### General Commands

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TASK</strong></td>
<td>OS Awareness for TRACE32</td>
</tr>
<tr>
<td>TASK</td>
<td>Control memory access</td>
</tr>
<tr>
<td>TASK.ACCESS</td>
<td>Deprecated</td>
</tr>
<tr>
<td>TASK.Break</td>
<td>Reread task list</td>
</tr>
<tr>
<td>TASK.CACHEFLUSH</td>
<td>Configure OS Awareness</td>
</tr>
<tr>
<td>TASK.COPYDOWN</td>
<td>Copy file from host into target</td>
</tr>
<tr>
<td>TASK.COPYUP</td>
<td>Copy file from target into host</td>
</tr>
<tr>
<td>TASK.CreateExtraID</td>
<td>Create a virtual task</td>
</tr>
<tr>
<td>TASK.CreateID</td>
<td>Create virtual task</td>
</tr>
<tr>
<td>TASK.Debug</td>
<td>Deprecated</td>
</tr>
<tr>
<td>TASK.DeleteID</td>
<td>Delete virtual task</td>
</tr>
<tr>
<td>TASK.DETACH</td>
<td>Detach from task</td>
</tr>
<tr>
<td>TASK.INSTALL</td>
<td>Deprecated</td>
</tr>
<tr>
<td>TASK.KILL</td>
<td>End task</td>
</tr>
<tr>
<td>TASK.List</td>
<td>Information about tasks</td>
</tr>
<tr>
<td>TASK.List.MACHINES</td>
<td>List machines</td>
</tr>
<tr>
<td>TASK.List.SPACES</td>
<td>List MMU spaces</td>
</tr>
<tr>
<td>TASK.List.tasks</td>
<td>List all running tasks</td>
</tr>
<tr>
<td>TASK.List.TREE</td>
<td>Display tasks in a tree structure</td>
</tr>
<tr>
<td>TASK.ListID</td>
<td>List virtual tasks</td>
</tr>
<tr>
<td>TASK.MTXTRC</td>
<td>Internal</td>
</tr>
<tr>
<td>TASK.NAME</td>
<td>Translation of task magic number to task name</td>
</tr>
<tr>
<td>TASK.NAME.DELETE</td>
<td>Delete a task name table entry</td>
</tr>
<tr>
<td>TASK.NAME.RESet</td>
<td>Reset task name table</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>TASK.NAME.Set</td>
<td>Set a task name table entry</td>
</tr>
<tr>
<td>TASK.NAME.view</td>
<td>Show task name translation table</td>
</tr>
<tr>
<td>TASK.NoBreak</td>
<td>Deprecated</td>
</tr>
<tr>
<td>TASK.NoDebug</td>
<td>Deprecated</td>
</tr>
<tr>
<td>TASK.NoStop</td>
<td>Deprecated</td>
</tr>
<tr>
<td>TASK.OFF</td>
<td>Deprecated</td>
</tr>
<tr>
<td>TASK.ON</td>
<td>Deprecated</td>
</tr>
<tr>
<td>TASK.ORTI</td>
<td>AUTOSAR/OSEK support</td>
</tr>
<tr>
<td>TASK.ORTI.CPU</td>
<td>Set OSEK SMP CPU number</td>
</tr>
<tr>
<td>TASK.ORTI.load</td>
<td>Configure OS Awareness for OSEK/ORTI</td>
</tr>
<tr>
<td>TASK.ORTI.NOSTACK</td>
<td>Exclude an ORTI task from stack evaluation</td>
</tr>
<tr>
<td>TASK.RESet</td>
<td>Reset OS Awareness</td>
</tr>
<tr>
<td>TASK.RUN</td>
<td>Load task</td>
</tr>
<tr>
<td>TASK.select</td>
<td>Display context of specified task</td>
</tr>
<tr>
<td>TASK.SETDIR</td>
<td>Set the awareness directory</td>
</tr>
<tr>
<td>TASK.STack</td>
<td>Stack usage coverage</td>
</tr>
<tr>
<td>TASK.STack.ADD</td>
<td>Add stack space coverage</td>
</tr>
<tr>
<td>TASK.STack.DIRECTION</td>
<td>Define stack growth direction</td>
</tr>
<tr>
<td>TASK.STack.Init</td>
<td>Initialize unused stack space</td>
</tr>
<tr>
<td>TASK.STack.PATTERN</td>
<td>Define stack check pattern</td>
</tr>
<tr>
<td>TASK.STack.PATTERNGAP</td>
<td>Define check pattern gap</td>
</tr>
<tr>
<td>TASK.STack.REMove</td>
<td>Remove stack space coverage</td>
</tr>
<tr>
<td>TASK.STack.RESet</td>
<td>Reset stack coverage</td>
</tr>
<tr>
<td>TASK.STack.view</td>
<td>Open stack space coverage</td>
</tr>
<tr>
<td>TASK.Stop</td>
<td>Deprecated</td>
</tr>
<tr>
<td>TCB</td>
<td>Trace control block</td>
</tr>
<tr>
<td>TCB.AllBranches</td>
<td>Broadcast all branches</td>
</tr>
<tr>
<td>TCB.CPU</td>
<td>Broadcast information for specified CPU only</td>
</tr>
<tr>
<td>TCB.CycleAccurate</td>
<td>Cycle accurate tracing</td>
</tr>
<tr>
<td>TCB.DataTrace</td>
<td>Broadcast specified address and data information</td>
</tr>
<tr>
<td>TCB.EX</td>
<td>Broadcast exception level information</td>
</tr>
<tr>
<td>TCB.FCR</td>
<td>Broadcast function call-return information</td>
</tr>
<tr>
<td>TCB.IM</td>
<td>Broadcast instruction cache miss information</td>
</tr>
<tr>
<td>TCB.InstructionCompletionSizeBits</td>
<td>Specify size of completion message</td>
</tr>
<tr>
<td>TCB.KE</td>
<td>Broadcast kernel mode information</td>
</tr>
<tr>
<td>TCB.LSM</td>
<td>Broadcast load store data cache information</td>
</tr>
<tr>
<td>TCB.OFF</td>
<td>Switch TCB off</td>
</tr>
<tr>
<td>TCB.ON</td>
<td>Switch TCB on</td>
</tr>
<tr>
<td>TCB.PCTrace</td>
<td>Broadcast program counter trace</td>
</tr>
<tr>
<td>TCB.PortFilter</td>
<td>Disable port filter</td>
</tr>
<tr>
<td>TCB.PortMode</td>
<td>Specify trace clock ratio</td>
</tr>
<tr>
<td>TCB.PortWidth</td>
<td>Specify trace port width</td>
</tr>
</tbody>
</table>

©1989-2019 Lauterbach GmbH
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCB.Register</td>
<td>Display TCB control register</td>
<td>63</td>
</tr>
<tr>
<td>TCB.RESet</td>
<td>Reset TCB setup to default</td>
<td>64</td>
</tr>
<tr>
<td>TCB.SourceSizeBits</td>
<td>Specify number of bit for core information in trace</td>
<td>64</td>
</tr>
<tr>
<td>TCB.STALL</td>
<td>Stall CPU for complete trace</td>
<td>64</td>
</tr>
<tr>
<td>TCB.state</td>
<td>Display TCB setup</td>
<td>65</td>
</tr>
<tr>
<td>TCB.SV</td>
<td>Broadcast supervisor mode information</td>
<td>66</td>
</tr>
<tr>
<td>TCB.SyncPeriod</td>
<td>Specify TCB sync period</td>
<td>66</td>
</tr>
<tr>
<td>TCB.TC</td>
<td>Broadcast information for specified HW thread</td>
<td>67</td>
</tr>
<tr>
<td>TCB.ThreadSizeBits</td>
<td>Specify number of bit for thread information in trace</td>
<td>67</td>
</tr>
<tr>
<td>TCB.Type</td>
<td>Specify TCB type</td>
<td>68</td>
</tr>
<tr>
<td>TCB.UM</td>
<td>Broadcast user mode information</td>
<td>68</td>
</tr>
<tr>
<td>TCB.Version</td>
<td>Specify trace cell version</td>
<td>69</td>
</tr>
</tbody>
</table>

**TERM**

Term emulation

### Overview

#### Interface Routines

- Interface Routines (EPROM Simulator)
- Interface Routines (Single Character Modes)
- Interface Routines (Buffered Modes)
- Interface Routines (Serial Line Debugger)
- Interface Routines (Special Hardware, JTAG)

#### Functions

- Fast Data Write

#### Interface Routines

- TERM.CLEAR Clear terminal window
- TERM.CLOSE Close files
- TERM.CMDLINE Specify a command line
- TERM.FastWRITE Fast data write
- TERM.GATE Terminal with virtual hosting
- TERM.HARDCOPY Print terminal window contents
- TERM.HEAPINFO Define memory heap parameters
- TERM.LocalEcho Enables/disables local echo for new terminal windows
- TERM.METHOD Select terminal protocol
- TERM.Mode Define terminal type
- TERM.Out Send data to virtual terminal
- TERM.PIPE Connect terminal to named pipe
- TERM_PIPEREAD Connect terminal input to named pipe
- TERM_PIPEWRITE Connect terminal output to named pipe
- TERM.PipeREAD Select terminal protocol
- TERM.PULSE Enable pulse generator for transfers
- TERM.Rate Define polling rate
- TERM.READ Get terminal input from file
- TERM.RESet Reset terminal parameters
TERM.SCROLL  Enable automatic scrolling for terminal window 86
TERM.SIZE    Define size of terminal window 86
TERM.TCP     Route terminal input/output to TCP port 87
TERM.TELNET  Open TELNET terminal window 87
TERM.TRIGGER Trigger on string in terminal window 88
TERM.Vector  Define interrupt vectors 89
TERM.view    Terminal display 90
TERM.WRITE   Write terminal output to file 91

TPIU ................................................................................................................ .......................... 92
TPIU Trace Port Interface Unit (TPIU) 92
Overview TPIU 92
TPIU.CLEAR Re-write the TPIU registers 93
TPIU.IGNOREZEROS Workaround for a special chip 93
TPIU.NOFLUSH Workaround for a chip bug affecting TPIU flush 93
TPIU.PortClock Inform debugger about HSSTP trace frequency 94
TPIU.PortMode Select the operation mode of the TPIU 95
TPIU.PortSize Select interface type and port size of the TPIU 96
TPIU.RefClock Set up reference clock for HSSTP 97
TPIU.Register Display TPIU registers 97
TPIU.RESet Reset TPIU settings 98
TPIU.state Display TPIU configuration window 98
TPIU.SWVPrescaler Set up SWV prescaler 99
TPIU.SWVZEROS Workaround for a chip bug 99
TPIU.SyncPeriod Set period of sync packet injection 100

TPU ................................................................................................................. .......................... 101
TPU.BASE Base address 101
TPU.Break Break TPU 101
TPU.Dump Memory display 101
TPU.Go Start TPU 101
TPU.Register.ALL Register operation mode 101
TPU.Register.NEWSTEP New debugging mode 101
TPU.Register.Set Register modification 101
TPU.Register.view Register display 102
TPU.RESet Disable TPU debugger 102
TPU.SCAN Scanning TPU 102
TPU.SELECT Select TPU for debugging 102
TPU.Step Single step TPU 102
TPU.view View TPU channels 102

Trace ............................................................................................................... .......................... 103
Trace Trace configuration and display 103
Overview Trace 104
About the Command Placeholder <trace> 105
What to know about the TRACE32 default settings for <trace> 105
Types of Replacements for <trace> 106
Replacing <trace> with a Trace Method - Examples 106
Replacing <trace> with a Trace Evaluation - Example 107
Replacing <trace> with RTS for Real-time Profiling - Example 108
Replacing <trace> with Trace Source and Trace Method - Examples 109
How to access the trace sources in TRACE32 111
List of <trace> Command Groups consisting of <trace_source><trace_method> 112
Related Trace Command Groups 113
<trace>.ABCDEF Sampling configuration for probes ABCDEF 114
<trace>.ACCESS Define access path to source code for trace decoding 115
<trace>.ADDRESS Software trace address 117
<trace>.Arm Arm the trace 118
<trace>.AutoArm Arm automatically 119
<trace>.AutoFocus Calibrate AUTOFOCUS preprocessor 119
Preprocessor with AUTOFOCUS Technology 122
<trace>.AutoInit Automatic initialization 124
<trace>.AutoStart Automatic start 124
<trace>.AutoTEST Continuous measurement 125
<trace>.BookMark Set a bookmark in trace listing 127
<trace>.BookMarkToggle Toggles a single trace bookmark 129
<trace>.Break Stop trace 130
<trace>.Chart Display trace contents graphically 131
Parameters 131
General Options 132
Drag and Drop 134
<trace>.Chart.Address Time between program events as a chart 139
<trace>.Chart.DatasYmbol Analyze pointer contents graphically 140
<trace>.Chart.DistriB Distribution display graphically 142
<trace>.Chart.Func Function activity chart 144
<trace>.Chart.GROUP Group activity chart 145
<trace>.Chart.Line Graphical HLL lines analysis 146
<trace>.Chart.Nesting Show function nesting at cursor position 147
<trace>.Chart.sYmbol Symbol analysis 148
<trace>.Chart.TASK Task activity chart 152
<trace>.Chart.TASKFunc Task related function run-time analysis (legacy) 153
<trace>.Chart.TASKINTR Display ISR2 time chart (ORTI) 153
<trace>.Chart.TASKKernel Task run-time chart with kernel markers (flat) 154
<trace>.Chart.TASKSRV Service routine run-time analysis 155
<trace>.Chart.TASKState Task state analysis 156
<trace>.Chart.TASKVSINTERRUPT Time chart 157
<trace>.Chart.TASKVSINTR Time chart 158
<trace>.Chart.VarState Variable activity chart 159
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;trace&gt;.CLEAR</code></td>
<td>Clear FDX communication buffers</td>
<td>162</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.CLOCK</code></td>
<td>Clock to calculate time out of cycle count information</td>
<td>162</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.CLOSE</code></td>
<td>Close FDX files</td>
<td>163</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.ComPare</code></td>
<td>Compare trace contents</td>
<td>163</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.CSELect</code></td>
<td>Select signal for counter</td>
<td>165</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.CustomTrace</code></td>
<td>Custom trace</td>
<td>166</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.CustomTrace.&lt;label&gt;.COMMAND</code></td>
<td>Send command to specific DLL</td>
<td>166</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.CustomTrace.&lt;label&gt;.UNLOAD</code></td>
<td>Unload a single DLL</td>
<td>166</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.CustomTraceLoad</code></td>
<td>Load a DLL for trace analysis/Unload all DLLs</td>
<td>167</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DISable</code></td>
<td>Disable the trace</td>
<td>168</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DISableChannel</code></td>
<td>Disable FDX communication</td>
<td>168</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DisConfig</code></td>
<td>Trace disassembler configuration</td>
<td>169</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DisConfig.CYcle</code></td>
<td>Trace disassemble setting</td>
<td>169</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DisConfig.FlowMode</code></td>
<td>Enable FlowTrace analysis</td>
<td>170</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DisConfig.RESet</code></td>
<td>Reset trace disassemble setting</td>
<td>171</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DisConfig.view</code></td>
<td>Trace disassemble setting</td>
<td>171</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DRAW</code></td>
<td>Plot trace data against time</td>
<td>172</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DRAW.channel</code></td>
<td>Plot no-data values against time</td>
<td>172</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DRAW.Data</code></td>
<td>Plot data values against time</td>
<td>174</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.DRAW.Var</code></td>
<td>Plot variable values against time</td>
<td>178</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.Enable</code></td>
<td>Operation mode</td>
<td>182</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.ENableChannel</code></td>
<td>Enable FDX communication</td>
<td>182</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT</code></td>
<td>Export trace data for processing in other applications</td>
<td>183</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.Ascii</code></td>
<td>Export trace data as ASCII</td>
<td>184</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.Bin</code></td>
<td>Export trace data as binary file</td>
<td>185</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.BRANCHFLOW</code></td>
<td>Export branch events from trace data</td>
<td>186</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.CSVFunc</code></td>
<td>Export the function nesting to a CSV file</td>
<td>187</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.flow</code></td>
<td>Export trace data</td>
<td>188</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.Func</code></td>
<td>Export function nesting</td>
<td>191</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.MTV</code></td>
<td>Export in MCDS Trace Viewer format</td>
<td>192</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.TASKEVENTS</code></td>
<td>Export task event to CSV</td>
<td>193</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.TracePort</code></td>
<td>Export trace packets as recorded at trace port</td>
<td>194</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.VCD</code></td>
<td>Export trace data in VCD format</td>
<td>195</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.VERILOG</code></td>
<td>Export trace data in VERILOG format</td>
<td>196</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.EXPORT.VHDL</code></td>
<td>Export trace data in VHDL format</td>
<td>196</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.FILE</code></td>
<td>Load a file into the file trace buffer</td>
<td>197</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.Find</code></td>
<td>Find specified entry in trace</td>
<td>199</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.FindAll</code></td>
<td>Find all specified entries in trace</td>
<td>203</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.FindChange</code></td>
<td>Search for changes in trace flow</td>
<td>204</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.FLOWPROCESS</code></td>
<td>Process flowtrace</td>
<td>205</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.FLOWSTART</code></td>
<td>Restart flowtrace processing</td>
<td>205</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.Get</code></td>
<td>Display input level</td>
<td>206</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.GOTO</code></td>
<td>Move cursor to specified trace record</td>
<td>208</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT</code></td>
<td>Import trace information</td>
<td>210</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT.CoreByteStream</code></td>
<td>Import pure single core trace data</td>
<td>212</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT.ETB</code></td>
<td>Import on-chip trace data</td>
<td>213</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT.flow</code></td>
<td>Import bus trace data</td>
<td>213</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT.GUESSWRAP</code></td>
<td>Guess wrap pointer</td>
<td>214</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT.STP</code></td>
<td>Import STP recording from file (nibble)</td>
<td>214</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT.STPByteStream</code></td>
<td>Import STP recording from file (byte)</td>
<td>215</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT.TraceFile</code></td>
<td>Import trace data where processing has failed</td>
<td>215</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT.TracePort</code></td>
<td>Import off-chip trace data</td>
<td>216</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT.VCD</code></td>
<td>Import recorded signals in VCD file format</td>
<td>216</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.IMPORT.WRAP</code></td>
<td>Define wrap pointer</td>
<td>217</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.InChannel</code></td>
<td>Inchannel state display</td>
<td>218</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.Init</code></td>
<td>Initialize trace</td>
<td>218</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.JKLMNO</code></td>
<td>Sampling configuration for probes JKLMNO</td>
<td>219</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.JOINFILE</code></td>
<td>Concatenate several trace recordings</td>
<td>219</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.LEVEL</code></td>
<td>Select trigger level manually</td>
<td>219</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.List</code></td>
<td>List trace contents</td>
<td>222</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.ListNesting</code></td>
<td>Analyze function nesting</td>
<td>234</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.ListVar</code></td>
<td>List variable recorded to trace</td>
<td>237</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.LOAD</code></td>
<td>Load trace file for offline processing</td>
<td>240</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.MERGEFILE</code></td>
<td>Combine two trace files into one</td>
<td>242</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.METHOD</code></td>
<td>Select trace method</td>
<td>243</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.Mode</code></td>
<td>Set the trace operation mode</td>
<td>245</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.MUX</code></td>
<td>Select channels</td>
<td>249</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.OFF</code></td>
<td>Switch off</td>
<td>249</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.Out</code></td>
<td>tbd.</td>
<td>249</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PC</code></td>
<td>Display PC in real-time</td>
<td>250</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PipePROTO</code></td>
<td>Unload all DLLs</td>
<td>251</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PipePROTO.COMMAND</code></td>
<td>Send command to DLLs</td>
<td>251</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PipePROTO.load</code></td>
<td>Define a user-supplied DLL as trace sink</td>
<td>251</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PipeWRITE</code></td>
<td>Connect to a named pipe to stream trace sink</td>
<td>252</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PortFilter</code></td>
<td>Specify utilization of trace memory</td>
<td>252</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PortSize</code></td>
<td>Set external port size</td>
<td>254</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PortType</code></td>
<td>Specify trace interface</td>
<td>254</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PROfile</code></td>
<td>Display counter profile</td>
<td>256</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PROfileChart</code></td>
<td>Profile charts</td>
<td>257</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PROfileChart.COUNTER</code></td>
<td>Display a profile chart</td>
<td>258</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PROfileChart.DISTance</code></td>
<td>Time interval for a single event</td>
<td>259</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PROfileChart.DistriB</code></td>
<td>Distribution display in timeslices</td>
<td>261</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PROfileChart.DURation</code></td>
<td>Time between two events</td>
<td>262</td>
</tr>
<tr>
<td>TRACE32 PowerTrace</td>
<td></td>
<td>262</td>
</tr>
<tr>
<td>TRACE32-ICE and TRACE32-FIRE</td>
<td></td>
<td>265</td>
</tr>
<tr>
<td><code>&lt;trace&gt;.PROfileChart.GROUP</code></td>
<td>Group profile chart</td>
<td>266</td>
</tr>
</tbody>
</table>
<trace>.PROfileChart.Rate  
Event frequency 267

<trace>.PROfileChart.sYmbol  
Dynamic program behavior graphically (flat) 269

<trace>.PROfileChart.TASK  
Dynamic task behavior graphically (flat) 273

<trace>.PROfileSTATistic  
Statistical analysis in a table versus time 275

<trace>.Program  
Program trigger unit 275

<trace>.PROTOcol  
Protocol analysis 277

<trace>.PROTOcol.Chart  
Graphic display for user-defined protocol 277

<trace>.PROTOcol.Draw  
Graphic display for user-defined protocol 279

<trace>.PROTOcol.EXPORT  
Export trace buffer for user-defined protocol 280

<trace>.PROTOcol.Find  
Find in trace buffer for user-defined protocol 281

<trace>.PROTOcol.List  
Display trace buffer for user-defined protocol 282

<trace>.PROTOcol.STATistic  
Display statistics for user-defined protocol 285

Protocol specific Options 287

Options for ASYNC 287

Options for CAN 287

Options for I2C 288

Options for JTAG 288

Options for USB 289

<trace>.Rate  
Select sampling rate 290

<trace>.REF  
Set reference point for time measurement 291

<trace>.ReProgram  
Program trigger unit 292

<trace>.RESet  
Reset command 292

<trace>.SAMPLE  
Set AutoFocus sample time offset 292

<trace>.SAVE  
Save trace for postprocessing in TRACE32 294

Parameters 294

Options 295

<trace>.Select  
Select trigger/counter line 298

<trace>.SelfArm  
Automatic restart of trace recording 298

<trace>.SET  
Select line for recording 300

<trace>.ShowFocus  
Display data eye for AUTOFOCUS preprocessor 300

<trace>.ShowFocusClockEye  
Display clock eye 304

<trace>.ShowFocusEye  
Display data eye 306

<trace>.SIZE  
Define buffer size 309

<trace>.SLAVE  
Select slave mode 310

<trace>.SnapShot  
Restart trace capturing once 311

<trace>.SPY  
Enable analysis of streaming file while recording 311

<trace>.state  
Display trace configuration window 313

<trace>.STATistic  
Statistic analysis 316

<trace>.STATistic.Address  
Time between up to 8 program events 317

<trace>.STATistic.AddressDISTance  
Time interval for single program event 318

<trace>.STATistic.AddressDURation  
Time between two program events 319

<trace>.STATistic.BondOut  
Bondout mode 321

<trace>.STATistic.ChildTREE  
Show callee context of a function 321

©1989-2019 Lauterbach GmbH
<trace>.STATistic.COLOR Assign colors to function for colored graphics 322
<trace>.STATistic.CYcle Analyze cycle types 322
<trace>.STATistic.DatasYmbol Analyze pointer contents numerically 325
<trace>.STATistic.DISTance Time interval for a single event 327
<trace>.STATistic.DistriB Distribution analysis 330
<trace>.STATistic.DURation Time between two events 333
<trace>.STATistic.FIRST Start point for statistic analysis 336
<trace>.STATistic.Func Nesting function runtime analysis 338
PowerTrace
Procedure for Measurement for TRACE32-ICE and TRACE32-FIRE 358
<trace>.STATistic.FuncDURation Statistic analysis of single function 363
<trace>.STATistic.FuncDURationInternal Statistic analysis of single func. 364
<trace>.STATistic.GROUP Group run-time analysis 365
<trace>.STATistic.Ignore Ignore false records in statistic 366
<trace>.STATistic.INTERRUPT Interrupt statistic 367
<trace>.STATistic.InterruptIsFunction Statistics interrupt processing 368
<trace>.STATistic.LAST End point for statistic analysis 370
<trace>.STATistic.LINE HLL-line analysis 372
<trace>.STATistic.LINKage Per caller statistic of function 375
<trace>.STATistic.LISTCONFIG tbd. 376
<trace>.STATistic.Measure Analyze the performance of a single signal 377
<trace>.STATistic.MODULE Code execution broken down by module 379
<trace>.STATistic.PAddress Which instructions accessed data address 379
<trace>.STATistic.ParentTREE Show the call context of a function 380
<trace>.STATistic.PIPELINE tbd. 381
<trace>.STATistic.PreFetch Prefetch detection 382
<trace>.STATistic.PROGRAM Code execution broken down by program 383
<trace>.STATistic.PsYmbol Shows which functions accessed data address 383
<trace>.STATistic.Sort Specify sorting criterion for statistic commands 385
<trace>.STATistic.sYmbol Flat run-time analysis 393
<trace>.STATistic.TASK Task activity statistic 398
<trace>.STATistic.TASKKernel Task analysis with kernel markers (flat) 413
<trace>.STATistic.TASKORINTERRUPT Statistic of interrupts and tasks 416
<trace>.STATistic.TASKSRV Analysis of time in OS service routines 416
<trace>.STATistic.TASKState Performance analysis 417
<trace>.STATistic.TASKTREE Tree display of task specific functions 421
<trace>.STATistic.TASKVSINTERRUPT Statistic of interrupts, task-related 422
<trace>.STATistic.TREE Tree display of nesting function run-time analysis 423
<trace>.STATistic.Use Use records 424
<trace>.STREAMCompression Select compression mode for streaming 425
General Commands Reference Guide

- **<trace>.STREAMFILE** Specify temporary streaming file path 426
- **<trace>.STREAMFileLimit** Set size limit for streaming file 427
- **<trace>.STREAMLOAD** Load streaming file from disk 428
- **<trace>.STREAMSAVE** Save streaming file to disk 430
- **<trace>.TCount** Set trigger counter 430
- **<trace>.TDelay** Trigger delay 431
- **<trace>.TERMINation** Use trace line termination of preprocessor 433
- **<trace>.TestFocus** Test trace port recording 433
- **<trace>.TestFocusClockEye** Scan clock eye 436
- **<trace>.TestFocusEye** Check signal integrity 436
- **<trace>.THreshold** Optimize threshold for trace lines 438
- **<trace>.TimeStamp** Configure timestamp usage of LOGGER trace 439
- **<trace>.Timing** Waveform of trace buffer 440
- **<trace>.TMode** Select trigger mode 442
- **<trace>.TOut** Enable trigger output line (PowerIntegrator) 442
- **<trace>.TPreDelay** Pre-trigger delay 443
- **<trace>.TraceChannel** tbd. 443
- **<trace>.TraceCLOCK** Export trace data 444
- **<trace>.TraceCONNECT** Select on-chip peripheral sink 444
- **<trace>.TRACK** Set tracking record 445
- **<trace>.TRIGGER** Trigger the trace 445
- **<trace>.TSELect** Select trigger source 446
- **<trace>.TSYNC** Select trigger line and mode 447
- **<trace>.TView** Display trigger settings 448
- **<trace>.TWidth** Set trigger filter 448
- **<trace>.View** Display single record 449
- **<trace>.XTrack** Cross system tracking 450
- **<trace>.ZERO** Align timestamps of trace and timing analyzers 451

**TRACEPORT** ........................................................................................................... 452

- **TRACEPORT** Configure trace hardware 452
- **TRACEPORT.EndsKiP** Define number of bytes skipped at the end of frame 453
- **TRACEPORT.LaneCount** Select port size of the trace port 454
- **TRACEPORT.LaneSpeed** Inform debugger about trace port frequency 454
- **TRACEPORT.MsgBltEndian** Change bit-order within each byte 455
- **TRACEPORT.MsgByteEndian** Change byte-order within each word 456
- **TRACEPORT.MsgLOngEndian** Change dword-order within each qword 456
- **TRACEPORT.MsgWOrdEndian** Change word-order within each dword 457
- **TRACEPORT.PinReMap** Adapt the lane order of the trace port 458
- **TRACEPORT.RefCLocK** Set up reference clock for trace port 459
- **TRACEPORT.RESet** Reset trace port configuration 459
- **TRACEPORT.StartsKiP** Define number of bytes skipped at the start of frame 460
- **TRACEPORT.state** Display trace port configuration window 461

**TrAddress** ........................................................................................................... 462

©1989-2019 Lauterbach GmbH
**General Commands Reference Guide**

**TRANSLATION**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRANSLATION</td>
<td>Debugger address translation</td>
<td>465</td>
</tr>
<tr>
<td>Overview TRANSLATION</td>
<td></td>
<td>465</td>
</tr>
<tr>
<td>TRANSLATION.AutoEnable</td>
<td>Auto-enable debugger MMU translation</td>
<td>467</td>
</tr>
<tr>
<td>TRANSLATION.AutoSCAN</td>
<td>Autoscan feature for debugger MMU</td>
<td>468</td>
</tr>
<tr>
<td>TRANSLATION.CacheFlush</td>
<td>Flush TRACE32 address translation cache</td>
<td>469</td>
</tr>
<tr>
<td>TRANSLATION.CLEANUP</td>
<td>Clean up MMU table</td>
<td>469</td>
</tr>
<tr>
<td>TRANSLATION.COMMON</td>
<td>Common address ranges for kernel and tasks</td>
<td>470</td>
</tr>
<tr>
<td>TRANSLATION.COMMON.ADD</td>
<td>Add another common address range</td>
<td>472</td>
</tr>
<tr>
<td>TRANSLATION.COMMON.CLEAR</td>
<td>Clear all common logical address ranges</td>
<td>472</td>
</tr>
<tr>
<td>TRANSLATION.Create</td>
<td>Create translation</td>
<td>473</td>
</tr>
<tr>
<td>TRANSLATION.CreateID</td>
<td>Add entry to MMU space ID table</td>
<td>474</td>
</tr>
<tr>
<td>TRANSLATION.CreateTab</td>
<td>Create multiple translations</td>
<td>474</td>
</tr>
<tr>
<td>TRANSLATION.Delete</td>
<td>Delete translation</td>
<td>475</td>
</tr>
<tr>
<td>TRANSLATION.DeleteID</td>
<td>Remove entry from MMU space ID table</td>
<td>475</td>
</tr>
<tr>
<td>TRANSLATION.List</td>
<td>List MMU translation table</td>
<td>476</td>
</tr>
<tr>
<td>TRANSLATION.ListID</td>
<td>List MMU space ID table</td>
<td>477</td>
</tr>
<tr>
<td>TRANSLATION.NoProtect</td>
<td>Unprotect memory</td>
<td>477</td>
</tr>
<tr>
<td>TRANSLATION.OFF</td>
<td>Deactivate debugger address translation</td>
<td>478</td>
</tr>
<tr>
<td>TRANSLATION.ON</td>
<td>Activate debugger address translation</td>
<td>478</td>
</tr>
<tr>
<td>TRANSLATION.PAGER</td>
<td>Allow paged breakpoints for Linux</td>
<td>479</td>
</tr>
<tr>
<td>TRANSLATION.Protect</td>
<td>Protect memory</td>
<td>480</td>
</tr>
<tr>
<td>TRANSLATION.Protect.ADD</td>
<td>Add range to protected memory ranges</td>
<td>480</td>
</tr>
<tr>
<td>TRANSLATION.Protect.OFF</td>
<td>Switch protection of target memory off</td>
<td>481</td>
</tr>
<tr>
<td>TRANSLATION.Protect.ON</td>
<td>Protect entire target memory</td>
<td>482</td>
</tr>
<tr>
<td>TRANSLATION.RESet</td>
<td>Reset MMU configuration</td>
<td>483</td>
</tr>
<tr>
<td>TRANSLATION.ScanAll</td>
<td>Scan MMU tables</td>
<td>483</td>
</tr>
<tr>
<td>TRANSLATION.ScanID</td>
<td>Scan MMU address space tables from kernel</td>
<td>483</td>
</tr>
<tr>
<td>TRANSLATION.SHADOW</td>
<td>Enable shadow access to target memory</td>
<td>484</td>
</tr>
<tr>
<td>TRANSLATION.state</td>
<td>Overview of translation settings</td>
<td>485</td>
</tr>
<tr>
<td>TRANSLATION.TableWalk</td>
<td>Automatic MMU page table walk</td>
<td>486</td>
</tr>
<tr>
<td>TRANSLATION.TlbAutoScan</td>
<td>Allow automatic TLB scans during table walk</td>
<td>487</td>
</tr>
<tr>
<td>TRANSLATION.TRANSparent</td>
<td>Transparent banking area</td>
<td>489</td>
</tr>
</tbody>
</table>

**TrBus**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrBus</td>
<td>Trigger bus</td>
<td>490</td>
</tr>
<tr>
<td>Overview TrBus</td>
<td></td>
<td>490</td>
</tr>
<tr>
<td>TrBus.Arm</td>
<td>Arm the trigger bus</td>
<td>494</td>
</tr>
<tr>
<td>TrBus.Connect</td>
<td>Configure TRIGGER as input or output</td>
<td>495</td>
</tr>
<tr>
<td>TrBus.Mode</td>
<td>Define polarity/edge for the trigger signal</td>
<td>496</td>
</tr>
<tr>
<td>TrBus.OFF</td>
<td>Switch trigger bus off</td>
<td>496</td>
</tr>
</tbody>
</table>

©1989-2019 Lauterbach GmbH
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrBus.Out</td>
<td>Define source for the external trigger pulse</td>
<td>497</td>
</tr>
<tr>
<td>TrBus.RESet</td>
<td>Reset setting for trigger bus</td>
<td>497</td>
</tr>
<tr>
<td>TrBus.Set</td>
<td>Define the target for the incoming trigger</td>
<td>497</td>
</tr>
<tr>
<td>TrBus.state</td>
<td>Display settings for the trigger bus</td>
<td>498</td>
</tr>
<tr>
<td>TrBus.Trigger</td>
<td>Stimulate a trigger on the trigger bus</td>
<td>498</td>
</tr>
</tbody>
</table>

**TrEvent**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrEvent</td>
<td>Event trigger system</td>
<td>500</td>
</tr>
<tr>
<td>Overview</td>
<td>TrEvent</td>
<td>500</td>
</tr>
<tr>
<td>TrEvent.Delay</td>
<td>Define delay</td>
<td>501</td>
</tr>
<tr>
<td>TrEvent.Enable</td>
<td>Select time windows</td>
<td>501</td>
</tr>
<tr>
<td>TrEvent.Init</td>
<td>Initialization</td>
<td>502</td>
</tr>
<tr>
<td>TrEvent.MinInit</td>
<td>Initialization</td>
<td>502</td>
</tr>
<tr>
<td>TrEvent.Mode</td>
<td>Select operation mode</td>
<td>502</td>
</tr>
<tr>
<td>TrEvent.OFF</td>
<td>Switch off</td>
<td>506</td>
</tr>
<tr>
<td>TrEvent.ON</td>
<td>Switch on</td>
<td>507</td>
</tr>
<tr>
<td>TrEvent.RESet</td>
<td>Reset command</td>
<td>508</td>
</tr>
<tr>
<td>TrEvent.Select</td>
<td>Select event source</td>
<td>508</td>
</tr>
<tr>
<td>TrEvent.view</td>
<td>State display</td>
<td>510</td>
</tr>
</tbody>
</table>

**TrIn**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrIn</td>
<td>Internal trigger logic</td>
<td>511</td>
</tr>
<tr>
<td>Overview</td>
<td>TrIn</td>
<td>511</td>
</tr>
<tr>
<td>TrIn.Clock</td>
<td>Define clock</td>
<td>513</td>
</tr>
<tr>
<td>TrIn.Data</td>
<td>Define data</td>
<td>516</td>
</tr>
<tr>
<td>TrIn.Mask</td>
<td>Define bits</td>
<td>516</td>
</tr>
<tr>
<td>TrIn.Normal</td>
<td>Level operation</td>
<td>517</td>
</tr>
<tr>
<td>TrIn.RESet</td>
<td>Reset command</td>
<td>517</td>
</tr>
<tr>
<td>TrIn.state</td>
<td>State display</td>
<td>517</td>
</tr>
<tr>
<td>TrIn.Transient</td>
<td>Transient operation</td>
<td>518</td>
</tr>
</tbody>
</table>

**TrMain**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>TrMain</td>
<td>Trigger system of TRACE32-ICE</td>
<td>520</td>
</tr>
<tr>
<td>Overview</td>
<td>TrMain</td>
<td>520</td>
</tr>
<tr>
<td>TrMain.ALways</td>
<td>Constant triggering</td>
<td>523</td>
</tr>
<tr>
<td>TrMain.Arm</td>
<td>Release and activate trigger system</td>
<td>523</td>
</tr>
<tr>
<td>TrMain.AutoInit</td>
<td>Automatic trigger initialization</td>
<td>523</td>
</tr>
<tr>
<td>TrMain.AutoStart</td>
<td>Automatic trigger initialization</td>
<td>524</td>
</tr>
<tr>
<td>TrMain.Break</td>
<td>Break</td>
<td>524</td>
</tr>
<tr>
<td>TrMain.Count</td>
<td>Set trigger counter</td>
<td>525</td>
</tr>
<tr>
<td>TrMain.Delay</td>
<td>Set trigger delay</td>
<td>525</td>
</tr>
<tr>
<td>TrMain.Init</td>
<td>Initialize trigger system</td>
<td>526</td>
</tr>
<tr>
<td>TrMain.Mode</td>
<td>Select mode</td>
<td>526</td>
</tr>
<tr>
<td>TrMain.OFF</td>
<td>Switch off trigger system</td>
<td>527</td>
</tr>
<tr>
<td>TrMain.Out</td>
<td>Output trigger pulse</td>
<td>527</td>
</tr>
</tbody>
</table>
TrMain.RESet  Reset trigger system 529
TrMain.Set  Select trigger sources 529
TrMain.state  Trigger state display 534
TrMain.Trigger  Trigger 534

TrOnchip  Onchip triggers 535
TrOnchip.Address  Adjust range breakpoint in onchip registers 536
TrOnchip.CONVert  tbd. 536
TrOnchip.CYcle  tbd. 537
TrOnchip.Data  tbd. 538
TrOnchip.RESet  Reset settings to defaults 538
TrOnchip.Set  Break on event 539
TrOnchip.state  Display onchip trigger window 540
TrOnchip.TaskID  tbd. 540
TrOnchip.TEnable  tbd. 540

TrPOD  Trigger probe 542
TrPOD.Clock  Defines data mask 542
TrPOD.ClockPOL  Defines data polarity 542
TrPOD.Data  Defines data mask 543
TrPOD.DataPOL  Defines data polarity 543
TrPOD.Mode  Defines data polarity 544
TrPOD.OFF  Switch off 545
TrPOD.ON  Switch on 545
TrPOD.RESet  Reset command 545
TrPOD.state  State display 546
TrPOD.Time  Defines the time for the pulse width trigger 547
History

27-Feb-19  Updated the commands `TRANSlation.ON`, `TRANSlation.OFF`, `TRANSlation.TableWalk`, and `TRANSlation.NoProtect`.

22-Feb-19  Description of the command group `TRANSlation.Protect`.

13-Aug-18  Description of the new command `TargetSystem.NewInstance`.

10-Aug-18  The channel number was added to the command syntax of most `TERM` commands. The channel number allows the concurrent use of several terminal windows.

03-Aug-18  Link to “Application Note for the Trace.DRAW Command” (app_trace_draw.pdf) added to `Trace.DRAW.Data` and `Trace.DRAW.Var` command. Related to this, a clean-up for both commands was performed.

10-Jul-18  Description of the command `TRANSlation.state`.

08-Mar-18  Description for command `Trace.ListVar` updated.

25-Jan-18  Syntax description and examples for commands `Trace.DRAW.Data` and `Trace.DRAW.Var` updated.

08-Dec-17  Description of the command group `TRACEPORT`.
Using the command group **TargetSystem**, you can start new TRACE32 PowerView instances from within a running instance and keep an overview of these instances.

The instances started with **TargetSystem.NewInstance** are automatically connected to the same PowerDebug hardware module or to the same MCI Server as the instance that initiated the start process. (In case of the MCI Server, the setting in the config file is: \texttt{PBI=MCISERVER}).

The **TargetSystem.state** window provides an overview of the status of the cores assigned to the various TRACE32 instances. The window also helps you keep an overview of the synchronization mechanism between the TRACE32 instances, which is set up with the **SYnch** command group.

In addition, the **TargetSystem.state** window displays the InterCom names and UDP port numbers used by the instances for communication with each other via the **InterCom** system.

**See also**

- **TargetSystem.NewInstance**
- **TargetSystem.state**
- **SYnch**
- **InterCom**

**NOTE:**

The **TargetSystem.NewInstance** command is not available for:

- The TRACE32 Instruction Set Simulator (\texttt{PBI=SIM} in the config file)
- The debuggers connected to the target via the GDI interface (\texttt{PBI=GDI})
- The debuggers connected to the target via the MCD interface (\texttt{PBI=MCD})
**TargetSystem.NewInstance**

Start new TRACE32 PowerView instance

### Format:

```
TargetSystem.NewInstance <intercom_name> [/<option>]
```

### <option>:

- `ARCHitecture <arch>`
- `APIPORT <port_number>`
- `ChipIndex <index>`
- `ChipIndexMin <index_min>`
- `ONCE`

### <arch>:

- 8051
- COLDFIRE
- ANDES
- AP3
- ARC
- ARM
- ARM64
- …

### <index>:

1. … 254.

### <index_min>:

1. … 254.

---

Allows a TRACE32 PowerView instance to start new TRACE32 PowerView instances (max. 15 new instances) for debugging AMP systems. In AMP (asynchronous multiprocessing) systems, each TRACE32 PowerView instance is responsible for an SMP subsystem or single core. For more information, see **CORE.ASSIGN**.

All instances started with **TargetSystem.NewInstance** are automatically connected to the same PowerDebug hardware module or the same MCI Server (PBI=MCISERVER in the config.t32 file) as the instance that initiated the start process.

The instance that starts another instance clones the current config file (by default config.t32) and extends the cloned file for the new instance.

---

**NOTE:**

The **TargetSystem.NewInstance** command is not available for:

- The TRACE32 Instruction Set Simulator (PBI=SIM in the config file)
- The debuggers connected to the target via the GDI interface (PBI=GDI)
- The debuggers connected to the target via the MCD interface (PBI=MCD)

---

<table>
<thead>
<tr>
<th>&lt;intercom_name&gt;</th>
<th>Assigns a user-defined InterCom name to the new TRACE32 instance.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARCHitecture &lt;arch&gt;</td>
<td>Selects the architecture of the new TRACE32 instance. If the ARCHitecture option is omitted, then a TRACE32 instance of the same architecture will be started.</td>
</tr>
<tr>
<td>APIPORT &lt;port_number&gt;</td>
<td>Parameter Type: Decimal value. Passes a UDP remote API &lt;port_number&gt; to the new TRACE32 instance.</td>
</tr>
</tbody>
</table>
### Examples

**Example 1:** This script shows how to start a second TRACE32 instance named `mySecondInstance` from within the current TRACE32 instance.

```
TargetSystem.NewInstance mySecondInstance /ARCHitecture ARM64
InterCom.execute mySecondInstance PRINT "started by the first instance"
```

**Example 2:** Let's assume you have started a number of instances and now want to quit a particular instance. This script shows how to quit a TRACE32 instance named `mySecondInstance` in a set of TRACE32 instances.

```
InterCom.execute mySecondInstance QUIT
```

### See also
- `TargetSystem`
- `TargetSystem.state`
- `InterCom.ENable`

▲ 'Release Information' in 'Release History'
Opens the **TargetSystem.state** window, providing an overview of the multicore system configuration and state across multiple TRACE32 instances sharing one PowerDebug hardware module or MCI Server. The indices on the first and second level are configured using `SYStem.CONFIG.CORE <chip> <core>`. The indices on the third level indicate the thread index of the SMP system that can be defined by `CORE.ASSIGN` or `CORE.NUMber`.

![TargetSystem.state window](image)

1st level: `<chip>`  
2nd level: `<core>`  
3rd level: thread

The **TargetSystem** window is not available for front-end debuggers.

To illustrate the **TargetSystem.state** command, the following use cases are provided:

- **Use case 1**: Diagnostic tool for the target system structure
- **Use case 2**: TRACE32 instance selector
- **Use case 3**: Manage the `SYnch` settings for all TRACE32 instances
### <columns> - Description of Columns in the TargetSystem.state Window

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>DEFAult</strong></td>
<td>Adds <strong>TargetSystem</strong>, <strong>CoreType</strong> and <strong>CoreState</strong> column. If no column is passed <strong>DEFAult</strong> is used automatically.</td>
</tr>
<tr>
<td><strong>ALL</strong></td>
<td>Displays all available columns in the <strong>TargetSystem.state</strong> window.</td>
</tr>
<tr>
<td><strong>TargetSystem</strong></td>
<td>Adds the <strong>TargetSystem</strong> column to show a hierarchical view on the system. If the column is left out, it will be added automatically. The parameter is used to tell the dialog that the <strong>DEFAult</strong> option is not active and only the <strong>TargetSystem</strong> column shall be shown.</td>
</tr>
<tr>
<td><strong>CoreType</strong></td>
<td>Adds a column to show the target architecture of a core and core family name if available.</td>
</tr>
<tr>
<td><strong>CoreState</strong></td>
<td>Shows the state of the core. The state can be system down (gray color), power down (red color), reset (red color), stopped (bold) or running. The running state can be extended by an attribute that indicates a run mode e.g. “no core clock”.</td>
</tr>
<tr>
<td><strong>Title</strong></td>
<td>Adds a column with the corresponding window title. The title can be set by the configuration file before start-up or by the <strong>TITLE</strong> command.</td>
</tr>
<tr>
<td><strong>InterComPort</strong></td>
<td>Adds a column with the InterCom UDP port numbers of TRACE32 instances. The InterCom port numbers are used by the <strong>InterCom</strong> commands and the <strong>SYnch</strong> commands. You can assign a new port number by double-clicking a port number in the <strong>ic port</strong> column. For an illustrated example, see <strong>InterCom.PORT</strong>.</td>
</tr>
<tr>
<td><strong>InterComName</strong></td>
<td>Adds a column with the InterCom names of TRACE32 instances. Names are created with the commands <strong>InterCom.NAME</strong> or <strong>InterCom.ENable</strong>. The names can then be used as arguments in <strong>InterCom</strong> and <strong>SYnch</strong> commands. You can rename an instance by double-clicking a name in the <strong>ic name</strong> column. For an illustrated example, see <strong>InterCom.NAME</strong>.</td>
</tr>
<tr>
<td><strong>INSTance</strong></td>
<td>Adds a column, showing the value of <strong>INSTANCE=</strong> from the config file. If <strong>INSTANCE=</strong> is missing in the config file, then 1 is displayed by default. That is, in this case the display value is equivalent to the explicit setting <strong>INSTANCE=1</strong> in the config file.</td>
</tr>
<tr>
<td><strong>UseCore</strong></td>
<td>Adds a column, showing the value of <strong>CORE=</strong> from the config file. If <strong>CORE=</strong> is missing in the config file, then 1 is displayed by default. That is, in this case the display value is equivalent to the explicit setting <strong>CORE=1</strong> in the config file. See also <strong>SYStem.USECORE()</strong>.</td>
</tr>
<tr>
<td><strong>SYnch.All</strong></td>
<td>Adds the columns <strong>SYnch.Go</strong>, <strong>SYnch.Step</strong>, <strong>SYnch.Break</strong> and <strong>SYnch.SystemMode</strong>.</td>
</tr>
<tr>
<td>----------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>SYnch.Go</strong></td>
<td>Adds the column to indicate and edit the <strong>SYnch.MasterGo</strong> and <strong>SYnch.SlaveGo</strong> setting. The header of the column is named <strong>SG</strong>.</td>
</tr>
<tr>
<td><strong>SYnch.Step</strong></td>
<td>Adds the column to indicate and edit the <strong>SYnch.MasterStep</strong> and <strong>SYnch.SlaveStep</strong> setting. The header of the column is named <strong>SS</strong>.</td>
</tr>
<tr>
<td><strong>SYnch.Break</strong></td>
<td>Adds the column to indicate and edit the <strong>SYnch.MasterBreak</strong> and <strong>SYnch.SlaveBreak</strong> setting. The header of the column is named <strong>SB</strong>.</td>
</tr>
<tr>
<td><strong>SYnch.SystemMode</strong></td>
<td>Adds the column to indicate and edit the <strong>SYnch.MasterSystemMode</strong> and <strong>SYnch.SlaveSystemMode</strong> setting. The header of the column is named <strong>SM</strong>.</td>
</tr>
</tbody>
</table>

**<options> - Options for the TargetSystem.state Window**

| **Global** | Don’t highlight specific information for the TRACE32 instance from where the dialog was opened. The dialog can be moved outside of the main window and used to act as an independent window to bring a certain instance to foreground by a double click to of an entry of the TargetSystem tree column. |
| **UseTitle** | Use the TRACE32 window title as name for an SMP Subsystem or Core. The title can be set by the configuration file before start-up or by the PRACTICE command **TITLE**. |
| **UseICName** | Use the TRACE32 InterCom name as window title for an SMP subsystem or core. The InterCom name can be set with the **InterCom.NAME** command. |
Use case 1: Diagnostic tool for the target system structure

The command opens the window showing the overall system. Nodes that belong to this TRACE32 instance are displayed in bold. A double-click to a thread selects this thread to be active.

```
TargetSystem.state CoreType /UseTitle
```

Use case 2: TRACE32 instance selector

The command opens the window showing the overall system and the state of the particular cores. The window can be moved outside of the TRACE32 instance where the command was executed. A double-click at an SMP system node or core will bring the assigned instance to foreground.

```
TargetSystem.state CoreState /UseTitle /Global
```
Use case 3: Manage the SYnch settings for all TRACE32 instances

The command opens the window showing the overall system and the SYnch settings.

```
TargetSystem.state SYnch.All /UseTitle /Global
```

A single click at an entry in one of the columns will change the setting in the SYnch dialog and set the connection ports.

<table>
<thead>
<tr>
<th>default</th>
<th>Neither master nor slave option is set.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st click M</td>
<td>master option set.</td>
</tr>
<tr>
<td>2nd click S</td>
<td>slave option is set.</td>
</tr>
<tr>
<td>3rd click MS</td>
<td>master and slave option is set.</td>
</tr>
</tbody>
</table>

See also
- TargetSystem
- SYnch
- SYnch.MasterSystemMode
- SYnch.SlaveSystemMode
- InterCom.execute
- TargetSystem.NewInstance
- SYnch.Connect
- SYnch.SlaveBreak
- SYnch.state
- CORE.ASSIGN
- SYnch.MasterGo
- SYnch.SlaveGo
- SYnch.SlaveStep
- InterCom
- CORE.NUMber
- SYnch.MasterStep
- SYnch.SlaveStep
- SYStem.CONFIG.CORE
- InterCom

©1989-2019 Lauterbach GmbH
Overview TASK

This chapter describes the OS Awareness features (aka kernel awareness), generic to all processors and kernels. Kernel specific features are described in additional manuals, see OS Awareness Manuals.

The OS Awareness may support the following main features:

- Display of kernel resources (e.g. tasks, queues, semaphores, messages).
- Task stack coverage.
- Task related breakpoints.
- Task context display.
- Manual execution of system calls.
- Operating system's MMU support.
- Dynamic task performance measurement
- Task runtime statistics and flowchart display out of the trace buffer. Display of task switches in the trace listing.
- Task state statistics and time chart out of the trace buffer, i.e. show how long each task is in a certain state (running, ready, etc.).
- Task-related function runtime statistics, flowchart display and function nesting display out of the trace buffer.
- Display system calls with parameters in the trace listing.
- Fast access to the features through dedicated menus.
OS Awareness Configurations

The OS Awareness is configured by the TASK.CONFIG command. The command loads a configuration file that tells the debugger all kernel related information. It can be adopted to any (RT)OS kernel. Lauterbach provides ready-to-start configuration files for a wide range of real-time operating systems. If you want to adapt it to your own proprietary kernel, ask Lauterbach for assistance.

What to know about Task Magic Numbers, Task IDs and Task Names

In TRACE32, operating system tasks (short: tasks) can be identified based on one of these values:

- Task magic number
- Task ID
- Task name

For OS-aware debugging and tracing, these three values are displayed in the TASK.List.tasks window and can be returned with the functions TASK.MAGIC(), TASK.ID(), and TASK.NAME(). In addition, the three values can be passed as parameters to task-related TRACE32 commands and options.

NOTE: In case of the TASK.CONFIG command, you will encounter the parameter <magic_address>.
- <task_magic> and <magic_address> are not the same.
- For information about <magic_address>, see TASK.CONFIG command.

Task Magic Number

The task magic number is an arbitrary hex value, used by TRACE32 to uniquely identify a task of an operating system. The meaning of the value depends on the OS Awareness; often it refers to the task control block of the target OS or to the task ID.

| <task_magic> | Parameter Type: Hex value.  
Example: TASK.select 0xEEF7B040 |

Task ID

This value refers to the numeric task ID as given by the operating system. If the OS does not provide a task ID, this option may not be available.

| <task_id> | Parameter Type: Decimal value.  
Example: TASK.select 1546. |
Task Name

This string refers to the task name as given by the operating system. If the OS does not provide a task name, this option may not be available.

If the task runs in a system involving virtualization, then the task name can be preceded with the machine name.

<table>
<thead>
<tr>
<th>&lt;task_name&gt;</th>
<th>Parameter Type: String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1: TASK.select &quot;adb:1546&quot;</td>
<td></td>
</tr>
<tr>
<td>Example 2: TASK.select &quot;FreeRTOS:::SieveDemo&quot;</td>
<td></td>
</tr>
</tbody>
</table>

Here, FreeRTOS is the name of the machine, and the three colons ::: serve as the separator between machine name and task name.

Glossary

For important OS and Hypervisor Awareness terms, such as task, thread, process, machine, kernel, MMU space, and virtual machine, refer to the “TRACE32 Glossary” (glossary.pdf).
TASK.ACCESS

Control memory access

Defines the memory access class used by TASK related windows.

TASK related windows may access the target memory (e.g. when reading task control blocks). 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, the window is temporarily frozen.

TASK.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 manuals for a description of E: and C: to your processor.

- Processor Architecture Manuals
- Target Guides FIRE
- Target Guides ICE

See also
- TASK

TASK.Break

ICE only

Deprecated

It was used in 68k ICE only for task selective debugging.

TASK.Break marks a task of the selective list to be breaked at the next entry.

See also
- TASK
**TASK.CACHEFLUSH**

**Reread task list**

ICE only

<table>
<thead>
<tr>
<th>Format:</th>
<th><strong>TASK.CACHEFLUSH</strong></th>
</tr>
</thead>
</table>

Usually not needed. Use only if advised to do so.

The debugger reads out the task list of the target at each single step or Go/Break sequence, and stores the list internally (see **TASK.List.tasks**). If the task list or task characteristics change while the target is halted, a manual update of the task list may be necessary. This command forces an immediate re-evaluation of the task list.

**See also**

■ **TASK**

---

**TASK.CONFIG**

**Configure OS Awareness**

<table>
<thead>
<tr>
<th>Format:</th>
<th><strong>TASK.CONFIG</strong> &lt;os_awareness_file&gt; &lt;magic_address&gt; &lt;args&gt; [/&lt;option&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;option&gt;:</td>
<td><strong>ACCESS</strong> &lt;class&gt;</td>
</tr>
</tbody>
</table>

Configures the OS Awareness using a given configuration file. Please refer to the OS-specific manual. See OS Awareness Manuals.

**Arguments:**

- `<os_awareness_file>`: File name of the configuration file.
- `<magic_address>`: Address of the memory location holding the task magic number of the currently running task. See “What to know about Task Magic Numbers, Task IDs and Task Names”, page 24.
- `<args>`: All other arguments are interpreted by the configuration file. Details of predefined files are described in the kernel-specific part of an OS Awareness Manual.
Options:

**ACCESS**

Defines the memory access class used by **TASK**-related windows. See **TASK.ACCESS**.

**TASK.COPYDOWN**

Copy file from host into target

**Format:**

```
TASK.COPYDOWN <source_file_host> <destination_file_target>
```

Copies a file from the host into the target. Only supported for Linux and QNX run mode debugging.

**See also**

- **TASK**
  - ‘Commands for Run Mode Debugging’ in ‘Run Mode Debugging Manual Linux’
  - ‘Commands for Run Mode Debugging’ in ‘Run Mode Debugging Manual QNX’
**TASK.COPYUP**

**Copy file from target into host**

Format:  

```
TASK.COPYUP <source_file_target> <destination_file_host>
```

Copies a file from the target into the host. Only supported for Linux and QNX run mode debugging.

**See also**

- TASK
- ‘Commands for Run Mode Debugging’ in ‘Run Mode Debugging Manual Linux’
- ‘Commands for Run Mode Debugging’ in ‘Run Mode Debugging Manual QNX’

**TASK.CreateExtraID**

**Create a virtual task**

Format:  

```
TASK.CreateExtraID <task_name> <task_id> <space_id> <trace_id>
```

Creates a virtual task ID for trace analysis. Trace analysis will use the given task ID for task identification rather than the task magic number. Only for some dedicated applications.

**See also**

- TASK

**TASK.CreateID**

**Create virtual task**

Format:  

```
TASK.CreateID <task_name> <task_id> <space_id> <trace_id>
```

Creates a virtual task name for trace analysis. Trace analysis will use the given task name for task identification, rather than the task magic. Only for some dedicated applications.

**See also**

- TASK
TASK.Debug

ICE only

It was used in 68k ICE only for task selective debugging. TASK.Debug marks a task of the selective list as active for debugging.

See also

- TASK
- 'LynxOS Commands' in 'OS Awareness Manual LynxOS'

TASK.DeleteID

Delete virtual task

Delete a virtual task created with TASK.CreateID or TASK.CreateExtraID.

See also

- TASK

TASK.DETACH

Debugger only

Detach from task

Requests the debug agent to detach from the process <id>.

Only applicable if GDB (Linux) is used as debug agent.

TASK.DETACH 41.

See also

- TASK
- 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual QNX'
TASK.INSTALL

Deprecated

Format: TASK.INSTALL (deprecated)

See also
■ TASK

TASK.KILL
End task

Debugger only

Format: TASK.KILL <id>

Request the debug agent to end the process <id>.

Only applicable if GDB (Linux) or TRK (Symbian) is used as debug agent.

TASK.KILL 41.

See also
■ TASK
▲ 'Pdebug Front-end Specific Commands' in 'TRACE32 pdebug Target Server for ARM'
▲ 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual Linux'
▲ 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual QNX'
The windows of the **TASK.List** command group provide information about processes, space IDs, MMU spaces, machines, and tasks known to the debugger in an RTOS and hypervisor environment. The debugger needs a so-called “awareness” of the RTOS or hypervisor to be able to read out these items from the target.

### See also
- **TASK.List.MACHINES**
- **TASK.List.SPACES**
- **TASK.List.tasks**
- **TASK.List.TREE**

### TASK.List.MACHINES

**List machines**

<table>
<thead>
<tr>
<th>Format:</th>
<th>TASK.List.MACHINES</th>
</tr>
</thead>
</table>

Lists information about all machines known to the debugger. Machines refer to virtual machines in a hypervisor environment. The hypervisor itself is listed as machine with ID 0.

Machines are only available if **SYStem.Option MACHINESPACES** is set to **ON**.

For several purposes, the debugger needs to know which machines are active in the system. The debugger uses the hypervisor specific awareness to read out all machine characteristics that it needs for its operation. **TASK.List.MACHINE** shows the machine characteristics that the debugger uses.

![Machine List Window](image)

A The machine that is currently running on the selected core is marked.

### Description of Columns in the TASK.List.MACHINE Window

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>magic</td>
<td>Machine magic number. Unique number for the machine. Usually the address of the control block structure.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the object, if available.</td>
</tr>
<tr>
<td>mid</td>
<td>Machine ID if a hypervisor system is set up.</td>
</tr>
<tr>
<td>access</td>
<td>Access class that an awareness uses for this machine.</td>
</tr>
</tbody>
</table>
LIST MMU spaces

**General Commands Reference Guide**

©1989-2019 Lauterbach GmbH

---

**See also**

- TASK.List
- TASK.List.tasks
- 'Release Information' in 'Release History'

---

**TASK.List.SPACES**

**List MMU spaces**

**Format:**

```
TASK.List.SPACES
```

Lists all **MMU spaces** known to the debugger. MMU spaces usually refer to processes in an OS/RTOS environment. MMU spaces are only available if **SYStem.Option MMUSPACES** is set to **ON**.

For several purposes, the debugger needs to know which MMU spaces are active in the system. The debugger uses the kernel specific awareness to read out all space characteristics that it needs for its operation. **TASK.List.SPACES** shows the space characteristics that the debugger uses.

Each kernel specific awareness has a different display command to show the active processes with the characteristics that are essential to the specific kernel. Please see the appropriate **OS Awareness Manual** (rtos_<os>.pdf) for this command.

**Description of Columns in the TASK.List.SPACES Window:**

- **magic**: Space magic number. Unique number for the space. Usually the address of the control block structure.
- **name**: Name of the object, if available.
- **id**: ID of the object, if available.

---

©1989-2019 Lauterbach GmbH
General Commands Reference Guide

©1989-2019 Lauterbach GmbH

See also
- TASK.List
- TASK.List.tasks

### TASK.List.tasks

**List all running tasks**

**Format:**

```
TASK.List.task
```

Lists all tasks known to the debugger. Additional information about machines and MMU spaces is only displayed if `SYS tempt.Option MMUSPACES` and `SYS tempt.Option MACHINESPACES` are set to `ON`.

For several purposes, the debugger needs to know which tasks are active in the system. The debugger uses the kernel specific awareness to read out all task characteristics that it needs for its operation. `TASK.List.tasks` shows the task characteristics that the debugger uses.

Each kernel specific awareness has a different display command to show the active tasks with the characteristics that are essential to the specific kernel. Please see the appropriate OS Awareness Manual (`rtos_.<os>.pdf`) for this command.

A The task that is currently running on the selected core is marked.

**Description of Columns in the TASK.List.tasks Window:**

- **magic**: Task magic number. Unique number for the task. Usually the address of the control block structure.
- **name**: Name of the object, if available.
- **id**: ID of the object, if available.
- **space**: Space name or ID if the OS uses MMU spaces.
- **traceid**: ID that identifies an object in the trace list.
core  Identifies in SMP systems at which core this task runs.

sel   Task selected for debugging (only in Run Mode Debugging).

stop  Task selected to stop on break (only in Run Mode Debugging).

machine  Machine name or ID if a hypervisor system is set up.

See also

- TASK.List
- TASK.List.MACHINES
- TASK.List.SPACES
- TASK.List.TREE

- 'Pdebug Front-end Specific Commands' in 'TRACE32 pdebug Target Server for ARM'
- 'Commands for Run Mode Debugging’ in ‘Run Mode Debugging Manual Linux’
- 'Commands for Run Mode Debugging’ in ‘Run Mode Debugging Manual QNX’
- 'Commands for Run Mode Debugging’ in ‘Run Mode Debugging Manual Symbian’
- ‘Specific Commands’ in ‘Native Process Debugger’
### TASK.List.TREE

Display tasks in a tree structure

**Format:**  

```
TASK.List.TREE
```

Displays machines, MMU spaces, and tasks in the form of a tree structure.

![Tree Structure Image]

- **A** Level 1 of the tree: Machines.
- **B** Level 2: MMU spaces.
- **C** Level 3: Tasks.
- **D** Yellow lines: The machine, the MMU space, and the task that are currently running on the selected core are marked.

### Description of Columns in the TASK.List.TREE Window

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>magic</td>
<td>Magic number. Unique number for each object (machine/MMU space/task). Usually the address of the control block structure.</td>
</tr>
<tr>
<td>name</td>
<td>Name of the object, if available.</td>
</tr>
</tbody>
</table>

**See also**

- TASK.List
- TASK.List.tasks
- 'Release Information' in 'Release History'

---

### TASK.ListID

List virtual tasks

**Format:**  

```
TASK.ListID
```

Opens the `TASK.ListID` window, displaying virtual tasks created with `TASK.CreateID` or `TASK.CreateExtraID`.

**See also**

- TASK

©1989-2019 Lauterbach GmbH
This command is for internal use only and should not be used. `TASK.MTXTRC` shows a trace of the OS Awareness hyper processor activity.

**See also**
- TASK

**Format:** `TASK.MTXTRC (deprecated)`
Several windows of the OS Awareness show task-related information, e.g. **TASK.STacK** or **Trace.Chart.TASK**. Internally, the OS Awareness always uses the task magic numbers to identify a task. When displaying the task-related information, the debugger can translate this task magic number into a more readable task name, using a task name translation table. If the debugger finds an entry with the appropriate task magic number, it shows the task name instead of the task magic number (or task ID).

The translation table can be populated manually or automatically. If the TASK configuration file supports it, the debugger automatically populates the table with the current available task magic numbers and their names. Additionally, or if no configuration file exists, or if the configuration doesn't support task names, table entries may be added manually. If a manual entry and an automatic entry have the same task magic number, the manual entry overwrites the automatic one.

---

### TASK.NAME

**Translation of task magic number to task name**

See also
- TASK.NAME.DELete
- TASK.NAME.RESet
- TASK.NAME.Set
- TASK.NAME.view
- TASK

#### TASK.NAME.DELete

Delete a task name table entry

**Format:**

```markdown
TASK.NAME.DELete <task_magic>
```

Deletes the entry, specified by `<task_magic>`, from the task name translation table. If the entry is an automatic entry, the next usage of task names may add the automatic entry again.

See also
- TASK.NAME
- TASK.NAME.view

#### TASK.NAME.RESet

Reset task name table

**Format:**

```markdown
TASK.NAME.RESet
```

Erases the whole task name translation table. If the TASK configuration file supports task name evaluation, the next usage of task names will populate the table again with automatic entries.

See also
- TASK.NAME
- TASK.NAME.view

©1989-2019 Lauterbach GmbH
TASK.NAME.Set  Set a task name table entry

**Format:**

```
TASK.NAME.Set <task_magic> <task_name>
```

Adds a manual entry to the task name translation table.

- `<task_magic>`
- `<task_name>`

The string specified by `<task_name>` is assigned to the task specified by `<task_magic>`. If the table contains already an automatic entry for the specified task magic number, it will be overwritten by the new entry!

**Example:**

```
TASK.NAME.Set 0x58D68 "My_Task 1"
```

**See also**
- TASK.NAME
- TASK.NAME.view

TASK.NAME.view  Show task name translation table

**Format:**

```
TASK.NAME.view
```

Shows the contents of the task name translation table.

**A** Flag “a”: The entry was set automatically by the TASK configuration file.

**B** Flag “m”: The entry was set manually by the TASK.NAME.Set command.

**See also**
- TASK.NAME
- TASK.NAME.DELETE
- TASK.NAME.RESET
- TASK.NAME.Set
**TASK.NoBreak**

ICE only

| Format: | TASK.NoBreak <task> (deprecated) |

It was used in 68k ICE only for task selective debugging. TASK.NoBreak marks a task of the selective list to be not breaked at the next entry.

**See also**
- TASK

---

**TASK.NoDebug**

ICE only

| Format: | TASK.NoDebug <task> (deprecated) |

It was used in 68k ICE only for task selective debugging. TASK.Debug marks a task of the selective list as inactive for debugging.

**See also**
- TASK

---

**TASK.NoStop**

ICE only

| Format: | TASK.NoStop <task> (deprecated) |

It was used in 68k ICE only for task selective debugging. TASK.NoStop marks a task of the selective list to be stopped.

**See also**
- TASK
It was used in 68k ICE only for task selective debugging. TASK.OFF switches off the task selective debugger.

See also
- TASK

It was used in 68k ICE only for task selective debugging. TASK.ON switches on the task selective debugger.

See also
- TASK
See also

- TASK.ORTI.CPU
- TASK.ORTI.load
- TASK.ORTI.NOSTACK
- TASK

▲ 'Release Information’ in 'Release History'

### TASK.ORTI.CPU

Set OSEK SMP CPU number

Format: `TASK.ORTI.CPU <cpu_id>`

If TRACE32 is set up in AMP mode (one PowerView instance for each core), it assigns a CPU ID to each individual core, starting with zero. An AUTOSAR/OSEK operating system in SMP mode may assign a different CPU ID to the cores, depending how the OS uses the chip.

This command instructs the debugger to use the given CPU ID when extracting core dependent information from the ORTI file.

See also

- TASK.ORTI

### TASK.ORTI.load

Configure OS Awareness for OSEK/ORTI

Format: `TASK.ORTI.load <file>`

Configures the OS Awareness for AUTOSAR/OSEK operating systems using ORTI. For a detailed description, please refer to the chapter “OS Awareness Manual OSEK/ORTI” (rtos_orti.pdf).

See also

- TASK.ORTI
When using the OS Awareness for ORTI (see TASK.ORTI.load), this command excludes a task from all stack evaluations, e.g. when performing a trace function analysis. Usually used for the idle routine if it isn't running as a separate task.

See also

- TASK.ORTI
**TASK.RESet**

**Reset OS Awareness**

Resets the OS Awareness.

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

**See also**

- TASK

---

**TASK.RUN**

**Load task**

**Format:**

```
TASK.RUN <process>
```

Loads `<process>` and prepares it for debugging.

Only applicable if GDB (Linux) or TRK (Symbian) is used as debug agent.

```
TASK.RUN /bin/hello
```

**See also**

- TASK

  ▲ 'Pdebug Front-end Specific Commands' in 'TRACE32 pdebug Target Server for ARM'
  ▲ 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual Linux'
  ▲ 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual QNX'
  ▲ 'Commands for Run Mode Debugging' in 'Run Mode Debugging Manual Symbian'

---

**TASK.select**

**Display context of specified task**

**Format:**

```
TASK.select <task_magic> | <task_id> | "<task_name>"
```

**Stop mode debugging:** In the case of an SMP system the currently selected core is changed to the core running the specified task. As a result the debugger view is changed to this core and all TRACE32 commands without `/CORE <number>` option apply to it.

If the specified task is not running, TRACE32 reads the register set of the specified task from the OS data structures. This is needed to display the context of the specified task in the TRACE32 GUI.
The TRACE32 state line changes to a reddish look-and-feel (see screenshot below) to indicate that the context of a not-running task is displayed. TRACE32 display commands such as List.auto, Register.view, Frame.view or Var.Local apply to this task. Whereas all other commands switch back to the currently running task before they are executed.

If the task is running on different virtual machine, TRACE32 reads the context of the VCPU that is processing the task on this machine.

```
<table border="1">
  <tr>
    <th><code>&lt;task_magic&gt;</code></th>
    <th>For information about the parameters, see “What to know about Task Magic Numbers, Task IDs and Task Names”, page 24.
    </th>
  </tr>
  <tr>
    <th><code>&lt;task_id&gt;</code></th>
    <th></th>
  </tr>
  <tr>
    <th><code>&lt;task_name&gt;</code></th>
    <th></th>
  </tr>
</table>
```

Run mode debugging: Selects the specified task for debugging (e.g. GDB (Linux) or TRK (Symbian)).

```
TASK.select 41.
```
Set the awareness directory

The Linux awareness and menu call scripts from the awareness directory. This directory is set per default to `~/demo/<arch>/kernel/linux/<linux_version>`. When loading the awareness outside this directory, TRACE32 prints a warning. With this command you can change the awareness directory. Scripts will be called then from the new directory.

**See also**

- TASK
The **TASK.STacK** command group allows to watch the stack usage in single tasking and multi-tasking systems. In single tasking systems, or in non supported operating systems, the user has to specify the stack area manually. The task magic number can be any number to identify a stack area.

In configured RTOS operation, the magic number must be the respective task magic number.

The debugger tries to get the current stack pointer. If the OS Awareness is configured, and the configuration file supports stack coverage, the current stack pointer is read out of the task control block of the application. When the application is stopped, the stack pointer is read from register and displayed at the current running task. Without any RTOS configuration the stack pointer will be displayed at the stack that fits to the pointer (pointer inside the stack). If no stack fits, or if the running task could not be found, the stack pointer of the register is displayed in an extra line. (See also **TASK.STacK.view**)

There are two methods to evaluate the maximum stack space. If flag memory is available (see **MAP.Flag**), the flag system can be used. Be sure that flag memory is mapped to the stack areas. The emulator searches from stack top to stack bottom for the first write flag. The found address is then used as maximum stack address. If no flag system is available, you can use a pattern search, if the stack is initialized with a known pattern. In this case, the debugger searches from stack top to stack bottom for the first byte, that is not equal to the specified pattern. (See also **TASK.STacK.PATtern**)

For more information on stack coverage in operating systems, refer to the **OS Awareness Manuals**.

---

**See also**

- TASK.STacK.ADD
- TASK.STacK.DIRecT
- TASK.STacK.Init
- TASK.STacK.RESet
- TASK.STacK.REMove
- TASK.STacK.view
- TASK.STacK.PATtern
- TASK.STacK.PATternGAP

▲ ‘Features’ in ‘Hypervisor Awareness Manual Wind River Hypervisor’
▲ ‘Features’ in ‘OS Awareness Manual AMX’
▲ ‘Features’ in ‘OS Awareness Manual ARTK’
▲ ‘Features’ in ‘OS Awareness Manual Atomthreads’
▲ ‘Features’ in ‘OS Awareness Manual DSP/BIOS’
▲ ‘Features’ in ‘OS Awareness Manual ChibiOS/RT’
▲ ‘Features’ in ‘OS Awareness Manual Chorus Classic’
▲ ‘Features’ in ‘OS Awareness Manual CMX’
▲ ‘Features’ in ‘OS Awareness Manual CMX-TINY+’
▲ ‘Features’ in ‘OS Awareness Manual eCos’
▲ ‘Features’ in ‘OS Awareness Manual embOS’
▲ ‘Features’ in ‘OS Awareness Manual FAMOS’
▲ ‘Features’ in ‘OS Awareness Manual FreeRTOS’
▲ ‘Features’ in ‘OS Awareness Manual Hi7000’
▲ ‘Features’ in ‘OS Awareness Manual Linux’
▲ ‘Features’ in ‘OS Awareness Manual LiteOS’
▲ ‘Features’ in ‘OS Awareness Manual MQX’
▲ ‘Features’ in ‘OS Awareness Manual MTOS-UX’
▲ ‘Features’ in ‘OS Awareness Manual NetBSD’
▲ ‘Features’ in ‘OS Awareness Manual NORTi’
▲ ‘Features’ in ‘OS Awareness Manual Nucleus PLUS’
▲ ‘Features’ in ‘OS Awareness Manual NuttX’
▲ ‘Features’ in ‘OS Awareness Manual OKL4’
▲ ‘Features’ in ‘OS Awareness Manual OSEK/ORTI’
▲ ‘Features’ in ‘OS Awareness Manual OS21’
▲ ‘Features’ in ‘OS Awareness Manual OSEck’
▲ ‘Features’ in ‘OS Awareness Manual OSE Delta’

©1989-2019 Lauterbach GmbH
TASK.STacK.ADD

Add stack space coverage

Format: **TASK.STacK.ADD** [<task_magic>] [<stackrange>]]

Adds one stack area to the **TASK.STacK.view** window.

In single tasking or non-configured multitasking systems, the magic is any number used to identify a stack area. In this case the stack range must be specified as a second parameter.

In configured RTOS systems, the magic number **must** be the task magic number. If the configuration file supplies automatic stack range detection (only possible in some OS's), then the stack range parameter can be omitted. Otherwise specify the stack area manually. If available, you can omit the magic and select a task from a task list.

See also

- TASK.STacK
- TASK.STacK.view
TASK.STacK.DIRection

**Define stack growth direction**

Format:  

```
TASK.STacK.DIRection [UP | DOWN]
```

Defines whether the stack grows downwards or upwards.

- **DOWN**  
  The stack starts with the high address and grows to a lower address.

- **UP**  
  The stack starts with the low address and grows to a higher address.

**See also**
- TASK.STacK
- TASK.STacK.view

TASK.STacK.Init

**Initialize unused stack space**

Format:  

```
TASK.STacK.Init [<task_magic>]
```

If flag system is used on ICE or FIRE:
Resets the write flags on the stack areas. The evaluation for maximum stack space will then start from scratch. If a magic is specified, the write flags of the according stack space will be deleted, otherwise the write flags of all displayed stacks are reset.

If pattern check is used:
Overwrites the currently unused stack space with the pattern defined by `TASK.STacK.PATtern`. The memory starting from the stack pointer onto the stack boundary address (equals to the low address, if the stack grows downwards) will be initialized with the pattern.

**CAUTION:**
If the stack is used in an unusual way, e.g. some stack space is used even if the stack pointer does not point behind the used area, relevant target data may be overwritten, and your application may crash.

**See also**
- TASK.STacK
- TASK.STacK.view

©1989-2019 Lauterbach GmbH
**TASK.STacK.PATtern**

Define stack check pattern

| Format: | TASK.STacK.PATtern [ON | OFF | [%<format>] <pattern>] |

Defines, whether to use flag system or pattern check for stack coverage calculation.

**OFF**

Pattern check is switched off. Stack coverage calculation is done with the flag system (write flags). This option is only available, if a flag system is available.

*Only available on ICE or FIRE systems with flag memory.*

*On ICD, the stack coverage calculation always uses pattern check.*

**ON**

Pattern check is switched on. Stack coverage calculation is done by comparing the stack data with the predefined pattern (see below). The default pattern is zero.

*Only available on ICE or FIRE systems with flag memory.*

*On ICD, the stack coverage calculation always uses pattern check.*

**<pattern>**

Pattern check is switched on. Stack coverage calculation is done by comparing the stack data with defined pattern. The pattern must be the value, which represents unused stack space. This will only work, if the stack space is initialized with this value. Use **TASK.STacK.Init** to re-initialize currently unused stack space with the pattern.

*<pattern>* can also be a string enclosed in quotes.

**<format>**

Use a *<format>* to define formats other than bytes e.g. *

See also

- **TASK.STacK**
- **TASK.STacK.view**

**TASK.STacK.PATternGAP**

Define check pattern gap

| Format: | TASK.STacK.PATternGAP [<value>] |

If the stack check pattern defined with **TASK.STacK.PATtern** is not contiguous, this command defines the gap between two consecutive patterns.

**<value>**

Number of bytes between two consecutive stack check patterns.
**Example:** If the stack is pre-filled with a 4-byte pattern 0xdeadbeef on each 64-byte boundary, specify:

```plaintext
TASK.STacK.PATtern %Long 0xDEADBEEF
TASK.STacK.PATternGAP 0x40-4
```

---

**TASK.STacK.ReMove**

Remove stack space coverage

Format: `TASK.STacK.ReMove [<task_magic>]`

Removes one stack area from the `TASK.STacK.view` window.

`<task_magic>` Specify the task magic number, which is shown in the `TASK.STacK.view` window.

In some configured RTOS systems, you can omit the magic and select a task from a task list.

---

**TASK.STacK.RESet**

Reset stack coverage

Format: `TASK.STacK.RESet [<task_magic>]`

Resets the stack coverage system and all manually defined stack areas.

If flag system is used on ICE or FIRE:
Resets the write flags on the stack areas. The evaluation for maximum stack space will then start from scratch. If a magic is specified, the write flags of the according stack space will be deleted, otherwise the write flags of all displayed stacks are reset.

If pattern check is used:
Resets the defined pattern to zero.
Opens a window with stack space coverage.

**Format:**  
```bash  
TASK.STacK.view [<task_magic> [<stackrange>]] [/HumanReadable]  
```  

**<task_magic>**  
In single tasking systems, or in non-supported multitasking systems, you have to specify the first stack manually. Use any task magic number as an ID, and specify the stack range to cover.

**<stackrange>**  
If the RTOS configuration file supports detection of the stack range, you can use the magic of a specific task and omit the stack range. The range will be automatically calculated from the information of the operating system. In the case of a fully supported operating system, you can start the window without any parameter. The debugger then automatically adds all current active tasks with its stacks to the window.

**HumanReadable**  
Shows the size of the stack and the spare stack memory in human readable form (byte, kilobytes, megabytes).

### Description of Columns in the TASK.STacK.view Window

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>Name or ID for the stack space. In configured RTOS environment it specifies the name or ID of the task.</td>
</tr>
<tr>
<td>low and high</td>
<td>The lowest and highest address of the stack range.</td>
</tr>
<tr>
<td>sp (gray)</td>
<td>Gray: The stack pointer, calculated from a task control block (if available).</td>
</tr>
<tr>
<td>sp (black)</td>
<td>Black: The current value of the stack pointer register when the application is halted. In non-configured systems, the black value is displayed at the stack, where the current sp fits inside the stack borders. In configured RTOS systems the sp is shown at the current running task. If no according stack could be found, sp appears in an extra line at the end.</td>
</tr>
<tr>
<td>sp (red)</td>
<td>Red: Either if the current sp does not fit into the stack range of the current task, or if the sp fits into a stack range that is not the current task.</td>
</tr>
<tr>
<td>%</td>
<td>Percentage of the currently used stack space.</td>
</tr>
<tr>
<td>Column</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>lowest</td>
<td>The lowest used stack address. If using the flag system, it shows the address, at which the first write flag in the stack area appears. If using pattern check, it shows the first address, at which the date is not equal to the pattern.</td>
</tr>
<tr>
<td>spare</td>
<td>Amount of bytes not used in the stack area.</td>
</tr>
<tr>
<td>max</td>
<td>The maximum stack space used in percent (calculated from 'lowest'). The following bar shows this percentage graphically.</td>
</tr>
</tbody>
</table>

**TASK.Stop**  
*ICE only*

**See also**
- TASK.STacK
- TASK.STacK.PATtern
- TASK.STacK.ADD
- TASK.STacK.PATternGAP
- TASK.STacK.DIRection
- TASK.STacK.Init
- TASK.STacK.ReMove
- TASK.STacK.RESet

**Format:**  

```
TASK.Stop <task> (deprecated)
```

It was used in 68k ICE only for task selective debugging. **TASK.Stop** marks a task of the selective list to be not stopped.

**See also**
- TASK
  - 'Windows CE Commands’ in ‘OS Awareness Manual Windows CE4/CE5’
  - ‘ZeOS Commands’ in ‘OS Awareness Manual ZeOS’
The TCB (Trace Control Block) is the HW control interface to the MIPS hardware trace block. For details please refer to the MIPS Trace specifications.

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

In the following, TCB specific controlling and associated commands are described.

See also
- TCB.AllBranches
- TCB.CycleAccurate
- TCB.EX
- TCB.IM
- TCB.KE
- TCB.OFF
- TCB.PCTrace
- TCB.PortMode
- TCB.Register
- TCB.SourceSizeBits
- TCB.state
- TCB.SyncPeriod
- TCB.ThreadSizeBits
- TCB.UM
- TCB.CPU
- TCB.DataTrace
- TCB.FCR
- TCB.InstructionCompletionSizeBits
- TCB.LSM
- TCB.ON
- TCB.PortFilter
- TCB.PortWidth
- TCB.RESet
- TCB.STALL
- TCB.SV
- TCB.TC
- TCB.Type
- TCB.Version
## TCB.AllBranches

**Broadcast all branches**

| Format:   | TCB.AllBranches [ON | OFF] |
|-----------|---------------------|

<table>
<thead>
<tr>
<th>OFF</th>
<th>The TCB broadcasts only the address information when the processor branches to a location that cannot be directly inferred from the source code.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(default)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ON</th>
<th>The TCB broadcasts the address information for all branches or jumps.</th>
</tr>
</thead>
</table>

See also
- TCB
- TCB.state

## TCB.CPU

**Broadcast information for specified CPU only**

<table>
<thead>
<tr>
<th>Format:</th>
<th>TCB.CPU ALL</th>
<th>&lt;cpu_x&gt;</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>&lt;cpu_x&gt;:</th>
<th>CPU0</th>
<th>CPU1</th>
</tr>
</thead>
</table>

The TCB broadcasts only information for the specified CPU.

<table>
<thead>
<tr>
<th>ALL</th>
<th>The TCB broadcasts information for executed instructions of all active CPU’s.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(default)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;cpu_x&gt;</th>
<th>The TCB broadcasts only information for executed instructions of &lt;cpu_x&gt;.</th>
</tr>
</thead>
</table>

See also
- TCB
- TCB.state

©1989-2019 Lauterbach GmbH
Cycle accurate tracing can be used to observe the exact number of cycles that a particular code sequence takes to execute. If cycle accurate tracing is used, trace information is generated for each clock cycle. In this case the `<core_clock>` can be used to calculate the time stamps for the trace information.

| Format: | TCB.CycleAccurate [ON | OFF] |
|---------|-------------------------------|
| ON      | The TCB broadcasts the information which instructions were executed, but additionally stall information. No time stamps are generated by TRACE32. |
| OFF (default) | The TCB broadcasts only the information which instructions were executed. Time stamps are generated by TRACE32. |

**Example:**

```
TCB.CycleAccurate ON

Trace.CLOCK 500.MHz ; specify the <core_clock> as
Trace.List ; base for the trace time stamps
; display the trace information
```

**See also**
- TCB
- TCB.state
### TCB.DataTrace

Broadcast specified address and data information.

<table>
<thead>
<tr>
<th>Format:</th>
<th>TCB.DataTrace &lt;def&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;def&gt;:</td>
<td>ON</td>
</tr>
</tbody>
</table>

The TCB broadcasts only specified address and data information.

| | The TCB broadcasts all address and data information. |
| ON | The TCB broadcasts all address and data information. |
| OFF (default) | The TCB broadcasts no address and data but only PC information. |
| Address | The TCB broadcasts all address information. |
| ReadAddress | The TCB broadcasts only address information in case of a read. |
| WriteAddress | The TCB broadcasts only address information in case of a write. |
| Data | The TCB broadcasts all data information. |
| ReadData | The TCB broadcasts only data information in case of a read. |
| WriteData | The TCB broadcasts only data information in case of a write. |
| Read | The TCB broadcasts address and data information in case of a read. |
| Write | The TCB broadcasts address and data information in case of a write. |

**See also**

- TCB
- TCB.state
TCB.EX  Broadcast exception level information

Format: TCB.EX [ON | OFF]

If enabled the TCB broadcasts information for instructions executed on exception level.

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON (default)</td>
<td>The TCB broadcast information for executed instructions in exception operating mode.</td>
</tr>
<tr>
<td>OFF</td>
<td>The TCB does not broadcast information for executed instructions in exception operating mode.</td>
</tr>
</tbody>
</table>

See also
- TCB
- TCB.state

TCB.FCR  Broadcast function call-return information

Format: TCB.FCR [ON | OFF]

Enables broadcasting of function call-return information. This information is not treated within TRACE32 PowerView but has to be taken into account for trace decoding especially in case of a belated trace analysis.

See also
- TCB
- TCB.state

TCB.IM  Broadcast instruction cache miss information

Format: TCB.IM [ON | OFF]

Enables broadcasting of instruction cache miss information. This information is not treated within TRACE32 PowerView but has to be taken into account for trace decoding especially in case of a belated trace analysis.

See also
- TCB
- TCB.state
TCB.InstructionCompletionSizeBits

Specify size of completion message

This command is only required if a TRACE32 Instruction Set Simulator is used for a belated analysis of SMP trace information.

This command allows to specify how many bits are used in the trace stream dot instruction completion message.

See also
- TCB
- TCB.state

Format:

```
TCB.InstructionCompletionSizeBits <number>
```

TCB.KE

Broadcast kernel mode information

If enabled the TCB broadcasts information for instructions executed in kernel mode.

<table>
<thead>
<tr>
<th>OFF</th>
<th>The TCB does not broadcast information for executed instructions in kernel operating mode.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON (default)</td>
<td>The TCB broadcast information for executed instructions in kernel operating mode.</td>
</tr>
</tbody>
</table>

See also
- TCB
- TCB.state

©1989-2019 Lauterbach GmbH
TCB.LSM

Broadcast load store data cache information

| Format: | TCB.LSM [ON | OFF] |

Enables broadcasting of load store data cache miss information. This information is not treated within TRACE32 PowerView but has to be taken into account for trace decoding especially in case of a belated trace analysis.

See also
- TCB
- TCB.state

TCB.OFF

Switch TCB off

| Format: | TCB.OFF |

Disables TCB functionality.

See also
- TCB
- TCB.state

TCB.ON

Switch TCB on

| Format: | TCB.ON |

Enables TCB functionality.

See also
- TCB
- TCB.state
TCB.PCTrace
Broadcast program counter trace

If enabled, the TCB broadcasts program counter trace information.

| Format: TCB.PCTrace [ON | OFF] |
|-------------------------------|
| OFF                           | The TCB does not broadcast program counter trace information. |
| ON (default)                  | The TCB broadcast program counter trace information. |

See also
- TCB
- TCB.state

TCB.PortFilter
Disable port filter

Format: TCB.PortFilter [ON | OFF]

Disable filtering of trace data within AutoFocus II Preprocessor. This command should only be used for diagnostic purpose!

See also
- TCB
- TCB.state
**TCB.PortMode**

Specify trace clock ratio

<table>
<thead>
<tr>
<th>Format:</th>
<th>TCB.PortMode &lt;trace_clock&gt;/&lt;cpu_clock&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;trace_clock&gt;</td>
<td>8/1</td>
</tr>
<tr>
<td>/&lt;cpu_clock&gt;:</td>
<td></td>
</tr>
</tbody>
</table>

Specify the ratio between trace- and CPU clock in case of off-chip trace.

```
TCB.PortMode 1/2 ; <trace_clock> is one half of <core_clock>.
```

See also
- TCB
- TCB.state

**TCB.PortWidth**

Specify trace port width

<table>
<thead>
<tr>
<th>Format:</th>
<th>TCB.PortWidth &lt;width&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;width&gt;:</td>
<td>4</td>
</tr>
</tbody>
</table>

Specify the trace port width in number of bits. This value is determined automatically by selecting trace method or reading trace configuration register from target. Therefore this command should only be used for diagnosis purpose or if necessary belated trace analysis.

See also
- TCB
- TCB.state
Format: **TCB.Register** [<file>] [/<option>]

**<option>:**

- **SpotLight**
- **DualPort**
- **Track**
- **CORE** `<core_number>`
- **Deport**

Default: OFF.

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

**Deport**

Updates the control registers while the program is running (only possible if **SYStem.MemAccess CPU** is selected).

**TCB.Register permipstcb.per**

; display the TCB control registers

; use the format description in

; permipstcb.per

**TCB.Register, /SpotLight**

; display the TCB control registers

; mark changes on the registers

See also

- TCB
- TCB.state

©1989-2019 Lauterbach GmbH

**General Commands Reference Guide T** 63
TCB.RESet

Reset TCB setup to default

| Format: | TCB.RESet |

Resets the TCB settings to default.

See also
- TCB
- TCB.state

TCB.SourceSizeBits

Specify number of bit for core information in trace

| Format: | TCB.SourceSizeBits <number> |

This command is only required if a TRACE32 Instruction Set Simulator is used for a belated analysis of SMP trace information.

This command allows to specify how many bits are used in the trace stream to identify the source core.

See also
- TCB
- TCB.state

TCB.STALL

Stall CPU for complete trace

| Format: | TCB.STALL [ON | OFF] |

If enabled, TCB broadcasts slow but complete trace information.

| OFF       | The TCB broadcasts trace information in real-time with the risk of broken trace flow. |
| ON (default) | The TCB stall CPU if necessary and broadcast always complete information. |

See also
- TCB
- TCB.state
**TCB.state**  
Display TCB setup

Format:  
**TCB.state**

Displays the TCB configuration window.

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

See also
- **TCB**
- **TCB.CPU**
- **TCB.DataTrace**
- **TCB.FCR**
- **TCB.InstructionCompletionSizeBits**
- **TCB.LSM**
- **TCB.ON**
- **TCB.PortFilter**
- **TCB.PortWidth**
- **TCB.RESet**
- **TCB.STALL**
- **TCB.SyncPeriod**
- **TCB.ThreadSizeBits**
- **TCB.UM**
- **TCB.AllBranches**
- **TCB.CycleAccurate**
- **TCB.EX**
- **TCB.IM**
- **TCB.KE**
- **TCB.OFF**
- **TCB.PCTrace**
- **TCB.PortMode**
- **TCB.Register**
- **TCB.SourceSizeBits**
- **TCB.SV**
- **TCB.TC**
- **TCB.Type**
- **TCB.Version**
TCB.SV

Broadcast supervisor mode information

Format:         TCB.SV [ON | OFF]

If enabled the TCB broadcasts information for instructions executed in supervisor mode.

| ON          | The TCB broadcast information for executed instructions in supervisor operating mode. |
| OFF        | The TCB does not broadcast information for executed instructions in supervisor operating mode. |

See also
■ TCB
■ TCB.state

TCB.SyncPeriod

Specify TCB sync period

Format:         TCB.SyncPeriod <period>

<period>: 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

Specify the period in cycles the TCB broadcasts a synchronization message.

<period> The TCB sync period in \(2^{(<period> + 5)}\) cycles.

See also
■ TCB
■ TCB.state
TCB.TC
Broadcast information for specified HW thread

<table>
<thead>
<tr>
<th>Format:</th>
<th>TCB.TC ALL</th>
<th>&lt;tc_x&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;tc_x&gt;:</td>
<td>TC0</td>
<td>TC1</td>
</tr>
</tbody>
</table>

The TCB broadcasts only information for the specified HW thread.

<table>
<thead>
<tr>
<th>ALL</th>
<th>The TCB broadcasts information for executed instructions of all active TCs.</th>
</tr>
</thead>
<tbody>
<tr>
<td>(default)</td>
<td></td>
</tr>
<tr>
<td>&lt;tc_x&gt;</td>
<td>The TCB broadcasts only information for executed instructions of &lt;tc_x&gt;.</td>
</tr>
</tbody>
</table>

See also
- TCB
- TCB.state

TCB.ThreadSizeBits
Specify number of bit for thread information in trace

<table>
<thead>
<tr>
<th>Format:</th>
<th>TCB.ThreadSizeBits &lt;number&gt;</th>
</tr>
</thead>
</table>

This command is only required if a TRACE32 Instruction Set Simulator is used for a belated analysis of SMP trace information.

This command allows to specify how many bits are used in the trace stream to identify the source thread context.

See also
- TCB
- TCB.state
**TCB.Type**  
Specify TCB type

<table>
<thead>
<tr>
<th>Format:</th>
<th>TCB.Type</th>
<th><code>&lt;tcb_type&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;tcb_type&gt;</code>:</td>
<td>PD</td>
<td>PD74K</td>
</tr>
<tr>
<td>PD</td>
<td>MIPS standard program and data trace control block.</td>
<td></td>
</tr>
<tr>
<td>PD74K</td>
<td>Specific MIPS74K program data trace control block.</td>
<td></td>
</tr>
<tr>
<td>IFLOW</td>
<td>MIPS standard instruction flow trace control block.</td>
<td></td>
</tr>
<tr>
<td>FALCON</td>
<td>Lantiq specific instruction flow trace control block.</td>
<td></td>
</tr>
<tr>
<td>ZEPHYR</td>
<td>Broadcom specific program and data trace control block.</td>
<td></td>
</tr>
</tbody>
</table>

This command is only required if a TRACE32 Instruction Set Simulator is used for a belated analysis of SMP trace information.

**TCB.UM**  
Broadcast user mode information

| Format: | TCB.UM | [ON | OFF] |

If enabled the TCB broadcasts information for instructions executed in user mode.

| ON (default) | The TCB broadcast information for executed instructions in user operating mode. |
| OFF | The TCB does not broadcast information for executed instructions in user operating mode. |

**See also**
- TCB
- TCB.state

©1989-2019 Lauterbach GmbH

General Commands Reference Guide T 68
TCB.Version

Specify trace cell version

Format:  **TCB.Version**  <number>

This command is only required if a TRACE32 Instruction Set Simulator is used for a belated trace analysis. This command allows to specify manually the version number of the TCB trace cell. The version number must fit to the TCB the trace data have been recorded with. It could be found in the header of the TCB window if TRACE32 is connected to the referring target.

See also

- TCB
- TCB.state
Overview TERM

Multitasking operating systems or monitor programs running on the target system often need a terminal interface for operation. This interface can be implemented either using peripherals (e.g. serial port) or as a memory based interface. The memory based interface can work in several operation modes. It can communicate either on character basis or with blocks of up to 255 bytes length. The memory access can either be made while the target is running (when the system supports such run-time memory accesses) or only when the target is stopped.

When the EPROM simulator (ESI) is used, the ESI can be used as communication port as well. Some processor architectures also provide a special communication interface which is accessible through the BDM/JTAG port (DCC modes).

The standard terminal window provides only the basic functions **Backspace**, **Return** and **LineFeed**. A VT100 emulation mode is also available. A character can only be entered when the cursor is positioned in an active window. The terminal window may also be used for “virtual hosting”. This allows to access some basic operation system functions and the file system of the host from the target program. This functionality is only available in the **TERM.GATE** command.
In this section:

- EPROM Simulator
- Single Character Modes
- Buffered Modes
- Serial Line Debugger
- Special Hardware, JTAG

Interface Routines (EPROM Simulator)

This is an example in C to access the terminal window. The address of the ports depends on the width and location of the EPROMs. The example assumes 8-bit wide EPROMs. For 16-bit EPROMs the addresses must be doubled and the types changed from char to short.

```c
extern volatile unsigned char input_port at 0x1000;
extern volatile unsigned char status_port at 0x1400;
extern volatile unsigned char output_port[256] at 0x1800;

void char_out(c)
unsigned char c;
{
    unsigned char dummy;
    if ( c == 0 )                      /* refuse to send 0 (break) character */
        return;
    while ( status_port&2);           /* wait until port is free */
    dummy = outport_port[c];          /* send character */
}

int char_in();
{
    unsigned char c;                 /* wait until character is ready */
    while (! (status_port&1));       /* read character */
    c = input_port;                  /* manual break executed ? */
    if ( c == 0 )
        break_emulation();
    return c;
}
```
Interface Routines (Single Character Modes)

This interface occupies two memory cells in which characters can be transferred. A zero means that no character is available and the interface is ready. When the target is not able to provide a dual-ported memory access it is possible to stop the target after it has placed a character in the communication area and the terminal command will restart the target automatically after it has processed the character.

This is an example in C to access the terminal window. By changing the char_in and char_out routines within the library, all more complex functions like printf() or scanf() are redirected to the terminal window.

NOTE: Some emulation heads have special dual-port access modes, that require special cycles to be executed (e.g. IDLE mode on H8 probes).

```c
extern volatile char input_port,
output_port

void char_out(c)
{
    char c;
    while (output_port != 0 ) ; /* wait until port is free */
    output_port = c;            /* send character */
}

int char_in();
{
    char c;
    while ( input_port == 0 ) ; /* wait until character is ready */
    c = input_port;            /* read character */
    input_port = 0;            /* clear input port */
    return c;
}
```

Interface Routines (Buffered Modes)

An example for using the buffered mode can be found in `~/demo/etc/terminal/t32term/t32term_memory.c`. This example contains also examples for using the virtual hosting feature of the TERM.GATE command.

Interface Routines (Serial Line Debugger)

The serial line can be used as usual. Only the data values 0 have a special meaning. Receiving such a value means an emulation break. Sending such a value is not allowed for the user program.

Interface Routines (Special Hardware, JTAG)

Check the target appendix for your processor for details and availability.
Functions

TERM.LINE(<address>, <line>)

Returns the line of a displayed terminal window as a string. Negative line numbers retrieve count from the bottom of the window (line -1 is the last line).

Fast Data Write

The fast data write system allows to transfer data from the target to a file on the host. The data transfer rate can be up to 250 KBytes/s. The max. reaction time is 50 µs when the transfer is not interruptible or 150 µs when the transfer is interruptible. Data can be transferred either 8, 16 or 32 bit wide. The principle is similar to the terminal emulation. The interface occupies two memory cells, one byte to control the transfer and a second byte or word to hold the data. A zero in the control cell means that the emulator is ready to accept data. Writing a '01' by the CPU causes the data to be transferred to the host. Writing '02' saves the current data buffer to the host. The time required by this disk save dependents on the host and communication speed. The data buffer is saved automatically after the buffer is full. The value '03' can be used as a NOP command to wait for the start of the transfer. Writing 'ff' terminates the data transfer. The Fast Data Write system has been replaced by the FDX system for ICD and FIRE systems.

Interface Routines

This is an example in C to access the fast data transfer.

```c
extern volatile char control_port;
extern volatile short data_port;

void word_out()
short c;
{
    while (control_port != 0) ;  /* wait until port is free */
data_port = c ;                /* place 16 bit in buffer */
control_port = 1;              /* send data to buffer/host */
}

int begin_transfer(c);
short c;
{
    while (control_port != 0) ;  /* wait until transfer is ready */
}

int end_transfer(c);
short c;
{
    while (control_port != 0) ;  /* wait until port is free */
control_port = 0xff;          /* stop transfer program */
}
```
TERM.CLEAR

Clear terminal window

Format: `TERM.CLEAR [<channel>] [<address>]`

- `<channel>`: `#<number>`

Clears the terminal window and places the cursor to the home position.

See also
- `TERM`
- `TERM.view`

TERM.CLOSE

Close files

Format: `TERM.CLOSE [<channel>] [<address>]`

- `<channel>`: `#<number>`

Closes the output file created with `TERM.WRITE`.

See also
- `TERM`
- `TERM.view`

TERM.CMDLINE

Specify a command line

Format: `TERM.CMDLINE <cmdline>`

The command can specify a command line for the SYS_GET_CMDLINE (0x15) system call if ARM compatible semihosting is used.

See also
- `TERM`
- `TERM.view`
**Format:**

```
TERM.FastWRITE <address> <filename> [/<options>]
```

**<options>:**

- **Byte | Word | Long**
- **BUFFER <size>**
- **NoBreak**
- **Single**

Write data from the user program to a disk file. The interface occupies two memory cells, one byte to control the transfer and a second byte or word to hold the data. The address parameter defines the location of the data byte, the control byte is at `address+4`. A zero in the control cell means that the emulator is ready to accept data. Writing a '01' by the CPU causes the data to be transferred to the host. Writing '02' saves the current data buffer to the host. The time required by this disk save depends on the host and communication speed. The data buffer is saved automatically after the buffer is full. The value '03' can be used as a NOP command to wait for the start of the transfer. Writing 'ff' terminates the data transfer. This command has been superseded by the **FDX** commands (which is available on all systems).

- **Byte, Word, Long**
  Defines the data width of the transfer location.

- **BUFFER**
  Defines the size of the buffer pool. The default size is 4096 bytes. As long as data fits into this pool, the response time to the host will be very short and independent of host and communication speed. If the buffer is full, it is transferred to the host. The time required for this transfer depends on the communication speed and host system.

- **NoBreak**
  Usually the command can be interrupted any time. This communication increases the maximum response time of the emulator. Interruptible transfers have a response time of 150 µs, non interruptible have 50 µs.

- **Single**
  Single shot mode. In this mode the transfer is terminated after the first buffer has been filled.

```plaintext
E::d.s e:0x100 0x3
E::go
E::term.fastwrite e:0x100 data_adc /long /single /buffer 300000.
E::term.close
```

Write 300000 bytes in one shot to the file 'data_adc'. The data is transferred in 32-bit words. The transfer function will stop after receiving all words from the user program.

```plaintext
E::d.s e:0x100 0x3
E::go
E::term.fastwrite e:0x100 data_adc /word /single /buffer 300000. /nobreak
E::term.close
```
Same as above, but data transfer now 16 bit and reducing the max. response time by making the function not interruptible.

See also
- TERM
- TERM.view

TERM.GATE  Terminal with virtual hosting

Format:    TERM.GATE [<channel>] [<output>] [<input>]

<channel>: #<number>

Creates a terminal emulation window which allows virtual hosting.

See also
- TERM
- TERM.view
- 'Release Information' in 'Release History'

TERM.HARDCOPY  Print terminal window contents

Format:    TERM.HARDCOPY [<channel>]

<channel>: #<number>

Opens the Print dialog of the operating system. From the Print dialog, you can select a printer to make a hardcopy of the terminal window contents or print the terminal window contents to file.

See also
- TERM
- TERM.view
TERM.HEAPINFO  Define memory heap parameters

```
Format: TERM.HEAPINFO [<heap_base>] [<heap_limit>] [<stack_base>]
        [<stack_limit>]
```

Defines the memory heap and stack locations returned by the ARM compatible semihosting calls. Only relevant when ARM compatible semihosting is used.

Please note that the heap grows toward higher memory addresses (heap_base < heap_limit) and the stack grows towards lower memory addresses (stack_base > stack_limit). <heap_base> = 0 advises the application to locate the heap at the top of the memory region.

---

See also
- TERM
- TERM.view

---

TERM.LocalEcho  Enables/disables local echo for new terminal windows

```
Format: TERM.LocalEcho [<channel>] [ON | OFF]

<channel>: #<number>
```

Defines, if terminal windows, which are opened after the TERM.LocalEcho command with the TERM.view or TERM.GATE command, will have a local echo or not. Terminal windows with enabled local echo also show the transmitted characters in addition to the received characters.

---

See also
- TERM
- TERM.view
## TERM.METHOD

Select terminal protocol

### Format

```
TERM.METHOD [channel] [method]
```

### <channel>:

```
#<number>
```

### <method>:

- SingleE [output] [input]
- BufferE [output] [input]
- SingleC <pc> [output] [input]
- BufferC <pc> [output] [input]
- SingleS [output] [input]
- BufferS [output] [input]
- COM [name] [baudrate] [bits] [parity] [stopbits] [handshake]
- TCP <host> [port]
- PIPE
- DCC
- DCC3
- DCC4A
- DCC4B
- SIM
- VIRTIO
- CCIO
- BRK1_14 [address]
- ARMSWI [address]
- CHORUS
- ESI
- SERIAL

### <input>:

```
<address>
```

### <output>:

```
<address>
```

### <name>:

Windows:

```
COM1 | COM2 | ... | COM9
```
alternatively (if COMx fails) and for ports >9:

```
\COM1 | \COM2 | ... | \COM10 | \COM11 | ...
```

Linux: path to device, e.g.

```
/dev/ttyS0 | /dev/ttyS1 | /dev/ttyUSB0 | ...
```

### <bits>:

```
5 | 6 | 7 | 8
```
Defines how data is exchanged between the target application and the debugger. On some targets additional processor specific modes may be available.

<table>
<thead>
<tr>
<th>&lt;methods&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleE</td>
<td>Single characters using real time access (e.g. Dualport)</td>
</tr>
<tr>
<td>BufferE</td>
<td>Buffered transfer using real time access</td>
</tr>
<tr>
<td>SingleS</td>
<td>Single characters using regular access at spot points.</td>
</tr>
<tr>
<td>BufferS</td>
<td>Buffered transfer using regular access at spot points.</td>
</tr>
<tr>
<td>BRK1_14</td>
<td>This is a CPU specific option for XTIENSA. For more information, see “CPU specific TERM.METHOD Command” (debugger_xtensa.pdf).</td>
</tr>
<tr>
<td>SingleC</td>
<td>Single characters, accessed when CPU is stopped.</td>
</tr>
<tr>
<td>BufferC</td>
<td>Buffered transfer, accessed when CPU is stopped.</td>
</tr>
<tr>
<td>ESI</td>
<td>Use the ESI for communication. This protocol can also be used when a BDM/JTAG debugger is used together with an ESI (EPROM simulator).</td>
</tr>
<tr>
<td>SERIAL</td>
<td>Use the serial (or ethernet) interface of the debug monitor to exchange data.</td>
</tr>
<tr>
<td>DCC</td>
<td>Use the DCC port of the JTAG interface (only on some architectures)</td>
</tr>
<tr>
<td>DCC3</td>
<td>Same as DCC, but transfer up to 3 characters at once.</td>
</tr>
<tr>
<td>DCC4A</td>
<td>Same as DCC, but transfer up to 4 ascii characters at once.</td>
</tr>
<tr>
<td>DCC4B</td>
<td>Same as DCC, but transfer always 4 characters at once.</td>
</tr>
<tr>
<td>ARMSWI</td>
<td>ARM compatible SWI bases semihosting via SWI breakpoint.</td>
</tr>
<tr>
<td>SIM</td>
<td>Terminal via simulator API.</td>
</tr>
<tr>
<td>VIRTIO</td>
<td>Semihosting of VIRTIO simulation.</td>
</tr>
<tr>
<td>COM</td>
<td>Serial interface of the host.</td>
</tr>
</tbody>
</table>
### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;output&gt;</code></td>
<td>Addresses of the output (target-&gt;debugger) and input (debugger-&gt;target) buffers for memory based terminals.</td>
</tr>
<tr>
<td><code>&lt;host&gt;</code></td>
<td>Host name or IP address of TCP terminal (TELNET)</td>
</tr>
<tr>
<td><code>&lt;port&gt;</code></td>
<td>TCP terminal port number (default: 23)</td>
</tr>
</tbody>
</table>

### Examples:

```
TERM.METHOD BufferE Var.ADDRESS("messagebufferout") \ 
 Var.ADDRESS("messagebufferin")
```

```
TERM.METHOD #1 BufferE Var.ADDRESS("messagebufferout") \ 
 Var.ADDRESS("messagebufferin")
```

### See also

- TERM
- TERM.view

▲ 'Release Information' in 'Release History'
TERM.Mode

Define terminal type

| Format: | TERM.Mode [<channel>] [ASCII | STRING | RAW | HEX | VT100] |
|----------|---------------------------------|
| <channel>: | #<number> |

Defines the terminal type used for new terminal windows.

**ASCII**
Terminal behaves like a typewriter. CR and LF are evaluated.

**STRING**
Terminal interprets data as single line strings. Needed for some Printf libraries. CR is ignored. LF is evaluated.

**RAW**
Terminal shows the incoming data like an HEX/ASCII dump. E.g. Spaces, Tabs, CRs, LFs are displayed as special characters only. CR is ignored. LF is evaluated.

**HEX**
Terminal shows the incoming bytes as HEX values. CR and LF are ignored.

**VT100**
Terminal interprets the VT100 protocol. Color Codes are evaluated e.g. Linux bash like console. CR and LF are evaluated.

---

**See also**

- TERM
- TERM.view

▲ 'Release Information' in 'Release History'
TERM.Out

Send data to virtual terminal

Format:  TERM.Out [<channel>] [<addr_in>] <string> …

<channel>:  #<number>

Sends characters to a terminal. Can be used to control the terminal through a PRACTICE script (*.cmm) or to input non-printable characters from command line.

Example:

; configure u-boot through serial terminal
TERM.METHOD COM COM1 115200 . 8 NONE 1STOP NONE
TERM.view
TERM.Out 10. ; send a single line feed
TERM.Out "setenv bootcmd bootm 0xfe000000 0xfe800000 0xffe00000" 10.
TERM.Out "setenv bootargs root=/dev/ram console=ttyS0,115200" 10.
TERM.Out "saveenv" 10.

See also
■ TERM
■ TERM.view

TERM.PIPE

Connect terminal to named pipe

Format:  TERM.PIPE [<channel>] [<output>] [<input>] <pipename>

<channel>:  #<number>

Connects the terminal to a bidirectional named pipe.

See also
■ TERM
■ TERM.view
TERM.PipeREAD Connect terminal input to named pipe

Format: \texttt{TERM.PipeREAD [\textit{channel}] [\textit{input}] \textit{file}}

\texttt{\textit{channel}}: \#\textit{number}

Connects the terminal to a pipe which sends data to the host.

\textbf{See also}
- TERM
- TERM.view

TERM.PipeWRITE Connect terminal output to named pipe

Format: \texttt{TERM.PipeWRITE [\textit{channel}] [\textit{output}] \textit{file}}

\texttt{\textit{channel}}: \#\textit{number}

Connects the terminal to a pipe which receives data from the host.

\textbf{See also}
- TERM
- TERM.view

TERM.Protocol Select terminal protocol

Format: \texttt{TERM.Protocol (deprecated)}

\textbf{Use} \texttt{TERM.METHOD} \textbf{instead.}

\textbf{See also}
- TERM
- TERM.view
TERM.PULSE

Enable pulse generator for transfers

Format:

TERM.PULSE [channel] [ON | OFF]

<channel>:

#<number>

Issues a pulse on the exception pulse generator (ICE/FIRE) or PODBUS trigger after each transfer. This pulse may be used to trigger an interrupt on the target system to trigger interrupt based communication.

See also

TERM
TERM.view

TERM.Rate

Define polling rate

FIRE / ICE only

Format:

TERM.Rate [samples/s]

<channel>:

#<number>

Defines the poll interval for the terminal (in dual-ported communication protocols on some hardware configurations). This allows to guarantee a minimum reaction time to outputs to the terminal.

Example:

E::TERM.Rate 1000. ; poll at least every millisecond

See also

TERM
TERM.view
**TERM.READ**

Get terminal input from file

Format: `TERM.READ [<channel>] [<input>] <filename>`

- `<channel>`: `#<number>`

The contents of the file are send to the terminal, defined by the optional address. The terminal must already exist to use this command. The `TERM.CLOSE` command closes the input file after or during transfer.

**Example:**

```plaintext
E::TERM.READ e:0x101 key_input.in
```

See also
- `TERM`
- `TERM.view`

---

**TERM.RESet**

Reset terminal parameters

Format: `TERM.RESet [<channel>]`

- `<channel>`: `#<number>`

Closes the I/O redirection files and set all parameters to default values.

See also
- `TERM`
- `TERM.view`
TERM.SCROLL
Enable automatic scrolling for terminal window

Format:  TERM.SCROLL [<channel>] [ON | OFF]

<channel>:  #<number>

Default: OFF.

Enables or disables automatic scrolling. With automatic scrolling enabled the visible window will follow the terminal cursor.

See also
- TERM
- TERM.view

TERM.SIZE
Define size of terminal window

Format:  TERM.SIZE [<channel>] [<columns>] [<lines>] [<backlog_size>]

<channel>:  #<number>

Defines the size of the virtual terminal in lines and columns.

<backlog_size> This value defines the lines of the backlog buffer.
The backlog is updated whenever a line scrolls out of the “real” part of the TERM.view window.

See also
- TERM
- TERM.view
- ‘Release Information’ in ‘Release History’
TERM.TCP

Route terminal input/output to TCP port

Format: TERM.TCP [<channel>] <port>

<channel>: #<number>

Routes terminal input/output to TCP port.

See also
- TERM
- TERM.view

TERM.TELNET

Open TELNET terminal window

Format: TERM.TELNET [<channel>]

<channel>: #<number>

Opens the terminal emulation window for TELNET.

TERM.METHOD TCP 10.2.23.140 ;using default port 23
TERM.MODE VT100
TERM.TELNET

See also
- TERM
- TERM.view

▲ 'Release Information' in 'Release History'
TERM.TRIGGER

Trigger on string in terminal window

Format:

TERM.TRIGGER [<channel>] [<addr_out>] <string>

Example:

Sets a trigger for the occurrence of a specific string (case sensitive) in the terminal window. The function TERM.TRIGGERED() returns if the trigger has occurred or not.

Example:

<addr_out> Only required for memory-based data exchange (SingleE, BufferE, SingleS, BufferS).
**Example:** A typical use case might be to automatize the boot process. The following script stops the boot process after the string “Hit any key to stop autoboot” appears in the terminal window.

```plaintext
; <example_terminal_output>
; U-Boot <year>.<month>
; CPU: example CPU
; Board: example Board
; Boot: SD-Card
; DRAM: 2 GiB
; MMC: SDHC: 0
; In: serial
; Out: serial
; Err: serial
; Normal Boot
; Hit any key to stop autoboot: 3
; </example_terminal_output>

; STATE.RUN() -> STOPPED
Break

; simple example - wait for trigger, press ENTER
TERM.TRIGGER "Hit any key"
; start CPU
Go
SCREEN.WAIT TERM.TRIGGERED(D:0)
TERM.OUT 0xA

; advanced example - wait for trigger with timeout, press ENTER
TERM.TRIGGER "Hit any key"
; start CPU
Go
SCREEN.WAIT TERM.TRIGGERED(D:0) 10.s
IF !TIMEOUT()
  (TERM.OUT 0xA
   WAIT 0.1s
   TERM.OUT "setenv bootargs ...."
  )
ELSE
  (
    ; error handler
  )
```

**See also**
- **TERM**
- **TERM.view**
- **TERM.TRIGGERED()**

©1989-2019 Lauterbach GmbH
TERM.Vector

Define interrupt vectors

ICE only

Format: TERM.Vector [<channel>] [<tx_vector>] [<rx_vector>]

See also
TERM
TERM.view

TERM.view

Terminal display

Opens the terminal emulation window. The protocol of the terminal is defined through TERM.METHOD. For protocols based on memory based data exchange (SingleE, BufferE, SingleS, BufferS), the communication buffer addresses can either be specified with TERM.METHOD or directly with TERM.view.

Example:

; see terminal source code in
; ~/demo/etc/terminal/t32term/t32term_memory.c
TERM.METHOD BufferE
TERM.MODE VT100
TERM.view E:0x00000100 E:0x00000200

; Hint: the prefixes WinExt and WinResist create a window “external” to
; the TRACE32 PowerView main window that is “resistant” to the WinCLEAR
; command.
WinExt.WinResist.TERM.view

See also
TERM
TERM.Clear
TERM.Gate
TERM.Method
TERM.Close
TERM.HARDCOPY
TERM.Heap
TERM.LocalEcho
TERM.METHOD
**TERM.WRITE**

Write terminal output to file

<table>
<thead>
<tr>
<th>Format:</th>
<th>TERM.WRITE [&lt;output&gt;] &lt;filename&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;channel&gt;:</td>
<td>#&lt;number&gt;</td>
</tr>
</tbody>
</table>

The output sent from the target to the terminal emulation window is written to the specified file. The terminal emulation window must be opened before using this command. The **TERM.CLOSE** command closes the output file after or during transfer.

Example:

```
TERM.WRITE E:0x00000100 term_out.lst
```

See also

- TERM
- TERM.view

▲ 'Release Information' in 'Release History'
Overview TPIU

The **TPIU** command group enables you to configure and control the Trace Port Interface Unit (TPIU) of an ARM processor system or a non-ARM processor system using the ARM CoreSight trace. The TPIU is a trace sink which sends the trace data off-chip for capturing by a trace tool.

The TPIU typically outputs trace data via a parallel trace interface consisting of up to 32 trace data signals, a trace clock and optionally a trace control signal (indicating idle).

Some chip designs use these signals internally as an input to a High Speed Serial Trace Port (HSSTP) which converts the parallel data into a serial Xilinx-Aurora-based protocol for sending the serial bit stream off-chip on differential lanes.

A variant of the TPIU is the Serial Wire Output (SWO) which outputs trace data of the Serial Wire Viewer (SWV) via a single signal line. This output has a much lower bandwidth, is typically used for system trace, and is typically found on Cortex-M based designs. This variant does normally not use a dedicated trace connector. Instead it re-uses the TDO pin of a debug connector.

For TPIU setup, use the TRACE32 command line, a PRACTICE script (*.cmm), or the **TPIU.state** window.
**TPIU.CLEAR**  
Re-write the TPIU registers

Re-writes the TPIU registers on the target with the settings displayed on the TPIU.state window.

**See also**  
- TPIU  
- TPIU.state

---

**TPIU.IGNOREZEROS**  
Workaround for a special chip

**Format:**  
TPIU.IGNOREZEROS [ON | OFF]

**See also**  
- TPIU  
- TPIU.state

---

**TPIU.NOFLUSH**  
Workaround for a chip bug affecting TPIU flush

**Default:** OFF.

Activates a workaround for a chip bug which caused serious issues when the trace tool caused a TPIU flush at the end of the trace recording.

**See also**  
- TPIU  
- TPIU.state
TPIU.PortClock

Inform debugger about HSSTP trace frequency

Default: 1500Mbps

Informs the debugger about the HSSTP trace frequency to improve the accuracy of the timestamp calculation.

**Example:**

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPIU.PortClock 3125Mbps</td>
<td>Informs the debugger about the HSSTP trace frequency to improve accuracy.</td>
</tr>
<tr>
<td>TPIU.PortClock 3125M</td>
<td>; M is the short form of Mbps</td>
</tr>
</tbody>
</table>

**See also**
- TPIU
- TPIU.state
Selects the operation mode of the TPIU.

### Modes for Parallel Trace and HSSTP

The TPIU can optionally output a trace control signal (TRACECTL) which indicates idle cycles of the trace port not worth to record. The TPIU formatter can be used to add the idle information to the trace packets. The formatter needs to be used in case of multiple trace sources to add the ID of the trace source.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bypass</td>
<td>TRACECTL pin is available, formatter is not used.</td>
</tr>
<tr>
<td>Wrapped</td>
<td>TRACECTL pin is available, formatter is used.</td>
</tr>
<tr>
<td>Continuous</td>
<td>TRACECTL pin is not available, formatter is used.</td>
</tr>
</tbody>
</table>

### Modes for Serial Wire Output

TRACE32 supports the UART/NRZ (NRZ = Non-Return-to-Zero) coding of the Serial Wire Output but not yet the Manchester coding. The bitrate of this asynchronous interface is derived by dividing the CPU frequency.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRZ</td>
<td>NRZ coding at CPU clock divided by <code>&lt;divisor&gt;</code> set up by:</td>
</tr>
<tr>
<td></td>
<td><code>TPIU.SWVPrescaler &lt;divisor&gt;</code> (default: 1)</td>
</tr>
<tr>
<td>NRZ/2 (deprecated)</td>
<td>See example below.</td>
</tr>
<tr>
<td>NRZ/3 (deprecated)</td>
<td>See example below.</td>
</tr>
<tr>
<td>NRZ/4 (deprecated)</td>
<td>See example below.</td>
</tr>
</tbody>
</table>

**Example:**

```
;;; (deprecated)
TPIU.PortMode NRZ/4

;please use these two commands instead of NRZ/<divisor>
TPIU.PortMode NRZ
TPIU.SWVPrescaler 4.
```
TPIU.PortSize

Select interface type and port size of the TPIU

<table>
<thead>
<tr>
<th>Format:</th>
<th>TPIU.PortSize &lt;size&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;size&gt;:</td>
<td>1</td>
</tr>
</tbody>
</table>

Specifies the interface type and port size of the TPIU.

### Size in case of Parallel Trace:

| 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 16, 18, 20, 24, 32 | Number of trace data signals. TRACE32 supports the listed sizes. A TPIU can support all sizes from 1 to 32 or only a few out of 1 to 32. |
| 8A, 12A, 16A, 16E | Variants of “8”, “12”, “16” in case of SoC from Texas Instruments. The selected size is the same, but additionally the Debug Resource Manager (DRM) gets configured which maps trace signals to output pins: |
| | • 16A: TRACEDATA[0:15] -> EMU[4:19] |

### Size in case of HSSTP:

| 1Lane, 2Lane, 3Lane, 4Lane, 5Lane, 6Lane | Number of used differential lanes. |

### Size in case of Serial Wire Viewer (SWV) / Serial Wire Output (SWO):

| SWV | Selects SWV/SWO which uses only one signal. |

See also

- TPIU
- TPIU.state
- <trace>.PortSize
**TPIU.RefClock**

Set up reference clock for HSSTP

<table>
<thead>
<tr>
<th>Format:</th>
<th>TPIU.RefClock [/&lt;option&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;option&gt;:</td>
<td>OFF</td>
</tr>
</tbody>
</table>

Defines the reference clock frequency the serial preprocessor outputs to the target. Defaults depending on architecture:

- PowerPC: bit clock frequency
- TriCore and RH850: 100MHz
- ARM: bit clock frequency

<table>
<thead>
<tr>
<th>OFF</th>
<th>TRACE32 does not send any reference clock to the target.</th>
</tr>
</thead>
<tbody>
<tr>
<td>OSC</td>
<td>An asynchronous oscillator will be enabled. Its frequency is architecture dependent.</td>
</tr>
<tr>
<td>1/&lt;x&gt;</td>
<td>A synchronous clock source will be enabled. Its dividers generate a reference clock as a fraction of the bit clock (lane speed), e.g. 100MHz at 5Gbps with divider 1/50. Once a divider is selected, the reference clock will automatically change with the lane speed.</td>
</tr>
</tbody>
</table>

**See also**
- TPIU
- TPIU.state

**TPIU.Register**

Display TPIU registers

<table>
<thead>
<tr>
<th>Format:</th>
<th>TPIU.Register [/&lt;option&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;option&gt;:</td>
<td>SpotLight</td>
</tr>
</tbody>
</table>

Opens the **TPIU.Register** window, displaying the TPIU registers and the registers of other trace related modules.

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

**See also**
- TPIU
- TPIU.state
TPIU.RESet

Reset TPIU settings

Format: TPIU.RESet

Resets the settings in the **TPIU.state** window to their default values and re-configures the TPIU registers on the target.

See also
- TPIU
- TPIU.state

TPIU.state

Display TPIU configuration window

Format: TPIU.state

Displays the **TPIU.state** configuration window.

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

Exceptions:
- The setting **TPIU.ON** and **TPIU.OFF** is read-only. The setting depends on the selected trace mode (Analyzer, Onchip, ...).
- The **Trace** button opens the main trace control window (**Trace.state**).
- The **List** button the main trace list window (**Trace.List**).

See also
- TPIU
- TPIU.PortClock
- TPIU.Register
- TPIU.SyncPeriod
- TPIU.CLEAR
- TPIU.PortMode
- TPIU.RESet
- TPIU.IGNOREZEROS
- TPIU.PortSize
- TPIU.SWVPrescaler
- TPIU.NOFLUSH
- TPIU.RefClock
- TPIU.SWVZEROS
TPIU.SWVPrescaler

Set up SWV prescaler

Format: `TPIU.SWVPrescaler <divisor>`

Default: 1.

In case of `TPIU.PortMode NRZ`, the bitrate of the Serial Wire Viewer / Serial Wire Output is derived by dividing the CPU frequency. The command `TPIU.SWVPrescaler` sets up the divisor, which can range from 0x1 to 0x1000 (1. to 4096.).

Examples:

```
TPIU.PortMode NRZ
TPIU.SWVPrescaler 7. ; NRZ coding at a 7th of the CPU clock
```

```
TPIU.PortMode NRZ
TPIU.SWVPrescaler 10. ; NRZ coding at a 10th of the CPU clock
```

See also

- `TPIU`
- `TPIU.state`

TPIU.SWVZEROS

Workaround for a chip bug

Format: `TPIU.SWVZEROS [ON | OFF]`

Default: OFF.

Activates a workaround for a chip bug affecting SWV/SWO data of a certain device.

See also

- `TPIU`
- `TPIU.state`
TPIU.SyncPeriod

Set period of sync packet injection

Format: 

\[ \text{TPIU.SyncPeriod [\langle packets\rangle]} \]

Sets the number of regular TPIU packets which will be output to the trace stream between two synchronization packets.

**What are synchronization packets?** Synchronization packets are periodic starting points in the trace stream, which allow the recorded flow trace data to be decoded. The result can then be visualized in the \(<\text{trace}>.*\) windows of TRACE32, e.g. the \text{Trace.List} or the \text{Trace.PROfileChart.sYmbol} window. A visualization of the flow trace data is usually not possible without synchronization packets in the trace stream.

<table>
<thead>
<tr>
<th>(&lt;\text{packets}&gt;)</th>
<th>If omitted, then the default number of regular packets between synchronization packets is chosen by the debugger or the chip.</th>
</tr>
</thead>
</table>

**In this example,** the number of regular packets is 1024.

\[
\begin{array}{cccc}
\text{RP} & \text{RP} & \text{SP} & \text{RP} \\
1024 & 1024 & 1024 & 1024
\end{array}
\]

RP = regular packet  
SP = synchronization packet

**See also**

- TPIU  
- TPIU.state

©1989-2019 Lauterbach GmbH
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPU.BASE</strong></td>
<td>Base address</td>
</tr>
<tr>
<td></td>
<td>See command [TPU.BASE](tpu.pdf, page 4).</td>
</tr>
<tr>
<td><strong>TPU.Break</strong></td>
<td>Break TPU</td>
</tr>
<tr>
<td></td>
<td>See command [TPU.Break](tpu.pdf, page 10).</td>
</tr>
<tr>
<td><strong>TPU.Dump</strong></td>
<td>Memory display</td>
</tr>
<tr>
<td></td>
<td>See command [TPU.Dump](tpu.pdf, page 8).</td>
</tr>
<tr>
<td><strong>TPU.Go</strong></td>
<td>Start TPU</td>
</tr>
<tr>
<td></td>
<td>See command [TPU.Go](tpu.pdf, page 10).</td>
</tr>
<tr>
<td><strong>TPU.Register.ALL</strong></td>
<td>Register operation mode</td>
</tr>
<tr>
<td></td>
<td>See command [TPU.Register.ALL](tpu.pdf, page 5).</td>
</tr>
<tr>
<td><strong>TPU.Register.NEWSTEP</strong></td>
<td>New debugging mode</td>
</tr>
<tr>
<td></td>
<td>See command [TPU.Register.NEWSTEP](tpu.pdf, page 6).</td>
</tr>
<tr>
<td><strong>TPU.Register.Set</strong></td>
<td>Register modification</td>
</tr>
<tr>
<td></td>
<td>See command [TPU.Register.Set](tpu.pdf, page 8).</td>
</tr>
</tbody>
</table>
TPU.Register.view

Register display

See command `TPU.Register.view` in ‘TPU Debugger’ (tpu.pdf, page 7).

TPU.RESet

Disable TPU debugger

See command `TPU.RESet` in ‘TPU Debugger’ (tpu.pdf, page 11).

TPU.SCAN

Scanning TPU


TPU.SESelect

Select TPU for debugging

See command `TPU.SESelect` in ‘TPU Debugger’ (tpu.pdf, page 11).

TPU.Step

Single step TPU

See command `TPU.Step` in ‘TPU Debugger’ (tpu.pdf, page 11).

TPU.view

View TPU channels

See command `TPU.view` in ‘TPU Debugger’ (tpu.pdf, page 5).
# Trace

## Trace configuration and display

<table>
<thead>
<tr>
<th>Format: Trace</th>
<th>&lt;trace&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;trace&gt;:</td>
<td>&lt;trace_method&gt;</td>
</tr>
</tbody>
</table>

## Trace

<table>
<thead>
<tr>
<th>Trace</th>
<th>For information, see section <strong>Overview Trace</strong> in this command group description.</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;trace&gt;</td>
<td>For information, see subsection <strong>About the Command Placeholder</strong> &lt;trace&gt; in this command group description</td>
</tr>
<tr>
<td>&lt;trace_method&gt;</td>
<td>For information, see subsection <strong>Replacing &lt;trace&gt; with a Trace Method - Examples</strong> in this command group description.</td>
</tr>
<tr>
<td>&lt;trace_source&gt;&lt;trace_method&gt;</td>
<td>For information, see subsection <strong>Replacing &lt;trace&gt; with Trace Source and Trace Method - Examples</strong> in this command group description.</td>
</tr>
</tbody>
</table>

### NOTE:

There is **NO** period between <trace_source><trace_method>.

This syntax convention is reserved for:
- Processing trace data from only one particular trace source, e.g. ITM.
- Processing trace data from more than one trace source, e.g. ITM and HTM.
- Processing trace data from very special trace sources.

## See also

- <trace>.ACCESS
- <trace>.Arm
- <trace>.AutoFocus
- <trace>.AutoStart
- <trace>.BookMark
- <trace>.ComPare
- <trace>.CustomTraceLoad
- <trace>.DisConfig
- <trace>.EXPORT
- <trace>.Find
- <trace>.FindChange
- <trace>.FLOWSTART
- <trace>.GOTO
- <trace>.List
- <trace>.ListVar
- <trace>.METHOD
- <trace>.OFF
- <trace>.ADDRESS
- <trace>.AutoArm
- <trace>.AutoInit
- <trace>.AutoTEST
- <trace>.Chart
- <trace>.CustomTrace
- <trace>.DISable
- <trace>.DRAW
- <trace>.FILE
- <trace>.FindAll
- <trace>.FLOWPROCESS
- <trace>.Get
- <trace>.Init
- <trace>.ListNesting
- <trace>.LOAD
- <trace>.Mode
- <trace>.Out

©1989-2019 Lauterbach GmbH
Overview Trace

The command Trace is a general command for trace configuration and trace display. It is available for all kind of trace methods provided by TRACE32. The currently used trace method is displayed under METHOD in the Trace.state window.

For descriptions of the trace methods, see <trace>.METHOD.

In this section:

- About the Command Placeholder <trace>
- What to know about the TRACE32 default settings for <trace>
- Types of Replacements for <trace>
- Replacing <trace> with a Trace Method - Examples
- Replacing <trace> with a Trace Evaluation - Example
- Replacing <trace> with RTS for Real-time Profiling - Example
- Replacing <trace> with Trace Source and Trace Method - Examples
- How to access the trace sources in TRACE32
- List of <trace> Command Groups consisting of <trace_source><trace_method>
- Related Trace Command Groups
About the Command Placeholder \texttt{<trace>}

In the TRACE32 manuals, \texttt{<trace>} is used as a placeholder for all types of trace commands. As the name \textit{placeholder} implies, it cannot be used directly in the TRACE32 command line. As soon as you type \texttt{<trace>.List} at the command line, you receive the error message “unknown command”. Consequently, you need to replace \texttt{<trace>} with the correct trace command before the command line accepts your input.

What to know about the TRACE32 default settings for \texttt{<trace>}

The easiest way to replace \texttt{<trace>} with a correct command is to type \texttt{Trace} at the command line. The meaning of \texttt{Trace}, e.g. in \texttt{Trace.List}, is then controlled by a sequence of TRACE32 default settings.

1. The TRACE32 \textit{hardware module} connected to your target board determines the \textit{trace method}. And this trace method will be used for recording the trace data. In the header of the \texttt{Trace.state} window, you can view the selected trace method.

   TRACE32 determines the default trace method as follows:
   - If the hardware module connected to your target board is a PowerTrace, then the \texttt{Analyzer} trace method becomes the default setting.
   - If a hardware module other than a PowerTrace is connected to your target board, TRACE32 adjusts the trace method accordingly.
   - If the chip has an onchip trace sink, then the \texttt{Onchip} trace method becomes the default setting. If the chip does \textit{not} have an onchip trace sink, then the \texttt{ART} trace method becomes the default setting.
   - If TRACE32 runs in software-only mode as an instruction set simulator, then it is again the \texttt{Analyzer} trace method that becomes the default setting.

2. The \texttt{Analyzer} trace method is designed to look for a specific \textit{trace source} that generates the program flow trace on the chip. For ARM chips, this trace source is called Embedded Trace Macrocell (ETM). For other chips, the trace source can be NEXUS or a proprietary trace block.

3. All \texttt{Trace} commands refer to the selected trace method.

In the following figure, the arrows illustrate the default settings used by TRACE32.

*\texttt{ETM}, \texttt{ITM}, and \texttt{HTM} are the names of \texttt{<trace_sources>} on a chip.
You can rely on the trace method that TRACE32 selects by default, but you can also select a trace method other than the default. As soon as you have selected the trace method you want in the Trace.state window, you can replace the placeholder <trace> with:

- **Trace** as explained in the previous section (Click here)
- The name of the trace method you have selected in the Trace.state window (Click here)
- Trace evaluation commands (Click here)
- RTS, the command for real-time profiling (Click here)
- Names of trace sources immediately followed by the name of the trace methods (Click here)

### Replacing <trace> with a Trace Method - Examples

You can replace <trace> with the name of the selected trace method. The trace method commands are displayed in the Trace.state window:

- Analyzer, CAalyzer, Onchip, ART, LOGGER, SNOOPer, FDX, LA, Integrator, Probe, IProbe, HAnalyzer

#### Example 1 for the trace method SNOOPer:

```plaintext
Trace.state ;select the trace method SNOOPer for recording
Trace.METHOD SNOOPer ;trace data.
;<configuration>

;trace data is recorded using the commands Go, WAIT, Break

Trace.List ;display the trace data recorded with SNOOPer
SNOOPer.List ;this is the equivalent and explicit command.
```

#### Example 2 for the trace method LOGGER:

```plaintext
Trace.state ;select the trace method LOGGER for recording
Trace.METHOD LOGGER ;trace data.
;<configuration>

;trace data is recorded using the commands Go, WAIT, Break

Trace.List ;display the trace data recorded with LOGGER
LOGGER.List ;this is the equivalent and explicit command.
```
For trace evaluations, you can replace `<trace>` with a trace evaluation command; the name of the trace method is omitted.

The trace evaluation commands are accessible via the TRACE32 softkey bar:

- COVerage, ISTATistic, MIPS, CTS, ETA, BMC

**Example:**

```
Trace.state ;select the trace method Analyzer for recording
Trace.METHOD Analyzer ;trace data.
;<configuration>

;trace data is recorded using the commands Go, WAIT, Break

COVerage.List ;<trace> is just replaced with the trace
 ;evaluation command, since the trace method
 ;Analyzer is defined above anyway.
```
For real-time profiling, you can replace the placeholder `<trace>` with RTS.

The RTS command is accessible via the TRACE32 softkey bar:

Example:

```
Trace.state ;select the trace method Analyzer for
Trace.METHOD Analyzer ;recording trace data.
;<configuration>

RTS.state
RTS.ON
;<configuration>

Go ;processes the trace data being recorded from
;the target while the target is running.

ISTATistic.ListModule ;ISTATistic windows display real-time
;trace data as long as RTS is switched ON
;(RTS.ON)
```
As stated in the blue Format table, the placeholder `<trace>` can be replaced with trace commands consisting of `<trace_source>` and `<trace_method>`.

![Diagram](image)

These `<trace>` command groups are accessible via the TRACE32 softkey bar and include for example:

- CoreSightTrace, ETMTrace, ETMAnalyzer, STMAnalyzer, CoreSightCAnalyzer, ...
- For an overview, see [List of `<trace>` Command Groups consisting of `<trace_source>`<trace_method>`].

Using these `<trace>` command groups, you can display trace data recorded from one or more trace sources.

**Example for displaying trace data from one trace source**: This script assumes that the CoreSight components of the chip output their trace data to the same trace sink.

```
Trace.state ;select the trace method Analyzer for recording
Trace.METHOD Analyzer ;trace data.
;<configuration>

ETM.ON ;switch on the trace source from which you want
;<configuration> ;to record trace data, here the ETM.

;trace data is recorded using the commands Go, WAIT, Break

Trace.List ;display the ETM trace data recorded with the
;trace method Analyzer as a trace listing.
Analyzer.List ;this is the equivalent and explicit command.
```
**Example for displaying trace data from two trace sources:** This script assumes that the CoreSight components of the chip output their trace data to the same trace sink.

```c
Trace.state ; select the trace method Analyzer for recording
Trace.METHOD Analyzer ; trace data.
;<configuration>

ETM.ON ; switch the 1st trace source ETM on.
;<configuration>

HTM.ON ; switch the 2nd trace source HTM on.
;<configuration>

; trace data is recorded using the commands Go, WAIT, Break
Trace.List ; display the ETM trace data.
HTMTrace.List ; display the HTM trace data.
```
How to access the trace sources in TRACE32

As you have seen in the previous sections, the **Trace.state** window is the starting point for configuring a trace recording and recording the trace data: It provides an overview of the trace methods [A], and it dynamically adjusts to the trace method you have selected [B].

In addition, the **Trace.state** window displays buttons for each trace source found on the chip [C]. Clicking a button lets you access a `<trace_source>.state` window, where you can configure the selected trace source directly in TRACE32.

**Example**: TRACE32 has found three trace sources on a QorIQ chip, including a NEXUS trace source [C]. Click the*NEXUS* button to open the **NEXUS.state** window [D]. You can now configure the NEXUS trace source.
List of `<trace>` Command Groups consisting of `<trace_source><trace_method>`

<table>
<thead>
<tr>
<th>AETAnalyzer</th>
<th>ETMXCAnalyzer</th>
<th>NPKReorderHAnalyzer</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMONTrace</td>
<td>ETMXHAnalyzer</td>
<td>NPKReorderLA</td>
</tr>
<tr>
<td>CIProbe</td>
<td>ETMXLA</td>
<td>NPKReorderTrace</td>
</tr>
<tr>
<td>CoreSightAnalyzer</td>
<td>ETMXOnchip</td>
<td>OCEANAnalyzer</td>
</tr>
<tr>
<td>CoreSightCAnalyzer</td>
<td>ETMXTrace</td>
<td>OCEANOncip</td>
</tr>
<tr>
<td>CoreSightHAnalyzer</td>
<td>FunnelAnalyzer</td>
<td>OCeanTrace</td>
</tr>
<tr>
<td>CoreSightLA</td>
<td>FunnelOnchip</td>
<td>PCPOncip</td>
</tr>
<tr>
<td>CoreSightOnchip</td>
<td>HAalyzer</td>
<td>RTPAnalyzer</td>
</tr>
<tr>
<td>CoreSightOnchip2</td>
<td>HTMAnalyzer</td>
<td>SFTAnalyzer</td>
</tr>
<tr>
<td>CoreSightTrace</td>
<td>HTMCAalyzer</td>
<td>SFTOnchip</td>
</tr>
<tr>
<td>DDRAnalyzer</td>
<td>HTMHAalyzer</td>
<td>SFTTrace</td>
</tr>
<tr>
<td>DDROnchip</td>
<td>HTMLA</td>
<td>STMAnalyzer</td>
</tr>
<tr>
<td>DDRTrace</td>
<td>HTMOncip</td>
<td>STMCAalyzer</td>
</tr>
<tr>
<td>DQMAalyzer</td>
<td>HTMTrace</td>
<td>STMHAalyzer</td>
</tr>
<tr>
<td>DQMONchip</td>
<td>ITHTrace</td>
<td>STMLA</td>
</tr>
<tr>
<td>DQMTTrace</td>
<td>ITMAnalyzer</td>
<td>STMONchip</td>
</tr>
<tr>
<td>DTMAnalyzer</td>
<td>ITMCAalyzer</td>
<td>STMONchip2</td>
</tr>
<tr>
<td>DTMCAnalyzer</td>
<td>ITMHAalyzer</td>
<td>STMTrace</td>
</tr>
<tr>
<td>DTMHAnalyzer</td>
<td>ITMLA</td>
<td>TSIAnalyzer</td>
</tr>
<tr>
<td>ETMAalyzer</td>
<td>ITMOncip</td>
<td>TSIAnalyzer</td>
</tr>
<tr>
<td>ETMCAnalyzer</td>
<td>ITMTrace</td>
<td>TSIHAnalyzer</td>
</tr>
<tr>
<td>ETMDAnalyzer</td>
<td>MCDSBaseAnalyzer</td>
<td>TSILA</td>
</tr>
<tr>
<td>ETMDCAalyzer</td>
<td>MCDSBaseCAalyzer</td>
<td>TSIOnchip</td>
</tr>
<tr>
<td>ETMDHAnalyzer</td>
<td>MCDSBaseOnchip</td>
<td>TSITrace</td>
</tr>
<tr>
<td>ETMDLA</td>
<td>MCDSDCAAnalyzer</td>
<td>UltraSOCHAnalyzer</td>
</tr>
<tr>
<td>ETMDOIOnchip</td>
<td>MCDSDCACAnalyzer</td>
<td>UltraSOCLA</td>
</tr>
<tr>
<td>ETMDTrace</td>
<td>MCDSDCAOnchip</td>
<td>UltraSOCTrace</td>
</tr>
<tr>
<td>ETMHAalyzer</td>
<td>MCDSDDTUAalyzer</td>
<td>XGateOnchip</td>
</tr>
<tr>
<td>ETMLA</td>
<td>MCDSDDTUCAnalyzer</td>
<td>XTICAnalyzer</td>
</tr>
<tr>
<td>Command Group</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>ETMOnchip</td>
<td>Displays and analyzes software messages.</td>
<td></td>
</tr>
<tr>
<td>ETMTrace</td>
<td>Displays and analyzes trace information generated by various trace sources.</td>
<td></td>
</tr>
<tr>
<td>ETMXAnalyzer</td>
<td>Clock management instrumentation trace by Texas Instruments on OMAP4.</td>
<td></td>
</tr>
<tr>
<td>NPKReorderAnalyzer</td>
<td>Power management instrumentation trace by Texas Instruments on OMAP4.</td>
<td></td>
</tr>
<tr>
<td>NPKReorderCAnalyzer</td>
<td>Statistics collector trace by Texas Instruments on OMAP4 and OMAP5.</td>
<td></td>
</tr>
<tr>
<td>PrintfTrace</td>
<td>OpenCore Protocol WatchPoint trace by Texas Instruments on OMAP4 and OMAP5.</td>
<td></td>
</tr>
<tr>
<td>SystemTrace</td>
<td>Real-time processing for HTM trace.</td>
<td></td>
</tr>
<tr>
<td>HTMRTS</td>
<td>SystemTrace</td>
<td></td>
</tr>
<tr>
<td>CMITrace</td>
<td>Clock management instrumentation trace by Texas Instruments on OMAP4.</td>
<td></td>
</tr>
<tr>
<td>PMITrace</td>
<td>Power management instrumentation trace by Texas Instruments on OMAP4.</td>
<td></td>
</tr>
<tr>
<td>StatColTrace</td>
<td>Statistics collector trace by Texas Instruments on OMAP4 and OMAP5.</td>
<td></td>
</tr>
<tr>
<td>OCPTrace</td>
<td>Allows to trace and analyze SYStem.LOG events.</td>
<td></td>
</tr>
</tbody>
</table>

(continued)
### Sampling configuration for probes ABCDEF

**PowerIntegrator only**

<table>
<thead>
<tr>
<th>Format:</th>
<th>Integrator.ABCDEF &lt;option&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>&lt;option&gt;:</strong></td>
<td>500MHZ</td>
</tr>
<tr>
<td></td>
<td>Fixed500MHZ</td>
</tr>
<tr>
<td></td>
<td>250MHZ</td>
</tr>
<tr>
<td></td>
<td>State</td>
</tr>
<tr>
<td></td>
<td>StatePLL</td>
</tr>
<tr>
<td></td>
<td>StatePLLBoth</td>
</tr>
<tr>
<td></td>
<td>CLKA</td>
</tr>
<tr>
<td></td>
<td>CLKB</td>
</tr>
<tr>
<td></td>
<td>Falling</td>
</tr>
<tr>
<td></td>
<td>Rising</td>
</tr>
<tr>
<td></td>
<td>DDR</td>
</tr>
<tr>
<td></td>
<td>SAMPLE</td>
</tr>
</tbody>
</table>

**500MHZ**
Timing Mode 500 MHz for probes ABCDEF only, probes JKLMNO get lost.

**Fixed500MHZ**
Timing Mode fixed sampling with 500 MHz for probes ABCDEF only, probes JKLMNO get lost.

**250MHZ**
Timing Mode 250 MHz.

**State**
State Mode clocked by CLKA or CLKB.

**StatePLL**
State PLL Mode, clocked by CLKA or CLKB.

**StatePLLBoth**
State PLL Mode, clocked by CLKA or CLKB, for all probes.

**CLKA**
Clock A select for State-Mode or State-PLL-Mode.

**CLKB**
Clock B select for State-Mode or State-PLL-Mode.

**Falling**
sampling on falling edge of selected clock CLKA or CLKB.

**Rising**
sampling on rising edge of selected clock CLKA or CLKB.

**DDR**
sampling on rising and falling edge of selected clock CLKA or CLKB.

**SAMPLE**
sampling delay of selected clock CLKA or CLKB (-3 ...+6 ns in steps of 250 ps), State-PLL-Mode only.
The core trace generation logic on the processor/chip generates trace packets to indicate the instruction execution sequence (program flow). TRACE32 merges the following sources of information in order to provide an intuitive display of the instruction execution sequence (flow trace).

- The trace packets recorded.
- The program code from the target memory (usually read via the JTAG interface).
- The symbol and debug information already loaded to TRACE32.

**Format:**

```
<trace>.ACCESS <path>
```

**Recorded trace packets**

Uploaded from the source of trace information

**Program code from target memory**

Read via JTAG interface

**Symbol and debug information loaded to TRACE32**

Trace packets generated by core trace logic
Troubleshooting

1. Trace information should be analyzed while the program execution is running and the debugger has no **run-time access** to the target memory to read the program code.

   NOACCESS in a trace display window indicates that the debugger can not read the target memory.

   You can overcome this problem by loading the program code to the **TRACE32 virtual memory**.

```
; load the program code additional to the TRACE32 virtual memory
; whenever you load it to the target memory
Data.LOAD.Elf diabc.x /PlusVM
```

2. Reading the target via JTAG is very slow therefore all trace display and analysis windows are slow.

   You can overcome this problem by loading the program code to the **TRACE32 virtual memory** and by specifying **Trace.ACCESS AutoVM**.

```
; load the program code additional to the TRACE32 virtual memory
; whenever you load it to the target memory
Data.LOAD.Elf diabc.x /PlusVM

; advise TRACE32 to read the target code from the virtual memory

; if no code is loaded to the virtual memory for a program address
; TRACE32 will read the code by using the best practice procedure
Trace.ACCESS AutoVM
```

3. Trace information should be inspected, but there is no source code available.

   You can overcome this problem by specifying **Trace.ACCESS Denied** to advise TRACE32 not to merge source code information. The **Trace.List** window will list the available program addresses and mark all cycles as unknown.
Recommended access paths:

**auto**

TRACE32 uses its own best practice procedure to read the program code. *(Note: For the ARM architecture this mode is usually *not* using the DualPort access.)*

**AutoVM**

If the program code for a program address is available via the TRACE32 virtual memory it is read from there. Otherwise the best practice procedure is used.

**VM**

The program code is always read from the TRACE32 virtual memory.

**Denied**

No source code information is read.

Rarely used access paths:

**OVS**

Code overlays are handled by the best practice procedure. If the best practice procedure does not deliver correct results, you can advise TRACE32 to read the program code by using the overlay table.

**CPU**

Advise TRACE32 to read the code via the CPU/core.

**DualPort**

Advise TRACE32 to read the code via the run-time access to the target memory.

See also

- <trace>.state
- Trace

**<trace>.ADDRESS**

Software trace address

Format: **<trace>.ADDRESS [<address>]**

**LOGGER.ADDRESS** defines the address of the logger trace buffer control block in target memory.

See also

- <trace>.state
- Trace
The trace memory and if available the trigger unit are prepared for recording and triggering. It is not possible to read the trace contents while the trace is in **Arm** state.

For most trace methods it is possible to **AutoArm** (<trace>.AutoArm) the trace. That means:

- Recording and triggering are prepared whenever the program execution is started.
- Recording and triggering are stopped whenever the program execution is stopped.

This is the default setting.

It is also possible to manually switch off the trace (<trace>.OFF) to read the trace contents and arm it again afterwards.

**See also**

- <trace>.AutoArm
- <trace>.AutoStart
- <trace>.Init
- <trace>.state
- Trace
- CLProbe
- Analyzer.STATE()
- SNOOPer.STATE()

▲ 'Emulator Functions' in 'FIRE User's Guide'
▲ 'Release Information' in 'Release History'
<trace>.AutoArm

Arm automatically

Format: `<trace>.AutoArm [ON | OFF]

Default: `<trace>.AutoArm ON.

- Recording and if available triggering is prepared whenever the program execution is started.
- Recording and if available triggering is stopped whenever the program execution is stopped.

See also
- `<trace>.Arm
- `<trace>.state
- Analyzer.STATE()
- SNOOPer.STATE()
- Trace
- CIProbe
- 'Emulator Functions' in 'FIRE User's Guide'

<trace>.AutoFocus

Calibrate AUTOFOCUS preprocessor

Format: `<trace>.AutoFocus [<address_range>] [/<option>]

<option>: Accumulate
KEEP
ALTERNATE
NoTHreshold

The command `Trace.AutoFocus` configures an AutoFocus preprocessor for an error-free sampling on a high-speed trace port.

For preprocessors without AUTOFOCUS technology, but adjustable reference voltage, this command will modify the reference voltage (see `Trace.THreshold`) and try to find a value were the trace capture is free of errors. This might take anywhere from a few up to 30 s.

If available the test pattern generator of the trace port is used to generate the trace data for the auto-configuration. Otherwise a test program is loaded and started by TRACE32.

If a test program is used, TRACE32 attempts to load the test program to the memory addressed by the PC or the stack pointer. It is also possible to define an `<address_range>` for the test program.

```
Trace.AutoFocus ; start the auto-configuration
Trace.AutoFocus 0x24000000++0xfff ; start auto-configuration, load ; the test program to address ; 0x24000000
```
If TRACE32 is unable to load the test program the following error message is displayed:
“Don’t know where to execute the test code”.

By default the original RAM contents is restored after the auto-configuration and the trace contents is deleted.

**Accumulate**
If the application program varies the CPU clock frequency, this affects also the trace port and the auto-configuration. In such a case it is recommended to overlay the auto-configurations for all relevant CPU clock frequencies by using the option /Accumulate.

**KEEP**
When the auto-configuration is completed, the test pattern generator/test program is started once again to test the correctness of the trace recording. After this test the trace is cleared and an eventually loaded test program is removed from the target RAM. With the option /KEEP the test trace is not cleared and can be viewed with the Trace.List command. If a test program was loaded by TRACE32 it also remains in the target RAM.

**ALTERNATE**
If the trace port provides a test pattern generator, it is always used for the auto-configuration. The option /ALTERNATE forces TRACE32 to use its own test program. This is recommended e.g. if a CoreSight test pattern generator is not stimulating the TRACECLT signal.

**NoTHreshold**
Do not calibrate the Trace.THreshold reference voltage.

The option /Accumulate allows to overlay several auto-configurations. It is recommended to proceed as follows:

1. Execute the command Trace.AutoFocus at the highest CPU clock frequency.
2. Reduce the CPU clock frequency and execute the command Trace.AutoFocus /Accumulate.
   If a preprocessor with AUTOFOCUS technology is used, the clock and data delays are adjusted, while the termination voltage, the clock reference voltage and the data reference voltage remain unchanged.
3. Repeat step 2 for all relevant frequencies.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace.AutoFocus</td>
<td>Execute the command for the highest CPU clock</td>
</tr>
<tr>
<td>Trace.AutoFocus /Accumulate</td>
<td>Re-execute the command for the next lower CPU clock</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>Trace.AutoFocus /Accumulate</td>
<td>Re-execute the command for the lowest relevant CPU clock</td>
</tr>
</tbody>
</table>
A failure in the Trace.AutoFocus command results in a stop of a PRACTICE script. The following workaround can be used to avoid this behavior:

```plaintext
; go to the label error.autofocus: if an error occurred in the script
ON ERROR GOTO error.autofocus
Trace.AutoFocus
; go to the label end: if an error occurred in the script
ON ERROR GOTO end
...
end:
ENDDO
error.autofocus:
PRINT %ERROR "Trace.AutoFocus failed. Script is aborted"
ENDDO
```

**NOTE:** The NEXUS AutoFocus adapter does not support this feature.
The Trace.AutoFocus command causes the preprocessor with AUTOFOCUS technology to configure itself. The auto-configuration searches for the best set of reference voltages and assures optimal sampling of the information broadcast by the trace port. The higher the trace port data rate, the more effort is put in the hardware configuration. For trace port data rates higher 200 Mbit/s the command may need up to 7 s for completion.

In contrast to Trace.TestFocus, the command Trace.AutoFocus does both the hardware configuration as well as a trace port test.

For preprocessors with AUTOFOCUS technology the hardware auto-configuration includes:

- Automatic setup of proper termination voltage to assure signal integrity.
- Automatic setup of clock reference voltage resulting in a stable clock with 50/50 duty cycle.
- Automatic setup of data reference voltage resulting in broad data eyes.
- Automatic setup of clock and data delays resulting in optimal sampling for each data channel.

The complete auto-configuration executes the following steps:

1. If available the trace port's test pattern generator is started. Otherwise a test program (maximum size 4 kB) is loaded by TRACE32 to the target RAM and started.

2. A hardware auto-configuration as described above is executed. When the optimal hardware configuration is found the test pattern generator/test program is stopped and the trace data is discarded. After executing the hardware auto-configuration the data eyes and optimal sampling points are known to the TRACE32 software and can be viewed by the user with the Trace.ShowFocus command.

3. The test pattern generator/test program is started once again and the program and data flow is recorded to the trace buffer to allow TRACE32 to verify the correctness of the trace recording.

If the self calibration was successful, the following message is displayed in the message line (f=<trace_port_frequency>):

<table>
<thead>
<tr>
<th>Analyzer data capture o.k. (f=156.0MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sample</td>
</tr>
<tr>
<td>1000000</td>
</tr>
</tbody>
</table>

**NOTE:** The trace port frequency does not necessarily equal the CPU clock frequency. E. g. for the ARM-ETM:

- An ETMv1 or ETMv2 operating at HalfRate results in an ETM clock frequency that is half the CPU clock frequency
- An ETMv3 operating with PortMode 1/2 results in an ETM frequency that is a quarter of the CPU clock frequency.

The result of the Trace.AutoFocus command can be displayed with the Trace.ShowFocus command. If the user wants to verify that the current hardware configuration is complying with the current requirements (e.g. after a frequency change) without wanting to change this configuration, the Trace.TestFocus command can be used.
If the auto-configuration fails and you need technical support, please use the *AutoFocus Diagnosis* to prepare all relevant information for the support person.

See also

- `<trace>.ShowFocus`
- `<trace>.ShowFocusEye`
- `<trace>.TestFocus`
- `Trace`
- `<trace>.TestFocusClockEye`
- `AUTOFOCUS.OK()`
- `AUTOFOCUS.FREQUENCY()`
- 'Introduction’ in 'AutoFocus User’s Guide’
- 'Introduction’ in 'PowerTrace Serial User’s Guide’
- 'Installation’ in 'ARM-ETM Trace’
- 'Release Information’ in 'Release History’
The `<trace>.Init` command will be executed automatically, when the user program is started (or stepped through). This causes that

- Trace memory contents is erased and previous records are no longer visible.
- The trigger unit is set to its initial state.
- All used counters are initialized and all used flags are set to OFF.

In combination with the command `<trace>.SelfArm` the trace is able to generate continuous recording and display like a trace snapshot.

### See also
- CIProbe
- `<trace>.Init`
- `<trace>.state`
- Trace
- Analyzer.STATE()
- SNOOPer.STATE()
- ‘Emulator Functions’ in ‘FIRE User’s Guide’
- ‘Release Information’ in ‘Release History’

---

The `<trace>.AutoStart` command will execute the `<trace>.Init` automatically, when a specified break event (as defined in the emulator trigger unit) is encountered and a user program is re-started with command GO or single step.

### See also
- `<trace>.Arm`
- `<trace>.state`
- Trace
- Analyzer.STATE()
- ‘Emulator Functions’ in ‘FIRE User’s Guide’
If the CPU is running and the trace is in OFF state, the command `<trace>.Arm` will be executed. If the `<trace>.AutoInit` function is activated, the command `<trace>.Init` will be executed prior to the ARM command, otherwise the trigger unit will be reset. After the trace goes to the OFF or break state again (by trace trigger break or full trace in stack mode) all windows are updated and the measurement is continued. The measurement will be stopped, if a display window of the trace is entered.

```
'Analyzer.Program'
(
    ADDR AlphaB flags ; access to first element of the array
    Sample.enable if AlphaB ; sample access to address flags
    BREAK if AlphaB ; break analyzer for display
)
Analyzer.Mode Fifo ; standard mode
Analyzer.AutoTEST ON ; select automatic test
Analyzer.AutoInit OFF ; endless recording
Analyzer.List ; display the accesses
Go ; start the user program
Port.OFF ; switch mode to timing analyzer
Port.Timing ; create a display window
Port.TSelect ALways
Port.TMode High
Port.TDelay 200.us ; make 200.us long sampling
Port.AutoTEST ON ; start continuous measurement
```
The memory accesses to flags are displayed dynamically in the list window.

```plaintext
; Mark all function entries with AlphaB and function exits with BetaB
Break.SetFunc

; Program the analyzer
Analyzer.ReProgram ~/demo/analyzer/perf

; Switch the trace buffer to stack mode
Analyzer.Mode Stack

; Limit the trace size
Analyzer.SIZE 1000.

; Select automatic test
Analyzer.AutoTEST ON

; Single measurements
Analyzer.AutoInit ON

; Display a statistic window
Analyzer.STATistic.Func

Go
```

A detailed dynamic performance analysis is displayed by using the trace to record the function entries and exits.

**See also**

- `<trace>.state`
- `Trace`
- 'Emulator Functions' in 'FIRE User's Guide'
Set a bookmark in trace listing

Format:  

<trace>.BookMark <string> [<time> | <value>] [/FILE]

Sets a trace bookmark in the trace listing. A small yellow rectangle next to the record number indicates a trace bookmark.

The BookMark.List window provides an overview of all trace bookmarks. Clicking a yellow trace bookmark takes you to the location of that trace bookmark. Additionally, you can use the Goto button in a <trace>.List window to jump to a bookmarked trace record.

Example 1:

;create a trace bookmark named "BM2" for the trace record -120000.
Trace.BookMark "BM2" -120000.
Trace.List DEFault /Track ;list the trace contents
BookMark.List ;display all bookmarks in a list

<string>  User-defined bookmark name. An auto-incremented bookmark name can be generated via the TRACE32 command line if a comma is entered instead of a user-defined name.

<time>    Creates a trace bookmark at a time stamp that is based on zero time. See example 2 below.

[value]   Creates a trace bookmark at the specified record number, e.g. -120000.
Example 2 shows how to create a bookmark 0.300ms after the zero-time reference point. The optional steps are included in this example to let you view on screen what happens behind the scenes.

```plaintext
;optional step: In the trace listing, the TIME.ZERO column is displayed as the first column, followed by the DEFAULT columns
Trace.List TIME.ZERO DEFAULT /Track

;optional step: go to the first trace record, i.e. the record with the lowest record number
Trace.GOTO Trace.FIRST()

;set the zero-time reference point to the first trace record
ZERO.offset Trace.RECORD.TIME(Trace.FIRST())

Trace.BookMark "BM3" 0.300ms ;create a bookmark 0.300ms after the zero-time reference point
Trace.GOTO "BM3" ;optional step: go to the new bookmark
BookMark.List ;optional step: display all bookmarks
```

See also

- `<trace>.List`
- `<trace>.state`
- `Trace`
- `<trace>.BookMarkToggle`
- `<trace>.BookMarkCreate`
- `<trace>.BookMarkEditRemark`

▲ 'BookMark' in 'General Commands Reference Guide B'
Toggles a single trace bookmark

Switches a single trace bookmark on or off. TRACE32 executes the same command when you right-click in a `<trace>.List` window, and then choose **Toggle Bookmark**. The resulting bookmark names are auto-incremented 1, 2, 3, etc. User-defined bookmark names can be created via the command line. A small **yellow** rectangle next to the record number indicates a trace bookmark.

**Example:**

```
Trace.List TIme.Zero DEFault /Track ;list the trace contents

;let's toggle two trace bookmarks with user-defined names
Trace.BookMarkToggle "TStart" -Trace.Records() ;bookmark at first record
Trace.BookMarkToggle "TEnd" -1.              ;bookmark at last record

BookMark.List        ;display all bookmarks in a list
```

**See also**

- `<trace>.BookMark`
- `BookMark`
- `BookMark.List`
- `BookMark.Toggle`

▲ 'BookMark' in 'General Commands Reference Guide B'

©1989-2019 Lauterbach GmbH
<trace>.Break

Format: `<trace>.Break`

The trace is stopped and the trace storage is ready for read-out.
The `<trace>.Chart` command group allows to display the analyzed trace information graphically. Examples are:

- Function run-time (<Trace.Chart.Func>)
- Time chart (<Trace.Chart.sYmbol>)
- Task run-time (<Trace.Chart.TASK>)
- Variable contents (<Trace.Chart.VarState>)

### Parameters

This section describes the optional `<trace_area>` parameters of the `<trace>.Chart` command group.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;record_range&gt;</code></td>
<td>Defines which part of the trace buffer is displayed. See example.</td>
</tr>
<tr>
<td><code>&lt;record&gt;</code></td>
<td>Defines which trace record is centered on the x-axis when the window is opened. Records at the beginning or end of the x-axis are not centered. See example.</td>
</tr>
<tr>
<td><code>&lt;time&gt;</code></td>
<td>Defines which timestamp is centered on the x-axis when the window is opened. Timestamps at the beginning or end of the x-axis are not centered. <strong>NOTE:</strong> Only zero-time timestamps can be used as <code>&lt;time&gt;</code> parameters. You can display the zero-time timestamps in a Trace window by adding the <code>TimeZero</code> option to <code>Trace.Chart.*</code> or by adding the <code>Time.Zero</code> column to <code>Trace.List</code>. See examples.</td>
</tr>
<tr>
<td><code>&lt;timerange&gt;</code></td>
<td>Defines which timestamp is displayed on left of the x-axis when the window is opened. <strong>NOTE:</strong> Only zero-time timestamps can be used as <code>&lt;timerange&gt;</code> parameters. You can display the zero-time timestamps in a Trace window by adding the <code>TimeZero</code> option to <code>Trace.Chart.*</code> or by adding the <code>Time.Zero</code> column to <code>Trace.List</code>. See example.</td>
</tr>
</tbody>
</table>
### General Options

This section describes the options of the `<trace>.Chart` command group.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;timescale&gt;</td>
<td>Rule of thumb: The smaller the <code>&lt;timescale&gt;</code> value, the higher the resolution and the wider the chart in the data area of a <code>&lt;trace&gt;.Chart.*</code> window.</td>
</tr>
<tr>
<td></td>
<td>The <code>&lt;timescale&gt;</code> parameter defines the display scaling as time per character. tbd.</td>
</tr>
<tr>
<td></td>
<td>It is useful for printing operations and allows to print out any timing chart in a fixed scale on multiple pages. tbd.</td>
</tr>
<tr>
<td></td>
<td>- See example.</td>
</tr>
<tr>
<td></td>
<td>- For the units of measurement, see <strong>“Parameter Types”</strong> (ide_user.pdf).</td>
</tr>
<tr>
<td>&lt;trace_bookmark&gt;</td>
<td>Defines which bookmark position is centered on the x-axis when the window is opened. Bookmark positions at the beginning or end of the x-axis are not centered.</td>
</tr>
<tr>
<td></td>
<td><strong>NOTE</strong>: You can only use the names of trace bookmarks, which are created with the <code>&lt;trace&gt;.BookMark</code> command.</td>
</tr>
<tr>
<td></td>
<td>See example.</td>
</tr>
<tr>
<td>Track</td>
<td>The cursor in the <code>&lt;trace&gt;.Chart</code> window follows the cursor movement in other trace windows. Default is a time tracking. If no time information is available tracking to record number is performed. The zoom factor of the <code>&lt;trace&gt;.Chart</code> window is retained, even if the trace content changes.</td>
</tr>
<tr>
<td>ZoomTrack</td>
<td>Same as option Track. If the tracking in performed with another <code>&lt;trace&gt;.Chart</code> window the same zoom factor is used.</td>
</tr>
<tr>
<td>Sort</td>
<td>Specify sorting criterion for analyzed items. For almost all commands the analyzed items are displayed in the order they are recorded by default.</td>
</tr>
<tr>
<td></td>
<td>Details on the sorting criterion can be found at the description of the command <strong>Trace.STATistic.Sort</strong>.</td>
</tr>
<tr>
<td>INCremental</td>
<td>Intermediate results are displayed while TRACE32 PowerView is processing the trace analysis (default).</td>
</tr>
<tr>
<td>FULL</td>
<td>TRACE32 PowerView displays the result when the processing is done.</td>
</tr>
<tr>
<td><strong>FILE</strong></td>
<td>Use the trace contents loaded with the command <code>&lt;trace&gt;.FILE</code>.</td>
</tr>
<tr>
<td>----------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TASK</strong></td>
<td>Operating system task in OS-aware debugging and tracing. For information about the parameters, see “What to know about Task Magic Numbers, Task IDs and Task Names”, page 24.</td>
</tr>
<tr>
<td><code>&lt;task_magic&gt;</code></td>
<td><code>&lt;task_id&gt;</code></td>
</tr>
<tr>
<td><strong>CORE</strong></td>
<td>Time chart is only displayed for the specified core.</td>
</tr>
<tr>
<td><code>&lt;n&gt;</code></td>
<td></td>
</tr>
<tr>
<td><strong>SplitCORE</strong></td>
<td>Trace information is analyzed independently for each core. The time chart displays these individual results.</td>
</tr>
<tr>
<td><strong>MergeCORE</strong></td>
<td>Trace information is analyzed independently for each core. The time chart summarizes these results to a single result.</td>
</tr>
<tr>
<td><strong>JoinCORE</strong></td>
<td>Core information is ignored for the time chart.</td>
</tr>
<tr>
<td><strong>RecScale</strong></td>
<td>Display trace in fixed record raster. This is the default.</td>
</tr>
<tr>
<td><strong>TimeScale</strong></td>
<td>Display trace as true time display, time relative to the trigger point (respectively the last record in the trace).</td>
</tr>
<tr>
<td><strong>TimeZero</strong></td>
<td>Display trace as true time display, time relative to zero point. For more information about the zero point refer to <code>ZERO</code>.</td>
</tr>
<tr>
<td><strong>TimeREF</strong></td>
<td>Display trace as true time display, time relative to the reference point. For more information about the reference point refer to <code>&lt;trace&gt;.REF</code>.</td>
</tr>
<tr>
<td><strong>FlowTrace</strong></td>
<td>Trace works as a program flow Trace. This option is usually not required.</td>
</tr>
<tr>
<td><strong>BusTrace</strong></td>
<td>Trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td><strong>CTS</strong></td>
<td>Use Context Tracking System to fill trace gaps and then perform <code>&lt;trace&gt;.Chart</code>.</td>
</tr>
<tr>
<td><strong>NoInline</strong></td>
<td>Inline functions are treated as separate functions. The option <strong>NoInline</strong> can be used to discard inline functions.</td>
</tr>
</tbody>
</table>
Drag and Drop

A **Trace.Chart** window may contain a Drag & Drop area which is marked by a straight line.

Items of interest can be dragged to the appropriate position in the Drag & Drop area with the left mouse button.

The sort order of all items outside of the Drag & Drop area remains unchanged.

Items can be removed from the Drag & Drop area by dropping them to the item description area.
Example for `<trace_bookmark>`

```plaintext
Trace.BookMark "begin" 10.005s
Trace.BookMark "end" 10.010s
Trace.Chart.Symbol "begin" /Track /TimeZERO
Trace.GOTO "begin" ;highlight the bookmark in the chart
BookMark.List ;optional; display all bookmarks in a list
```

A To display the zero-time timestamps on the x-axis, the **TimeZero** option is used.

Example for `<record>`

```plaintext
;print distribution of data values written to flags[3], with the record
; -1950. centered on the x-axis of the window
Trace.Chart.Distrib -1950. Data.L /Filter Address Var.RANGE(flags[3]) \ /RecScale
```

A To display the record numbers on the x-axis, the **RecScale** option is used.

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

;print distribution of data values written to flags[3] for the
;record range (-2000.)--(-1000.)
Trace.Chart.Distrib (-2000.)--(-1000.) Data.L /Filter Address \Var.RANGE(flags[3]) /RecScale

A To display the record numbers on the x-axis, the RecScale option is used.

Examples for <time>

Example 1:

;open the chart window with the zero-time timestamp 10.009s and set the
;<timescale> resolution to 10us (optional)
Trace.Chart.TREE 10.009s 10us /Track /TimeZero
Trace.GOTO 10.009s ;highlight the timestamp in the chart

A To display the zero-time timestamps on the x-axis, the TimeZero option is used.
Example 2: This PRACTICE script shows how to open the Trace.ChartSymbol window with a <time> parameter that is located 50 microseconds after the 4th occurrence of the HLL symbol sieve.

```plaintext
;find the first occurrence of the HLL symbol 'sieve'
Trace.Find, Symbol sieve

RePeaT 3. ;find the next three occurrences of 'sieve'
Trace.Find

IF FOUND()==TRUE() ;if the 4th occurrences of 'sieve' has been found
{
;get the timestamp of the 4th occurrence and add an offset of 50.us
&time=TRACK.TIME()+50.us

;open the chart window with the calculated timestamp and set the
;<timescale> resolution to 9.5us
Trace.Chart.Symbol &time 9.5us /Address encode||sieve \ /Track /TimeZero

Trace.GOTO &time ;highlight the timestamp in the chart
}
```

Example for <timerange>

Trace.Chart.Symbol (10.005s)--(10.010s) 10.us /Track /TimeZero
Examples for `<timescale>`

**Example 1:** Using **WinPrint**, you can print the window content without actually opening the window.

```
PRinTer.select WIN ; select the printer to which you want to print

; print distribution of data values written to flags[3] for the
; record range (-2000.)--(-1000.), use resolution 10.us per pixel
Address Var.RANGE{flags[3]}  
```

**Example 2:** Using the **WinPOS** command, you can assign a name to a window. Then you open the window and print it with **WinPRT <name>**. This example illustrates three different `<timescale>` resolutions.

```
; the following resolutions are used:
; [A] 5.us per pixel, [B] 1.us per pixel, [C] 0.5us per pixel
PRinTer.select WIN ; select the printer to which you want to print

WinPOS , , , , , W0
Trace.Chart.DistriB (-2000.)--(-1000.) 5.us Data.L /Filter Address \ 
Var.RANGE{flags[3]}

WinPOS , , , , , W1
Trace.Chart.DistriB (-2000.)--(-1000.) 1.us Data.L /Filter Address \ 
Var.RANGE{flags[3]}

WinPOS , , , , , W2
Trace.Chart.DistriB (-2000.)--(-1000.) 0.5us Data.L /Filter Address \ 
Var.RANGE{flags[3]}

WinPRT W0 ; print the window named W0
```
<trace>.Chart.Address  

Time between program events as a chart

Format:  

<trace>.Chart.Address <address1> [<address1> ...]

Displays the time interval between up to 8 program events as a chart. The <trace>.Chart.Address command is the counterpart of the <trace>.STATistic.Address command.

See also

■ <trace>.Chart_addr
■ <trace>.Chart.Func
■ <trace>.Chart.Line
■ <trace>.Chart.TASK
■ <trace>.Chart.TASKState
■ <trace>.state
■ <trace>.Chart.Address
■ <trace>.Chart.Func
■ <trace>.Chart.Line
■ <trace>.Chart.sYmbol
■ <trace>.Chart.TASKFunc
■ <trace>.Chart.TASKKernel
■ <trace>.Chart.TASKState
■ <trace>.Chart.TASKVSINTR
■ <trace>.PROfileChart
■ <trace>.STATistic

▲ 'How to use the PROTOanalyzer' in 'DigRF Protocol Analyzer'
▲ 'How to use the PROTOanalyzer' in 'FlexRay Protocol Analyzer'
▲ 'How to use the PROTOanalyzer' in 'LIN Bus Protocol Analyzer'
▲ 'Release Information' in 'Release History'

See also

■ <trace>.Chart
■ <trace>.Chart.DistriB
■ <trace>.Chart.GROUP
■ <trace>.Chart.sYmbol
■ <trace>.Chart.TASKSRV
■ <trace>.Chart.VarState
■ Trace
■ <trace>.Chart.DistriB
■ <trace>.Chart.GROUP
■ <trace>.Chart.Nesting
■ <trace>.Chart.TASK
■ <trace>.Chart.TASKINTR
■ <trace>.Chart.TASKSRV
■ <trace>.Chart.TASKVSINTERRUPT
■ <trace>.Chart.VarState
■ <trace>.PROfileSTATistic

See also

■ <trace>.Chart.DatasYmbol
■ <trace>.Chart.Func
■ <trace>.Chart.Line
■ <trace>.Chart.TASK
■ <trace>.Chart.TASKState
■ <trace>.state
■ <trace>.Chart.Address
■ <trace>.Chart.DistriB
■ <trace>.Chart.Func
■ <trace>.Chart.GROUP
■ <trace>.Chart.Line
■ <trace>.Chart.sYmbol
■ <trace>.Chart.TASK
■ <trace>.Chart.TASKSRV
■ <trace>.Chart.TASKState
■ <trace>.Chart.TASKFunc
■ <trace>.Chart.TASKINTR
■ <trace>.Chart.TASKKernel
■ <trace>.Chart.TASKSRV
■ <trace>.Chart.TASKVSINTERRUPT
■ <trace>.Chart.TASKVSINTR
■ <trace>.Chart.VarState
■ <trace>.PROfileChart
■ <trace>.PROfileSTATistic

See also

■ <trace>.Chart.DatasYmbol
■ <trace>.Chart.Func
■ <trace>.Chart.Line
■ <trace>.Chart.TASK
■ <trace>.Chart.TASKState
■ <trace>.state
■ <trace>.Chart.Address
■ <trace>.Chart.DistriB
The command **Trace.Chart.DatasYmbol** analyzes the contents of a pointer graphically.

If a full program and data trace is analyzed, the following command is recommended:

```plaintext
; analyze the contents of the pointer vpchar graphically
Trace.Chart.DatasYmbol /Filter Address vpchar
```
A more effective usage of the trace memory is possible, if only write accesses to the pointer are recorded to the trace.

```
; set a filter to record only write cycles to the pointer vpchar to the
; trace
Var.Break.Set vpchar /Write /TraceEnable

...

; analyze the contents of the pointer
Trace.Chart.DatasYmbol

; analyze the contents of the pointer, sort the result by symbol names
Trace.Chart.DatasYmbol /Sort sYmbol
```

See also

- `<trace>.Chart`
- 'Release Information' in 'Release History'
Distribution display graphically


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

<option>: `FILE
FlowTrace | BusTrace | CTS
Track
RecScale | TimeScale | TimeZero | TimeREF
Filter <item>
<general_options>`

The distribution of any trace data is displayed if `<item>` is specified. Without argument the distribution of the addresses is displayed symbolically.

<trace_area> For parameter descriptions and examples, see Parameters.

<option> For the general options for all `<trace>.Chart` commands, refer to `<trace>.Chart`.

Filter `<item>` If no selective trace on the requested `<item>` is performed, a filter can be used.

If no selective tracing is done, use the option /Filter to filter out the `<item>` of interest.

; Display distribution of data value for flags[3]
Trace.Chart.DistriB Data.L /Filter Address V.RANGE(flags[3])

; Print distribution of data value written for flags[3] for the
; record range (-2000.)--(-1000.)
WinPrint.Trace.Chart.DistriB (-2000.)--(-1000.) Data.L /Filter Address
V.RANGE(flags[3])
Example for TRACE32-ICE or TRACE32-FIRE:

Perform a selective trace on the data of interest:

```
Analyzer.ReProgram
(
  ADDR AlphaBreak V.RANGE(GlobalStateVar)
  Sample.enable IF AlphaBreak
)
Go
...
Break
Trace.Chart.DistriB Data.W
```

; selective trace
; program the analyzer to
; sample all accesses to
; the variable
; GlobalStateVar

; measurement
; display the time spent
; in different states

Example for all trace methods:

```
Trace.STATistic.Sort Address
Trace.Chart.DistriB
```

; sort chart by addresses
; display the time spent at different
; addresses

See also
- |trace>.Chart
- 'Filter and Trigger - Single-Core and AMP' in 'AURIX Trace Training'
- 'Filter and Trigger - SMP Systems' in 'AURIX Trace Training'
The time spent in different functions is displayed graphically. The measurement is the same as for the command `<trace>.STATistic.Func`.

```
E:\Trace.CHART.func

 range   -2.000ms -1.800ms -1.600ms
(root)
\\mcc\mcc\main+0x4        [---------------------]
\\mcc\mcc\func2+0x4        [---------------------]
\\mcc\mcc\func1g+0x4       [---------------------]
\\mcc\mcc\func4+0x4        [---------------------]
\\mcc\mcc\func6+0x4        [---------------------]
\\mcc\mcc\func7+0x4        [---------------------]
\\mcc\mcc\func8+0x4        [---------------------]
\\mcc\mcc\func9+0x4        [---------------------]
\\mcc\mcc\func10+0x4       [---------------------]
\\mcc\mcc\func13+0x4       [---------------------]
\\mcc\mcc\func13a+0x4      [---------------------]
\\mcc\mcc\sieve+0x4        [---------------------]
```

See also
- `<trace>.Chart`
- `CTS.CACHE.Chart`

For parameter descriptions and examples, see Parameters.

For the general options for all `<trace>.Chart` commands, refer to `<trace>.Chart`.
Displays a GROUP time chart (flat statistic).

For parameter descriptions and examples, see Parameters.

For the general options for all <trace>.Chart commands, refer to <trace>.Chart.

Example:

GROUP.Create "INPUT" \jquant2 \jquant1 \jidctred \jdinput /AQUA
GROUP.Create "JPEG" \jdapimin \jdcolor \jddctmgr \jdcoefct /NAVY
Go
Break
Trace.Chart.GROUP

See also
■ <trace>.Chart
■ GROUP.Create
▲ 'Release Information' in 'Release History'
**<trace>.Chart.Line**

Graphical HLL lines analysis

**Format:**

```
<trace>.Chart.Line [<trace_area>] [/<option>]
```

**<trace_area>:**

```
<trace_bookmark> | <record> | <record_range> | <time> | <time_range> 
```

**<option>:**

```
FILE
FlowTrace | BusTrace | CTS
Track
RecScale | TimeScale | TimeZero | TimeREF
Filter <item>
```

The time spent in different HLL lines is analyzed graphically.

- **<trace_area>**
  For parameter descriptions and examples, see **Parameters**.

- **<option>**
  For the general options for all <trace>.Chart commands, refer to <trace>.Chart.

---

**Example for TRACE32-ICD and PowerTrace:***

If no selective trace is possible use the option /Filter to filter out the module or function of interest.

```
Go
Break
Trace.Chart.Line /Filter V.RANGE(sieve)
```
Example for TRACE32-ICE and TRACE32-FIRE:

If only a specific module or function should be analyzed, perform a selective trace on the module or function:

```c
Analyzer.ReProgram
(
    ADDR AlphaBreak V.RANGE(module1)
    Sample.Enable IF AlphaBreak
)
Go
Break
Trace.Chart.Line
```

If only a specific module or function should be analyzed, perform a selective trace on the module or function:

See also
- `<trace>.Chart`
- 'Release Information' in 'Release History'

### `<trace>.Chart.Nesting`

Show function nesting at cursor position

**Format:**

```
<trace>.Chart.Nesting [<trace_area>] [/<option>]
```

- `<trace_area>`: `<trace_bookmark> | <record> | <record_range> | <time> | <time_range>`
  
- `<option>`: `FILE | FlowTrace | BusTrace | RecScale | TimeScale | TimeZero | TimeREF | INCremental | FULL`
  
  `Sort <item>`
  
  `Track`
  
  `ZoomTrack`
  
  `TASK`

Shows the function call stack as a time chart.

- `<trace_area>`
  
  For parameter descriptions and examples, see Parameters.

- `<option>`
  
  For the general options for all `<trace>.Chart` commands, refer to `<trace>.Chart`.

See also
- `<trace>.Chart`

©1989-2019 Lauterbach GmbH
The distribution of program execution time at different symbols is displayed as a time chart. This can be used to get a quick overview about the functions sampled in the trace buffer.

<trace_area> For parameter descriptions and examples, see Parameters.

<option> For the general options for all <trace>.Chart commands, refer to <trace>.Chart.
Example for TRACE32-ICD and TRACE32-PowerTrace:

Go
Break
Trace.STATistic.Sort sYmbol ; sort the result alphabetically

; draw time chart for specified functions, assign time for all other
; functions to (other)
Trace.Chart.sYmbol /Address func2||func10||sfpDoubleNormalize

; draw time chart for specified functions (address range), assign time
; for all other functions to (other)
Trace.Chart.sYmbol /Address func2--func10

The GROUP command provides more features to structure your time chart.
; filter specified functions out of the address stream
; and draw time chart for filtered trace information
Trace.Chart.Symbol /Filter Address main|func2|func10|func26

Recording (filtered functions are displayed in black)

Analysis result
Example for TRACE32-ICE and TRACE32-FIRE:

If only a specific module or function should be analyzed, perform a selective trace on the module or function:

```plaintext
Analyzer.ReProgram
    ADDR AlphaBreak Y.SECRANGE(\diab555\.text)
    Sample.Enable IF AlphaBreak
)
Go
Break
Trace.STATistic.Sort Ratio
Trace.Chart.sYmbol
```

See also

- `<trace>.Chart`
- `<trace>.STATistic.sYmbol`
- `<trace>.Chart`
- `CTS.CACHE.Chart`
- `CTS.state`
- 'Release Information' in 'Release History'
- 'Function Run-Times Analysis' in 'ARM-ETM Training'
- 'Flat Function-Runtime Analysis - Single-Core and AMP' in 'AURIX Trace Training'
- 'Function Run-Times Analysis - Single' in 'Nexus Training'
Displays the time spent in different tasks. The measurement is done similar to the command `<trace>.STATistic.TASK`.

For parameter descriptions and examples, see Parameters.

For the general options for all `<trace>.Chart` commands, refer to `<trace>.Chart`.

See also
- `<trace>.Chart`
- 'Release Information' in 'Release History'
- 'OS-Aware Tracing - Single-Core and AMP' in 'AURIX Trace Training'
- 'OS-Aware Tracing - SMP Systems' in 'AURIX Trace Training'
- 'OS-Aware Tracing' in 'Intel® Processor Trace Training'
- 'OS-Aware Tracing - Single Core' in 'Nexus Training'
**<trace>.Chart.TASKFunc**  
Task related function run-time analysis (legacy)

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;trace&gt;.Chart.TASKFunc [&lt;record_range&gt;] [&lt;scale&gt;] [/&lt;option&gt;] (legacy)</th>
</tr>
</thead>
</table>
| <option>: | FILE  
FlowTrace | BusTrace | CTS  
Track  
RecScale | TimeScale | TimeZero | TimeREF |

For details, refer to `<trace>.Chart.Func`.

See also
- `<trace>.Chart`

**<trace>.Chart.TASKINTR**  
Display ISR2 time chart (ORTI)

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;trace&gt;.Chart.TASKINTR [&lt;trace_area&gt;] [/&lt;option&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;trace_area&gt;:</td>
<td>&lt;trace_bookmark&gt;</td>
</tr>
<tr>
<td></td>
<td>[&lt;time_scale&gt;]</td>
</tr>
</tbody>
</table>

Displays a ISR2 time chart.

- `<trace_area>` For parameter descriptions and examples, see **Parameters**.
- `<option>` For the general options for all `<trace>.Chart` commands, refer to `<trace>.Chart`.

See also
- `<trace>.Chart`
- 'OS-Aware Tracing - Single-Core and AMP’ in ‘AURIX Trace Training’
- 'OS-Aware Tracing - SMP Systems’ in ‘AURIX Trace Training’
Task run-time chart with kernel markers (flat)

Format:  
<trace>.Chart.TASKKernel [<trace_area>] [/<option>]

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

Time chart for results of Trace.STATistic.TASKKernel.

<trace_area>  
For parameter descriptions and examples, see Parameters.

<option>  
For the general options for all <trace>.Chart commands, refer to <trace>.Chart.

See also  
■ <trace>.Chart
Format: `<trace>.Chart.TASKSRV` `[<trace_area>] [/<option>]`

- `<trace_area>`: `<trace_bookmark> | <record> | <record_range> | <time> | <time_range> [ <time_scale> ]`
- `<option>`: `FILE` | `FlowTrace` | `BusTrace` | `CTS` | `Track` | `RecScale` | `TimeScale` | `TimeZero` | `TimeREF`

The time spent in OS service routines and different tasks is displayed. Service routines that are used by multiple tasks are displayed for each task. The measurement is similar to the command `<trace>.STATistic.TASKSRV`.

- `<trace_area>`: For parameter descriptions and examples, see Parameters.
- `<option>`: For the general options for all `<trace>.Chart` commands, refer to `<trace>.Chart`.

See also
- `<trace>.Chart`
- ‘OS-Aware Tracing - Single-Core and AMP’ in ‘AURIX Trace Training’
- ‘OS-Aware Tracing - SMP Systems’ in ‘AURIX Trace Training’
<trace>.Chart.TASKState

Task state analysis

Format:  <trace>.Chart.TASKState [<trace_area>] [ /<option>]

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

<option>:  FILE
FlowTrace | BusTrace | CTS
Track
RecScale | TimeScale | TimeZero | TimeREF

The time different task spent in specific states is displayed. The measurement is similar to the command <trace>.STATistic.TASKState.

<trace_area>  For parameter descriptions and examples, see Parameters.

<option>  For the general options for all <trace>.Chart commands, refer to <trace>.Chart.

### Graphics

<table>
<thead>
<tr>
<th>State</th>
<th>Representation</th>
</tr>
</thead>
<tbody>
<tr>
<td>running</td>
<td>solid black bar</td>
</tr>
<tr>
<td>ready</td>
<td>medium blue bar</td>
</tr>
<tr>
<td>waiting</td>
<td>two thin red lines</td>
</tr>
<tr>
<td>suspended</td>
<td>thin grey line</td>
</tr>
<tr>
<td>undefined/unknown</td>
<td>no line</td>
</tr>
</tbody>
</table>

See also

- <trace>.Chart

©1989-2019 Lauterbach GmbH
<trace>.Chart.TASKVSINTERRUPT

Time chart

Shows a graphical representation of tasks that were interrupted by interrupt service routines.

Format:  <trace>.Chart.TASKVSINTERRUPT [trace_area] [\option]

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

<option>:  FILE
FlowTrace | BusTrace
Track
ZoomTrack
RecScale | TimeScale | TimeZero | TimeREF
INCremental | FULL
Filter <item>
Sort <item>

For parameter descriptions and examples, see Parameters.

For the general options for all <trace>.Chart commands, refer to <trace>.Chart.

See also
- <trace>.Chart
### <trace>.Chart.TASKVSINTR

**Time chart**

Displays a time-chart for task-related interrupt service routines.

#### Format:

<trace>.Chart.TASKVSINTR [<trace_area>] [/<options> ...]

#### <trace_area>:

<trace_bookmark> | <record> | <record_range> | <time> | <time_range> | [<time_scale>]

#### <option>:

- FILE
- FlowTrace | BusTrace
- Track
- ZoomTrack
- RecScale | TimeScale | TimeZero | TimeREF
- INCremental | FULL
- Sort <item>

For parameter descriptions and examples, see [Parameters](#).

For the general options for all `<trace>.Chart` commands, refer to [trace].Chart.

---

**See also**

- [trace].Chart
- ‘OS-Aware Tracing - Single-Core and AMP’ in ‘AURIX Trace Training’
- ‘OS-Aware Tracing - SMP Systems’ in ‘AURIX Trace Training’
<trace>.Chart.VarState

Variable activity chart

Format:  
<trace>.Chart.VarState [<trace_area>] [/<option>]

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

<option>:  
FILE
FlowTrace | BusTrace | CTS
Track
RecScale | TimeScale | TimeZero | TimeREF
Fill | FillFirst
DECODE <value> …
Filter <item>

Displays the contents of variables over the time. Each variable access must be sampled with one single
CPU cycle. If an address is not a variable it is displayed in form of a single marker. This can be used to track
program execution addresses.

<trace_area>  
For parameter descriptions and examples, see Parameters.

<option>  
For the general options for all <trace>.Chart commands, refer to <trace>.Chart.
Example 1 for TRACE32-ICE and TRACE32-FIRE:

Perform a selective trace on the data section.

Analyzer.ReProgram

(  
  ADDR AlphaBreak Y.SECRANGE(.data)  
  Sample.Enable IF AlphaBreak  
)

Go

Break

Trace.Chart.VarState

; selective trace

Go

Break

Trace.Chart.VarState

; sample data only

; collect data

; Display variable contents

; over the time

F::Trace.Chart.VarState

<table>
<thead>
<tr>
<th>range</th>
<th>900.000us</th>
<th>1.000ms</th>
<th>1.100ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>ast.word</td>
<td>0x7E563412</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vtripplearray[0][0][0]</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vtripplearray[0][0][1]</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vtripplearray[1][0][0]</td>
<td>140</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ast.count</td>
<td>-2007616494</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ast.right</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[0]</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[1]</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[2]</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ast.right</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[3]</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>flags[4]</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>flags[5]</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[6]</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[7]</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>flags[8]</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[9]</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>flags[10]</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[11]</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Example 2 for TRACE32-ICE and TRACE32-FIRE:

If no selective trace is possible use the option `/Filter` to filter out the variables.

Go

Break

Trace.Chart.VarState /Filter Y.SECRANGE(.bss)
**Fill**

Repeat the value instead of displaying the value only directly after the transition.

| FillFirst | Repeat the value without any space instead of displaying the value only directly after the transition. |
| FillFirst | Repeat the value without any space instead of displaying the value only directly after the transition. |
| DECODE <value> | Define a decoding for enumeration variables. |

**Trace.Chart.VarState /DECODE 2 4 7**

<table>
<thead>
<tr>
<th>range</th>
<th>900.000us</th>
<th>1.000ms</th>
<th>1.100ms</th>
</tr>
</thead>
<tbody>
<tr>
<td>ast.word</td>
<td>0x7E563412 0x7E563412 0x7E563412 0x7E563412</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vtripplearray[0][0][0]</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>vtripplearray[0][0][1]</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[7]</td>
<td>1 1 1 1 1 1 0 0 0 0 1 1 1 0 0 0 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[8]</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[9]</td>
<td>1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[10]</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>flags[11]</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**See also**

- `<trace>.Chart`
- 'Filter and Trigger - Single-Core and AMP’ in 'AURIX Trace Training’
- 'Filter and Trigger - SMP Systems’ in 'AURIX Trace Training’
- 'Release Information’ in 'Release History’
**FDX.CLEAR** clears the communication buffers of a FDX channel. All buffer contents are lost. Without arguments all FDX channels will be cleared.

**<trace>.CLOCK** Clock to calculate time out of cycle count information

Some trace protocols can generate cycle count information. TRACE32 can calculate time information out of the cycle count information if the appropriate clock frequency is specified with the **Trace.CLOCK** command.

For most trace protocols cycle count indicates the number of core clock cycles. That's why `<freq>` has to be the core clock frequency. Please be aware the specifying the core clock frequency only makes sense, if the frequency was constant while recording.

**Example for the ARM-ETM:**

```
ETM.TImeMode CycleAccurate
Trace.CLOCK 800.MHz
```

If the cores of an SMP run at different speeds, the frequency can be specified per core.

```
ETM.TImeMode CycleAccurate
Trace.CLOCK 800.MHz 600.MHz 1.GHz
```
<trace>.CLOSE

**Close FDX files**

**Format:**  
<trace>.CLOSE [<address>]

Closes the file related to the given FDX channel. Without arguments all files used by FDX are closed.

<trace>.ComPare

**Compare trace contents**

**Format:**  
<trace>.ComPare [<record_range>] [<record_number>] [{{<items>}}]  
[{{<options>}}]

**<options>:**  
Tolerance <count>  
FILE  
Back

Compares the trace contents. If the command <trace>.ComPare is used without arguments the previous compare is repeated.

<table>
<thead>
<tr>
<th><strong>&lt;item&gt;</strong></th>
<th>Only the given &lt;item&gt; … are compared.</th>
</tr>
</thead>
</table>
| <record_range>,  
<record_number> | If <record_range> and <record_number> are not used, a comparison of the complete trace is performed. |
| FILE | Compare the trace contents with the loaded file. See also Trace.FILE. |
| Back | Compare backwards. |
| Tolerance <count> | When external asynchronous data are traced, a jitter in the signal will result in different sampling data. In this case the precision of the compare function may be controlled by the option Tolerance. |

The compare function will set the pointers for the tracking option. All analyzer windows, which are in track mode, will follow these pointers.

For valid channel names refer to the:

- Processor Architecture Manuals
- Target Guides FIRE
- Target Guides ICE

©1989-2019 Lauterbach GmbH
Examples:

; compare the current trace contents from record (500.--1000.) with the
; current trace contents starting at record number 5000. with regards to
; the address
Trace.Compare (500.--1000.) 5000. Address

; load saved trace contents
Trace.FILE old_trace

; compare the current trace contents from record (500.--1000.) with the
; loaded trace contents starting at record number 300. with regards to
; the data on byte 0
Trace.Compare (500.--1000.) 300. Data.B0 /FILE

; load saved trace contents
Trace.FILE old_trace

; compare the complete current trace contents with the complete
; loaded trace contents with regards to the data on byte 0
Trace.Compare Data.B0 /FILE

; Repeat the previous compare
Trace.Compare

; load saved trace contents
Trace.FILE old_trace

; compare the complete current trace contents with the complete
; loaded trace contents with regards to the data and address
Trace.Compare Data Address /FILE
; compare against file TEST1 on line RXD
Port.FILE TEST1 ; load reference file
Port.ComPare RXD /Tolerance 3. /FILE ; compare line RXD
IF FOUND()
    PRINT "Difference found"
... ; print result if difference will be found
Port.ComPare ; search for next difference

See also
■ <trace>.state
■ Trace

<trace>.CSELect

Select signal for counter

Format: <trace>.CSELect <channel>

See also
■ Trace
PRACTICE script examples of custom trace demos can be found in the following *_demo.cmm files:

- ~/demo/customtrace/pipe_dll/dll_stp_demo.cmm
- ~/demo/customtrace/pipe_dll/dll_csstm_demo.cmm
- ~/demo/customtrace/pipe_dll/dll_itm_demo.cmm

For details about these files, refer to the readme.txt in the demo folder.

See also
- <trace>.CustomTrace.<label>.COMMAND
- <trace>.CustomTrace.<label>.UNLOAD
- <trace>.CustomTraceLoad
- <trace>.state

See also
- ‘Software Trace with the ITM’ in ‘CombiProbe for Cortex-M User’s Guide’
- ‘Software Trace with the ITM’ in ‘uTrace for Cortex-M User’s Guide’

**<trace>.CustomTrace.<label>.COMMAND**

Send command to specific DLL

Format 1: `<trace>.CustomTrace.<label>.COMMAND <command_line_args>`

Sends a command to a specific DLL that has been assigned a user-defined `<label>`.

See also
- <trace>.CustomTrace

**<trace>.CustomTrace.<label>.UNLOAD**

Unload a single DLL

Format 1: `<trace>.CustomTrace.<label>.UNLOAD`

Unloads a single DLL identified by `<label>`.

See also
- <trace>.CustomTrace
Format 1: `<trace>.CustomTraceLoad "<name>" <file>`

Format 2: `<trace>.CustomTraceLoad ""`

Format 1: TRACE32 supports a mechanism for passing trace data to a shared library or DLL allowing for custom trace handling. This command loads the shared object.

Format 2: When executed with an empty string, the command unloads all DLLs.

**NOTE:** Use the command `<trace>.CustomTrace.<label>.UNLOAD` to unload a single DLL.

`<name>` A user-defined name for the DLL or shared object. TRACE32 supports up to 8 loaded shared objects at any one time. The `<name>` is used to differentiate them.

`<file>` A shared library or DLL which is appropriate for your host Operating System. This DLL will receive trace data from TRACE32 and perform custom analysis on it.

See also
- `<trace>.CustomTrace`
- `<trace>.state`
- `Trace`
<trace>.DISable

Disable the trace

**Format:**

```
<trace>.DISable
```

Disables the trace.

**See also**

- CIProbe
- <trace>.state
- Trace

<trace>.DISableChannel

**Disable FDX communication**

**Format:**

```
<trace>.DISableChannel [<address>]
```

Disables an FDX communication channel. Without parameters all channels are disabled. Disabling keeps the buffer contents of FDX. Communication can be re-enabled with FDX.ENableChannel.

**See also**

- Trace
For background information and examples about how to use the `<trace>.DisConfig` command group, see:

- “PowerIntegrator Trace DisConfig Application Note” (powerintegrator_app_dc.pdf)

See also

- `<trace>.DisConfig.CYcle`
- `<trace>.DisConfig.RESet`
- `<trace>.state`
- `<trace>.DisConfig.FlowMode`
- `<trace>.DisConfig.view`
- Trace

’General Function’ in ‘PowerIntegrator Trace DisConfig Application Note’

---

**<trace>.DisConfig.CYcle**

Trace disassemble setting

**Debugger only**

Format:  

```
<trace>.DisConfig.CYcle "<name> [, <ext>]" <cycle>
```

**<cycle>:**  
Read <definition>
Write <definition>
Fetch <definition>
FLOW <definition>
Fetch1 <definition>
ReadOrFetch <definition>
ReadSpecial <definition>
WriteSpecial <definition>
MERGE ["<name>" <offset>...]

**<definition>:**  
TransientStrobe [<time>] [<channels>]

Strobe[2|3] [[<channel> [Low | High | Falling | Rising]]

Strobe[2|3]Sample [Last | Next | AT number] [<channel> [Low | High | Falling | Rising]]

Address[2]Sample [Last | Next | AT number] [<channel> [Low | High | Falling | Rising]]

Address[2] [<channels>]

Address[2] SHift <value>
The command `<trace>.DisConfig.CYcle` informs the trace software where to find program-fetch, data-read and data-write cycles in a not qualified trace recording which was taken by the PowerProbe or PowerIntegrator. With this information a standard bus trace listing can be generated.

```
AddressBase <address>

Data[2]Sample [Last | Next | AT number] [<channel> [Low | High | Falling | Rising]]

Data[2] [<channels>]

Data[2]SHift <value>

DataUnknown tbd.

DataWidthUnknown

SpaceID | SpaceIDSample tbd.

Word | Group | Integrator.<x> | eXt.<x>
```

The `<name>, <ext>` “name” is displayed in the cycle-type row of the Trace.List window. Its length is limited to 7. The "ext" is not displayed but used to differ between cycle types. Example: “rd_byte,0” --> rd_byte. This way it is possible to define different cycle types (rd_byte,0; rd_byte,1 …) which are displayed in the same way (rd_byte).

The `<cycle>` is used by the trace disassembler
- **Read**: data read cycle
- **Write**: data write cycle
- **Fetch**: program fetch cycle
- **Fetch1**: first program fetch code of an instruction
- **ReadOrFetch**: data-read or program-fetch cycle. The disassembler will do the final decision out of the program flow knowledge
- **ReadSpecial**: special cycle (e.g. dma)
- **WriteSpecial**: tbd.
- **MERGE**: merge the data of multiple cycles

The `<definition>` defines where to find a `<cycle>` in the trace, where to find the appropriate address and data, and how to display them.

**See also**
- `<trace>.DisConfig`
- `<trace>.DisConfig.view`

©1989-2019 Lauterbach GmbH
<trace>.DisConfig.FlowMode

Enable FlowTrace analysis
Debugger only

|---------|-----------------------------------------------|

Enables the analysis of certain FlowTrace protocols like ARM-ETM.

- **OFF**
  - FlowTrace analysis disabled

- **ETMB**
  - ARM-ETM FlowTrace analysis enabled, Mictor probe AB in use.

- **ETMK**
  - ARM-ETM FlowTrace analysis enabled, Mictor probe JK in use.

**See also**
- <trace>.DisConfig
- <trace>.DisConfig.view

<trace>.DisConfig.RESet

Reset trace disassemble setting
Debugger only

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;trace&gt;.DisConfig.RESet</th>
</tr>
</thead>
</table>

Resets the trace disassemble setting.

**See also**
- <trace>.DisConfig
- <trace>.DisConfig.view

<trace>.DisConfig.view

Trace disassemble setting

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;trace&gt;.DisConfig.view</th>
</tr>
</thead>
</table>

tbd.

**See also**
- <trace>.DisConfig
- <trace>.DisConfig.FlowMode
- <trace>.DisConfig.CYcle
- <trace>.DisConfig.RESet
The `<trace>.DRAW` command group can be used to plot the values of recorded trace data against time.

**See also**

- `<trace>.state`
- `<trace>.DRAW.Var`
- `Data.DRAWXY`
- `<trace>.DRAW.channel`
- `<trace>.DRAW.Data`

▲ ‘Introduction’ in ‘Application Note for the Trace.DRAW Command’
▲ ‘How to use the PROTOanalyzer’ in ‘DigRF Protocol Analyzer’
▲ ‘Release Information’ in ‘Release History’

---

**<trace>.DRAW.channel**

Plot no-data values against time

Format:

```
<trace>.DRAW.channel [<record_range>][%<format>]<items …> [I<options>]
```

```
<trace>.Chart.Draw (deprecated)
```

<record_range>:

```
<start_record>--<end_record>
```

<format>:

- Decimal. [<width>]
- DecimalU. [<width>]
- Hex. [<width>]
- Float. [leee | leeeDbI | leeeExT | leeeMFFP | …]

<width>:

```
DEFault | Byte | Word | Long | Quad | TByte | HByte
```

<items>:

```
ENERGY.Abs | POWER ...
```

/options>:

```
<draw_options> | FILE | BusTrace | RecScale | TimeScale | TimeZero | TimeREF | MinMax | Color | LOG | FIRST | Filter <filter_items> | Track | ZoomTrack
```

<draw_options>:

```
Points | Vector | MarkedVector | Steps | Impulses
```

<filter_items>:

```
<range> | <address> | <bitmask>
```

Plot specified `<item>` against time. This command is mainly used to plot no-data items.
The example below shows a temperature measurement recorded by a logic analyzer. The `Trace.DRAW.channel` command is used to show the temperature profile.

See also
- `<trace>.DRAW`
- 'Release Information' in 'Release History'
### <trace>.DRAW.Data

Plot data values against time.

**Format 1:**

```
<trace>.DRAW.Data [%<format>] {<data_address> | <data_range>} [/<options>]
```

**Format 2:**

```
<trace>.DRAW.Data [<start> |<range>] [ <hscale> | <vscale> <v_offset> ] [%<format>] {<data_address>} [/<options>]
```

**<format>:**

```
[ <radix> .] <width>
```

**<start>:**

```
<bookmark> | <record> | <time>
```

**<range>:**

```
<record_range> | <time_range>
```

**<vscale>:**

```
<float>
```

**<v_offset>:**

```

```

**<hscale>:**

```
<time>
```

**<radix>:**

```
Decimal | DecimalU | Hex | HexS | OCTal
```

**<width>:**

```
DEFault
Byte | Word | Long | Quad | …
```

**<option>:**

```
[ <source_option> ] [ <draw_option> ] [ <scale_option> ] [ <zoom_option> ]
```

**<source_option>:**

```
FILE | FlowTrace | BusTrace
```

**<draw_option>:**

```
Points | Vector | MarkedVector | Steps | Impulses | MinMax | LOG
```

**<scale_option>:**

```
RecScale | TimeScale | TimeZero | TimeREF
```

**<zoom_option>:**

```
Track | ZoomTrack
```

Plots one or more data values. An introduction to the usage of the Trace.DRAW.Data command is provided in "Application Note for the Trace.DRAW Command" (app_trace_draw.pdf).
Example for a core trace, trace filter applied, <time_range> specified:

```
PRIVATE &address
&address=0x40004068

Break.Set &address /Write /TraceEnable
Trace.List Time.Zero Default

Trace.Find Address &address Data 0x95
IF FOUND()
  (  
    ZERO Trace.RECORD.TIME(TRACK.RECORD())
  )
; <time_range> %<format> <data_address>
Trace.DRAW.Data 0uS--20.mS %Decimal.Byte &address /TimeZero
```

Example that scales the plot vertically into a window of the specified size:

```
WinPOS 20. 20. 120. 13. 16. 2. W001
; <vscale> <v_offset> %<format> <data_address>
Trace.DRAW.Data 1.9 0.0 %DecimalU.Byte 0x40004068
```

Example that
- specifies an starting point for the plot
- specifies an horizontal scale
- specifies a vertical scale
- specifies a vertical offset:

```
WinPOS 20. 20. 120. 13. 16. 2. W001  
Trace.Find Address vchar
IF FOUND()
  (  
    Trace.DRAW.Data TRACK.RECORD() 100uS 1.9 0.0 %DecimalU.Byte vchar
```
Description of the `<trace>.DRAW.Data Parameters`

**<record_range>**
To specify the record range, open the **Trace.List** window. Then choose the record range you want to plot.

**Example:**

```
; Opens the Trace.List window.
Trace.List /Track

; Plot graph for the specified record range.
Trace.DRAW.Data (-131072.)--(-121070.) %Hex.Word 0x1234 /Track
```

**Supported Radices**

- **Decimal, DecimalU** (Decimal Unsigned), **Hex**, and **Float** format the display of the y-axis.
- **Float.** The following floating-point formats are available:
  - IEEE | IEEEU | IeeeXt | IeeeQuad | IeeeXt10 | IeeeRev | IeeeS | IeeeDbIS | IeeeDbIT |
  - MFFP | Pdp11 | Pdp11Db | RTOSUH | RTOSUHD |
  - Dsp16 | Dsp16C | Dsp16Fix | Dsp32Fix |
  - M56 | M560 | M561 | LACCUM |
  - Fract8 | Fract16 | Fract24 | Fract32 | Fract48 | Fract64 |
  - UFract8 | UFract16 | UFract24 | UFract32 | UFract48 | UFract64 | Fract40G |
  - MICRO | MICRO64 | MILLI | MILLI64 | NANO64 | PICO64

**Basic Options**

- **FILE:** Visualizes the trace contents loaded with the command `<trace>.FILE`.
- **BusTrace:** This option is usually not required. It switches off the FlowTrace decoder. In the bus trace mode, all valid bus cycles are sampled.
RecScale: The resolution of the x-axis is based on records, e.g. if timestamps are not available. The record numbers are displayed on the x-axis.

<record_range>

TimeScale: The resolution of the x-axis is based on timestamps. The timestamps are displayed on the x-axis.

TimeZero: Displays the trace as a real-time display, time relative to the zero point. For more information about the zero point refer to ZERO.TimeREF: Displays the trace as a real-time display, time relative to the reference point. For more information about the reference point refer to <trace>.REF.

FIRST <address>: Defines which address contains the first part of the data value if the data value cannot be sampled within one bus cycle (e.g. a 16 bit data value on a 8 bit data bus).

See also

- <trace>.DRAW
- ‘Release Information’ in ‘Release History’
Plots the value changes of one or more variables against time, based on the recorded trace information. An introduction to the usage of the `Trace.DRAW.Var` command is provided in “Application Note for the Trace.DRAW Command” (app_trace_draw.pdf).
<vscale> Units per pixel of y-axis (floating point).

E.g. a signal has a max. height of 50 units shall be visualized window that has a height of 400 pixels: 50 units divided by 400 pixels = 0.125

By default the scale factor is set so that the window displays the complete possible value range for the selected variable.

<v_offset> Offset of y-axis (floating point). Default: 0.0.

<format> Numbering system of y-axis (decimal or hex).

<draw_options> Points: Displays each data value as a dot.
Vector: Connects the dots for the data values by vectors (default).
MarkedVector: Same as Vector, with every trace record holding a data value marked with a vertical line.
Steps: Connects the dots for the data values by steps.
Impulses: Draws each data value as a single pulse.
LOG: Displays the data values in a logarithmic format.

Example for a core trace with data trace enabled, no trace filter applied:

```plaintext
; plot value of a single variable
;      <format>   <var>
Trace.DRAW.Var %DEFault mstatic1

; plot values of two variables
; colors are assigned by TRACE32
Trace.DRAW.Var %DEFault mstatic1 fstatic fstatic2

; plot values of three variables
; colors are assigned by TRACE32
; <display_option> Steps
Trace.DRAW.Var %DEFault mstatic1 fstatic fstatic2 /Steps
```
Example for a core trace, trace filter applied, `<record_range>` specified:

```plaintext`
; advice trace generation logic to only generate trace messages for
; write accesses to variable vchar
Var.Break.Set vchar /Write /TraceEnable

; plot values of variable vchar for
; specified `<record_range>`
;                       %<record_range>          %<format>    <var>
Trace.DRAW.Var (-30000.)--(-29000.) %DEFault vchar
```

Example for a core trace, trace filter applied, `<time_range>` specified:

```plaintext`
Var.Break.Set vchar /Write /TraceEnable
Trace.List TIme.Zero DEFault

Trace.Find Address Var.RANGE("vchar") Data 0x95
IF FOUND()
(
    ZERO Trace.RECORD.TIME(TRACK.RECORD())
)
;                             %<time_range>          %<format>    <var>       /<scale_option>
Trace.DRAW.Var 0us--20.ms %DEFault vchar /TimeZero
```

Example that scales the plot vertically into a window of the specified size:

```plaintext`
WinPOS 20. 20. 120. 13. 16. 2. W001
; <vscale> <v_offset> <var>
Trace.Draw.Var 1.9 0.0 vchar
```

Example that
- specifies an starting point for the plot
- specifies an horizontal scale
- specifies a vertical scale
- specifies a vertical offset:

```plaintext`
WinPOS 20. 20. 120. 13. 16. 2. W001
Trace.Find Address Var.RANGE("vchar")
IF FOUND()
;             <record>           <hscale> <vscale> <v_offset> <var>
Trace.Draw.Var TRACK.RECORD() 100us 1.9 0.0 vchar
```
Perform a selective trace on the data of interest.

```
Analyzer.ReProgram
(
   ADDR AlphaBreak V.RANGE(P1) ; sample all accesses to P1
   Sample.Enable IF AlphaBreak
)
Go ; collect data
Break ;
Trace.DRAW %DEFAULT P1 ; Graphical display of P1
```

Multi-channel trace of three variables:

```
Analyzer.ReProgram
(
   ADDR AlphaBreak x_axis
   ADDR BetaBreak y_axis
   ADDR CharlyBreak z_axis

   Sample.Enable IF (AlphaBreak||BetaBreak||CharlBreak)&&Write
)
Go ; collect data
Break
Trace.DRAW %DEFAULT x_axis y_axis z_axis ; plot
```

See also
- `<trace>.DRAW`
- 'Release Information' in 'Release History'
- 'Filter and Trigger - Single-Core and AMP' in 'AURIX Trace Training'
If the port analyzer is in timing mode (not slave), it will be run either totally free or controlled by the emulator operation.

**ALways**

The analyzer can be used independently of the state of the emulator.

**Running**

The port analyzer is enabled only, while the emulator system is in real-time emulation. When running to a breakpoint or stopped by an asynchronous event, the timing analyzer will be stopped, too. The trigger system of the port analyzer is not activated until the emulation is started.

---

**<trace>.EnableChannel**

Enable FDX communication

Format: `<trace>.EnableChannel [address]`

Enables the data transfer over a FDX channel. Without parameters all existing FDX channels are enabled.
Using the `<trace>.EXPORT` command group, you can export trace data for processing in other applications. Various export file formats are available, including ASCII, binary, PGT, VERILOG, etc.

**NOTE:**

The various export formats are primarily designed for import into other applications. Trace data exported with the `<trace>.EXPORT.*` commands can only be imported back into TRACE32 if you inform the debugger about all the trace-relevant circumstances.

We recommend the following approach if you want to view and analyze recorded trace data in a subsequent TRACE32 session:

1. Save the trace data to file using `<trace>.SAVE`.
2. To load this file back into TRACE32, use `<trace>.LOAD`.

---

**See also**

- `<trace>.SAVE`
- `<trace>.state`
- `<trace>.EXPORT.Ascii`
- `<trace>.state`
- `<trace>.EXPORT.Bin`
- `<trace>.EXPORT.Binary`
- `<trace>.EXPORT.CSVFunc`
- `<trace>.EXPORT flow`
- `<trace>.EXPORT.Func`
- `<trace>.EXPORT.MTV`
- `<trace>.EXPORT.TASKEVENTS`
- `<trace>.EXPORT.TracePort`
- `<trace>.EXPORT.VCD`
- `<trace>.EXPORT.VERILOG`
- `<trace>.EXPORT.VHDL`
- `<trace>.IMPORT`

▲ 'Further NEXUS Trace Analysis’ in ‘ARM Application Note for MXC Chips’
▲ 'Release Information’ in ‘Release History’
▲ 'Further NEXUS Trace Analysis’ in ‘StarCore Application Note for MXC Chips’
Exports the trace contents to an ASCII file. White spaces are used as delimiters. The default extension of the file name is *.ad.

**FILTER**  
Exports only records matching the filter. For an example, see below.

**ShowRecord**  
Includes the trace record numbers in the export file.

<option>  
For a description of the other <options>, see <trace>.EXPORT.flow.

**Example:**

```plaintext
Trace.EXPORT.Ascii ~~\myfile.ad (-120000.)--(-1.) /ShowRecord \ 
/FILTER ADDRESS Var.RANGE(sieve)
```

The backslash \ is used as a line continuation character. No white space permitted after the backslash.

**See also**
- <trace>.EXPORT
- <trace>.EXPORT.flow
Exports the trace contents to a file in binary format. This command is used to export logic analyzer (PowerProbe, Integrator, IProbe) recordings. The data is stored in little endian format.

The file starts with a text header describing item names and byte size of each item. Each record begins with an 8 byte time stamp (1 ns per tick), followed by the selected items in the order as given in the command. Each item has a minimum width of 1 byte (max. 8 byte). The following options are available:

**FILE** | Exports the trace contents loaded with `<trace>.FILE`.  
**NoDummy** | Exclude records which do not hold flow information (do not use when exporting logic analyzer data).  
**NoHeader** | The resulting file does not contain a header.  
**NoTimeStamps** | The records do not contain the 8 byte timestamp.  
**NoFetch** | Exclude control cycles from export.

**Example:** Export data from a parallel port recorded with the IProbe

```plaintext
; define the data word of the port, connected to signals ip.00...ip.07  
NAME.WORD W.PARPORT ip.00 ip.01 ip.02 ip.03 ip.04 ip.05 ip.06 ip.07

; export analyzer data  
IPProbe.EXPORT.Bin pardat.ad W.PARPORT /NoHeader

; show resulting file: one record has 9 byte (W.PARPORT has 1 bytes)  
DUMP pardat.ad /WIDTH 9
```

See also
- `<trace>.EXPORT`  
- `<trace>.EXPORT.flow`

©1989-2019 Lauterbach GmbH
Exports the branch events from the trace data.

**Format:**

```
<trace>.EXPORT.BRANCHFLOW <file> [<record_range>] [/<options>]
```

**<option>:**

- **TRaceRecord**: Branch events are exported with trace record numbers.
- **NOINNER**: Only branch events that jump to the current symbol are exported. The internal branch is not exported.
- **NOSymbol**: Branch events are exported with addresses instead of symbols.
- **CALLer**: Branch events are exported with caller events.

**See also**

- <trace>.EXPORT
- <trace>.EXPORT.flow
Exports the function nesting of the recorded trace data to a CSV file for processing by an external tool. The default extension of the file name is *\csv.

Example:

```plaintext
; export the entire function nesting
Analyzer.EXPORT.CSVFunc ~\csvfunc_all.csv
EDIT ~\csvfunc_all.csv
```

See also

- `<trace>.EXPORT`
- `<trace>.EXPORT.flow`
- 'Release Information' in 'Release History'
Export trace data

Format:  \texttt{<trace>.EXPORT.flow <file> [<record\_range>] [/<options>]}  

\texttt{<option>}:  
\begin{itemize}
  \item FILE
  \item ZIP
  \item CORE <number>
  \item SplitCORE  |  MergeCORE  |  JoinCORE  (SMP tracing only)
\end{itemize}

Exports the trace contents for postprocessing by an external analysis tool. The default extension of the file name is *.ad.

The trace contents can only be exported when the trace is in \texttt{OFF} or \texttt{break} state. Please refer to the \texttt{Trace.state} command for more information.

The default export format is binary. A description of the binary format is given at the end of this command description.

| FILE | Exports the trace contents loaded with \texttt{<trace>.FILE}. |
| ZIP  | File is compressed with the gzip archive format.          |

In the case of an SMP system, the following options are provided:

| MergeCORE (default) | The trace information for all cores is exported. |
| SplitCORE          | Same as MergeCORE.                                  |
| JoinCORE           | Same as MergeCORE.                                  |
| CORE <number>      | Only the trace information for the specified core is exported. |
When an exported file contains a file header (not the case e.g. for /ByteStream, /CoreByteStream, ...) it has the following format:

<table>
<thead>
<tr>
<th>Byte Nr.</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..31</td>
<td>Export file header string (“trace32 analyzer export data” 0x1a 0x00)</td>
</tr>
<tr>
<td>32</td>
<td>Reserved (set to zero for IMPORT)</td>
</tr>
<tr>
<td>33</td>
<td>CPU code</td>
</tr>
<tr>
<td>34</td>
<td>Timestamp available flag</td>
</tr>
<tr>
<td>35</td>
<td>Prestore mode flag</td>
</tr>
<tr>
<td>36</td>
<td>Trigger unit available flag</td>
</tr>
<tr>
<td>37</td>
<td>Port analyzer available/mode flag</td>
</tr>
<tr>
<td>38</td>
<td>Analyzer type</td>
</tr>
<tr>
<td>39</td>
<td>Reserved</td>
</tr>
<tr>
<td>40</td>
<td>Length of one record in bytes (0x20)</td>
</tr>
<tr>
<td>41..43</td>
<td>Reserved</td>
</tr>
<tr>
<td>44..47</td>
<td>Number of records in file (if record number can exceed 32 bits, e.g. Trace.Mode.STREAM, calculate number of records based on file size)</td>
</tr>
<tr>
<td>48..51</td>
<td>Record number of last recorded record</td>
</tr>
<tr>
<td>52..55</td>
<td>Reference record number</td>
</tr>
<tr>
<td>56..63</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Byte Nr.** | **Meaning**
---|---
0..3 | Cycle information flags:
  Bit 0: data cycle
  Bit 1: program cycle
  Bit 6: write cycle
  Bit 8: Power Architecture MPC5XXX: read/write cycle of peripheral NEXUS bus master
  Bit 21: FLOW ERROR
  Bit 25: FIFO OVERFLOW
  Bit 31: OWNERSHIP Cycle
<table>
<thead>
<tr>
<th>Byte Nr.</th>
<th>Meaning</th>
</tr>
</thead>
</table>
| 4        | Data byte enable mask  
Bit 0: Byte 0 valid  
Bit 1: Byte 1 valid  
… |
| 5        | CPU specific information  
SH2A I-bus marker (bit meaning is device specific):  
Bit 0: iadma bus  
Bit 1: idma bus  
Bit 2: icpu1 bus  
Bit 3: icpu2 bus  
ARM Bustrace:  
Bit 0: EXEC signal (relevant only when SYStem.Option EXEC is set to ON)  
ARM Flowtrace (ETM/PTM):  
Bit 1: Thumb Mode  
Bit 2: ARM Mode  
Bit 5: not executed  
Bit 6: executed |
| 6        | Reserved |
| 7        | Core number (only on SMP targets) |
| 8..11    | Address (bus/data) |
| 12..15   | Address (upper part or program flow address) |
| 16..23   | Data bytes (64 bits) |
| 24..31   | Timestamp (time relative to ZERO in ns) |

See also

- `<trace>.EXPORT`  
- `<trace>.EXPORT.Bin`  
- `<trace>.EXPORT.CSVFunc`  
- `<trace>.EXPORT.MTV`  
- `<trace>.EXPORT.TracePort`  
- `<trace>.EXPORT.VERILOG`  
- `<trace>.EXPORT.Arscii`  
- `<trace>.EXPORT.BRANCHFLOW`  
- `<trace>.EXPORT.Func`  
- `<trace>.EXPORT.TASKEVENTS`  
- `<trace>.EXPORT.VCD`  
- `<trace>.EXPORT.VHDL`
Export function nesting

Exports the function nesting from the trace contents to a binary file. The default extension of the file name is *.ad.

Exported function nestings contain the function entries and exits as well as task switches with task entries and exits. Function nestings are displayed in the `<trace>.ListNesting` window.

Example:

```
Analyzer.EXPORT.Func ~~~\trace.ad (-131072.)--(-100000.)
```

See also

- `<trace>.EXPORT`
- `<trace>.EXPORT.flow`
Export in MCDS Trace Viewer format

Format:  
<trace>.EXPORT.MTV <file> [<record_range>] [/<options>]

<option>:  
FILE | BusTrace | NoDummy

Exports a trace recording in the MCDS Trace Viewer format.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Exports the trace contents loaded with &lt;trace&gt;.FILE.</td>
</tr>
<tr>
<td>BusTrace</td>
<td>The trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td>NoDummy</td>
<td>Exclude records which do not hold flow information (do not use when exporting logic analyzer data).</td>
</tr>
</tbody>
</table>

See also
- <trace>.EXPORT
- <trace>.EXPORT.flow
Generates a CSV file that contains task event information and time information.


See also

- `<trace>.EXPORT`  
- `<trace>.EXPORT.flow`

- 'Release Information' in 'Release History'
Format: `<trace>.EXPORT.TracePort <file> [record_range] [/<options>]`

*<option>*: FILE

ZIP

FullByteStream | ByteStream | CoreByteStream | TimedByteStream | TPStream | TimedCoreByteStream (ETMv3 only) | NibbleStream

Exports the recorded trace data in a low-level binary format. Available options depend on the used processor architecture and trace port.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ByteStream</strong></td>
<td>Exports the byte stream broadcast by the ETM (same as TP column if command <code>Trace.List TP DEFAULT</code> is used).</td>
</tr>
<tr>
<td><strong>CoreByteStream</strong></td>
<td>Similar to the option <strong>ByteStream</strong>, but strips away synchronisation patterns (continuous mode) and trace source identifiers (e.g. in case of multicore systems). The exported data is that shown in the <strong>TPC</strong> column in the command <code>Trace.List TPC DEFAULT</code>. By default, the data corresponding to the currently active core is exported (selected by the <strong>CORE</strong> command), but this can be overridden by the <code>/CORE &lt;number&gt;</code> option.</td>
</tr>
<tr>
<td><strong>TimedByteStream</strong></td>
<td>Exports the byte stream broadcast by the ETM together with the <strong>Time.Zero</strong> timestamp information. For a description of the file format, see below.</td>
</tr>
<tr>
<td><strong>TPStream</strong></td>
<td>Power Architecture only. Exports NEXUS packets received through Aurora interface.</td>
</tr>
<tr>
<td><strong>TimedCoreByteStream (ETMv3 only)</strong></td>
<td>Exports the unwrapped byte stream broadcast by the ETM together with the <strong>Time.Zero</strong> timestamp information. This format also supports multiple cores in SMP configuration.</td>
</tr>
<tr>
<td><strong>FullByteStream</strong></td>
<td>Exports the trace data in the format that allows to re-import it using the command <code>LA.IMPORT &lt;file&gt;</code>.</td>
</tr>
<tr>
<td><strong>NibbleStream</strong></td>
<td>Exports just pure STP data, excluding non-STP headers (STP = System Trace Protocol).</td>
</tr>
</tbody>
</table>
The **TimedByteStream** format consists of two-byte records; possible formats are:

- **0y0xxxxxx** `<tracedata_byte>`
  - `xxxxxx`: Time relative to previous records (in nanoseconds).

- **0y10xxxxxx 0yxxxxxxxx**
  - `xxxxxx`: Time relative to previous record (bits 7 to 12).
  - `xxxxxxxx`: Upper bits (bits 13 to 20).

- **0y11000xxx 0yxxxxxxxx**
  - `xxx`: Selects which part of the absolute time is transferred.
  - `xxxxxxx`: Byte of absolute timestamp.

- **0y11001000 0yxxxxxxxx**
  - `xxxxxxxx`: Selects to which core the following data belongs (only in **CoreByteStream** with SMP).

See also
- ![<trace>.EXPORT](#)
- ![<trace>.EXPORT.flow](#)

**<trace>.EXPORT.VCD**

Export trace data in VCD format

**Format:**

```
<trace>.EXPORT.VCD <file> [<record_range>] [<items> ...] [<options>]
```

Exports the trace contents collected by the TRACE32 logic analyzers PowerProbe and PowerIntegrator to a file in VCD format. The default extension of the file name is `*.ad`.

**<options>**

For a description of the `<options>`, see ![<trace>.EXPORT.flow](#).

See also
- ![<trace>.EXPORT](#)
- ![<trace>.EXPORT.flow](#)
**<trace>.EXPORT.VERILOG**

Export trace data in VERILOG format

Format:  
<trace>.EXPORT.VERILOG <file> [<record_range>] [<item> …] [/<options>]

Exports the trace contents collected by the TRACE32 logic analyzers PowerProbe and PowerIntegrator to a file in VERILOG format. The default extension of the file name is *ad*.

<options>  
For a description of the <options>, see <trace>.EXPORT.flow.

See also
- <trace>.EXPORT
- <trace>.EXPORT.flow

---

**<trace>.EXPORT.VHDL**

Export trace data in VHDL format

Format:  
<trace>.EXPORT.VHDL <file> [<record_range>] [<item> …] [/<options>]

Exports the trace contents collected by the TRACE32 logic analyzers PowerProbe and PowerIntegrator to a file in VHDL format. The default extension of the file name is *ad*.

<options>  
For a description of the <options>, see <trace>.EXPORT.flow.

See also
- <trace>.EXPORT
- <trace>.EXPORT.flow
Load a file into the file trace buffer

**Format:**

```
<trace>.FILE <file> [/Config]
```

Loads trace data from a file into a dedicated file trace buffer on the host. Typically this feature is used to analyze data in a simulator or to compare different recordings. The default extension of the file name is *.ad.

**Config**

Restore analyzer and NAME settings contained in `<file>`. Only applicable for Trace.METHOD Probe and Trace.METHOD Integrator.

**Example:** To use the file trace buffer as source for trace-related commands, the commands need to be invoked with the additional parameter `/FILE`

```
Trace.FILE myfile.ad
Trace.List /FILE
```

Windows working on trace contents loaded with the Trace.FILE command are marked with a red label FILE in the bottom-left corner:

Trace-related commands without the parameter /FILE keep operating on the trace data stored in the “normal” trace buffer which is filled when recording data using the analyzer hardware (e.g. PowerTrace, PowerProbe, PowerIntegrator).
Using the file trace buffer and the “normal” trace buffer concurrently allows to compare trace data stored in a file from a previous recording with recently recorded data as shown in the following example:

```
Trace.FILE test4 ; load trace contents from test4.ad
Trace.List /FILE ; display loaded trace contents
Trace.Chart.sYmbol /FILE ; works on loaded trace data

; compare the recently recorded trace with the trace contents loaded
; from test4.ad regarding to the addresses
Trace.ComPare , Address /FILE
```

**NOTE:**
In addition to `Trace.FILE` there is a command `Trace.LOAD` for loading trace data from a file into the “normal” trace buffer. Therefore data loaded with `Trace.LOAD` is treated as if it was recently recorded by the analyzer hardware. As a consequence all standard trace commands automatically work on the loaded via `Trace.LOAD` (without specifying additional parameters).

**See also**
- `<trace>.LOAD`
- `<trace>.state`
- `Trace`

▲ 'Release Information' in 'Release History'
### <trace>.Find

Find specified entry in trace

#### Format:

```
<trace>.Find [<record_number> | <record_range>] [ <item> … ] [ <options> ]
```

#### <item>:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Address</td>
<td>`&lt;address&gt;</td>
</tr>
<tr>
<td>Address.MATCH</td>
<td>`&lt;address&gt;</td>
</tr>
<tr>
<td>FAddress</td>
<td>`&lt;address&gt;</td>
</tr>
<tr>
<td>Data</td>
<td>`&lt;value&gt;</td>
</tr>
<tr>
<td>Data !</td>
<td>`&lt;value&gt;</td>
</tr>
<tr>
<td>Data `</td>
<td>`&lt;value&gt;</td>
</tr>
<tr>
<td>CYcle</td>
<td><code>&lt;cycle_type&gt;</code></td>
</tr>
<tr>
<td>Var</td>
<td><code>&lt;group_name&gt;</code></td>
</tr>
<tr>
<td>Time.Back</td>
<td><code>&lt;time_range&gt;</code></td>
</tr>
<tr>
<td>Time.Zero</td>
<td><code>&lt;time_range&gt;</code></td>
</tr>
<tr>
<td>Time.AddressBack</td>
<td><code>&lt;time_range&gt;</code></td>
</tr>
<tr>
<td>Time.Address Fore</td>
<td><code>&lt;time_range&gt;</code></td>
</tr>
</tbody>
</table>

#### <item>:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXCEPTION</td>
<td>INTERRUPT</td>
</tr>
<tr>
<td>FIFO FULL</td>
<td>FLOWERROR</td>
</tr>
<tr>
<td>CORE</td>
<td>IGNORE</td>
</tr>
</tbody>
</table>

#### <option>:

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Back</td>
<td>FILE</td>
</tr>
<tr>
<td>NoFind</td>
<td>ALL</td>
</tr>
<tr>
<td>FlowTrace</td>
<td>BusTrace</td>
</tr>
</tbody>
</table>

Searches for matching items in the given range of trace records. The default search range is the complete trace. When the command is invoked without parameters, the previous search is repeated.

If the search finds a matching trace record, the PRACTICE function `FOUND()` will return `TRUE()`. If a matching trace record was found, `TRACK.RECORD()` returns the record number of the matching record.

Details about the `<trace>.Find` command can be found in “Application Note for the Trace.Find Command” (app_trace_find.pdf).
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ALL</strong></td>
<td>Searches for all occurrences and displays the result in the message line. The number of occurrences can be returned with the function <code>FOUND.COUNT()</code>.</td>
</tr>
<tr>
<td><strong>Back</strong></td>
<td>Search backwards.</td>
</tr>
<tr>
<td><strong>FILE</strong></td>
<td>Takes trace memory contents loaded by <code>Trace.FILE</code>.</td>
</tr>
<tr>
<td><strong>NoFind</strong></td>
<td>Set up search, but don’t search. Search can be done at a later point by using the <code>&lt;trace&gt;.Find</code> command without parameters.</td>
</tr>
<tr>
<td><strong>FlowTrace</strong></td>
<td>The trace works as a program flow trace. This option is usually not required.</td>
</tr>
<tr>
<td><strong>BusTrace</strong></td>
<td>The trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td><strong>Var</strong></td>
<td>Search for static variable access cycles.</td>
</tr>
</tbody>
</table>

**Examples:**

```
; find matching address and data, 
; start search at the beginning of the trace recording
Trace.Find Address 0x100--0x200 Data.B 0x55
; find next match
Trace.Find
```

```
; find matching address and data, 
; start search at the beginning of the trace recording 
; print all matching records
Trace.Find Address 0x100--0x200 Data.B 0x55
WHILE FOUND()
  PRINT "Matching record: " TRACK.RECORD()
  Trace.Find
)
```

```
; find any instruction of the function sieve, 
; start search at the end of the trace recording
Trace.Find Address Var.RANGE(sieve) /Back
```
; find specified data value in record range (-1000.)--(-700.)
; start search at the beginning of the trace recording
Trace.Find (-1000.)--(-700.) Data 0x100

; find specified data,
; start search at the beginning of the trace recording
Trace.Find Data 0x0--0xAa

; find write accesses to variable flags with specified data values
; start search at specified record number down to the end of the
; trace recording
Trace.Find -3224833. Address V.RANGE("flags") Data 0x0--0xaa CYcle Write
; find next match
Trace.Find

; find read access to variable flags[3]
; start search at the specified record number up to the beginning of the
; trace recording
Trace.Find -3224832. Address Var.RANGE("flags[3]") CYcle Read /Back

; find any trace information assigned to the group sieve
; start search at the beginning of the trace recording
Trace.Find GROUP "sieve"

; find all trace entries with a TIme.Back time between 500.us--700.us
Trace.Find TIme.Back 0.500us--0.700us

; find ptrace cycle which contains address 0x40000B7C
Trace.Find FAddress 0x40000B7C

; find one of the specified data values
Trace.Find Data 0x5--0x44 OR Data 0x55 OR Data 0x0

; search for rising edge of
; the NMI signal
Trace.Find NMI ON AT -1. NMI OFF
; search for trace entry generated by core 0 in an SMP system
Trace.Find , CORE 0

; searching for pin RXD high
Port.Find Port.RXD High

; searching for pin RXD high and pin CTS low
Port.Find Port.RXD High Port.CTS Low

; searching for pin RXD high or pin CTS low
Port.Find Port.RXD High OR Port.CTS Low

; searching for pin RXD changing from low to high
Port.Find Port.RXD Low AT 1. Port.RXD High

; searching for pin RXD changing from low to high, but staying high for
; minimum 2 clock cycles
Port.Find Port.RXD Low AT 1. Port.RXD High AT 2. Port.RXD High

See also
- <trace>.FindAll
- <trace>.FindChange
- <trace>.state
- Trace
- FOUND()
- FOUND.COUNT()

▲ 'The Trace Find Dialog' in 'Application Note for the Trace.Find Command'
▲ 'Release Information' in 'Release History'
▲ 'Displaying the Trace' in 'Training FIRE Analyzer'
## `<trace>.FindAll`

Find all specified entries in trace

| Format: | `<trace>.FindAll` [ `<record_number>` | `<record_range>` ] `<items>` ... [ `<options>` ] |
|---|---|
| `<option>`: | Back
| | FILE
| | FlowTrace | BusTrace
| | `<other_options>` |

Searches for and displays all entries matching the item specification. Without range, the complete trace memory is searched for matching entries.

### Back

The option **Back** reverses the direction of the search command.

### BusTrace

The trace works as a bus trace. This option is usually not required.

### FILE

Takes trace memory contents loaded by `Trace.FILE`.

### FlowTrace

The trace works as a program flow trace. This option is usually not required.

### List

Change the default display of the result.

### `<other_options>`

Details about the `<trace>.FindAll` command can be found in “Application Note for the Trace.Find Command” (app_trace_find.pdf).

### Example:

```
Trace.FindAll , sYmbol sieve /List TIme.Zero DEFault
```

### See also

- `<trace>.Find`
- `<trace>.FindChange`
- `<trace>.state`
- `Trace`
- `FOUND()`
- `FOUND.COUNT()`
- 'Release Information' in 'Release History'

©1989-2019 Lauterbach GmbH
<trace>.FindChange

Search for changes in trace flow

Format:  
<trace>.FindChange [<record_number> | <record_range>] [{<items>}]  
[<options>]

<iitems>:  
OR
<channels>
AT <offset>

<options>:  
Back
FILE
FlowTrace | BusTrace
<other_options>

Searches for entries in the given range where the specified items have new values. Without range the entry is searched within the complete trace memory. Without items the command searches for changes in program flow. This is useful to search for the end of a complex program loop, or in general to search for “something happens” in a traced program flow.

BACK  
Reverses the direction of the search command.

FILE  
Takes trace memory contents loaded by Trace.FILE.

FlowTrace  
The trace works as a program flow trace. This option is usually not required.

BusTrace  
The trace works as a bus trace. This option is usually not required.

<other_options>  
Details about the <trace>.FindChange command can be found in “Application Note for the Trace.Find Command” (app_trace_find.pdf).

See also

■ <trace>.Find  ■ <trace>.FindAll  ■ <trace>.state  ■ Trace

©1989-2019 Lauterbach GmbH

General Commands Reference Guide T  
204
Processes all trace data in the analyzer and calculates the instruction flow for all of it. This is in contrast to <trace>.FLOWSTART which discards the processing results and thus indirectly causes a reprocessing of the limited set of trace data required to draw the currently open windows (reprocessing on demand).

The command is used mostly for diagnostic purposes.

See also

- <trace>.state
- Trace

### <trace>.FLOWSTART

**Restart flowtrace processing**

**Debugger only**

Format:  
\[
<\text{trace}>.FLOWSTART [\langle\text{address}\rangle]
\]

Discards all results from previous decoding of instruction flow. This indirectly causes a reprocessing of the limited set of trace data required to draw the currently open windows (reprocessing on demand). Effectively the decoding of flow information is done again “from the start”.

The command is typically used when the memory contents at the time of decoding was wrong and the decoding is therefore incorrect (contains flow errors). The command is executed after providing a correct memory image (e.g. by activating chip selects) to re-initialize the flow processing.

The optional address parameter can be used to indicate the address of the first instruction executed by the processor. In this way the debugger can correctly decode code sequences even before the first sync message appears in the trace stream.

See also

- <trace>.state
- Trace
- FOUND()
Displays the current state of all input lines. The format of the channel definition is similar to the `<trace>.View` command. This command can be executed, while the port analyzer is running.

For valid channel names refer to the:

- **Processor Architecture Manuals**
- **Target Guides FIRE**
- **Target Guides ICE**
Displays the state of all port lines in hex and HIGH/LOW format.

```
E::Port.Get
direct  00 LOW LOW LOW LOW LOW LOW LOW 33 HIGH HIGH LOW
        LOW HIGH HIGH LOW 0F HIGH HIGH HIGH LOW LOW LOW LOW LOW 55
        p.50 p.51 p.52 p.53 p.54 p.55 p.56 p.57 p.x p.x0 p.x1 p.x2 p.x3 p.x4 p.x5
HIGH LOW HIGH LOW LOW LOW 00 LOW LOW LOW LOW LOW LOW LOW LOW
LOW LOW FF HIGH HIGH HIGH HIGH HIGH HIGH HIGH HIGH HIGH FF HIGH HIGH HIGH
        p.13 p.14 p.15 p.16 p.17 p.0 p.00 p.01 p.02 p.03 p.04 p.05 p.06 p.07
HIGH HIGH HIGH HIGH HIGH 07 HIGH HIGH LOW LOW LOW LOW LOW
```

Displays the state of port lines P2 in binary format, lines P3.0, P3.1 and P3.2 in timing waveform, port lines P5 in decimal format, port lines P4 in hex format and port PX in ASCII format.

```
E::Port.Get ALL
record  p.ipl0 p.ipl1 p.ipl2 p.bclr p.cs0 p.cs1 p.cs2 p.cs3 p.pb8
direct  
        p.iack1 p.tin1 p.tout1 p.tin2 p.tout2 p.wdog p.rxd1 p.txd1 p.rclk1
        p.tclk1 p.cts1 p.rts1 p.cd1 p.brg1 p.rxd2 p.txd2 p.rclk2 p.tclk2
        p.rts3 p.cd3 p.brg3 p.x0 p.x1 p.x2 p.x3 p.x4 p.x5 p.x6 p.x7 p.bnk0
```

Displays the state of all port lines in timing waveform.

See also
- `<trace>.state`
- `Trace`

©1989-2019 Lauterbach GmbH
<trace>.GOTO

Move cursor to specified trace record

[Examples]

Format:  
<trace>.GOTO "<bookmark>" | <record_number> | <time> [/<options>]

<option>:  
FILE  
FlowTrace | BusTrace | CORE <number>

Goes to the specified trace record in a Trace.* window by moving the cursor to that trace record. Alternatively, click the Goto button in a Trace.* window, and enter a record number, a time index, or the name of a trace bookmark.

A To go to a trace <bookmark>, enclose the bookmark name in quotation marks.

B To go to a trace <record_number>, append a period (.). Mind the + or - sign of the record number.

C To go to a <time>, prepend a plus or minus sign and append the unit of measurement. To view the <time>, include the TIme.ZERO column in the Trace.List command, as shown in the example below.

<table>
<thead>
<tr>
<th>BusTrace</th>
<th>The trace works as a bus trace. This option is usually not required.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Takes trace memory contents loaded by Trace.FILE.</td>
</tr>
<tr>
<td>FlowTrace</td>
<td>The trace works as a program flow trace. This option is usually not required.</td>
</tr>
<tr>
<td>CORE</td>
<td>The goto operation takes the specified core number into account. Only available for SMP multicore tracing.</td>
</tr>
</tbody>
</table>

Description of Buttons in the Trace Goto Dialog

<table>
<thead>
<tr>
<th>Previous / Next</th>
<th>Go to the previous / next user-defined trace bookmark. Trace bookmarks are created with Trace.BookMark.</th>
</tr>
</thead>
<tbody>
<tr>
<td>First / Last</td>
<td>Go to the first / last trace record.</td>
</tr>
<tr>
<td>Trigger</td>
<td>Go to the trigger record.</td>
</tr>
<tr>
<td>Ref</td>
<td>Go to the reference point, which has been set with the Trace.REF command. You can also set the reference point by right-clicking in a Trace.* window and pointing to Set Ref in the Trace popup menu.</td>
</tr>
</tbody>
</table>
### Examples

The **Trace.List** window is always opened with the **Track** option. Thanks to the **Track** option, the subsequent **Trace.GOTO** command scrolls to the desired trace record in the **Trace.List** window.

**Example 1:** Go to a `<record_number>`.

```plaintext
; open the Trace.List window
Trace.List /Track

; go to this `record_number` in the Trace.List window
Trace.GOTO -12000.
```

**Example 2:** Go to a trace `<bookmark>`.

```plaintext
; create a trace bookmark named 'BM1' for record -14000.
Trace.BookMark "BM1" -14000.

; open the Trace.List window
Trace.List /Track

; go to this `bookmark` in the Trace.List window
Trace.GOTO "BM1"
```

**Example 3:** Go to a `<time>` index.

```plaintext
; first_column other_columns
Trace.List Time.ZERO Default /Track

; go to this `<time>` in the Trace.List window
Trace.GOTO -5.000ms
```

### See also

- CIProbe
- `<trace>.REF`
- `<trace>.TRACK`
- `<trace>.state`
- BookMark
- Analyzer.RECORD.ADDRESS()
- Analyzer.RECORD.OFFSET()
- Analyzer.REF()
The **LA.IMPORT** command group is used to load trace data from a file into TRACE32 and to analyze it just like data recorded with a TRACE32 trace tool.

The trace data can be obtained by the application software itself or by another tool or by TRACE32 in a previous debug session in which the processing could not be performed for some reasons.

Trace data successfully obtained and analyzed by TRACE32 can be stored by **<trace>.SAVE** and re-viewed by using the **<trace>.LOAD** command. This is the more convenient way because **<trace>.SAVE** stores a lot of additional information used for the analysis. **LA.IMPORT** imports only the trace raw data. For proper processing you need to inform the debugger about all the trace-relevant circumstances.

All kind of trace postprocessing is only possible with the trace method ‘LA’ (Logic Analyzer). Therefore you need to use **LA.IMPORT** and ‘LA.’ command group for all analysis commands or maybe better switch the trace method to ‘LA’ (**Trace.METHOD LA**) and use the command group ‘Trace.’ for all further operations.

**LA.IMPORT** supports different kinds of trace data and formats. Therefore different commands are provided. For command descriptions, see **LA.IMPORT.ETB**, **LA.IMPORT.flow**, **LA.IMPORT.TraceFile**, **LA.IMPORT.TracePort**, **LA.IMPORT.VCD**, below.

Most trace data is stored in the file in the timely order the data had been generated.

An exception is the on-chip trace buffer, which is typically used as a circular ring buffer overwriting the trace data all the time until the point of interest is reached.
If this buffer is saved into a file, you need to know the wrap pointer for being able to get the data in a timely order. **LA.IMPORT.WRAP** and **LA.IMPORT.GUESSWRAP** will deal with this concern.

For post processing trace data loaded by **LA.IMPORT** you need to take the following steps:

1. Start TRACE32 to run as simulator (config.t32 -> PBI=SIM). You neither need a debugger hardware nor a target. You can run TRACE32 as debugger as well, but for the postprocessing this is not needed.

2. Adjust all trace relevant settings like for a real target by running the start-up script you used for generating the trace data. For postprocessing an ETMv4 even further setups might be needed which normally the debugger would read out from the ETM module (ETM.COND, ETM.INSTPO, ETM.QE).

   If the start-up script is **not** available, then try this:

   - At best selecting the chip you are debugging (**SYStem.CPU ...**) is sufficient.
   - For trace data coming from a ARM CoreSight system, all commands describing the trace system on the chip are required (**SYStem.CONFIG ...**).
   - Further all settings for the trace sources done at recording time are needed (e.g. ETM. ...).

3. Load your target application (**Data.LOAD ...**).

4. Import the trace raw data (**LA.IMPORT. ...**).

5. Now you can use all trace display and analysis functions, e.g.

   **LA.List TP TPC TPINFO DEFault List.NoDummy.OFF ; with diagnostics**
Example:

```plaintext
SYstem.CPU CortexA15

SYstem.CONFIG COREDEBUG.Base 0x82010000
SYstem.CONFIG.ETM.Base 0x8201c000
SYstem.CONFIG.FUNNEL.Base 0x80040000
SYstem.CONFIG.FUNNEL.ATBSource ETM 0
SYstem.CONFIG.ETB.Base 0x80010000

ETM.PortMode.Wrapped
ETM.TraceID 0x55

SYstem.Up

Data.LOAD.Elf myfile.elf

Trace.METHOD.LA
Trace.IMPORT.ETB mydata.bin
Trace.IMPORT.GUESSWRAP

Trace.List TP TPC TPINFO DEFault List.NoDummy.OFF
```

**See also**
- `<trace>.IMPORT.CoreByteStream`
- `<trace>.IMPORT.flow`
- `<trace>.IMPORT.STP`
- `<trace>.IMPORT.TraceFile`
- `<trace>.IMPORT.VCD`
- `<trace>.IMPORT.WRAP`
- `<trace>.IMPORT.ETB`
- `<trace>.IMPORT.GUESSWRAP`
- `<trace>.IMPORT.STPByteStream`
- `<trace>.IMPORT.TracePort`
- `<trace>.IMPORT.WRAP`

---

**<trace>.IMPORT.CoreByteStream**

Import pure single core trace data

**Format:**

```
LA.IMPORT.CoreByteStream <file>
```

Imports pure single core trace data (for x86 IPT traces).

**See also**

- `<trace>.IMPORT`
**<trace>.IMPORT.ETB**

**Import on-chip trace data**

Imports a pure binary trace data file obtained from an on-chip trace buffer like ARM CoreSight ETB, ETF, ETR.

You additionally need to use `LA.IMPORT.WRAP` or `LA.IMPORT.GUESSWRAP` if the following conditions apply:

- The on-chip trace buffer was used as a circular ring buffer.
- The on-chip trace data was stored as is, it was not read out in the timely order starting from the write pointer position.

`LA.Mode FlowTrace` will automatically be set when using this command.

**See also**

- `<trace>.IMPORT`
- ‘Release Information’ in ‘Release History’

**<trace>.IMPORT.flow**

**Import bus trace data**

Re-imports a file that has been exported with `<trace>.EXPORT.flow`. This bus trace data comes from capturing the fetched instructions and data accesses done on an external bus to figure out the program behavior. It works only if no cache is used and if the bus accesses can be captured. Nowadays this method is rarely used.

**See also**

- `<trace>.IMPORT`
Guess wrap pointer

Format: \texttt{LA.IMPORT.GUESSWRAP \texttt{[\textit{record\_number}]}}

Reformats external trace data loaded to TRACE32 in a timely order. The external trace data of a circular ring buffer is loaded to TRACE32 using \texttt{LA.IMPORT.ETB}. The command \texttt{LA.IMPORT.GUESSWRAP} scans the loaded trace data and guesses where the wrap pointer might have been.

Optionally, you can pass a record number where the search for the wrap pointer shall start. Without a parameter it starts from the beginning.

Use \texttt{LA.IMPORT.WRAP} if you know where the wrap pointer is.

See the figures in the introduction to \texttt{<trace>.IMPORT}.

See also

- \texttt{<trace>.IMPORT}
- \texttt{<trace>.IMPORT.WRAP}

Import STP recording from file (nibble)

Format: \texttt{LA.IMPORT.STP \texttt{<file>}}

Imports an STP trace from \texttt{<file>} to process it within TRACE32. One trace record is generated per nibble.

In order to unwrap the trace information for processing, TRACE32 needs to know the following information: STM base address and the STP protocol version.

If TRACE32 is aware of the chip characteristic, setting up the chip is sufficient.

Example:

\begin{verbatim}
SYStem.CPU OMAP4430APP1

LA.IMPORT.STP my_recording.stp

STMLA.List
\end{verbatim}
Otherwise the following setup has to be done.

```plaintext
SYStem.CONFIG.STM.Base DAP:0xd4161000  ; any base address != 0x0 is fine
SYStem.CONFIG.STM.Mode STPv2             ; specify the STP protocol version
LA.IMPORT.STP my_recording.stp
STMLA.List
```

See also
- `<trace>.IMPORT`

### `<trace>.IMPORT.STP ByteString`

**Import STP recording from file (byte)**

**Format:**

```
LA.IMPORT.STPByteStream <file>
```

Same as `LA.IMPORT.STP`, but one trace record is generated per byte.

See also
- `<trace>.IMPORT`
  - 'Release Information' in 'Release History'

### `<trace>.IMPORT.TraceFile`

**Import trace data where processing has failed**

**Format:**

```
LA.IMPORT.TraceFile <file>
```

Re-imports trace data stored by `<trace>.SAVE` for re-processing. This is useful if processing was not possible when the trace recording was made. For example if you had no access to the target code at that moment.

Only the trace raw data will be extracted from the saved (*.ad) file.

`LA.Mode FlowTrace` will automatically be set when using this command.

See also
- `<trace>.IMPORT`
  - 'Release Information' in 'Release History'

©1989-2019 Lauterbach GmbH

General Commands Reference Guide T 215
<trace>.IMPORT.TracePort

Import off-chip trace data

| Format: | LA.IMPORT.TracePort <file> |

Imports a pure binary trace data file from an external trace port like an ARM CoreSight TPIU. Unlike on-chip trace data, off-chip trace data includes synchronization packages and depend on the port size of the trace port.

**LA.Mode FlowTrace** will automatically be set when using this command.

**See also**
- <trace>.IMPORT
- ‘Release Information’ in ‘Release History’

<trace>.IMPORT.VCD

Import recorded signals in VCD file format

| Format: | LA.IMPORT.VCD <file> |

Imports a VCD (Value Change Dump) file, which is an industrial standard format for waveforms (not for program trace). It is used for visualizing and analyzing the captured signals in the <trace>.Timing window.

**See also**
- <trace>.IMPORT
<trace>.IMPORT.WRAP

Define wrap pointer

<table>
<thead>
<tr>
<th>Format:</th>
<th>LA.IMPORT.WRAP &lt;record_number&gt;</th>
</tr>
</thead>
</table>

Reformats external trace data loaded to TRACE32 in a timely order. The external trace data of a circular ring buffer is loaded to TRACE32 using LA.IMPORT.ETB.

<record_number> You pass the <record_number> of the first trace record in time (wrap pointer). This is the write pointer location of a circular ring buffer the moment the data has been stored.

**NOTE:** On a CoreSight trace, the write pointer points to a 32-bit value. You need to multiply this value by 4 because each CoreSight trace record is 8 bit in size.

Use LA.IMPORT.GUESSWRAP if you do not know where the wrap pointer is.

See the figures in the introduction to <trace>.IMPORT.

See also

- <trace>.IMPORT
- <trace>.IMPORT.GUESSWRAP
**<trace>.InChannel**

Inchannel state display

Format: `<trace>.InChannel`

Show the state of the input channel.

<table>
<thead>
<tr>
<th>state</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>DISableChannel</td>
<td></td>
</tr>
<tr>
<td>ENableChannel</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>hostfifo</th>
<th>used</th>
<th>done</th>
</tr>
</thead>
</table>

**<trace>.Init**

Initialize trace

Format: `<trace>.Init`

The contents of the trace memory/streaming file is erased. All user setups, like the trace mode or trace memory size, remain unchanged.

If the chip includes an onchip trigger unit, counters and trigger levels are cleared. The detailed behavior strongly depends on the onchip trigger unit.

The trace is in OFF state, after a Trace.Init was executed.

**TRACE32-ICE and TRACE32-FIRE**

The contents of the trace memory is erased. The trigger unit is returned to its initial settings (the counters, the flags and the trigger levels are cleared). All user setups, like the trace mode or trace memory size, remain unchanged.

See also

- CIProbe
- `<trace>.Arm`
- `<trace>.AutolInit`
- `<trace>.state`
- Trace
  - 'Emulator Functions' in 'FIRE User's Guide'
  - 'Release Information' in 'Release History'

©1989-2019 Lauterbach GmbH
Sampling configuration for probes JKLMNO

PowerIntegrator only

Format: \texttt{Integrator.JKLMNO} \texttt{<option>}

\texttt{<option>}: \texttt{250MHZ}
\texttt{State}
\texttt{StatePLL}
\texttt{CLKJ}
\texttt{CLKK}
\texttt{Falling}
\texttt{Rising}
\texttt{SAMPLE}

\textbf{250MHZ} \quad \text{Timing Mode 250 MHz.}
\textbf{State} \quad \text{State Mode, clocked by CLKJ or CLKK.}
\textbf{StatePLL} \quad \text{State PLL Mode, clocked by CLKJ or CLKK.}
\textbf{CLKJ} \quad \text{Clock J select for State-Mode or State-PLL-Mode.}
\textbf{CLKK} \quad \text{Clock K select for State-Mode or State-PLL-Mode.}
\textbf{Falling} \quad \text{sampling on falling edge of selected clock CLKJ or CLKK.}
\textbf{Rising} \quad \text{sampling on rising edge of selected clock CLKJ or CLKK.}
\textbf{SAMPLE} \quad \text{sampling delay of selected clock CLKJ or CLKK (-3 \ldots +6 \text{ ns in steps of 250 ps}), State-PLL-Mode only.}

\texttt{<trace>.JOINFILE} \quad \textbf{Concatenate several trace recordings}

Format: \texttt{<trace>.JOINFILE} \texttt{<file>} \texttt{[<records>] \texttt{[/<option>]}}

\texttt{<records>}: \texttt{<string> | <range> | <value> | <timerange>}

\texttt{<option>}: \texttt{ZIP | NoCompress | Compress | TIMEGAP}

Concatenates several trace recordings to increase the volume of trace information to be analyzed.
The reference point is automatically set to the start of the last added trace recording.

Time gaps between the trace recording result in a large **Time.Back** time (see screenshot above). The option **TIMEGAP <time>** allows a seamless concatenation with regards to the time stamp.

```
Trace.SAVE my_joinfile.ad ; save current trace contents to file
Trace.FILE my_joinfile.ad ; load trace contents from file
... ; run program to fill the trace
Trace.JOINFILE my_joinfile /TIMEGAP 0.1us ; append current trace contents to loaded trace contents
... ; run program to fill the trace
Trace.JOINFILE my_joinfile /TIMEGAP 0.1us ; append current trace contents
Trace.Chart.sYmbol /FILE ; display timing for concatenated trace
...
Trace.SAVE ; close loaded trace file
```

```
; use record numbers to specify the trace recording to be added
Trace.JOINFILE my_joinfile (4665.)--(5168.)

; use bookmarks to specify record range
Trace.JOINFILE my_joinfile "start" "end" /TIMEGAP 0.1us
```
ICE only

Format: \texttt{<trace>.LEVEL}

Only valid for Ha120 and SA120.

Selects the level of the analyzer trigger unit manually.
**<trace>.List**

List trace contents

| Format: | `<trace>.List` [ `<record> | <record_range>` ] [ `<items>` … ] [ `/<options>` ] |
|---------|-------------------------------------------------|
| `<option>`: | FILE  
Track  
FlowTrace | BusTrace  
NorthWestGravity  
CORE `<number>` | SplitCORE  
Mark `<item>`  
Raw  
TimeZero  |
| `<items>`: | `%<format>`  
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>`] | FLAG[, `<flagindex>`]  
Trigger | Trigger.A | Trigger.B  
SPARE  
`<special_lines>`  |
| `<format>`: | Ascii | BIrNary | Decimal | Hex | Signed | UnsIGNED  
HighLow | Timing  
TimeAuto | TimeFixed  
LEN `<size>` |

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

The columns of the `<trace>.List` window can be defined using the `<items>`. The order of the columns in the window is according to the order of the `<item>` parameters given (with a few exceptions like the run column that always appears at the very left).
Note that the default columns are hidden, when you manually specify the columns you want to display. The default columns can be included again in the user-defined column display using the option DEFault.

**Example:**

```
Trace.List List.address DEFault
```

For details on the available columns, see further down.

For trace modes other than RTS, the trace contents can only be displayed if the trace is in **OFF** or **break** state. Please refer to the `<trace>.state` command for more information.

For target-specific information and options see:

- **Processor Architecture Manuals**
- **Target Guides FIRE**
- **Target Guides ICE**

### Description of Buttons in the `<trace>.List` Window

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup</td>
<td>Open a <code>&lt;trace&gt;.state</code> window, to configure the trace.</td>
</tr>
<tr>
<td>Goto</td>
<td>Open a <code>&lt;trace&gt;.GOTO</code> dialog box, to move the cursor to a specific record.</td>
</tr>
<tr>
<td>Find</td>
<td>Open a <code>&lt;trace&gt;.Find</code> dialog box, to search for specific entries in the trace.</td>
</tr>
<tr>
<td>Chart</td>
<td>Display the program execution time at different symbols as a time chart. See the <code>&lt;trace&gt;.Chart.sYmbol</code> command.</td>
</tr>
<tr>
<td>Profile</td>
<td>Open a <code>&lt;trace&gt;.PROfileChart.sYmbol</code> window.</td>
</tr>
<tr>
<td>MIPS</td>
<td>Open a MIPS.PROfileChart.sYmbol window.</td>
</tr>
<tr>
<td>More/Less</td>
<td>Switch step-by-step from full display (all CPU cycles including dummies) to HLL display and vise versa.</td>
</tr>
</tbody>
</table>

If no parameters are specified, a predefined set of items will appear in the window. By selecting items, specific items can be displayed in any order defined by the user. It is possible to remove a selection from the list by appending the keyword.**OFF**. The display format of the entries can be changed by the `%<format>` options.
## Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FILE</strong></td>
<td>Displays trace memory contents loaded with <code>Trace.FILE</code>.</td>
</tr>
<tr>
<td><strong>FlowTrace</strong></td>
<td>The trace works as a program flow trace. This option is usually not required.</td>
</tr>
<tr>
<td><strong>BusTrace</strong></td>
<td>The trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td><strong>Track</strong></td>
<td>Track the <code>&lt;trace&gt;.List</code> window with other trace list windows (tracking to record number or time possible).</td>
</tr>
<tr>
<td><strong>Mark <code>&lt;item&gt;</code></strong></td>
<td>Bold print all cycles on a yellow background which contain the specified item.</td>
</tr>
<tr>
<td><strong>NorthWestGravity</strong></td>
<td>With <strong>NorthWestGravity</strong>: The record numbering in the top left corner stays fixed as you resize the <code>&lt;trace&gt;.List</code> window. Without <strong>NorthWestGravity</strong>: The record numbering scrolls as you resize the window.</td>
</tr>
<tr>
<td><strong>Raw</strong></td>
<td>Displays all channels as raw hexadecimal values (where applicable)</td>
</tr>
<tr>
<td><strong>TimeZero</strong></td>
<td>Use timestamp of first entry in listing as global reference (item <code>Time.Zero</code>).</td>
</tr>
</tbody>
</table>

```plaintext
Trace.FILE test1 ; load trace file
Trace.List /File ; display trace listing, source for the ; trace data is the loaded file
Trace.List /Mark Address sieve ; mark all trace lines which contain the ; address sieve
```

In the case of an SMP system, the following options are provided:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SplitCORE</strong></td>
<td>Displays the trace recording of all cores side by side.</td>
</tr>
<tr>
<td><strong>CORE <code>&lt;number&gt;</code></strong></td>
<td>Displays the trace recording of the specified core.</td>
</tr>
</tbody>
</table>
**Formats**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ascii</td>
<td>Displays single bytes as ascii characters</td>
</tr>
<tr>
<td>Binary</td>
<td>Displays single bytes in binary values</td>
</tr>
<tr>
<td>Decimal</td>
<td>Displays single bytes in decimal values</td>
</tr>
<tr>
<td>Hex</td>
<td>Displays single bytes in hex values</td>
</tr>
<tr>
<td>HighLow</td>
<td>Displays single bits as 'H' or 'L' character</td>
</tr>
<tr>
<td>LEN &lt;size&gt;</td>
<td>Specifies the width of non numeric fields (e.g. symbols)</td>
</tr>
<tr>
<td>Signed</td>
<td>Displays single bytes signed</td>
</tr>
<tr>
<td>TimeAuto</td>
<td>Displays time values in a floating display format (short)</td>
</tr>
<tr>
<td>TimeFixed</td>
<td>Displays time values in a fixed point format (long format)</td>
</tr>
<tr>
<td>Timing</td>
<td>Displays single bits as vertical timing</td>
</tr>
<tr>
<td>Unsigned</td>
<td>Displays single bytes unsigned</td>
</tr>
</tbody>
</table>

**Examples:**

```plaintext
; display trace listing, limit the symbol names to 20 characters
Trace.List Address CYcle Data.L %LEN 20. sYmbol TIme.Back

; display trace listing, show the external trigger input 0 as vertical timing
Trace.List %TIMING T.0 DEFault
```

**The following <items> define the columns shown in the <trace>.List windows**

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFault</td>
<td>Default trace display. The default trace display can be configured with the command <strong>SETUP.ALIST</strong>.</td>
</tr>
<tr>
<td>ALL</td>
<td>Select all available channels (superset of DEFault)</td>
</tr>
<tr>
<td>CPU</td>
<td>Set of channels describing the CPU state (similar to the original setting of DEFault but no source code display).</td>
</tr>
<tr>
<td>LINE</td>
<td>Set of channels which contains all CPU control lines.</td>
</tr>
</tbody>
</table>
**Run**

Gives various information about the execution of the current record.

- **GO**: the first instruction that was executed by the CPU after starting program execution with **Go**.
  (TRACE32-ICE only, sampling of the first instruction can be switched off by **Analyzer.Mode PREPOST OFF**)
- **BRK**: Indicates that the program execution was stopped.
- **T**: Indicates a trigger event.
- **f**: Foreground program
- **b**: Background program
- **ft**: Trigger event occurred in the foreground program
- **bt**: Trigger event occurred in the background program
- **0,1,2,3 ...** in SMP systems, the run column indicates the number of the core that executed the given code; additionally, the background color of the records changes to highlight the relevant core (light red, light green, ...).

<table>
<thead>
<tr>
<th><strong>Address</strong></th>
<th>start address of each displayed <strong>block of executed opcodes</strong>; for displaying the address of each single opcode, use the channel <strong>List.Address</strong>.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>sYmbol</strong></td>
<td>Symbolic address with path and offset (as find item will search on all processor busses)</td>
</tr>
<tr>
<td><strong>sYmbolN</strong></td>
<td>Symbolic address without path but with offset</td>
</tr>
<tr>
<td><strong>sYmbolinline</strong></td>
<td>Inline symbol name with path.</td>
</tr>
<tr>
<td><strong>sYmbolinlineN</strong></td>
<td>Inline symbol name without path.</td>
</tr>
<tr>
<td><strong>AAddress</strong></td>
<td>Physical (absolute) CPU address</td>
</tr>
<tr>
<td><strong>AAddress.0--31</strong></td>
<td>Physical address bits A0..A31</td>
</tr>
<tr>
<td><strong>PAddress</strong></td>
<td>This column display the address of the instruction that was executed before a read or write access was performed (B).</td>
</tr>
<tr>
<td></td>
<td>Prestore address (E), refer to <strong>Analyzer.Mode Prestore</strong> for more information. Poststore address (F), refer to <strong>Analyzer.Mode Poststore</strong> for more information.</td>
</tr>
<tr>
<td><strong>PsYmbol</strong></td>
<td>This column display the address of the instruction that was executed before a read or write access was performed (B). Symbolic prestore address with path and offset (E), symbolic poststore address with path and offset (F).</td>
</tr>
<tr>
<td><strong>FAddress</strong></td>
<td>Flowtrace execution address (when flowtrace available)</td>
</tr>
<tr>
<td><strong>FsYmbol</strong></td>
<td>Symbolic flowtrace execution address</td>
</tr>
<tr>
<td><strong>BAddress</strong></td>
<td>Bus address, same like physical address, but also displayed when the bus is not transferring data</td>
</tr>
<tr>
<td><strong>Var</strong></td>
<td>Symbolic display of data accesses to HLL variables</td>
</tr>
<tr>
<td><strong>CYcle</strong></td>
<td>Bus cycle</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>CPU data full width</td>
</tr>
<tr>
<td><strong>Data.B</strong></td>
<td>CPU data single byte</td>
</tr>
<tr>
<td><strong>Data.B0</strong></td>
<td>CPU data lower byte</td>
</tr>
<tr>
<td><strong>Data.W0</strong></td>
<td>CPU data lower word</td>
</tr>
<tr>
<td><strong>Data.T0</strong></td>
<td>CPU data lower triple</td>
</tr>
<tr>
<td><strong>Data.0..31</strong></td>
<td>CPU data bit 0 to 31</td>
</tr>
<tr>
<td><strong>Data.0--7</strong></td>
<td>CPU data bits 0 to 7 as single bits (8 bit processor)</td>
</tr>
<tr>
<td><strong>Data.0--15</strong></td>
<td>CPU data bits 0 to 15 as single bits (16 bit processor)</td>
</tr>
<tr>
<td><strong>Data.0--31</strong></td>
<td>CPU data bits 0 to 31 as single bits (32 bit processor)</td>
</tr>
<tr>
<td><strong>Data.sYmbol</strong></td>
<td>Display the data value symbolically</td>
</tr>
<tr>
<td><strong>BData</strong></td>
<td>Like Data, but always displays the data even when the bus is idle</td>
</tr>
<tr>
<td><strong>List.Address</strong></td>
<td>Lists the address for each individual opcode (instead of the start address of blocks of executed opcodes)</td>
</tr>
<tr>
<td><strong>List.Asm</strong></td>
<td>Disassembled mnemonics</td>
</tr>
<tr>
<td><strong>List.Mix</strong></td>
<td>Disassembled mnemonics and HLL source</td>
</tr>
<tr>
<td><strong>List.Hll</strong></td>
<td>HLL source only, dequeueing based on disassembler</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>List.HIIOnly</td>
<td>HLL source only no dequeueing</td>
</tr>
<tr>
<td>List.NoFetch</td>
<td>Suppresses the display of op-fetches</td>
</tr>
<tr>
<td>List.NoPFetch</td>
<td>Suppresses the display of prefetch cycles</td>
</tr>
<tr>
<td>List.NoCycle</td>
<td>Suppresses the display of more than one cycle between lines</td>
</tr>
<tr>
<td>List.Label</td>
<td>Label of disassembled mnemonic</td>
</tr>
<tr>
<td>List.Comment</td>
<td>Comments to disassembled mnemonics</td>
</tr>
<tr>
<td>List.Queue</td>
<td>Start address of disassembled mnemonic</td>
</tr>
<tr>
<td>List.TASK</td>
<td>Displays OS Awareness information (system-calls etc.)</td>
</tr>
<tr>
<td>List.Reorder</td>
<td>Reorders bus cycles logically (only some processors)</td>
</tr>
<tr>
<td>List.NoDummy</td>
<td>Suppresses the display of dummy cycles (where applicable)</td>
</tr>
<tr>
<td>List.Bondout</td>
<td>Display internal bondout information (where applicable)</td>
</tr>
<tr>
<td>List.TTime</td>
<td>Display time information in assembler or HLL lines</td>
</tr>
<tr>
<td>List.CTS</td>
<td>Display CTS information (Context Tracking System)</td>
</tr>
<tr>
<td>List.SOURCE-FILE</td>
<td>Display source file name for each line</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>Time marker (default Time.Fore)</td>
</tr>
<tr>
<td>Time.Fore</td>
<td>Time marker, relative time to next record</td>
</tr>
<tr>
<td>Time.Back</td>
<td>Time marker, relative time to previous record</td>
</tr>
<tr>
<td>Time.Zero</td>
<td>Time marker, relative to global reference</td>
</tr>
<tr>
<td>Time.REF</td>
<td>Time marker, relative to reference point</td>
</tr>
<tr>
<td>Time.Trigger</td>
<td>Time marker, relative to trigger point</td>
</tr>
<tr>
<td>Time.FUNC</td>
<td>Time spent in a function (*1)</td>
</tr>
<tr>
<td>Time.FUNCEX</td>
<td>Time spent in calls (*1)</td>
</tr>
<tr>
<td>Time.FUNCIN</td>
<td>Time spent in code of function (*1)</td>
</tr>
<tr>
<td>Time.MARKAB</td>
<td>Time relative back to the last marker A</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Time.MARKAF</td>
<td>Time relative forward to the next marker A</td>
</tr>
<tr>
<td>Time.MARKBB</td>
<td>Time relative back to the last marker B</td>
</tr>
<tr>
<td>Time.MARKBF</td>
<td>Time relative forward to the next marker B</td>
</tr>
<tr>
<td>Time.MARKCB</td>
<td>Time relative back to the last marker C</td>
</tr>
<tr>
<td>Time.MARKCF</td>
<td>Time relative forward to the next marker C</td>
</tr>
<tr>
<td>Time.MARKDB</td>
<td>Time relative back to the last marker D</td>
</tr>
<tr>
<td>Time.MARKDF</td>
<td>Time relative forward to the next marker D</td>
</tr>
<tr>
<td>CLOCKS.Back</td>
<td>Number of clocks relative time to previous record</td>
</tr>
<tr>
<td>CLOCKS.Fore</td>
<td>Number of clocks relative time to next record</td>
</tr>
<tr>
<td>CLOCKS.Trigger</td>
<td>Number of clocks relative to trigger point</td>
</tr>
<tr>
<td>CLOCKS.REF</td>
<td>Number of clocks relative to reference point</td>
</tr>
<tr>
<td>CLOCKS.Zero</td>
<td>Number of clocks relative to global zero point</td>
</tr>
<tr>
<td>FUNC</td>
<td>Function nesting display (*1)</td>
</tr>
<tr>
<td>FUNCVar</td>
<td>Function nesting plus variables</td>
</tr>
<tr>
<td>FUNCR</td>
<td>Record number associated with this entry/exit point (*1)</td>
</tr>
<tr>
<td>IGNORE</td>
<td>Record ignored or used for performance/nesting analysis</td>
</tr>
<tr>
<td>LeVel</td>
<td>Trigger unit logical level</td>
</tr>
<tr>
<td>MARK.all</td>
<td>Display markers</td>
</tr>
<tr>
<td>MARK.A</td>
<td>Display marker A</td>
</tr>
<tr>
<td>FLAG.all</td>
<td>Flags of the trigger unit in a short form</td>
</tr>
<tr>
<td>FLAG.0</td>
<td>Flag 0 of the trigger unit</td>
</tr>
<tr>
<td>Trigger.A</td>
<td>External trigger input A (E)</td>
</tr>
<tr>
<td></td>
<td>External trigger input EXTA (bit 0) and EXTB (bit 1) (F)</td>
</tr>
</tbody>
</table>

©1989-2019 Lauterbach GmbH
<table>
<thead>
<tr>
<th>Trigger.B</th>
<th>External trigger input B (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trigger.0</td>
<td>External trigger bit 0</td>
</tr>
<tr>
<td>Trigger.0--7</td>
<td>External trigger input bit 0--7</td>
</tr>
<tr>
<td>SPARE</td>
<td>Displays an empty block</td>
</tr>
<tr>
<td>VarsSymbol</td>
<td>HLL display of accesses to variables including bitfields and symbols.</td>
</tr>
</tbody>
</table>

(1): The trace must be the same as for the command `<trace>.STATistic.Func`. The combination of the **FUNC** keyword with the **List.TASK** keyword makes the function nesting display task sensitive.

**FLOW ERROR Diagnosis**

FLOWERROR | Display flow error column

**Flow Trace Decoding**

FAddress | To decompress the recorded trace information the program code starting at FAddress is read.

FsYmbol | Symbolic address of FAddress.
FCOUNT
To decompress the recorded trace information FCOUNT number of byte is read.

FLen
(deprecated)

Trace Raw Data and Packet Decoding

<table>
<thead>
<tr>
<th>TP</th>
<th>All raw trace data as recorded at the trace port. For multicore systems the stream of trace data may contain information for multiple cores (e.g. in “wrapped mode” for CoreSight systems).</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPC</td>
<td>Raw trace data pertaining to a single core. For multicore systems this data is extracted from the overall trace data stream.</td>
</tr>
<tr>
<td>TPINFO</td>
<td>Information obtained by decoding a single trace packet (which may consist of multiple bytes).</td>
</tr>
<tr>
<td>TSINFO</td>
<td>Timestamp calculation background information, required for the diagnosis of complex timing scenarios.</td>
</tr>
</tbody>
</table>
Context ID/Ownership Trace Packet Decoding

If a Context ID or ownership packet is decoded and if it is assignable to a task, the “task” cycle type and the task name is displayed. The displayed data value is a TRACE32 internal value.

If a Context ID or ownership packet is decoded and if it can not be assigned to a task or any other protocol-specific content such as service, intr etc. the cycle type “traceid” and the packet content is displayed.

If the machine ID is encoded, the machine name is also displayed (“sender” in the screenshot below).

See also
- <trace>.BookMark
- <trace>.Timing
- Trace
  - Analyzer.RECORD.ADDRESS()
  - Analyzer.RECORD.OFFSET()
  - Analyzer.RECORD.DATA()
  - Analyzer.RECORDS()
- <trace>.State
- <trace>.View
- CIProbe
  - Analyzer.RECORD.OFFSET()
▲ ‘How to use the PROTOanalyzer’ in ‘DigRF Protocol Analyzer’
▲ ‘How to use the PROTOanalyzer’ in ‘FlexRay Protocol Analyzer’
▲ ‘How to use the PROTOanalyzer’ in ‘LIN Bus Protocol Analyzer’
▲ ‘Release Information’ in ‘Release History’
▲ ‘Displaying the Trace’ in ‘Training FIRE Analyzer’
▲ ‘Emulator Functions’ in ‘FIRE User’s Guide’
The command **Trace.ListNesting** is mainly used to investigate issues in the construction of the call tree for the nesting function run-time analysis. Typical commands for the nesting function run-time analysis are the commands **Trace.STATistic(Func** or **Trace.STATistic.TREE**.

<table>
<thead>
<tr>
<th>&lt;option&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE &lt;number&gt;</td>
<td>SMP tracing only. Filters the <strong>Trace.ListNesting</strong> window by the specified core. Processing is done for all cores, but only the specified core is displayed. All other cores are temporarily hidden in the window.</td>
</tr>
<tr>
<td>SplitCORE</td>
<td>SMP tracing only. Displays the trace recording of the cores side by side in the <strong>Trace.ListNesting</strong> window.</td>
</tr>
</tbody>
</table>

The **Trace.ListNesting** window provides the nesting details. If a function entry point is selected, the path to the function exit is highlighted.

If the function exit is located far apart, you can use the Down Arrow (v) to jump to the function exit.
The interrupt nesting if marked specially (see screenshot below).

Code optimizations are the main reason for issues in the construction of the call tree. TRACE32 indicates these issues as PROBLEMS or WORKAROUNDS.

**PROBLEMS**

- A PROBLEM is a point in the trace recording that TRACE32 can not integrate into the current nesting.
- PROBLEMS are marked with (!) in the **Trace.ListNesting** window. The name of the expected function is shown.
- PROBLEMS are ignored in the construction of the call tree.
- PROBLEMS may affect the construction of the call tree, so it is important to inspect them. The **Statistic Markers** can be used to solve a PROBLEM.

To inspect a PROBLEM, proceed as follows:

1. Go to the start of the trace recording.
2. Use the **Find…** command from the **Edit** menu. Type (!) as find item.
3. Open a Trace Listing to inspect the problem in detail.
WORKAROUND

- A WORKAROUND is a point in the trace recording that TRACE32 can not integrate into the current nesting.
- TRACE32 attempts to integrate this point into the function nesting, by deriving information from previous scenarios in the nesting.
- WORKAROUNDS are marked with (?) in the Trace.ListNesting window.
- WORKAROUNDS may affect the construction of the call tree, if the derived information is wrong. It is recommended to inspect the WORKAROUNDS.

To inspect a WORKAROUND, proceed as follows:

1. Go to the start of the trace recording.
2. Use the Find... command from the Edit menu. Type (?) as find item.
3. Open a Trace Listing to inspect the problem in detail.

See also
- <trace>.state
- Trace
- ‘Release Information’ in ‘Release History’
### <trace>.ListVar

List variable recorded to trace

<table>
<thead>
<tr>
<th>Format 1:</th>
<th><code>&lt;trace&gt;.ListVar [%&lt;format&gt;] [{&lt;var&gt;}][[{&lt;options&gt; }]]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Format 2:</td>
<td><code>&lt;trace&gt;.ListVar &lt;range&gt; [%&lt;format&gt;] [{&lt;var&gt;}]</code></td>
</tr>
<tr>
<td><code>&lt;format&gt;</code>:</td>
<td>DEFault</td>
</tr>
<tr>
<td><code>&lt;range&gt;</code>:</td>
<td><code>&lt;record_range&gt;</code></td>
</tr>
<tr>
<td><code>&lt;options&gt;</code>:</td>
<td>Split</td>
</tr>
</tbody>
</table>

Displays a list of all variable recorded if it is used without parameters.

The option **Mark** allows to mark the specified variable access.

```plaintext
; mark trace entry when a 0x0 is written to variable mstatic1
Trace.ListVar /Mark Address Var.RANGE(mstatic1) CYcle Write Data 0x0
```
Format 1 represents the standard syntax, in which the variable names follow the %<format> parameter. The following options provide a representation in which variable values can be better compared:

<table>
<thead>
<tr>
<th>Split</th>
<th>Each specified variable gets its own column. If a variable is accessed its value is displayed. Write accesses are printed in black, read accesses are printed in gray.</th>
</tr>
</thead>
<tbody>
<tr>
<td>SplitFill</td>
<td>Each specified variable gets its own column. Whenever one of the specified variables is displayed, the current values of all other specified variables are displayed as well. Write accesses are printed in black, everything else is printed in gray.</td>
</tr>
</tbody>
</table>

Examples for Format 1:

```c
//Display all accesses to the variable mstatic1
Trace.ListVar %DEFault mstatic1

//Display all accesses to the listed variables
Trace.ListVar %DEFault mstatic1 fstatic fstatic2

//Display all accesses to the listed variables, but display the values of each variable in a separate column
Trace.ListVar %DEFault mstatic1 fstatic fstatic2 /Split
Trace.ListVar %Hex mstatic1 fstatic fstatic2 /Split

//Display all accesses to the listed variables, but display the values of each variable in a separate column
//fill in the current value of the not accessed variable to each line
Trace.ListVar %DEFault mstatic1 fstatic vlong /SplitFill
```
**Format 2** represents the advanced syntax, here it is possible to restrict the display to the specified `<record_range>` or `<time_range>`.

**Examples for Format 2:**

```
Trace.ListVar (-14874903.)--(-14874761.)
Trace.ListVar 1.8s--10.8s
Trace.ListVar (-14874903.)--(-14874761.) vfloat
Trace.ListVar 1.8s--10.8s mstatic1 fstatic fstatic2 /SplitFill
```

**See also**
- `<trace>.state`
- `Trace`

▲ 'Release Information' in 'Release History'
Load trace file for offline processing

Format:  \texttt{<trace>.\textsc{load} [<file>] [/Config]}

Loads trace data from a file into the debugger. Typically \texttt{<trace>.\textsc{load}} is used to analyze data in a simulator or to compare different recordings.

The command loads the data into the ”normal” trace buffer i.e. the same buffer that is filled when recording data using an analyzer (e.g. via PowerTrace, PowerProbe, PowerIntegrator etc.). As the standard trace commands work on this buffer, they automatically work on the loaded data. To highlight that loaded data is displayed, windows are marked by a red label \texttt{load} label in the bottom-left corner.

To save trace data, use the command \texttt{<trace>.\textsc{save}}.

The default extension for the file name is \texttt{*ad}.

\textbf{NOTE:} There is a similar but slightly different command \texttt{<trace>.\textsc{file}}. It loads the trace data into a dedicated \textit{file trace buffer}. To have trace commands (e.g. \texttt{Trace.List}) work on the \textit{file trace buffer}, they need to be invoked with the parameter \texttt{/FILE}.

An example for working on loaded trace data:

\begin{verbatim}
Trace.LOAD test4 ; load trace contents from file
Data.LOAD.Elf demo.elf /NoCODE ; load symbol information for the ; post-processing
Trace.List ; display loaded trace contents
Trace.Chart.sYmbol ; symbol analysis of trace
Trace.STATistic.Func ; function run-time analysis
\end{verbatim}
The TRACE display and analysis commands are re-directed to the selected \textit{trace method} if:

- \textbf{Trace.LOAD} is executed without the parameter $<$\textit{file}$>$.

\begin{verbatim}
Trace.LOAD ; Re-direct trace display and ; analysis commands to the selected ; trace method
\end{verbatim}

- A trace configuration command is executed.

\begin{verbatim}
Trace.Init ; the trace configuration command ; \textbf{Trace.Init} re-directs the trace ; display and analysis commands to ; the selected trace method
\end{verbatim}

- The program execution is started while \textbf{Trace.AutoArm} is set to ON.

If the \textbf{Trace.METHOD Probe} or \textbf{Trace.METHOD Integrator} was selected, when the trace contents were saved, the option \textit{/Config} can be used to re-activate the \textit{Probe/Integrator} and \textit{NAME} settings.

\begin{verbatim}
Trace.METHOD Probe ; select the trace method Probe ; for the PowerProbe ...
Trace.SAVE probetest1 ; save the trace contents to the ; file probetest1 ...
QUIT ; end TRACE32
\end{verbatim}

\begin{verbatim}
Trace.LOAD probetest1 /Config ; load the trace contents from the ; file probetest1 ; load the Probe settings and NAMEs
Trace.List
\end{verbatim}

\textbf{See also}
- $<$trace$>$.\texttt{FILE}
- $<$trace$>$.\texttt{SAVE}
- $<$trace$>$.\texttt{state}
- Trace
- `Emulator Functions` in `FIRE User’s Guide`
- `Release Information` in `Release History`
Combine two trace files into one. This is useful for traces recorded for different cores working in AMP mode.

**Format:**  
<trace>.MERGEFILE <file> [<trace_area>] [/<option> ...]

**<trace_area>:**  
<string>  
<range>  
<value>  
<timerange>

**<option>:**  
ZIP  
NoCompress  
QuickCompress  
Compress  
TIMEGAP <time>

ZIP File is compressed with the gzip archive format.

NoCompress Obsolete.

QuickCompress Obsolete.

Compress Obsolete.

TIMEGAP <time> Allows a seamless concatenation with regards to the timestamp

**NOTE:** The ZIP or Compress options have meanwhile become obsolete because the resulting file is compressed by default.

**See also**
- Trace
<trace>.METHOD

Select trace method

Format: Trace.METHOD <method>

<method>:
- Analyzer
- ART
- CAnalyzer
- FDX
- HAnalyzer
- Integrator
- IProbe
- LA
- LOGGER
- Onchip
- Port
- PROBE
- SNOOPer

Selects the trace method that you want to use.

<table>
<thead>
<tr>
<th>Trace Methods</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analyzer</td>
<td>Trace RAM is provided by a TRACE32 tool (e.g. PowerTrace, ICE, FIRE), but not CombiProbe.</td>
</tr>
<tr>
<td>ART</td>
<td>Advanced Register Trace. For more information refer to “ART - Advanced Register Trace” (debugger_user.pdf).</td>
</tr>
<tr>
<td>CAnalyzer</td>
<td>Trace RAM is provided by the CombiProbe.</td>
</tr>
<tr>
<td>FDX</td>
<td>Fast Data eXchange. The target application needs to write the required trace information to a small ring buffer (min. size 2 trace records). The contents of the ring buffer is transferred to the TRACE32 software while the program execution is running and saved there for later display. If the on-chip debug unit provides a Debug Communications Channel (DCC) the required trace information can be transferred directly to the TRACE32 software.</td>
</tr>
<tr>
<td>HAnalyzer</td>
<td>Trace RAM is provided by the host. This method is used for targets that provide a specifically implemented trace channel over interfaces like USB3.</td>
</tr>
<tr>
<td>Integrator</td>
<td>The Lauterbach logic analyzer PowerIntegrator is used to record the trace information.</td>
</tr>
<tr>
<td>IProbe</td>
<td>The Lauterbach IProbe logic analyzer within the Power Trace II is used to record signals.</td>
</tr>
<tr>
<td>Trace Methods</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>LA</td>
<td>LA (Logic Analyzer). Trace information not recorded by TRACE32 can be loaded and processed. This requires that the TRACE32 software is familiar with the format of the trace information.</td>
</tr>
<tr>
<td>LOGGER</td>
<td>The target application can write the required trace information to target RAM. TRACE32 loads the trace information from the target RAM for display and processing.</td>
</tr>
<tr>
<td>MergedAnalyzer</td>
<td>The trace RAM is provided by the PowerTrace. The trace information of all cores in an SMP system is displayed and analyzed en bloc. A different use for some processor architectures.</td>
</tr>
<tr>
<td>Onchip</td>
<td>The trace information is saved in a trace RAM provided by the chip.</td>
</tr>
<tr>
<td>PORT</td>
<td>Port Analyzer.</td>
</tr>
<tr>
<td>Probe</td>
<td>The Lauterbach logic analyzer PowerProbe is used to record the trace information.</td>
</tr>
<tr>
<td>SNOOPer</td>
<td>Snooper. For details, see “Application Note for the SNOOPer Trace” (app_snooper.pdf).</td>
</tr>
</tbody>
</table>

See also

- `<trace>.state`
- `CAnalyzer`
- `IProbe`
- `Probe`
- `Trace.METHOD.ART()`
- `Trace.METHOD.CAnalyzer()`
- `Trace.METHOD.Integrator()`
- `Trace.METHOD.LA()`
- `Trace.METHOD.ONCHIP()`
- `Trace.METHOD.SNOOPer()`
- `Analyzer`
- `HAnalyzer`
- `LOGGER`
- `SystemTrace`
- `Trace.METHOD.FDX()`
- `Trace.METHOD.IProbe()`
- `Trace.METHOD.LA()`
- `Trace.METHOD.SNOOPer()`
- `ART`
- `Integrator`
- `Onchip`
- `Trace.METHOD.Analyzer()`
- `Trace.METHOD.HAnalyzer()`
- `Trace.METHOD.LOGGER()`

▲ 'Trace Functions' in 'General Function Reference'
▲ 'Release Information' in 'Release History'
<trace>.Mode

Set the trace operation mode

Format:  

<mode>:  

Fifo
Stack
Leash
STREAM
PIPE
RTS

FAST
Create
64Bit
Change
Compress
Prestore [ON | OFF] (deprecated) Use ETM.DataTracePrestore instead.
SLAVE [ON | OFF]
PrePost [ON | OFF] MasterBrk [ON | OFF]
SlaveBrk [ON | OFF]
HT [Trace | Prestore | Selective | OFF]
100MHZ
DCC
Memory
DetailTrace
EventTrace
FlowTrace
LoopTrace
SPB [ON | OFF]
RPB [ON | OFF]

Selects the trace operation mode. The most common operation modes are:

- **Fifo**: If the trace is full, new records will overwrite older records. The trace records always the last cycles before the break.

- **Stack**: If the trace is full recording will be stopped. The trace always records the first cycles after starting the trace.

- **Leash**: Stops the program execution when trace is nearly full.
STREAM
The trace data is immediately conveyed to a file on the host after it was placed into the trace. This procedure extends the size of the trace memory to up to several T Frames.

STREAM mode can only be used if the average data rate at the trace port does not exceed the maximum transmission rate of the host interface in use. Peak loads at the trace port are intercepted by the trace memory, which can be considered to be operating as a large FIFO.

The streaming file is placed into the TRACE32 temp directory (OS.PresentTemporaryDirectory()) by default and is named <trace32_instance_id>streama.t32 (OS.ID()). If you explicitly want to specify a location for the streaming file use the command <trace>.STREAMFILE <file>.

Note that the contents of the streaming file are in a propriety format and not intended for use in external applications. See Trace.EXPORT for details on how to export trace data for external applications.

Please be aware that the streaming file is deleted as soon as you de-select the STREAM mode or when you quit TRACE32.

- TRACE32 PowerTrace II and the CombiProbe are required for the STREAM mode.
- TRACE32 POWERTRACE/ ETHERNET supports the STREAM mode for the following processor architectures: ARM ETM, Ceva-X ETM, Hexagon QETM, MicroBlaze, MIPS32, NIOS-II, TEAKLITE-III ETM, TMS320C55X, TMS320C6x00. For information about other processor architectures, please contact our support.

PIPE
The trace data is immediately conveyed to the host and distributed to user-defined trace sinks. Not supported with PowerTrace Ethernet 256/512MB. See <trace>.PipeWRITE.

RTS
The RTS radio button is only an indicator that shows if Real-time Profiling is enabled. For enabling RTS use the command RTS.ON.

Further operation modes:

- **100MHz**
  Port analyzer only
  100 MHz mode.

- **64Bit**
  LOGGER only
  LOGGER mode for 64-bit traces.

- **BusTrace**
  BusTrace mode.

- **ClockTrace**
  Clock trace mode.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compress</td>
<td>Compressed information transfer for FDX trace.</td>
</tr>
<tr>
<td>Create</td>
<td>Create LOGGER software trace control block by debugger.</td>
</tr>
<tr>
<td>DetailTrace</td>
<td>Address and data of all cycles except for fetches and free cycles are recorded.</td>
</tr>
<tr>
<td>E</td>
<td>Dualport access (E).</td>
</tr>
<tr>
<td>EventTrace</td>
<td>Recording starts after trigger event.</td>
</tr>
<tr>
<td>FlowTrace</td>
<td>FlowTrace mode.</td>
</tr>
<tr>
<td>HT OFF</td>
<td>The Hypertrace unit is turned off.</td>
</tr>
<tr>
<td>HT Prestore</td>
<td>The Hypertrace samples the last 64 (32) bus cycles before a selective trace with the regular trace is made.</td>
</tr>
<tr>
<td>HT Selective</td>
<td>The Hypertrace samples the same records like the regular trace. This mode can be used to make large selective address traces or function nesting analysis.</td>
</tr>
<tr>
<td>HT Trace</td>
<td>The Hypertrace unit samples all cycles (this is the default).</td>
</tr>
<tr>
<td>LATCH</td>
<td>Latch transients.</td>
</tr>
<tr>
<td>LoopTrace</td>
<td>Flow trace inhibiting redundant entries to capture memory.</td>
</tr>
<tr>
<td>MasterBrk</td>
<td>When activated, the break state of the HA120 unit will cause all other traces to enter break state (e.g. TA32).</td>
</tr>
<tr>
<td>PCP</td>
<td>PCP trace.</td>
</tr>
<tr>
<td>Poststore</td>
<td>tbd.</td>
</tr>
<tr>
<td>PostTrace</td>
<td>tbd.</td>
</tr>
<tr>
<td>PrePost</td>
<td>Forces a recording of the opcode fetch cycle at the beginning and at the end of program execution. The recording does not depend on how the trace trigger unit was programmed. The special markers GO and BRK in the list window emphasize these forced recordings.</td>
</tr>
</tbody>
</table>
Prestore
(deprecated)

Samples the last opcode-fetch address before a selective data access. The size of the trace memory is halved automatically. By using this mode is it possible to get information about which program has made an access to a variable without tracing the complete program flow.

RPB
TriCore only

When enabling a bus trace, all read and write accesses on this bus will be traced by default, including data values and addresses. This trace is independent of Onchip.Mode DataTrace which only affects TriCore and PCP. The recorded data will be mixed into the program flow at the point it was placed into the trace memory by the silicon, e.g. wd-rpb marks a write access on remote peripheral bus. There is no support for data cycle assignment as it is for TriCore LMB bus trace.

SLAVE OFF

Separates the trace from the program execution, i.e. trace is recording even when the program execution is stopped (very rarely used command).

SLAVE ON

Ties the trace to the execution of the program, i.e. trace and the filter/trigger work only during user program execution (very rarely used command).

SlaveBrk

When activated, the break of other traces (e.g. TA32) will cause the HA120 unit to enter break state.

NOTE: The break state is not shown in the status window!

SPB
TriCore only

When enabling a bus trace, all read and write accesses on this bus will be traced by default, including data values and addresses. This trace is independent of Onchip.Mode DataTrace which only affects TriCore and PCP. The recorded data will be mixed into the program flow at the point it was placed into the trace memory by the silicon, e.g. rd-spb marks a read access on system peripheral bus. There is no support for data cycle assignment as it is for TriCore LMB bus trace.

See also

- <trace>.state
- Trace
- ClProbe
- COVerage.METHOD
- Analyzer.MODE.FLOW()
- ‘On-chip Trace’ in ‘StarCore Debugger and Trace’
- ‘CPU specific Commands’ in ‘TriCore Debugger and Trace’
- ‘HCS12 On-chip Debug Module’ in ‘FIRE Emulator for HC12/MCS12’
- ‘HCS12 On-chip Debug Module’ in ‘FIRE Emulator for HC12/MCS12’
- ‘Release Information’ in ‘Release History’
- ‘CPU specific Trace Commands’ in ‘Simulator for TriCore’
Selects which channels are connected to the port analyzer. This option is only available when the emulation head has more than 64 port analyzer channels. The AUTO selection is only available in port analyzer version 2.

See also
- Trace

Switch off

Disables both trace memory and the trigger unit. The trace memory can be read and the trigger unit be programmed.

See also
- <trace>.state
- CIProbe
- Trace
- Analyzer.STATE()
- 'Emulator Functions' in 'FIRE User's Guide'

TBD.

TBD.
<trace>.PC

Display PC in real-time

| Format: | <trace>.PC [ON | OFF] |
|---------|----------------------|

A certain hardware on the trace allows the displaying of the PC (program counter), while the CPU is running in real time. Without parameter the PC is displayed in the message line. It is displayed in the state line continuously, with the option ON. This command is useful if the program is running in loops (e.g. waiting for input).

See also

- <trace>.state
- Trace
- Analyzer.PC()

▲ 'Emulator Functions' in 'FIRE User's Guide'
<trace>.PipePROTO

Unload all DLLs

Format:  <trace>.PipePROTO (deprecated)
         Use <trace>.CustomTraceLoad "" instead.

See also
■ <trace>.PipePROTO.COMMAND
■ <trace>.state
■ <trace>.PipePROTO.load
■ Trace

<trace>.PipePROTO.COMMAND

Send command to DLLs

Format:  <trace>.PipePROTO.COMMAND [cmd_line_args] (deprecated)
         Use <trace>.CustomTrace.<label>.COMMAND instead.

See also
■ <trace>.PipePROTO

<trace>.PipePROTO.load

Define a user-supplied DLL as trace sink

Format:  <trace>.PipePROTO.load  <dll_name>  [cmd_line_args] (deprecated)
         Use <trace>.CustomTraceLoad instead.

Loads a user supplied DLL (or “shared object” under Unix) as trace sink. Up to 8 DLLs can be loaded as trace sinks simultaneously. The debugger software will pass each received message to all loaded DLLs. The DLLs can then filter and process the messages in any possible way.

See also
■ <trace>.PipePROTO
<trace>.PipeWRITE

Connect to a named pipe to stream trace data

**Format:**

```
<trace>.PipeWRITE [ <name> ]
```

Connect to a named pipe to stream the raw trace data to an external application. If `<name>` is omitted, the debugger disconnects from the named pipe.

**Example:** (for Windows)

```
Trace.Mode PIPE ; switch to PIPE mode
Trace.PipeWRITE \\.\pipe\ptrace ; connect to named pipe
; run test
Go ... ; trace data now streamed to
Break
Trace.PipeWRITE ; disconnect from named pipe
```

**See also**

- Trace

<trace>.PortFilter

Specify utilization of trace memory

**Format:**

```
<trace>.PortFilter ON | OFF | PACK | MAX | AUTO
```

If the trace information is conveyed to the host computer at the recording time, it is advantageous to reduce the amount of data to be conveyed. This goal can be achieved by the following:

- No recording of idle cycles (applies only if the on-chip trace logic generates idle cycles).
- No generation of the TRACE32 tool time stamp, if it is not required for the intended analysis.

The command `Trace.PortFilter` allows the following configurations:

<table>
<thead>
<tr>
<th>Setting</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON</td>
<td>Idle cycles are partly not recorded. This setting is recommended for most accurate TRACE32 tool time stamps. (See also: ETM.TimeMode)</td>
</tr>
<tr>
<td>OFF</td>
<td>All generated trace information is recorded (diagnose purpose only).</td>
</tr>
</tbody>
</table>
| PACK       | No idle cycles are recorded. Caveats: The accuracy of the TRACE32 tool time stamp is reduced.  
|           | This setting is the default with Trace.Method CAnalyzer (CombiProbe/µTrace).  
|           | This setting is recommended for trace mode STREAM. |
| MAX       | No idle cycles are recorded and no TRACE32 tool time stamp is generated.  
|           | This setting is recommended if the trace mode STREAM is used for Trace-based Code Coverage.  
|           | This setting is automatically activated when the command RTS.ON is executed (only ETMv3 and later) |
| AUTO      | Best setting is done automatically by TRACE32 (default).  
|           | With real-time streaming (Trace.Mode STREAM): PortFilter operates in PACK mode  
|           | With real-time profiling (RTS.ON): PortFilter operates in MAX mode  
|           | In all other cases:  
|           | • Analyzer (PowerTrace) operates in ON mode  
|           | • CAnalyzer (CombiProbe/µTrace) operates in PACK mode. |

See also
- RTS.ON
- 'Trace-based Code Coverage' in 'Nexus Training'
**<trace>.PortSize**

*Set external port size*

Informs the debugger that the externally visible port size differs from the internal port size setting of `TPIU.PortSize` and sets the specified external port size. Use this command if there is application-specific logic between the TPIU and the analyzer, for example in the programmable logic part of an FPGA SoC.

The external port size value refers to the number of data pins that are physically connected to the analyzer.

The internal port size value refers to the setting that will be programmed into the target's TPIU.

**AUTO**
- (default)
- The external port size value of `<trace>.PortSize` equals the internal port size value of `TPIU.PortSize`.

**1 … 16**
- Use the specified number of data pins as the external port size.

See also
- `TPIU.PortSize`
- “Introduction” in ‘Debugging Embedded Cores in Xilinx FPGAs [Zynq]’

---

**<trace>.PortType**

*Specify trace interface*

**Format:**

```
<trace>.PortType TPIU | STM | SWV (CombiProbe)
<trace>.PortType TPIU | TPIUX2 | TPIUX3 | TPIUX4 | STM | RTP | TPIU+RTP (Preprocessor AutoFocus II)
<trace>.PortType HSSTP | SETM3 (Preprocessor Serial)
```

Inform TRACE32 PowerView about the trace port interface type provided by your target. This might be necessary for the following TRACE32 trace tools:

**TRACE32 CombiProbe:**

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TPIU (default)</td>
<td>CombiProbe is connected to TPIU.</td>
</tr>
<tr>
<td>STM</td>
<td>CombiProbe is connected to STM interface.</td>
</tr>
<tr>
<td>SWV</td>
<td>CombiProbe is connected to Serial Wire Viewer interface.</td>
</tr>
</tbody>
</table>
### TRACE32 Preprocessor AutoFocus II:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TPIU</strong></td>
<td>TRACE32 AutoFocus II Preprocessor is connected to TPIU (default). Also supported by LA-7991 PP-ARM-ETM-AF.</td>
</tr>
<tr>
<td><strong>TPIUX2</strong></td>
<td>TRACE32 AutoFocus II is connected to a trace port interface that provides 2 ETMv3 interfaces, multicore chip without TPIU (NEC Triton only).</td>
</tr>
<tr>
<td><strong>TPIUX3</strong></td>
<td>TRACE32 AutoFocus II is connected to a trace port interface that provides 3 ETMv3 interfaces, multicore chip without TPIU (NEC Triton only).</td>
</tr>
<tr>
<td><strong>TPIUX4</strong></td>
<td>TRACE32 AutoFocus II is connected to a trace port interface that provides 4 ETMv3 interfaces, multicore chip without TPIU (NEC Triton only).</td>
</tr>
<tr>
<td><strong>STM</strong></td>
<td>TRACE32 AutoFocus II Preprocessor is connected to STM interface. Also supported by LA-7991 PP-ARM-ETM-AF.</td>
</tr>
<tr>
<td><strong>RTP</strong></td>
<td>TRACE32 AutoFocus II Preprocessor is connected to Ram Trace Port interface.</td>
</tr>
<tr>
<td><strong>TPIU+RTP</strong></td>
<td>TRACE32 AutoFocus II Preprocessor is connected to a trace port interface that includes a TPIU and a Ram Trace Port interface.</td>
</tr>
</tbody>
</table>

### TRACE32 Preprocessor Serial:

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>HSSTP</strong></td>
<td>TRACE32 Preprocessor Serial is connected to a HSSTP interface (default).</td>
</tr>
<tr>
<td><strong>SETM</strong></td>
<td>TRACE32 Preprocessor Serial is connected to a SETM interface.</td>
</tr>
</tbody>
</table>

---

**See also**
- `<trace>.state`
- Trace
The contents of a trigger unit counter can be displayed as a function of time. Time counters are displayed in percent and event counters as events/s. (see also `Count.PROfile`).

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;trace&gt;.PROfile &lt;counter&gt; [&lt;gate&gt;]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;gate&gt;</code>:</td>
<td>`0.1s</td>
</tr>
</tbody>
</table>

See also
- `<trace>.state`
- `Trace`
The `<trace>.PROfileChart` command group displays distributions versus time graphically. For example, the command group can be used to show the processing ratio of selected tasks versus time.

**See also**

- `<trace>.state`
- `<trace>.Chart`
- `<trace>.PROfileChart.DISTance`
- `<trace>.PROfileChart.DURATION`
- `<trace>.PROfileChart.RATE`
- `<trace>.PROfileChart.TASK`
- `<trace>.STATistic`

▲ 'Release Information' in 'Release History'
### Display a profile chart

**Format:**

```
<trace>.PROfileChart.COUNTER [%<format>] [<record_range>] [<scale>] [items] [/<option>]
```

**<format>:**

- ZeroUp. [<width>]
- Up. [<width>]
- Down. [<width>]
- Frequency. [<width>]
- POWER. [<width>]

**<width>:**

- DEFault
- Byte
- Word
- Long
- Quad
- TByte
- HByte

**<items>:**

- DEFault
- ALL
- <cpu>
- <signals>
- Port.<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_option>:**

- Vector
- Steps
- Color

Shows the time profiles of a counter that is traced as data value.

**See also**

- `<trace>.PROfileChart`
Display the time interval for a single event graphically. For the general options for all `<trace>.PROfileChart` commands refer to `<trace>.Chart`.

**Example for TRACE32 ICD and TRACE32 PowerTrace:**

If no selective tracing is possible, use the option `/Filter` to filter out the event of interest.

```
B::Go
B::Break
B::Trace.PROfileChart.DIStance /FILTER Address InterruptEntry
```

**Example for TRACE32-ICE or TRACE32-FIRE:**

Perform a selective trace on the event of interest.

```
Analyzer.ReProgram
(
    ADDR AlphaB InterruptEntry ; program the analyzer to sample all
    Sample.enable IF AlphaBreak ; InterruptEntry
)
Go
...
Break
Trace.PROfileChart.DIStance ; display the time interval between
                        ; the sampled InterruptEntry
                        ; graphically
```

Format: `<trace>.PROfileChart.DIStance` [<record_range>] [<scale>] [/<option>]
        `<trace>.Chart.DIStance` (deprecated)
See also

- `<trace>.PROfileChart`
Distribution display in timeslices

Format: `<trace>.PROfileChart.DistriB` [<`record_range>`, <`scale`>, <`option`>]

`<option>`:
- FILE
- FlowTrace | BusTrace | CTS
- Track
- RecScale | TimeScale | TimeZero | TimeREF
- Filter `<item>`

Shows a graphical representation of the specified trace item as a percentage of a timeslice.

Examples:

```plaintext
; Display distribution of data value for AVG_QADC
Trace.PROfileChart.DistriB Data.L /Filter Address AVG_QADC

; Print distribution of data value written for AVG_QADC for the
; record range (-2000.)--(-1000.)
WinPrint.Trace.PROfileChart.DistriB (-2000.)--(-1000.) Data.L /Filter
Address V.RANGE(AVG_QADC)
```

See also
- `<trace>.PROfileChart`
- MIPS.PROfileChart
**<trace>.PROfileChart.DURation**

Time between two events

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;trace&gt;.PROfileChart.DURation</code> [ <code>&lt;record_range&gt;</code> ] [ <code>&lt;scale&gt;</code> ] [ <code>/&lt;option&gt;</code> ]</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;option&gt;</code>:</td>
<td>FILE</td>
</tr>
<tr>
<td></td>
<td>FlowTrace</td>
</tr>
<tr>
<td></td>
<td>Accumulate</td>
</tr>
<tr>
<td></td>
<td>INCremental</td>
</tr>
<tr>
<td></td>
<td>ATOA</td>
</tr>
<tr>
<td></td>
<td>BTOA</td>
</tr>
<tr>
<td></td>
<td>FilterA <code>&lt;filter&gt;</code></td>
</tr>
<tr>
<td></td>
<td>RecScale</td>
</tr>
</tbody>
</table>

Graphical display of time intervals between two events.

**TRACE32 PowerTrace**

In order to use the command `Trace.STATistic.DURation`:

- Check if both events are exported by a trace packet. Information reconstructed by TRACE32 is not analyzed.
- Alternatively use a `TraceEnable` breakpoint export the event as a trace packet.

The options **FilterA** and **FilterB** provide you with the means to describe your event.

- **FilterA** `<item>` Specify the first event.
- **FilterB** `<item>` Specify the second event.
Trace.Mode Leash

Break.Set 0x9cb0 /Program /TraceEnable

Break.Set 0x9e3c /Program /TraceEnable

Go

WAIT !STATE.RUN()

Trace.STATistic.DURation /FilterA Address 0x9cb0 /FilterB Address 0x9e3c

Trace.PROfileChart.DURation /FilterA Address 0x9cb0
  /FilterB Address 0x9e3c

 Displays min and max duration per 10 pixels
Displays min and max duration per 10 pixels (with a higher resolution)
Perform a selective trace on both events. Mark the first event in the trace with A and the second event with B.

Analyzer.ReProgram ; selective trace
(
  ADDR AlphaBreak InterruptEntry ; program the analyzer to sample
  ADDR BetaBreak InterruptExit ; only InterruptEntry and
      ; InterruptExit

  Sample.enable IF AlphaBreak||BetaBreak

  Mark.A IF AlphaBreak ; mark InterruptEntry with A
  Mark.B IF BetaBreak ; mark InterruptExit with B
)

E::Go ; measurement
E::Break
E::Trace.PROfileChart.DURation ; display time between
      ; InterruptEntry and
      ; InterruptExit graphically

ATOA Display the time interval from A to A.
BTOA Display the time interval from B to A.
BTOB Display the time interval from B to B.
If no selective tracing is possible and more specific events should be displayed it is also possible to use the options:

By default the time is taken from the A marked event to the B marked event.

See also

- <trace>.PROfileChart
Group profile chart

Format:
<trace>.PROfileChart.GROUP [<record_range>] [<scale>] [/<option>]

<option>:
FILE
FlowTrace | BusTrace | CTS
Track
RecScale | TimeScale | TimeZero | TimeREF
Filter <item>

Analyzes the group behavior and displays the result as a color chart with fixed time intervals.

Example:

GROUP.Create "INPUT" /jquant2 /jquant1 /jidctred /jdinput /AQUA
GROUP.Create "JPEG" /jadapimin /jdcolor /jddctmgr /jdcoeffct /NAVY
Go
Break
Trace.PROfileChart.GROUP

See also
- <trace>.PROfileChart
- ‘Release Information’ in ‘Release History’
Graphical display of the event frequency over the time. For the general options for all `<trace>.PROfileChart` commands refer to `<trace>.Chart`.

**Example:**

Display the **TARGET FIFO OVERFLOW** (FIFOFULL) rate over the time.

```
Trace.PROfileChart.Rate /Filter FIFOFULL
```
Example for TRACE32-ICD and TRACE32-PowerTrace:

If no selective tracing is possible, use the option /Filter to filter out the data of interest.

B::Go
B::Break
B::B::Trace.PROfileChart.Rate /FILTER Address InterruptEntry

Example for TRACE32-ICE and TRACE32-FIRE:

Perform a selective trace on the data of interest.

Analyzer.ReProgram
(
    ADDR AlphaBreak InterruptEntry ; sample all InterruptEntry
    Sample.Enable IF AlphaBreak
)
Go ; collect data
Break
Trace.PROfileChart.Rate ; Display event frequency of
E::Trace.PROfileChart.Rate ; InterruptEntry graphically

<table>
<thead>
<tr>
<th>events/s</th>
<th>0s</th>
<th>-40.000s</th>
<th>-30.000s</th>
<th>-20.000s</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>144.</td>
<td>128.</td>
<td>112.</td>
<td>96.</td>
</tr>
<tr>
<td></td>
<td>80.</td>
<td>64.</td>
<td>48.</td>
<td>32.</td>
</tr>
<tr>
<td></td>
<td>16.</td>
<td>0.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

See also
- <trace>.PROfileChart
**Trace.PROfileChart.sYmbol** is based on a flat function run-time analysis.

Push the **Profile** button to get information on the dynamic behavior of the program.

To draw the **Trace.PROfileChart.sYmbol** graphic, TRACE32 PowerView partition the recorded instruction flow information into time intervals. The default interval size is 10.us.

©1989-2019 Lauterbach GmbH
For each time interval rectangles are draw that represent the time ratio the executed functions consumed within the time interval. For the final display this basic graph is smoothed.

<table>
<thead>
<tr>
<th><strong>Fine</strong></th>
<th>Decrease the time interval size by the factor 10</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Coarse</strong></td>
<td>Increase the time interval size by the factor 10</td>
</tr>
</tbody>
</table>

The time interval size can also be set manually.

```
Trace.PROfileChart.svgmb /InterVal 5.ms ; change the time interval size to 5.ms
```
Color Assignment - Basics

- The tooltip at the cursor position shows the color assignment and the used interval size.

- Use the control handle on the right upper corner of the Trace.PROfileChart.Symbol window to get a color legend.
## Color Assignment - Statically or Dynamically

<table>
<thead>
<tr>
<th><strong>FixedColors</strong></th>
<th>Colors are assigned fixed to functions (default).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed color assignment has the risk that two functions with the same color are drawn side by side and thus may convey a wrong impression of the dynamic behavior.</td>
<td></td>
</tr>
</tbody>
</table>

| **AlternatingColors** | Colors are assigned by the recording order of the functions again and again for each measurement. |

---

### See also

- `<trace>.PROfileChart`
Analyzes the dynamic task behavior and displays the result as a color chart with fixed time intervals. This command requires OS-ware tracing.

Example to analyze CPU load:

```plaintext
; group all tasks that contain an idle loop to the group "Idle"
; all other tasks are members of the group "other"

; merge the result of all "Idle" group members and
; use white as "Idle" group color
GROUP.CreateTASK "Idle" "Idle_Task" /Merge /WHITE

; merge the result of all "other" group members
GROUP.Merge "other"

; use green as "other" group color
GROUP.COLOR "other" GREEN

; display the CPU load graphically
Trace.PROfileChart.TASK
```
(unknown) represents the time before the first task information was recorded to the trace.

See also
- `<trace>.PROfileChart`
The command group `<trace>.PROfileSTATistic` shows the results of numerical interval analysis in tabular format.

**Format:**  
```
<trace>.PROfileSTATistic [<trace_area>] [I<option>]
```

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

See also

- `<trace>.state`
- Trace
- `<trace>.Chart`
- `<trace>.PROfileChart`

▲ 'Release Information' in 'Release History'
<trace>.Program

Opens an editor window, which is used for programming the trigger unit. The program entry in the window is guided by softkeys and the online-help system. To program the trace press the **Compile** button. After successful programming, the file name will be displayed in the trace status window. The default file name is the file currently used by the trigger unit. The file name default extension is `.ts`. This command can be executed by clicking the **Edit** button in the trace state window. If the state of the trace is ARM and this command is executed, the state is switched to **OFF** before programming the trigger unit, automatically. See **Analyzer Programming Guide** for details about the programming language.

See also

- `<trace>.ReProgram`
- `<trace>.state`
- Trace
- SETUP.EDITOR

▲ 'Editing Files in the TRACE32 Editors - Syntax Highlighting' in 'PowerView User's Guide'
▲ 'Release Information' in 'Release History'
▲ 'Trigger Programming' in 'Training FIRE Analyzer'
<trace>.PROTOcol

See also
- <trace>.PROTOcol.STATistic
- <trace>.PROTOcol.Chart
- <trace>.PROTOcol.Draw
- <trace>.PROTOcol.Find
- <trace>.PROTOcol.EXPORT
- <trace>.PROTOcol.List
- <trace>.PROTOcol.STATistic
- <trace>.state
- Trace

<trace>.PROTOcol.Chart

Graphic display for user-defined protocol

Format:  
<trace>.PROTOcol.Chart <protocol> <parlist> [<items> ...] [/<options>]

<parlist>:  
<line> | <lines> <options>

<line>:  
<signal>

<lines>:  
<signal>...

<options>:  
'options for ASYCH'  
'options for CAN'

'options for I2C'

'options for JTAG'

'options for USB'

'user_defined_options'

<option>:  
FILE
Track
CTS
RecScale
TimeScale
TimeZero
TimeREF
Filter

<items>:  
'%'<format>

<line>

DEFault | ALL

Time.Back | Time.Fore | Time.REF

Time.Zero | Time.Trigger

SyncClock

SPARE
Options:

FILE       Display trace memory contents loaded with the command Trace.FILE.
Track      Track other trace list window (tracks to record number or time)
RecScale   Record Scaling
TimeScale  Timed Scaling
Filter

Items:

TIme.REF   Time marker, relative to reference point
TIme.Trigger Time marker, relative to trigger point
TIme.Zero  Time marker, relative to global reference
SyncClock  Synchronous clock event

Example: Display the user-defined protocol “proto1” on line x.0

PP::a.proto.c proto1 x.0

See also
- <trace>.PROTOcol
### Format:

```plaintext
<trace>.PROTOcol.Draw <protocol> <parlist> [<items> …] [/<options>]
```

### <parlist>:

```plaintext
<line> | <lines> <options>
```

### <line>:

```plaintext
<signal>
```

### <lines>:

```plaintext
<signal> …
```

### <options>:

- `<options for ASYCH>`
- `<options for CAN>`
- `<options for I2C>`
- `<options for JTAG>`
- `<options for USB>`
- `<user_defined_options>`

### <option>:

- `FILE`
- `Track`
- `CTS`
- `RecScale`
- `TimeScale`
- `TimeZero`
- `TimeREF`
- `Filter`

### <items>:

- `%<format>`
- `<line>
  DEFault | ALL
  TIme.Back | TIme.Fore | TIme.REF
  TIme.Zero | TIme.Trigger
  SyncClock
  SPARE

### Options:

- **FILE**
  Display trace memory contents loaded with the command `Trace.FILE`.

- **Track**
  Track other trace list window
  (tracks to record number or time)

- **RecScale**
  Record Scaling

- **TimeScale**
  Timed Scaling

- **Filter**
  tbd.
Items:

- **TIme.REF**: Time marker, relative to reference point
- **TIme.Trigger**: Time marker, relative to trigger point
- **TIme.Zero**: Time marker, relative to global reference
- **SyncClock**: Synchronous clock event

**Example**: Display the user-defined protocol “proto1” on line x.0

```plaintext
PP::a.proto.d proto1 x.0
```

**See also**
- `<trace>.PROTOcol`

---

**<trace>.PROTOcol.EXPORT**

Export trace buffer for user-defined protocol

**Format**:  
```
<trace>.PROTOcol.EXPORT.W <protocol> <parlist> <file> [range] …
```

**<parlist>**:  
```
<protocol_parameters>
```

**Example**: Export the user-defined protocol “proto1” on line x.0 to test.lst

```plaintext
PP::a.proto.EXPORT proto1 x.0 test.lst
```

**See also**
- `<trace>.PROTOcol`
Find in trace buffer for user-defined protocol

Format:

```
<trace>.PROTOcol.Find <protocol> <parlist> [<items>...] [/<options>]
```

```
<parlist>:  <line> | <lines> <options>
```

```
<line>:     <signal>
<lines>:    <signal> ...
```

```
<options>:  <options for ASYCH>
            <options for CAN>
            <options for I2C>
            <options for JTAG>
            <options for USB>
            <user_defined_options>
```

```
<option>:   FILE
           Back
           NoFind
```

Options:

**FILE**

Display trace memory contents loaded with the command `Trace.FILE`.

**Back**

Search back

**NoFind**

Example: Find in the user-defined protocol “proto1” on line x.0

```
PP::a.proto.f proto1 x.0
```

See also

- **<trace>.PROTOcol**
- **<trace>.PROTOcol.List**
Format:  \texttt{<trace>.PROTOcol.List} <protocol> <parlist> [[<items>]] [/<options>]

\texttt{<parlist>}:  \texttt{<line> | <lines> <options>}

\texttt{<line>}:  \texttt{<signal>}
\texttt{<lines>}:  \texttt{<signal> \ldots}

\texttt{<options>}:  \texttt{<options for ASYCH>}
\texttt{<options for CAN>}
\texttt{<options for I"2C>}
\texttt{<options for JTAG>}
\texttt{<options for USB>}
\texttt{<user\_defined\_options>}

\texttt{<option>}:  FILE
Track

\texttt{<items>}:  \%<format>
\texttt{<line>}
DEFault \texttt{ | ALL}
TIme.Back \texttt{ | TIme.Fore \texttt{ | TIme.REF}}
TIme.Zero \texttt{ | TIme.Trigger}
SyncClock
SPARE

\texttt{<format>}:  Hex \texttt{ | Decimal \texttt{ | BINary \texttt{ | Ascii}}}
Timing \texttt{ | HighLow}
LEN \texttt{<size>}
TImeAuto \texttt{ | TImeFixed}

Options:

\textbf{FILE}  Display trace memory contents loaded with the command \texttt{Trace.FILE}.

\textbf{Track}  Track other trace list window
(tracks to record number or time)

Formats:

\textbf{Timing}  Display single bits as vertical timing

\textbf{HighLow}  Display single bits as HIGH/LOW value

\textbf{Hex}  Display single bytes in hex values
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decimal</td>
<td>Display single bytes in decimal values</td>
</tr>
<tr>
<td>Binary</td>
<td>Display single bytes in binary values</td>
</tr>
<tr>
<td>Ascii</td>
<td>Display single bytes as ascii characters</td>
</tr>
<tr>
<td>LEN &lt;size&gt;</td>
<td>Specify the width of non numeric fields (e.g. symbols)</td>
</tr>
<tr>
<td>TimeAuto</td>
<td>The unit of time is selected automatically.</td>
</tr>
<tr>
<td>TimeFixed</td>
<td>The displayed unit of time is fixed.</td>
</tr>
</tbody>
</table>

**Items:**

- **DEFault**: Default selections (see list below)
- **ALL**: Select all recorded data (superset of DEFault)
- **Time**: Time marker (default Time.Fore)
- **Time.Back**: Time marker, relative time to previous record
- **Time.Fore**: Time marker, relative time to next record
- **Time.REF**: Time marker, relative to reference point
- **Time.Trigger**: Time marker, relative to trigger point
- **Time.Zero**: Time marker, relative to global reference
- **SyncClock**: Synchronous clock event
- **SPARE**: Displays an empty block

**Example:** Displays the user-defined protocol “proto1” on line x.0

```
PP::a.proto1 proto1 x.0
```
; JTAG <tck> <tms> <tdi> <tdo> <trst> <initstate>
; when the sampling is started the JTAG state machine is in state
; run-test/idle
Trace.ListProtocol JTAG X.8 X.9 X.4 X.12 X.14 run-test/idle

; CAN <canline> <frequency> DEFault | ALL
; the frequency is defined in Hz
Trace.ListProtocol CAN X.7 1000000. DEFault

; USB <+signal> <-signal>
Trace.ListProtocol USB X.17 X.18

; I2C <scl> <sda>
Trace.ListProtocol I2C X.22 X.23

; asynchronous communication interface
; ASYNC <asyline> <frequency> +- <parity> <length> <stopbit>

Trace.ListProtocol ASYNC X.6 3600. + EVEN 7 1STOP STRING
Trace.ListProtocol ASYNC X.5 2400. - NONE 5 2STOP CHAR

; special protocols
; TRACE32 offers a API that allows to use special, customer specific
; protocols
Trace.ListProtocol protojtag.dll X.4 X.12 X.14

; examples for special protocols are provided under ~/demo/proto
; on the TRACE32 software DVD

See also
■ <trace>.PROTOcol
■ <trace>.PROTOcol.STATistic
■ <trace>.PROTOcol.Find
■ <trace>.REF
▲ 'Release Information' in 'Release History'
### Display statistics for user-defined protocol

**Format:**

```
<trace>.PROTOcol.STATistic <protocol> <parlist> [[<items>]] [l<options>]
```

- **<parlist>:** `<line> | <lines> <options>`
- **<line>:** `<signal>`
- **<lines>:** `<signal> ...`

- **<options>:** `<options for ASYCH>`
  - `<options for CAN>`
  - `<options for I2C>`
  - `<options for JTAG>`
  - `<options for USB>`
  - `<user_defined_options>`

- **<option>:** `FILE
  BEFORE
  AFTER
  List
  Filter
  Accumulate
  INCremental
  FULL`

- **<items>:** `%<format>`
  - `<line>`
    - `DEFault | ALL
      Time.Back | Time.Fore | Time.REF
      Time.Zero | Time.Trigger
      SyncClock
      SPARE`

- **<format>:** `Hex | Decimal | BINary | Ascii
  Timing | HighLow
  LEN <size>
  TimeAuto | TimeFixed`

### Options:

- **FILE**
  - Display trace memory contents loaded with the command `Trace.FILE`.

- **Track**
  - Track other trace list window
    (tracks to record number or time)
**Example:** Display statistics in the user-defined protocol “proto1” on line x.0

```
PP::a.proto.stat proto1 x.0
```

---

**See also**

- `<trace>.PROTOcol`
- `<trace>.PROTOcol.List`
Protocol specific Options

Options for ASYNC

```
<line>:          <trans> | <rec>
<trans>:        <signal>
<rec>:          <signal>

<options>:      <parlist>
<parlist>:      [<baudrate> [<polarity> [<parity> [<bits> [<stopbits> [<disp>]]]]]]]

<baudrate>:     1. ... 1000000.
<polarity>:     + | -
<parity>:       NONE | ODD | EVEN
<bits>:         5 | 6 | 7 | 8
<stopbits>:     1STOP | 2STOP
<disp>:         CHAR | STRING
```

Options for CAN

```
<line>:          <signal>

<options>:      <frequency>

<frequency>:    frequency in Hz
```
### Options for I2C

<table>
<thead>
<tr>
<th>&lt;lines&gt;</th>
<th>&lt;sck&gt;</th>
<th>&lt;sda&gt;</th>
</tr>
</thead>
</table>

### Options for JTAG

<table>
<thead>
<tr>
<th>&lt;lines&gt;</th>
<th>&lt;tck&gt;</th>
<th>&lt;tms&gt;</th>
<th>&lt;tdi&gt;</th>
<th>&lt;tdo&gt;</th>
<th>[&lt;trst&gt;]</th>
<th>[&lt;initial_state&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;tck&gt;</td>
<td>&lt;signal&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tms&gt;</td>
<td>&lt;signal&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tdi&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;tdo&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;trst&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&lt;initial_state&gt;</td>
<td>Exit2-DR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exit1-DR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shift-DR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pause-DR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select-IR-scan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Update-DR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Capture-DR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Select-DR-scan</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exit2-IR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Exit1-IR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Shift-IR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Pause-IR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Run-Test/Idle</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Update-IR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Capture-IR</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Test-Logic-Reset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Options for USB

<lines>:  <+signal> <-signal> [<state>]

 <+signal>:  <signal>
 <-signal>:  <signal>

<state>:  BUSRESET
          GAP
          SYNC
          EOP
          ERRORS
          SOF
          #SETUP
          #PRE
          #IN
          #OUT
          #ACK
          #NACK
          #STALL
          #DATA0
          #DATA1
          #OTHER
Selects the sampling rate in samples/s. Legal values and the resulting sample times are given in the following table:

### PortAnalyzer ICE

<table>
<thead>
<tr>
<th>Rate</th>
<th>Resolution</th>
<th>Total sampling time</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000000.</td>
<td>100 ns</td>
<td>3.27 ms</td>
</tr>
<tr>
<td>5000000.</td>
<td>200 ns</td>
<td>6.55 ms</td>
</tr>
<tr>
<td>2500000.</td>
<td>400 ns</td>
<td>13.1 ms</td>
</tr>
<tr>
<td>1000000.</td>
<td>1 µs</td>
<td>32.7 ms</td>
</tr>
<tr>
<td>500000.</td>
<td>2 µs</td>
<td>65.5 ms</td>
</tr>
<tr>
<td>250000.</td>
<td>4 µs</td>
<td>131.0 ms</td>
</tr>
<tr>
<td>200000.</td>
<td>5 µs</td>
<td>164.0 ms</td>
</tr>
<tr>
<td>100000.</td>
<td>10 µs</td>
<td>327.0 ms</td>
</tr>
<tr>
<td>50000.</td>
<td>20 µs</td>
<td>655.0 ms</td>
</tr>
<tr>
<td>25000.</td>
<td>40 µs</td>
<td>1.31 s</td>
</tr>
<tr>
<td>20000.</td>
<td>50 µs</td>
<td>1.64 s</td>
</tr>
<tr>
<td>10000.</td>
<td>100 µs</td>
<td>3.27 s</td>
</tr>
<tr>
<td>5000.</td>
<td>200 µs</td>
<td>6.55 s</td>
</tr>
<tr>
<td>2500.</td>
<td>400 µs</td>
<td>13.1 s</td>
</tr>
<tr>
<td>2000.</td>
<td>500 µs</td>
<td>16.4 s</td>
</tr>
<tr>
<td>1000.</td>
<td>1 ms</td>
<td>32.7 s</td>
</tr>
<tr>
<td>500.</td>
<td>2 ms</td>
<td>65.5 s</td>
</tr>
<tr>
<td>250.</td>
<td>4 ms</td>
<td>131.0 s</td>
</tr>
<tr>
<td>200.</td>
<td>5 ms</td>
<td>164.0 s</td>
</tr>
<tr>
<td>100.</td>
<td>10 ms</td>
<td>327.0 s</td>
</tr>
</tbody>
</table>
With a mouse click to the corresponding area in the port analyzer state window that command can be executed, too.

```
Port.Rate 1000000. ; set to 1 MHz sample rate
Port.Rate 1.us ; same operation, 1 MHz sample rate
```

### `<trace>.REF`

**Set reference point for time measurement**

| Format: | `<trace>.REF [<time> | <record> | "<trace_bookmark>"`] |
|---|---|
| `<option>`: | FILE |

Sets the reference point for time measurements using the `TIme.REF` column of the `Trace.List` window. The default reference point is always the last record in trace memory. The reference point can also be set via context menu entry Set Ref in `Trace.List`, `Trace.Chart`, `Trace.Timing` etc. windows.

- **<time>**
  Sets the reference point to the global ZERO point. If the time for each trace event is calculated based on timestamps generated by the processor, the global ZERO point is at the start of the trace recording currently stored in the trace buffer. If the time for each trace event is based on timestamps generated by the trace module, the global ZERO point is set to the start of the first debug session after the start of TRACE32 PowerView.

- **<record>**
  Sets the reference point to the time index of the specified record number.

- **<trace_bookmark>**
  Sets the reference point to the time index of the specified bookmark location. You can create trace bookmarks with the `<trace>.BookMark` command.

**Examples:**

```
Trace.REF +2000. ; set reference to record +2000
Trace.REF 100us ; set ref. point to absolute time
Trace.REF "MyRef" ; set ref. point to bookmark "MyRef"
```

**See also**

- `<trace>.GOTO`
- `<trace>.PROTOCOL.List`
- `<trace>.state`
- `<trace>.Timing`
- `<trace>.View`
- Trace
- Analyzer.REF()
<trace>.ReProgram

**Program trigger unit**

Format: `<trace>.ReProgram [<file>]`

Programs the trigger unit of the trace with an existing program. The program should be error free, when using this command. To write an error free program, use the command `<trace>.Program`.

See also
- `<trace>.Program`
- `<trace>.state`
- `Trace`
- ‘Release Information’ in ‘Release History’
- ‘Trigger Programming’ in ‘Training FIRE Analyzer’

<trace>.RESet

**Reset command**

Format: `<trace>.RESet`

Resets the trace unit to its default settings.

See also
- `<trace>.state`
- `ClProbe`
- `Trace`
- ‘Emulator Functions’ in ‘FIRE User’s Guide’
Use this command to manually configure sample times of the trace channels. It is typically used to restore values previously stored using the store button of the Trace.ShowFocus window, or the STOre ANALYZERFOCUS command.

Format: `<trace>.SAMPLE [<channel>] <time1> [time2]`

- **<channel>** Trace signal to be configured
  If the parameter is omitted, all signals are configured with the `<time>` setting.
- **<time1>** 
  **Parameter Type:** Float. The value is interpreted as time in nanoseconds. Sample time offset to trace clock.
  - Positive value: later
  - Negative value: earlier

- **<time2>** 
  **Parameter Type:** Float. The value is interpreted as time in nanoseconds.
  For double data rate mode only:
  Sample time offset to inverted trace clock (usually this is the sample time relative to the falling edge of the trace clock).
  If the parameter is omitted, `<time1>` is used for both sample time offsets.

The available AutoFocus settings and features depend on the preprocessor used:

<table>
<thead>
<tr>
<th>Preprocessor:</th>
<th>Features:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoFocus II</td>
<td>configuration: time resolution: max. range: max. inter-channel range:</td>
<td>individual sample time per channel about 78ps per step +/- 63 * 0.078ns ABS(MAX(time1)-MIN(time1)) &lt; 63*0.078ns</td>
</tr>
<tr>
<td>NEXUS-AutoFocus</td>
<td>configuration: time resolution: max. range: DDR-Mode:</td>
<td>single sample time applied to all channels about 750ps per step +/- 15 * 0.75ns ABS(time1-time2) &lt; 15*0.75ns</td>
</tr>
</tbody>
</table>

See also
- `<trace>.state`
- Trace
- 'Release Information' in 'Release History'
The trace memory contents are stored to the selected file. The default extension of the file name is *.ad.

What is actually saved to the file depends on the Trace.Mode:

- **Trace.Mode FlowTrace**: Trace raw data plus decompressed addresses, data and op-codes are saved.

  If the program and data flow is output by the CPU in a compressed format, it is decompressed before saving it to a file for postprocessing. By reading the source code information the addresses, data value and op-codes are decompressed.

- All other settings for Trace.Mode: Only the trace raw data are saved into <file> if the information from the external busses, ports etc. are recorded to the trace buffer.

### Parameters

| <file> | For some TRACE32 devices additional setting are saved to <file>:
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerProbe (Trace.METHOD Probe)</td>
<td>All Probe settings and all NAME settings are saved to &lt;file&gt;.</td>
</tr>
<tr>
<td>PowerIntegrator (Trace.METHOD Integrator)</td>
<td>All Integrator settings and all NAME settings are saved to &lt;file&gt;.</td>
</tr>
</tbody>
</table>
### Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FlowTrace</td>
<td>Obsolete.</td>
</tr>
<tr>
<td>BusTrace</td>
<td>Save only the trace raw data, if a flow trace is used. This option is helping if a decompression of the program and data trace information is not possible.</td>
</tr>
<tr>
<td>PACK</td>
<td>Save the trace contents is a compact way. <strong>PACK</strong> is less effective and slower then <strong>ZIP</strong>. It is only recommended if the option <strong>ZIP</strong> is not available.</td>
</tr>
<tr>
<td>ZIP</td>
<td>Save the trace contents is a compact way.</td>
</tr>
<tr>
<td>NoCompress</td>
<td>Obsolete.</td>
</tr>
</tbody>
</table>

### <trace_area>

**<trace_bookmark>**

Specify two `<trace_bookmarks>` to define the `<trace_area>` you want to save to the file.

- If you specify only one `<trace_bookmark>`, then the `<trace_area>` ranges from that bookmark up to the end of the trace recording.

See example.

**<record_range>**

You need to specify two record numbers to define the `<trace_area>` you want to save to the file.

See example.

**<record>**

Specify two record numbers to define the `<trace_area>` that you want to save to the file.

- If you specify only one `<record>`, then the `<trace_area>` ranges from that record number up to the end of the trace recording.

See example.

**<timerange>**

You need to specify two absolute time stamps that are based on the zero time to define the `<trace_area>` you want to save to file.

See example.

**<time>**

Specify two absolute time stamps that are based on the zero time to define the `<trace_area>` you want to save to the file.

- If you specify only one absolute time stamp, then the `<trace_area>` ranges from that time stamp up to the end of the trace recording.
Example for `<trace_bookmark>`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace.List</td>
<td>display trace listing</td>
</tr>
<tr>
<td>Trace.BookMark &quot;First&quot; -123366.</td>
<td>mark the trace record -123366. with the bookmark &quot;First&quot;</td>
</tr>
<tr>
<td>Trace.BookMark &quot;Last&quot; -36675.</td>
<td>mark the trace record -36675. with the bookmark &quot;Last&quot;</td>
</tr>
<tr>
<td>BookMark.List</td>
<td>list all bookmarks</td>
</tr>
<tr>
<td>Trace.Save testb &quot;First&quot; &quot;Last&quot;</td>
<td>save trace contents between bookmarks &quot;First&quot; and &quot;Last&quot; to the file testb</td>
</tr>
</tbody>
</table>

Example for `<record_range>`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>; display trace listing Trace.List</td>
<td></td>
</tr>
<tr>
<td>; save trace contents between record -107032. and record -21243. to the file testr Trace.SAVE testr (-107032.)--(-21243.)</td>
<td></td>
</tr>
</tbody>
</table>

Example for `<record>` `<record2>`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace.List</td>
<td>display trace listing</td>
</tr>
<tr>
<td>Trace.SAVE testv -107032. -21243.</td>
<td>save trace contents between record -107032. and record -21243. to the file testv</td>
</tr>
</tbody>
</table>

Example for `<timerange>`

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>; display trace listing Trace.List TIme.Zero DEFault</td>
<td></td>
</tr>
<tr>
<td>; save trace contents between the point of time 4.us and the point of time 1.952ms to the file testt Trace.SAVE testt 4.us--1.952ms</td>
<td></td>
</tr>
</tbody>
</table>
More Examples

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace.SAVE test4</td>
<td>save trace contents to the file test4</td>
</tr>
<tr>
<td>QUIT</td>
<td>end TRACE32</td>
</tr>
<tr>
<td></td>
<td>off-line postprocessing of the trace contents e.g. with a TRACE32 Instruction Set Simulator</td>
</tr>
<tr>
<td>Trace.LOAD test4</td>
<td>load the saved trace contents</td>
</tr>
<tr>
<td>Data.LOAD.Elf demo.elf /NoCODE</td>
<td>load the symbol information if you like to have HLL information for the trace analysis</td>
</tr>
<tr>
<td>Trace.List</td>
<td>display the loaded trace contents as trace listing</td>
</tr>
<tr>
<td>Trace.STATistic.Func</td>
<td>perform a function run-time analysis on the loaded trace contents</td>
</tr>
<tr>
<td></td>
<td>HLL information is required</td>
</tr>
</tbody>
</table>

; save trace contents to the file test4
Trace.SAVE test4

... ; use saved trace contents as reference

; load the saved trace contents
Trace.FILE test4

; display the trace contents loaded from the file test4.ad as trace listing
Trace.List /FILE

; compare the current trace contents with the trace contents loaded from test4.ad with regards to the addresses
Trace.ComPare (-27093.)--(-8986.) Address /FILE

See also

- `<trace>.EXPORT`
- `<trace>.LOAD`
- `<trace>.state`
- `Trace`
- `<trace>.STREAMSAVE`

▲ 'Release Information' in 'Release History'
▲ 'Emulator Functions' in 'FIRE User's Guide'

©1989-2019 Lauterbach GmbH
Select trigger/counter line

Format: `Port.Select [channel]`

Selects a port line for the frequency counter and the glitch detector. If in the "TSelect" area of the state window "Port" is active, this port will be used to trigger the port analyzer.

For valid channel names refer to the:

- Processor Architecture Manuals
- Target Guides FIRE
- Target Guides ICE

Examples:

```
Port.Select Port.20
Port.Select Port.RXD
```

Automatic restart of trace recording

Format: `<trace>.SelfArm [ON | OFF]`

`Trace.SelfArm ON` automatically restarts the trace recording. There are mainly two use cases for this command.

Snapshot without stopping the program execution

If stopping the program execution is not advisable, but you are interested in the target state at a specific point of the program execution, proceed as follows:

1. Display the information of interest on the screen.
   Please make sure to display only information that can be updated while the program execution is running.
2. Use a trigger to specify your point of interest.
3. Activate the self-arm mode.

Whenever the trace recording is stopped by the trigger, all information displayed by TRACE32 is updated before the trace recording is automatically restarted.
Example for ARM11:

```
Data.LOAD.Elf arml.a.Elf /PlusVM ; load source code to virtual
; memory within TRACE32 in order to
; enable the trace display while
; program execution is running

Trace.List                     ; display the information of
                                ; interest

Break.Set sieve /TraceTrigger  ; specify the trigger point

Trace.Mode AutoInit ON         ; clear the trace buffer and
; re-activate the trigger
; before the trace recording
; is automatically restarted

Trace.Mode SelfArm ON          ; activate the self-arm mode

Go
...

Trace.Mode SelfArm OFF         ; stop automatic restarting of
; trace recording
```

Automated run-time analysis

To automate an incremental run-time analysis, proceed as follows:

1. Prepare the run-time analysis and open a run-time analysis window.
2. Switch the trace to Leash mode.
3. Activate the self-arm mode.

Whenever the trace recording/program execution is stopped because the trace buffer is full, the current trace contents is analyzed and the analysis window is updated correspondingly. Afterwards the program execution is restarted.
### Example:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Trace.STATistic.Func /Accumulate</td>
<td>open a window that performs a continuous nested function run-time analysis</td>
</tr>
<tr>
<td>Trace.Mode Leash</td>
<td>switch the trace to Leash mode</td>
</tr>
<tr>
<td>Trace.Mode AutoInit ON</td>
<td>clear the trace buffer before the trace recording is automatically restarted</td>
</tr>
<tr>
<td>Trace.Mode SelfArm ON</td>
<td>activate the self-arm mode</td>
</tr>
<tr>
<td>Trace.Mode SelfArm OFF</td>
<td>stop automatic restarting of trace recording</td>
</tr>
<tr>
<td>Go</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

#### See also
- `<trace>.SnapShot`
- `<trace>.state`
- `Trace`

---

### `<trace>.SET`

**Select line for recording**

<table>
<thead>
<tr>
<th>Format: Port.SESelect [channel]</th>
</tr>
</thead>
</table>

Selects a port line for recording.
The **Trace.ShowFocus** command displays the data eyes as they are “seen” by a preprocessor with AUTOFOCUS technology resulting from the following commands:

- **Trace.AutoFocus**
- **Trace.TestFocus**

### Description of Buttons in the Trace.ShowFocus Window

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setup …</td>
<td>Open <strong>Trace.state</strong> window to configure the trace.</td>
</tr>
<tr>
<td>Scan</td>
<td>Perform a <strong>Trace.TestFocus</strong> scan.</td>
</tr>
<tr>
<td>Scan+</td>
<td>Perform a <strong>Trace.TestFocus /Accumulate</strong> scan.</td>
</tr>
<tr>
<td>AutoFocus</td>
<td>Perform a <strong>Trace.AutoFocus</strong> scan.</td>
</tr>
<tr>
<td>Data</td>
<td>Open a <strong>Trace.ShowFocusEye</strong> window.</td>
</tr>
<tr>
<td>Clock</td>
<td>Open a <strong>Trace.ShowFocusClockEye</strong> window.</td>
</tr>
<tr>
<td>Store …</td>
<td>Save the current AUTOFOCUS configuration to a file (<strong>STOre &lt;file&gt; AnalyzerFocus</strong>).</td>
</tr>
<tr>
<td>Load …</td>
<td>Load an AUTOFOCUS configuration from a file (<strong>DO &lt;file&gt;</strong>).</td>
</tr>
<tr>
<td>🔄</td>
<td>Move all sampling points $1 \ast &lt;time_clock&gt;$ to the left.</td>
</tr>
<tr>
<td>🔄</td>
<td>Move all sampling points $1 \ast &lt;time_clock&gt;$ to the right.</td>
</tr>
</tbody>
</table>
Description of the Trace.ShowFocus Window

In the Trace.ShowFocus window the data eyes are white, whereas setup violations are marked as follows:

| Setup violation on the rising edge | High red line |
| Setup violation on the falling edge | Low red line |
| Setup violation on both edges       | Grey bar     |

The x-axis of the Trace.ShowFocus window corresponds to the time axis, whereas the y-axis corresponds to the data channels of the ETM trace port. In the example above we have an 8-bit ETMv1.x architecture with the channels TRACESYNC (TS), PIPESTAT (PS[3:0]) and TRACEPKT (TP[7:0]).
A preprocessor with AUTOFOCUS technology has programmable delays for the clock channel as well as all data channels. With that in mind the x-axis (time-axis) has the following meaning:

<table>
<thead>
<tr>
<th>Data delay greater than clock delay</th>
<th>Negative value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Both clock and data delay are zero</td>
<td>Zero</td>
</tr>
<tr>
<td>Clock delay greater than data delay</td>
<td>Positive value</td>
</tr>
</tbody>
</table>

In the example above there is a channel to channel skew of more than 1 ns that has been compensated by choosing individual sampling points for each data channel. The time resolution for clock and data channel adjustment is not necessarily the same. In the example the time resolution for data channel adjustment is relatively coarse (500-600 ps), whereas the clock channel can be adjusted in fine delay steps (78 ps). The actual values are functions of voltage, temperature and process. However they are measured for each `Trace.AutoFocus` or `Trace.TestFocus` execution, so the numbers displayed in the `Trace.ShowFocus` window do have a physical meaning (time unit is ns).

The example shows the `Trace.ShowFocus` window as it might appear when using the LA-7991 OTP (see `Preprocessor for ARM-ETM AutoFocus` for details). For the re-programmable version both clock and data delays are typically 270 ps and the `Trace.ShowFocus` window for the same application might look like this:

![Trace.ShowFocus window](image)

**NOTE:** The NEXUS AutoFocus adapter does not support this feature.

See also

- `<trace>.ShowFocusClockEye`
- `<trace>.AutoFocus`
- `<trace>.TestFocus`
- `Trace`
- `<trace>.TestFocusClockEye`
The command `Trace.ShowFocusClockEye` displays a graph reflecting the clock waveform. Basically it shows data eyes from a different point of view.

The result of the command `Trace.ShowFocusClockEye` shows a single ETM channel or all ETM channels superimposed. Further are:

- X-axis: time range [ns]
- Y-axis: voltage range [V]

### Color Legend

<table>
<thead>
<tr>
<th>Area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>White area</td>
<td>Data eye.</td>
</tr>
<tr>
<td>Green area</td>
<td>Setup violation on the rising edge.</td>
</tr>
<tr>
<td>Red area</td>
<td>Setup violation on the falling edge.</td>
</tr>
<tr>
<td>Superimposed area (green and red)</td>
<td>Setup violation on both edges.</td>
</tr>
</tbody>
</table>
Description of Buttons in the <trace>.ShowFocusClockEye Window

<table>
<thead>
<tr>
<th>Setup …</th>
<th>Open <strong>Trace.state</strong> window to configure the trace.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scan</td>
<td>Perform a <strong>Trace.TestFocusEye</strong> scan.</td>
</tr>
<tr>
<td>Scan+</td>
<td>Perform a <strong>Trace.TestFocusEye /Accumulate</strong> scan.</td>
</tr>
<tr>
<td>AutoFocus</td>
<td>Perform a <strong>Trace.AutoFocus</strong>.</td>
</tr>
<tr>
<td>Digital</td>
<td>Open a <strong>Trace.ShowFocus</strong> window scan.</td>
</tr>
<tr>
<td>Channel (previous)</td>
<td>Display <strong>Trace.ShowFocusClockEye</strong> for a single trace line (previous).</td>
</tr>
<tr>
<td>Channel (next)</td>
<td>Display <strong>Trace.ShowFocusClockEye</strong> for a single trace line (next).</td>
</tr>
<tr>
<td>←</td>
<td>Move all sampling points 1 * <code>&lt;time_clock&gt;</code> to the left.</td>
</tr>
<tr>
<td>→</td>
<td>Move all sampling points 1 * <code>&lt;time_clock&gt;</code> to the right.</td>
</tr>
</tbody>
</table>

Examples

```
Trace.ShowFocusEye                              ; Display data eye with
                                                   ; all trace channels superimposed
Trace.ShowFocusEye PS2                           ; Display data eye for the
                                                   ; trace channel PS2
```

**NOTE:** The NEXUS AutoFocus adapter does not support this feature.

See also

- **<trace>.ShowFocus**
- **<trace>.ShowFocusEye**
- **<trace>.AutoFocus**
- **<trace>.state**
- **<trace>.TestFocus**
- **<trace>.TestFocusClockEye**

▲ ‘Diagnosis’ in ‘AutoFocus User’s Guide’
▲ ‘Diagnosis’ in ‘ARM-ETM Trace’
The command `Trace.ShowFocusEye` displays the data eye as it is 'seen' by a preprocessor with AUTOFOCUS technology or PowerTrace Serial resulting from the command `Trace.TestFocusEye`.

The result of the command `Trace.ShowFocusEye` shows a single trace channel or all trace channels superimposed. The unit of the axis differs for AUTOFOCUS:

- X-axis: time range [ns] or [UI]
- Y-axis: voltage range [V] or [percentage]

And PowerTrace Serial technology:

- X-axis: time range [UI]
- Y-axis: voltage range [percentage of eye height]
### Color Legend for AUTOFOCUS

<table>
<thead>
<tr>
<th>Color Area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>White area</td>
<td>Stable data.</td>
</tr>
<tr>
<td>Green area</td>
<td>Setup violation on the rising edge.</td>
</tr>
<tr>
<td>Red area</td>
<td>Setup violation on the falling edge.</td>
</tr>
<tr>
<td>Superimposed area (green and red)</td>
<td>Setup violation on both edges.</td>
</tr>
</tbody>
</table>

### Color Legend for PowerTrace Serial

<table>
<thead>
<tr>
<th>Color Area</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>White area</td>
<td>Stable data eye.</td>
</tr>
<tr>
<td>Black areas</td>
<td>Unstable data.</td>
</tr>
<tr>
<td>Grey/Black areas</td>
<td>Unstable data.</td>
</tr>
<tr>
<td>Dark Blue area</td>
<td>Stable data eye.</td>
</tr>
<tr>
<td>Blue/Green/Orange/</td>
<td>Unstable data.</td>
</tr>
<tr>
<td>Yellow/Red areas</td>
<td></td>
</tr>
</tbody>
</table>
Descriptions of Buttons in the `<trace>.ShowFocusEye` Window:

| Setup ... | Open `Trace.state` window to configure the trace. |
| Scan      | Perform a `Trace.TestFocusEye` scan. |
| Scan+     | Perform a `Trace.TestFocusEye/Accumulate` scan. |
| AutoFocus | Perform a `Trace.AutoFocus` scan. |
| Digital   | Open a `Trace.ShowFocus` window. |
| Channel (previous) | Display `Trace.ShowFocusEye` for a single trace line (previous). |
| Channel (next) | Display `Trace.ShowFocusEye` for a single trace line (next). |
|           | Move all sampling points 1 * `<time_clock>` to the left. |
|           | Move all sampling points 1 * `<time_clock>` to the right. |

Examples

- `Trace.ShowFocusEye` ; Display data eye with ; all trace channels superimposed
- `Trace.ShowFocusEye PS2` ; Display data eye for the ; trace channel PS2

**NOTE:** The NEXUS AutoFocus preprocessor does not support this feature.

See also
- `<trace>.ShowFocus`
- `<trace>.AutoFocus`
- `<trace>.TestFocusEye`
- `<trace>.TestFocus`
- `<trace>.ShowFocusClockEye`
- `<trace>.state`
- `Trace`
- `<trace>.TestFocusClockEye`
Define buffer size

Format: `<trace>.SIZE [<size>]`

Sets the `<size>` of trace memory which is used for trace recording. If the command is called with size zero, the trace size will be set to its maximum size.

Reducing the size used for trace recording helps to reduce time needed for trace data download and trace analysis (statistical analysis, trace chart display etc., searching for an event in the trace), because of the smaller amount of recorded data. There is no other benefit besides that.

The maximum trace size for the **Advanced Register Trace (ART) and ICE/FIRE** port and state analyzers is fixed:

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ART</td>
<td>65535.</td>
</tr>
<tr>
<td>ICE State Analyzer</td>
<td>32767.</td>
</tr>
<tr>
<td>ICE Port Analyzer</td>
<td></td>
</tr>
<tr>
<td>FIRE State Analyzer</td>
<td>65535.</td>
</tr>
<tr>
<td>FIRE Port Analyzer</td>
<td></td>
</tr>
<tr>
<td>FIRE State Analyzer 512K</td>
<td>524288.</td>
</tr>
<tr>
<td>FIRE Port Analyzer 512K</td>
<td></td>
</tr>
</tbody>
</table>

The maximum trace size for logic analyzers **PowerProbe and PowerIntegrator** depends on module memory size and sample frequency:

<table>
<thead>
<tr>
<th>Module / Mode</th>
<th>&lt;= 100MHz</th>
<th>200MHz</th>
<th>400MHz</th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerProbe 128K (LA-7930)</td>
<td>131071.</td>
<td>262142.</td>
<td>524284.</td>
</tr>
<tr>
<td>PowerProbe 256k (LA-7931)</td>
<td>262143.</td>
<td>524286.</td>
<td>1048572.</td>
</tr>
<tr>
<td>PowerIntegrator (LA-7940)</td>
<td>524287.</td>
<td>1048576.</td>
<td>2097152.</td>
</tr>
</tbody>
</table>
The maximum trace size for trace modules PowerTrace and RiscTrace depends on the module’s memory size and a factor depending on the trace protocol:

<table>
<thead>
<tr>
<th>Module</th>
<th>Number of trace records</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICD RiscTrace (LA-7859, LA-7870)</td>
<td>65536</td>
</tr>
<tr>
<td>PowerTrace 256MB (LA-7707)</td>
<td>16777216</td>
</tr>
<tr>
<td>PowerTrace 512MB (LA-7690)</td>
<td>33554432</td>
</tr>
<tr>
<td>PowerTrace II 1G (LA-7692)</td>
<td>67108864</td>
</tr>
<tr>
<td>PowerTrace II 2G (LA-7693)</td>
<td>134217728</td>
</tr>
<tr>
<td>PowerTrace II 4G (LA-7694)</td>
<td>268435456</td>
</tr>
</tbody>
</table>

Trace sizes for several trace protocols:

<table>
<thead>
<tr>
<th>Trace protocol</th>
<th>Factor (size = factor * records)</th>
<th>Trace size PowerTrace 512MB</th>
<th>Trace size PowerTrace II 4G</th>
</tr>
</thead>
<tbody>
<tr>
<td>ETM</td>
<td>4.</td>
<td>134217728</td>
<td>1073741824</td>
</tr>
<tr>
<td>NEXUS</td>
<td>1.</td>
<td>33554432</td>
<td>268435456</td>
</tr>
</tbody>
</table>

See also

- `<trace>.state`
- `Trace`
- `CIProbe`
- `Analyzer.RECORDS()`
- `Analyzer.SIZE()`
- 'Emulator Functions’ in ‘FIRE User’s Guide’
- 'Release Information’ in ‘Release History’

**<trace>.SLAVE**

Select slave mode

Format: `<trace>.SLAVE`

The trace memory of the port analyzer is used as extension to the SA120 or HA120 state analyzer unit. All other setups of the port analyzer are not valid.
Restart the trace capturing. Effectively the same as executing the commands `Trace.OFF`, `Trace.Init` and `Trace.Arm`.

Most often used to restart the trace recording:

- After the trace capturing was stopped by a trigger (e.g. by a TraceTrigger breakpoint).
- When the trace works in **Stack** mode and trace capturing was stopped because the trace buffer was full.

**See also**
- `<trace>.SelfArm`
- `<trace>.state`
- `Trace`

---

**<trace>.SPY**

Enable analysis of streaming file while recording

**Format:**

```
<trace>.SPY
```

`Trace.SPY` allows to analyze and display intermediate results while streaming (see `<trace>.Mode STREAM`).

**Example:**

```
...  
Trace.Mode STREAM ; select trace mode STREAM
Go ; start the program execution
...
Trace.SPY ; enable analysis of streaming file
Trace.Chart.sYmbol
Trace.Arm ; switch back to standard recording
...
```
SPY in the Trace state line indicates that the Trace is in SPY state. The number in the used field indicates now the number of records in the streaming file that can be analyzed. Since TRACE32 continues with streaming in SPY state, number is updated to the current number of records in the streaming file when you switch back to Arm state.

See also
- Trace
Displays the trace configuration window. The trace methods are displayed at the top of the window. The configuration options below the trace methods adjust to the currently selected trace method, compare screenshot on the left with the screenshot on the right.

A For descriptions of the commands in the **<trace>.state** window, please refer to the **<trace>.* commands in this chapter.** **Example:** For information about **OFF**, see **<trace>.OFF**.

B After you have selected the desired trace method (**Trace.METHOD**), you can work with the commands that start with **Trace**. This principle is illustrated in the two PRACTICE script snippets below.

**Trace.METHOD Analyzer** ;Select the trace method, here Analyzer
<your_code>
**Trace.List** ;The trace listing now refers to the ;method Analyzer
<your_code>

**Trace.METHOD SNOOPer** ;Select the trace method, here SNOOPer
<your_code>
**Trace.List** ;The trace listing now refers to the method SNOOPer
<your_code>
Using this command the operating mode of the analyzer may be selected. During operation this command
displays the current state of the analyzer.

<table>
<thead>
<tr>
<th>State Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>Displays the current port analyzer state.</td>
</tr>
</tbody>
</table>

| used |
| Displays the current recording depth in graphical format. Recording depth display is also updated during analyzer activation, but only in steps of 512 entries. If the sampling is stopped, the correct value will be displayed. |

<table>
<thead>
<tr>
<th>State Field</th>
</tr>
</thead>
<tbody>
<tr>
<td>SLAVE The analyzer is in slave mode to the SA120/HA120 unit.</td>
</tr>
<tr>
<td>OFF Indicates that trace memory is deactivated and no trace is done. Memory can be read out.</td>
</tr>
<tr>
<td>ARM Indicates that the analyzer is activated (for sampling and triggering events). It cannot be read out.</td>
</tr>
<tr>
<td>trigger Indicates that the specified trigger event has been encountered.</td>
</tr>
<tr>
<td>break Indicates that the specified trigger event has been encountered and trigger delay time has elapsed. At the same time, the analyzer is set to OFF mode.</td>
</tr>
</tbody>
</table>
Ports

Displays continuously the state of the ports. The order is physically (logically depending on the emulated microcontroller) just as the order, used in the list command.

With a mouse click to the corresponding area all commands can be executed or selects can be switched on or off.

See also

- `<trace>.ACCESS`
- `<trace>.Arm`
- `<trace>.AutoFocus`
- `<trace>.AutoStart`
- `<trace>.BookMark`
- `<trace>.ComPare`
- `<trace>.CustomTraceLoad`
- `<trace>.DisConfig`
- `<trace>.EXPORT`
- `<trace>.Find`
- `<trace>.FindChange`
- `<trace>.FLOWSTART`
- `<trace>.GOTO`
- `<trace>.List`
- `<trace>.ListVar`
- `<trace>.METHOD`
- `<trace>.OFF`
- `<trace>.PC`
- `<trace>.PortType`
- `<trace>.PROfileChart`
- `<trace>.Program`
- `<trace>.REF`
- `<trace>.RESet`
- `<trace>.SAVE`
- `<trace>.ShowFocus`
- `<trace>.ShowFocusEye`
- `<trace>.SnapShot`
- `<trace>.TDelay`
- `<trace>.TestFocus`
- `<trace>.TThreshold`
- `<trace>.TraceCONNECT`
- `<trace>.View`
- `<trace>.ZERO`
- Trace.METHOD()
- Trace.STATE()

▲ 'Trace Functions’ in ‘General Function Reference’
▲ 'Displaying the Trace’ in ‘Training FIRE Analyzer’
▲ 'HCS12 On-chip Debug Module’ in ‘FIRE Emulator for HC12/MCS12’
▲ 'Emulator Functions’ in ‘FIRE User’s Guide’
▲ 'Release Information’ in ‘Release History’

©1989-2019 Lauterbach GmbH
The `<trace>.STATistic` commands can be used for statistical analysis based on the information sampled to the trace buffer.

In contrast to the performance analyzer (PERF commands), the statistical analysis commands provide a higher precision and much more information about the analyzed item, but since the size of the trace buffer is limited, the observation time is limited. Statistic evaluations can be made after the trace memory stops sampling. For short calculation time minimize the trace buffer size.

See also

- `<trace>.state`
- `<trace>.Chart`
- `<trace>.PROfileSTATistic`
- `<trace>.STATistic.AddressDIStance`
- `<trace>.STATistic.BondOut`
- `<trace>.STATistic.COLOR`
- `<trace>.STATistic.DatasYmbol`
- `<trace>.STATistic.DistriB`
- `<trace>.STATistic.FIRST`
- `<trace>.STATistic.FuncDURation`
- `<trace>.STATistic.GROUP`
- `<trace>.STATistic INTERRUPT`
- `<trace>.STATistic.LAST`
- `<trace>.STATistic.LINKage`
- `<trace>.STATistic.Measure`
- `<trace>.STATistic.PAddress`
- `<trace>.STATistic.PIPELINE`
- `<trace>.STATistic.PROGRAM`
- `<trace>.STATistic.Sort`
- `<trace>.STATistic.TASK`
- `<trace>.STATistic.TASKKernel`
- `<trace>.STATistic.TASKSRV`
- `<trace>.STATistic.TASKTREE`
- `<trace>.STATistic.TREE`

- Trace
- `<trace>.PROfileChart`
- `<trace>.STATistic.AddressDURation`
- `<trace>.STATistic.ChildTREE`
- `<trace>.STATistic.CYcle`
- `<trace>.STATistic.DISTance`
- `<trace>.STATistic.DUration`
- `<trace>.STATistic.Func`
- `<trace>.STATistic.FuncDURationInternal`
- `<trace>.STATistic.Ignore`
- `<trace>.STATistic.InterruptIsFunction`
- `<trace>.STATistic.Line`
- `<trace>.STATistic.LISTCONFIG`
- `<trace>.STATistic.MODULE`
- `<trace>.STATistic.ParentTREE`
- `<trace>.STATistic.PREFetch`
- `<trace>.STATistic.PsYmbol`
- `<trace>.STATistic.sYmbol`
- `<trace>.STATistic.TASKFunc`
- `<trace>.STATistic.TASKORINTERRUPT`
- `<trace>.STATistic.TASKState`
- `<trace>.STATistic.TASKVSINTERRUPT`
- `<trace>.STATistic.Use`

Analyzer.RECORDS()

Analyzer.SIZE()
**Format:**

```
<trace>.STATistic.Address <address1> [<address1> ...]
```

Displays the time interval between up to 8 program events.

Trace.STATistic.Address sieve response buffer

Trace.STATistic.Address 0x125c 0x1264 0x1274 0x1290 0x12ac 0x12b8 0x12d8

Analysis background:

```
address1
address2
address2
address3
address1
address1
address2
```

**See also**

- `<trace>.STATistic`
- 'Release Information' in 'Release History'
Display the time interval for a single program event. Without parameter the assignment of classes (16) is done automatically. With arguments the classes can be set up manually.

The following 2 commands are equivalents:

```plaintext
B::Trace.STATistic.AddressDISTance InterruptEntry
B::Trace.STATistic.DISTance /FILTER Address InterruptEntry
```

If selective tracing is possible, use the `/TraceEnable` filter to extend the observation time:

```plaintext
B::Break.Set InterruptEntry /Program /TraceEnable
B::Go
B::Break
B::Trace.STATistic.AddressDISTance InterruptEntry
```

In the case of an SMP system, the following options are provided:

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>JoinCORE</strong></td>
<td>(default) Analysis is performed for all cores. The core information is discarded.</td>
</tr>
<tr>
<td><strong>SplitCORE</strong></td>
<td>Same as JoinCORE.</td>
</tr>
<tr>
<td><strong>MergeCORE</strong></td>
<td>Same as JoinCORE.</td>
</tr>
<tr>
<td><strong>CORE &lt;number&gt;</strong></td>
<td>Analysis is performed for the specified core.</td>
</tr>
</tbody>
</table>

See also
- `<trace>.STATistic`
- `<trace>.STATistic.DISTance`
- 'Release Information’ in 'Release History’
The statistic distribution between two program events is analyzed. This command can be used to analyze the run-time of a single function or interrupt response times.

; Analyze the run-time of a single function
; func9: start address of the function
; sYmbol.EXIT(func9): Exit address of the given function
Trace.STATistic.AddressDURation func9 sYmbol.EXIT(func9)

By default TRACE32 PowerView builds 16 result classes. For a graphical display of the results use the command Trace.PROfileChart.DURation.
The `<option>` **Number** allows a user-defined number of result classes.

```
Trace.STATistic.AddressDURation func9 sYmbol.EXIT(func9) /Number 6.
```

The parameter `<timemin>` allows to specify the time for the first result class, the parameter `<inc>` allows to specify the increment for the next result class.

```
Trace.STATistic.AddressDURation func9 sYmbol.EXIT(func9) 15.us 1.us
```

Trace filter allow a more effective usage of the trace memory:

```
Trace.Mode Leash

Break.Set func9 /Program /TraceEnable
Break.Set sYmbol.EXIT(func9) /Program /TraceEnable
Go
WAIT !STATE.RUN()
Trace.STATistic.AddressDURation func9 sYmbol.EXIT(func9)
```

In the case of an SMP system, the following options are provided:

<table>
<thead>
<tr>
<th><strong>JoinCORE</strong> (default)</th>
<th>Analysis is performed for all cores. The core information is discarded.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SplitCORE</strong></td>
<td>Same as JoinCORE.</td>
</tr>
<tr>
<td><strong>MergeCORE</strong></td>
<td>Same as JoinCORE.</td>
</tr>
<tr>
<td><strong>CORE &lt;number&gt;</strong></td>
<td>Analysis is performed for the specified core.</td>
</tr>
</tbody>
</table>

**See also**

- `<trace>.STATistic`

  ▲ 'Release Information' in 'Release History'
**<trace>.STATistic.BondOut**  
Bondout mode

| Format: | <trace>.STATistic.BondOut [ON | OFF] |

If ON is selected, the software will use the bondout busses to capture the function entry and exit data. This command is only available on some emulation targets.

**See also**
- <trace>.STATistic

---

**<trace>.STATistic.ChildTREE**  
Show callee context of a function

| Format: | <trace>.STATistic.ChildTREE <address> |

Show call tree and run-time of all functions called by the specified function. The function is specified by its start <address>.

**Example:**

Trace.STATistic.ChildTREE master_selection

---

See also
- <trace>.STATistic
- <trace>.STATistic.ParentTREE
**<trace>.STATistic.COLOR**

Assign colors to function for colored graphics

Format:  

<trace>.STATistic.COLOR FixedColors | AlternatingColors

<table>
<thead>
<tr>
<th><strong>FixedColors</strong> (default)</th>
<th>Colors are assigned fixed to functions.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AlternatingColors</strong></td>
<td>Colors are assigned by the recording order of the functions for each measurement.</td>
</tr>
</tbody>
</table>

See also
- <trace>.STATistic
- 'PowerView - Screen Display' in 'PowerView User's Guide'

---

**<trace>.STATistic.CYcle**

Analyze cycle types

Format:  

<trace>.STATistic.CYcle [<timerange>] [/<option>]

<option>:
- FILE | Accumulate
- INCremental | FULL
- IdleThreshold <clocks>
- CORE <number> | SplitCORE | MergeCORE | JoinCORE (SMP tracing only)

Performs a statistical analysis of the cycle types.

In the case of an SMP system, the following options are provided:

<table>
<thead>
<tr>
<th><strong>MergeCORE</strong> (default)</th>
<th>Analysis is performed independently for each core. The results are summarized and displayed as a single result.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SplitCORE</strong></td>
<td>Same as MergeCORE.</td>
</tr>
<tr>
<td><strong>JoinCORE</strong></td>
<td>Same as MergeCORE.</td>
</tr>
<tr>
<td><strong>CORE &lt;number&gt;</strong></td>
<td>Analysis is performed for the specified core.</td>
</tr>
</tbody>
</table>
Example based on CoreSight ETMv3 for a Cortex-R4:

```plaintext
ETM.DataTrace ON ; full data trace
ETM.CycleAccurate ON ; cycle accurate tracing
Trace.CLOCK 450.MHz ; inform TRACE32 about the core; clock
```

### survey

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>records</td>
<td>Number of records in the trace</td>
</tr>
<tr>
<td>time</td>
<td>Time period recorded by the trace</td>
</tr>
<tr>
<td>clocks</td>
<td>Number of clock cycles recorded by the trace</td>
</tr>
<tr>
<td></td>
<td>(cycle-accurate tracing only)</td>
</tr>
<tr>
<td>flow cycles</td>
<td>Number of flow cycles</td>
</tr>
<tr>
<td></td>
<td>(flow execute + flow read + flow write)</td>
</tr>
<tr>
<td>bus cycles</td>
<td>Number of data read and data write cycles</td>
</tr>
<tr>
<td>cpi</td>
<td>Average clocks per instruction</td>
</tr>
<tr>
<td></td>
<td>(clocks/instr)</td>
</tr>
<tr>
<td></td>
<td>(cycle-accurate tracing only)</td>
</tr>
<tr>
<td><strong>details</strong></td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>flow execute</strong></td>
<td>Number of cycles that executed/not executed instructions</td>
</tr>
<tr>
<td><strong>flow read</strong></td>
<td>Number of cycles that performed a read access</td>
</tr>
<tr>
<td><strong>flow write</strong></td>
<td>Number of cycles that performed a write access</td>
</tr>
<tr>
<td><strong>bus fetch</strong></td>
<td>—</td>
</tr>
<tr>
<td><strong>bus read</strong></td>
<td>Number of data read cycles</td>
</tr>
<tr>
<td><strong>bus write</strong></td>
<td>Number of data write cycles</td>
</tr>
<tr>
<td><strong>instr</strong></td>
<td>Number of executed/not executed instruction</td>
</tr>
<tr>
<td><strong>slot instr</strong></td>
<td>—</td>
</tr>
<tr>
<td><strong>fail cond</strong></td>
<td>Number of conditional instructions that failed (failed branch instructions not included)</td>
</tr>
<tr>
<td><strong>pass cond</strong></td>
<td>Number of conditional instructions that passed (taken branches not included)</td>
</tr>
<tr>
<td><strong>fail branch</strong></td>
<td>Number of failed branch instructions</td>
</tr>
<tr>
<td><strong>dir branch</strong></td>
<td>Number of direct branches</td>
</tr>
<tr>
<td><strong>indir branch</strong></td>
<td>Number of indirect branches</td>
</tr>
<tr>
<td><strong>load instr</strong></td>
<td>Number of load instructions</td>
</tr>
<tr>
<td><strong>store instr</strong></td>
<td>Number of store instructions</td>
</tr>
<tr>
<td><strong>modify instr</strong></td>
<td>Number of modify instructions (swp, swpb)</td>
</tr>
<tr>
<td><strong>traps</strong></td>
<td>Number of traps</td>
</tr>
<tr>
<td><strong>interrupts</strong></td>
<td>Number of interrupts</td>
</tr>
<tr>
<td><strong>idles</strong></td>
<td>Number of “wait for interrupt” (coprocessor instruction or WFI instruction) or number of times that 1000. clock cycles passed without a broadcast of trace information. The option <strong>IdleThreshold</strong> allows to modify the number of clock cycles that need to pass for a idle detection.</td>
</tr>
<tr>
<td><strong>trace gaps</strong></td>
<td>Number of trace gaps (FIFOFULL, filtered trace information …)</td>
</tr>
</tbody>
</table>
The command `Trace.STATistic.DatasYmbol` analyzes the contents of a pointer numerically.

If a full program and data trace is analyzed, the following command is recommended:

```
; analyze the contents of the pointer vpchar numerically
Trace.STATistic.DatasYmbol /Filter Address vpchar
```
A more effective usage of the trace memory is possible, if only write accesses to the pointer are recorded in the trace.

```
; set a filter to record only write cycles to the pointer vpchar to the
; trace
Var.Break.Set vpchar /Write /TraceEnable

...

; analyze the contents of the pointer
Trace.STATistic.DatasYmbol

; analyze the contents of the pointer, sort the result by symbol names
Trace.STATistic.DatasYmbol /Sort sYmbol
```

See also
- `<trace>.STATistic`
- 'Release Information' in 'Release History'
### Format

<trace>.STATistic.DISTance [<timemin>] [<inc>] [/<option>]

#### Option:

- FILE
- FlowTrace | BusTrace
- Accumulate
- INCremental | FULL
- Sort <item>
- Track
- NoMerge
- Filter <item>
- Number <number>
- LOG
- LiNEar

Displays the time interval for a single event. Without parameter the assignment of classes (16) is done automatically. With arguments the classes can be set up manually.

| E::Trace.STATistic.DISTance samples: 73. avr: 39.636us min: 0.500us max: 461.125us |
|---------------------------------|---------|-------------------------------|--------|--------|--------|--------|--------|--------|--------|
|                                  | up to   | count | ratio | 1% | 2% | 5% | 10% | 20% | 50% | 10 |
| < 0.000                          | 0.000  | 0.000% |       |    |    |    |    |    |    |    |
| 40.960us                         | 55.754| 75.342% |       |    |    |    |    |    |    |    |
| 81.920us                         | 10.137| 13.698% |       |    |    |    |    |    |    |    |
| 122.880us                        | 1.136| 1.369% |       |    |    |    |    |    |    |    |
| 163.840us                        | 2.274| 2.740% |       |    |    |    |    |    |    |    |
| 204.800us                        | 2.274| 2.740% |       |    |    |    |    |    |    |    |
| 245.760us                        | 0.000| 0.000% |       |    |    |    |    |    |    |    |
| 286.720us                        | 1.136| 1.369% |       |    |    |    |    |    |    |    |
| 327.680us                        | 0.000| 0.000% |       |    |    |    |    |    |    |    |

### Basic <options>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Displays trace memory contents loaded with Trace.FILE.</td>
</tr>
<tr>
<td>FlowTrace</td>
<td>The trace works as flow trace. This option is usually not required.</td>
</tr>
<tr>
<td>BusTrace</td>
<td>The trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td>Accumulate</td>
<td>By default only the current trace contents is analyzed by the statistic functions. The option /Accumulate allows to add the current trace contents to the already displayed results.</td>
</tr>
<tr>
<td>INCremental</td>
<td>Intermediate results are displayed while the TRACE32 software analyses the trace contents (default).</td>
</tr>
</tbody>
</table>
Example for TRACE32-ICD and TRACE32-PowerTrace:

If no selective tracing is possible, use the option /Filter to filter out the event of interest.

B::Go
B::Break
B::Trace.STATistic.DISTance /FILTER Address InterruptEntry
B::Trace.STATistic.DISTance 10.ms 0.2ms /FILTER Address InterruptEntry

If selective tracing is possible, use the /TraceEnable filter to extend the observation time:

B::Break.Set InterruptEntry /Program /TraceEnable
B::Go
B::Break
B::Trace.STATistic.DISTance /FILTER Address InterruptEntry
Example for TRACE32-ICE or TRACE32-FIRE:

Perform a selective trace on the event of interest

Analyzer.ReProgram
(
  ADDR AlphaBreak InterruptEntry
  Sample.enable IF AlphaBreak
)
Go

Break

Trace.STATistic.DISTance

; selective trace
; program the analyzer to sample, all InterruptEntry
; measurement
; analyze the time interval; between the sampled; InterruptEntry

See also

■ <trace>.STATistic.AddressDISTance
■ <trace>.STATistic

▲ ‘Release Information’ in ‘Release History’
▲ ‘Statistic Functions’ in ‘Training FIRE Analyzer’
▲ ‘Statistic Functions’ in ‘Training ICE Analyzer’
The statistic distribution of any data is displayed if `<item>` is specified. Displayed are the number of occurrences and the time after the events, i.e. the time an event is assumed to be valid. Without `<item>` the statistic is based on the symbolic addresses.

<table>
<thead>
<tr>
<th>Basic <code>&lt;options&gt;</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Displays trace memory contents loaded with <code>Trace.FILE</code>.</td>
</tr>
<tr>
<td>FlowTrace</td>
<td>The trace works as flow trace. This option is usually not required.</td>
</tr>
<tr>
<td>BusTrace</td>
<td>The trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td>Accumulate</td>
<td>By default only the current trace contents is analyzed by the statistic functions. The option <code>/Accumulate</code> allows to add the current trace contents to the already displayed results.</td>
</tr>
<tr>
<td>INCremental</td>
<td>Intermediate results are displayed while the TRACE32 software analyses the trace contents (default).</td>
</tr>
<tr>
<td>FULL</td>
<td>The result is displayed after the TRACE32 software finished the analysis.</td>
</tr>
<tr>
<td>Track</td>
<td>Track the <code>Trace.STATistic</code> window with other trace list windows (tracking to record number or time possible).</td>
</tr>
<tr>
<td>NoMerge</td>
<td>(For diagnosis purpose only).</td>
</tr>
</tbody>
</table>
**BEFORE**
Display the time before the event. That means how long the previous state lasted until the listed state was reached.

**AFTER**
Display the time after the event. That means how long the state lasted after it was reached (default).

**List <list_items>**
Specify the result that should be displayed in the window.

**Filter <item>**
Filter the described item.

The **List** option defines which values are calculated and displayed. The **<list_items>** can be arranged by pushing the **Config** button in the **Trace.STATistic.Distrib** window.

**<list_item>**

**Time**
Total time the event was true.

**MIN, MAX**
Minimum and maximum time the event was true.

**AVeRage**
Average time the event was true.

**Count**
Number of occurrences of the event.

**Ratio, BAR.log, BAR.LINear**
Ratio of time spent in events to total measurement time in percent and as graphical bars.

**CRatio, CBAR.log, CBAR.LINear**
Ratio of count to total count in percent and as graphical bars.

All time displays depend on the options **AFTER** or **BEFORE**.
Example for TRACE32-ICD and TRACE32-PowerTrace:

If no selective tracing is possible, use the option /Filter to filter out the event of interest.

```
B::Go
B::Break
B::Trace.STATistic.Distrib Data.B /Filter Address V.RANGE(flags[3])
```

Example for TRACE32-ICE or TRACE32-FIRE:

Perform a selective trace on the event of interest.

```
Analyzer.ReProgram
(
    ADDR AlphaBreak V.RANGE(flags[3])
    Sample.enable IF AlphaBreak
)
Go

Break
Trace.Chart.Distrib Data.B
```

; selective trace
; program the analyzer to
; sample all accesses to the
; variable flags[3]

; measurement

; display the time spent in
; different states

```

<table>
<thead>
<tr>
<th>samples: 620.</th>
<th>total: 9.456ms</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>class</td>
<td>time</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------</td>
</tr>
<tr>
<td>d.b=0x0</td>
<td>6.583ms</td>
</tr>
<tr>
<td>d.b=0x1</td>
<td>2.854ms</td>
</tr>
</tbody>
</table>

```

See also

- `<trace>.STATistic`
- ‘Release Information’ in ‘Release History’
### <trace>.STATistic.DURation

Time between two events

**Format:**

```plaintext
<trace>.STATistic.DURation [timemin] [inc] [option]
```

**<option>:**

- **FILE**
  - FlowTrace | BusTrace
  - Accumulate | INcremental | FULL

- **ATOA** | **ATOB** | **ATOC** | **ATOD**
- **BTOA** | **BTOB** | **BTOC** | **BTOD**
- **CTOA** | **CTOB** | **CTOC** | **CTOD**
- **DTOA** | **DTOB** | **DTOC** | **DTOD**

- FilterA <filter> | FilterB <filter>
- Number | LOG | LINear

Analyzes the statistic distribution between two events.

**<option>**

**Basis Options**

- **FILE**
  - Displays trace memory contents loaded with `Trace.FILE`.

- **FlowTrace**
  - The trace works as flow trace. This option is usually not required.

- **BusTrace**
  - The trace works as a bus trace. This option is usually not required.

- **Accumulate**
  - By default only the current trace contents is analyzed by the statistic functions. The option `/Accumulate` allows to add the current trace contents to the already displayed results.

- **INcremental**
  - Intermediate results are displayed while the TRACE32 software analyzes the trace contents (default).

- **FULL**
  - The result is displayed after the TRACE32 software finished the analysis.

- **Number**
  - Define the number of classes.

- **LOG**
  - Display the bars in the result display in a logarithmic format (default).

- **LINear**
  - Display the bars in the result display in a linear format.
Example for TRACE32-PowerTrace

To determine the time interval between two instructions (addresses) \texttt{Trace.STATistic.AddressDURation} is more suitable.

This example analyzes how long it takes when the contents of a variable changes from 0x0 to 0x1.

\begin{verbatim}
Var.Break.Set flags /Write /TraceEnable

Trace.STATistic.DURation /FilterA Data 0x0 /FilterB Data 0x1
\end{verbatim}

In order to use the command \texttt{Trace.STATistic.DURation}:

\begin{itemize}
  \item Check if both events are exported by a trace packet. Information reconstructed by TRACE32 is not analyzed.
  \item Alternatively use a \texttt{TraceEnable} breakpoint export the event as a trace packet.
\end{itemize}

The options \texttt{FilterA} and \texttt{FilterB} provide you with the means to describe your event.

\begin{itemize}
  \item \texttt{FilterA <item>} Specify the first event.
  \item \texttt{FilterB <item>} Specify the second event.
\end{itemize}
By default the time interval between the first event and the second event is displayed \textit{(ATOB)}. This can be changed by the following options:

\begin{verbatim}
Analyzer.ReProgram
(
    ADDR AlphaBreak InterruptEntry
    ADDR BetaBreak InterruptExit
    Sample.enable IF AlphaBreak||BetaBreak
    Mark.A IF AlphaBreak
    Mark.B IF BetaBreak
)

E::Go                ; measurement
E::Break

E::Trace_STATistic_DURation
; analyze time between
; InterruptEntry and
; InterruptExit

E::Trace_STATistic_DURation 40.us 40.us
; analyze the time
; between InterruptEntry
; and InterruptExit, the
; first class should
; start at 40.us and the
; increment to the next
; class is 40.us
\end{verbatim}

\textbf{ATOA} \hspace{1cm} Display the time interval from A to A.

\textbf{BTOA} \hspace{1cm} Display the time interval from B to A.

\textbf{BTOB} \hspace{1cm} Display the time interval from B to B.

\ldots

\textbf{See also}
\begin{itemize}
\item \texttt{<trace>\_STATistic}
\item \texttt{\textasciitilde'}\textit{Statistic Functions}' in 'Training FIRE Analyzer'
\item \texttt{\textasciitilde'}\textit{Statistic Functions}' in 'Training ICE Analyzer'
\end{itemize}
The **Trace.STATistic** commands analyze the complete trace contents by default. The command **Trace.STATistic.FIRST** allows to freely select a start point for the statistic analysis.

**Example for <value>:**

```
Trace.List ; display trace listing

Trace.STATistic.FIRST -123366. ; select trace record -123366.
 ; as start point for the trace
 ; analysis

Trace.STATistic.LAST -36675. ; select trace record -36675.
 ; as end point for the trace
 ; analysis

Trace.STATistic.Func ; perform a function run-time
 ; analysis
```
Example for `<time>`:

```
Trace.List TIme.ZerO DEFault ; display trace listing

Trace.STATistic.FIRST 0.3us ; select trace record with time
; stamp 0.3 µs (zero time)
; as start point for the trace
; analysis

Trace.STATistic.Func ; perform a function run-time
; analysis between the specified
; start point and the end of the
; trace buffer
```

See also

- `<trace>.STATistic`
- `<trace>.STATistic.LAST`

▲ 'Release Information' in 'Release History'
The command **Trace.STATistic.Func** analyzes the function nesting and calculates the time spent in functions and the number of function calls.

The format is:

```
<trace>.STATistic.Func [%<format>] [<list_items> …] [<option>]
<trace>.STATistic.TASKFunc
```

### <format>:
- **DEFault**
- **LEN**
- **TimeAuto**
- **TimeFixed**

### <list_item>:
- **DEFault**
- **ALL**
- **Total**
- **MIN**
- **MAX**
- **AVeRage**
- **Count**
- **NAME**
- **TASK**
- **Internal**
- **IAVeRage**
- **IMIN**
- **IMAX**
- **InternalRatio**
- **InternalBAR**
- **External**
- **EAVeRage**
- **EMIN**
- **EMAX**
- **INTR**
- **INTRMAX**
- **INTRCount**
- **ExternalTASK**
- **ExternalTASKMAX**
- **TASKCount**
- **TotalRatio**
- **TotalBAR**

### <option>:
- **FILE**
- **FlowTrace**
- **BusTrace**
- **Accumulate**
- **INCremental**
- **FULL**
- **Sort <item>**
- **Track**
- **NoMerge**
- **IncludeOwn**
- **IncludeTASK**
- **IncludeINTR**
- **INTRROOT**
- **INTRTASK**
- **TASK <task>**
- **TASK !<task>**

### <task>:
- **<task_magic>**
- **<task_id>**
- **<task_name>**

For information about the parameters, see "**What to know about Task Magic Numbers, Task IDs and Task Names**", page 24.
Considerations

Please be aware that any gap in the trace recording (FIFOFULL) might result in an incorrect analysis results.

The trace can be tested for FIFOFULLs as follows:

```plaintext
; Process the complete trace contents
Trace.FLOWPROCESS

IF A.FLOW.FIFOFULL() != 0
    PRINT "Trace.STATistic.Func not possible due to FIFOFULL errors."
```

If it is not possible to eliminate the FIFOFULLs, it is recommended to use the command `Trace.STATistic.sYmbol`. 
In order to prepare the results for the command Trace.STATistic.Func, TRACE32 postprocesses the program flow recorded by the PowerTrace to find:

- **Function entries**
  The execution of the first instruction of an HLL function is regarded as function entry. Additional identifications for function entries are implemented depending on the processor architecture and the used compiler.

- **Function exits**
  A RETURN instruction within an HLL function is regarded as function exit. Additional identifications for function exits are implemented depending on the processor architecture and the used compiler.

- **Entries to interrupt service routines (asynchronous)**
  If an interrupt was identified, the following entry to an HLL function is regarded as entry to the interrupt service routine. Interrupts are identified as follows:
  - The trace port broadcasts the occurrence of an interrupt (e.g. PPC4xx).
  - An entry to the vector table is detected and the vector address indicates an asynchronous/hardware interrupt (e.g. ARM9).
  - If the vector table base address is configurable the usage of the command SYStem.Option VECTORS might be necessary (e.g. MPC5xxx).

If an interrupt is detected in the trace, it is marked as in the screenshot below.

- **Exits of interrupt service routines**
  A RETURN / RETURN FROM INTERRUPT within the HLL interrupt service routine is regarded as exit of the interrupt service routine.
Entries to TRAP handlers (synchronous)

If an entry to the vector table was identified and if the vector address indicates a synchronous interrupt/trap the following entry to an HLL function is regarded as entry to the trap handler.

If a TRAP is detected in the trace, it is marked as in the screenshot below.

Exits of TRAP handlers

A RETURN / RETURN FROM INTERRUPT within the HLL trap handler is regarded as exit of the trap handler.
Interpretation of the Result

Number of analyzed functions: 1390
Total measurement time: 20.586s
Total time in interrupt service routines over the total measurement time: 59.741ms
<table>
<thead>
<tr>
<th>&lt;list_item&gt;</th>
<th>Default Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Displays the function name. By default the functions are sorted by their occurrence in the trace.</td>
</tr>
<tr>
<td>TASK</td>
<td>Displays the task name.</td>
</tr>
</tbody>
</table>

Some additional explanations with regards to the function name (column range):

- **(root)**: is the root of the analyzed function nesting.
- **HLL interrupt service routines**: HLL interrupt service routines are indicated in the analysis as shown below:
  ```
  \mnts\bute\build\intr_os_wrapper\intr_os_prologue60
  ```
- **HLL trap handler**: HLL trap handler are indicated in the analysis as shown below:
  ```
  \____.ArmVectorSwi
  ```

If **Trace.STATistic.TASKFunc** was performed instead of **Trace.STATistic.Func**, because TRACE32 detected an RTOS, the following function names will appear:

- **<function>@<task_name>**: The name of the task in which the function is called is appended to the function name.
  ```
  \rom\Div64\UDiv64@Timer_Task
  ```

- **(root)<task_name>**: is the root of the analyzed function nesting for the task `<task_name>`.
- **(root)@(root)**: program section where no task-assignment is possible (e.g. measurement started within a task) are summarized here.
The following description of the `<list_item>` that provide the analysis results is kept quite general. An accurate description is given together with the Analysis Options.

<table>
<thead>
<tr>
<th><code>&lt;list_item&gt;</code></th>
<th>Default Display</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total</td>
<td>The total time within the function.</td>
</tr>
<tr>
<td>MIN</td>
<td>The shortest measured time it took to execute the function. The time includes the execution times of all subfunction calls. The time used for interrupt requests is not included, unless the window is opened with option IncludeINTR. If the function was never executed completely, the MIN time is not displayed.</td>
</tr>
<tr>
<td>MAX</td>
<td>The longest measured time it took to execute the function. The time includes the execution times of all subfunction calls. The time used for interrupt requests is not included, unless the window is opened with option IncludeINTR.</td>
</tr>
<tr>
<td>AVerage</td>
<td>The average time it took to execute the function. The time includes the execution times of all subfunction calls. The time used for interrupt requests is not included, unless the window is opened with option IncludeINTR.</td>
</tr>
<tr>
<td>Count</td>
<td>Number of calls of the function. If a function is never completely executed, no number of calls is displayed.</td>
</tr>
</tbody>
</table>

If function entries or exits are missing, this is displayed in the following format:

`<times within the function>. (<number of missing function entries>/<number of missing function exits>).`

**count**

| `2.(2/0)` |

**Interpretation examples:**

1. 950. (0/1): 950. times within the function, 1 function exit is missing.
2. 9. (1/0): 9. times within the function, 1 function entry is missing.
3. 11. (1/1): 11. times within the function, 1 function entry and 1 function exit is missing.
4. 9. (0/3): 9. times within the function, 3 function exits missing.

If the number of missing function entries or exits is higher than 1, the analysis performed by the command `Trace.STATistic.Func` might fail due to nesting problems. A detailed view to the trace contents is recommended.

In some cases a further treatment of the trace contents might help. For more information refer to Adjusting the Measurement.
<table>
<thead>
<tr>
<th><strong>&lt;list_item&gt;</strong></th>
<th><strong>Time only in Function</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Internal</strong></td>
<td>Total time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines, other tasks …</td>
</tr>
<tr>
<td><strong>IAVeRage</strong></td>
<td>Average time between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines, other tasks …</td>
</tr>
<tr>
<td><strong>IMIN</strong></td>
<td>Shortest between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines, other tasks …</td>
</tr>
<tr>
<td><strong>IMAX</strong></td>
<td>Longest time spent in the function between function entry and exit without called sub-functions, TRAP handlers, interrupt service routines, other tasks …</td>
</tr>
<tr>
<td><strong>InternalRatio</strong></td>
<td><code>&lt;internal_time_of_function&gt;/total_measurement_time&gt;</code> as a numeric value.</td>
</tr>
<tr>
<td><strong>InternalBAR</strong></td>
<td><code>&lt;internal_time_of_function&gt;/total_measurement_time</code> graphically.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>&lt;list_item&gt;</strong></th>
<th><strong>Time in Sub-Functions</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>External</strong></td>
<td>Total time spent within called sub-functions, TRAP handlers, interrupt service routines, other tasks …</td>
</tr>
<tr>
<td><strong>EAVeRage</strong></td>
<td>Average time spent within called sub-functions, TRAP handlers, interrupt service routines, other tasks …</td>
</tr>
<tr>
<td><strong>EMIN</strong></td>
<td>Shortest time spent within called sub-functions, TRAP handlers, interrupt service routines, other tasks …</td>
</tr>
<tr>
<td><strong>EMAX</strong></td>
<td>Longest time spent within called sub-functions, TRAP handlers, interrupt service routines, other tasks …</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>&lt;list_item&gt;</strong></th>
<th><strong>Interrupt Times</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>INTR</strong></td>
<td>Total time the function was interrupted.</td>
</tr>
<tr>
<td><strong>INTRMAX</strong></td>
<td>Max. time 1 function pass was interrupted.</td>
</tr>
<tr>
<td><strong>INTRCount</strong></td>
<td>Number of interrupts that occurred during the function run-time.</td>
</tr>
</tbody>
</table>
The `<list_items>` can be arranged as shown in the following examples:

```
Trace.STATistic.Func INTR DEFault
Trace.STATistic.Func Internal IAVeRage Count INTR InternalRatio
```

The `<list_items>` can also be arranged by pushing the Config button in the Trace.STATistic.Func window.
<table>
<thead>
<tr>
<th>Basic &lt;options&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FILE</strong></td>
<td>Displays trace memory contents loaded with Trace.FILE.</td>
</tr>
<tr>
<td><strong>FlowTrace</strong></td>
<td>The trace works as flow trace. This option is usually not required.</td>
</tr>
<tr>
<td><strong>BusTrace</strong></td>
<td>The trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td><strong>Accumulate</strong></td>
<td>By default only the current trace contents is analyzed by the statistic functions. The option /Accumulate allows to add the current trace contents to the already displayed results.</td>
</tr>
<tr>
<td><strong>INCremental</strong></td>
<td>Intermediate results are displayed while the TRACE32 software analyses the trace contents (default).</td>
</tr>
<tr>
<td><strong>FULL</strong></td>
<td>The result is displayed after the TRACE32 software finished the analysis.</td>
</tr>
<tr>
<td><strong>Track</strong></td>
<td>Track the Trace.STATistic window with other trace list windows (tracking to record number or time possible).</td>
</tr>
<tr>
<td><strong>NoMerge</strong></td>
<td>(For diagnosis purpose only).</td>
</tr>
<tr>
<td>COMMAND</td>
<td>DESCRIPTION</td>
</tr>
<tr>
<td>---------</td>
<td>-------------</td>
</tr>
<tr>
<td>TASK &lt;task&gt;</td>
<td>Perform nesting function run-time analysis only on specified task. For information about the parameters, see “What to know about Task Magic Numbers, Task IDs and Task Names”, page 24.</td>
</tr>
<tr>
<td>TASK !&lt;task&gt;</td>
<td>Exclude specified task from nesting function run-time analysis. This option can be useful if the nesting analysis for the specified task is problematic. For information about the parameters, see “What to know about Task Magic Numbers, Task IDs and Task Names”, page 24.</td>
</tr>
</tbody>
</table>

---

![Image of a table showing function analysis results](image-url)
### Analysis Options

<table>
<thead>
<tr>
<th>Option</th>
<th>Configuration of the Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>(default)</td>
<td>Function run-times are calculated without interrupts.</td>
</tr>
</tbody>
</table>

#### Diagram:

- **Start of measurement**
- Entry to `func1`
- Exit of `func1`
- Entry to `func1`
- `func2`
- TRAP1
- `func3`
- `interrupt 1`
- Exit of `func1`
- Entry to `func1`
- Exit of `func1`
- End of measurement
<table>
<thead>
<tr>
<th>&lt;option&gt;</th>
<th>Configuration of the Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncludeINTR</td>
<td>Function run-times include times in interrupts. In other words, interrupts are treated as sub-functions.</td>
</tr>
</tbody>
</table>

![Diagram showing function run-times including times in interrupts.](image-url)
<table>
<thead>
<tr>
<th>&lt;option&gt;</th>
<th>Configuration of the Analysis (RTOS)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IncludeOWN +</strong></td>
<td>Function run-times without interrupts and without times in other tasks (default).</td>
</tr>
<tr>
<td><strong>INTRROOT</strong></td>
<td>Interrupts are assigned to (root)@ (root)</td>
</tr>
</tbody>
</table>

The diagram illustrates the flow of execution and events in different tasks and functions, including:

- **Start of measurement**
- **First entry to TASK1**
- **First task switch recorded to trace**
- **Entry to func1 in TASK1**
- **func2 in TASK1**
- **func3 in TASK1**
- **TRAP1 in TASK1**
- **func4 in TASK1**
- **func4 in TASK1**
- **interrupt1 in TASK1**
- **Exit of func1 in TASK1**
- **Entry to func1 in TASK1**
- **Exit of func1 in TASK1**

**Total of (root)@root**

**Total of (root)@TASK1**

**Total of func1@TASK1**

**Internal of func1@TASK1**

**External TASK of func1@TASK1**

**External of func1@TASK1**

**Internal of func1@TASK1**

**Entry to func1 in TASK1**

**Exit of func1 in TASK1**

**Last exit of TASK1**

©1989-2019 Lauterbach GmbH
<table>
<thead>
<tr>
<th>&lt;option&gt;</th>
<th>Configuration of the Analysis (RTOS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncludeTASK + INTRROOT</td>
<td>Function run-times without interrupts but with times in other tasks.</td>
</tr>
<tr>
<td></td>
<td>Interrupts are assigned to (root)@ (root)</td>
</tr>
</tbody>
</table>

<Diagram>

- **Start of measurement**
- **First task switch recorded to trace**
- **Entry to func1 in TASK1**
- **func2 in TASK1**
- **func2 in TASK1**
- **func3 in TASK1**
- **TRAP1 in TASK1**
- **func4 in TASK1**
- **func4 in TASK1**
- **interrupt1 in TASK1**
- **Exit to func1 in TASK1**
- **Entry to func1 in TASK1**
- **Exit of func1 in TASK1**

©1989-2019 Lauterbach GmbH
<table>
<thead>
<tr>
<th>&lt;option&gt;</th>
<th>Configuration of the Analysis (RTOS)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IncludeOWN + INTRTASK</td>
<td>Function run-times without interrupts and without times in other tasks (default). Interrupts are assigned to (root) @ &lt;task_name&gt;</td>
</tr>
</tbody>
</table>
Sorting

<table>
<thead>
<tr>
<th>/Sort &lt;item&gt;</th>
<th>Sorting the Analysis Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>Sorting by program flow (default)</td>
</tr>
<tr>
<td>Nesting</td>
<td>Sorting by nesting</td>
</tr>
<tr>
<td>Address</td>
<td>Sorting by addresses</td>
</tr>
<tr>
<td>sYmbol</td>
<td>Sorting by names</td>
</tr>
<tr>
<td>TotalRatio/Ratio</td>
<td>Sorting by TotalRatio</td>
</tr>
<tr>
<td>Count</td>
<td>Sorting by Count</td>
</tr>
<tr>
<td>Window Global</td>
<td>(ineffectual)</td>
</tr>
</tbody>
</table>

The sorting can also be arranged by pushing the Config button in the Trace.STATistic.Func window.

If All Windows is selected, the selected sorting method is applied to all Trace.STATistic and Trace.Chart windows.

See also Trace.STATistic.Sort.
• **Trace.STATistic.FIRST**/ **Trace.STATistic.LAST**

The *Trace.STATistic* commands analyze the complete trace contents by default. The command **Trace.STATistic.FIRST** allows to freely select the start point for the statistic analysis; the command **Trace.STATistic.LAST** allows to freely select the end point for the statistic analysis.

• **sYmbol.NEW.MARKER FENTRY / FEXIT**

If the function nesting analysis can’t identify code sections as HLL functions (e.g. assembler function, unusual function exits) these code sections can be marked manually as functions by using the marker FENTRY and FEXIT.

Example 1:

Since func3 is the HLL function executed after an interrupt occurred, it is regarded as interrupt service routine.

Since ass_int is now marked as a function, it is correctly identified as interrupt service routine.

```
; mark the entry of the assembler function ass_int as function entry
sYmbol.New.MARKER FENTRY ass_int

; mark the exit of the assembler function ass_int as function exit
sYmbol.New.MARKER FEXIT ass_int+0x15F

; list the marker
sYmbol.List.MARKER
```
Example 2:

Since interrupt1 is the HLL function executed after an interrupt occurred, it is regarded as interrupt service routine. The assembler code from ass_int is added to the time in func2.

Since ass_int is now marked as function, it is correctly identified as interrupt service routine. interrupt1 is a sub-function called by ass_int now.
If the KERNEL is using special methods to call/end KERNEL functions, this might annoy the function nesting analysis. In such a case it is recommended to exclude the KERNEL from the function nesting by using the markers KENTRY/KEXIT.

Example:

```
Entry to func1 in TASK1
  func2 in TASK1
    KERNEL prologue
      kfunca in KERNEL
        kfuncb in KERNEL
          KERNEL epilogue
            func2 in TASK1
```

The KERNEL is manipulating the return address on the stack in order to return quickly into TASK1. This behavior will annoy the function nesting analysis.

```
Entry to func1 in TASK1
  KENTRY
      KERNEL prologue
        KERNEL is excluded from the function nesting
```

The usage of the markers KENTRY/KEXIT excluded the KERNEL from the function nesting in order to get a correct function nesting.
Advanced example for RTOS RTXC on a StarCore CPU:

```plaintext
; mark all interrupt service routines as kernel entries
sYmbol.ForEach "sYmbol.NEW.MARKER KENTRY " _isr_*"

; mark all RTE instructions in the specified program range as kernel exit
Data.Find P:RTXCProlog--P:RTXCProlog_end %Word 0x9f73
WHILE FOUND()

(  sYmbol.NEW.MARKER KEXIT P:TRACK.ADDRESS()
    Data.Find
)

sYmbol.List.MARKER
```

### Procedure for Measurement for TRACE32-ICE and TRACE32-FIRE

#### Mark the Functions

The measurement is based on a selective trace of all function entries and exits. In order to perform this measurement all function entries have to be marked with an Alpha breakpoint, and all function exits have to be marked with a Beta breakpoint.

These breakpoints can be set automatically with the command `Break.SetFunc`, if HLL-functions are loaded. Assembler functions or e.g. loops within HLL-functions can also be marked manually.

The entry points of interrupt routines should be marked with Alpha and Charly (or Beta on ECC8) breakpoints. Thus the analysis ignores interrupt function times and takes care about double fetches caused by interrupted programs.

If all interrupt routines are located in a specific memory range or within a specific HLL module the interrupt routines can be marked automatically by using the command `Break.SetFunction <range>| <module>/INTR`.

If there is a name convention for interrupt routines use the `sYmbol.ForEach` command to mark the interrupt routines.
Programming of the Trigger Unit

The analyzer is programmed to record accesses to ALPHA or BETA breakpoints and mark the records in the trace buffer as follows:

- Function entries are marked with an A marker.
- Function exits are marked with a B marker.
- Interrupt entries are marked with an A and a C marker.
- Interrupt exits are marked with a B marker.

```plaintext
Analyzer.ReProgram
(
    Sample.Enable IF AlphaBreak||BetaBreak
    Mark.A if AlphaBreak
    Mark.B if BetaBreak
    Mark.C if CharlyBreak ; not on ECC8
)
```

Recording

The interesting program flow is recorded. The trace memory should be cleared before starting the recording. This is automatically done if `Analyzer.AutoInit` is ON.

Check the trace by viewing the records with `<trace>.List MARK DEFault`.

The display of the nesting is possible with the command `Analyzer.List FUNC FUNCR` (this command uses the same strategy to determine function entries and exits).

If the nesting of the functions is not correct (each function-entry must have a corresponding function-exit), the results will be incorrect and an error message will be displayed in the statistic window. In this case use the command `<trace>.STATistic.Prefetch` to solve the problem. If still single records are wrong, the command `<trace>.STATistic.Ignore` can be used to remove them from the measurement.
### Basic <options>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Displays trace memory contents loaded with <code>Trace.FILE</code>.</td>
</tr>
<tr>
<td>FlowTrace</td>
<td>The trace works as flow trace. This option is usually not required.</td>
</tr>
<tr>
<td>BusTrace</td>
<td>The trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td>Accumulate</td>
<td>By default only the current trace contents is analyzed by the statistic functions. The option <code>/Accumulate</code> allows to add the current trace contents to the already displayed results.</td>
</tr>
<tr>
<td>INCremental</td>
<td>Intermediate results are displayed while the TRACE32 software analyses the trace contents (default).</td>
</tr>
<tr>
<td>FULL</td>
<td>The result is displayed after the TRACE32 software finished the analysis.</td>
</tr>
<tr>
<td>Track</td>
<td>Track the <code>Trace.STATistic</code> window with other trace list windows (tracking to record number or time possible).</td>
</tr>
<tr>
<td>NoMerge</td>
<td>(For diagnosis purpose only).</td>
</tr>
</tbody>
</table>

### CTS

Use **Context Tracking System** to fill trace gaps and then perform the `<trace>.STATistic.FUNC` command. This is only useful for the ARM-ETM, SH4 and NEXUS.

### INTR

The time spent in interrupts is included to the measurement like a function call.
The `<list_items>` can be arranged by pushing the Config button in the `<trace>.STATistic.FUNC` window.

<table>
<thead>
<tr>
<th>NAME</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Displays the function name. Interrupt functions are marked with ➔.</td>
</tr>
<tr>
<td>Time</td>
<td>Time between entry and exit summed over the complete measurement time. By default the time spent in interrupt routines is taken out of the measurement.</td>
</tr>
<tr>
<td>MIN</td>
<td>Shortest time from function entry to exit.</td>
</tr>
<tr>
<td>MAX</td>
<td>Longest time from function entry to exit.</td>
</tr>
<tr>
<td>AVerAge</td>
<td>Average time from function entry to exit.</td>
</tr>
<tr>
<td>Internal</td>
<td>Time spent within the function (without called sub functions).</td>
</tr>
<tr>
<td>IAVerAge</td>
<td>Average time spent in the function (without called sub functions).</td>
</tr>
<tr>
<td>IMIN</td>
<td>Shortest time spent in the function (without called sub functions).</td>
</tr>
<tr>
<td>IMAX</td>
<td>Longest time spent in the function (without called sub functions).</td>
</tr>
<tr>
<td>External</td>
<td>Time spent within sub functions.</td>
</tr>
<tr>
<td>EAVerAge</td>
<td>Average time spent within sub functions.</td>
</tr>
<tr>
<td>EMIN</td>
<td>Shortest time spent within sub functions.</td>
</tr>
<tr>
<td>EMAX</td>
<td>Longest time spent within sub functions.</td>
</tr>
<tr>
<td>MAXIntr</td>
<td>Maximum time the function was interrupted by an interrupt.</td>
</tr>
<tr>
<td>Count</td>
<td>Number of calls of the function. A negative value in brackets shows the number of not complete entries or exits to the function (i.e. when the measurement is stopped).</td>
</tr>
<tr>
<td>Ratio</td>
<td>Ratio of time spent within the function over the complete measurement time (without called subroutines).</td>
</tr>
<tr>
<td>BAR.log, BAR.LIN</td>
<td>Graphical display of the ratio (linear or logarithmic).</td>
</tr>
<tr>
<td>TRatio</td>
<td>Ratio of time spent within the function over the complete measurement time (called subroutines included).</td>
</tr>
<tr>
<td>TBAR.log, TBAR.LIN</td>
<td>Graphical display of the ratio (linear or logarithmic).</td>
</tr>
</tbody>
</table>
Format

LEN <size> Specifies the width of non numeric fields (e.g. symbols)

TimeAuto Adapt the time display. (default)

TimeFixed Display all time information in seconds.

See also

■ <trace>.STATistic

▲ 'Release Information' in 'Release History'
▲ 'Function Run-Times Analysis' in 'ARM-ETM Training'
▲ 'Nesting Function Run-Time Analysis - Single' in 'AURIX Trace Training'
▲ 'Function Run-Times Analysis - Single' in 'Nexus Training'
<trace>.STATistic.FuncDURation

Statistic analysis of single function

Format:  <trace>.STATistic.FuncDURation  <function_name>

Analyzes the function runtime between function entry and exit.

- The time spent in called subroutines is *included*.
- The time spent in called interrupt service routine and other tasks is *excluded*.

See also

- <trace>.STATistic.FuncDURationInternal
- <trace>.STATistic
- ‘Release Information’ in ‘Release History’
- ‘Function Run-Times Analysis - Single’ in ‘Nexus Training’
<trace>.STATistic(FuncDURationInternal

Statistic analysis of single func.

Format:  
<trace>.STATistic(FuncDURationInternal <function_name>)

Analyzes the function runtime between function entry and exit. The time spent in called subroutines, traps, interrupt service routine and other tasks is excluded.

See also

- <trace>.STATistic(FuncDURation
- <trace>.STATistic

▲ 'Release Information' in 'Release History'
GROUP run-time analysis

The time spent in groups and the number of calls is calculated (flat statistic).

Example:

GROUP.Create "INPUT" \jquant2 \jquant1 \jidctred \jdinput /AQUA
GROUP.Create "JPEG" \jdapimin \jdcolor \jddctmgr \jdcoefct /NAVY
Go
Break
Trace.STATistic.GROUP

See also

- <trace>.STATistic
- GROUP.Create

▲ 'Release Information' in 'Release History'

©1989-2019 Lauterbach GmbH
<trace>.STATistic.Ignore

Ignore false records in statistic

Format:  <trace>.STATistic.Ignore [<record> | <range>] [/<options>]

<option>:  FILE | BusTrace

The specified record(s) are ignored in the statistic analysis. This command can be used, when single records (caused by prefetch etc.) confuse the statistic analysis.

**FILE**
Displays trace memory contents loaded with Trace.FILE.

**BusTrace**
The trace works as a bus trace. This option is usually not required.

**Example:** The state of the ignore bit for each record can be displayed in the <trace>.List window by the IGNORE field.

measurement ...
...
Trace.STATistic.PreFetch ON ; enable prefetching -> will ignore most prefetch cases
Trace.STATistic.TREE ; display function nesting tree
Trace.List IGNORE Func MARK CPU ; control window for nesting analysis ...
Trace.STATistic.Ignore (-1000.)--(-995.) ; ignore six records

---

**E::Trace.List IGNORE Func MARK Address**

<table>
<thead>
<tr>
<th>record</th>
<th>ore</th>
<th>func</th>
<th>MARK</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>-000047</td>
<td></td>
<td>func10+0x194</td>
<td>-B-</td>
<td>SP:00001516</td>
</tr>
<tr>
<td>-000046</td>
<td></td>
<td>func13+0x2</td>
<td>A--</td>
<td>SP:0000159C</td>
</tr>
<tr>
<td>-000045</td>
<td>ign</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-000044</td>
<td>ign</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-000043</td>
<td>ign</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-000042</td>
<td></td>
<td>func13+0x48</td>
<td>-B-</td>
<td>SP:000015E2</td>
</tr>
<tr>
<td>-000041</td>
<td>ign</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-000040</td>
<td>ign</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-000039</td>
<td>ign</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-000038</td>
<td></td>
<td>func1g+0x2</td>
<td>A--</td>
<td>SP:000015E6</td>
</tr>
<tr>
<td>-000037</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-000036</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-000035</td>
<td></td>
<td>func1g+0x8</td>
<td>A--</td>
<td>SP:00001062</td>
</tr>
<tr>
<td>-000034</td>
<td></td>
<td>func1g+0x8</td>
<td>-B-</td>
<td>SP:00001068</td>
</tr>
</tbody>
</table>

---

See also
- <trace>.STATistic

©1989-2019 Lauterbach GmbH
Format: `<trace>.STATistic.INTERRUPT`

tbd.

See also
- `<trace>.STATistic`
- 'Release Information' in 'Release History'
In order to calculate the results for the *nesting function run-time analysis* the trace recording is post-processed. One important issue in this processing is the identification of interrupt entries and exits.

TRACE32 provides two methods to identify interrupt entries and exits:

- Default: `Trace.STATistic.InterruptIsFunction` OFF
- Recommended: `Trace.STATistic.InterruptIsFunction` ON

**Trace.STATistic.InterruptIsFunction OFF**

The screenshot below shows the function nesting for the interrupt.

1. The first HLL function called after the indirect branch to the Interrupt Vector Table is regarded as interrupt service routine (here `OSInterruptDispatcher1`).

2. The return from interrupt is regarded as the exit of this function (here `OSInterruptDispatcher1`).

Please be also aware that some trace port protocols require special setups for the Interrupt Vector Table. For details, please refer to your *Processor Architecture Manual*. 

Format: `<trace>.STATistic.InterruptIsFunction ON | OFF`
1. Interrupt entry is the point in the trace recording at which the indirect branch to the Interrupt Vector Table occurs.

2. Interrupt exit is the point in the trace recording at which the return from interrupt is executed.

TRACE32 handles the time between interrupt entry and exit as a function. The name given to this function is the label of the interrupt vector address.

Please be aware that method only works if interrupts are exit by regular return from interrupt.

Please be also aware that some trace port protocols require special setups for the Interrupt Vector Table. For details, please refer to your Processor Architecture Manual.

See also

- <trace>.STATistic
The **Trace.STATistic** commands analyze the complete trace contents by default. The command **Trace.STATistic.LAST** allows to freely select an end point for the statistic analysis.

### Format:

```plaintext
<trace>.STATistic.LAST <value> | <time> | <string>
```

### Example for `<value>`:

- **Trace.List** ; display trace listing
- **Trace.STATistic.FIRST -123366.** ; select trace record -123366.
  ; as start point for the trace
  ; analysis
- **Trace.STATistic.LAST -36675.** ; select trace record -36675.
  ; as end point for the trace
  ; analysis
- **Trace.STATistic.Func** ; perform a function run-time
  ; analysis
Example for `<time>`:

```
Trace.List Time.Zero Default ; display trace listing
Trace.STATistic.LAST 468.2us ; select trace record with time stamp
                      ; 468.2 µs (zero time) as end point for
                      ; the trace analysis
Trace.STATistic.Func ; perform a function run-time analysis
                      ; from the beginning of the trace buffer
                      ; to the specified end point
```

See also

- `<trace>.STATistic`
- `<trace>.STATistic.FIRST`
- 'Release Information' in 'Release History'
<trace>.STATistic.Line

HLL-line analysis

Format:  <trace>.STATistic.Line [/<option>]

<options>:
- FILE
- FlowTrace | BusTrace
- Accumulate
- INCremental | FULL
- Sort <item>
- Track
- NoMerge
- BEFORE | AFTER
- List <item> ...
- Filter <item>

Analyzes the time spent in HLL lines.

Options

<table>
<thead>
<tr>
<th>Basic &lt;options&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Displays trace memory contents loaded with Trace.FILE.</td>
</tr>
<tr>
<td>FlowTrace</td>
<td>The trace works as flow trace. This option is usually not required.</td>
</tr>
<tr>
<td>BusTrace</td>
<td>The trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td>Accumulate</td>
<td>By default only the current trace contents is analyzed by the statistic functions. The option /Accumulate allows to add the current trace contents to the already displayed results.</td>
</tr>
<tr>
<td>INCremental</td>
<td>Intermediate results are displayed while the TRACE32 software analyses the trace contents (default).</td>
</tr>
<tr>
<td>FULL</td>
<td>The result is displayed after the TRACE32 software finished the analysis.</td>
</tr>
<tr>
<td>Basic &lt;options&gt;</td>
<td>Description</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Track</td>
<td>Track the Trace.STATistic window with other trace list windows (tracking to record number or time possible).</td>
</tr>
<tr>
<td>NoMerge</td>
<td>(For diagnosis purpose only).</td>
</tr>
</tbody>
</table>

**BEFORE** Display the time before the program entered the listed HLL line, that means how long the program was is the previous HLL line before the listed HLL line was entered.

**AFTER** Display the time after the program entered the listed HLL line, that means how long the program was in the listed HLL line (default).

**List <items>** Specify the result that should be displayed in the window.

**Filter <item>** Filter the HLL lines to analyze only a specific function or module.

The **List** option defines which values are calculated and displayed. The **<list_items>** can be arranged by pushing the **Config** button in the Trace.STATistic.Line window.

**<list_item>**

**TIme** Total time in the HLL line.

**MIN, MAX** Minimum and maximum times in the HLL line.

**AVeRage** Average time in the HLL line.

**Count** Number of executions of the HLL line.

**Ratio, BAR.log, BAR.LINear** Ratio of time spent in the HLL line to total measurement time in percent and as graphical bars.

**CRatio, CBAR.log, CBAR.LINear** Ratio of count to total count in percent and as graphical bars.

All time displays depend on the options **AFTER** or **BEFORE**.
Example for TRACE32-ICD and PowerTrace:

If no selective trace is possible use the option `/Filter` to filter out the module or function of interest.

```plaintext
Go
Break
Trace.STATistic.Line /Filter V.RANGE(sieve)
```

Example for TRACE32-ICE and TRACE32-FIRE:

If only a specific module or function should be analyzed, perform a selective trace on the module or function.

```plaintext
Analyzer.ReProgram
(
    ADDR AlphaBreak V.RANGE(module1) ; sample only instruction from
    Sample.Enable IF AlphaBreak ; module1
)
Go
Break ; collect data
Trace.STATistic.Line ; graphical display of HLL ; analysis
```

See also

- `<trace>.STATistic`
- 'Release Information' in 'Release History'
Perform a function run-time statistic for a single function itemized by its callers. `<address>` has to be the function entry address.

The procedure for recording the data is the same as for the `<trace>.STATistic.Func` command.
The function `alloc_small` was called by the listed 20 functions. The dependency between the run-time of the function `alloc_small` and its callers is analyzed.

See also

- `<trace>.STATistic`
- 'Release Information' in 'Release History'

```plaintext
<trace>.STATistic.LISTCONFIG
tbd.
```

Format: `<trace>.STATistic.LISTCONFIG` tbd.

See also

- `<trace>.STATistic`
This command allows to analyze the performance of a single signal. It is mainly used with PowerProbe or PowerIntegrator.

Typical application for the `<trace>.STATistic.Measure`:

- to check the best threshold level for a symmetric signal (e.g. a symmetric clock signal).
- to detect spikes (e.g. a signal has a defined period of 10 ns, detect if there is any much smaller period).

### Format:

```
<trace>.STATistic.Measure [<record> | <range>] [<items> ...] [<options>]
```

<table>
<thead>
<tr>
<th>recs</th>
<th>The number of records that are analyzed.</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>The time that is analyzed.</td>
</tr>
<tr>
<td>lead</td>
<td>The time from the beginning of the analysis until the first edge.</td>
</tr>
<tr>
<td>tail</td>
<td>The time from the last edge until the end of the analysis.</td>
</tr>
<tr>
<td>____</td>
<td>The number of low states.</td>
</tr>
<tr>
<td>____</td>
<td>The number of high states</td>
</tr>
</tbody>
</table>
The analysis can also be activated by selecting the signal in the **Trace.Timing** display and by using the pull-down menu provided via the right mouse button.

It is also possible to analyze only the selected part of the complete recording time.

See also
- `<trace>.STATistic`
- ‘Release Information’ in ‘Release History’
<trace>.STATistic.MODULE

Code execution broken down by module

Format:  
<trace>.STATistic.MODULE [\%<format>] [\<list_items> …] [/\<option>]

Shows a statistical analysis of the code execution broken down by module.

See also
- <trace>.STATistic
- <trace>.STATistic.PROGRAM

<trace>.STATistic.PAddress

Which instructions accessed data address

Format:  
<trace>.STATistic.PAddress /Filter Address <address>

The command provides a statistic about the instructions that accessed the data address specified by the Filter option.

Trace.STATistic.PAddress /Filter Address mstatic1

See also
- <trace>.STATistic
- 'Release Information' in 'Release History'

©1989-2019 Lauterbach GmbH
Show the call context of a function.

Format: `<trace>.STATistic.ParentTREE <address>`

Show call tree and run-time of all callers of the specified function. The function is specified by its start `<address>`.

Trace.STATistic.ParentTREE alloc_small

See also
- `<trace>.STATistic`
- `<trace>.STATistic.ChildTREE`
<trace>.STATistic.PIPELINE

Format: <trace>.STATistic.PIPELINE tbd.

tbd.

See also

- <trace>.STATistic
This command should only be used when a prefetching CPUs is used!

If ON is selected, the software will try to eliminate the errors in statistics generated by prefetches. The command affects the result of the commands `<trace>.STATistic`, `<trace>.Chart` and `<trace>.List FUNC` (and all related commands).

When the option cycles is used, another software prefetch detection strategy is added. In this case after each marked record there must be a number of consecutive cycles, otherwise the record is treated as a prefetch cycle. This technique can be used to eliminate false entry/exits detections caused by prefetching and interrupts.

If the prefetch problem cannot be solved by this command, an alternative can be tagging the code (command `Data.TAGFunc`) or by manually removing records from the measurement (command `<trace>.STATistic.Ignore`).

The following typical prefetch conditions are detected:

### Alpha/Beta prefetched direct after Beta

The beginning of a function may be sampled during leaving the prior function by a prefetch. This may happen, if the breakpoints were set manually or with the command `Break.SetFunc /NPreFetch`. The prefetch will be detected by the software, if an access to Alpha or Beta is made directly after an access to Beta, without any other program fetch cycles between. If the record after the ignored record is also prefetched, it will be ignored too.

### Double fetches after interrupts

Problems with the prefetch may occur, if interrupts are handled by the program during the analysis. After accessing a tag breakpoint on processors with prefetch it is possible that the interrupt program is entered. After leaving the interrupt program the same breakpoint is read again, causing a nesting error in the analysis. If the `PreFetch` option is active and the interrupt program is marked with a Charly breakpoint, the software will assume that a second read of a breakpoint tag after the interrupt function was caused by a prefetch. A result may be that recursive functions may cause nesting errors, if the interrupt occurs in such a procedure. A 100% solution is not possible, when working with prefetching CPUs and sampling program fetches. The best solution in this case is to use data tags (see command `Break.SetFunc`).

### Multiple prefetched Beta through program structure

If the epilog code of a function contains a loop, which accesses the Beta breakpoint more than once, it will be detected. The last access to the Beta will be taken into account. If the beta breakpoint is accessed by a prefetch, but the function continues this will also be detected.

See also

- `<trace>.STATistic`
<trace>.STATistic.PROGRAM    Code execution broken down by program

Format: <trace>.STATistic.PROGRAM [\%<format>] [<list_items> …] [/<option>]

Shows a statistical analysis of the code execution broken down by program.

See also
■ <trace>.STATistic    ■ <trace>.STATistic.MODULE

<trace>.STATistic.PsYmbol    Shows which functions accessed data address

Format: <trace>.STATistic.PsYmbol /Filter <filter>

The command provides a statistic about the functions that accessed the data address specified by the /Filter option.

Trace.STATistic.PsYmbol /Filter sYmbol mstatic1
Trace.STATistic.PsYmbol /Filter sYmbol mstatic1 CYcle Write

Preconditions:
  • Has to be implemented for the processor architecture in use.
  • Data access has to be clearly assignable to an instruction.
If TRACE32 was able to clearly assign the data access to an instruction can be checked as follows:

```
Trace.FindAll sYmbol mstatic1
```

A read cycle type indicates that a clear assignment was not possible.

```
; PAddress: address of instruction that performed the data access
; PsYmbol: symbolic address of instruction that performed the data access
Trace.List PAddress PsYmbol DEFault
```

Both columns are empty if no clear assignment is possible.

See also
- `<trace>.STATistic`
- ‘Release Information’ in ‘Release History’
### Trace.STATistic.Sort

Specify sorting criterion for statistic commands.

If the command is entered without parameters, a **Trace.STATistic.Sort** dialog is displayed.

The sorting criterion specified by **Trace.STATistic.Sort** applies to all **Trace.STATistic** and **Trace.Chart** analysis windows (check box All Windows ON).

#### Format

```
<trace>.STATistic.Sort [<sort_visible>] [<sort_core>] [<sort>]
```

- **<sort_visible>:** Window | Global
- **<sort_core>:** CoreTogether | CoreSeparated (SMP systems only)
- **<sort>:** OFF
  - Address | sYmbol [wildcard_list ...] | Group
  - Nesting | InternalRatio | TotalRatio
  - Ratio
  - Count
  - TotalMAX | RatioMAX
To specify the sorting criterion for an individual statistic window use the **Config** button of this statistic window or use the **/Sort** option when you enter the command.

```
Trace.Chart.Symbol /Sort TotalRatio ; sort the time chart by the
; criterion TotalRatio
```

### Default Sorting Criterion

**OFF** is the default mode for most statistic windows. **OFF** means that the analyzed items are displayed in their recording order.

Statistic windows that are focused on the program's call hierarchy e.g **Trace.STATistic.TREE** use **Nesting** as default mode.
## Window vs. Global

<table>
<thead>
<tr>
<th><strong>Global (default)</strong></th>
<th>The sorting criterion is strictly maintained.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Window</strong></td>
<td>The sorting criterion is applied. The analyzed items active in the displayed time interval are displayed first, followed by the non-active items. Window might be useful if you scroll horizontally.</td>
</tr>
</tbody>
</table>

---

**Trace.Chart.Symbol /Sort Window Symbol**

---

![Trace chart example](image-url)
<table>
<thead>
<tr>
<th>CoreTogether (default)</th>
<th>The analyzed items are displayed per core. Additional sorting criteria apply to this per core order.</th>
</tr>
</thead>
<tbody>
<tr>
<td>CoreSeparately</td>
<td>The core information has no impact on the sorting order.</td>
</tr>
</tbody>
</table>

```
Trace.Chart.Symbol /ZoomTrack /Sort CoreTogether Symbol
Trace.Chart.Symbol /ZoomTrack /Sort CoreSeparated Symbol
```
Standard Sorting Criteria

<table>
<thead>
<tr>
<th>Address</th>
<th>Sort result by address</th>
</tr>
</thead>
<tbody>
<tr>
<td>sYmbol</td>
<td>Sort result alphabetically by symbol names</td>
</tr>
<tr>
<td>[&lt;wildcard_list ...&gt;]</td>
<td>Sort result by their grouping</td>
</tr>
<tr>
<td>GROUP</td>
<td>Sort analyzed items by their occurrence</td>
</tr>
</tbody>
</table>

Example for sort criterion sYmbol [<wildcard_list ...>].

```plaintext
; display items starting with string "SPI" first, then items starting 
; with string "SUP" then rest
Trace.STATistic.Sort sYmbol Spi* SUP*
Trace.Chart.symbol
```

![Image of sorting criteria example](image_url)
Example for sort criterion `GROUP`.

```
GROUP.List
Trace.STATistic.Sort GROUP
Trace.Chart.symbol
```

GROUP other

GROUP my_code

GROUP toms_code

GROUP ralfs_code
### Sorting Criteria for the Nesting Analysis

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nesting</td>
<td>Calling functions are displayed atop of called function.</td>
</tr>
<tr>
<td>InternalRatio</td>
<td>Sort result be internal ratio. &lt;br&gt;<strong>InternalRatio</strong>: <code>&lt;time_in_function&gt;/total_measurement_time</code> as a numeric value.</td>
</tr>
<tr>
<td>TotalRatio</td>
<td>Sort result by total ratio. &lt;br&gt;<strong>InternalRatio</strong>: <code>&lt;total_time_of_function&gt;/total_measurement_time</code> as a numeric value, <code>&lt;total_time_of_function&gt;</code> includes called subfunctions and traps.</td>
</tr>
</tbody>
</table>

Example for criterion **Nesting**.

```
Trace.Chart.Func /ZoomTrack /Sort Nesting
```

![Diagram showing the Nesting analysis in BiTrace]
## Sorting Criteria for the Flat Analysis

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Ratio</strong></td>
<td>Sort analyzed items by their ratio.</td>
</tr>
<tr>
<td><strong>TotalMAX</strong></td>
<td>Flat analysis with InterVal option only. Sort analyzed items by maximal total time per specified interval.</td>
</tr>
<tr>
<td><strong>RatioMAX</strong></td>
<td>Flat analysis with InterVal option only. Sort analyzed items by maximal ratio per specified interval.</td>
</tr>
</tbody>
</table>

Trace.STATistic.sYmbol /InterVal 10.ms /Sort RatioMAX

See also
- `<trace>.STATistic`
- 'Release Information' in 'Release History'

©1989-2019 Lauterbach GmbH
The execution time in different symbol regions is displayed. Displayed are the number of entries into the range and the time spent in the range.

Format: <trace> . STATistic . sYmbol [ %<format> ] [ <list_item> … ] [/<option>]

<format>: DEFault | LEN
         TimeAuto | TimeFixed

<list_item>: DEFault | ALL
            Total | MIN | MAX | AVerage
            Count
            NAME | CountRatio | CountBAR
            CountChange | CountFirst | CountALL

=options>: FILE
            FlowTrace | BusTrace
            Accumulate
            INCremental | FULL
            Sort <item>
            Track
            NoMerge
            SplitTASK
            NoInline
            BEFORE | AFTER
            Address <function1> Il <function2> …
            Address <function_m> Il <function_n>
            Filter Address <function1> Il <function2> …
            Filter Address <function_m> Il <function_n>
            ALL
### Options

#### Configuration of Count column

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CountFirst</td>
<td>Count the occurrence of the start address of a program symbol region or of a function.</td>
</tr>
<tr>
<td>CountChange</td>
<td>Count how often the address range of a program symbol region or of a function was entered.</td>
</tr>
<tr>
<td>CountALL</td>
<td>Count all executed instructions.</td>
</tr>
</tbody>
</table>

#### Basic <options> Description

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Displays trace memory contents loaded with <code>Trace.FILE</code>.</td>
</tr>
<tr>
<td>FlowTrace</td>
<td>The trace works as flow trace. This option is usually not required.</td>
</tr>
<tr>
<td>BusTrace</td>
<td>The trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td>Accumulate</td>
<td>By default only the current trace contents is analyzed by the statistic functions. The option <code>/Accumulate</code> allows to add the current trace contents to the already displayed results.</td>
</tr>
<tr>
<td>INCremental</td>
<td>Intermediate results are displayed while the TRACE32 software analyses the trace contents (default).</td>
</tr>
<tr>
<td>FULL</td>
<td>The result is displayed after the TRACE32 software finished the analysis.</td>
</tr>
<tr>
<td>Track</td>
<td>Track the <code>Trace.STATistic</code> window with other trace list windows (tracking to record number or time possible).</td>
</tr>
<tr>
<td>NoMerge</td>
<td>(For diagnosis purpose only).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SplitTASK</td>
<td>Splits up the results for different tasks.</td>
</tr>
<tr>
<td>NoInline</td>
<td>Inline functions are treated as separate functions. The option <code>NoInline</code> can be used to discard inline functions.</td>
</tr>
<tr>
<td>BEFORE</td>
<td>Display the time before the program entered the listed symbol range, that means how long the program was in the previous symbol range until the listed symbol range was entered.</td>
</tr>
</tbody>
</table>
Display the time after the program entered the listed symbol range, that means how long the program was in the listed symbol range (default).

; filter the specified functions out of the trace stream
; and then analyze the filtered trace information
Trace.STATistic.Symbol /Filter Address main|func2|func10|func26

Recording (filtered functions are displayed in black)

Analysis result
Perform statistic on specified functions, assign statistic information for all other functions to (other).

The **GROUP** command provides more features to structure your statistic.

```
Trace.STATistic.sYmbol /Address func2||func10||sfpDoubleNormalize
Trace.STATistic.sYmbol /Address func2--func7
```
The `<list_items>` can be arranged by pushing the **Config** button in the `<trace>.STATistic.sYmbol` window.

<table>
<thead>
<tr>
<th><strong>&lt;list_item&gt;</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TIme</strong></td>
<td>Total time in the symbol range.</td>
</tr>
<tr>
<td><strong>MIN, MAX</strong></td>
<td>Minimum and maximum times in the symbol range.</td>
</tr>
<tr>
<td><strong>AVeRage</strong></td>
<td>Average time in the symbol range.</td>
</tr>
<tr>
<td><strong>Count</strong></td>
<td>Number of entries to the symbol range.</td>
</tr>
<tr>
<td><strong>Ratio, BAR.log, BAR.LINear</strong></td>
<td>Ratio of time spent in the symbol range to total measurement time in percent and as graphical bars.</td>
</tr>
<tr>
<td><strong>CRatio, CBAR.log, CBAR.LINear</strong></td>
<td>Ratio of count to total count in percent and as graphical bars.</td>
</tr>
</tbody>
</table>

All time displays depend on the options **AFTER** or **BEFORE**.

**Example for PowerTrace:**

If no selective trace is possible use the option **/Filter** to filter out the module or function of interest.

```plaintext
Go
Break
Trace.STATistic.Sort sYmbol ; sort the result alphabetically
Trace.STATistic.sYmbol /Filter Address Y.SECRANGE(\diab555\.text)
```

**Example for TRACE32-ICE and TRACE32-FIRE:**

If only a specific module or function should be analyzed, perform a selective trace on the module or function.

```plaintext
Analyzer.ReProgram
{
    ADDR AlphaBreak ; sample only instructions from
    Y.SECRANGE(\diab555\.text) ; program \diab555
    Sample.Enable IF AlphaBreak
}
Go
Break
Trace.STATistic.Sort Ratio ; sort the result by ratio
Trace.STATistic.sYmbol ; display the result
```

**See also**
- `<trace>.Chart.sYmbol`
- `<trace>.STATistic`
- 'Release Information' in 'Release History'

©1989-2019 Lauterbach GmbH
Task activity statistic

Format: \texttt{<trace>.STATistic.TASK [%<format>] [<list_items>] [...] [<option>]} 

\texttt{<format>}:  
- DEFault \mid LEN \linebreak  
- TimeAuto \mid TimeFixed 

\texttt{<list_items>}:  
- DEFault \mid ALL \linebreak  
- Total \mid TotalMIN \mid TotalMAX \linebreak  
- MIN \mid MAX \mid AVeRage \linebreak  
- Count \mid CountMIN \mid CountMAX \linebreak  
- Ratio \mid RatioMIN \mid RatioMAX \linebreak  
- BAR.log \mid BAR.LINear \linebreak  
- TASK \mid GROUP 

\texttt{<option>}:  
- FILE \linebreak  
- InterVal \texttt{<time> \mid <event>} \linebreak  
- Accumulate \linebreak  
- INCremental \mid FULL \linebreak  
- Sort \texttt{<item>} \linebreak  
- Track \linebreak  

\texttt{FlowTrace \mid BusTrace} (diagnosis only) 

\texttt{CORE \texttt{<n> \mid SplitCORE} (default) \mid MergeCORE \mid JoinCORE} \linebreak  
(SMP Systems only) 

Task run-times are analyzed.

If a core trace is used, \textbf{“OS-aware Tracing”} (glossary.pdf) has to be enabled in order to use this command.
The column layout can be configured by using the Config… button of the Trace.STATistic.TASK window or

<table>
<thead>
<tr>
<th>column</th>
<th>&lt;list_item&gt;</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>range</td>
<td>(unknown)</td>
<td>Task name</td>
</tr>
<tr>
<td></td>
<td>(unknown)</td>
<td>(unknown): TRACE32 assigns all trace information generated before the first task information to the (unknown) task.</td>
</tr>
<tr>
<td>total</td>
<td>Total</td>
<td>Time period in the task during the recorded time period.</td>
</tr>
<tr>
<td>min</td>
<td>MIN</td>
<td>Shortest time in task.</td>
</tr>
<tr>
<td>max</td>
<td>MAX</td>
<td>Longest time in task.</td>
</tr>
<tr>
<td>avr</td>
<td>AVerRage</td>
<td>Average time in task.</td>
</tr>
<tr>
<td>count</td>
<td>Count</td>
<td>Number of time in task.</td>
</tr>
<tr>
<td>ratio</td>
<td>Ratio</td>
<td>Ratio of time in the task with regards to the total time period recorded.</td>
</tr>
<tr>
<td>(graphical bar)</td>
<td>BAR.LOG</td>
<td>Ratio of time in the task with regards to the total time period recorded graphically.</td>
</tr>
<tr>
<td>group</td>
<td>GROUP</td>
<td>Display of group name assigned by command GROUP.CreateTASK.</td>
</tr>
</tbody>
</table>
via the command line.

### Trace.STATisticTASK [[<list_item>]]

```plaintext
Trace.STATistic.TASK Count AVerage Ratio BAR.log
```

#### InterVal Analysis

The **InterVal** option allows to divide the time period recorded by the trace (total) into time slices. Additional analysis details can be displayed for these time slices.

```plaintext
Trace.STATistic.TASK /InterVal <time> | <event>
```

; divide trace into 10.ms time slices

```plaintext
Trace.STATistic.TASK /InterVal 10.ms
```

; divide trace in time slices, a new time slice is started when the
; function Func cpu0_generateData is entered

```plaintext
Trace.STATistic.TASK /InterVal sYmbol Func cpu0_generateData
```
### Survey (InterVal option)

<table>
<thead>
<tr>
<th>column</th>
<th>item</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>task</td>
<td></td>
<td>Number of recorded tasks.</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td>Time period recorded by trace.</td>
</tr>
<tr>
<td>intervals</td>
<td></td>
<td>Number of intervals.</td>
</tr>
<tr>
<td>avr</td>
<td></td>
<td>Average interval length.</td>
</tr>
<tr>
<td>min</td>
<td></td>
<td>Shortest interval.</td>
</tr>
<tr>
<td>max</td>
<td></td>
<td>Longest interval.</td>
</tr>
</tbody>
</table>

### Task details (InterVal option)

<table>
<thead>
<tr>
<th>column</th>
<th>item</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>totalmax</td>
<td>TotalMAX</td>
<td>Longest time period in the task within an interval.</td>
</tr>
<tr>
<td>ratiomax</td>
<td>RatioMAX</td>
<td>Highest ratio of time in the task within an interval.</td>
</tr>
<tr>
<td>countmax</td>
<td>CountMAX</td>
<td>Highest number of time in the task within an interval</td>
</tr>
<tr>
<td>totalmin</td>
<td>TotalMIN</td>
<td>Shortest time period in the task within an interval.</td>
</tr>
<tr>
<td>ratiomin</td>
<td>RatioMIN</td>
<td>Shortest ratio of time in the task within an interval.</td>
</tr>
<tr>
<td>countmin</td>
<td>CountMIN</td>
<td>Shortest number of time in the task within an interval</td>
</tr>
</tbody>
</table>
Basis Options

FILE Displays trace memory contents loaded with Trace.FILE command.

InterVal Divide the time period recorded by the trace (total) into time slices and analyze the time slices, see examples.

Accumulate By default only the current trace contents is analyzed. The option /Accumulate allows to add the current trace contents to the already displayed results.

INCremental Intermediate results are displayed while the TRACE32 software analyzes the trace contents (default).

FULL The result is displayed after the TRACE32 software finished the analysis.

Sort <item> By default the result is sorted by the recording order. Other sorting criteria are possible. The Config button provides a quick access. For a detailed overview refer to Trace.statistic.Sort.

CLOCKS The measurement results display the number of clocks instead of time information.

FlowTrace (Diagnosis only).

BusTrace (Diagnosis only).

SMP Options

CORE <n> TRACE32 analyzes the result only for the core with the specified number.

SplitCORE TRACE32 displays the result per core (default).

MergeCORE Trace information is analyzed independently for each core. The statistic summarizes these results to a single result.

JoinCORE (Unused).

Formatting

LEN <size> Specifies the width of non numeric fields (e.g. symbols)
Before using this command two things must be done:

1. the switch of the RTOS kernel must be marked for recording
2. the trigger unit has to be programmed

This trigger program performs the recording of all accesses to the magic word (a memory location that defines which task is running).

Mark the task switch and program the trigger unit (TASK configured)

If the TASK command is configured, the magic word (the memory location that defines which task is running) has to be marked with an AlphaBreak. The trigger program samples all write accesses to the address marked by an AlphaBreak.

Prepare the recording after using a TASK configuration

```plaintext
; Mark the magic location with an Alpha breakpoint
Break.Set task.config(magic)++(task.config(magicsize)-1) /Alpha

; Program the Analyzer to record only task switches
Analyzer.ReProgram
(    Sample.Enable if AlphaBreak&&Write
)
```

Mark the task switch and program the trigger unit (TASK not configured)

If the TASK command is not configured, the following procedure is necessary:

- Mark the memory location that defines which task is running by an AlphaBreak.
- Sample only write accesses to the address marked by an AlphaBreak to the trace.
- Mark the memory location that defines which task is running in the trace with a C marker.
Prepare the recording:

```plaintext
; Mark the magic location with an Alpha breakpoint
Break.Set v.range(current_task) /Alpha

; Program the Analyzer to record and mark only task switches
Analyzer.ReProgram
(
    Sample.Enable if AlphaBreak&&Write
    Mark.C if AlphaBreak&&Write
)
```

Recording

The trace memory should be cleared before starting the recording. The command `<trace>.Chart.TASK` displays a time chart of the running tasks.

Please see also the **OS Awareness Manuals** for the supported kernels.

See also

- `<trace>.STATistic.TASKFunc`
- `<trace>.STATistic`
- 'Release Information' in 'Release History'
- 'OS-Aware Tracing' in 'ARM-ETM Training'
- 'OS-Aware Tracing - Single-Core and AMP' in 'AURIX Trace Training'
- 'OS-Aware Tracing - SMP Systems' in 'AURIX Trace Training'
- 'OS-Aware Tracing' in 'Intel® Processor Trace Training'
- 'OS-Aware Tracing - Single Core' in 'Nexus Training'
- 'OS-Aware Tracing - SMP Systems' in 'Nexus Training'
The time spent in functions and the number of calls is measured. The application can run under a multitask kernel. Functions that are used by multiple tasks are displayed for each task.
The **INTR** option includes interrupt functions in the measurement like function calls. Without this option, the time spent in interrupt functions is taken out of the measurement.

The **INTRTASK** option takes interrupts out of the measurement, but displays the interrupt times for each task in a separate line.

| E::Trace.STATistic.TASKFunc ALL |
|-------------------------------|----------------|
| total: 2.469s intr: 17.966s   |
| range                        | time    | min  | max    | avr    | include | i   |
| (root)@MSG                   | 561.160us | 0.000 | 561.160us | 561.160us | 561.160us | 561 |
| (root)@SINK                  | 163.700ms | 0.000 | 163.700ms | 163.700ms | 163.700ms | 163 |
| (root)@SRCE                  | 121.730ms | 0.000 | 121.730ms | 121.730ms | 121.730ms | 121 |
| MO\_r_bench@MEM1             | 376.848ms | 5.341ms | 5.954ms | 5.541ms | 15.797ms | 232 |
| MO\_r_bench@MEM2             | 229.561ms | 5.341ms | 6.269ms | 5.599ms | 9.364ms | 228 |
| _r_dmy_read@IO1              | 23.679ms | 812.990us | 1.480ms | 845.698us | 23.679ms | 845 |
| _r_dmy_write@IO2             | 22.548ms | 839.985us | 1.184ms | 835.139us | 22.548ms | 835 |
| MO\_r_sieve@MEM1             | 361.051ms | 512.215us | 1.083ms | 530.957us | 361.051ms | 530 |
| MO\_r_sieve@MEM2             | 220.196ms | 512.210us | 818.615us | 537.065us | 220.196ms | 537 |

| E::Trace.STATistic.TASKFunc ALL |
|-------------------------------|----------------|
| total: 2.469s intr: 17.966s   |
| range                        | exclude | eavr | maxintr | count | ratio 1% |
| (root)@MSG                   | 0.000   | 0.000 | 1.249s  | 1.(-2) | 0.022% |
| (root)@SINK                  | 0.000   | 0.000 | 2.277s  | 1.(-2) | 6.629% |
| (root)@SRCE                  | 0.000   | 0.000 | 2.323s  | 1.(-2) | 4.929% |
| MO\_r_bench@MEM1             | 361.051ms | 5.309ms | 30.349ms | 68. | 0.639% |
| MO\_r_bench@MEM2             | 220.196ms | 5.370ms | 113.519ms | 41. | 0.379% |
| _r_dmy_read@IO1              | 0.000   | 0.000 | 127.073ms | 28.(-2) | 0.958% |
| _r_dmy_write@IO2             | 0.000   | 0.000 | 119.853ms | 27.(-2) | 0.913% |
| MO\_r_sieve@MEM1             | 0.000   | 0.000 | 30.349ms | 680. | 14.621% |
| MO\_r_sieve@MEM2             | 0.000   | 0.000 | 113.519ms | 410. | 8.917% |
The command Trace.STATistic.TASKFunc can be used, if the trace information output by the processor includes data write cycles.

The measurement is based on marking:

- All function entries and exits
- All task switches

**Mark the Functions**

Here for most CPUs it isn't possible to perform a selective trace. In order to perform the measurement set Alpha breakpoints implemented as software breakpoints to all function entries and Beta breakpoints implemented as software breakpoints to all function exits.

These breakpoints can be set automatically with the command Break.SetFunc, if HLL-functions are loaded. Assembler functions or e.g. loops within HLL-functions can also be marked manually.

The entry points of interrupt routines should be marked with Alpha and Charly breakpoints. Thus the analysis ignores interrupt function times and takes care about double fetches caused by interrupted programs.

If all interrupt routines are located in a specific memory range or within a specific HLL module the interrupt routines can be marked automatically by using the command Break.SetFunction <range>|<module>/INTR.

If there is a name convention for interrupt routines use the Symbol.ForEach command to mark the interrupt routines.

**Mark the Task Switch (TASK configured)**

If a TASK configuration is used, no extra preparations are necessary. If the processor allows to restrict the output to specific data write cycles, it is recommended to restrict the output to write cycles to task.config(magic)++(task.config(magicsize)-1) since more information can be sampled into the trace buffer.

**Mark the Task Switch (TASK not configured)**

If no TASK configuration is used, set an Alpha, Beta and Charly breakpoint implemented as software breakpoint to “current_task” (assuming “current_task” to be the variable which holds the ID of the current running task). The data present at this cycle will be used in the statistic to identify the task.

```
Break.Set v.range(current_task) /Alpha /Beta /Charly /Soft
```

If the time spent in the kernel should be excluded from the measurement the entry of the kernel has to be marked with a Charly breakpoint implemented as software breakpoint and exit of the kernel have to be marked with a Charly and a Beta breakpoint implemented as software breakpoint.
Recording

The complete program flow is recorded and the records of interest (function entries, function exits, interrupt entries and interrupt exits) are marked in the trace. The trace memory should be cleared before starting the recording. This is automatically done if Analyzer.AutoInit is ON.

Check the trace by viewing the record with <trace>.List MARK DEFault.

Procedure for Measurement for TRACE32-ICE and TRACE32-FIRE

The measurement is based on selective tracing of:

- all function entries and exits
- all task switches

Mark the Functions

In order to perform this measurement all function entries have to be marked with an Alpha breakpoint, and all function exits have to be marked with a Beta breakpoint. These breakpoints can be set automatically with the command Break.SetFunc, if HLL-functions are loaded. Assembler functions or e.g. loops within HLL-functions can also be marked manually.

The entry points of interrupt routines should be marked with Alpha and Charly (or Beta on ECC8) breakpoints. Thus the analysis ignores interrupt function times and takes care about double fetches caused by interrupted programs.

If all interrupt routines are located in a specific memory range or within a specific HLL module the interrupt routines can be marked automatically by using the command Break.SetFunction <range>| <module> /INTR.

If there is a name convention for interrupt routines use the sYmbol.ForEach command to mark the interrupt routines.

Mark the Task Switch (TASK configured)

If the TASK command is configured, the magic word (the memory location that defines which task is running) has to be marked with an AlphaBreak.

Mark the Task Switch (without TASK configured)

If the TASK command is not configured, the access cycle identifying the next running task has to be traced and marked in the trace by 'A'+‘B’+‘C’ markers. The data present at this cycle will be displayed in the statistic to identify the task. If the time spent in the kernel should be excluded from the measurement the entry and exit of the kernel have to be traced and marked with ‘C’ and ‘C’+‘B’.
Programming of the Trigger Unit

The trigger unit is programmed to record accesses to ALPHA, BETA or CHARLY breakpoints and mark the record in the trace buffer as follows:

- Function entries are marked with an A marker.
- Function exits are marked with a B marker.
- Interrupt entries are marked with an A and a C marker.
- Interrupt exits are marked with a B marker.
- Task switches are marked with an A, B and C marker (without TASK configuration only)
- Entries to the kernel are marked with a C marker (without TASK configuration only)
- Exits of the kernel are marked with a B and C marker (without TASK configuration only)

```
Analyzer.ReProgram
(
    Sample.Enable IF AlphaBreak||BetaBreak||CharlyBreak
    Mark.A IF AlphaBreak
    Mark.B IF BetaBreak
    Mark.C IF CharlyBreak
)
```

Recording

The trace memory should be cleared before starting the recording. Check the trace by viewing the record with `<trace>.List MARK List.TASK DEFault`.

A display of the nesting is possible with the command `<trace>.List FUNC FUNCR List.TASK` (this command uses the same strategy to determine function entries and exits). If the nesting of the functions is not correct (each function entry must have its corresponding function exit), the results will be incorrect and an error message will be displayed in the statistic window. If single records are wrong, the commands `<trace>.STATistic.Prefetch` or `<trace>.STATistic.Ignore` can be used to remove them from the measurement.

The command `<trace>.STATistic.TASKTREE` can also be used to verify the correct function nesting. The command `<trace>.Chart.TASKFunc` displays a time chart of the function usage.
<table>
<thead>
<tr>
<th>Basic &lt;options&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILE</td>
<td>Displays trace memory contents loaded with Trace.FILE.</td>
</tr>
<tr>
<td>FlowTrace</td>
<td>The trace works as flow trace. This option is usually not required.</td>
</tr>
<tr>
<td>BusTrace</td>
<td>The trace works as a bus trace. This option is usually not required.</td>
</tr>
<tr>
<td>Accumulate</td>
<td>By default only the current trace contents is analyzed by the statistic functions. The option /Accumulate allows to add the current trace contents to the already displayed results.</td>
</tr>
<tr>
<td>INCremental</td>
<td>Intermediate results are displayed while the TRACE32 software analyses the trace contents (default).</td>
</tr>
<tr>
<td>FULL</td>
<td>The result is displayed after the TRACE32 software finished the analysis.</td>
</tr>
<tr>
<td>Track</td>
<td>Track the Trace.STATistic window with other trace list windows (tracking to record number or time possible).</td>
</tr>
<tr>
<td>NoMerge</td>
<td>(For diagnosis purpose only).</td>
</tr>
</tbody>
</table>

| CTS | Use Context Tracking System to fill trace gaps and then perform the <trace>.STATistic.TASKFunc command. This is only useful for the ARM-ETM, SH4 and NEXUS. |
| INTR | The INTR option includes interrupt functions in the measurement like function calls. Without this option the time spent in interrupt functions is taken out of the measurement. |
| INTRTASK | The INTRTASK option takes interrupts out of the measurement, but displays the interrupt times for each task in a separate line. |
The `<list_items>` can be arranged by pushing the `Config` button in the `<trace>.STATistic.FUNC` window.

<table>
<thead>
<tr>
<th><strong>NAME</strong></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAME</td>
<td>Displays the function name. Interrupt functions are marked with ➗.</td>
</tr>
<tr>
<td>Time</td>
<td>Time between entry and exit summed over the complete measurement time. By default the time spent in interrupt routines is taken out of the measurement.</td>
</tr>
<tr>
<td>MIN</td>
<td>Shortest time from function entry to exit.</td>
</tr>
<tr>
<td>MAX</td>
<td>Longest time from function entry to exit.</td>
</tr>
<tr>
<td>AVerage</td>
<td>Average time from function entry to exit.</td>
</tr>
<tr>
<td>Internal</td>
<td>Time spent within the function (without called sub functions).</td>
</tr>
<tr>
<td>IAAverage</td>
<td>Average time spent in the function (without called sub functions).</td>
</tr>
<tr>
<td>IMIN</td>
<td>Shortest time spent in the function (without called sub functions).</td>
</tr>
<tr>
<td>IMAX</td>
<td>Longest time spent in the function (without called sub functions).</td>
</tr>
<tr>
<td>External</td>
<td>Time spent within sub functions.</td>
</tr>
<tr>
<td>EAverage</td>
<td>Average time spent within sub functions.</td>
</tr>
<tr>
<td>EMIN</td>
<td>Shortest time spent within sub functions.</td>
</tr>
<tr>
<td>EMAX</td>
<td>Longest time spent within sub functions.</td>
</tr>
<tr>
<td>MAXIntr</td>
<td>Maximum time the function was interrupted by an interrupt.</td>
</tr>
<tr>
<td>MAXTask</td>
<td>Time in interrupt routines for each task.</td>
</tr>
<tr>
<td>Count</td>
<td>Number of calls of the function. A negative value in brackets shows the number of not complete entries or exits to the function (i.e. when the measurement is stopped).</td>
</tr>
<tr>
<td>Ratio</td>
<td>Ratio of time spent within the function over the complete measurement time (without called subroutines).</td>
</tr>
<tr>
<td>BAR.log, BAR.LIN</td>
<td>Graphical display of the ratio (linear or logarithmic).</td>
</tr>
<tr>
<td>TRatio</td>
<td>Ratio of time spent within the function over the complete measurement time (called subroutines included).</td>
</tr>
<tr>
<td>TBAR.log, TBAR.LIN</td>
<td>Graphical display of the ratio (linear or logarithmic).</td>
</tr>
</tbody>
</table>
**Formats**

 LEN <size>  Specifies the width of non numeric fields (e.g. symbols)

 TimeAuto  Adapt the time display. (default)

 TimeFixed  Display all time information in seconds.

**See also**

- <trace>.STATistic.TASK
- <trace>.STATistic
- 'Release Information' in 'Release History'
The command **Trace.STATistic.TASKKernel** refines the command **Trace.STATistic.TASK** for RTOS systems that don’t assign a task ID to the kernel. In such a case no task run-time is calculated for the kernel if the command **Trace.STATistic.TASK** is used.

If the TRACE32 TASK awareness was configured, TRACE32 implies that the kernel writes the identifier of the current task to the address **TASK.CONFIG(magic)**.

```plaintext
PRINT TASK.CONFIG(magic)
```
Measurement performed by Trace.STATistic.TASK (no task ID for the kernel):

The refined measurement of Trace.STATistic.TASKKernel requires that the kernel entries and kernel exits are marked by the command **sYmbol.NEW.MARKER**.

```
sYmbol.NEW.MARKER KENTRY os_prologue ; mark the address os_prologue as kernel entry point
sYmbol.NEW.MARKER KEXIT os_epilogue ; mark the address os_epilogue as kernel exit point
sYmbol.List.MARKER ; list all markers
```
Advanced example for RTOS RTXC on a StarCore CPU:

```
; mark all interrupt service routines as kernel entries
symbol.ForEach "symbol.NEW.MARKER KENTRY ** _isr_*"

; mark all RTE instructions in the specified program range as kernel exit
Data.Find P:RTXCProlog--P:RTXCProlog_end %Word 0x9f73
WHILE FOUND()
    (symbol.NEW.MARKER KEXIT P:TRACK.ADDRESS())
Data.Find
)
symbol.List.MARKER
```

If the processor allows to restrict the trace information output to the program flow and specific write accesses, it is recommended to restrict the output to the program flow plus write cycles to task.config(magic), since more information can be recorded into the trace buffer.

```
Break.Set TASK.CONFIG(magic) /Write /TraceData
Go
Break
Trace.STATistic.TASKKernel
```

See also
- `<trace>.STATistic`
- 'Release Information' in 'Release History'

©1989-2019 Lauterbach GmbH
### <trace>.STATistic.TASKORINTERRUPT

**Statistic of interrupts and tasks**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;trace&gt;.STATistic.TASKORINTERRUPT</th>
</tr>
</thead>
</table>

tbd.

See also

- <trace>.STATistic
- ‘Release Information’ in ‘Release History’

### <trace>.STATistic.TASKSRV

**Analysis of time in OS service routines**

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;trace&gt;.STATistic.TASKSRV [{%&lt;format&gt;}] [items] [/&lt;option&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;option&gt;:</td>
<td>FILE</td>
</tr>
</tbody>
</table>

The time spent in OS service routines and the number of calls is measured.

This feature is only available, if an OSEK/ORTI system is used, and if the OS Awareness is configured with the TASK.ORTI command.

See also

- <trace>.STATistic
- ‘OS-Aware Tracing - Single-Core and AMP’ in ‘AURIX Trace Training’
- ‘OS-Aware Tracing - SMP Systems’ in ‘AURIX Trace Training’
- ‘Release Information’ in ‘Release History’

©1989-2019 Lauterbach GmbH
The time tasks spent in different states is measured.

Before using that function the task state transitions must be sampled by the trace. This feature is highly dependent on the used RTOS kernel, and needs the TASK to be configured. Please see kernel specific “OS Awareness Manuals” manuals for more information.
**E::Trace.STATistic.TASKState**

<table>
<thead>
<tr>
<th>task</th>
<th>time.und</th>
<th>time.run</th>
<th>time.rdy</th>
<th>time.wait</th>
<th>time.susp</th>
</tr>
</thead>
<tbody>
<tr>
<td>(root)</td>
<td>0.000</td>
<td>0.000</td>
<td>1.954s</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>IDLE</td>
<td>252.049ms</td>
<td>78.787ms</td>
<td>1.623s</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>IO1</td>
<td>10.340ms</td>
<td>100.968ms</td>
<td>345.670ms</td>
<td>653.006ms</td>
<td>844.202ms</td>
</tr>
<tr>
<td>IO2</td>
<td>11.186ms</td>
<td>247.481ms</td>
<td>153.662ms</td>
<td>724.730ms</td>
<td>817.127ms</td>
</tr>
<tr>
<td>MEM1</td>
<td>50.364ms</td>
<td>389.512ms</td>
<td>499.453ms</td>
<td>1.014s</td>
<td>0.000</td>
</tr>
<tr>
<td>MEM2</td>
<td>0.000</td>
<td>454.146ms</td>
<td>1.209s</td>
<td>0.000</td>
<td>290.577ms</td>
</tr>
<tr>
<td>MSG</td>
<td>420.354ms</td>
<td>1.654ms</td>
<td>132.875us</td>
<td>0.000</td>
<td>1.532s</td>
</tr>
<tr>
<td>SINK</td>
<td>20.342ms</td>
<td>371.914ms</td>
<td>341.089ms</td>
<td>1.220s</td>
<td>0.000</td>
</tr>
<tr>
<td>SRCE</td>
<td>27.991ms</td>
<td>303.206ms</td>
<td>7.734ms</td>
<td>1.615s</td>
<td>0.000</td>
</tr>
</tbody>
</table>

**E::Trace.STATistic.TASKState**

<table>
<thead>
<tr>
<th>task</th>
<th>max.und</th>
<th>max.run</th>
<th>max.rdy</th>
<th>max.wait</th>
<th>max.susp</th>
</tr>
</thead>
<tbody>
<tr>
<td>(root)</td>
<td>0.000</td>
<td>0.000</td>
<td>1.954s</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>IDLE</td>
<td>252.049ms</td>
<td>23.346ms</td>
<td>426.751ms</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>IO1</td>
<td>10.337ms</td>
<td>4.598ms</td>
<td>27.738ms</td>
<td>50.386ms</td>
<td>64.353ms</td>
</tr>
<tr>
<td>IO2</td>
<td>11.181ms</td>
<td>11.596ms</td>
<td>17.166ms</td>
<td>47.320ms</td>
<td>66.393ms</td>
</tr>
<tr>
<td>MEM1</td>
<td>50.360ms</td>
<td>22.228ms</td>
<td>29.950ms</td>
<td>87.455ms</td>
<td>0.000</td>
</tr>
<tr>
<td>MEM2</td>
<td>0.000</td>
<td>35.941ms</td>
<td>78.177ms</td>
<td>0.000</td>
<td>68.315ms</td>
</tr>
<tr>
<td>MSG</td>
<td>420.328ms</td>
<td>805.625us</td>
<td>42.000us</td>
<td>0.000</td>
<td>1.229s</td>
</tr>
<tr>
<td>SINK</td>
<td>20.338ms</td>
<td>8.103ms</td>
<td>11.523ms</td>
<td>83.515ms</td>
<td>0.000</td>
</tr>
<tr>
<td>SRCE</td>
<td>27.991ms</td>
<td>7.373ms</td>
<td>1.085s</td>
<td>92.603ms</td>
<td>0.000</td>
</tr>
</tbody>
</table>

**E::Trace.STATistic.TASKState**

<table>
<thead>
<tr>
<th>task</th>
<th>avr.und</th>
<th>avr.run</th>
<th>avr.rdy</th>
<th>avr.wait</th>
<th>avr.susp</th>
</tr>
</thead>
<tbody>
<tr>
<td>(root)</td>
<td>0.000</td>
<td>0.000</td>
<td>1.954s</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>IDLE</td>
<td>252.049ms</td>
<td>7.162ms</td>
<td>147.577ms</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>IO1</td>
<td>10.340ms</td>
<td>1.628ms</td>
<td>4.215ms</td>
<td>31.095ms</td>
<td>20.100ms</td>
</tr>
<tr>
<td>IO2</td>
<td>11.186ms</td>
<td>4.057ms</td>
<td>1.873ms</td>
<td>34.510ms</td>
<td>19.929ms</td>
</tr>
<tr>
<td>MEM1</td>
<td>50.364ms</td>
<td>9.058ms</td>
<td>8.055ms</td>
<td>50.742ms</td>
<td>0.000</td>
</tr>
<tr>
<td>MEM2</td>
<td>0.000</td>
<td>8.410ms</td>
<td>22.397ms</td>
<td>0.000</td>
<td>20.755ms</td>
</tr>
<tr>
<td>MSG</td>
<td>420.354ms</td>
<td>413.531us</td>
<td>33.218us</td>
<td>0.000</td>
<td>306.409ms</td>
</tr>
<tr>
<td>SINK</td>
<td>20.342ms</td>
<td>4.375ms</td>
<td>2.842ms</td>
<td>33.912ms</td>
<td>0.000</td>
</tr>
<tr>
<td>SRCE</td>
<td>27.991ms</td>
<td>3.002ms</td>
<td>54.854us</td>
<td>19.941ms</td>
<td>0.000</td>
</tr>
</tbody>
</table>

**E::Trace.STATistic.TASKState**

<table>
<thead>
<tr>
<th>task</th>
<th>c.und</th>
<th>c.run</th>
<th>c.rdy</th>
<th>c.wait</th>
<th>c.susp</th>
<th>ratio 1%</th>
<th>2%</th>
</tr>
</thead>
<tbody>
<tr>
<td>(root)</td>
<td>1.0</td>
<td>1.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.000%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDLE</td>
<td>1.0</td>
<td>11.</td>
<td>11.</td>
<td>0.0</td>
<td>4.031%</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IO1</td>
<td>1.</td>
<td>62.</td>
<td>82.</td>
<td>21.</td>
<td>42.</td>
<td>5.166%</td>
<td></td>
</tr>
<tr>
<td>IO2</td>
<td>1.</td>
<td>61.</td>
<td>82.</td>
<td>21.</td>
<td>41.</td>
<td>12.664%</td>
<td></td>
</tr>
<tr>
<td>MEM1</td>
<td>1.</td>
<td>43.</td>
<td>62.</td>
<td>20.</td>
<td>0.</td>
<td>19.932%</td>
<td></td>
</tr>
<tr>
<td>MEM2</td>
<td>1.</td>
<td>54.</td>
<td>54.</td>
<td>0.</td>
<td>14.</td>
<td>23.239%</td>
<td></td>
</tr>
<tr>
<td>MSG</td>
<td>1.</td>
<td>4.</td>
<td>4.</td>
<td>0.</td>
<td>5.</td>
<td>0.084%</td>
<td></td>
</tr>
<tr>
<td>SINK</td>
<td>1.</td>
<td>85.</td>
<td>120.</td>
<td>36.</td>
<td>0.</td>
<td>19.031%</td>
<td></td>
</tr>
<tr>
<td>SRCE</td>
<td>1.</td>
<td>101.</td>
<td>141.</td>
<td>81.</td>
<td>0.</td>
<td>15.515%</td>
<td></td>
</tr>
<tr>
<td>Basic &lt;options&gt;</td>
<td>Description</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FILE</strong></td>
<td>Displays trace memory contents loaded with Trace.FILE.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FlowTrace</strong></td>
<td>The trace works as flow trace. This option is usually not required.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BusTrace</strong></td>
<td>The trace works as a bus trace. This option is usually not required.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Accumulate</strong></td>
<td>By default only the current trace contents is analyzed by the statistic functions. The option /Accumulate allows to add the current trace contents to the already displayed results.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>INCremental</strong></td>
<td>Intermediate results are displayed while the TRACE32 software analyses the trace contents (default).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FULL</strong></td>
<td>The result is displayed after the TRACE32 software finished the analysis.</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Track</strong></td>
<td>Track the Trace.STATistic window with other trace list windows (tracking to record number or time possible).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>NoMerge</strong></td>
<td>(For diagnosis purpose only).</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
List items

**TIME[.<state>]**
- The total time the task was in this state.

**MAX[.<state>]**
- The maximum time the task was in this state.

**AVerage[.<state>]**
- The average time the task was in this state.
  **NOTE:** This value can be wrong if intermediate states exist.

**Count[.<state>]**
- The number of times a state was entered.
  **NOTE:** This value can be wrong if intermediate states exist.

**Ratio**
- The ratio of CPU runtime consumed by this task.

**BAR.log, BAR.LIN**
- Graphical display of ratio column.

Possible <state> are: **UNDefined**, **RUNning**, **ReaDY**, **WAITing**, **SUSPended**.

**See also**
- `<trace>.STATistic`
- 'Release Information' in 'Release History'
### Format:

```
<trace>.STATistic.TASKTREE [%<format>] [<list_items> …] [/<option>]
```

#### <format>:
- DEFault
- LEN
- TimeAuto
- TimeFixed

#### <list_items>:
- DEFault
- ALL
- Total
- MIN
- MAX
- AVErAge
- Count
- NAME
- TASK
- Internal
- IAVErAge
- IMIN
- IMAX
- InternalRatio
- InternalBAR
- External
- EAVErAge
- EMIN
- EMAX
- INTR
- INTRMAX
- INTRCount
- ExternalTASK
- ExternalTASKMAX
- TASKCount
- TotalRatio
- TotalBAR

#### <option>:
- FILE
- FlowTrace
- BusTrace
- Accumulate
- INCremental
- FULL
- Sort <item>
- Track
- NoMerge
- IncludeOwn
- IncludeTASK
- IncludeINTR
- INTRROOT
- INTRTASK

---

The results of this command shows a graphical tree of the function nesting. The measurement is done like for the command `Analyzer.STATistic.TASKFunc`.

For a description of `<format>`, `<list_items>`, and `<option>`, refer to `<trace>.STATistic.TASKFunc`. 
### E::Analyzer.STATistic.TASKTREE

<table>
<thead>
<tr>
<th>time</th>
<th>min</th>
<th>max</th>
<th>avr</th>
</tr>
</thead>
<tbody>
<tr>
<td>rnel) (root)@(kernel)</td>
<td>786.129ms</td>
<td>0.000</td>
<td>786.129ms</td>
</tr>
<tr>
<td>root) (root)</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td>@IDLE (root)</td>
<td>35.053ms</td>
<td>0.000</td>
<td>35.053ms</td>
</tr>
<tr>
<td>@IO1 (root)</td>
<td>72.308ms</td>
<td>0.000</td>
<td>72.308ms</td>
</tr>
<tr>
<td>@IO1 _r_dmy_read@IO1</td>
<td>22.760ms</td>
<td>813.000us</td>
<td>1.287ms</td>
</tr>
<tr>
<td>@IO2 (root)</td>
<td>247.015ms</td>
<td>0.000</td>
<td>247.015ms</td>
</tr>
<tr>
<td>@IO2 _r_dmy_write@IO2</td>
<td>21.341ms</td>
<td>839.995us</td>
<td>849.375us</td>
</tr>
<tr>
<td>@MEM1 (root)</td>
<td>437.910ms</td>
<td>0.000</td>
<td>437.910ms</td>
</tr>
<tr>
<td>@MEM1 _r_bench@MEM1</td>
<td>370.363ms</td>
<td>5.341ms</td>
<td>5.941ms</td>
</tr>
<tr>
<td>@MEM1 _r_sieve@MEM1</td>
<td>355.344ms</td>
<td>512.245us</td>
<td>804.010us</td>
</tr>
<tr>
<td>@MEM2 (root)</td>
<td>555.178ms</td>
<td>0.000</td>
<td>555.178ms</td>
</tr>
<tr>
<td>@MEM2 _r_bench@MEM2</td>
<td>257.509ms</td>
<td>5.341ms</td>
<td>5.965ms</td>
</tr>
<tr>
<td>@MEM2 _r_sieve@MEM2</td>
<td>247.122ms</td>
<td>512.245us</td>
<td>1.104ms</td>
</tr>
</tbody>
</table>

See also
- ![trace].STATistic.TREE
- ![trace].STATistic

▲ ‘Release Information’ in ‘Release History’

### <trace>.STATistic.TASKVSINTERRUPT

Statistc of interrupts, task-related

<table>
<thead>
<tr>
<th>format</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;trace&gt;.STATistic.TASKVSINTERRUPT</td>
</tr>
</tbody>
</table>

tbd.

See also
- ![trace].STATistic

▲ ‘Release Information’ in ‘Release History’
The results of this command shows a graphical tree of the function nesting. The measurement is done like for the command `Analyzer.STATistic.TASKFunc`.

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;trace&gt;.STATistic.TREE</code> [%&lt;format&gt;] [[&lt;list_items&gt;]] [&lt;option&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;format&gt;:</td>
<td>DEFault</td>
</tr>
<tr>
<td></td>
<td>TimeAuto</td>
</tr>
<tr>
<td>&lt;list_items&gt;:</td>
<td>DEFault</td>
</tr>
<tr>
<td></td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td>Count</td>
</tr>
<tr>
<td></td>
<td>NAME</td>
</tr>
<tr>
<td></td>
<td>Internal</td>
</tr>
<tr>
<td></td>
<td>External</td>
</tr>
<tr>
<td></td>
<td>INTR</td>
</tr>
<tr>
<td></td>
<td>ExternalTASK</td>
</tr>
<tr>
<td></td>
<td>TotalRatio</td>
</tr>
<tr>
<td>&lt;option&gt;:</td>
<td>FILE</td>
</tr>
<tr>
<td></td>
<td>FlowTrace</td>
</tr>
<tr>
<td></td>
<td>Accumulate</td>
</tr>
<tr>
<td></td>
<td>INCremental</td>
</tr>
<tr>
<td>Sort &lt;item&gt;:</td>
<td></td>
</tr>
<tr>
<td>Track:</td>
<td>NoMerge</td>
</tr>
<tr>
<td>IncludeOwn</td>
<td>IncludeTASK</td>
</tr>
<tr>
<td>INTRROOT</td>
<td>INTRTASK</td>
</tr>
</tbody>
</table>
The specified record(s) are used for performance and nesting analysis. This command can be used, when some records should be used, which are ignored due to the `<trace>.STATistic.Ignore` or `<trace>.STATistic.PreFetch` commands.

See also

- `<trace>.STATistic.Use`
- `<trace>.STATistic.TASKTREE`
- '<Release Information' in 'Release History'
### <trace>.STREAMCompression

Select compression mode for streaming

**Format:**

```markdown
<trace>.STREAMCompression OFF | LOW | MID | HIGH
```

<table>
<thead>
<tr>
<th>Mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OFF</td>
<td>Trace information is streamed uncompressed and saved uncompressed to file on hard-disk (diagnose purpose only).</td>
</tr>
<tr>
<td>LOW</td>
<td>Trace information is streamed compressed (hardware compression for all ETMv3 and higher, software compression otherwise). Trace information is saved to file on hard-disk as received.</td>
</tr>
<tr>
<td>MID</td>
<td>Trace information is streamed compressed (hardware compression for all ETMv3s and higher, software compression otherwise). Trace information is zipped before it is saved to file on hard-disk.</td>
</tr>
<tr>
<td>HIGH</td>
<td>Trace information is streamed compressed (hardware compression for all ETMv3s and higher, software compression otherwise). Trace information is zipped very compact before it is saved to file on hard-disk.</td>
</tr>
</tbody>
</table>

**Example:**

```markdown
Trace.STREAMCompression LOW
Trace.Mode STREAM
```

**See also**

- `<trace>.STREAMFILE`
- `<trace>.STREAMFileLimit`
- `<trace>.STREAMLOAD`
- `<trace>.STREAMSAVE`
- `CIProbe`

©1989-2019 Lauterbach GmbH
**<trace>.STREAMFILE**

Specify temporary streaming file path

<table>
<thead>
<tr>
<th>Format:</th>
<th><code>&lt;trace&gt;.STREAMFILE &lt;file&gt;</code></th>
</tr>
</thead>
</table>

Set the path and file name for the temporary stream file. By default, TRACE32 will choose the standard TEMP folder set in the operating system. The only intention of this command is to select a different path / drive, e.g. a high-capacity drive dedicated for this use case.

The contents of the streaming file are in a proprietary format and not intended for use in external applications. Also it is not possible to use this file for later analysis.

In order to store a trace recording for later analysis, either use `Trace.SAVE` or `Trace.STREAMSAVE`. In order to export trace data for use in external applications, see `Trace.EXPORT`.

**Example:**

```
Trace.STREAMFILE d:\temp\mystream.t32 ; specify the location for your streaming file
Trace.Mode STREAM ; select the trace mode STREAM
```

**See also**

- `<trace>.STREAMFileLimit`
- `<trace>.STREAMLOAD`
- `CIProbe`
- `<trace>.STREAMCompression`
- `<trace>.STREAMSAVE`
<trace>.STREAMFileLimit

Set size limit for streaming file

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;trace&gt;.STREAMFileLimit +/- limit_in_bytes</th>
</tr>
</thead>
</table>

Sets the maximum size allowed for a streaming file. If the maximum size is exceeded, streaming is stopped.

The limit value is given in bytes and can have a positive or negative sign:

- **Positive value**: The maximum size of the streaming file in bytes
- **Negative value**: Specifies the amount of space to leave on the disk before stopping streaming. The maximum file size is calculated based on the amount of available disk space at the time of starting streaming.

The default setting is -1.000.000.000 i.e. stops streaming when less than a GB of space is left on the target storage medium.

**NOTE:** The file limit and the name of the streaming file must be set before starting streaming. Later changes are not taken into account.

Slow trace ports allow high compression. Fast trace ports allow no compression.

**Example:**

```plaintext
Trace.STREAMCompression LOW ; select software compression
Trace.Mode STREAM ; mode LOW for streaming file
```
Load **streaming file** to TRACE32. In order to display decompressed trace information the target state at the recording time has to be reconstructed within TRACE32. This can be very complicated especially if a software with an operating system that uses dynamic memory management to handle processes/tasks (e.g. Linux) is used.

**Example 1**: Reconstruction of the target state at the recording time for a bar metal Cortex-R4 application.

```
; specify the target CPU
SYSTEM.CPU TMS570PSFC61

SYStem.Option BigEndian ON

SYStem.Up

; specify ETM setting valid at the recording time
ETM.PortSize 16.
ETM.PortMode Bypass
ETM.DataTrace OFF
ETM.ContextID OFF
ETM.ON

; load source code and debug information
Data.LOAD.ELF I:\EVB\arm\tms570psfc61\demo.axf

; load saved streaming file
Trace.STREAMLOAD C:\T32_ARM\r4_max.sad

Trace.List
```
Example 2: Reconstruction of the target state at the recording time for NEXUS Power Architecture:

; specify the target CPU
SYSstem.CPU MPC5646C

; specify the NEXUS setting valid at the recording time
NEXUS.PortSize MDO12
NEXUS.PortMode 1/2
NEXUS.BTM ON
NEXUS.HTM ON
NEXUS.PTCM BL_HTM ON
NEXUS.ON

SYSstem.Up

; mapping logical to physical address is 1:1

; load source code and debug information
Data.LOAD.Elf J:\AND\PPC5XXXX_OTM\im02bf1x.elf

; load the OS Awareness
TASK.ORTI J:\AND\PPC5XXXX_OTM\im02bf1x.ort

; load saved streaming file
Trace.STREAMLOAD J:\AND\PPC5XXXX_OTM\my_stream

Trace.List

See also
■ <trace>.STREAMCompression
■ <trace>.STREAMFileLimit
■ <trace>.STREAMFILE
■ <trace>.STREAMSAVE
Save the **streaming file**. The default extension for the streaming file is `.sad`. Use `Trace.STREAMLOAD` to load the file for analysis.

The contents of the streaming file are in a proprietary format and not intended for use in external applications. Also it is not possible to use this file for later analysis.

In order to store a trace recording for later analysis, either use `Trace.SAVE` or `Trace.STREAMSAVE`. In order to export trace data for use in external applications, see `Trace.EXPORT`.

### See also
- `trace`.STREAMCompression
- `trace`.STREAMFileLimit
- `trace`.SAVE
- `trace`.STREAMFILE
- `trace`.STREAMLOAD

### `trace`.TCount

**Set trigger counter**

**Format:**

```plaintext
<value>: 0. … 16777215.
```

Sets the number of trigger events that will be ignored by the trace or logic analyzer, before a trigger event ends the recording (state: break). A counter value zero means that the recording stops immediately after the first trigger. A value of 1 halts the recording at the second trigger event, and so on.

**Trigger Signal**

```
| Trigger Counter = 0 | Trigger Counter = 1 | Trigger Counter = 2 |
```

©1989-2019 Lauterbach GmbH
**Format:**  
<trace>.TDelay <time> | <cycles> | <percent>%

- **<time>:** 0 ... 200.s
- **<percent>:** 0 ... 1000%
- **<cycles>:** 0 ... 4000000000.

Selects the delay time between trigger point and break (end of recording). Use this command in order to record events that occurred after the trigger point.

The trigger delay may also be defined in percent of the trace buffer size. The delay can be larger (up to 10x) than the total trace buffer size. Logic analyzers also support setting a delay time.

Analyzer.TDelay 40% ; trigger delay is 40% of trace depth.

Selects the delay time between trigger point and break of the port analyzer. The time can be larger than the time for a full sample of the analyzer. The trigger delay time may be defined in percent relating to the total trace time.

```
ARM  Pretrigg.Delay  Trigger point  TDelay time  Break

Trigger system active

Sampling
```
With a mouse click to the corresponding area in the port analyzer state window this command can be executed too.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Port.TDelay 10.ms</td>
<td>the trigger delay is 10 ms</td>
</tr>
<tr>
<td>Port.TDelay 50%</td>
<td>the trigger point is in the mid of the trace; memory</td>
</tr>
<tr>
<td>Port.TDelay 99%</td>
<td>the trigger point is at the beginning to the trace; memory</td>
</tr>
<tr>
<td>Port.TDelay 200.</td>
<td>the trigger point 200 record before end of trace</td>
</tr>
</tbody>
</table>

### Trigger delay

- **0%**: Trigger point is at the start of the trace.
- **25%**: Trigger point is in the middle of the trace.
- **50%**: Trigger point is at the end of the trace.
- **75%**: Trigger point is near the end of the trace.
- **100%**: Trigger point is at the end of the trace.
- **200%**: Trigger point is at the end of the trace.

See also

- `<trace>.state`
- `Trace`
By default the trace line termination of the preprocessor is used during a trace capture. Undefinable
FLOWERRORs may occur if the output drivers of the CPU are not strong enough. In this case it is
recommended to switch the trace line termination OFF.

See also
- <trace>.state
- Trace
The command `<trace>.TestFocus` tests the recording at a high-speed trace port.

The command `<trace>.TestFocus` can be used if:

- **The program execution is stopped.**
  
  To test the trace port, the test pattern generator of the trace port is used if available. Otherwise, a test program is loaded and started by TRACE32.

- **The program execution and the trace recording is running.**
  
  Testing the trace port while the application program is running might be helpful to detect trace port problems caused by the application program.
  
  The trace data from the application program are used to test the trace port. Here a reduced test scenario is processed that checks the correctness of the program flow recording and for shorts-circuits between the trace port lines. This test requires that the program code is loaded to the virtual memory.

  ```plaintext
  Data.LOAD.Elf arm.elf /PlusVM ; Load the application code
  ; to the target memories and
  ; to the virtual memory of
  ; TRACE32
  ```

- **The program execution is running and the trace recording is stopped.**

  To test the trace port, the test pattern generator of the trace port is used if available. Otherwise, the trace data from the application program are used.

If a test program is used, TRACE32 attempts to load the test program to the memory addressed by the PC or the stack pointer. It is also possible to define an `<address_range>` for the test program.

  ```plaintext
  Trace.TestFocus ; start trace port test
  Trace.AutoFocus 0x24000000++0xffff ; start the test and load
  ; test program to address
  ; 0x24000000
  ```
If TRACE32 is unable to load the test program the following error message is displayed:
“Don’t know where to execute the test code”.

By default, the original RAM content is restored after the trace port test and the trace recording is deleted.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accumulate</td>
<td>If the application program varies the CPU clock frequency, this affects also the trace port. In such a case it is recommended to overlay the test results for all relevant CPU clock frequencies by using the option /Accumulate.</td>
</tr>
<tr>
<td>Config</td>
<td>Allows to define a RAM address range for the download of the test program.</td>
</tr>
<tr>
<td>KEEP</td>
<td>After a trace port test the trace is cleared and any loaded test program is removed from the target RAM.</td>
</tr>
<tr>
<td></td>
<td>With the option /KEEP the test trace is not cleared and can be viewed with the Trace.List command. If a test program was loaded by TRACE32, it also remains in the target RAM.</td>
</tr>
<tr>
<td>ALTERNATE</td>
<td>If the trace port provides a test pattern generator, it is always used for the test. The option /ALTERNATE forces TRACE32 to use its own test program.</td>
</tr>
<tr>
<td>NoTraceControl</td>
<td>Informs the TRACE32 software that the trace control signal is not available on the trace connector.</td>
</tr>
</tbody>
</table>

; advise the command Trace.TestFocus to download the test program
; always to the address range 0x24000000++0xffff
Trace.TestFocus 0x24000000++0xffff /Config

The result of the command Trace.TestFocus can be processed in a PRACTICE script as follows:

```
Trace.TestFocus

IF FOUND()
    PRINT %ERROR "Trace port test failed"
ELSE
    PRINT "Trace port recording ok"
```
The **Trace.TestFocus** command calls the data eye finder for the current hardware configuration of a preprocessor with AUTOFOCUS technology and verifies the correctness of traced test data. In contrast to **Trace.AutoFocus**, the preprocessor configuration remains unchanged.

A complete trace port test executes the following steps:

1. The data eye finder is called. The source for the trace data for the test are the trace port's pattern generator, a test program or the application program.
2. When the eye finder is done, the test is started once again to verify the correctness of the trace recording.
3. The data eyes resulting from the <trace>.TestFocus command can be viewed in the <trace>.ShowFocus window.

**See also**
- <trace>.TestFocusEye
- <trace>.ShowFocus
- <trace>.ShowFocusEye
- Trace
- AUTOFOCUS.FREQUENCY()
- ‘Release Information’ in ‘Release History’

### <trace>.TestFocusClockEye

**Scan clock eye**

Format:  
<trace>.TestFocusClockEye [address_range] [//option>

<option>:

Accumulate | Config | KEEP | ALTERNATE | Utilisation | NoTraceControl

Scans the clock eye. To view the result, use the command **Trace.ShowFocusClockEye**.

**NOTE:** The NEXUS AutoFocus adapter does not support this feature.

<option>  
For a description of the options, see **Trace.TestFocus**.

**See also**
- <trace>.TestFocus
- <trace>.ShowFocus
- <trace>.ShowFocusEye
- <trace>.AutoFocus
- <trace>.ShowFocusClockEye
- <trace>.TestFocusEye
- AUTOFOCUS.OK()
Check signal integrity

Scans the data eye to determine the integrity of the electrical trace signals.

The command `Trace.TestFocusEye` starts an eye finder to test the quality of the trace signals, if a preprocessor with AUTOFOCUS technology is used. The test result can be displayed with the command `Trace.ShowFocusEye`. If the result shows that an individual trace signal has a significantly smaller data eye than other signals, the hardware layout should be checked to see if this signal shows any unusual features.

The test procedure and the options used by the command `Trace.TestFocusEye` are similar to the command `Trace.TestFocus`.

For a description of the options, see `Trace.TestFocus`.

The command `Trace.TestFocusEye` can also be used with PowerTrace Serial. For this tool no additional parameters (e.g. address_range) or options are available.

See also

- `<trace>.AutoFocus`
- `<trace>.ShowFocusClockEye`
- `<trace>.state`
- `<trace>.TestFocus`
- `<trace>.ShowFocus`
- `<trace>.ShowFocusEye`
- `Trace`
- `<trace>.TestFocusClockEye`

▲ 'Release Information' in 'Release History'
The command **Trace.THreshold** can be used to optimize the threshold level for the trace lines sampled via a TRACE32-Preprocessor (e.g. ARM-ETM, OCDS Level 2, AUD …). The optimization of the threshold level should result in less errors in the trace recording.

**VCC**

The preprocessor and the TRACE32 software measure the VCC of the target. 1/2 VCC is then automatically used as the threshold level for the trace lines. The result is also displayed in the THreshold field of the `<trace>.state` window.

**CLOCK**

The threshold level is changed until the duty cycle of the trace clock reaches a ratio of 1:1. This setting is only recommended if the trace clock has a duty cycle of 1:1. The result is displayed in the THreshold field of the `<trace>.state` window.

**<level>**

The threshold level can be entered directly.

```
Trace.THreshold VCC
Trace.THreshold CLOCK
Trace.THreshold 1.6 ; the unit is Volt
```
Enhanced parameters for the Preprocessor for ARM-ETM with AUTOFOCUS:

```plaintext
<clock> <data>
```

For the Preprocessor for ARM-ETM with AUTOFOCUS different threshold levels can be defined for the clock and the data lines

```plaintext
Trace.THreshold 0.86 0.79 ; the unit is Volt
```

See also
- `<trace>.state`
- Trace
- 'Release Information' in 'Release History'

### `<trace>.TimeStamp`

Configure timestamp usage of LOGGER trace

**LOGGER only**

<table>
<thead>
<tr>
<th>Format: LOGGER.TimeStamp OFF</th>
<th>Up</th>
<th>Down</th>
<th>Rate &lt;rate&gt;</th>
</tr>
</thead>
</table>

Configure timestamps for the **LOGGER** trace. The LOGGER trace record format includes a timestamp field for up to 48 bit timestamps. The direction and rate information passed by this command is required to convert the timestamp into the time in seconds.

**OFF** (default)

Disable timestamps. Use this setting if the LOGGER target code does not store timestamps in the LOGGER trace records. When this setting is used, the x-direction in chart views is the record number axis instead of the time axis.

**Up**

Enable timestamp counter, counting upwards. Use this setting if the LOGGER target code stores timestamps in the LOGGER trace records and if the timestamp increments with each timer tick.

**Down**

Enable timestamp counter, counting downwards. Use this setting if the LOGGER target code stores timestamps in the LOGGER trace records and if the timestamp decrements with each timer tick.

**Rate <rate>**

Frequency of the timestamp in ticks per second.

**AllCycles**

Set timestamp generation frequency.

- **ON | OFF**
  - **OFF** (default): Generate a single timestamp for 6 trace cycles.
  - **ON**: Generate dedicated timestamps for all trace cycles.
Example: The timestamp used by the LOGGER target code increments at a rate of 16 million per second (16 MHz):

```
LOGGER.TimeStamp Up
LOGGER.TimeStamp Rate 16000000.
```

```
<trace>.Timing Waveform of trace buffer
```

Format: 

```
<trace>.Timing [<record_range>] [[<items>]] [/<options>]
```

```
<option>:
FILE
Track
RecScale
TimeScale
TimeZero
TimeREF
```

Displays the trace memory contents like command `<trace>.List`, but in form of a timing display. As a default the external trigger channels are displayed.

**FILE**

Display trace memory contents loaded with `Trace.FILE`.

**Track**

The cursor in the `<trace>.Timing` window follows the cursor movement in other trace windows. Default is a time tracking. If no time information is available tracking to record number is performed.

The zoom factor of the `<trace>.Timing` window is retained, even if the trace content changes.

**RecScale**

Display trace in fixed record raster. This is the default.

**TimeScale**

Display trace as true time display, time relative to the trigger point.

**TimeZero**

Display trace as true time display, time relative to zero.

**TimeREF**

Display trace as true time display, time relative to the reference point.
### Buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom In T</td>
<td>Zooms in Trace by a factor of 2.</td>
</tr>
<tr>
<td>Zoom Out T</td>
<td>Zooms out Trace by a factor of 2.</td>
</tr>
<tr>
<td>Zoom Full T</td>
<td>Display the complete trace buffer in the window.</td>
</tr>
<tr>
<td>Goto …</td>
<td>Open an <code>&lt;trace&gt;.GOTO</code> dialog box.</td>
</tr>
<tr>
<td>Find …</td>
<td>Open an <code>&lt;trace&gt;.Find</code> dialog box.</td>
</tr>
<tr>
<td>Set Ref</td>
<td>Set an analyzer reference point to the current record.</td>
</tr>
<tr>
<td>Set Zero</td>
<td>Set the global time reference to the current record.</td>
</tr>
<tr>
<td>View</td>
<td>Display all information about the current record (<code>&lt;trace&gt;.View</code>).</td>
</tr>
<tr>
<td>List</td>
<td>Open an <code>&lt;trace&gt;.List</code> window.</td>
</tr>
</tbody>
</table>

### Examples:

```plaintext
; Open Port Analyzer timing window in standard display format
E::Port.Timing

; Open Port Analyzer timing window and display last file loaded with
; " Port.FILE <file>"
E::Port.Timing /FILE

; Open Port Analyzer timing window starting at record -100. in standard
display format
E::Port.Timing -100. DEFault
```

### See also

- `<trace>.List`
- `<trace>.REF`
- `<trace>.View`
- Analyzer.RECORD.ADDRESS()
- Analyzer.RECORD.OFFSET()
- Analyzer.REF()

- ‘Release Information’ in ‘Release History’
- ‘Displaying the Trace’ in ‘Training FIRE Analyzer’
- ‘Emulator Functions’ in ‘FIRE User's Guide’
Select trigger mode

**Format:**

```
<trace>.TMode [High | Low | Rising | Falling]
```

Selects the trigger condition, edge or level trigger and the corresponding line polarity. The edge trigger is asynchronous and needs a minimum pulse width of 20 ns.

**Example:**

```
E::Port.TMode Rising
E::Port.TSelect Port
E::Port.Select Port.00 ; trigger on the rising edge of P.00
E::Port.TDelay 100.us ; sample till 100.us after trigger
```

**See also**
- Trace

---

Enable trigger output line (PowerIntegrator)

**Format:**

```
Integrator.TOut ON | OFF
```

Enables or disables the trigger output line of the integrator.

**See also**
- Trace
**<trace>.TPreDelay**

Pre-trigger delay

Format: `<trace>.TPreDelay [ <percent> | <time> | DEFAULT]`

- `<percent>`: 0 … 1000%
- `<time>`: 0 … 200.s
- `<cycles>`: 0 … 4000000000.

Selects a delay time between the start of the recording and the release of the trigger system. This delay is very useful if previous events before a trigger point are of interest. The trigger event is immediately valid after the arming of the analyzer. If the port analyzer is slave to the emulator, usually no delay will be needed.

| DEFAULT | The trigger predelay is max. 10% of the selected trace depth. |

```
ARM  Delay time  Trigger point  TDelay time  Break
```

Trigger system active

Sampling

With a mouse click to the corresponding area in the port analyzer state window that command can be executed, too.

**See also**

- **Trace**

---

**<trace>.TraceChannel**

tbd.

Format: `<trace>.TraceChannel`

©1989-2019 Lauterbach GmbH
If the AutoFocus preprocessor is connected to a Single-Wire-Trace module (i.e. Trace.PortType SWV), then this command is used to specify the bit rate. The <data_rate> is then set to 1/(min-bit-size).

See also

- Trace
- ‘Release Information’ in ‘Release History’

<trace>.TraceCONNECT

Select on-chip peripheral sink

Format:  <trace>.TraceCONNECT <component>
<trace>.TraceCONNECT NONE
<trace>.TraceCONNECT AUTO

Default: AUTO.

Selects the on-chip peripheral used as trace sink on the SoC.

Example: The two ETFs of an ARM CoreSight based SoC are selected as trace sink.

;note that the two approaches to select the first ETF are equivalent:
Onchip.TraceCONNECT ETF1 ; selects the ETF1 as onchip-trace sink
; or
Trace.METHOD Onchip
Trace.TraceCONNECT ETF1 ; selects the ETF1 as onchip-trace sink

;note that the two approaches to select the second ETF are equivalent:
Onchip.TraceCONNECT ETF2 ; selects the ETF2 as onchip-trace sink
; or
Trace.METHOD Onchip
Trace.TraceCONNECT ETF2 ; selects the ETF2 as onchip-trace sink

See also

- <trace>.state
- Trace
<trace>.TRACK

Sets the tracking record to the specified trace bookmark, time, or record number. The blue cursor moves to the specified destination in all Trace.* windows opened with the /Track option. All other Trace.* windows opened without the /Track option do not respond to the <trace>.TRACK command.

Example:

; set the tracking record to the record -12000.
Trace.TRACK -12000.

; without /Track: this window does not respond to the Trace.TRACK command
Trace.List %TimeFixed Time.Zero Default

; with /Track: this window responds to the Trace.TRACK command, i.e. the ;blue cursor selects the tracking record -12000.
Trace.List %TimeFixed Time.Zero Default /Track

; display only selected trace information about the record currently ; selected in the Trace.List ... /Track window
Trace.View %TimeFixed Time.Zero Default /Track

See also
- <trace>.GOTO
- CiProbe
- TRACK.ADDRESS()
- TRACK.RECORD()
- TRACK.TIME()
- Trace
- BookMark

<trace>.TRIGGER

Triggers the trace

Format: <trace>.TRIGGER

Forces a manual trigger.

See also
- Trace
Selects the trigger source for the port analyzer.

### Format

```
<trace>.TSELegt [<source>]
```

<table>
<thead>
<tr>
<th>&lt;source&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BusA [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>BusB [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>BusC [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>BusD [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>EXT [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>Port [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>Trigger [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>Break [ON</td>
<td>OFF]</td>
</tr>
<tr>
<td>Always [ON</td>
<td>OFF]</td>
</tr>
</tbody>
</table>

**See also**

- Trace
The command `Integrator.TSYNC.SIMPLE` resets the values of TPreDelay, TWidth, TCount and TDelay to defaults. Command `Integrator.TSYNC.SELect` doesn't modify these values.

The trigger signal can be generated out of the 204 port channels. Each signal can be qualified as high, low, rising or falling edge.

More than 1 edge can be combined to a trigger word. To detect a valid combination of edges, the edges must have a max. skew of 4 ns.

Edges and state signals can be combined. The state signal must be stable 4 ns before the edge. The sampling of the state signal is guaranteed before the edge is detected.

See also
- Trace
- 'Release Information' in 'Release History'

©1989-2019 Lauterbach GmbH
**<trace>.TView**

Display trigger settings

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;trace&gt;.TView</th>
</tr>
</thead>
</table>

The state of the trigger unit and the trigger settings are displayed.

- **level**
  Indicates the encountered logical level of the current trigger program.

- **setup file**
  Indicates the file name containing the trigger unit program. By clicking on that field will open an editor window for modification.

- **symbol, flag, counter**
  Indicates the values and symbolic names of the flags and counters of the trigger unit.

---

**<trace>.TWidth**

Set trigger filter

<table>
<thead>
<tr>
<th>Format:</th>
<th>&lt;trace&gt;.TWidth [value]</th>
</tr>
</thead>
<tbody>
<tr>
<td>value:</td>
<td>0 ... 2.5 us</td>
</tr>
</tbody>
</table>

The trigger filter time is defined. All trigger events which are shorter than this value are ignored.

**Example:**

i.s i.a20 high
i.s i.rxd rising
<trace>.View

Display single record

Format:  
<trace>.View [<record>] [<channels>] [/<options>]

<option>:
FILE
Track

Displays one record in a more detailed format. The syntax of the channel definitions is the same as for the <trace>.List command. Without arguments all channels are displayed.

Example 1:

;display all information about a specific record, here record -12000.  
Trace.View -12000.

;display only selected trace information about a specific record  
Trace.View -12000. TIme.ZERO DEFault CORE

Example 2:

;open a Trace.List window with all records. Display the ti.zero column  
;as the first column, followed by the DEFault columns  
Trace.List TIme.ZERO DEFault /Track

;display only selected trace information about the record currently  
;selected in the Trace.List window  
Trace.View TIme.Zero DEFault CORE /Track

See also

- <trace>.List
- <trace>.state
- Trace
- Analyzer.RECORD.DATA()
- Analyzer.RECORDS()
- Analyzer.SIZE()
- <trace>.REF
- <trace>.Timing
- Analyzer.RECORD.ADDRESS()
- Analyzer.RECORD.OFFSET()
- Analyzer.REF()
The cross system tracking allows to use the **Track** option for all trace windows between multiple emulators. This allows to correlate events in the analyzer in a multiprocessor environment. The communication between the different emulators is done by the **InterCom** system. **InterCom** allows the exchange of data between different emulators. The InterCom name is either the name and port number of a UDP port used by another emulator (see Installation Manuals for details) or the file name used by another emulator (see command **InterCom** for details). Using TCP/IP for communication is preferred. The tracking can be based on two different synchronization techniques: full time correlation between the system with a synch cable or triggering the systems at the same time. When a synch cable is used and the **SYnch** command is turned on, then the time stamps of all systems will show the same time. If the synch cable is not available then the emulators may be triggered by an event that is available on all units. The tracking assumes that the trigger event, sampled by the analyzers of all emulators, was sampled at the same time on all systems. Triggering all systems at the same time can be done by using trigger input and output lines. The bidirectional coaxial trigger A line of SA120 and HA120 units are very useful for triggering in multiprocessor systems.

**NOTE:** As the time bases of the systems are then not correlated, long measurements can lead to small errors, due to inexact local times.

The following examples uses TCP/IP for the cross tracking:

```
emulator 1          emulator 2
'Analyzer.XTrackste:20002'  'Analyzer.XTrackste:20001'
```

If TCP/IP is not available the communication can by made by a file:

```
emulator 1          emulator 2
InterCom c:\tmp\emu_1
'Analyzer.XTrackc:\tmp\emu_2'
```

The following analyzer trigger program can serve as a template:

```
Out.A IF <local_trigger_condition> ; trigger all emulators if the local trigger condition arrives
Trigger.A IF inouta ; local trigger if any processors in the system triggers
```

**See also**
- [trace].state
- Trace
Align timestamps of trace and timing analyzers

Format: `<trace>.ZERO [<time> | <record> | "<trace_bookmark>"]`

Use this command to align the zero time point for trace and timing analyzer sources with time bases of different origin.

- `<time>` Moves the ZERO point by specified time.
- `<record>` Sets the ZERO point to the time index of the specified record number.
- `<bookmark>` Sets the ZERO point to the time index of the specified bookmark location. You can create trace bookmarks with the `<trace>.BookMark` command.
- `no parameter` Reset zero time point back to initial location.

The table below shows the different sources for time information. As the different sources are not related, they all have an individual zero time point.

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>Trace data source</th>
<th>Original zero time point</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timestamps generated by TRACE32 hardware</td>
<td>Analyzer (no processor generated timestamps) PowerProbe, Integrator, IProbe, etc.</td>
<td>Permanently set to beginning of first debug session or trace recording after starting up TRACE32 PowerView. All trace data sources using TRACE32 hardware generated timestamps have a common zero time point.</td>
</tr>
<tr>
<td>Timestamps generated by target processor</td>
<td>Onchip Trace Analyzer Trace (with processor generated timestamps enabled)</td>
<td>Depends on CPU architecture and trace protocol. Starting a new trace recording usually moves the zero time point to a new location.</td>
</tr>
<tr>
<td>Timestamps loaded from files.</td>
<td>Trace.LOAD <code>&lt;file&gt;/FILE</code></td>
<td>Same as in original recording</td>
</tr>
</tbody>
</table>

Due to the different zero time points of the various data sources, it is required to align the zero time points, before trace or timing recordings can be observed in a correlated manner. This is usually achieved by locating a common event in the different sources and selecting this event as common zero time point.

Example tbd.

See also
- `<trace>.state`
- `Trace`

©1989-2019 Lauterbach GmbH

General Commands Reference Guide T 451
Using the **TRACEPORT** command group, you can configure the communication between the target trace port and the TRACE32 PowerTrace tool. Logically the **TRACEPORT** command group is located between the physical pins of the target platform and the TRACE32 trace input stage (preprocessor), see illustration below.

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

### See also
- **TRACEPORT.EndsKiP**
- **TRACEPORT.LaneCount**
- **TRACEPORT.LaneSpeed**
- **TRACEPORT.MsgBitEndian**
- **TRACEPORT.MsgByteEndian**
- **TRACEPORT.MsgLongEndian**
- **TRACEPORT.MsgWordEndian**
- **TRACEPORT.PinReMap**
- **TRACEPORT.RESet**
- **TRACEPORT.StartsKiP**
- **TRACEPORT.state**
TRACEPORT.EndsWithKiP  Define number of bytes skipped at the end of frame
For serial trace ports (AURORA) only

Format:  TRACEPORT.EndsWithKiP [<option>]

<option>:  AUTO | 0 | 2 | 8

Allows to cut off data bytes at the end of each data packet or data frame. Depending on the target configuration, the last bytes of a frame contain CRC information, which is not used by TRACE32. With the command TRACEPORT.EndsWithKiP it is possible to remove the unused bytes.

<table>
<thead>
<tr>
<th>OPTION</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO</td>
<td>TRACE32 defines the number of bytes to be cut.</td>
</tr>
<tr>
<td>0</td>
<td>Don’t cut any bytes.</td>
</tr>
<tr>
<td>2</td>
<td>Cut 2 bytes at the end of each frame.</td>
</tr>
<tr>
<td>8</td>
<td>Cut 8 bytes at the end of each frame.</td>
</tr>
</tbody>
</table>

See also
- TRACEPORT
- TRACEPORT.state
TRACEPORT.LaneCount
Select port size of the trace port
For serial trace ports (AURORA/PCIe) only

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRACEPORT.LaneCount &lt;size&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;size&gt;:</td>
<td>AUTO</td>
</tr>
</tbody>
</table>

Specifies the number of used lanes for the trace port. The number must match the target configuration, else the trace link between the target and the TRACE32 hardware cannot be established.

<table>
<thead>
<tr>
<th>AUTO</th>
<th>TRACE32 defines the lane count.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1Lane, 2Lane, 3Lane, 4Lane, 5Lane, 6Lane, 7Lane, 8Lane</td>
<td>Number of used lanes. In case of PCIe the lane setup will be done automatically.</td>
</tr>
</tbody>
</table>

See also
- TRACEPORT
- TRACEPORT.state

TRACEPORT.LaneSpeed
Inform debugger about trace port frequency
For serial trace ports (AURORA/PCIe) only

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRACEPORT.LaneSpeed &lt;data_rate&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;data_rate&gt;:</td>
<td>AUTO</td>
</tr>
</tbody>
</table>

Informs the debugger about the lane <data_rate>. The data rate must match the configuration on the target side, else the link between the target and the TRACE32 hardware (Aurora trace channel) cannot be established.

<table>
<thead>
<tr>
<th>&lt;data_rate&gt;:</th>
<th>GEN1</th>
<th>GEN2</th>
<th>GEN3</th>
</tr>
</thead>
<tbody>
<tr>
<td>For PCIe only</td>
<td>GEN1</td>
<td>GEN2</td>
<td>GEN3</td>
</tr>
</tbody>
</table>
Remember that not all TRACE32 PowerTrace tools support all data rates. Contact icrstp-support@lauterbach.com if a lane speed is not supported.

<table>
<thead>
<tr>
<th><strong>AUTO</strong></th>
<th>TRACE32 defines the value.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>625Mbps, ...</strong></td>
<td>Data rate in megabits per second.</td>
</tr>
<tr>
<td><strong>GEN1, ...</strong></td>
<td>Limits the data rate of the PCIe link to 2500Mbps (GEN1), 5000Mbps (GEN2) or 8000Mbps (GEN3).</td>
</tr>
</tbody>
</table>

**Example:**

```
TPIU.PortClock 3125Mbps
TPIU.PortClock 3125M ; M is the short form of Mbps
```

**See also**

- TRACEPORT
- TRACEPORT.state

**TRACEPORT.MsgBltEndian**

*Change bit-order within each byte*  
*For serial trace ports (AURORA) only*

**Format:**

```
TRACEPORT.MsgBltEndian [<option>]
```

**<option>:**

```
AUTO | LittleEndian | BigEndian
```

Allows you to change the bit order of the payload data if the bit order used by the target differs from the default bit order. This might be necessary in case of bus connection errors on the target side between the Aurora logic and the trace source.

<table>
<thead>
<tr>
<th><strong>AUTO</strong></th>
<th>TRACE32 defines the value.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>LittleEndian</strong></td>
<td>Bit order is normal ([31-24],[23-16],[15-8],[7-0]).</td>
</tr>
<tr>
<td><strong>BigEndian</strong></td>
<td>Bit order is reversed ([24-31],[16-23],[8-15],[0-7]).</td>
</tr>
</tbody>
</table>

**See also**

- TRACEPORT
- TRACEPORT.state
### TRACEPORT.MsgBYteEndian

Change byte-order within each word

For serial trace ports (AURORA) only

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRACEPORT.MsgBYteEndian [&lt;option&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>&lt;option&gt;:</strong></td>
<td>AUTO</td>
</tr>
</tbody>
</table>

Allows you to change the byte order of the payload data if the byte order used by the target differs from the default bit order. This might be necessary in case of bus connection errors on the target side between the Aurora logic and the trace source.

<table>
<thead>
<tr>
<th>AUTO</th>
<th>TRACE32 defines the value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LittleEndian</td>
<td>Byte order is normal ([31-24],[23-16],[15-8],[7-0]).</td>
</tr>
<tr>
<td>BigEndian</td>
<td>Byte order is reversed ([23-16],[31-24],[7-0],[15-8]).</td>
</tr>
</tbody>
</table>

See also
- TRACEPORT
- TRACEPORT.state

### TRACEPORT.MsgLOngEndian

Change dword-order within each qword

For serial trace ports (AURORA) only

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRACEPORT.MsgLOngEndian [&lt;option&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>&lt;option&gt;:</strong></td>
<td>AUTO</td>
</tr>
</tbody>
</table>

Allows you to change the byte order of the payload data if the byte order used by the target differs from the default bit order. This might be necessary in case of bus connection errors on the target side between the Aurora logic and the trace source.

<table>
<thead>
<tr>
<th>AUTO</th>
<th>TRACE32 defines the value.</th>
</tr>
</thead>
<tbody>
<tr>
<td>LittleEndian</td>
<td>Double-word order is normal ([63-32],[31-0]).</td>
</tr>
<tr>
<td>BigEndian</td>
<td>Double-word order is reversed ([31-0],[63-32]).</td>
</tr>
</tbody>
</table>

See also
- TRACEPORT
- TRACEPORT.state
TRACEPORT.MsgWOrdEndian  
Change word-order within each dword

For serial trace ports (AURORA) only

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO</td>
<td>TRACE32 defines the value.</td>
</tr>
<tr>
<td>LittleEndian</td>
<td>Word order is normal ([31-16],[15-0]).</td>
</tr>
<tr>
<td>BigEndian</td>
<td>Word order is reversed ([15-0],[31-16]).</td>
</tr>
</tbody>
</table>

See also
- TRACEPORT
- TRACEPORT.state
TRACEPORT.PinReMap

Adapt the lane order of the trace port
For serial trace ports (AURORA) only

Adapts the lane order of the trace port to the lane order of your target. You need the TRACEPORT.PinReMap command only in rare cases where the lane orders of trace port and target actually differ from each other.

**Format:**

```
TRACEPORT.PinReMap <source_lane> <destination_lane> [ <option> ]
```

- `<source_lane>`:
  - `0 | 1 | … | <n>

- `<destination_lane>`:
  - AUTO | 0 | 1 | … | <n>

- `<option>`: RESET

TRACE32 defines the values.
RESET Sets all values to AUTO again.

- `<source_lane>`: Number of the target lane which needs to be remapped.
- `<destination_lane>`: Number of the TRACE32 tool lane which will get the new `<source_lane>`.
  Number `<n>` is TRACE32 tool dependent; e.g. for PowerTrace Serial `<n>` can be 5 or 7 depending on the used tool connector.

**Example:**

```
TRACEPORT.state /PinReMap ; optionally, open the TRACEPORT.state window
TRACEPORT.LaneCount 6Lane ; the number of used lanes for the trace port
TRACEPORT.PinReMap 4. 5. ; map source lane 4. to destination lane 5.
TRACEPORT.PinReMap 5. 4. ; map source lane 5. to destination lane 4.
```

**See also**

- TRACEPORT
- TRACEPORT.state
TRACEPORT.RefCLocK

Set up reference clock for trace port

For serial trace ports (AURORA) only

---

**Format:**

```
TRACEPORT.RefCLocK [<option>]
```

**<option>:**

```
AUTO  |  OFF  |  OSC  |  1/1  |  1/2  |  1/20 |  1/25 |  1/30  |  1/40  |  1/50
```

Defines the reference clock frequency the serial trace hardware outputs to the target. The availability of parameters and the default values depend on the architecture:

- **PowerPC:** not configurable
- **TriCore:** not configurable
- **RH850:** not configurable
- **ARM:** configurable

<table>
<thead>
<tr>
<th><strong>AUTO</strong> (default)</th>
<th>TRACE32 defines the value.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OFF</strong></td>
<td>TRACE32 does not send any reference clock to the target.</td>
</tr>
<tr>
<td><strong>OSC</strong></td>
<td>An asynchronous oscillator will be enabled. Its frequency is architecture dependent.</td>
</tr>
<tr>
<td><strong>1/&lt;x&gt;</strong></td>
<td>A synchronous clock source will be enabled. Its dividers generate a reference clock as a fraction of the bit clock (lane speed), e.g. 100MHz at 5Gbps with divider 1/50. Once a divider is selected, the reference clock will automatically change with the lane speed.</td>
</tr>
</tbody>
</table>

**See also**

- TRACEPORT
- TRACEPORT.state

---

TRACEPORT.RESet

Reset trace port configuration

---

**Format:**

```
TRACEPORT.RESet
```

Resets the trace port configuration to its default values (AUTO).

**See also**

- TRACEPORT
- TRACEPORT.state

---

©1989-2019 Lauterbach GmbH

General Commands Reference Guide T 459
TRACEPORT.StartsKiP  Define number of bytes skipped at the start of frame

Format:  TRACEPORT.StartsKiP [<option>]

<Option>:  AUTO | 0 | 1

Allows to cut off leading bytes of each data packet or data frame. Only a few targets require this due to protocol irregularities.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUTO (default)</td>
<td>TRACE32 defines the value.</td>
</tr>
<tr>
<td>0</td>
<td>No data byte will be cut off.</td>
</tr>
<tr>
<td>1</td>
<td>The first data byte of each data frame will be cut off.</td>
</tr>
</tbody>
</table>

See also
- TRACEPORT
- TRACEPORT.state
TRACEPORT.state

Display trace port configuration window

Format: 

```
TRACEPORT.state [<gui_option>]
```

<gui_option>:

- ADVanced
- PinReMap

Displays the TRACEPORT.state window, where you can configure the communication between the target trace port and the TRACE32 PowerTrace tool.

A For descriptions of the commands in the TRACEPORT.state window, please refer to the TRACEPORT.* commands in this chapter.

Example: For information about the RESet button, see TRACEPORT.RESet.

B Click advanced and pin mapping to display more configuration options in the window.

<table>
<thead>
<tr>
<th>ADVanced</th>
<th>Extends the list of options in the configuration section.</th>
</tr>
</thead>
<tbody>
<tr>
<td>PinReMap</td>
<td>Displays the PinReMap section. For an example, see TRACEPORT.PinReMap.</td>
</tr>
</tbody>
</table>

See also

- TRACEPORT
- TRACEPORT.LaneCount
- TRACEPORT.MsgBitEndian
- TRACEPORT.MsgLOngEndian
- TRACEPORT.PinReMap
- TRACEPORT.RESet
- TRACEPORT.EndsKiP
- TRACEPORT.LaneSpeed
- TRACEPORT.MsgBYteEndian
- TRACEPORT.MsgWordEndian
- TRACEPORT.RefCLocK
- TRACEPORT.StartsKiP
A pulse can be created at the STROBE probe (ECU32) on accessing a certain address. The pulse width is approx. 30 ns (active-low pulse). The pulse appears at the end of the cycle and is synchronous to the CPU cycles signal. This function is often used to trigger a scope while a software loop is running.

**Breakpoint at address 1002H**

<table>
<thead>
<tr>
<th>Address</th>
<th>STROBE.Pin3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000H</td>
<td></td>
</tr>
<tr>
<td>1002H</td>
<td></td>
</tr>
<tr>
<td>1004H</td>
<td></td>
</tr>
</tbody>
</table>

**Pin assignment of the STROBE probe (ECU32)**

<table>
<thead>
<tr>
<th>LED</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>0</td>
<td>13</td>
</tr>
<tr>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>0</td>
<td>7</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>0</td>
<td>14</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>0</td>
<td>8</td>
</tr>
<tr>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

- Pin 1: Line 0, EVENT
- Pin 3: Line 1, TriggerAddress
- Pin 5: Line 2, RUN- (Foreground)
- Pin 7: Line 3, TRIGGER
- Pin 9: Line 4, SIGnal
- Pin 11: Line 5, RUNCYCLE-
- Pin 13: Line 6, PULSe2
- Pin 15: Line 7, PULSe
- Pin 2,4,6,8,10,12,14,16: Ground

On ECC8 emulation controller no **TriggerAddress** command will be supported. Instead the breakpoint CHARLY is routed to the STROBE probe directly.
Breakpoint CHARLY at address 1002H

<table>
<thead>
<tr>
<th>Address</th>
<th>1000H</th>
<th>1002H</th>
<th>1004H</th>
</tr>
</thead>
<tbody>
<tr>
<td>STROBE.Pin9</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Pin assignment of the STROBE probe (ECC8)

<table>
<thead>
<tr>
<th>Pin</th>
<th>15</th>
<th>13</th>
<th>11</th>
<th>9</th>
<th>7</th>
<th>5</th>
<th>3</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>LED</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Pin</td>
<td>16</td>
<td>14</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Pin 1**: Line 0, OUT.C
- **Pin 3**: Line 1, OUT.D
- **Pin 5**: Line 2, RUN- (Foreground)
- **Pin 7**: Line 3, TRIGGER
- **Pin 9**: Line 4, CharlyBreak
- **Pin 11**: Line 5, RUNCYCLE-
- **Pin 13**: Line 6, PULSe2
- **Pin 15**: Line 7, PULSe
- **Pin 2,4,6,8,10,12,14,16**: Ground
<table>
<thead>
<tr>
<th>Format:</th>
<th><strong>TrAddress</strong> [&lt;break&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;break&gt;:</td>
<td>Program</td>
</tr>
<tr>
<td></td>
<td>HII</td>
</tr>
<tr>
<td></td>
<td>Spot</td>
</tr>
<tr>
<td></td>
<td>Read</td>
</tr>
<tr>
<td></td>
<td>Write</td>
</tr>
<tr>
<td></td>
<td>Alpha</td>
</tr>
<tr>
<td></td>
<td>Beta</td>
</tr>
<tr>
<td></td>
<td>Charly</td>
</tr>
</tbody>
</table>

```
ta ; current state is displayed

ta alpha ; alpha breakpoint is selected for generating an address ; signal
```
TRANSlation

Debugger address translation

See also

- TRANSlation.AutoEnable
- TRANSlation.CacheFlush
- TRANSlation.COMMON
- TRANSlation.CreateID
- TRANSlation.Delete
- TRANSlation.List
- TRANSlation.NoProtect
- TRANSlation.ON
- TRANSlation.Protect
- TRANSlation.SCANall
- TRANSlation.SHADOW
- TRANSlation.TableWalk
- TRANSlation.TRANSparent
- MMU
- TRANS.TRANSPARENT( )
- TRANS.LINEAR( )
- TRANS.FORMAT( )
- TRANS.ENABLE( )

▲ ‘TRANS Functions (Debugger Address Translation)’ in ‘General Function Reference’
▲ ‘Release Information’ in ‘Release History’

Overview TRANSlation

NOTE: Formerly, the MMU command group was used for address translation inside the debugger. With the wide-spread adoption of hardware MMUs, it was necessary to rename this command group to TRANSlation to avoid confusion with hardware MMUs.

What is the difference between the command groups...?

<table>
<thead>
<tr>
<th>TRANSlation</th>
<th>MMU</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configures and controls the TRACE32 internal debugger address translation. This feature is used to mimic the translations within the real hardware MMU so that the debugger can access code and data of any OS process at any time.</td>
<td>Lets you access and view the real hardware MMU.</td>
</tr>
</tbody>
</table>
The **TRANSlation** commands are used for the following purposes:

- To debug an OS that runs several processes at the same logical addresses (e.g. Linux, PikeOS, etc.).
- To allow a transparent display of hardware translation tables and OS-based translation tables.
- To provide the user with unrestricted access to the target memory using either logical or physical addresses.

## MMU Tables

To apply the MMU commands properly, it is important to differentiate between the following MMU table types:

1. **The hardware MMU table**

   The hardware MMU usually consists of registers and/or dedicated memory areas and is held in the CPU. It holds the translation tables that are used by the CPU to translate the logical addresses used by the CPU into the physical addresses required for memory accesses.

   In OSs like Linux, PikeOS etc. each process has its own address space. Usually all processes start at the logical address 0x0. The result is that, while a process is running, the process has only access to its own address space and to the address space of the kernel.

   The hardware MMU is programmed by the scheduler for this view. If, for example, process 2 is running, the hardware MMU provides only translation tables for process 2 and the kernel.

   ![MMU Tables Diagram]

   - If the OS uses demand paging, the hardware MMU table is extended at each page fault.
   - At each process switch the hardware MMU is reprogrammed so that the logical address space of the current process can be translated to the physical address area.

2. **The software/OS MMU table**

   If an OS like Linux, PikeOS etc. is used, the OS maintains the translation tables for all processes, because the OS is responsible for the reprogramming of the hardware MMU on a process switch.

   The hardware MMU is usually only a subset of the OS MMU tables.
3. The debugger MMU table

If an OS that runs several processes at the same logical addresses (e.g. Linux) is used, the hardware MMU in the CPU only holds translation tables that allow the debugger memory accesses to the code/data of the kernel and the currently running process.

The debugger can access code/data from a not currently running process only with the help of the OS MMU tables. Based on the information held in the translation tables of the OS MMU, the debugger can translate any logical address to a physical address and that way perform a memory access without changing the hardware MMU. If demand paging is used, the OS MMU table contains the translation from logical to physical address only if the page was loaded before.

Reading the OS MMU tables on every memory access in quite time consuming. Therefore the debugger can scan the OS MMU tables once and re-use the scan for all following accesses.

The OS MMU table is scanned into the so-called debugger MMU. The debugger MMU provides also the flexibility to add user-defined entries.

Please be aware that as soon as the debugger MMU is active, all memory accesses performed by the debugger use only the information of the debugger MMU.

Please be aware that the OS MMU tables have to be scanned again if the OS has added or removed entries from these tables while running.

TRANSlation.AutoEnable  Auto-enable debugger MMU translation

Format:  TRANSlation.AutoEnable

Auto-enable the debugger address translation if the CPU's hardware MMU is enabled. When the hardware MMU is on, the debugger also performs translations. When the hardware MMU is off, the debugger performs no translation and treats all logical addresses as physical addresses. The state of the hardware MMU is read from the CPU-specific MMU-enable bit in a system control register. This command is only available for certain CPUs.

See also
■ TRANSlation
■ TRANSlation.List
■ TRANSlation.OFF
■ TRANSlation.ON
If the operating system adds or removes entries from its page table while running those changes are not performed within the debugger MMU. Trying to access those newly created logical addresses with the debugger may cause an error. If **TRANSlation.AutoSCAN** is set to ON the translation tables hold by the operating system are automatically scanned into the debugger MMU, if the debugger fails to access a logical address.

**TRANSlation.AutoSCAN** scans only pages that are already present. Depending on the JTAG speed of the processor and on the number of processes in the system scanning the translation tables can take some time. In this cases autoscanning may be more disturbing than helping.

See also

- **TRANSlation**
- **TRANSlation.List**
TRANSlation.CacheFlush
Flush TRACE32 address translation cache

| Format: | TRANSlation.CacheFlush [ALL] |

Successful MMU address translations are cached internally by TRACE32. This speeds up recurring accesses to a logical address in debug mode - mostly when the OS Awareness is enabled. Caching is most beneficial for translations done via an MMU table walk as this generates many memory accesses while parsing the OS page table.

TRANSlation.CacheFlush flushes the TRACE32 internal address translation caches, so a new readout of the OS page table is enforced for the next memory access. This can be useful when modifying page table content or debugging MMU table walks.

ALL
Additionally invalidates the complete register set cached by the debugger, including all cached MMU registers. Upon the next MMU page table walk, the registers will be re-read from the target.

See also
- TRANSlation
- TRANSlation.List
- TRANSlation.TableWalk

TRANSlation.CLEANUP
Clean up MMU table

| Format: | TRANSlation.CLEANUP MMU.CLEANUP (deprecated) |

Removes multiple translations for one physical address, directly joining translations and double translations.

See also
- TRANSlation
- TRANSlation.List
Defines one or more mappings of logical address ranges that are shared by the kernel and the tasks.

When the address of a memory access falls into a common address range, TRACE32 uses the kernel address translation (and not the task page table of the current process). Internally, TRACE32 always uses space ID 0x0000 to find the translation of a common address.

This allows to apply the kernel address translation to modules or libraries that are called by a user process in the context of the currently running task.

Format:

```
TRANSlation.COMMON <logical_range> [<logical_range>]
```

MMU.COMMON (deprecated)

A Space ID of the kernel = 0x0000

B Space IDs of the tasks ≠ 0x0000

TRACE32 assigns space IDs if SYStem.Option MMUSPACES is set to ON

C Common logical address ranges are flagged as “COMMON” in the TRANSlation.List window, which displays the mappings between logical and physical address ranges.

You can specify up to 10 common ranges in one line. For an example with two common address ranges, see below.

Overlapping common address ranges are merged automatically.

**NOTE:**

Executing the TRANSlation.COMMON command again discards all previously existing common address ranges.

Use TRANSlation.COMMON.ADD to add further common address ranges without discarding existing common address ranges.
Example:

;Enable the space IDs to display them in the TRANSLation.List window
SYStem.Option MMUSPACES ON

TRANSLation.List ;Open the Translation.List window

;Create some translation entries for a particular debug session
TRANSLation.Create 0x0:0x00000000++0xffff 0x10000
TRANSLation.Create 0x123:0x00000000++0xffff 0x101000
TRANSLation.Create 0x042:0x00000000++0xffff 0x102000

;Define two common logical address ranges:
;
;Alternatively, you can define the common ranges as follows:
;TRANSLation.COMMON 0x80004020--0x800041ff 0x80004700--0x800049ff

See also

- TRANSLation.COMMON.ADD
- TRANSLation
- MMU.FORMAT

▲ 'Release Information' in 'Release History'
**TRANSlation.COMMON.ADD**  Add another common address range

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRANSlation.COMMON.ADD &lt;logical_range&gt;</th>
</tr>
</thead>
</table>

Adds another mapping for a common logical address range that is shared by the kernel and the tasks.

**NOTE:** Use **TRANSlation.COMMON.ADD** to add further common address ranges without discarding existing common address ranges. Executing **TRANSlation.COMMON** again discards all previously existing common address ranges.

**Example:**

; Define the first common logical address range
TRANSlation.COMMON 0x80000200--0x80007ff

; Add two additional ranges
TRANSlation.COMMON.ADD 0x80004020--0x800041ff
TRANSlation.COMMON.ADD 0x80004700--0x800049ff

**See also**
- **TRANSlation.COMMON**
- **TRANSlation.COMMON.CLEAR**

**TRANSlation.COMMON.CLEAR**  Clear all common logical address ranges

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRANSlation.COMMON.CLEAR</th>
</tr>
</thead>
</table>

Clears only those logical address ranges that are flagged as “COMMON” in the **TRANSlation.List** window.

**See also**
- **TRANSlation.COMMON**
- **TRANSlation.COMMON.ADD**

©1989-2019 Lauterbach GmbH
The defined translation can either be function code specific or generic for all function codes (except I/O). The physical address or range is not allowed on probes with fixed MMU translation (e.g. 80186,Z180).

**More**

The More option suppresses the generation of the MMU tables. This speeds up the entry of large translation tables with PRACTICE scripts (*.cmm). The last translation command should not have a More option, otherwise the translations are not accessible.

**Logical**

The Logical and Physical options create translations that work only in one direction. This allows to create multiple logical addresses that map to the same physical address and still having a well-defined logical address for the reverse translation.

**Example:** Translation for 68030 TRANSLation

```
TRANSlation.Create 0x1000--0x1fff a:0x20000--0x20fff /More
TRANSlation.Create sd:0x2000--0x2fff asd:0x0--0x0fff /More
TRANSlation.Create ud:0x2000--0x2fff aud:0x1000--0x1fff /More
TRANSlation.Create sp:0x2000--0x2fff asp:0x2000--0x2fff /More
TRANSlation.Create up:0x2000--0x2fff aup:0x3000--0x3fff
```

**See also**

- TRANSlation
- TRANSlation.TlbAutoScan
- MMU.SCAN
- MMU.CREATE
- MMU.FORMAT
- SYStem.Option MMUSPACES
- 'ARM Specific Implementations' in 'ARM Debugger'
- 'ARM Specific Implementations' in 'ARMv8-A/-R Debugger'
**TRANSlation.CreateID**

Add entry to MMU space ID table

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRANSlation.CreateID &lt;space_id&gt;:0x0 &lt;base_address&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;space_id&gt;</td>
<td>Space ID to be added.</td>
</tr>
<tr>
<td>&lt;base_address&gt;</td>
<td>Physical base address of task page table associated with &lt;space_id&gt;.</td>
</tr>
</tbody>
</table>

**See also**

- TRANSlation
- TRANSlation.DeleteID
- TRANSlation.List
- TRANSlation.ListID

**TRANSlation.CreateTab**

Create multiple translations

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRANSlation.CreateTab &lt;logical_range&gt; &lt;increment&gt; &lt;logical_range&gt; [physical_range] [option]</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMU.CreateTab (deprecated)</td>
<td></td>
</tr>
<tr>
<td>&lt;option&gt;:</td>
<td>More Logical Physical</td>
</tr>
</tbody>
</table>

Same as TRANSlation.Create, but creates multiple translations with one command. The first range defines the logical range for the created translations. The increment parameter is the offset added to the logical address to generate the next address. The other parameters are interpreted like the TRANSlation.Create command.

**Example:**

```
 ; Translation for COMMON area from 0x08000--0x0ffff
 TRANSlation.CT 0x0--0x0fffff 0x10000 0x08000--0x0ffff 0x08000--0x0ffff

 ; Translation for 16 BANKS
 TRANSlation.CreateTab 0x0--0x0ffff 0x10000 0x0--0x7fff
```

**See also**

- TRANSlation
- TRANSlation.List
**TRANSlation.Delete**  
Delete translation

The defined translation is removed from the list; see **TRANSlation.List**. Use the command **TRANSlation.RESet** to clear the whole list.

**Example:**

```
TRANSlation.Delete 0x1000--0x1fff
```

See also
- **TRANSlation**
- **TRANSlation.List**

### TRANSlation.DeleteID

Remove entry from MMU space ID table

**Format:**

```
TRANSlation.DeleteID <space_id>::0x0
```

**<space_id>**  
Space ID to be removed.

See also
- **TRANSlation**
- **TRANSlation.CreateID**
- **TRANSlation.List**
- **TRANSlation.ListID**
- **TRANSlation.ScanID**
Displays the list of static address translations created with the commands `TRANSLation.Create` or `MMU.SCAN`

The static MMU translation table of TRACE32 contains relations between logical address spaces and physical address spaces. This table is consulted when the debugger address translation is enabled with `TRANSLation.ON` and a logical address must be converted into a physical address. In some cases this table is also used for reverse translating a physical address into its logical counterpart.

### Format:

```
TRANSLation.List [Logical | Physical]
```

**MMU.List** (deprecated)

---

**A** Space ID of the kernel = 0x0000

**B** Space IDs of the tasks ≠ 0x0000

TRACE32 assigns space IDs if `SYStem.Option MMUSPACES` is set to `ON`

**C** The default logical-to-physical address translation, which is used for fast memory accesses into the kernel address range.

The default address translation is specified with the command `MMU.FORMAT`.

**D** Common address ranges are created with the commands `TRANSLation.COMMON` or `TRANSLation.COMMON.ADD`

<table>
<thead>
<tr>
<th>Logical</th>
<th>Sorts logical addresses in ascending order.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physical</td>
<td>Sorts physical addresses in ascending order.</td>
</tr>
</tbody>
</table>

---

**See also**

- `TRANSLation.ListID`
- `TRANSLation.CacheFlush`
- `TRANSLation.CreateID`
- `TRANSLation.NoProtect`
- `TRANSLation.Protect`
- `TRANSLation.SHADOW`
- `TRANSLation.TRANSparent`
- `MMU FORMAT()`
- `TRANSLation`
- `TRANSLation.CLEANUP`
- `TRANSLation.CreateTab`
- `TRANSLation.OFF`
- `TRANSLation.RESet`
- `TRANSLation.state`
- `MMU FORMAT`
- `TRANSLation.AutoEnable`
- `TRANSLation.COMMON`
- `TRANSLation.Delete`
- `TRANSLation.ON`
- `TRANSLation.SCAnAll`
- `TRANSLation.TableWalk`
- `MMU SCAN`
- `TRANSLation.AutoSCAN`
- `TRANSLation.Create`
- `TRANSLation.DeleteID`
- `TRANSLation.PAGER`
- `TRANSLation.ScanID`
- `TRANSLation.TabAutoScan`
- `MMU DEFAULTPT()`

▲ 'Release Information' in 'Release History'

©1989-2019 Lauterbach GmbH
TRANSlation.ListID

List MMU space ID table

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRANSlation.ListID</th>
</tr>
</thead>
</table>

The table is used to translate MMU root pointer contents into a memory space ID. Memory space IDs may also be obtained from the OS Awareness without the use of this table.

See also
- TRANSlation.List
- TRANSlation
- TRANSlation.CreateID
- TRANSlation.DeleteID

TRANSlation.NoProtect

Unprotect memory

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRANSlation.NoProtect &lt;logical_range&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MMU.NoProtect (deprecated)</td>
</tr>
</tbody>
</table>

Removes the protection for the specified logical address range. As a result, the debugger can access this range. See TRANSlation.Protect.ADD.

Example:

```plaintext
TRANSlation.Protect.ADD 0x100000--0x1fffff ;no access here
TRANSlation.Protect.ADD 0x280000--0x0fff000 ;no access here
TRANSlation.Protect.ADD 0x1000000--0x0ffffffff ;no access here

TRANSlation.ON
TRANSlation.List ;display overview of protected memory ranges

;your code

;remove this logical address range from the list of protected memory ranges
;ranges
TRANSlation.NoProtect 0x1000000--0x0ffffffff
```

See also
- TRANSlation
- TRANSlation.List
- TRANSlation.Protect
- TRANSlation.Protect.ADD
**TRANSlation.OFF**  
Deactivate debugger address translation  

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRANSlation.OFF MMU.OFF (deprecated)</th>
</tr>
</thead>
</table>

Deactivates the TRACE32 internal debugger address translation.

**Logical addresses** used by the debugger are directly sent to the target CPU without translation. Also, the protection of address ranges which have been declared as protected is disabled.

**See also**
- TRANSlation.ON
- TRANSlation
- TRANSlation.AutoEnable
- TRANSlation.List

---

**TRANSlation.ON**  
Activate debugger address translation  

<table>
<thead>
<tr>
<th>Format:</th>
<th>TRANSlation.ON MMU.ON (deprecated)</th>
</tr>
</thead>
</table>

Activates the TRACE32 internal debugger address translation. For Intel® architecture debuggers, the address translation is enabled by default. For all other architectures, the default is TRANSlation.OFF.

With **TRANSlation.ON**, the following features are enabled:

- **Logical addresses** are translated to **physical addresses**. The address translation is based on the following translation tables:
  - The static address translation list (see TRANSlation.List)
  - Intel® architectures only: the segment translation for boot mode, real mode and protected mode (see MMU.view, MMU.DUMP.GDT, MMU.DUMP.LDT and SYStem.Option.MEMoryMODEL)
  - MIPS architectures only: the EVA or fixed mapping KSEG0/1 translations are done.
  - OS page tables if the TRACE32 table walk is enabled (see TRANSlation.TableWalk and MMU.FORMAT).
- For some architectures, TLBs can be evaluated. This feature is also enabled with TRANSlation.TableWalk and MMU.FORMAT.
- Address ranges declared as protected are no longer accessible to the debugger (see TRANSlation.Protect).
For an overview of the state of the debugger address translation, see `TRANSlation.state`.

**See also**

- `TRANSlation.OFF`
- `TRANS.EABEL()`
- `TRANSlation`  
- `TRANSlation.AutoEnable`  
- `TRANSlation.List`

---

**TRANSlation.PAGER**

Allow paged breakpoints for Linux

Format:

```
TRANSlation.PAGER ON <address> | OFF
MMU.Protect (deprecated)
```

The TRACE32 software and a suitable Linux patch enable a software breakpoint to be set for program code that has not yet been loaded.

Details on the Linux patch can be found in the directory `~/demo/arm/kernel/linux/etc/t32pager`

---

**See also**

- `TRANSlation`
- `TRANSlation.List`
Using the `TRANSlation.Protect` command group, you can protect the entire logical address range or individual logical address ranges from debugger access. This can be useful if an access would otherwise cause a fatal hardware error or cause the debugger to go down.

**What is the difference between...?**

<table>
<thead>
<tr>
<th><code>TRANSlation.Protect.ON</code></th>
<th><code>TRANSlation.Protect.ADD</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>• Protects the <em>entire</em> logical address range from debugger access.</td>
<td>• Protects <em>individual</em> logical address ranges from debugger access.</td>
</tr>
<tr>
<td>• However, you can allow debugger access to individual logical address ranges by specifying them with <code>TRANSlation.Create &lt;logical_range&gt;</code>.</td>
<td>• <code>TRANSlation.Protect</code> must not be set to ON.</td>
</tr>
</tbody>
</table>

**See also**

- `TRANSlation.Protect.ADD`
- `TRANSlation.Protect.OFF`
- `TRANSlation.Protect.ON`
- `TRANSlation.List`
- `TRANSlation.NoProtect`
- `TRANSlation.Protect.ADD`
- `TRANSlation.Protect.OFF`
- `TRANSlation.Protect.ON`

**`TRANSlation.Protect.ADD`**

Add range to protected memory ranges

**[Example]**

Format: `TRANSlation.Protect.ADD <logical_range>`

Protects the specified logical address range from debugger access.

**NOTE:**

Use `MAP.DenyAccess` to protect physical address ranges from debugger access.

Use `TRANSlation.Protect.ADD` to protect logical address ranges from debugger access.
Example:

```
;[A] allow debugger access to the logical address ranges 0x0--0x103F
;    and 0x1070--0xFFFFFFFF, i.e. almost the entire logical range, ...

;[B] ...but protect this logical address range from debugger access
TRANSlation.Protect.ADD 0x1040--0x106F

TRANSlation.ON

;display overview of protected memory range(s)
TRANSlation.List

;let’s open this window for demo purposes to visualize the result
Data.dump 0x1020 /NoAscii
```

See also
- `TRANSlation.Protect`
- `TRANSlation.NoProtect`
- `MAP.DenyAccess`

`TRANSlation.Protect.OFF`  
Switch protection of target memory off

Format:  
`TRANSlation.Protect.OFF`

Re-allows debugger access to the entire logical address range. See `TRANSlation.Protect.ON`.

See also
- `TRANSlation.Protect`
**TRANSlation.Protect.ON**

Protects the entire logical address range from debugger access, provided the address translation is enabled with **TRANSlation.ON**.

**Example:**

```
TRANSlation.ON

;protect entire logical address range from debugger access (see red [A])
TRANSlation.Protect.ON

;but allow debugger access to this logical address range (see green [B])
TRANSlation.Create 0x1040--0x106F

;display overview of static translations
TRANSlation.List

;let’s open this window for demo purposes to visualize the result
Data.dump 0x1020 /NoAscii
```

See also
- TRANSlation.Protect
**TRANSlation.RESet**  
**Reset MMU configuration**

| Format: | TRANSlation.RESet  
|         | MMU.RESet (deprecated) |

The translation table is cleared and all setups are set to the defaults.

See also
- TRANSlation  
- TRANSlation.List

**TRANSlation.SCANall**  
**Scan MMU tables**

| Format: | TRANSlation.SCANall  
|         | MMU.SCAN ALL (as an alias)  
|         | MMU.SCANALL (deprecated) |

Scans all page translation tables known to the debugger into the static translation list. That is, this command is a repeated call of the MMU.SCAN command for all known page tables of an architecture known to the debugger.

See also
- TRANSlation  
- TRANSlation.List

**TRANSlation.ScanID**  
**Scan MMU address space tables from kernel**

| Format: | TRANSlation.ScanID |

Scans the translation information from the kernel into the MMU space ID table. The operation is target and kernel dependent.

See also
- TRANSlation  
- TRANSlation.CreateID  
- TRANSlation.DeleteID  
- TRANSlation.List
TRANSlation.SHADOW

Enable shadow access to target memory

Use VM: for data access, if the address translation on the target failed.

The debugger first tries to resolve a logical address with the standard address translation, and then accesses the target to read the requested data. If the translation fails (due to missing table entries, or due to an access error), and if TRANSlation.SHADOW is ON, the debugger uses the data within VM: at the requested address.

The debugger provides a “virtual memory” (access class VM:) that is not accessible from the CPU, but only by the debugger (stored within the host). The idea is to have a (partial) copy of the target memory in the host for unlimited access.

VM: usually is a “virtual physical memory”. The debugger does an address translation (logical -> physical), then accesses VM: with the physical address. I.e. VM: maps a physical memory.

If TRANSlation.SHADOW and SYStem.Option MMUSPACES is ON, VM: is used as several logical addressed memory areas, separated by the space ID. No address translation is done, instead the debugger directly accesses the memory in VM: with space ID:address. I.e. VM: maps several logical memory areas. In complex OS target systems (e.g. Linux), you may load the code of several processes into VM: to have access to the code, even if the target does currently not allow memory access.

See also

- TRANSlation
- TRANSlation.List
Opens the **TRANSlation.state** window.

A The header displays an overview of all settings affecting the debugger address translation:

- Address translation: ON, OFF = **TRANSlation.ON** or **TRANSlation.OFF**
- MMU protection: ON, OFF = **TRANSlation.Protect.ON** or **TRANSlation.Protect.OFF**
- Table walk: ON, OFF = **TRANSlation.TableWalk** [ON | OFF]
- MMU spaces: ON, OFF = **SYStem.Option.MMUSPACES** [ON | OFF]
- Zone spaces: ON, OFF = **SYStem.Option.ZoneSPACES** [ON | OFF]
- Machine spaces: ON, OFF = **SYStem.Option.MACHINESPACES** [ON | OFF]
- Architecture-specific settings (here LPAE)

B The columns below the header list the settings configured with the **MMU.FORMAT** command.

### Description of Columns in the TRANSlation.state Window

| Zone | For information about zones, refer to the glossary. |
| MMU format | The MMU formats for each zone. |
| Default page table | The start addresses of the default page tables for all active zones. |

**See also**

- **TRANSlation**
- **TRANSlation.List**
**Format:**

```
TRANSlation.TableWalk [ON | OFF]
```

Configures the debugger to perform an MMU page table walk (short: table walk). If enabled, the debugger will try the following steps upon a logical-to-physical address translation request:

1. Look up the logical address in the debugger's static address translation table (see `TRANSlation.List` and `TRANSlation.Create` for details about the static address translation table).
2. If the address lookup in the static address translation table fails, walk through the software/OS MMU tables to find a valid logical-to-physical translation.
3. For Intel® architecture debuggers, the boot mode, real mode, or protected mode segment translation is done before the page table walk is performed.
4. For MIPS architectures only: the EVA or fixed mapping KSEG0/1 translations are done before the page table walk is performed.

Valid address translations found are cached by TRACE32. When debug mode is left, i.e. at a Go or Step, the cached translations are flushed because page table contents may change when the target continues execution.

<table>
<thead>
<tr>
<th>ON</th>
<th>Configure TRACE32 to use the automatic MMU table walk. Only physical addresses are sent to the target.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><strong>NOTE for expert users:</strong></td>
</tr>
<tr>
<td></td>
<td>For some architectures, although a valid translation is available, TRACE32 sends logical addresses in certain situations in order to ensure cache coherency.</td>
</tr>
<tr>
<td></td>
<td>This behavior can be controlled with the architecture-specific command <code>SYStem.Option.MMUPhysLogMemaccess</code>.</td>
</tr>
<tr>
<td>OFF</td>
<td>Configure TRACE32 to not use the automatic MMU table walk.</td>
</tr>
</tbody>
</table>

**NOTE:** Page tables are dynamic structures and are frequently modified by the OS.

The MMU page table walk of the debugger dynamically parses the page tables on demand for every debugger address translation. The table walk ensures that the debugger address translations correspond to the current OS address translations.
If no valid translation could be found for a logical address in any available translation table, then the error handling depends on whether \texttt{TRANSlation.TableWalk} is set to \texttt{OFF} or \texttt{ON}:

| OFF | No error will be produced by TRACE32. The logical address will be sent to the target CPU without translation. |
| ON  | A “MMU translation failed” error will be produced by TRACE32. Scripts will stop upon a failing translation. This mimics the behavior of the target MMU, where a failing translation causes an exception. |

See also

- \texttt{TRANSlation}
- \texttt{TRANSlation.CacheFlush}
- \texttt{TRANSlation.List}
- \texttt{MMU FORMAT}
- \texttt{TRANS.TABLEWALK()}  
- ‘ARM Specific Implementations’ in ‘ARM Debugger’
- ‘ARM Specific Implementations’ in ‘ARMv8-A/-R Debugger’
- ‘Release Information’ in ‘Release History’

\textbf{TRANSlation.TlbAutoScan}  
\textbf{Allow automatic TLB scans during table walk}

Format: \texttt{TRANSlation.TlbAutoScan [ON | OFF] [<logical_range>] [<logical_range>]}  

Enable automatic scan of the TLBs for missing kernel address translations during MMU table walks. Ignore TLB entries with logical addresses outside the specified \texttt{<logical_range>}.  

\textbf{NOTE:}  
This command is not available for all architectures  

Some OS specify logical base addresses for kernel or task page tables. The table walk algorithm must translate them to physical addresses before the page table can be parsed. If there is no suitable user-defined default translation (\texttt{MMU FORMAT}) or debugger MMU table (\texttt{TRANSlation.List}) entry, \texttt{TRANSlation.TlbAutoScan} will search the target MMU TLBs for a suitable translation that has been set up and used by the OS itself. If a suitable translation is found, it is copied into the debugger MMU table. This automatism can prevent debugger memory access failures caused by incomplete MMU setup scripts.  

\textbf{NOTE:}  
Only TLB entries in the kernel address range must be auto-extracted from TLBs. If you specify the typical kernel address range(s) for your target’s OS in \texttt{<logical_range>}, \texttt{TRANSlation.TlbAutoScan} will ignore dynamic TLB entries used for user processes.
Place the **TRANSlation.TlbAutoScan** command into the MMU section of your PRACTICE script preparing the debugger for OS Awareness as in this example:

```plaintext
;=================================================================================
; example MMU setup section for Linux awareness
; - "TRANSlation.TlbAutoScan ON" replaces the explicit
;   default translation in MMU FORMAT and fixed kernel
;   address translations in TRANSlation.Create.
;=================================================================================

PRINT "Initializing debugger MMU..."
MMU.FORMAT LINUX swapper_pg_dir
TRANSlation.COMMON 0xC0000000--0xFFFFFFFF

; this translation will be auto-extracted by TlbAutoScan from the TLB
; TRANSlation.Create 0xC0000000--0xFFFFFFFF 0x0

; enable TlbAutoScan - TLB entries in 0x80000000--0xFFFFFFFF are kernel
; addresses here and ok to be auto-extracted
TRANSlation.TlbAutoScan ON 0xC0000000--0xFFFFFFFF

TRANSlation.TableWalk ON
TRANSlation.ON
```

**See also**

- TRANSlation
- TRANSlation.Create
- TRANSlation.List
- MMU.DUMP
- MMU FORMAT
- MMU SCAN
A debugger access to a logical address within `<logical_range>` will not be translated to a physical address, even if a page table translation for it is defined. Instead, this access will use the logical address.

**Example**

In a banked memory system, we want the debugger to see the current memory bank (selected by the CPU's BNK register) for memory accesses within `<logical_range>`. The following example shows a PRACTICE script for such a setup for a CPU with 16 bit logical addresses:

```plaintext
sYmbol.RESet
TRANSlation.Reset
; define fixed translation window into bank 0
TRANSlation.Create 0x1000000--0x100ffff A:0x00000--0x0ffff
; define fixed translation window into bank 1
TRANSlation.Create 0x1010000--0x101ffff A:0x10000--0x1ffff
; define transparent address window (no translation in this range)
TRANSlation.TRANSparent 0x0--0xffff
TRANSlation.ON
; load code into current bank, somewhere in 0x0--0xffff
Data.LOAD.UBROF example.dbg
; shift symbols to logical addresses at 1000000
sYmbol.RELOCate C:0x1000000
```

Any access within 0x0..0xffff is defined as transparent and will thus not be translated to a physical address by the debugger. Instead, such an access will be carried out with the logical address, so the CPU's “current bank” register will decide which data is seen. That is, examining a variable pointing to a certain logical address somewhere within 0x0..0xffff with bank 1 being active, will show the data stored in bank 1.

We want to make sure that symbols belonging to code or data loaded into a certain bank are always tied to the correct bank. Addresses in 0x0..0xffff may show any bank, depending on the BNK register. So we first define fixed translation windows of 0x1000000..0x100ffff to bank 0 and 0x1010000..0x101ffff to bank 1. Note that those address windows exist only for the debugger.

Now we load code (assuming bank 0 being selected by register BNK) into memory. Finally, we shift the symbols belonging to the code into the address window belonging to bank 0, i.e. we add an offset of 0x1000000 after loading. Now we have a clear assignment between the symbols and the data in bank 0, while debugger accesses to logical addresses in 0x0..0xffff still see the data the CPU sees currently.

**See also**

- TRANSlation
- TRANSlation.List
Overview TrBus

The *TrBus* command group allows:

- to generate a trigger pulse that can be used to trigger an external device e.g. a Logic Analyzer.
- to connect an incoming trigger signal to TRACE32-ICD or TRACE32-FIRE.

In both cases the TRIG connector is used. The TRIG connector has the following characteristics on the different TRACE32 tools:

<table>
<thead>
<tr>
<th>TRACE32 tool</th>
<th>Output voltage</th>
<th>Input voltage</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>PowerDebug PRO</td>
<td>4.4V</td>
<td>3.3V</td>
<td>Input: 5V tolerant, 10K pull-up/down*</td>
</tr>
<tr>
<td>uTrace for Cortex-M</td>
<td>3.3V</td>
<td>3.3V</td>
<td>Input: 5V tolerant, 10K pull-up/down*</td>
</tr>
<tr>
<td>PowerDebug Module USB 3.0</td>
<td>4.4V</td>
<td>3.3V</td>
<td>Input: 5V tolerant, 10K pull-up/down*</td>
</tr>
<tr>
<td>PowerDebug II Ethernet</td>
<td>5.0V</td>
<td>3.3V</td>
<td>Input: 5V tolerant, 10K pull-up/down*</td>
</tr>
<tr>
<td>PowerDebug Module Ethernet / PowerTrace Ethernet</td>
<td>3.3V</td>
<td>3.3V</td>
<td>Input: 5V tolerant, 10K pull-up/down*</td>
</tr>
<tr>
<td>Power Debug Module USB 2.0</td>
<td>3.3V</td>
<td>3.3V</td>
<td>Input: 5V tolerant, 10K pull-up/down*</td>
</tr>
<tr>
<td>PODBUS Ethernet Controller</td>
<td>3.3V</td>
<td>3.3V</td>
<td>Input: 5V tolerant, 10K pull-up/down*</td>
</tr>
<tr>
<td>Power Debug Module USB 1.x</td>
<td>3.3V</td>
<td>3.3V</td>
<td>Input: 5V tolerant, 10K pull-up/down*</td>
</tr>
</tbody>
</table>
Pull-up/down selected automatically depending on low-active or high-active settings.

An **external trigger pulse** of at least 100ns can be generated when:

- The program execution is stopped.
- A trigger is generated for the trace (not available on all CPUs, depends on the implementation of the trace trigger feature).
- The sampling to the trace buffer is stopped ((not available on all CPUs, depends on the implementation of the trace trigger feature).
- A breakpoint with the Action BusTrigger is used (not available on all CPUs).

An **incoming trigger signal** can be used:

- To stop the program execution.
- To generate a trigger for the trace (not available on all CPUs, depends on the implementation of the trace trigger feature).

The sources for the external trigger pulse and the targets for the incoming trigger signal are connected to the trigger bus.
The following picture shows the Trigger Bus on the PowerTrace as an example.

**Example:** Generate a trigger for the trace at a falling edge of the incoming trigger signal.

```plaintext
TrBus.Arm            ; Switch the trigger bus ON
TrBus.Connect In     ; Configure the TRIGGER connector as input
TrBus.Mode.Falling   ; define that the trigger target should react
                     ; on the falling edge of the incoming trigger
                     ; signal
TrBus.Set ATrigger ON ; generate a trigger for the trace (trigger
                     ; target) on the falling edge of the external
                     ; trigger signal
                     ; a trigger for the trace can stop the
                     ; sampling to the trace directly or it can be
                     ; delayed by the command Analyzer.TDelay
TrBus.Set Break OFF   ; switch all other sources and targets to OFF
TrBus.Out Break OFF
TrBus.Out ABreak OFF
TrBus.Out ATrigger OFF
```

©1989-2019 Lauterbach GmbH
Interaction Between Independent PODBUS Devices

If several independent PODBUS devices are plugged together, they share the same trigger bus. Example configurations are:

- A POWER DEBUG II and a POWER TRACE II
- A POWER DEBUG INTERFACE / USB and a POWERPROBE
- A POWERTRACE / ETHERNET and a POWERINTERGRATOR
- Several POWER DEBUG INTERFACEs that form a multi-processor debugging environment.

The common trigger bus allows a synchronization between the PODBUS devices.
Example: A soon as the POWERPROBE is stopped by a trigger, the program execution should be stopped via the connected POWER DEBUG INTERFACE:

```
PowerProbe
...
PP:Analyzer.TOut BUSA ON   ; definition of the trigger condition
                       ; generate a trigger for the trigger bus
                       ; when the defined trigger event occurs

Debugger
TrBus.Arm              ; switch the trigger bus ON
TrBus.Connect Out      ; Configure the TRIGGER connector as output
TrBus.Set Break ON     ; allow any trigger from the trigger bus to
                       ; stop the program execution
```

The trigger bus also allows to stop the processors in a multi-processor configuration synchronously. Precondition is that the JTAG/OCDS/BDM … connector provides:

- A signal which indicates that the program execution was stopped (stop indication).
- A signal that allows to stop the program execution immediately (stop request).

After the configuration for the synchronous start and stop by the SYNch command is done, you can configure the stop synchronization per hardware by the TrBus commands.

Multicore chip sets provide normally internal (chip specific) trigger connections.
Example: Configure the stop synchronization per hardware for the TRICORE OCDS connector:

```
TrBus.Arm ; Switch the trigger bus ON
TrBus.Connect Out ; Configure the TRIGGER connector as output
TrBus.Set Break ON ; Connect Break to stop request
TrBus.Out Break ON ; Connect Break to stop indication
```

**TrBus.Arm**

*Arm the trigger bus*

**Debugger / FIRE only**

Format: `TrBus.Arm`

Arms the trigger bus.

See also

- `TrBus`
- `TrBus.state`
The TRIGGER connector should work as:
- Input for an incoming trigger signal.
- Output for the generation of an external trigger signal.

See also:
- TrBus
- TrBus.state

**TrBus.Connect**

Configure TRIGGER as input or output

Debugger / FIRE only

Format:  

```
TrBus.Connect In | Out
```

**TrBus.Mode**

Define polarity/edge for the trigger signal

Debugger / FIRE only

Format:  

```
TrBus.Mode Low | High | Falling | Rising
```

If `TrBus.Connect Out` is set a Low or High pulse is generated on TRIGGER (at least 100 ns) as soon as the defined source becomes active.

If `TrBus.Connect In` is set, the defined target can react on a Low/High pulse or Falling/Rising edge of the incoming trigger signal.

See also:
- TrBus
- TrBus.state

**TrBus.OFF**

Switch trigger bus off

Debugger / FIRE only

Format:  

```
TrBus.OFF
```

Switches the trigger bus off.

See also:
- TrBus
- TrBus.state
**TrBus.Out**

Define source for the external trigger pulse

| Format: | TrBus.Out Break | ABreak | ATrigger [ON | OFF] |

Defines the source for the external trigger pulse.

- **Break**  
  Generate an external trigger pulse when the program execution is stopped.

- **ABreak**  
  Generate an external trigger pulse when the sampling to the trace buffer is stopped.

- **ATrigger**  
  Generate an external trigger pulse when a trigger is generated for the trace. A trigger for the trace can be used to stop the sampling to the trace buffer after a specified delay `Analyzer.TDelay`.

**See also**
- TrBus
- TrBus.state
- 'Trigger Commands' in 'FIRE Emulator for C166 Cell-Based-Core'
- 'CPU specific Trigger Commands' in 'MAC71xx/72xx NEXUS Debugger and Trace'
- 'Release Information' in 'Release History'

**TrBus.RESet**

Reset setting for trigger bus

**Debugger / FIRE only**

**Format:**

```
TrBus.RESet
```

Resets the settings for the trigger bus.

**See also**
- TrBus
- TrBus.state
**TrBus.Set**  
Define the target for the incoming trigger

**Debugger / FIRE only**

| Format: | TrBus.Set Break | ATrigger [ON | OFF] |

Selects the target for the incoming trigger signal.

**Break**  
Stop the program execution as soon as the external trigger signal becomes active.

**ATrigger**  
Generate a trigger for the trace as soon as the external trigger signal becomes active. A trigger for the trace can be used to stop the sampling to the trace buffer directly or after a specified delay **Analyzer.TDelay**.

See also

- TrBus
- TrBus.state
- ‘Trigger Commands’ in ‘FIRE Emulator for C166 Cell-Based-Core’
- ‘CPU specific Trigger Commands’ in ‘MAC71xx/72xx NEXUS Debugger and Trace’

---

**TrBus.state**  
Display settings for the trigger bus

**Debugger / FIRE only**

| Format: | TrBus.state |

Displays all settings for the trigger bus.

See also

- TrBus
- TrBus.Arm
- TrBus.Connect
- TrBus.Mode
- TrBus.OFF
- TrBus.Out
- TrBus.RESet
- TrBus.Set
- TrBus.Trigger
Stimulates a trigger on the trigger bus.

See also
- TrBus
- TrBus.state
Overview TrEvent

The event trigger function facilitates the connection between a time counter, or an event counter, and individual trigger events. This function can be particularly useful whenever a trigger should take place after a defined number of events, or when events fail to take place. On ECC8 no event trigger system is available, but the analyzer trigger system will work very similar.
Defines the number of occurrences or the delay time.

```plaintext
; Trigger after 1000 accesses to breakpoint ALPHA

ten.s alpha ; select breakpoint ALPHA
ten.m count ; count mode
ten.d 1000. ; event counter
ten.on ; switch-on

; Trigger after 1000 edges to probe EXTERNAL

ten.s ext ; select external input
ten.m count ; count mode
ten.d 1000. ; event counter
ten.on ; switch-on

; Trigger if breakpoint ALPHA does not respond at least every 10 ms

ten.s alpha ; select breakpoint ALPHA
ten.m notime ; mode
ten.d 10.ms ; set delay
ten.on ; switch-on

; Trigger 10 ms after breakpoint ALPHA

ten.s alpha ; select breakpoint ALPHA
ten.m thentime ; mode
ten.d 10.ms ; set delay
ten.on ; switch-on
```

See also

- TrEvent
- TrEvent.view
If **Always** is selected, the event trigger system will be active (time counters are running) when the emulation has stopped. On default set to **Running**.

**TrEvent.Enable**

**Initialization**

**ICE only**

Format:  

```
TrEvent.Enable Always | Running
```

**See also**

- TrEvent
- TrEvent.view
- 'Event Trigger System’ in 'ICE User's Guide'
- 'Event Trigger System’ in 'ICE User's Guide'

**TrEvent.Init**

**Initialization**

Format:  

```
TrEvent.Init
```

Event or time counters are initialized.

**See also**

- TrEvent
- TrEvent.view
- 'Event Trigger System’ in 'ICE User's Guide'

**TrEvent.MinInit**

**Initialization**

**ICE only**

Format:  

```
TrEvent.MinInit
```

The minimum value of the event display is reset to the maximum value.

**See also**

- TrEvent
- TrEvent.view
TrEvent.Mode

Select operation mode

ICE only

<table>
<thead>
<tr>
<th>Format: TrEvent.Mode &lt;mode&gt; [&lt;delay&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;: Count ThenCycle ThenTime NotCycle NotTime AllCycle AllTime</td>
</tr>
</tbody>
</table>

Defines the operation mode of the event trigger system.

**Count**

Event is counted, trigger is executed at event counter set to 0.

<table>
<thead>
<tr>
<th>Event</th>
<th>Value</th>
<th>Trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>Init to 3</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>Count down</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>Trigger at zero</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

;-----------------------------------------------------------------
; Trigger after 1000. rising edges on input probe T0

c.s T0; select input probe by counter
t.e.s signal+; route counter signal to event trigger
t.e.m count; use counter function
t.e.d 1000.; define delay
t.e.on; arm

;-----------------------------------------------------------------
; Trigger after 1000. accesses to breakpoint alpha

t.e.s alpha; route counter signal to event trigger
t.e.m count; use counter function
t.e.d 1000.; define delay
t.e.on; arm

©1989-2019 Lauterbach GmbH
ThenCycle

Event is recorded, trigger is executed after a given number of cycles.

<table>
<thead>
<tr>
<th>Event</th>
<th>CPU cycles</th>
<th>Value</th>
<th>Trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3 2 1 0</td>
<td></td>
</tr>
<tr>
<td>Init to 3</td>
<td>Event</td>
<td>Count down</td>
<td>Trigger at zero</td>
</tr>
</tbody>
</table>

;-----------------------------------------------------------------
; Trigger 1000 cycles after breakpoint ALPHA is reached
; select breakpoint
; use delay trigger function
; define delay value
; arm

t.e.s alpha

t.e.m thencycle

t.e.d 1000.

t.e.on

ThenTime

Event is recorded, trigger is executed after the delay time.

<table>
<thead>
<tr>
<th>Event</th>
<th>Time pulse</th>
<th>Value</th>
<th>Trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>9 9 8 7 6 5 4 3 2 1 0</td>
<td></td>
</tr>
<tr>
<td>Init time</td>
<td>Event</td>
<td>Count down</td>
<td>Trigger at zero</td>
</tr>
</tbody>
</table>

;-----------------------------------------------------------------
; Trigger 1000 cycles after falling edges on input probe T0
; select input probe by counter
; route counter signal to event trigger
; use delay trigger function
; define delay value
; arm

c.s T0

t.e.s signal-

t.e.m thentime

t.e.d 1.ms

t.e.on
NotCycle

If no event occurs during a specific number of cycles, the trigger signal will be set.

<table>
<thead>
<tr>
<th>Event</th>
<th>---------------</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CPU cycles</th>
<th>---------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>3 2 1 3 2 1 0</td>
</tr>
<tr>
<td>Trigger</td>
<td>---------------</td>
</tr>
<tr>
<td>Init to 3</td>
<td>Event Count Event Count down Trigger at zero</td>
</tr>
</tbody>
</table>

NotTime

If no event occurs within a specific amount of time, the trigger signal will be activated.

<table>
<thead>
<tr>
<th>Event</th>
<th>---------------</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Time pulse</th>
<th>---------------</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>9 8 9 8 7 5 4 3 2 9 8 7 5 4 3 2 1 0</td>
</tr>
<tr>
<td>Trigger</td>
<td>---------------</td>
</tr>
<tr>
<td>Init time</td>
<td>Event Count Event Count down Trigger at zero</td>
</tr>
</tbody>
</table>

;-----------------------------------------------------------------
;
; Trigger if the distance between two CPU cycles increases 100 us,
; e.g. by DMA cycles or CPU reset
;
;-----------------------------------------------------------------
;
; set mode
; select cycle signal for retrigger
; define time-out value
; arm event trigger system

;-----------------------------------------------------------------
;
; Trigger if the distance between two timer interrupts increases
; 1.5 ms, e.g. if the interrupt will be disabled for long time
;
; set mode
; select breakpoint
; set breakpoint to interrupt routine
; define time-out value
; arm event trigger system
### AllCycle

Trigger is always activated after a certain number of CPU cycles.

<table>
<thead>
<tr>
<th>CPU cycles</th>
<th>Value</th>
<th>Trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3 2 1 0</td>
<td>Init to 3 Count down Trigger at zero</td>
</tr>
</tbody>
</table>

### AllTime

Trigger is always executed after a fixed time.

<table>
<thead>
<tr>
<th>Event</th>
<th>Time pulse</th>
<th>Value</th>
<th>Trigger</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>9 8 7 6 5 4 3 2 1 0</td>
<td>Init time Count down Trigger at zero</td>
<td></td>
</tr>
</tbody>
</table>

```
; Run for a fixed time

te.m alltime
te.delay 1000.s
te.on
go
...```

### See also

- TrEvent
- TrEvent.view
- Event Trigger System in ICE User's Guide
The event trigger system does not trigger. The input source selection can still be used for the frequency counter or other purposes (Event field in the frequency counter sources).

```
; Use event trigger unit to measure the interrupt rate

; switch off
te.off
; select alpha breakpoint
te.select alpha
; set breakpoint to interrupt routine
b.s INT_Routine1 /a
c.s event
; select event signal by counter
c.m f
; set counter mode to frequency
c.go
    print "Interrupt rate ":
count.value()
...

de.off
; Switch cycle signal to BNC connector

t.e.s always
; select signal

; Switch breakpoint signal to BNC connector

t.e.off
; disarm event trigger
t.e.s alpha
; select breakpoint
```

See also

- TrEvent
- TrEvent.view

▲ 'Event Trigger System' in 'ICE User's Guide'
**TrEvent.ON**

**ICE only**

**Switch on**

<table>
<thead>
<tr>
<th>Format:</th>
<th>TrEvent.ON</th>
</tr>
</thead>
</table>

The event trigger system is activated and will trigger if the defined event occurs.

**See also**
- TrEvent
- TrEvent.view
- 'Event Trigger System' in 'ICE User's Guide'

---

**TrEvent.RESet**

**ICE only**

**Reset command**

<table>
<thead>
<tr>
<th>Format:</th>
<th>TrEvent.RESet</th>
</tr>
</thead>
</table>

Reset to original condition, event trigger is disabled.

**See also**
- TrEvent
- TrEvent.view
- 'Event Trigger System' in 'ICE User's Guide'
TrEvent.Select

Select event source

ICE only

Format: \texttt{TrEvent.Select \langle source\rangle}

\texttt{<source>:}

- Program | Hll | Spot
- Read | Write
- Alpha | Beta | Charly
- DataRead | DataWrite
- ExtData | ExtSynch | ExtComp
- eXception | TrInput | Glitch
- TimeOut | AnalyzerA | RBW
- EXTernal | SIGnal+ | SIGnal-
- ALways

Defines the event which is selected as source for the event trigger system. This selector can also be used to pass a line to the frequency counter (source field \texttt{Event}) or synchronize other instruments. The output to BNC is on backside.

Program, Hll, Spot, Read, Write, Alpha, Beta, Charly

Breakpoints

DataRead, DataWrite

Read/Write breakpoints and data fetch.

ExtData

External trigger input, sampled at time of CPU data acceptance.

ExtSynch

External trigger input, sampled on external cycle.

ExtComp

External trigger input, asynchronous recording.

eXception

Exception trigger (see \texttt{eXception.Trigger} command).

TrInput

Reserved.

Glitch

Glitch detector.

TimeOut

Timeout bus access (see \texttt{SYStem.TimeOut} command).

AnalyzerA

Analyzer trigger outputs.

EXTernal

External trigger input signal.

SIGnal+, SIGnal-

Universal counter signal.

ALways

Cycle signal (occurs once per CPU cycle).

\textbf{See also}

- TrEvent
- TrEvent.view
- 'Event Trigger System' in 'ICE User's Guide'

©1989-2019 Lauterbach GmbH
Displays all settings and the current status of the event trigger system.

The initial and current counter values are displayed in the top line. The value on the right (min) results from the smallest displayed value. This display value does not necessarily correspond to the minimum value because it is only updated when a new value is displayed on the monitor (software, not hardware comparator).

See also

- TrEvent
- TrEvent.MinInit
- TrEvent.RESet
- TrEvent.Delay
- TrEvent.Mode
- TrEvent.Select
- TrEvent.Enable
- TrEvent.ON
- TrEvent.OFF
- TrEvent.Init

▲ 'Event Trigger System' in 'ICE User's Guide'
See also

- TrIn.Clock
- TrIn.RESet
- TrIn.Data
- TrIn.state
- TrIn.Mask
- TrIn.Transient
- TrIn.Normal

Overview TrIn

[Diagram of TrIn Internal trigger logic]
The **TriggerIn** function can be used in conjunction with EXTERNAL probe. It allows to combine external asynchronous or synchronous events and to synchronize these with the internal trigger logic. The trigger logic consists of a programmable comparator, combined with a D-flip-flop with delayed reset for generating a short pulse. Flip-flop data signals and clock signals can be switched to any probe input, internal signals or comparator output. The output of the flip-flop (named EXT), as well as the output of the comparator (named ExtComp) can be used as a trigger event. These signals can also be routed directly to the counter and the event trigger unit for complex triggering. Synchronized to the CPU cycle the state analyzer can trigger on this events. In addition, a "transient" mode is available and can be triggered by status changes to individual input signals.

An EXTERNAL probe is not available on ECC8.

### EXTERNAL-Probe Layout

<table>
<thead>
<tr>
<th>LED</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pin</td>
<td>16</td>
<td>14</td>
<td>12</td>
<td>10</td>
<td>8</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>Pin</td>
<td>15</td>
<td>13</td>
<td>11</td>
<td>9</td>
<td>7</td>
<td>5</td>
<td>3</td>
</tr>
</tbody>
</table>

Pin 1   Line E0  
Pin 3   Line E1  
Pin 5   Line E2  
Pin 7   Line E3  
Pin 9   Line E4  
Pin 11  Line E5  
Pin 13  Line E6  
Pin 15  Line E7  
Pin 2,4,6,8,10,12,14,16 GND
Examples

;----------------------------------------------------------------------
; trigger on rising edge of E6 input line

ti.res
ti.data true ; unqualified data
ti.clock E6 + ; select clock
t.s extsynch on ; activate trigger input

;----------------------------------------------------------------------
; trigger on high level of E6 input line

ti.res
ti.data E6 ; select data bit
ti.clock cyclemid ; select internal clock
t.s extsynch on ; activate trigger input

;----------------------------------------------------------------------
; trigger if E1 is low and E0 is high at the end of the cycle

ti.res
ti.mask 0x0xxxxxx01 ; define mask for E0 and E1
ti.data extcomp ; select comparator for data input
ti.clock cycleend ; select internal clock
t.s extsynch on ; activate trigger

;----------------------------------------------------------------------
; trigger if E1 is low and E0 is high at the end of the cycle

ti.res
ti.mask 0x0xxxxxx01 ; set trigger mask
t.s extdata on ; route comparator to trigger synchronously

;----------------------------------------------------------------------
; trigger if E1 and E0 are high for a short time


ti.res
ti.mask 0x0xxxxxx11 ; set comparator mask
t.s extcomp on ; activate asynchronous trigger

;----------------------------------------------------------------------
; trigger if E0 and E1 are low at the falling edge of E6

ti.res
ti.mask 0x0xxxxxx00 ; define mask for data
ti.data extcomp ; select comparator
ti.clock E6 - ; select clock line and polarity
t.s extsynch on ; activate standard trigger

;----------------------------------------------------------------------
; trigger on every transient on input E0 and E1
ti.res
  ti.mask 0x0xxxxxx00 ; set mask for transient detection
  ti.transient ; set mode
  t.s extdata on ; set trigger synch. to CPU cycles

;---------------------------------------------
;  trigger if no clock edge arrives within 1.ms on E6

  te.select ext
  te.mode nottime
  te.delay 1.ms
  te.on

;---------------------------------------------
;  count the clock edges on E6 while E0 and E1 are both low

  count.select ext
  count.mode eventhigh
  count.go ; read-out counter value
  print "Cycles" ; count.value()
# Define clock

**Format:** \texttt{TrIn.Clock \langle clock\rangle \langle polarity\rangle}

**\langle clock\rangle:**
- \texttt{SIGnal}
- \texttt{CycleMid}
- \texttt{CycleEnd}
- \texttt{ExtComp}
- \texttt{Clock}
- \texttt{E6}
- \texttt{E7}
- \texttt{E67}

**\langle polarity\rangle:** + | -

Selects the clock signal used by the flip-flop. For more information see chapter *Function*.

<table>
<thead>
<tr>
<th>\texttt{SIGnal}</th>
<th>Universal counter signal.</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{CycleMid}</td>
<td>Sampling in mid cycle.</td>
</tr>
<tr>
<td>\texttt{CycleEnd}</td>
<td>Sampling at the end of the cycle.</td>
</tr>
<tr>
<td>\texttt{ExtComp}</td>
<td>External trigger input via comparator.</td>
</tr>
<tr>
<td>\texttt{Clock}</td>
<td>CPU clock.</td>
</tr>
<tr>
<td>\texttt{E6}</td>
<td>External trigger input, Bit 6.</td>
</tr>
<tr>
<td>\texttt{E7}</td>
<td>External trigger input, Bit 7.</td>
</tr>
<tr>
<td>\texttt{E67}</td>
<td>External trigger input, Bit 6 and Bit 7, OR-operation (AND for active-low signals).</td>
</tr>
</tbody>
</table>

See also

- \texttt{TrIn}
- \texttt{TrIn.state}
- 'External Trigger Input' in 'ICE User's Guide'

©1989-2019 Lauterbach GmbH
### Define data

**TrIn.Data**

Format:  

\[ \text{TrIn.Data} \ <\text{data}> \ [\ <\text{polarity}>] \]

#### <data>:

- **SIGnal**
- **True**
- **ExtComp**
- **E4**
- **E5**
- **E6**
- **E7**

#### <polarity>:

- +
- -

Selects the data signal used by the flip-flop. For more information see chapter **Overview TrIn**.

- **SIGnal**
  - Signal frequency counter.
- **True**
  - Active-high signal. It is used, whenever a trigger event is generated on each clock edge (unqualified trigger).
- **ExtComp**
  - EXTERNAL trigger input probe via comparator.
- **E4, E5, E6, E7**
  - EXTERNAL input probe, Bit 4, Bit 5, Bit 6 or Bit 7.

**See also**

- **TrIn**
- **Trin.state**
- 'External Trigger Input’ in 'ICE User's Guide’
**TrIn.Mask**

**ICE only**

<table>
<thead>
<tr>
<th>Format:</th>
<th>TrIn.Mask &lt;mask&gt;</th>
</tr>
</thead>
</table>

The comparator mask can be defined by the characters "0", "1" or "X" (don't care).

```plaintext
  ti.mask 0x0xxxxxxxx1 ; true if bit 0 is high
  ti.mask 0x0xxxxxx01  ; true if bit 0 is high and bit 1 is low
  ti.mask 0x01111111000000 ; true if all input lines or low
```

**See also**

- Trln
- Trln.state
- 'External Trigger Input' in 'ICE User's Guide'

---

**TrIn.Normal**

**Level operation**

**ICE only**

<table>
<thead>
<tr>
<th>Format:</th>
<th>TrIn.Normal</th>
</tr>
</thead>
</table>

The trigger system searches for the defined events. Transient trigger is switched off.

**See also**

- Trln
- Trln.state

---

**TrIn.RESet**

**Reset command**

**ICE only**

<table>
<thead>
<tr>
<th>Format:</th>
<th>TrIn.RESet</th>
</tr>
</thead>
</table>

Reset of the TrIn command to the default settings.

**See also**

- Trln
- Trln.state
- 'External Trigger Input' in 'ICE User's Guide'
Displays all setups of the **TrIn** command.

The current level of the probe line is displayed in the **state** field.

### See also
- **TrIn**
- **TrIn.Clock**
- **TrIn.Data**
- **TrIn.Normal**
- **TrIn.RESet**
- **TrIn.Transient**
- **TrIn.Mask**
- **TRIN.VALUE()**

▲ 'External Trigger Input' in 'ICE User's Guide'
TrIn.Transient

ICE only

In case of a transient trigger, the trigger event is generated whenever the level of at least one selected signal line between two sampling points changes. The selection of active inputs results from the comparator mask definition. All inputs labeled "0" or "1" are valid. Polarity has no consequences.

Format:  TrIn.Transient

; Trigger event when a change in polarity occurs at line 0 or 2 of the
; external probe

  ti.t ; select transient mode
  ti.m 0x1X1 ; define mask
  t.s ed on ; switch on

See also

* TrIn
* TrIn.state

▲ 'External Trigger Input' in 'ICE User's Guide'
TrMain

Trigger system of TRACE32-ICE

See also

- TrMain.ALways
- TrMain.Arm
- TrMain.AutoInit
- TrMain.AutoStart
- TrMain.Break
- TrMain.Count
- TrMain.Delay
- TrMain.Init
- TrMain.OFF
- TrMain.Out
- TrMain.RESet
- TrMain.Mode
- TrMain.state
- TrMain.Set
- TrMain.Trigger

▲ 'TRIGGER Functions (ICE only)' in 'General Function Reference'

Overview TrMain

The trigger system is used for collecting asynchronous events, converting them into a trigger signal, and passing this trigger signal on to the analyzer and emulation controller. The trigger system can exhibit the following conditions: OFF, ARMED, TRIGGERED or BREAKED. If the trigger system is in the ARMED condition and a trigger event occurs, the TRIGGERED condition will be the result. After the trigger delay sequence is completed the BREAK state will be the result.

Reaching the break state the emulation system or the analyzer will be stopped. The trigger source and trigger address (not ECC8) will be displayed.

There are three trigger modes:

- **Emulator Trigger**: The emulation system is stopped. The real-time emulation will be stopped at the next HLL or ASM line. If the analyzer slave mode is selected, it will be stopped too.

- **Analyzer Trigger**: The state analyzer is stopped, the real-time emulation is not affected.

- **Memory Trigger**: In memory trigger mode the state of emulation memory is locked (write protection). This function is useful only if emulation memory is 'shadowed' to the target memory.

**NOTE:** The trigger system will be blocked, if the PERF function is activated.
Trigger switch

Trigger delay

Trigger mode

E
A
M

Trigger (BNC)

Analyzer A
Analyzer B
Analyzer C

Alpha
Beta
Charly

... ...

Trigger counter
48 Bit

Event trigger

Trigger Functions

Condition: OFF
Trigger disabled

Trigger.ARM

Condition: ARMED
Trigger disabled

Trigger.Trigger
Trigger Event

Condition: TRIGGERED
Trigger event has occurred

Trigger.Break
Trigger delay finished

Condition: BREAKED
Trigger executed

Trigger.Off

Trigger.Arm

Trigger Sequence

Program
H11
Spot
Read
Write
Alpha
Beta
Charly
... ...

©1989-2019 Lauterbach GmbH

General Commands Reference Guide T 521
## Functions

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRIGGER.STATE()</td>
<td>State of the trigger system</td>
</tr>
<tr>
<td>0 = OFF</td>
<td></td>
</tr>
<tr>
<td>1 = ARMED</td>
<td></td>
</tr>
<tr>
<td>2 = TRIGGERED</td>
<td></td>
</tr>
<tr>
<td>3 = BREAKED</td>
<td></td>
</tr>
<tr>
<td>TRIGGER.ADDRESS()</td>
<td>Address of the trigger event</td>
</tr>
<tr>
<td>TRIGGER.COUNT,alpha()</td>
<td>Value of the trigger counter</td>
</tr>
<tr>
<td>TRIGGER.COUNT.beta()</td>
<td></td>
</tr>
<tr>
<td>TRIGGER.COUNT.charly()</td>
<td></td>
</tr>
<tr>
<td>TRIGGER.DELAY.TIME()</td>
<td>Value of the trigger delay counter</td>
</tr>
<tr>
<td>TRIGGER.DELAY.CYCLE()</td>
<td></td>
</tr>
<tr>
<td>TRIGGER.DELAY.TRACE()</td>
<td></td>
</tr>
<tr>
<td>TRIGGER.SOURCE()</td>
<td>Source of last trigger event as hex code (not ECC8).</td>
</tr>
<tr>
<td>TRIGGER.CYCLE()</td>
<td>Cycle type of the trigger cycle, bit 0 is read/write (not ECC8).</td>
</tr>
<tr>
<td>TRIGGER.BYTES()</td>
<td>Upper four bits contain the byte enable signals on a 32-bit bus (not ECC8).</td>
</tr>
</tbody>
</table>
TrMain.ALways

**Format:**

```
TrMain.ALways [ON | OFF]
```

Ensures that a trigger event is always generated at each cycle during real-time emulation. This function is normally used together with the **TrMain.Delay** option to run for a fixed time.

```plaintext
; Program breaks immediately after 100 µs

tm.m emulator ; emulator mode
tm.al on ; trigger at first cycle
tm.d time 100.us ; delay time is 100.us
g ; start emulation

; Analyzer break after 1000 sampled CPU cycles

tm.res ; reset trigger system
tm.m analyzer ; break analyzer only
tm.always on ; trigger immediately
tm.delay trace 1000. ; delay set to 1000 traced cycles
tm.a ; arm trigger system
```

**See also**

- TrMain
- TrMain.state
- ‘Trigger System’ in ‘ICE User’s Guide’

---

TrMain.Arm

**Release and activate trigger system**

**ICE only**

**Format:**

```
TrMain.Arm
```

After being armed, the trigger system waits for a trigger event. If the option **AutoArm** is set, the trigger system would be activated (ARMED) automatically.

```plaintext
tm.a ; Initialize and activate trigger system
```

**See also**

- TrMain.AutoInit
- TrMain.AutoStart
- TrMain
- TrMain.state
- ‘Trigger System’ in ‘ICE User’s Guide’
**TrMain.AutoInit**

**Automatic trigger initialization**

ICE only

Format: `TrMain.AutoInit [ON | OFF]`

The trigger logic can be used independently of emulation commands and emulator state. In most cases, however, both trigger and emulation system must work together. Using the `TrMain.AutoInit` command, the trigger logic will be initialized whenever real-time emulation is started.

```
tm.ai on ; Activate AUTOINIT mode
```

See also
- `TrMain.Arm`
- `TrMain`
- `TrMain.state`

▲ 'Trigger System' in 'ICE User's Guide'

---

**TrMain.AutoStart**

**Automatic trigger initialization**

ICE only

Format: `TrMain.AutoStart [ON | OFF]`

The trigger logic will be restarted only, if the trigger system is in **Break** state and emulation is started. No action is done, if the trigger system remains in **ARMED** or **TRIGGERED** state.

```
tm.ai on ; activate AUTOSTART mode
s
s
... ; no trigger
g
... ; trigger event is reached
g
; trigger is ARMED, the real-time
; emulation is started
```

See also
- `TrMain.Arm`
- `TrMain`
- `TrMain.state`

▲ 'Trigger System' in 'ICE User's Guide'
**TrMain.Break**

**ICE only**

Format: \[\text{TrMain.Break}\]

Forces the trigger system to enter the \textbf{Break} state. The command will work only, if the CPU generates cycles and the real-time emulation is running.

\textbf{See also}

- \textbf{TrMain}
- \textbf{TrMain.state}
- 'Trigger System' in 'ICE User's Guide'

**TrMain.Count**

**Set trigger counter**

ICE only

Format: \[\text{TrMain.Count} \ <counter> \ [<delay>] \ | \ \text{ON} \ | \ \text{OFF}\]

\textbf{<counter>}:

- \textbf{Alpha}
- \textbf{Beta}
- \textbf{Charly}

\textbf{<delay>}:

- \textbf{0.} ...

The three trigger counters (one in case of ECC8) can be used to count the number of accesses to a certain address. They are directly coupled with the breakpoint memory bits. If a counter is set, it will generate a trigger signal running through zero (not for ECC8).

```
E::b.s flags /alpha ; set breakpoint
E::tm.c alpha 100000. ; trigger after 100000 accesses

E::b.s flags /a ; set breakpoint
E::tm.c a 1000. ; set delay count
E::tm.c a off ; switch off trigger function
```

\textbf{See also}

- \textbf{TrMain}
- \textbf{TrMain.state}
- 'Trigger System' in 'ICE User's Guide'
TrMain.Delay

ICE only

Set trigger delay

Three different delay modes, occurring between trigger event and break state, can be defined:

- **TRace**: The number of traced records in trace memory.
- **TIme**: The trigger delay can be set from 100 ns to 300 days.
- **Cycle**: The trigger delay after CPU cycles.

Trigger delays can be used together. The first delay function which runs to zero will force the trigger system to BREAK state.

```
  tm.d trace 1000. ; Delay of 1000 trace cycles
  tm.d time 100.us ; Delay of 100 µs
  tm.d cycle 100. ; Delay of 100 cycles
```

See also
- TrMain
- TrMain.state
- 'Trigger System' in 'ICE User's Guide'

Format:  

```
TrMain.Delay <counter> [<delay> | ON | OFF]
```

- **<counter>**: TRace | TIme | Cycle
- **<delay>**: 0. …
TrMain.Init

Initialize trigger system

ICE only

Format: TrMain.Init

This function is similar to that of TrMain.Arm. If the trigger system was OFF, it would not automatically be switched-on.

See also
- TrMain
- TrMain.state

TrMain.Mode

Select mode

ICE only

Format: TrMain.Mode <mode> [ON | OFF]
<mode>: Emulator | Analyzer | Memory

The trigger mode defines the destination of the break signal.

Emulator
Real-time emulation is stopped on the next HLL or ASM instruction. If the analyzer is in SLAVE mode, it will be stopped too.

Analyzer
Analyzer triggering. Sampling is stopped, the analyzer will be ready for read-out. Real-time emulation is not terminated.

Memory
Memory triggering. Emulation memory and flag memory are locked. Access to target system memory is not affected.

See also
- TrMain
- TrMain.state
**TrMain.OFF**

**ICE only**

Switch off trigger system

Format: **TrMain.OFF**

The trigger system will not break or trigger, regardless of the other settings.

**NOTE:** Program and HLL-breakpoints are not affected, because they are not part of the trigger system.

**See also**

- TrMain
- TrMain.state
- 'Trigger System' in 'ICE User's Guide'

**TrMain.Out**

**Debugger only**

Output trigger pulse

Format: **TrMain.Out**

Outputs a high pulse of 200ns.

An analyzer trigger program of the TRACE32-ICE system controls the reaction to a trigger pulse of the BDM Debugger.

```
trigger.a if busa ; asserts trigger A of the TRACE32-ICE system,
; when the BDM Debugger outputs a trigger pulse
```

**See also**

- TrMain
- TrMain.state
### TrMain.RESet

**Reset trigger system**

<table>
<thead>
<tr>
<th>Format:</th>
<th>TrMain.RESet</th>
</tr>
</thead>
</table>

Resets trigger system to its original condition.

*(E) ReadData, WriteData* and *AnalyzerA* are selected as sources, AutoStart is activated and emulator triggering is selected.

**See also**
- TrMain
- TrMain.state

▲ ‘Trigger System’ in ‘ICE User’s Guide’

### TrMain.Set

**Select trigger sources**

| Format:          | TrMain.Set <source> [ON | OFF] |
|------------------|--------------------------------|

**<source>:**
- Program | Hll | Spot
- Read | Write
- Alpha | Beta | Charly
- DataRead | DataWrite
- ExtData | ExtSynch | ExtComp
- eXception | TrInput | Glitch
- TimeOut | AnalyzerA | RBW
- BUSA

Selects the possible trigger sources. The selected sources are ored together, i.e. the trigger will occur if one source becomes true.
This function activates breakpoints (address markers) for triggering (default OFF). The trigger system makes no difference if the bus cycle is data-fetch, op-fetch, or pre-fetch only.

DataRead, DataWrite (E)

Data read (write) to the Read (Write) breakpoints generate a trigger signal. Opfetch or other CPU cycles are ignored. This function is activated on default, because it will be used as the standard breakpoint function for variable access. To set up memory protection automatically, use the Break.SetSec command.
**ExtData (E)**

External trigger input, sampled synchronous together with the CPU data bus sampling. The trigger message will qualify exactly the cycle of the trigger event. The trigger level is defined within the `TrIn.Mask` command.

```
; Trigger if both pin 0 and 1 of EXTERNAL probe are high
  ti.mask 0x0xxxxxxx11 ; set trigger mask
  tm.s extdata on     ; activate
```

**ExtSynch (E)**

External trigger input, sampled on external clock cycle and synchronized to the CPU cycle signal. For more information see `TrIn` commands.

```
; Trigger if pin 0 is high in the rising edge of pin 7 of
; the EXTERNAL probe
  ti.mask 0x0xxxxxxxxl  ; select pin 0 only
  ti.data extcomp +     ; select comparator for data
  ti.clock e7 +         ; select input 7 for clock
  tm.s extsynch on
```
**ExtComp (E)**

External trigger input, asynchronous triggering. See `TrIn` command.

**AS-**

**Event**

**Sample**

**Trigger**

; Trigger if pin 0 and 1 of the external probe are high at the same time

ti.mask 0x0xxxxxx11 ; select pin 0 and 1 only
tm.s extcomp on ; activate asynchronous trigger

**eXception (E)**

Exception trigger (see `eXception.Trigger` command).

; Trigger asynchronous to RESET signal

x.trigger off
x.trigger reset on
t.s x on

**TrInput (E)**

Reserved for future use.

**Glitch (E)**

The glitch detector output is used as trigger signal.

; Trigger to glitch on external probe pin 0

c.select e0 ; select probe input to glitch detector
tm.s glitch on ; activate glitch trigger
g ; start real-time emulation
wait trigger.state()>=2 ; wait for trigger
print "Glitch found when accessing address "
trigger.address() ... "Glitch found when accessing address "

**TimeOut (E)**

©1989-2019 Lauterbach GmbH
Time-out on bus access (see SYStem.TimeOut command).

```
; Trigger to bus time-out

sys.to 10.us ; set time-out value
tm.s timeout on ; activate time-out trigger
g wait trigger.state()>=2 ; start real-time emulation
print "No READY signal generated on address ";
  trigger.address()
...
```

**AnalyzerA (E)**

Analyzer trigger output.

**RWB (E)**

Read-before-write triggering. This function is used to detect accesses to uninitialized variables or memory areas. To preset the flags in order to the linker information use the FLAG.SetSec command.

```
; Read-before-write trigger

flag.setsec ; set write flag in program and
  tm.s rbw on ; initialized data areas
  g wait trigger.state()>=2 ; activate read-before-write trigger
print "Uninitialized variable on address ";
  trigger.address()
...
```

**See also**

- TrMain
- TrMain.state
- 'Trigger System' in 'ICE User's Guide'
- 'Release Information' in 'Release History'
TrMain.state

Trigger state display

Format: TrMain.state

See also
- TrMain
- TrMain.AutoStart
- TrMain.Init
- TrMain.RESet
- TRIGGER.ADDRESS()
- TRIGGER.SOURCE()
- TrMain.ALways
- TrMain.Break
- TrMain.Mode
- TrMain.Set
- TRIGGER.BYTES()
- TRIGGER.CYCLE()
- TrMain.Count
- TrMain.OFF
- TrMain.Trigger
- TRIGGER.ACCESS()
- TRIGGER.OFFSET()
- TrMain.Arm
- TrMain.Count
- TrMain.OFF
- TrMain.Trigger
- TRIGGER.ACCESS()
- TRIGGER.OFFSET()
- TrMain.AutoInit
- TrMain.Delay
- TrMain.Out
- TRIGGER.ACCESS()
- TRIGGER.OFFSET()

▲ 'Trigger System' in 'ICE User's Guide'

TrMain.Trigger

Trigger

Format: TrMain.Trigger

Execute trigger manually or by script.

See also
- TrMain
- TrMain.state

▲ 'Trigger System' in 'ICE User's Guide'
See also

- **TrOnchip.Address**
- **TrOnchip.CONVert**
- **TrOnchip.CYcle**
- **TrOnchip.Data**
- **TrOnchip.RESet**
- **TrOnchip.Set**
- **TrOnchip.state**
- **TrOnchip.TaskID**
- **TrOnchip.TEnable**
- 'TrOnchip Commands' in 'XC2000/XC16x/C166CBC Debugger'
- 'CPU specific TrOnchip Commands' in 'CPU32/ColdFire Debugger and Trace'
- 'AndesCore Specific TrOnchip Commands' in 'Andes Debugger'
- 'APEX Specific TrOnchip Commands' in 'APEX Debugger'
- 'On-chip Breakpoints/Actionpoints' in 'ARC Debugger'
- 'ARM Specific TrOnchip Commands' in 'ARM Debugger'
- 'ARM specific TrOnchip Commands' in 'ARMv8-A/-R Debugger'
- 'CPU specific TrOnchip Commands' in 'AVR32 Debugger and NEXUS Trace'
- 'CPU specific TrOnchip Commands' in 'AVR8 Debugger'
- 'Beyond Specific TrOnchip Commands' in 'Beyond Debugger and Trace'
- 'TrOnchip Commands' in 'TMS320C2X Debugger'
- 'TrOnchip Commands' in 'TMS320C5X Debugger'
- 'TrOnchip Commands' in 'TMS320C6X Debugger'
- 'ARM specific TrOnchip Commands' in 'Cortex-M Debugger'
- 'CPU specific TrOnchip Commands' in 'eTPU Debugger and Trace'
- 'TrOnchip Commands' in 'H8S/23x9 Debugger'
- 'TrOnchip Commands' in 'MCS12 Debugger'
- 'TrOnchip Commands' in 'Hexagon Debugger'
- 'IPU Specific TrOnchip Commands' in 'IPU Debugger'
- 'TrOnchip Commands' in 'M8051EW Debugger'
- 'Trigger On-chip Commands' in 'M-Core Debugger'
- 'TrOnchip Commands' in 'MicroBlaze Debugger and Trace'
- 'On-chip Breakpoints' in 'MIPS Debugger and Trace'
- 'CPU specific TrOnchip Commands' in 'MMDSP Debugger'
- 'Command Reference: TrOnchip' in 'Qoriva MPC5xxx/SPC5xx Debugger and NEXUS Trace'
- 'msp430 Specific TrOnchip Commands' in 'MSP430 Debugger'
- 'TrOnchip Commands' in 'NIOS II Debugger and Trace'
- 'CPU specific TrOnchip Commands' in 'PPC600 Family Debugger'
- 'CPU specific TrOnchip Commands' in 'MPC5xx/Sxx Debugger and Trace'
- 'CPU specific TrOnchip Commands' in 'PQIII Debugger'
- 'CPU specific TrOnchip Commands' in 'QoriQ Debugger and NEXUS Trace'
- 'TrOnchip Commands' in 'R8051XC Debugger'
- 'CPU specific TrOnchip Commands' in 'RH850 Debugger and Trace'
- 'Nexus specific TrOnchip Commands' in 'RH850 Debugger and Trace'
- 'CPU specific TrOnchip Commands' in 'RH850 Debugger and Trace'
- 'TrOnchip Commands' in 'RX Debugger'
- 'CPU specific TrOnchip Commands' in 'SH2, SH3 and SH4 Debugger'
- 'TrOnchip' in 'StarCore Debugger and Trace'
- 'CPU specific TrOnchip Commands' in 'TriCore Debugger and Trace'
- 'TrOnchip Commands' in 'V850 Debugger and Trace'
- 'CPU specific TrOnchip Commands - Onchip Triggers' in 'Intel® x86/x64 Debugger'
- 'TrOnchip Commands' in 'XC800 Debugger'
- 'CPU specific TrOnchip Commands' in 'XTENSA Debugger'
- 'CPU specific TrOnchip Commands' in 'ZSP Debugger'
- 'Specific TrOnchip Commands' in 'FIRE Emulator for C166 Cell-Based-Core'
- 'On-chip Trigger System' in 'FIRE Emulator for C166 Family'
- 'CPU specific TrOnchip Commands' in 'ARM and XSCALE Monitor'
- 'TrOnchip' in 'C166 Monitor'
- 'TrOnchip Commands' in 'SH2 Monitor'
- 'ARM specific TrOnchip Commands' in 'MAC71xx/72xx NEXUS Debugger and Trace'

©1989-2019 Lauterbach GmbH
TrOnchip.Address

Format: TrOnchip.Address

See also
- TrOnchip
- TrOnchip.state
- ‘CPU specific TrOnchip Commands’ in ‘MMDSP NEXUS Debugger and Trace’
- ‘Release Information’ in ‘Release History’
- ‘On-chip Breakpoints/Actionpoints’ in ‘Simulator for ARC’

TrOnchip.CONVert

Adjust range breakpoint in onchip registers

Format: TrOnchip.CONVert

See also
- TrOnchip
- TrOnchip.state
- ‘CPU specific TrOnchip Commands’ in ‘MMDSP NEXUS Debugger and Trace’
- ‘Release Information’ in ‘Release History’
- ‘On-chip Breakpoints/Actionpoints’ in ‘Simulator for ARC’

©1989-2019 Lauterbach GmbH
TrOnchip.CYcle

See also

- TrOnchip
- TrOnchip.state

- 'Trigger On-chip Commands' in 'M-Core Debugger'
- 'CPU specific TrOnchip Commands' in 'TriCore Debugger and Trace'
- 'Specific TrOnchip Commands' in 'FIRE Emulator for C166 Cell-Based-Core'
- 'CPU specific TrOnchip Commands' in 'ARM and XSCALE Monitor'
- 'CPU specific TrOnchip Commands' in 'Simulator for TriCore'

Format: 

```
TrOnchip.CYcle
```
TrOnchip.RESet

Reset settings to defaults

Set on-chip trigger system to initial state.

See also

- `TrOnchip`
- `TrOnchip.state`
- 'Nexus specific TrOnchip Commands’ in ‘Qorivva MPC5xxx/SPC5xx Debugger and NEXUS Trace’
- 'Nexus specific TrOnchip Commands’ in ‘RH850 Debugger and Trace’
- ‘Specific TrOnchip Commands’ in ‘FIRE Emulator for C166 Cell-Based-Core’
TrOnchip.Set

See also

- TrOnchip
- TrOnchip.state

- CPU specific TrOnchip Commands
  - in 'PPC400/PPC440 Debugger and Trace'
  - in 'PPC600 Family Debugger'
  - in 'MPC5xx/8xx Debugger and Trace'
  - in 'PQIII Debugger'
  - in 'QorIQ Debugger and NEXUS Trace'
  - in 'R8051XC Debugger'
  - in 'RH850 Debugger and Trace'
  - in 'PPC600 Family Debugger'
  - in 'StarCore Debugger and Trace'
  - in 'QorIQ Debugger and NEXUS Trace'
  - in 'PPC400/PPC440 Debugger and Trace'
  - in 'MPC5xx/8xx Debugger and Trace'
  - in 'PQIII Debugger'
  - in 'QorIQ Debugger and NEXUS Trace'
  - in 'TrOnchip' in 'FIRE Emulator for C166 Cell-Based-Core'
  - in 'FIRE Emulator for HC12/MCS12'
  - in 'FIRE Emulator for SH2'
  - in 'ARM and XSCALE Monitor'
  - in 'C166 Monitor'
  - in 'SH2 Monitor'
  - in 'TrOnchip' in 'TriCore Monitor'
  - in 'x186 Monitor'
  - in 'ARM Debugger'
  - in 'ARMv8-A/R Debugger'
  - in 'Cortex-M Debugger'
  - in 'eTPU Debugger and Trace'
  - in 'Ooriva MPC5xx/SPC5xx Debugger and NEXUS Trace'
  - in 'PPC400/PPC440 Debugger and Trace'
  - in 'MPC5xx/8xx Debugger and Trace'
  - in 'PQIII Debugger'
  - in 'QorIQ Debugger and NEXUS Trace'
  - in 'FIRE Emulator for SH2'
  - in 'MAC71xx/72xx NEXUS Debugger and Trace'
  - in 'Simulator for ARC'
  - in 'Simulator for TriCore'

Break on event
TrOnchip.state

Display onchip trigger window

**Format:**

```
TrOnchip.state
```

Displays a window with the state of the on-chip trigger setting.

**See also**

- TrOnchip
- TrOnchip.Address
- TrOnchip.CONVert
- TrOnchip.CYcle
- TrOnchip.Data
- TrOnchip.RESet
- TrOnchip.Set
- TrOnchip.TaskID

- ‘ARM specific TrOnchip Commands’ in ‘ARMv8-A/-R Debugger’
- ‘TrOnchip Commands’ in ‘CEVA-X Debugger’
- ‘TrOnchip’ in ‘M32R Debugger and Trace’
- ‘TrOnchip Commands’ in ‘CEVA-Oak/Teak/TeakLite Debugger’
- ‘CPU specific TrOnchip Commands’ in ‘PPC400/PPC440 Debugger and Trace’
- ‘TrOnchip’ in ‘StarCore Debugger and Trace’
- ‘CPU specific TrOnchip Commands’ in ‘TriCore Debugger and Trace’
- ‘CPU specific TrOnchip Commands’ in ‘FIRE Emulator for HC12/MCS12’
- ‘Specific TrOnchip Commands’ in ‘FIRE Emulator for SH2’
- ‘TrOnchip Commands’ in ‘x186 Monitor’
- ‘Release Information’ in ‘Release History’
- ‘CPU specific TrOnchip Commands’ in ‘Simulator for TriCore’

TrOnchip.TaskID

tbd.

**Format:**

```
TrOnchip.TaskID
```

**See also**

- TrOnchip
- TrOnchip.state

- ‘Specific TrOnchip Commands’ in ‘FIRE Emulator for C166 Cell-Based-Core’

TrOnchip.TEnable

tbd.

**See also**

- TrOnchip
- TrOnchip.state

- ‘TrOnchip Commands’ in ‘XC2000/XC16x/C166CBC Debugger’
- ‘CPU specific TrOnchip Commands’ in ‘CPU32/ColdFire Debugger and Trace’
- ‘ARM Specific TrOnchip Commands’ in ‘ARM Debugger’
- ‘CPU specific TrOnchip Commands’ in ‘eTPU Debugger and Trace’
- ‘CPU specific TrOnchip Commands’ in ‘PPC400/PPC440 Debugger and Trace’
- ‘CPU specific TrOnchip Commands’ in ‘PPC600 Family Debugger’
- ‘CPU specific TrOnchip Commands’ in ‘MPC5xx/8xx Debugger and Trace’
- ‘TrOnchip’ in ‘C166 Monitor’
▲ ‘TrOnchip’ in ‘TriCore Monitor’
▲ ‘ARM specific TrOnchip Commands’ in ‘MAC71xx/72xx NEXUS Debugger and Trace’
TrPOD

Trigger probe

See also
- TrPOD.Clock
- TrPOD.Mode
- TrPOD.state
- TrPOD.ClockPOL
- TrPOD.OFF
- TrPOD.ON
- TrPOD.Data
- TrPOD.DataPOL
- TrPOD.RESet

TrPOD.Clock

Defines data mask

Format: \texttt{TrPOD.Clock [mask]}

The clock mask is defined. Every input line can be high or low or don’t care.

See also
- TrPOD
- TrPOD.state

TrPOD.ClockPOL

Defines data polarity

Format: \texttt{TrPOD.ClockPOL [polarity]}

\texttt{<polarity>:} + \mid -

The clock polarity can be set to true or false.

See also
- TrPOD
- TrPOD.state
TrPOD.Data  Defines data mask

Format:  

\texttt{TrPOD.Data } \begin{pmatrix} <mask> \end{pmatrix}

The data mask is defined. Every input line can be high or low or don't care.

See also

- TrPOD
- TrPOD.state

TrPOD.DataPOL  Defines data polarity

Format:  

\texttt{TrPOD.DataPOL } \begin{pmatrix} <polarity> \end{pmatrix}

\begin{align*}
\begin{array}{c}
<polarity> : \\
+ | - \\
\end{array}
\end{align*}

The data polarity can be set to true or false.

See also

- TrPOD
- TrPOD.state
TrPOD.Mode

Defines data polarity

<table>
<thead>
<tr>
<th>Format:</th>
<th>TrPOD.Mode [&lt;mode&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;mode&gt;:</td>
<td>DATA, CLOCK, SYNC, LONGER, SHORTER, GLITCH, GLITCH+, GLITCH-</td>
</tr>
</tbody>
</table>

The state display shows all the settings of the trigger probe and the level of the input pins.

**DATA**
Asynchronous trigger on inputs with data comparator

**CLOCK**
Asynchronous trigger on inputs with clock comparator

**SYNC**
Synchronous trigger

**LONGER**
Pulse width trigger when pulse exceeds time

**SHORTER**
Pulse width trigger when pulse width below time limit

**GLITCH**
Glitch trigger on both edges

**GLITCH+**
Glitch trigger on positive glitch

**GLITCH-**
Glitch trigger on negative glitch

See also
- TrPOD
- TrPOD.state

©1989-2019 Lauterbach GmbH
**TrPOD.OFF**  
**Switch off**

Format:  
`TrPOD.OFF`

The trigger probe is disabled.

See also  
- TrPOD  
- TrPOD.state

**TrPOD.ON**  
**Switch on**

Format:  
`TrPOD.ON`

The trigger probe is enabled.

See also  
- TrPOD  
- TrPOD.state

**TrPOD.RESet**  
**Reset command**

Format:  
`TrPOD.RESet`

The trigger probe is initialized to the default setup condition.

See also  
- TrPOD  
- TrPOD.state
Using this command the operating mode of the analyzer may be selected. During operation this command displays the current state of the analyzer.

**state** Displaces the current signal levels on the input lines.

**OFF** Indicates that the trigger is deactivated

**ON** Indicates that the trigger probe is activated

**See also**
- TrPOD
- TrPOD.Clock
- TrPOD.ClockPOL
- TrPOD.Data
- TrPOD.DataPOL
- TrPOD.Mode
- TrPOD.ON
- TrPOD.OFF
- TrPOD.RESet
- TrPOD.Time

▲ 'Release Information' in 'Release History'
TrPOD.Time

Defines the time for the pulse width trigger

Format:  

\texttt{TrPOD.Time [<time>]}

The time limit for the pulse width detection can be set between 20 ns and 6 ms.

See also

- TrPOD
- TrPOD.state