DEFault** | **ALL** **Total** | **TotalMIN** | **TotalMAX** **Ratio** | **RatioMIN** | **RatioMAX** **BAR**[**.log** | **.LINear**] **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<*option*>:	**FILE** | **FlowTrace** | **BusTrace** **InterVal** <*time*> **Accumulate** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** <*item*> **Track**

Measures the energy consumed by different tasks and the number of calls. This feature is only available if TRACE32 has been set for OS-aware debugging.

Format:	**ETA.STATistic.TASKINFO** [**%**<*format*>] [<*list_items*> …] [**/**<*option*>]
<*format*>:	**DEFault** | **LEN** <*size*> **TimeAuto** | **TimeFixed**
<*list_items*>:	**DEFault** | **ALL** **Total** | **TotalMIN** | **TotalMAX** **Ratio** | **RatioMIN** | **RatioMAX** **BAR**[**.log** | **.LINear**] **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<*option*>:	**FILE** | **FlowTrace** | **BusTrace** **InterVal** <*time*> **Accumulate** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** <*item*> **Track**

Displays a statistic table of the energy consumption for special values written to the Context ID register for ETM trace. Please refer to **<trace>.STATistic.TASKINFO** for more information.

Format: **ETA.STATistic.TASKINTR** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

<format>: **DEFault** | **LEN** *<size>*
TimeAuto | **TimeFixed**

<list_items>: **DEFault** | **ALL**
Total | **TotalMIN** | **TotalMAX**
Ratio | **RatioMIN** | **RatioMAX**
BAR[**.log** | **.LINear**] | **Count** | **CountMIN** | **CountMAX**
MIN | **MAX** | **AVeRage**

<option>: **FILE** | **FlowTrace** | **BusTrace**
InterVal *<time>*
Accumulate
INCremental | **FULL**
CLOCKS
NoMerge
Sort *<item>*
Track

Performs a statistical analysis of the energy consumed by interrupt service routines. This feature can only be used if ISR2 can be traced based on the information provided by the ORTI file. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

Format:	**ETA.STATistic.TASKKernel** [**%***<format>*] [*<list_items>* …] [**/***<option>*]
<format>:	**DEFault** | **LEN** *<size>* **TimeAuto** | **TimeFixed**
<list_items>:	**DEFault** | **ALL** **Total** | **TotalMIN** | **TotalMAX** **Ratio** | **RatioMIN** | **RatioMAX** **BAR**[**.log** | **.LINear**] | **Count** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **InterVal** *<time>* **Accumulate** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **Track**

Performs a statistical analysis of the energy consumed by tasks and kernel run-times. Kernel entry and exit must be marked with **sYmbol.MARKER.Create** commands. This feature is only available if TRACE32 has been set for OS-aware debugging.

Format: **ETA.STATistic.TASKORINTERRUPT** [**%**<*format*>] [<*list_items*> …] [**/**<*option*>]

<*format*>: **DEFault** | **LEN** <*size*>
TimeAuto | **TimeFixed**

<*list_items*>: **DEFault** | **ALL**
Total | **TotalMIN** | **TotalMAX**
Ratio | **RatioMIN** | **RatioMAX**
BAR[**.log** | **.LINear**]
Count | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX**
MIN | **MAX** | **AVeRage**

<*option*>: **FILE** | **FlowTrace** | **BusTrace**
InterVal <*time*>
Accumulate
INCremental | **FULL**
CLOCKS
NoMerge
Sort <*item*>
Track

Measures the energy consumed by different tasks and interrupts and the number of calls. This feature is only available if TRACE32 has been set for OS-aware debugging.

Format:	**ETA.STATistic.TASKSRV** [**%***<formats>*] [*<list_items>*] [**/***<options>*]
<formats>:	**DEFault** | **LEN** *<size>* **TimeAuto** | **TimeFixed**
<list_items>:	**DEFault** | **ALL** **Total** | **TotalMIN** | **TotalMAX** **Ratio** | **RatioMIN RatioMAX** **BAR**[**.log** | **.LINear**] | **Count** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<option>:	**FILE** | **FlowTrace** | **BusTrace** **InterVal** *<time>* **Accumulate** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** *<item>* **Track**

Measures the energy consumed by OS service routines and the number of calls. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

Format:	**ETA.STATistic.TREE** [**%**<*format*>] [<*list_items*> …] [**/**<*option*>]
<*format*>:	**DEFault** | **LEN** <*size*> **TimeAuto** | **TimeFixed**
<*list_items*>:	**DEFault** | **ALL** **Total** | **TotalMIN** | **TotalMAX** **Ratio** | **RatioMIN** | **RatioMAX** **BAR**[**.log** | **.LINear**] **Count** | **CountRatio** | **CountBAR** | **CountMIN** | **CountMAX** **MIN** | **MAX** | **AVeRage**
<*option*>:	**FILE** | **FlowTrace** | **BusTrace** **InterVal** <*time*> **Accumulate** **INCremental** | **FULL** **CLOCKS** **NoMerge** **Sort** <*item*> **Track**

Displays a statistical analysis of the energy consumption for nesting functions as tree.

ETM

Embedded Trace Macrocell command group.

- Refer to **"Arm ETM Trace"** (trace_arm_etm.pdf) for a description of the ETM commands which are specific to the Arm/Cortex ETM/PTM
- Refer to **"Hexagon-ETM Trace"** (trace_hexagon_etm.pdf) for a description of the Hexagon specific ETM commands.
- Refer to **"CEVA-X Debugger and Trace"** (debugger_cevax.pdf) for a description of the CEVA-X specific ETM commands. Generic commands are described in **"Arm ETM Trace"** (trace_arm_etm.pdf).
- Refer to **"CEVA-Oak/Teak/TeakLite Debugger and Trace"** (debugger_oak.pdf) for a description of the TeakLite specific ETM commands. Generic commands are described in **"Arm ETM Trace"** (trace_arm_etm.pdf).

EVENTS

The EVENTS command group allow to analyze events against time in the program flow.

Example: display the branch mis-predictions for ETMv4

```
Break.CLEAR
Break.ReProgram
(
  IF BMC(bpmis)
    EVENT 0
)
Go
WAIT 3.s
Break
EVENTS.PROfileChart
```

EVENTS.List — List the events trace data

[build 135171 - DVD 09/2021]

Format:	**EVENTS.List** [*<record>* | *<record_range>* | *<time>* | *<time_range>* | *<bookmark>*] [*<items>*...] | [**/***<options>*]
<items>:	**%***<formats>* **DEFault** | **ALL** | **CPU** | **LINE** | **PORT** **Run** **CYcle** | **Data**[**.***<subitem>* | **BDATA** | **List**[**.***<subitem>*] **Address** | **BAddress** | **FAddress** |**sYmbol** | **sYmbolN** | **PAddress** | **PsYmbol** | **Var** **TIme**[**.***<subitem>*] | **CLOCKS**[**.***<subitem>*] **FUNC** | **FUNCR** | **FUNCVar** | **IGNORE** **LeVel** | **MARK**[**.***<marker>* **SPARE**
<formats>:	**DEFault** | **LEN** | **Timing** | **HighLow** | **01** | **Hex** | **Decimal** | **BINary** | **Ascii** **Signed** | **Unsigned** | **TimeAuto** | **TimeFixed**
<options>:	**FILE** | **FlowTrace** | **BusTrace** | **MACHINE** | **NorthWestGravity** | **Mark** | **Track**

Opens a window showing the recorded trace data starting at the specified *<record>* or for a range of trace records *<record_range>* (e.g. `(-10000.)--(-2000.)`).

The columns of the window can be defined using the *<list_items>*. The order of the columns in the window is according to the order of the *<list_item>* parameters given.

<options> For a detailed description of all other parameters and options, refer to the **<trace>.List** command.

EVENTS.ListNesting Show program nesting

[build 135171 - DVD 09/2021]

Format: **EVENTS.ListNesting** [*<record>* | *<record_range>* | *<time>* | *<time_range>* | *<bookmark>*] [*<items>* …] [**/***<options>*]

<items>:
%*<formats>*
DEFault | **ALL** | **CPU** | **LINE** | **PORT**
Run
CYcle | **Data**[**.***<subitem>* | **BDATA** | **List**[**.***<subitem>*]
Address | **BAddress** | **FAddress** |**sYmbol** | **sYmbolN** | **PAddress** | **PsYmbol** | **Var**
TIme[**.***<subitem>*] | **CLOCKS**[**.***<subitem>*]
FUNC | **FUNCR** | **FUNCVar** | **IGNORE**
LeVel | **MARK**[**.***<marker>*
SPARE

<formats>: **DEFault** | **LEN** | **Timing** | **HighLow** | **01** | **Hex** | **Decimal** | **BINary** | **Ascii** **Signed** | **Unsigned** | **TimeAuto** | **TimeFixed**

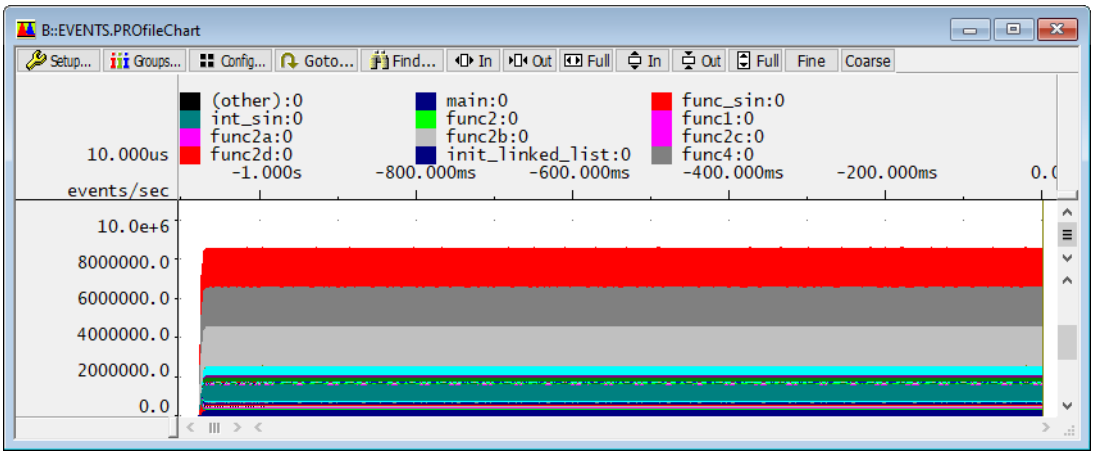
<options>: **FILE** | **FlowTrace** | **BusTrace** | **MACHINE** | **NorthWestGravity** | **Mark** | **Track**

Shows program nesting with event count.

<options> For a detailed description of all other parameters and options, refer to the **<trace>.List** command.

EVENTS.PROfileChart — Profile chart for events

The **EVENTS.PROfileChart** command group display event distributions versus time graphically as color chart with fixed time intervals. The result is a stacked graph where the total number of events/sec represent the sum of the events at that time. Please refer for more information to **<trace>.PROfileChart**.



See also

- <trace>.PROfileChart

EVENTS.PROfileChart.AddressGROUP — Event profile chart for groups

Format:	**EVENTS.PROfileChart.AddressGROUP** [*<trace_area>*] [**/***<option>*]
<trace_area>:	*<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for address **groups** created with the command **GROUP.Create**. The results include groups for both program and data.

<trace_area>	Refer to **Trace.PROfileChart.AddressGROUP**.
<option>	

See also

- EVENTS.PROfileChart.GROUP

EVENTS.PROfileChart.ALL Event profile chart for program run

Format: **EVENTS.PROfileChart.ALL** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for the program execution.

EVENTS.PROfileChart.DatasYmbol Symbolic statistics for data as a chart

Format: **EVENTS.PROfileChart.DatasYmbol** [<trace_area>] [/<option>]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Analyzes data symbols and plots the number of events/s as a graphical chart.

<trace_area> Refer to **Trace.PROfileChart.DatasYmbol**.
<option>

EVENTS.PROfileChart.DistriB Distribution statistical analysis

Format: **EVENTS.PROfileChart.DistriB** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile chart for or the statistical distribution of a selected item or based on the symbolic addresses if no item is specified.

<trace_area> Refer to **Trace.PROfileSTATistic.DistriB**.
<option>

Format: **EVENTS.PROfileChart.GROUP** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for **groups** created with the command **GROUP.Create**. The results only include groups within the program range. Groups for data addresses are not included.

<trace_area> Refer to **Trace.PROfileChart.GROUP**.
<option>

See also

- EVENTS.PROfileChart.AddressGROUP

EVENTS.PROfileChart.Line Events per high-level language line graphically

Format: **EVENTS.PROfileChart.Line** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for all executed high-level source code lines.

<trace_area> Refer to **Trace.PROfileChart.Line**.
<option>

EVENTS.PROfileChart.MODULE Event profile chart for modules

Format: **EVENTS.PROfileChart.MODULE** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for symbol modules. The list of loaded modules can be displayed with **sYmbol.List.Module**.

<trace_area> Refer to **Trace.PROfileChart.MODULE**.
<option>

EVENTS.PROfileChart.PROGRAM Event profile chart for programs

Format: **EVENTS.PROfileChart.MODULE** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for loaded object file programs. The loaded programs can be displayed with the command **sYmbol.Browse ***.

EVENTS.PROfileChart.sYmbol Event for all program symbols graphically

Format: **EVENTS.PROfileChart.sYmbol** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for all program symbols.

<trace_area> Refer to **Trace.PROfileChart.sYmbol**.
<option>

Format: **EVENTS.PROfileChart.TASK** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for all tasks. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileChart.TASK**.

<option>

Format: **EVENTS.PROfileChart.TASKINFO** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for special values written to the Context ID register (ETM trace). Please refer to **Trace.PROfileChart.TASKINFO** for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKINFO**.
<option>

EVENTS.PROfileChart.TASKINTR Events profile chart for ISR2 (ORTI)

Format: **EVENTS.PROfileChart.TASKINTR** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for ORTI based ISR2. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKINTR**.
<option>

Format: **EVENTS.PROfileChart.TASKKernel** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Refines the command **EVENTS.PROfileChart.TASK** for RTOS systems that don't assign a task ID to the kernel. Refer to **Trace.STATistic.TASKKernel** for more information. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileChart.TASKKernel**.
<option>

EVENTS.PROfileChart.TASKORINTERRUPT EVENTS per task/interrupt

Format: **EVENTS.PROfileChart.TASKORINTERRUPT** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for all tasks and interrupts. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileChart.TASKORINTERRUPT**.
<option>

Format: **EVENTS.PROfileChart.TASKSRV** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for OS service routines. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKSRV**.
<option>

EVENTS.PROfileChart.TASKVSINTR Events for task-related interrupts

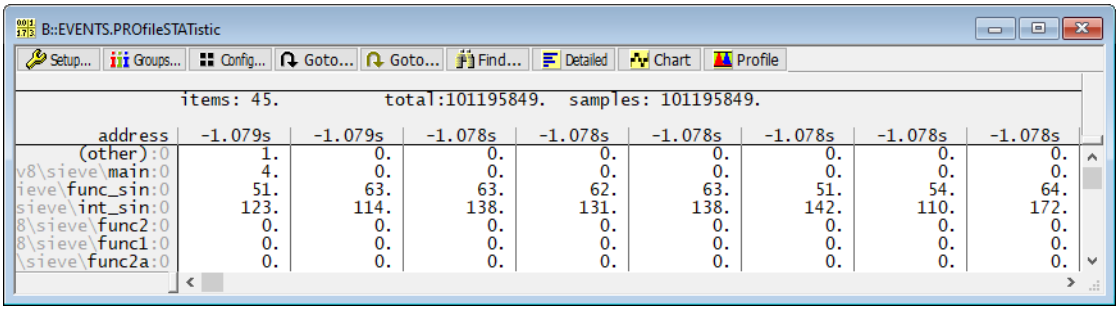
Format: **EVENTS.PROfileChart.TASKVSINTR** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

The number of events/s is displayed as graphical profile chart for task-related interrupt service routines. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileChart.TASKINTR**.
<option>

The command group **EVENTS.PROfileSTATistic** shows the results of numerical interval analysis in tabular format for events.



See also

- <trace>.PROfileSTATistic

EVENTS.PROfileSTATistic.Address Events per address as profile statistic

Format: **EVENTS.PROfileSTATistic.Address** [*<trace_area>*] *<address1>* [*<address2>* ...] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic for addresses.

<trace_area> Refer to **Trace.PROfileSTATistic.Address**.
<option>

EVENTS.PROfileSTATistic.AddressGROUP Events per address GROUP

Format: **EVENTS.PROfileSTATistic.AddressGROUP** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic for address **groups**. The results include groups for both program and data addresses.

<trace_area> Refer to **Trace.PROfileSTATistic.GROUP**.
<option>

See also

- ETA.PROfileSTATistic.GROUP

EVENTS.PROfileSTATistic.ALL Event profile statistic for program run

Format: **EVENTS.PROfileSTATistic.ALL** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic for the program run.

Format: **EVENTS.PROfileSTATistic.DatasYmbol** [<trace_area>] [/<option>]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Analyzes data symbols and displays the number of events/s as a profile statistic in tabular format.

<trace_area> Refer to **Trace.PROfileSTATistic.DatasYmbol**.
<option>

EVENTS.PROfileSTATistic.DistriB Distribution statistical analysis

Format: **EVENTS.PROfileSTATistic.DistriB** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic for or the statistical distribution of a selected item or based on the symbolic addresses if no item is specified.

<trace_area> Refer to **Trace.PROfileSTATistic.DistriB**.
<option>

EVENTS.PROfileSTATistic.GROUP Events per GROUP as profile statistic

Format: **EVENTS.PROfileSTATistic.GROUP** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic for **groups**. The results only include groups within the program range. Groups for data addresses are not included.

<trace_area> Refer to **Trace.PROfileSTATistic.GROUP**.
<option>

EVENTS.PROfileSTATistic.INTERRUPT Events per interrupt as table

Format: **EVENTS.PROfileSTATistic.INTERRUPT** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for interrupts. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.INTERRUPT**.
<option>

EVENTS.PROfileSTATistic.Line Events per high-level language line as table

Format: **EVENTS.PROfileSTATistic.Line** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for high-level language lines.

EVENTS.PROfileSTATistic.MODULE Events per module as profile statistic

Format: **EVENTS.PROfileSTATistic.MODULE** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for symbol modules. The list of loaded modules can be displayed with **sYmbol.List.Module**.

<trace_area> Refer to **Trace.PROfileSTATistic.MODULE**.
<option>

EVENTS.PROfileSTATistic.PROGRAM Events per program

Format: **EVENTS.PROfileSTATistic.PROGRAM** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for loaded object file programs. The loaded programs can be displayed with the command **sYmbol.Browse ***.

<trace_area> Refer to **Trace.PROfileSTATistic.PROGRAM**.
<option>

EVENTS.PROfileSTATistic.RUNNABLE Events per runnable as table

Format: **Events.PROfileSTATistic.RUNNABLE** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for AUTOSAR Runnables. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.RUNNABLE**.
<option>

EVENTS.PROfileSTATistic.sYmbol Events for all program symbols as table

Format: **EVENTS.PROfileSTATistic.sYmbol** [*<trace_area>*] [**/***<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for all program symbols.

<trace_area> Refer to **Trace.PROfileSTATistic.sYmbol**.
<option>

EVENTS.PROfileSTATistic.TASK Events per task as table

Format: **EVENTS.PROfileSTATistic.TASK** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for tasks. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.TASK**.
<option>

EVENTS.PROfileSTATistic.TASKINFO Events per context ID message

Format: **EVENTS.PROfileSTATistic.TASK** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for special values written to the Context ID register (ETM trace). Please refer to **Trace.PROfileSTATistic.TASKINFO** for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKINFO**.
<option>

EVENTS.PROfileSTATistic.TASKINTR Events per ISR2 (ORTI) as table

Format: **EVENTS.PROfileSTATistic.TASKINTR** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for ORTI based ISR2. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKINTR**.
<option>

EVENTS.PROfileSTATistic.TASKKernel Events per task as table

Format: **EVENTS.PROfileSTATistic.TASKKernel** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Refines the command **EVENTS.PROfileSTATistic.TASK** for RTOS systems that don't assign a task ID to the kernel. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKKernel**.
<option>

EVENTS.PROfileSTATistic.TASKORINTERRUPT Events per task as table

Format: **EVENTS.PROfileSTATistic.TASKORINTERRUPT** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for tasks and interrupts. This feature is only available if TRACE32 has been set for OS-aware debugging.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKORINTERRUPT**.
<option>

EVENTS.PROfileSTATistic.TASKSRV Events per OS service routine

Format: **EVENTS.PROfileSTATistic.TASKSRV** [*<trace_area>*] [*/<option>*]

<trace_area>: *<trace_bookmark>* | *<record>* | *<record_range>* | *<time>* | *<time_range>* [*<time_scale>*]

Displays events as profile statistic table for OS service routines. This feature is only available if an OSEK/ORTI system is used and if the OS Awareness is configured with the **TASK.ORTI** command. Please refer to **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf) for more information.

<trace_area> Refer to **Trace.PROfileSTATistic.TASKSRV**.
<option>

EVENTS.STATistic — Statistic for events

The **EVENTS.STATistic** commands can be used for statistical analysis for the events sampled in the trace.

See also

- <trace>.STATistic

EVENTS.STATistic.ChildTREE — Events for the callee context of a function

Format: **EVENTS.STATistic.ChildTREE** *<address>*

Displays events for all functions called by the specified function numerically as call tree.

EVENTS.STATistic.Func — Events for functions numerically

Format: **EVENTS.STATistic.Func** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Displays events for functions numerically.

EVENTS.STATistic.GROUP — Events statistic for groups

Format: **EVENTS.STATistic.GROUP** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Event statistic for **groups**. The results only include groups within the program range. Groups for data addresses are not included.

EVENTS.STATistic.LINKage — Per caller event statistic of function

Format: **EVENTS.STATistic.LINKage** [**%***<format>*] [*<items>* …] [**/***<option>*]

Performs a function event statistic for a single function itemized by its callers.

EVENTS.STATistic.MODULE Events for modules numerically

Format: **EVENTS.STATistic.MODULE** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Displays events for symbol modules numerically. The list of loaded modules can be displayed with **sYmbol.List.Module**.

EVENTS.STATistic.ParentTREE Event statistic for call context of a function

Format: **EVENTS.STATistic.ParentTREE** *<address>*

Show call tree and events of all callers of the specified function. The function is specified by its start *<address>*.

EVENTS.STATistic.PROGRAM Events for programs numerically

Format: **EVENTS.STATistic.PROGRAM** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Displays events for loaded object file programs numerically. The loaded programs can be displayed with the command **sYmbol.Browse ***.

EVENTS.STATistic.sYmbol Events for all program symbols numerically

Format: **EVENTS.STATistic.sYmbol** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Events for all program symbols numerically.

EVENTS.STATistic.TASK Events per task numerically

Format: **EVENTS.STATistic.TASK** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Events per task numerically. This feature is only available if TRACE32 has been set for OS-aware debugging.

EVENTS.STATistic.TASKINFO Events per context ID message numerically

Format: **EVENTS.STATistic.TASKINFO** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Events are displayed numerically for special values written to the Context ID register (ETM trace). Please refer to **Trace.STATistic.TASKINFO** for more information

EVENTS.STATistic.TASKINTR Events per ISR2 numerically

Format: **EVENTS.STATistic.TASKINTR** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Events per ORTI based ISR2 numerically. This feature is only available, if an OSEK/ORTI system is used, and if the OS Awareness is configured with the **TASK.ORTI** command.

EVENTS.STATistic.TASKKernel Events task analysis with kernel markers

Format: **EVENTS.STATistic.TASKernel** [**%***<format>*] [*<list_items>* …] [**/***<option>*]

Refines the command **EVENTS.STATistic.TASK** for RTOS systems that don't assign a task ID to the kernel. This feature is only available if TRACE32 has been set for OS-aware debugging.

Format: **EVENTS.STATistic.TASKSRV** [**%**<*format*>] [<*list_items*> …] [**/**<*option*>]

Events per OS service routine numerically. This feature is only available, if an OSEK/ORTI system is used, and if the OS Awareness is configured with the **TASK.ORTI** command.

EVENTS.STATistic.TREE Tree display of nesting functions with events

Format: **EVENTS.STATistic.TREE** [**%**<*format*>] [<*list_items*> …] [**/**<*option*>]

Displays a graphical tree of the function nesting with events per function results.

EXTension — Extend the TRACE32 debugger with custom features

The **EXTension** command group enables you to extend the TRACE32 debugger with custom features. For example, you can create new commands, windows, PRACTICE functions for the debugger, as well as OS Awarenesses and Hypervisor Awarenesses. To create such an extension, you need the “TRACE32 Extension Development Kit” (EDK) from Lauterbach. The EDK contains a guide with detailed information about extension definitions for TRACE32.

See also

- EXTension.ACCESS
- EXTension.CONFIG
- EXTension.DEBUG
- EXTension.DELETE
- EXTension.LOAD
- EXTension.MaxVSize
- EXTension.RESet
- EXTension.SETDIR
- EXTension.TimeOut
- TASK

EXTension.ACCESS — Control memory access

Format: **EXTension.ACCESS** [*<class>*] [*<extension_file>*]

Defines the memory access class used by commands and functions of an extension.

An extension may access the target memory. If the access class is set to E:, the debugger uses emulation memory access to read the memory (e.g. emulation memory, shadow memory or pseudo-dual-port access). If set to C:, the debugger uses CPU access. If the appropriate access is not possible, an extension window is temporarily frozen.

EXTension.ACCESS without parameter enables the default mode, which uses E:, if the application is running, and C: if the application is stopped.

Please see also the target-specific guides for a description of E: and C: to your processor.

- **Processor Architecture Manuals**

The second parameter can be used to specify the extension definition file or extension name in case multiple extensions are loaded.

See also

- EXTension
- EXTension.LOAD

EXTension.CONFIG Configure extension

Format:	**EXTension.CONFIG** *<extension_file>* *<args>* [*/<option>*]
<option>:	**ACCESS** *<class>*

Configures the extension using a given extension definition file. Previously loaded extensions are removed.

<option>, etc.	For descriptions, see **EXTension.LOAD**.

See also

- EXTension
- EXTension.LOAD

EXTension.DEBUG Debug outputs of extension

Format:	**EXTension.DEBUG** *<value>*

Switches on and off debug outputs of an extension.

See also

- EXTension
- EXTension.LOAD

EXTension.DELETE Delete loaded extension

Format:	**EXTension.DELETE** *<extension_file>*

Removes the extension loaded with **EXTension.LOAD**.

See also

- EXTension
- EXTension.LOAD

[Examples]

Format:	**EXTension.LOAD** *<extension_file>* *<args>* [*/<option>*]
<option>	**ACCESS** *<class>* **Machine** *<machine_id>* **NAME** *<name>*

Loads an extension using a given extension definition file. **EXTension.LOAD** allows to load more than one extension (in contrast to **EXTension.CONFIG**).

<extension_file>	File name of the extension definition file.
<args>	All other arguments are interpreted by the extension definition file.
ACCESS	The **ACCESS** option defines the memory access class used by the commands and functions of an extension. See **EXTension.ACCESS**
Machine	In hypervisor environments, specify the machine ID to which this extension belongs.
NAME	Assigns a user-defined name to the extension. Without the **NAME** option, the base name of the extension definition file will be used.

Examples: These script lines show how to load the Hypervisor Awareness and several OS Awarenesses.

```
PRINT "Loading Xen awareness..."
EXTension.LOAD ~~/demo/arm/kernel/xen/xen.t32 \
/Machine 0 /NAME "Xen"
…

PRINT "Loading Linux awareness on Dom0..."
EXTension.LOAD ~~/demo/arm/kernel/linux/linux-3.x/linux3.t32 \
/Machine 1 /NAME "Dom0"
…

PRINT "configuring Linux awareness..."
EXTension.LOAD ~~/demo/arm/kernel/linux/linux-3.x/linux3.t32 \
/Machine 2 /NAME "Linux"
…

PRINT "configuring FreeRTOS awareness..."
EXTension.LOAD ~~/demo/arm/kernel/freertos/freertos.t32 \
/Machine 3 /NAME "FreeRTOS"
```

See also

- EXTension
- EXTension.ACCESS
- EXTension.CONFIG
- EXTension.DEBUG
- EXTension.DELETE
- EXTension.MaxVSize
- EXTension.RESet
- EXTension.SETDIR
- EXTension.TimeOut
- TASK.CONFIG

EXTension.MaxVSize — Set max. vertical size of extension windows

Format: **EXTension.MaxVSize** *<value>*

Sets the maximum vertical size of extension windows. Default value is 2000 (dec).

See also

- EXTension
- EXTension.LOAD

EXTension.ORTI.Delete — Unload ORTI file

[build 143677 - DVD 02/2022]

Format: **EXTension.ORTI.Delete** *<file>* | *<"name">* | *<machine_id>*

Unload an ORTI file previously loaded using **EXT.ORTI.LOAD**.

<file>	Filepath to ORTI file.
<name>	Name of ORTI file. Compare option /NAME <> of EXT.ORTI.LOAD. E.g. `EXT.ORTI.LOAD <file> /NAME "OS1"`
<machine_id>	Machine ID the ORTI file was loaded to.

Examples:

```
; Example <file>
EXTension.ORTI.LOAD orti/v1/os.orti
CD orti
EXTension.ORTI.Delete v1/os.orti

; Example <name>
EXTension.ORTI.LOAD os.orti /NAME "OS1"
EXTension.ORTI.Delete "OS1"

; Example <machine>
; Example uses machine 5 for demonstration
SYStem.Option MACHINESPACES ON
Data.LOAD.Elf symbols.elf 0x5:::0x0 /NoCODE
EXTension.ORTI.LOAD os.orti /NAME "OS1" /Machine 5.
EXTension.ORTI.Delete 5.
```

EXTension.ORTI.LOAD — Load ORTI file

[build 143677 - DVD 02/2022]

Format:	**EXTension.ORTI.LOAD** *<file>* [**/***<option>*]
<option>	**Machine** *<machine_id>* **NAME** *<name>*

Loads an ORTI file belonging to an AutoSAR/OSEK operating system for OS aware debugging. **EXTension.ORTI.LOAD** allows to load more than one ORTI file (in contrast to **TASK.ORTI**) and is supposed to be used in hypervisor and iAMP debug scenarios.For a detailed description, please refer to the chapter **"OS Awareness Manual OSEK/ORTI"** (rtos_orti.pdf).

<file>	Filepath to ORTI file.
Machine	In hypervisor environments, specify the machine ID to which this extension belongs.
NAME	Assigns a user-defined name to the extension. Without the **NAME** option, the base name of the extension definition file will be used.

Example:

```
; Setup two AutoSAR/OSEK OS aware machines.
SYStem.Option MACHINESPACES ON
Data.LOAD.Elf os1/os.elf 0x1:::0x0 /NoCODE
EXTension.ORTI.LOAD os1/os.orti /NAME "OS1" /Machine 1.
Data.LOAD.Elf os2/os.elf 0x2:::0x0 /NoCODE
EXTension.ORTI.LOAD os2/os.orti /NAME "OS2" /Machine 2.
; This creates two menu entries
; - OS1
; - OS2
; Commands are mapped as follows:
; - EXT.OS1.D<...>
; - EXT.OS2.D<...>
; Functions are mapped as follows:
; - EXT.OS1.CONFIG(...)
; - EXT.OS2.CONFIG(...)
; Exact features are highly dependent on the ORTI file.
```

EXTension.ORTI.RESet — Unload all ORTI files

[build 143677 - DVD 02/2022]

Format:	**EXTension.ORTI.RESet**

Unloads all ORTI files loaded.

EXTension.RESet — Reset extension definition

Format:	**EXTension.RESet**

The extension configuration is cleared, all additional commands and features are removed.

See also

- EXTension
- EXTension.LOAD

Format: **EXTension.SETDIR** *<dir>* [*<extension_file>*]

An extension can query the directory of the extension files to refer to other files and call scripts from this directory. Use this command to set the extension directory if necessary. The second parameter can be used to specify the extension definition file or extension name in case multiple extensions are loaded.

See also

- EXTension
- EXTension.LOAD

EXTension.TimeOut Set timeout of extension

Format: **EXTension.TimeOut** *<value>*

The extension processing is interrupted after a timeout period to prevent infinite loops. This command sets the timeout in milliseconds. Default value is 10000 (dec).

See also

- EXTension
- EXTension.LOAD