TRACE32 Online Help

TRACE32 Directory

TRACE32 Index

TRACE32 Documents .................................................................................................................. 1

General Commands .............................................................................................................. 1

MAP .......................................................................................................................................... 7

Mapping-System of TRACE32-ICE
Selecting Mode
Definition Phase
Activation Phase
Premapper
Fine Mapping
Internal/External Mapping
Write Protection
Processor Specific Memory Attributes
Wait States
Mapping the EPROM Simulator for BDM/ROM

MAP.Ack Generate acknowledge signals 11
MAP.ADelay tbd. 13
MAP.AFlag tbd. 13
MAP.BANK Set bank range 13
MAP.BOnchip Use on-chip breakpoints 14
MAP.Break Map break memory 14
MAP.BURST Burst area mapping 15

The Usage of the MAP.BUSx Commands
Debugger 16
EEPROM/FLASH Simulator 17
TRACE32-ICE 18

MAP.BUS8 Bus width mapping 18
MAP.BUS16 Bus width mapping 19
MAP.BUS24 Bus width mapping 19
MAP.BUS32 Bus width mapping 20
MAP.BUSEXT External bus mapping 20
MAP.BYTE Set EPROM width 21
MAP.CFlag tbd. 21
MAP.COMSTART tbd. 22

©1989-2015 Lauterbach GmbH
MAP.CONST    Mapped address range contains constants 22
MAP.CS       tbd. 23
MAP.Data     Map data memory 23
MAP.DEFault  Standard memory mapping 23
MAP.DenyAccess Deny memory access by TRACE32 24
MAP.DenyBurst Deny burst access to memory by TRACE32 25
MAP.DMA      DMA area mapping 26
MAP.DMUX tbd. 26
MAP.Extern   External memory 26
MAP.Flag     Map flag memory 26
MAP.FRAG     Form fragment 27
MAP.GAP      Define gap 27
MAP.Intern   Internal memory 28
MAP.List     List allocation 29
MAP.LittleEndian Define little endian area 30
MAP.MFlag tbd. 31
MAP.Mirror   Mirroring 31
MAP.Mode     Mode 32
MAP.MONITOR  tbd. 33
MAP.NEW      Initialization 33
MAP.NoAck    Disable acknowledge signals 33
MAP.NoAFlag  tbd. 34
MAP.NOBANK   Release bank area 34
MAP.NoBOchip Use on-chip breakpoints 35
MAP.NoBreak  Release break memory 35
MAP.NoBURST  Burst area mapping 35
MAP.NOBUS16  Bus width mapping 36
MAP.NOBUS32  Bus width mapping 36
MAP.NOBUS8   Bus width mapping 36
MAP.NoCache  Cache area mapping 37
MAP.NoCONST  Undo the MAP.CONST settings 37
MAP.NOCs tbd. 37
MAP.NoData  Release data memory 37
MAP.NoDenyAccess Switch off deny access for TRACE32 38
MAP.NoDenyBurst Undo settings of MAP.DENYBURST command 38
MAP.NODMA DMA area mapping 39
MAP.NoDMUX tbd. 39
MAP.NoFlag  Release flag memory 39
MAP.NOFrag  Switch off fragmentation 39
MAP.NOGAP  Switch off gap 40
MAP.NoLittleEndian Switch off little endian 40
MAP.NoMFlag tbd. 40
MAP.NoOPFetch Switch off opfetch area mapping 41
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP.NOPAGE</td>
<td>Define pages</td>
<td>41</td>
</tr>
<tr>
<td>MAP.NoPOOL</td>
<td>Undo setting of MAP.POOL command</td>
<td>42</td>
</tr>
<tr>
<td>MAP.NoProtect</td>
<td>Erase write protection</td>
<td>42</td>
</tr>
<tr>
<td>MAP.NoRam</td>
<td>Release RAM</td>
<td>42</td>
</tr>
<tr>
<td>MAP.NoReadFlag</td>
<td>Undo setting of MAP.READFLAG command</td>
<td>42</td>
</tr>
<tr>
<td>MAP.NOROM</td>
<td>Unmap ESI</td>
<td>43</td>
</tr>
<tr>
<td>MAP.NoShadow</td>
<td>Undo setting of MAP.SHADOW command</td>
<td>43</td>
</tr>
<tr>
<td>MAP.NoSOOnchip</td>
<td>Undo setting of MAP.SONCHIP command</td>
<td>43</td>
</tr>
<tr>
<td>MAP.NOSWAP</td>
<td>Keep byte order</td>
<td>44</td>
</tr>
<tr>
<td>MAP.NoUpdateOnce</td>
<td>Undo setting of MAP.UPDATEONCE command</td>
<td>44</td>
</tr>
<tr>
<td>MAP.NoVMREAD</td>
<td>Undo the setting of the MAP.VMREAD command</td>
<td>45</td>
</tr>
<tr>
<td>MAP.NoVOLATILE</td>
<td>Undo the setting of the MAP.VOLATILE command</td>
<td>45</td>
</tr>
<tr>
<td>MAP.NOWB</td>
<td>Premapper settings</td>
<td>45</td>
</tr>
<tr>
<td>MAP.NoXBus</td>
<td>Maps XBUS to XPER device</td>
<td>45</td>
</tr>
<tr>
<td>MAP.OPFetch</td>
<td>Opfetch area mapping</td>
<td>46</td>
</tr>
<tr>
<td>MAP.PAGE</td>
<td>Define pages</td>
<td>46</td>
</tr>
<tr>
<td>MAP.POOL</td>
<td>tbd</td>
<td>47</td>
</tr>
<tr>
<td>MAP.PRE</td>
<td>Premapper</td>
<td>47</td>
</tr>
<tr>
<td>MAP.Protect</td>
<td>Write protection</td>
<td>48</td>
</tr>
<tr>
<td>MAP.Ram</td>
<td>Map RAM</td>
<td>48</td>
</tr>
<tr>
<td>MAP.ReadFlag</td>
<td>tbd</td>
<td>49</td>
</tr>
<tr>
<td>MAP.RELOCate</td>
<td>Relocate ROM area</td>
<td>49</td>
</tr>
<tr>
<td>MAP.RESet</td>
<td>Reset</td>
<td>50</td>
</tr>
<tr>
<td>MAP.ROM</td>
<td>Map ESI</td>
<td>50</td>
</tr>
<tr>
<td>MAP.Shadow</td>
<td>tbd</td>
<td>51</td>
</tr>
<tr>
<td>MAP.SOOnchip</td>
<td>tbd</td>
<td>51</td>
</tr>
<tr>
<td>MAP.SPlit</td>
<td>Splitting</td>
<td>51</td>
</tr>
<tr>
<td>MAP.state</td>
<td>State</td>
<td>52</td>
</tr>
<tr>
<td>MAP.SWAP</td>
<td>Change byte order</td>
<td>53</td>
</tr>
<tr>
<td>MAP.UpdateOnce</td>
<td>Read memory only once each time CPU stops</td>
<td>53</td>
</tr>
<tr>
<td>MAP.VERIFY</td>
<td>Internal usage only</td>
<td>54</td>
</tr>
<tr>
<td>MAP.VMREAD</td>
<td>Redirect memory reads/writes to TRACE32 virtual memory</td>
<td>54</td>
</tr>
<tr>
<td>MAP.VOLATILE</td>
<td>Mapped address range is volatile</td>
<td>54</td>
</tr>
<tr>
<td>MAP.Wait</td>
<td>Wait cycles</td>
<td>54</td>
</tr>
<tr>
<td>MAP.WORD</td>
<td>Set EPROM width</td>
<td>55</td>
</tr>
<tr>
<td>MAP.WriteFlag</td>
<td>tbd</td>
<td>56</td>
</tr>
<tr>
<td>MAP.XBus</td>
<td>Maps the XBUS to emulation memory</td>
<td>56</td>
</tr>
<tr>
<td>MCDS</td>
<td>Multicore debug solution</td>
<td>57</td>
</tr>
<tr>
<td>MCDS.CLEAR</td>
<td>Clear programming and initialize MCDS registers</td>
<td>57</td>
</tr>
<tr>
<td>MCDS.CLOCK</td>
<td>Configure MCDS clock system</td>
<td>58</td>
</tr>
<tr>
<td>MCDS.CLOCK DEPRECATED</td>
<td>Enable deprecated MCDS clock programming</td>
<td>59</td>
</tr>
<tr>
<td>MCDS.CLOCK EXTerm</td>
<td>Set the external clock frequency</td>
<td>60</td>
</tr>
<tr>
<td>Command</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td></td>
</tr>
<tr>
<td>MCDS.CLOCK Frequency</td>
<td>Specify MCDS-related frequencies by commands</td>
<td></td>
</tr>
<tr>
<td>MCDS.CLOCK Frequency.McdsClock</td>
<td>Specify the MCDS clock</td>
<td></td>
</tr>
<tr>
<td>MCDS.CLOCK Frequency.ReferenceClock</td>
<td>Specify the reference clock</td>
<td></td>
</tr>
<tr>
<td>MCDS.CLOCK MCDSDIV</td>
<td>Set divider for generating the MCDS clock</td>
<td></td>
</tr>
<tr>
<td>MCDS.CLOCK REFDIV</td>
<td>Set divider for generating the reference clock</td>
<td></td>
</tr>
<tr>
<td>MCDS.CLOCK SYStem</td>
<td>Set the system clock frequency</td>
<td></td>
</tr>
<tr>
<td>MCDS.CLOCK TIMER</td>
<td>Setup timer for periodic trigger event</td>
<td></td>
</tr>
<tr>
<td>MCDS.CLOCK TimeStamp</td>
<td>Force decoding of timestamp messages</td>
<td></td>
</tr>
<tr>
<td>MCDS.DIAG</td>
<td>Enable diagnostic output</td>
<td></td>
</tr>
<tr>
<td>MCDS.INFO</td>
<td>Information on MCDS and usage</td>
<td></td>
</tr>
<tr>
<td>MCDS.Init</td>
<td>Initialize MCDS registers</td>
<td></td>
</tr>
<tr>
<td>MCDS.LOG</td>
<td>MCDS register access logging</td>
<td></td>
</tr>
<tr>
<td>MCDS.LOG CLEAR</td>
<td>Clears all MCDS register access log entries</td>
<td></td>
</tr>
<tr>
<td>MCDS.LOG OFF</td>
<td>Disable MCDS register accesses logging</td>
<td></td>
</tr>
<tr>
<td>MCDS.LOG ON</td>
<td>Enable MCDS register accesses logging</td>
<td></td>
</tr>
<tr>
<td>MCDS.LOG OWner</td>
<td>Select MCDS owners for logging</td>
<td></td>
</tr>
<tr>
<td>MCDS.LOG view</td>
<td>View MCDS register accesses</td>
<td></td>
</tr>
<tr>
<td>MCDS.OFF</td>
<td>Disable MCDS programming</td>
<td></td>
</tr>
<tr>
<td>MCDS.ON</td>
<td>Enable MCDS programming</td>
<td></td>
</tr>
<tr>
<td>MCDS.Option</td>
<td>Control MCDS feature behavior</td>
<td></td>
</tr>
<tr>
<td>MCDS.Option AddressBreak</td>
<td>Use MCDS for address breakpoints</td>
<td></td>
</tr>
<tr>
<td>MCDS.Option DataAssign</td>
<td>Data assignment in trace listing</td>
<td></td>
</tr>
<tr>
<td>MCDS.Option DataBreak</td>
<td>Use MCDS for data breakpoints</td>
<td></td>
</tr>
<tr>
<td>MCDS.Option eXception</td>
<td>Exception identification in trace decoder</td>
<td></td>
</tr>
<tr>
<td>MCDS.Option ProgramBreak</td>
<td>Use MCDS for program breakpoints</td>
<td></td>
</tr>
<tr>
<td>MCDS.Option QuickOFF</td>
<td>Disable trace recording by hardware</td>
<td></td>
</tr>
<tr>
<td>MCDS.Option TTRESet</td>
<td>Enable generation of reset information in trace</td>
<td></td>
</tr>
<tr>
<td>MCDS.PortSIZE</td>
<td>Set number of used Aurora lanes</td>
<td></td>
</tr>
<tr>
<td>MCDS.PortSPEED</td>
<td>Set Aurora lane speed</td>
<td></td>
</tr>
<tr>
<td>MCDS.Program</td>
<td>Program MCDS triggers</td>
<td></td>
</tr>
<tr>
<td>MCDS.Register</td>
<td>Open window with MCDS registers</td>
<td></td>
</tr>
<tr>
<td>MCDS.ReProgram</td>
<td>Load and execute an OCTL trigger program</td>
<td></td>
</tr>
<tr>
<td>MCDS.RESet</td>
<td>Reset the MCDS unit in the debug tool</td>
<td></td>
</tr>
<tr>
<td>MCDS.RM</td>
<td>MCDS resource management commands</td>
<td></td>
</tr>
<tr>
<td>MCDS.RM ReSTore</td>
<td>Restore MCDS registers</td>
<td></td>
</tr>
<tr>
<td>MCDS.RM WriteTarget</td>
<td>Flush MCDS register cache</td>
<td></td>
</tr>
<tr>
<td>MCDS.SessionKEY</td>
<td>Provide MCDS session key</td>
<td></td>
</tr>
<tr>
<td>MCDS.Set</td>
<td>Program MCDS on hardware level</td>
<td></td>
</tr>
<tr>
<td>MCDS.SOURCE</td>
<td>Set MCDS trace sources</td>
<td></td>
</tr>
<tr>
<td>MCDS.SOURCE.ALL</td>
<td>Enable all MCDS trace sources</td>
<td></td>
</tr>
<tr>
<td>MCDS.SOURCE.DEFault</td>
<td>Set default MCDS trace sources.</td>
<td></td>
</tr>
<tr>
<td>MCDS.SOURCE.Set</td>
<td>Set individual MCDS trace sources.</td>
<td></td>
</tr>
</tbody>
</table>
MCDS.SOURCE.NONE Disable all MCDS trace sources.
MCDS.state Display MCDS configuration window
MCDS.TimeStamp Enable MCDS trace sources
MCDS.TraceBuffer Configure MCDS trace buffer
MCDS.TraceBuffer ARRAY Select MCDS trace buffer array
MCDS.TraceBuffer DETECT Auto-detect MCDS trace buffer configuration
MCDS.TraceBuffer LowerGAP Set MCDS trace buffer lower gap
MCDS.TraceBuffer NoStealing Prevent conflicts with third-party tools
MCDS.TraceBuffer SIZE Set MCDS trace buffer size
MCDS.TraceBuffer UpperGAP Set MCDS trace buffer upper gap
MCDSBase<trace> Non-optimized trace MCDS trace processing
MCDSDCA<trace> MCDS trace processing with data cycle assignment
MCDSDDTU<trace> MCDS trace processing with DDTU reordering

MergedAnalyzer .......................................................... 100
Merge Program-Flow-only ETM and HTM Data Access Trace
Merge Program-Flow-only ETM and ITM DataPC Trace

MIPS .......................................................... 103
Basics
Tips
MIPS.PROfileChart Profile charts for MIPS
MIPS.PROfileChart.GROUP MIPS profile chart for groups
MIPS.PROfileChart.Line MIPS per high-level language line graphically
MIPS.PROfileChart.RWINST MIPS per cycle type graphically
MIPS.PROfileChart.sYmbol MIPS for all program symbols graphically
MIPS.PROfileChart.TASK MIPS per task graphically
MIPS.PROfileSTATistic Profile statistics for MIPS
MIPS.PROfileSTATistic.GROUP MIPS per GROUP as profile chart
MIPS.PROfileSTATistic.Line MIPS per high-level language line as table
MIPS.PROfileSTATistic.RWINST MIPS per cycle type as table
MIPS.PROfileSTATistic.sYmbol MIPS for all program symbols as table
MIPS.PROfileSTATistic.TASK MIPS per task as table
MIPS.STATistic Statistical analysis for MIPS
MIPS.STATistic.GROUP MIPS statistic for groups
MIPS.STATistic.RWINST MIPS per cycle type numerically
MIPS.STATistic.sYmbol MIPS for all program symbols numerically
MIPS.STATistic.TASK MIPS per task numerically

MMU .......................................................... 116
MMU Memory management unit
MMU.DUMP Dump MMU tables
MMU.FORMAT Define MMU table structure
MMU.INFO View all translation information related to an address
MMU.ITLB tbd.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMU.ITLBSCAN</td>
<td>Handles MMU table for the current process</td>
<td>122</td>
</tr>
<tr>
<td>MMU.PageTable</td>
<td></td>
<td>122</td>
</tr>
<tr>
<td>MMU.PTE</td>
<td>Display MMU PTE entries</td>
<td>122</td>
</tr>
<tr>
<td>MMU.PTESCAN</td>
<td>Load MMU PTE table from probe</td>
<td>123</td>
</tr>
<tr>
<td>MMU.SCAN</td>
<td>Scan MMU tables (static snapshot)</td>
<td>123</td>
</tr>
<tr>
<td>MMU.Set</td>
<td>Set MMU registers or tables</td>
<td>125</td>
</tr>
<tr>
<td>MMU.TaskPageTable</td>
<td>Handles MMU table for a specific process</td>
<td>126</td>
</tr>
<tr>
<td>MMU.TDUMP</td>
<td></td>
<td>tbd.</td>
</tr>
<tr>
<td>MMU.TLB</td>
<td>Display MMU TLB entries</td>
<td>127</td>
</tr>
<tr>
<td>MMU.TLBINIT</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>MMU.TLBRESET</td>
<td></td>
<td>127</td>
</tr>
<tr>
<td>MMU.TLBSCAN</td>
<td>Load MMU TLB table from probe</td>
<td>128</td>
</tr>
<tr>
<td>MMU.TLBSET</td>
<td>Set MMU TLB entry</td>
<td>128</td>
</tr>
<tr>
<td>MMU.TRANS</td>
<td>Transparent banking area</td>
<td>129</td>
</tr>
<tr>
<td>MMU.TSCAN</td>
<td></td>
<td>129</td>
</tr>
<tr>
<td>MMU.UTLB</td>
<td></td>
<td>tbd.</td>
</tr>
<tr>
<td>MMU.UTLBSCAN</td>
<td></td>
<td>tbd.</td>
</tr>
<tr>
<td>MMU.view</td>
<td>View MMU registers</td>
<td>130</td>
</tr>
<tr>
<td>MMX</td>
<td>MMX registers (MultiMedia eXtension)</td>
<td>131</td>
</tr>
<tr>
<td>MMX.Init</td>
<td>Initialize MMX registers</td>
<td>131</td>
</tr>
<tr>
<td>MMX.OFF</td>
<td>Inhibit MMX accesses by the debugger</td>
<td>131</td>
</tr>
<tr>
<td>MMX.ON</td>
<td>Permit MMX accesses by the debugger</td>
<td>132</td>
</tr>
<tr>
<td>MMX.Set</td>
<td>Modify MMX registers</td>
<td>132</td>
</tr>
<tr>
<td>MMX.view</td>
<td>Open MMX register window</td>
<td>133</td>
</tr>
<tr>
<td>Mode</td>
<td>Emulation mode</td>
<td>134</td>
</tr>
</tbody>
</table>

**Appendix - <format> Options of MMU.FORMAT**

Usage:

(B) command only available for ICD
(E) command only available for ICE
(F) command only available for FIRE
25-Aug-15  Reworked and updated the **MCDS** command group.

20-Apr-15  Updated the description of **MMU.FORMAT**.

27-Jan-15  Revised the **MCDS** command group.

31-Jul-14  Added descriptions for **MAP.VMREAD** and **MAP.NoVMREAD**.

08-May-14  Updated **MMU.DUMP**, **MMU.SCAN**, and **MMU.Set**.

05-Mar-14  Reworked **MCDS.TraceBuffer** command group, removed description of Emulation Memory and referred to MCDS User's Guide.

### MAP

<table>
<thead>
<tr>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAP.Ack</strong></td>
</tr>
<tr>
<td><strong>MAP.BOnchip</strong></td>
</tr>
<tr>
<td><strong>MAP.BUS24</strong></td>
</tr>
<tr>
<td><strong>MAP.BYTE</strong></td>
</tr>
<tr>
<td><strong>MAP.CONST</strong></td>
</tr>
<tr>
<td><strong>MAP.DenyAccess</strong></td>
</tr>
<tr>
<td><strong>MAP.Extern</strong></td>
</tr>
<tr>
<td><strong>MAP.Intern</strong></td>
</tr>
<tr>
<td><strong>MAP.Mirror</strong></td>
</tr>
<tr>
<td><strong>MAP.NoAck</strong></td>
</tr>
<tr>
<td><strong>MAP.NoBreak</strong></td>
</tr>
<tr>
<td><strong>MAP.NOBUS8</strong></td>
</tr>
<tr>
<td><strong>MAP.NoData</strong></td>
</tr>
<tr>
<td><strong>MAP.NoDMUX</strong></td>
</tr>
<tr>
<td><strong>MAP.NoLittleEndian</strong></td>
</tr>
<tr>
<td><strong>MAP.NoPOOL</strong></td>
</tr>
<tr>
<td><strong>MAP.NOROM</strong></td>
</tr>
<tr>
<td><strong>MAP.NoUpdateOnce</strong></td>
</tr>
<tr>
<td><strong>MAP.NoXBUS</strong></td>
</tr>
<tr>
<td><strong>MAP.PRE</strong></td>
</tr>
<tr>
<td><strong>MAP.RELOCate</strong></td>
</tr>
<tr>
<td><strong>MAP.SOnchip</strong></td>
</tr>
<tr>
<td><strong>MAP.UpdateOnce</strong></td>
</tr>
<tr>
<td><strong>MAP.Wait</strong></td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAP.ADelay</strong></td>
</tr>
<tr>
<td><strong>MAP.Break</strong></td>
</tr>
<tr>
<td><strong>MAP.BUS32</strong></td>
</tr>
<tr>
<td><strong>MAP.Cache</strong></td>
</tr>
<tr>
<td><strong>MAP.CS</strong></td>
</tr>
<tr>
<td><strong>MAP.Data</strong></td>
</tr>
<tr>
<td><strong>MAP.DMA</strong></td>
</tr>
<tr>
<td><strong>MAP.Frag</strong></td>
</tr>
<tr>
<td><strong>MAP.Dump</strong></td>
</tr>
<tr>
<td><strong>MAP.Flash</strong></td>
</tr>
<tr>
<td><strong>MAP.Intr</strong></td>
</tr>
<tr>
<td><strong>MAP.List</strong></td>
</tr>
<tr>
<td><strong>MAP.Mode</strong></td>
</tr>
<tr>
<td><strong>MAP.Mirror</strong></td>
</tr>
<tr>
<td><strong>MAP.NoFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.NoBurst</strong></td>
</tr>
<tr>
<td><strong>MAP.NoCache</strong></td>
</tr>
<tr>
<td><strong>MAP.NoDenyAccess</strong></td>
</tr>
<tr>
<td><strong>MAP.NoFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.NoMFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.NoProtect</strong></td>
</tr>
<tr>
<td><strong>MAP.NoShadow</strong></td>
</tr>
<tr>
<td><strong>MAP.NoVMREAD</strong></td>
</tr>
<tr>
<td><strong>MAP.OPFetch</strong></td>
</tr>
<tr>
<td><strong>MAP.Protec</strong></td>
</tr>
<tr>
<td><strong>MAPRESET</strong></td>
</tr>
<tr>
<td><strong>MAP.Setup</strong></td>
</tr>
<tr>
<td><strong>MAP.Verify</strong></td>
</tr>
<tr>
<td><strong>MAP.Word</strong></td>
</tr>
</tbody>
</table>

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MAP.AFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.BURST</strong></td>
</tr>
<tr>
<td><strong>MAP.BUS8</strong></td>
</tr>
<tr>
<td><strong>MAP.CFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.Data</strong></td>
</tr>
<tr>
<td><strong>MAP.DMA</strong></td>
</tr>
<tr>
<td><strong>MAP.Err</strong></td>
</tr>
<tr>
<td><strong>MAP.GAP</strong></td>
</tr>
<tr>
<td><strong>MAP.GFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.HFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.H_MODE</strong></td>
</tr>
<tr>
<td><strong>MAP.IFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.IS</strong></td>
</tr>
<tr>
<td><strong>MAP.JFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.LFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.MFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.Mirror</strong></td>
</tr>
<tr>
<td><strong>MAP.OFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.PAGE</strong></td>
</tr>
<tr>
<td><strong>MAP.Pack</strong></td>
</tr>
<tr>
<td><strong>MAP.PROM</strong></td>
</tr>
<tr>
<td><strong>MAP.Read</strong></td>
</tr>
<tr>
<td><strong>MAP.ReadFlag</strong></td>
</tr>
<tr>
<td><strong>MAP.Reset</strong></td>
</tr>
<tr>
<td><strong>MAP.Sequence</strong></td>
</tr>
<tr>
<td><strong>MAP.Set</strong></td>
</tr>
<tr>
<td><strong>MAP.Split</strong></td>
</tr>
<tr>
<td><strong>MAP.VMREAD</strong></td>
</tr>
<tr>
<td><strong>MAP.Write</strong></td>
</tr>
<tr>
<td><strong>MAP.XBus</strong></td>
</tr>
</tbody>
</table>

©1989-2015 Lauterbach GmbH
The mapping system reproduces the emulation CPU's logical address range onto the available physical memory system and defines all address-dependent system parameters. These parameters include write-protection and optional wait states for certain address ranges. The TRACE32 system has 3 memory systems, one for data (emulation memory), one for breakpoints (break memory) and one for marking memory accesses (flag memory). Each memory group can be activated in 4 KByte blocks within the target system's address range. Memory mapping takes place in three steps:

1. Selecting the mode
2. Definition phase
3. Activation phase

Selecting Mode

The translation of the CPU's address onto the memory address needs time, which has normally to be added to the access time of the used memory. If this should be avoided, there must be no time difference between the CPU's and memory's address signal. This means, however, that the emulation memory may not be mapped onto the CPU address area without restriction. This is why you can select either a fast or a slow mapper mode. Switching over the different modes is made by the command MAP.Mode. If dynamic RAM is used, the memory may be mapped in any way because address conversion and memory access may occur simultaneously.

Definition Phase

During the definition phase the system receives information as to how the memory of the target system is organized. After power up (or MAP.RESet) it is assumed that no address mirroring exists and that the up to four different memory areas (DATA, PROGRAM, ...) use all the same memory.

If the target system has separate memory areas, the memory will be split up accordingly using the MAP.SPlit command. If the target system contains address mirrors these will be defined by using the MAP.MIrror command.

When the definition phase is completed, the memory mapping can be started. All definition commands are locked and can be reactivated by a MAP.RESet or MAP.NEW command only.

```plaintext
;-----------------------------------------------------------
; A23 in not decoded
MAP.MIrror sd:0x0--0x7fffffff 0x800000
;-----------------------------------------------------------
; use same memory for program and data area
MAP.MIrror sp:0x0--0x0fffff d:
```
Activation Phase

When the definition phase is completed, the available memory can be activated by using the following commands: MAP.Ram, MAP.Data, MAP.Break and MAP.Flag. The MAP.Break command will be normally not used since breakpoint memory will be automatically activated by setting a breakpoint. Memory mapping takes place in blocks of 4 KBytes. Please note that this is not identical to the smallest available boundary. The MAP.NoRam, MAP.NoData, MAP.NoBreak or MAP.NoFlag commands are used for deactivation.

Premapper

The structure of the mapper is designed for CPUs with 24 address lines at maximum. If greater address size exists this has to be converted to a 24-bit address range. This is made by defining so-called work benches. There are 16 work benches available with a size of 1 MByte. Solely within these areas the mapping of emulation, breakpoint or flag memory is possible. Global breakpoints and waits which are placed at a MByte boundary, however, can be set.

Fine Mapping

Whereas emulation memory is fundamentally divided into 4 KByte blocks, write protection, internal and external emulation memory allocation and additional wait state activation can be set with byte boundaries. Fine mapping is transparent to the user. However, only two different 16 (32) KB blocks can be set with byte boundaries.

Internal/External Mapping

In addition to, and independent of how memory has been split, the memory area from which the emulation CPU will access its data is defined separately. If this memory area is defined by using the MAP.Ram or MAP.Data commands, the emulation CPU will access both memory systems within this address range. Note that write accesses occur to both memories. When reading from memory, either external memory or emulation memory is used, depending upon which memory was selected by the MAP.Intern or MAP.Extern commands.
Write Protection

For replacing EPROM memory by emulation memory, every address range within the emulation memory can be write-protected by the `MAP.Protect` and `MAP.NoProtect` commands.

Processor Specific Memory Attributes

For some processors additional attributes can be mapped. These attributes can define the access type, (like `OPFetch`) or the memory bus size (`BUS8`, `BUS16`, `BUS32`). For details, see the Emulation Target Manual of your processor.

Wait States

If system specifications require wait states for memory access these will be set in the required address range using the `MAP.Wait` command.
The monitor/EPROM-simulator can support two 8-bit or one 16-bit EPROM. The combination of several modules allows 32- and 64-bit configuration to be supported.

During the simulation the EPROM configuration of the target system is imitated by software. Using this technique paged and banked EPROM's can be simulated.

The imitation of the EPROM configuration is done as follows:

1. Reset the mapping system (**MAP.RESet** command).
2. Map the EPROM simulator within the specified range (**MAP.ROM** command).
3. Set the EPROM bus size (**MAP.BUSXX** command). The default bus size is 8 bit.
4. Set the EPROM width (**MAP.BYTE** or **MAP.WORD** command). By default an 8 bit organized EPROM is assumed.

```
;--------------------------------------------------
; maps one 8K x 8 EPROM
; 8 bit adapter low
b:
MAP.RESet
MAP.ROM 0x0--0x01fff

;--------------------------------------------------
; maps two 8K x 8 EPROMS in parallel
; 8 bit adapter low and high
b:
MAP.RESet
MAP.ROM 0x0--0x03fff
MAP.BUS16 0x0--0x03fff

;--------------------------------------------------
; maps one 4K x 16 EPROM
; 16 bit adapter
b:
MAP.RESet
MAP.ROM 0x0--0x01fff
MAP.BUS16 0x0--0x01fff
MAP.WORD 0x0--0x01fff
```
; maps one paged addressed EPROM with 4 pages (4 x 16K x 8)
; 8 bit adapter low
b:
MAP.RESet

MAP.ROM 0x00000--0x03fff
MAP.ROM 0x04000--0x07fff
MAP.ROM 0x08000--0x0bfff
MAP.ROM 0x0c000--0x0ffff

MAP.PAGE 0 0x00000--0x03fff
MAP.PAGE 1 0x04000--0x07fff
MAP.PAGE 2 0x08000--0x0bfff
MAP.PAGE 3 0x0c000--0x0ffff

; maps two fragments in one 8 bit EPROM
; 8 bit adapter low
b:
MAP.ROM 0x00--0x7fff
MAP.ROM 0x10000--0x17fff
MAP.FRAG 1 0 0x00--0x7fff
MAP.FRAG 1 8000 0x10000--0x17fff

; relocates one 128K x 8 EPROM mapped from 0x00--0x1ffff to 0x40000
; while the system is up
b:
MAP.RELOCate 0x40000 0x00--0x1ffff

; maps four 64K x 8 EPROMs for a bus size of 32 bit
; two EPROM simulators
; for each 8 bit adapter high and low
MAP.ROM 0x00--0x3ffff
MAP.BUS32 0x00--0x3ffff

; maps two 64K x 16 EPROMs for a bus size of 32 bit
; two EPROM simulators
; for each 16 bit adapter
MAP.ROM 0x00--0x3ffff
MAP.BUS32 0x00--0x3ffff
MAP.WORD
### MAP.Ack

**ICE only**

**Generate acknowledge signals**

In the specified range the bus acknowledge lines (DTACK, DSACK, READY) are forced by the emulator system. As default the whole address range is active.

| Format: | MAP.Ack [\<address\> | \<addressrange\>] |
|---------|----------------------|

```plaintext
; the system runs with 3 wait states independent of the target hardware
MAP.Ack 0x0--0x0ffff ; set DSACK generation from 0 to 0FFFFH
MAP.Wait 3 0x0--0x0ffff ; set wait states
```

**See also**
- MAP.state
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
- ‘Mapper’ in ‘ICE User’s Guide’

### MAP.ADelay

**Debugger / FIRE only**

**tbd.**

**MAP.ADelay**

**Format:** MAP.ADelay [\<address\> | \<addressrange\>]

**See also**
- MAP.state

### MAP.AFlag

**ICE only**

**tbd.**

**MAP.AFlag**

**Format:** MAP.AFlag [\<address\> | \<addressrange\>]

**See also**
- MAP.state
Set bank range

**MAP.BANK**

*ICE only*

**Format:**

```
MAP.BANK <addressrange>
```

This function is required only by 8-bit emulators and is used in emulating 8-bit banked EPROMs. The mapping is mirrored to every 64K page. For further informations see *emulation probe manuals* chapter *Banked Target Systems*.

```
SYStem.BankFile test ; load bank file
MAP.BANK 0x4000--0x7fff ; banked EPROM is from 16 K to 32 K
```

**See also**

- MAP.NOBANK
- MAP.state

---

**MAP.BOnchip**

*Debugger / FIRE only*

**Format:**

```
MAP.BOnchip <addressrange>
```

This definition will be used for setting breakpoints. Any breakpoints that touch the defined area will be implemented using on-chip resources. This allows program breakpoints in read only memories or data breakpoints that also consider CPU internal operations. The capabilities of the on-chip breakpoints are CPU dependent.

**See also**

- MAP.state
**MAP.Break**

Map break memory

ICE only

<table>
<thead>
<tr>
<th>Format:</th>
<th>MAP.Break &lt;address&gt;</th>
<th>&lt;addressrange&gt; [/&lt;option&gt;]</th>
</tr>
</thead>
</table>

**<option>:**

- Static
- Dynamic
- Any

Only break memory is mapped within the specified range. The default option is **ANY**.

**See also**

- MAP.NoBreak
- MAP.state
- MAP.RAMSIZE()
- MAP.ROMSIZE()

▲ ‘Mapper’ in ‘ICE User’s Guide’
▲ ‘Starting-up the Emulator’ in ‘Training ICE Basics’

---

**MAP.BURST**

Burst area mapping

<table>
<thead>
<tr>
<th>Format:</th>
<th>MAP.BURST [&lt;addressrange&gt;]</th>
</tr>
</thead>
</table>

Defines burstable areas for the CPU.

**See also**

- MAP.NoBURST
- MAP.state

▲ ‘Mapping’ in ‘ICE Emulator for MC68040/60’
# The Usage of the MAP.BUSx Commands

The command **MAP.BUSx** constrains the debugger to read/write data in the specified access width.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP.BUS8 0x0++1FFFFF</td>
<td>constrain the debugger to 8-bit reads/writes</td>
</tr>
<tr>
<td>Data.dump 0x1000 /Long</td>
<td>display a memory dump in 32-bit format</td>
</tr>
<tr>
<td>Data.Set 0x300 %Long 0xAAAAAAAA</td>
<td>write 32-bit data to memory</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP.BUS32 0x060000000++1FFFFF</td>
<td>constrain the debugger to 32-bit reads/writes</td>
</tr>
<tr>
<td>Data.dump 0x06000003C /Byte</td>
<td>display a memory dump in 8-bit format</td>
</tr>
<tr>
<td>Data.Set 0x0600007A9 %Byte 0xAA</td>
<td>write 8-bit data to memory</td>
</tr>
</tbody>
</table>

The command **MAP.BUS32** doesn’t affect the access width for patching the software breakpoint code. Therefore the option **SYStem.Option SOFTLONG** is still required to patch the breakpoint code 32-bit wise.
The TRACE32 EPROM Simulator (ESI) has to model the memory organization in order to simulate the EEPROM/FLASH. Part of this model is the bus width between the CPU and the EEPROM/FLASH.

The command `MAP.BUSx` is used to specify this bus width.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP.ROM 0x0--0x1FFF</td>
<td>; specify the address range of the simulated EEPROM/FLASH</td>
</tr>
<tr>
<td>MAP.BYTE</td>
<td>; inform TRACE32 that an 8-bit adapter is connected to the ESI (default)</td>
</tr>
<tr>
<td>MAP.BUS8</td>
<td>; inform TRACE32 that the bus width between the CPU and the EEPROM/FLASH is 8</td>
</tr>
<tr>
<td></td>
<td>; (default)</td>
</tr>
<tr>
<td>MAP.ROM 0x0--0x3FFF</td>
<td>; specify the address range of the simulated EEPROM/FLASH</td>
</tr>
<tr>
<td>MAP.BYTE</td>
<td>; inform TRACE32 that 2 8-bit adapters are connected to the ESI (default)</td>
</tr>
<tr>
<td>MAP.BUS16</td>
<td>; inform TRACE32 that the bus width between the CPU and the EEPROM/FLASH is 16</td>
</tr>
<tr>
<td>MAP.ROM 0x0--0x1FFF</td>
<td>; specify the address range of the simulated EEPROM/FLASH</td>
</tr>
<tr>
<td>MAP.WORD</td>
<td>; inform TRACE32 that an 16-bit adapter is connected to the ESI</td>
</tr>
<tr>
<td>MAP.BUS16</td>
<td>; inform TRACE32 that the bus width between the CPU and the EEPROM/FLASH is 16</td>
</tr>
</tbody>
</table>
The in-circuit emulator TRACE32-ICE has to model the memory organization for the target memory block that are replaced by emulation memory. Part of this model is the bus width between the CPU and the memory blocks.

The command **MAP.BUSx** is used to specify this bus width.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAP.Ram 0x0--0xFFFFF</td>
<td>allocate emulation memory for the specified address range</td>
</tr>
<tr>
<td>MAP.Intern 0x0--0xFFFFF</td>
<td>activate emulation memory for the specified address range</td>
</tr>
<tr>
<td>MAP.BUS8 0x0--0xFFFFF</td>
<td>the bus width between the CPU and the specified address range is 8</td>
</tr>
</tbody>
</table>

**MAP.BUS8**

**Bus width mapping**

**Format:**

```
MAP.BUS8 [addressrange]
```

<table>
<thead>
<tr>
<th>Debugger</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Constrain TRACE32 to 8-bit reads/writes for the specified target memory block.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ESI, TRACE32-ICE</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Inform TRACE32 that the bus width between CPU and specified memory block is 8 bit.</td>
</tr>
</tbody>
</table>

**See also**

- MAPBUS16
- MAPBUS24
- MAPBUS32
- MAPBUSEXT
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
- ‘Mapping’ in ‘ICE Emulator for MC68000 and MC6830X’
- ‘Mapping’ in ‘ICE Emulator for MC6833X’
- ‘Mapping’ in ‘ICE Emulator for Motorola 68360/349’
- ‘General SYStem Settings and Restrictions’ in ‘ICE Emulator for 90CL301’
- ‘General SYStem Settings and Restrictions’ in ‘ICE Emulator for C166/ST10’
- ‘Mapping’ in ‘ICE Emulator for 386/486’
- ‘Mapping’ in ‘ICE Emulator for PowerPC’
- ‘Special Settings 186ES, 188ES, 186ED and Restrictions’ in ‘ICE Emulator for the 80186 and 80196’
- ‘Release Information’ in ‘Release History’
- ‘Starting-up the Emulator’ in ‘Training ICE Basics’
### MAP.BUS16

**Format:**

```
MAP.BUS16 [<addressrange>]
```

<table>
<thead>
<tr>
<th>Debugger</th>
<th>Constrain TRACE32 to 16-bit reads/writes for the specified target memory block.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESI, TRACE32-ICE</td>
<td>Inform TRACE32 that the bus width between CPU and specified memory block is 16 bit.</td>
</tr>
</tbody>
</table>

**See also**

- MAP.BUS24
- MAP.BUS32
- MAP.BUS8
- MAP.BUSEXT
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
- ‘Mapping’ in ‘ICE Emulator for Motorola 68360/349’
- ‘Mapping’ in ‘ICE Emulator for 386/486’
- ‘Mapping’ in ‘ICE Emulator for PowerPC’
- ‘Release Information’ in ‘Release History’

### MAP.BUS24

**Format:**

```
MAP.BUS24 [<addressrange>]
```

<table>
<thead>
<tr>
<th>Debugger</th>
<th>Constrain TRACE32 to 24-bit reads/writes for the specified target memory block.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESI, TRACE32-ICE</td>
<td>Inform TRACE32 that the bus width between CPU and specified memory block is 24 bit.</td>
</tr>
</tbody>
</table>

**See also**

- MAP.BUS16
- MAP.BUS32
- MAP.BUS8
- MAP.BUSEXT
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
### MAP.BUS32

**Bus width mapping**

<table>
<thead>
<tr>
<th>Debugger</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Constrain TRACE32 to 32-bit reads/writes for the specified target memory block.</td>
<td></td>
</tr>
<tr>
<td><strong>ESI, TRACE32-ICE</strong></td>
<td>Inform TRACE32 that the bus width between CPU and specified memory block is 32 bit.</td>
</tr>
</tbody>
</table>

**See also**
- MAP.BUS16
- MAP.BUS24
- MAP.BUS8
- MAP.BUSEXT
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
- ‘Mapping’ in ‘ICE Emulator for Motorola 68360/349’
- ‘Mapping’ in ‘ICE Emulator for 386/486’
- ‘Mapping’ in ‘ICE Emulator for PowerPC’
- ‘Release Information’ in ‘Release History’

### MAP.BUSEXT

**External bus mapping**

| Format: MAP.BUSEXT [<addressrange>] |

**See also**
- MAP.BUS16
- MAP.BUS24
- MAP.BUS32
- MAP.BUS8
- MAP.state
- ‘Mapping’ in ‘ICE Emulator for Motorola 68360/349’
- ‘Mapping’ in ‘ICE Emulator for 386/486’
- ‘Mapping’ in ‘ICE Emulator for PowerPC’
### MAP.BYTE
**Set EPROM width**

Debugger only

**Format:**

```
MAP.BYTE [<addressrange>]
```

The EPROM is organized by 8 bits per word in the specified range.

**See also**

- MAP.state
- MAP.WORD
- 'Mapping' in 'EPROM/FLASH Simulator'

### MAP.Cache
**Cache area mapping**

**Format:**

```
MAP.Cache [<addressrange>]
```

Defines cachable areas for the CPU.

**See also**

- MAP.state
- 'Mapping' in 'ICE Emulator for MC68040/60'
- 'Mapping' in 'ICE Emulator for 386/486'

### MAP.CFlag

**tbd.**

**Format:**

```
MAP.CFlag [<range>]
```

tbd.

**See also**

- MAP.state
MAP.COMSTART

Format: MAP.COMSTART [<range>]

tbd.

See also
- MAP.state

MAP.CONST

Mapped address range contains constants

Format: MAP.CONST [<range>]

The defined address range contains constants. The address range for the constants can be declared in two ways:

- The compiler provides a constant section

  MAP.CONST y.secrange(\sdata2) ; map the section \sdata2 as ; address range for constants

- The constants are merged into the code

  MAP.CONST 0x0--0x3ffff ;map the address range of a FLASH ;as address range for constants

This command is closely related to the command CTS.UseConst.

See also
- MAP.NoCONST
- MAP.state
See also

- MAP.NOCs
- MAP.state

---

### MAP.CS

Format: `MAP.CS [<range>]`

---

### MAP.Data

**ICE only**

Map data memory

Format: `MAP.Data <address> | <addressrange> [/<option>]`

- `<option>`:
  - Static
  - Dynamic
  - Any

Only data memory is mapped within the specified range.
The option specifies the emulation memory type. The default option is **ANY**.

See also

- MAP.NoData
- MAP.state
- MAP::RAMSIZE()
- MAP::ROMSIZE()
ICE only

---

**MAP.DEFault**

All available memory is mapped starting at address 0 and switched to the internal memory. A given address is interpreted as a start address. The restriction to a certain address range or to a certain memory type is possible. The default memory type is **Any**.

- **Static**: Only static memory is used for mapping.
- **Dynamic**: Only dynamic memory is used for mapping.
- **Any**: Both, static and dynamic memory are used for mapping.

**Format:**

```
MAP.DEFault [<address> | <addressrange>] [/<option>]
```

- **<option>:**
  - Static
  - Dynamic
  - Any

**Examples:**

```
MAP.DEFault ; map all available memory from address 0x0
MAP.DEFault 0x0fff0000 ; map all available memory from address 0x0FFF0000H
MAP.DEFault 0x0--0xffff ; map memory and set internal in first 64K page
```

**See also**

- MAP.state
- MAP.RAMSIZE()
- MAP.ROMSIZE()

- 'Mapper' in 'ICE User's Guide'
- 'Starting-up the Emulator' in 'Training ICE Basics'
MAP.DenyAccess

Deny memory access by TRACE32

The TRACE32 software can’t access the specified target address range. This command can be used if accesses by the development tool to specific target memory address ranges cause problems (e.g. “emulation debug port fail”, “emulator berr error”).

The address ranges that can’t be accessed by the TRACE32 software can be displayed by the MAP.List command.

<table>
<thead>
<tr>
<th>address type</th>
<th>bus denyaccess attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>C:00000000--003EFFFF</td>
<td>denyaccess</td>
</tr>
<tr>
<td>C:003F0000--003FFFFF</td>
<td></td>
</tr>
<tr>
<td>C:00400000--FFFFFFF</td>
<td></td>
</tr>
</tbody>
</table>

MAP.DenyAccess is switched off by the command MAP.NoDenyAccess.

See also
- MAP.state

MAP.DenyBurst

Deny burst access to memory by TRACE32

Format: MAP.DenyBurst [<address> | <addressrange>]

MPC7441, MPC7445, MPC7447, MPC7447A, MPC7450, MPC7451, MPC7455 and MPC7457 only, because for these processors, memory is access is performed through 32-byte burst access per default.

In order to access peripherals which do not support burst accesses (e.g. register and flash address space of MV64xxx), use this command to prevent burst accesses. Please note that the non-burst memory access on these processors is very slow.

MAP.DenyBurst is switched off by the command MAP.NoDenyBurst

See also
- MAP.state
MAP.DMA

DMA area mapping

Format: MAP.DMA [<addressrange>]

See also
- MAP.state
- ‘Exception Control’ in ‘ICE Emulator for Z80 and Z180’

MAP.DMUX

Format: MAP.DMUX [<range>]

tbd.

See also
- MAP.NoDMUX
- MAP.state

MAP.Extern

External memory

ICE only

Format: MAP.Extern [<address> | <addressrange>]

The specified range is mapped externally, e.g. it will be read from the target system. Without an address value given by the user, the whole mapped data area is mapped external (target).

map.extern ; set all to external

See also
- MAP.Intern
- MAP.state
- ‘Starting-up the TRACE32-FIRE’ in ‘Training FIRE Basics’
- ‘Starting-up the Emulator’ in ‘Training ICE Basics’
### MAP.Flag

**Map flag memory**

ICE only

<table>
<thead>
<tr>
<th>Format:</th>
<th>MAP.Flag &lt;address&gt; &lt;addressrange&gt; [/&lt;option&gt;]</th>
</tr>
</thead>
</table>
| <option>: | Static  
| | Dynamic  
| | Any |

Only flag memory is mapped to the specified range. The option selects the RAM type to be used. The default option is **Any**.

**See also**
- MAP.NoFlag
- MAP.state
- MAP.RAMSIZE()
- MAP.ROMSIZE()
- 'Emulator Functions’ in ‘FIRE User’s Guide’
- ‘Mapper’ in ‘ICE User's Guide’
- ‘Starting-up the Emulator’ in ‘Training ICE Basics’

### MAP.FRAG

**Form fragment**

Debugger only

<table>
<thead>
<tr>
<th>Format:</th>
<th>MAP.FRAG &lt;frag&gt; &lt;address&gt; [&lt;addressrange&gt;]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;frag&gt;:</td>
<td>1..255.</td>
</tr>
</tbody>
</table>

Combines two ROM areas to a fragment. One fragment can be simulated by one EPROM simulator.

- MAP.ROM 0x0--0x7fff
- MAP.ROM 0x10000--0x17fff
- MAP.FRAG 1 0 0x0--0xffff
- MAP.FRAG 1 0x8000 0x10000--0x17fff

**See also**
- MAP.NOFRAG
- MAP.state
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
**MAP.GAP**

**Debugger only**

<table>
<thead>
<tr>
<th>Format:</th>
<th>MAP.GAP &lt;frag&gt; &lt;address&gt; [addressrange]</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;frag&gt;:</td>
<td>1..255.</td>
</tr>
</tbody>
</table>

Defines a gap in the ROM area simulated by an EPROM simulator. This could be useful e.g. if this area is used by internal peripherals.

```plaintext
MAP.FRAG 1 0 0x0--0x7aff
MAP.GAP 1 0x7b00-0xffff
MAP.FRAG 1 0x8000 0x8000--0x8fff
```

See also

- MAP.NOGAP
- MAP.state

▲ 'Mapping' in 'EPROM/FLASH Simulator'

**MAP.Intern**

**ICE only**

| Format: | MAP.Intern [address | addressrange] |
|---------|------------------|

The specified range is mapped internally, e.g. the emulation memory is used. Without an address value given by the user the whole data range is mapped to internal.

```plaintext
map.e ; map all to external
map.i p:0x0--0xffff ; map program area to internal
map.i io:0x33 ; map one byte in the IO area to internal
; for port simulation
```

See also

- MAP.Extern
- MAP.state

▲ 'Starting-up the Emulator’ in 'Training ICE Basics'
The display of the mapper configuration for the logical workspace. The information can be reduced by the options. With no option set everything is displayed, except the mapping of the breakpoint memory (option DEFault).

+--------------------------+-------------------------------------------------+
<table>
<thead>
<tr>
<th>&lt;option&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEFault</td>
<td>All information, except the breakpoint memory, is displayed.</td>
</tr>
<tr>
<td>ALL</td>
<td>All information is displayed.</td>
</tr>
<tr>
<td>Data, Break, Flag</td>
<td>Only the selected memory type is displayed.</td>
</tr>
<tr>
<td>Ack</td>
<td>The areas in which the bus cycles would be acknowledged are displayed.</td>
</tr>
<tr>
<td>BANK</td>
<td>The mapping of banked and non-banked memory is displayed.</td>
</tr>
<tr>
<td>BUS8, Onchip, DMA, OPFetch, ...</td>
<td>These are special mapper flags solely used for special emulation probes. For further informations see emulation probe manuals.</td>
</tr>
<tr>
<td>Intern</td>
<td>The internal/external mapping is displayed.</td>
</tr>
<tr>
<td>Protect</td>
<td>The memory-write protection is displayed.</td>
</tr>
<tr>
<td>Wait</td>
<td>The number of wait states is displayed.</td>
</tr>
</tbody>
</table>

The display of the mapper configuration for the logical workspace. The information can be reduced by the options. With no option set everything is displayed, except the mapping of the breakpoint memory (option DEFault).
Explanation of the abbreviations within the columns from the left to right:

A | Acknowledge signal will be generated
I | dataram is mapped Internal
P | memory-write Protection is turned on
B|8 | Banking, bus8 or Burst area
O|8|M|C | Onchip or Opfetch or bus8 or DMA or Cache area
N | Nocache area
B | Breakram is mapped
F | Flagram is mapped
D | Dataram is mapped
d | dynamic ram is mapped
s | static ram is mapped
1..250 | the number of wait states

MAP.List Window for BDM/ROM

<table>
<thead>
<tr>
<th>Address</th>
<th>UP</th>
<th>UD</th>
<th>SP</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000--0001FFF</td>
<td>IP BFDs 4</td>
<td>IP BFDs 4</td>
<td>IP BFDs 4</td>
<td>I</td>
</tr>
<tr>
<td>00002000--0003FFF</td>
<td>I BFDs 4</td>
<td>I BFDs 4</td>
<td>I BFDs 4</td>
<td>I</td>
</tr>
<tr>
<td>00004000--0007FFF</td>
<td>I BFDs 2</td>
<td>I BFDs 2</td>
<td>I BFDs 2</td>
<td>I</td>
</tr>
<tr>
<td>00008000--000EFFF</td>
<td>BFDs 2</td>
<td>BFDs 2</td>
<td>BFDs 2</td>
<td>I</td>
</tr>
<tr>
<td>0000F000--000FFFFE</td>
<td>BFDs</td>
<td>BFDs</td>
<td>BFDs</td>
<td>I</td>
</tr>
<tr>
<td>00000FFF--000FFFF</td>
<td>I BFDs 30</td>
<td>I BFDs 30</td>
<td>I BFDs 30</td>
<td>I</td>
</tr>
<tr>
<td>00010000--00FEFFFF</td>
<td>BFDs</td>
<td>BFDs</td>
<td>BFDs</td>
<td>I</td>
</tr>
<tr>
<td>00FF0000--00FFFFF</td>
<td>BFDs</td>
<td>BFDs</td>
<td>BFDs</td>
<td>I</td>
</tr>
</tbody>
</table>

See also
- MAP.state
- MAP.RAMSIZE()
- MAP.ROMSIZE()
- 'Mapping' in 'EPROM/FLASH Simulator'
- 'Emulator Functions' in 'FIRE User's Guide'
- 'Mapper' in 'ICE User's Guide'
**MAP.LittleEndian**

**Debugger only**

| Format: | `MAP.LittleEndian [<range>]` |

Defines the memory address area where the variable value display is switched to little endian word memory interpretation.

**See also**
- `MAP.state`

---

**MAP.MFlag**

**ICE only**

| Format: | `MAP.MFlag [<address> | <addressrange>]` |

**tbd.**

**See also**
- `MAP.state`
**MAP.Mirror**

**Mirroring**

ICE only

| Format: | MAP.Mirror <addressrange> <address> |

This command will be used if the target system contains address mirrors or banked areas. For further informations see [mapper definition phase](#) and [emulation probe manuals](#) chapter Banked Target Systems. The memory access class of the range must be unique.

```plaintext
MAP.Mirror sd:0x0--0x3fff 0x4000 ; area with 16 KByte is mirrored ; to 4000 HEX
MAP.Mirror d:0x0--0x0ffff io: ; IO and memory is accessed in the ; same way
MAP.Mirror io:0x0--0x7fff io:0x8000 ; IO area is not fully decoded
```

See also
- MAP.state
- ‘Mapper’ in ‘ICE User's Guide’

**MAP.Mode**

**Mode**

ICE only

| Format: | MAP.Mode SLOW | FAST |

When switching the mapper mode to FAST, the CPU can work at higher frequencies with the same number of wait states. The only disadvantage of this mode are the restrictions on the mapping of emulation memory.

```plaintext
map.res ; initialization
map.m s ; select slow mapper mode
```

See also
- MAP.state
- ‘Mapper’ in ‘ICE User's Guide’
**MAP.MONITOR**

Format: **MAP.MONITOR [<range>]**

tbd.

See also
- **MAP.state**
- 'Commands' in 'C166 Family Trace'

**MAP.NEW**

Initialization

ICE only

Format: **MAP.NEW**

Same as **MAP.RESet**, but operation mode and setups from the definition phase stay valid.

```
MAP.RESet                   ; reset
MAP.Split p:0x0--0x0ffff d:0 ; define splitting
MAP.Data p:0x0--0x0ffff
...
...
MAP.NEW                     ; start again, but already splitted
    MAP.Data ...
...
```

See also
- **MAP.RESet**
- **MAP.state**
- 'Mapper' in 'ICE User's Guide'
MAP.NoAck

ICE only

Disable acknowledge signals

| Format: | MAP.NoAck [<address> | <addressrange>] |

The acknowledge signals (DSACK, DTACK, READY, …) generated by the emulator system are disabled.

See also:
- MAP.state
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
- ‘Mapper’ in ‘ICE User's Guide’

MAP.NoAFlag

ICE only

tbd.

| Format: | MAP.NoAFlag [<address> | <addressrange>] |

See also:
- MAP.state

MAP.NOBANK

ICE only

Release bank area

| Format: | MAP.NOBANK |

This function is required by 8-bit emulators only, for emulating 8-bit banked EPROMs. For further informations see emulation probe manuals chapter Banked Target Systems.

See also:
- MAP.BANK
- MAP.state
MAP.NoBOnchip

Use on-chip breakpoints

Format:  MAP.NoBOnchip [<range>]

Undo the settings done by using the MAP.BOnchip command.

See also
- MAP.state

MAP.NoBreak

Release break memory

ICE only

Format:  MAP.NoBreak [<addressrange>]

By this command all break RAM in the specified address range is released.

See also
- MAP.Break
- MAP.state
- ‘Mapper’ in ‘ICE User’s Guide’
- ‘Starting-up the Emulator’ in ‘Training ICE Basics’

MAP.NoBURST

Burst area mapping

Format:  MAP.NoBURST [<range>]

Undo the settings done by using the MAP.BURST command.

See also
- MAP.BURST
- MAP.state
**MAP.NOBUS16**  
**Bus width mapping**

```
Format:  MAP.NOBUS16 [<addressrange>]
```

Undo the settings done by using the **MAP.BUS16** command.

**See also**
- MAP.state

**MAP.NOBUS32**  
**Bus width mapping**

```
Format:  MAP.NOBUS32 [<addressrange>]
```

Undo the settings done by using the **MAP.BUS32** command.

**See also**
- MAP.state

**MAP.NOBUS8**  
**Bus width mapping**

```
Format:  MAP.NOBUS8 [<addressrange>]
```

Undo the settings done by using the **MAP.BUS8** command.

**See also**
- MAP.state
- 'Mapping' in 'ICE Emulator for MC68000 and MC6830X'
- 'Mapping' in 'ICE Emulator for MC6833X'
- 'General SYStem Settings and Restrictions' in 'ICE Emulator for 90CL301'
- 'General SYStem Settings and Restrictions' in 'ICE Emulator for C166/ST10'
- 'Special Settings 186ES, 188ES, 186ED and Restrictions’ in 'ICE Emulator for the 80186 and 80196'
MAP.NoCache

Defines not cachable areas for the CPU.

See also
- MAP.state
- "Mapping" in 'ICE Emulator for MC68040/60'
- "Mapping" in 'ICE Emulator for 386/486'

MAP.NoCONST

Undo the MAP.CONST settings

Format: MAP.NoCONST [<range>]

Undo the settings done by using the MAP.CONST command.

See also
- MAP.CONST
- MAP.state

MAP.NOCS

tbd.

Format: MAP.NOCS [<range>]

Undo the settings done by using the MAP.CS command.

See also
- MAP.CS
- MAP.state
**MAP.NoData**  
 ICE only

Release data memory

This command releases all data RAM (emulation) in the specified address range. The default address range is the whole address space of the CPU.

See also
- MAP.Data
- MAP.state
- ‘Emulator Functions’ in ‘FIRE User's Guide’
- ‘Mapper’ in ‘ICE User's Guide’

**Format:**

```
MAP.NoData [<addressrange>]
```

**MAP.NoDenyAccess**  
 Debugger / FIRE only

Switch off deny access for TRACE32

This command switches off the deny access for the TRACE32 software that was specified for a specific target memory range.

See also
- MAP.state

**Format:**

```
MAP.NoDenyAccess [<addressrange>]
```

**MAP.NoDenyBurst**  
 ICE only

Undo settings of MAP.DENYBURST command

Undo the settings done by using the MAP.DenyBurst command.

See also
- MAP.state
MAP.NODMA

DMA area mapping

Undo the settings done by using the MAP.DMA command.

See also
- MAP.state
- ‘Exception Control’ in ‘ICE Emulator for Z80 and Z180’

MAP.NoDMUX

tbd.

Undo the settings done by using the MAP.DMUX command.

See also
- MAP.DMUX
- MAP.state

MAP.NoFlag

Release flag memory

ICE only

This command releases all flag RAM in the specified address range. The default address range is the whole address space of the CPU.

See also
- MAP.Flag
- MAP.state
- ‘Emulator Functions’ in ‘FIRE User's Guide’
- ‘Mapper’ in ‘ICE User's Guide’
- ‘Starting-up the TRACE32-FIRE’ in ‘Training FIRE Basics’
MAP.NOFrag

Debugger only

Switch off fragmentation

Format: `MAP.NOFrag [<addressrange>]`

By this command the fragmentation in the specified range is switched off.

See also

- MAP.FRAG
- MAP.state
- 'Mapping' in 'EPROM/FLASH Simulator'

MAP.NOGAP

Debugger only

Switch off gap

Format: `MAP.NOGAP [<addressrange>]`

By this command the gap in the specified range is switched off.

See also

- MAP.GAP
- MAP.state
- 'Mapping' in 'EPROM/FLASH Simulator'

MAP.NoLittleEndian

Debugger only

Switch off little endian

Format: `MAP.NoLittleEndian [<range>]`

Undo the settings done by using the MAP.LittleEndian command.

See also

- MAP.state
MAP.NoMFlag

ICE only

Format: MAP.NoMFlag [<address> | <addressrange>]

See also
- MAP.state

MAP.NoOPFetch

Switch off opfetch area mapping

Format: MAP.NoOPFetch [<addressrange>]

Undo the settings done by using the MAP.OPFetch command.

See also
- MAP.state
- 'Mapping’ in 'ICE Emulator for MC68000 and MC6830X’
- 'General SYStem Settings and Restrictions’ in 'ICE Emulator for C166/ST10’
- 'General System Settings and Restrictions’ in 'ICE Emulator for Hitachi H8/300 and H8/500’
- 'SYStem Settings’ in 'ICE Emulator for MELPS 7700’

MAP.NOPAGE

Debugger only

Define pages

Format: MAP.NOPAGE <addressrange>

By this command the page in the specified range is switch off.

See also
- MAPPAGE
- MAP.state
- 'Mapping’ in 'EPROM/FLASH Simulator’
**MAP.NoPOOL**

undo setting of MAP.POOL command

Format: `MAP.NoPOOL [<range>]`

Undo the settings of the `MAP.POOL` command.

See also
- MAP.POOL
- MAP.state

**MAP.NoProtect**

Erase write protection

ICE only

Format: `MAP.NoProtect [<address> | <addressrange>]`

In the specified range the write protection is switched off. Without an address value given by the user, the whole mapped data memory is no longer write protected.

See also
- MAP.Protect
- MAP.state
- ‘Emulator Functions’ in ‘FIRE User's Guide’
- ‘Mapper’ in ‘ICE User's Guide’

**MAP.NoRam**

Release RAM

ICE only

Format: `MAP.NoRam [<addressrange>]`

By this command all kinds of RAM in the specified address range are released. The default address range is the whole address space of the CPU.

See also
- MAP.Ram
- MAP.state
- ‘Emulator Functions’ in ‘FIRE User's Guide’
- ‘Mapper’ in ‘ICE User's Guide’

©1989-2015 Lauterbach GmbH

General Commands Reference Guide 42
**MAP.NoReadFlag**

*Undo setting of MAP.READFLAG command*

**FIRE / ICE only**

Format: `MAP.NoReadFlag [<address> | <addressrange>]`

Undo the settings of the **MAP.ReadFlag** command.

See also

- MAP.ReadFlag
- MAP.state

---

**MAP.NOROM**

*Unmap ESI*

**Debugger only**

Format: `MAP.NOROM [<addressrange>]`

By this command the EPROM simulator is unmapped in the specified range.

See also

- MAP.ROM
- MAP.state
- ‘Mapping’ in ‘EPROM/FLASH Simulator’

---

**MAP.NoShadow**

*Undo setting of MAP.SHADOW command*

**ICE only**

Format: `MAP.NoShadow [<address> | <addressrange>]`

Undo the settings of the **MAP.SHADOW** command.

See also

- MAP.state
**MAP.NoSOnchip**

Undo setting of MAP.SONCHIP command

Debugger only

Format: `MAP.NoSOnchip [address] | [addressrange]`

Undo the settings of the **MAP.SOnchip** command.

See also
- MAP.state

---

**MAP.NOSWAP**

Keep byte order

Debugger only

Format: `MAP.NOSWAP [addressrange]`

No changes are made to the byte order determined by the target CPU.

See also
- MAP.state
- MAP.SWAP
- ‘Mapping’ in ‘EPROM/FLASH Simulator’

---

**MAP.NoUpdateOnce**

Undo setting of MAP.UPDATEONCE command

Debugger only

Format: `MAP.NoUpdateOnce [address] | [addressrange]`

Undo the settings of the **MAP.UpdateOnce** command.

See also
- MAP.state
MAP.NoVMREAD  Undo the setting of the MAP.VMREAD command

Format:  MAP.NoVMREAD [<address_range>]

Undoes the setting of the MAP.VMREAD command.

See also  ■ MAP.state  ■ MAP.VMREAD

MAP.NoVOLATILE  Undo the setting of the MAP.VOLATILE command

Format:  MAP.NoVOLATILE [<range>]

Undo the settings of the MAP.VOLATILE command.

See also  ■ MAP.state  ■ MAP.VOLATILE

MAP.NOWB  Premapper settings

ICE only

Format:  MAP.NOWB [<range>]

Removes all with MAP.PRE defined 1 MB workbenches. This increase the internal mapper speed for higher CPU frequencies.

See also  ■ MAP.state
**MAP.NoXBus**
Maps XBUS to XPER device

FIRE only

Format:

```
MAP.NoXBus [<address> | <addressrange>]
```

See also
- MAP.state
- 'XPER Mapping and Shadowing' in 'FIRE Emulator for C166 Family'

---

**MAP.OPFetch**
Opfetch area mapping

Format:

```
MAP.OPFetch [<addressrange>]
```

See also
- MAP.state
- 'Mapping' in 'ICE Emulator for MC68000 and MC6830X'
- 'General SYStem Settings and Restrictions' in 'ICE Emulator for C166/ST10'
- 'General System Settings and Restrictions' in 'ICE Emulator for Hitachi H8/300 and H8/500'
- 'SYStem Settings' in 'ICE Emulator for MELPS 7700'
- 'Starting-up the Emulator' in 'Training ICE Basics'
MAP.PAGE

Debugger only

This command is used for paged EPROMs. The appropriate page is set by the value in the page register of the EPROM.

Format: MAP.PAGE <page_number> <address_range>

See also

- MAP.NOPAGE
- MAP.state

See also

- MAP.NOPAGE
- MAP.state

MAP.POOL

tbd.

Format: MAP.POOL [<range>]

tbd.

See also

- MAP.NOPool
- MAP.state
### MAP.PRE

**Premapper**

ICE only

The command must be used defining workbenches in 32-bit applications. It can be used several times until all 16 workbenches of 1 Mbyte are defined. Prior to this command the mapper should be initialized with `MAP.RESet` command.

Without an address range the command generates a list of the actually defined workbenches.

For further information see [mapper definition phase](#).

```
map.pre                        ; list workbench areas
map.pre 0x0fff00000--0x0ffffffff ; workbench in top of memory
```

**See also**

- `MAP.RESet`
- `MAP.state`
- ‘Mapper’ in ‘ICE User’s Guide’

---

### MAP.Protect

**Write protection**

ICE only

The specified range is write protected. Thus, ROM can be simulated. Without an address value given by the user, the whole mapped dataram is write protected. Protection is only activated at realtime emulation. Data set and loading of programs is not prevented. Protection is for emulation memory only. Target memory can not be write protected.

```
map.data p:0x0--0x0ffff        ; map memory to program area
map.p   p:0x0--0x0ffff         ; protect against write
```

**See also**

- `MAP.NoProtect`
- `MAP.state`
- ‘Emulator Functions’ in ‘FIRE User’s Guide’
- ‘Mapper’ in ‘ICE User's Guide’
- ‘Release Information’ in ‘Release History’
- ‘Starting-up the Emulator’ in ‘Training ICE Basics’
MAP.Ram

ICE only

Format: MAP.Ram <address> | <addressrange> [/<option>]

<option>:
Static
Dynamic
Any

Data, break and flag memory are mapped within the specified range. The default option is ANY.

map.ram p:0x0--0x0ffff ; map to program area
map.i p:0x0--0x0ffff ; map internal
map.p p:0x0--0x0ffff ; protect (EPROM)

See also
- MAP.NoRam
- MAP.state
- ‘Emulator Functions’ in ‘FIRE User’s Guide’
- ‘Mapper’ in ‘ICE User’s Guide’
- ‘Starting-up the TRACE32-FIRE’ in ‘Training FIRE Basics’
- ‘Starting-up the Emulator’ in ‘Training ICE Basics’

MAP.ReadFlag

Format: MAP.ReadFlag [<range>]

tbd.

See also
- MAP.NoReadFlag
- MAP.state
- ‘Starting-up the TRACE32-FIRE’ in ‘Training FIRE Basics’
MAP.RELOCate  
Relocate ROM area

Debugger only

| Format: | MAP.RELOCate <address> <addressrange> |

Relocates the ROM area while the system is up.

See also
- MAP.state
- 'Mapping’ in 'EPROM/FLASH Simulator’

MAP.RESet  
Reset

| Format: | MAP.RESet |

(E) The mapping system is reset to its default state. No memory is mapped. The CPU has access to the external memory in the target.

(B) The mapping system is reset to its default state. The EPROM simulator is switched off.

See also
- MAP.NEW
- MAP:PRE
- MAP.state
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
- ‘Emulator Functions’ in ‘FIRE User's Guide’
- ‘Mapper’ in ‘ICE User's Guide’
- ‘Starting-up the TRACE32-FIRE’ in ‘Training FIRE Basics’

MAP.ROM  
Map ESI

Debugger only

| Format: | MAP.ROM <addressrange> |

The EPROM Simulator is mapped within the specified range.

See also
- MAP.NOROM
- MAP:PRE
- MAP.state
- MAP.RAMSIZE()  
- MAP.ROMSIZE() 
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
MAP.Shadow

Format: MAP.Shadow [<address> | <addressrange>]

tbd.

See also
- MAP.state

MAP.SOnchip

Debugger only

Format: MAP.SOnchip [<address> | <addressrange>]

See also
- MAP.state

MAP.SPlit

Splitting

Format: MAP.SPlit <addressrange>

This function is used for target systems with separate memory areas.

For further informations see mapper definition phase.

map.split d:0x0--0x0ffff  ; 64 KByte data area is separated
map.ram  d:0x0--0x0ffff  ; map memory in data area
map.ram  p:0x0--0x0ffff  ; map memory in program area
map.i   0--0xffff        ; map internal both program and data
d.s   p:0x0 "Program"
d.s   d:0x0 "Data"
w.d   p:0x0             ; display program
w.d   d:0x0             ; display data

See also
- MAP.state
- ‘Mapper’ in ‘ICE User’s Guide’
Format: \texttt{MAP.state}

General view of the total available and the actually activated memory.
### MAP.state Window for TRACE32-ICE

<table>
<thead>
<tr>
<th>KByte</th>
<th>static 70 ns</th>
<th>dynamic 60 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Dataram</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>512</td>
<td>4096</td>
</tr>
<tr>
<td>used</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>free</td>
<td>512</td>
<td>4096</td>
</tr>
<tr>
<td><strong>Flagram</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>512</td>
<td>4096</td>
</tr>
<tr>
<td>used</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>free</td>
<td>512</td>
<td>4096</td>
</tr>
<tr>
<td><strong>Breakram</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>total</td>
<td>512</td>
<td>4096</td>
</tr>
<tr>
<td>used</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>free</td>
<td>512</td>
<td>4096</td>
</tr>
</tbody>
</table>

### MAP.state Window BDM/ROM

<table>
<thead>
<tr>
<th>KByte</th>
<th>static 70 ns</th>
<th>dynamic 60 ns</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>B::w.map</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESI1L 1MB 8-bit</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESI1H 1MB 8-bit</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

**See also**

- MAPAck
- MAPBOnchip
- MAPBUS24
- MAPBYTE
- MAPCONST
- MAPDenyAccess
- MAPExtern
- MAPIntern
- MAPMirror
- MAPNoAck
- MAPNoBreak
- MAPNOBUS8
- MAPNoData
- MAPNoDMUX
- MAPNoLittleEndian
- MAPNoPOOL
- MAPNoROM
- MAPNoUpdateOnce
- MAPNoXBus
- MAPPRE
- MAPRELOCate
- MAPSOnchip
- MAPVERIFY
- MAPWORD

▲ 'Mapping' in 'EPROM/FLASH Simulator'
▲ 'Emulator Functions' in 'FIRE User's Guide'
▲ 'Mapper' in 'ICE User's Guide'
▲ 'Starting-up the TRACE32-FIRE' in 'Training FIRE Basics'
▲ 'Starting-up the Emulator' in 'Training ICE Basics'
MAP.SWAP

Change byte order

Debugger only

Format: MAP.SWAP

This command changes the byte order from little endian to big endian or vice versa depending on the target CPU.

See also
- MAP:NOSWAP
- MAP.state
- 'Mapping' in 'EPROM/FLASH Simulator'

MAP.UpdateOnce

Read memory only once each time CPU stops

Debugger only

Format: MAP.UpdateOnce [address | addressrange]

Confirms the debugger to limit accesses to the specified memory address range to a single access per address. The debugger will store the accessed data in an internal buffer and will use the buffered data for all following accesses.

The data in the internal buffer will be discarded every time the CPU stops for the debugger, e.g. after single step, hitting a breakpoint or a manual break. Discarding the buffered data can be enforced by calling Data.UPDATE.

See also
- MAP.state

MAP.VERIFY

Internal usage only

Format: MAP.VERIFY

This command is meant for internal diagnostic usage only.

See also
- MAP.state

©1989-2015 Lauterbach GmbH

General Commands Reference Guide M 54 MAP
MAP.VMREAD Redirects memory reads/writes in the given address range to the TRACE32 virtual memory (VM:).

MAP.VMREAD can be used when target memory cannot be accessed (but is constant) or to avoid “unnecessary” memory reads to constant memory.

See also
- MAP.NoVMREAD
- MAP.state
- ‘Release Information’ in ‘Release History’

MAP.VOLATILE Mapped address range is volatile

Format: MAP.VOLATILE [<range>]

The defined memory range is not only changed by the processor core, this means that not all changes to this memory range are sampled to the trace buffer. E.g.: peripherals, dual-ported memory etc. Due to this attribute this memory range can not be used for CTS.

See also
- CTS.UseMemory
- MAP.NoVOLATILE
- MAP.state
**MAP.Wait**  
Wait cycles

**ICE only**

Format:  

\[
\text{MAP.Wait } \langle \text{waitcycles} \rangle \ [\langle \text{address} \rangle \mid \langle \text{addressrange} \rangle ]
\]

**<waitcycles>:**  
0. … 250.

Wait states are inserted by access to the specified range. Additional wait states will be generated by the target system.

\[
\begin{align*}
\text{MAP.Wait } 3 & \ p:0x0--0xffff \quad ; \text{map 3 wait states in program area} \\
\text{MAP.Wait } 3 & \ io:0x33 \quad ; \text{map wait states to one IO port} \\
\text{MAP.Wait } 3 & \ p:0x0 \quad \text{map.a p:0x0} \quad ; \text{map 3 wait states not influenced by the target hardware}
\end{align*}
\]

**See also**

- MAP.state
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
- ‘Mapper’ in ‘ICE User’s Guide’
- ‘Starting-up the Emulator’ in ‘Training ICE Basics’

**MAP.WORD**  
Set EPROM width

**Debugger only**

Format:  

\[
\text{MAP.WORD } \langle \text{addressrange} \rangle
\]

The EPROM is organized by 16 bits per word in the specified range.

**See also**

- MAP.BYTE
- MAP.state
- ‘Mapping’ in ‘EPROM/FLASH Simulator’
MAP.WriteFlag

Format: `MAP.WriteFlag [<range>]`

See also
- MAP.state
- ‘Starting-up the TRACE32-FIRE’ in ‘Training FIRE Basics’

MAP.XBus

Maps the XBUS to emulation memory

FIRE only

Format: `MAP.XBus <addressrange>`

If no XPER device is activated in this range, the XRAM can be supported by the standard emulation memory (ARAM) of the TRACE32-FIRE. The XPER chip selects must be activated, the external bus can stay in single chip or external bus mode.

```
MAP.RAM 0xc000--0xdfff; activate shadow memory
MAP.XRAM 0xc000--0xdfff; activate for read and write access
Dump E:0xc000; show shadow memory
```
The MCDS (MultiCore Debug Solution) is an on-chip trigger and trace solution from Infineon, available for the Infineon TriCore, PCP, GTM, XC2000 and C166 architectures.

There are two basic MCDS concepts:

- The regular MCDS is part of the Emulation Extension Chip (EEC). It supports a big feature set for trace and trigger of the CPUs and buses.
- The miniMCDS is part of the Product Chip (PC). It has a reduced feature set only, e.g. only one core is traceable and no buses. Also the trace buffer size is limited.

The MCDS commands described in this document are for reference only, so please refer to the

- "MCDS User's Guide" (mcds_user.pdf) for understanding the MCDS concept and the TRACE32 MCDS support
- "AURIX Trace Training" (training_aurix_trace.pdf) for examples on how to use the MCDS on TriCore AURIX
- The Infineon documentation for devices specific information

See also

- MCDS.CLEAR
- MCDS.DIAG
- MCDS.Init
- MCDS.OFF
- MCDS.Option
- MCDS.PortSPEED
- MCDS.Register
- MCDS.RESet
- MCDS.SessionKEY
- MCDS.SOURCE
- MCDS.TimeStamp
- MCDS.GAP()
- MCDS.STATE()
- MCDS.TraceBuffer.SIZE()
MCDS.CLEAR

Clear programming and initialize MCDS registers

Format: MCDS.CLEAR

The **MCDS.CLEAR** command performs the following actions:

- Performs **MCDS.Init**:
  - Enables the MCDS (**MCDS.ON**)  
  - Initializes all counters e.g., used by the **BMC** commands.  
  - Reprograms the entire MCDS breakpoint, trigger and trace configuration. All MCDS registers are re-written to ensure coherency between the setting assumed by TRACE32 and the target.
- Initializes the MCDS related traces (**Trace.Init**)  
- Clears all settings of an OCTL program  
- Clears all settings done by the command **MCDS.Set**

**See also**
- **MCDS**
- **MCDS.state**

MCDS.CLOCK

Configure MCDS clock system

The **MCDS.CLOCK** command group is used for functionality related to the MCDS clocks and clock system:

- Inform TRACE32 about the MCDS clock configuration. This is required for a correct decoding of the MCDS timestamps and to calculate the CPU clock cycles. There are two different strategies:
  - Use the CLOCK feature for an automatic detection of the on-chip clock programming.
  - If the on-chip clock programming cannot be used, e.g. for post-mortem analysis, the clock configuration can be specified manually.
- For C166 and XC2000ED, the programming of the MCDS on-chip clocks can be done by TRACE32 based on the manual configuration.
- Configuration of a timer to generate a periodic trigger.

Please refer to "Clock System" (mcds_user.pdf) for more information on the concept and usage of the **MCDS.CLOCK** commands.

**See also**
- **MCDS**
- **MCDS.CLOCK MCDSDIV**
- **MCDS.CLOCK TIMER**
- **MCDS.CLOCK DEPRECATED**
- **MCDS.CLOCK REFDIV**
- **MCDS.CLOCK TimeStamp**
- **MCDS.CLOCK EXTern**
- **MCDS.CLOCK REFerence**
- **MCDS.CLOCK SYStem**
- **MCDS.CLOCK Frequency**
- **MCDS.CLOCK state**

©1989-2015 Lauterbach GmbH
MCDS.CLOCK DEPRECATED  Enable deprecated MCDS clock programming

| Format: | MCDS.CLOCK DEPRECATED [OFF | ON] (not for GTM) |
|---------|-----------------------------------------------|

Configures the usage of the deprecated commands for specifying and programming the MCDS clocks.

- The command is available for TriCore AUDO devices only to enable deprecated historic functionality. In order to avoid programming conflicts between the application and TRACE32 users are strongly recommended not to use the deprecated functionality any more. Migration to the recommended clock system configuration may require changing the application.
- This command and the commands enabled by it are not supported for TriCore AURIX devices and newer, including GTM.
- This command is not available for C166 and XC2000ED. For these architectures, the deprecated commands are the default.

**OFF** (default TriCore, PCP)  Get MCDS clock configuration by reading target registers or manual configuration. Reading the target’s clock configuration requires **CLOCK.ON**. The **CLOCK** feature is not available for all architectures, e.g. not for C166 and XC2000ED.

**ON** (default C166, XC2000ED)  Use the deprecated clock configuration method.

For more information on the MCDS clock system and configuration, refer to "Clock System" (mcds_user.pdf).

See also
- MCDS.CLOCK
MCDS.CLOCK EXTern

Set the external clock frequency

| Format: | MCDS.CLOCK EXTern <frequency> (deprecated for TriCore, PCP) |

Default: Device dependent

Specifies the frequency of the external clock $f_{EXT}$. This is required for decoding absolute timestamps and for configuring the periodic trigger event MCDS.CLOCK TIMER.

For TriCore and PCP this command has to be enabled using MCDS.CLOCK DEPRECATED.

For more information on the MCDS specific clock generation of your device, see "Device Specific Details" (mcds_user.pdf).

See also

- MCDS.CLOCK
MCDS.CLOCK Frequency Specify MCDS-related frequencies by commands

If the MCDS related frequencies cannot be evaluated reading the target registers, e.g. in case of post-mortem analysis, the frequencies for timestamp decoding have to be specified manually. Another use case is configuring the periodic trigger event MCDS.CLOCK TIMER.

The manually configured frequencies are used if CLOCK.OFF or the CLOCK feature is not available, and MCDS.CLOCK DEPRECATED is OFF.

For more information on the MCDS clock system see "Clock System" (mcds_user.pdf).

See also
- MCDS.CLOCK
- MCDS.CLOCK Frequency.ReferenceClock
- MCDS.CLOCK Frequency.McdsClock

MCDS.CLOCK Frequency.McdsClock Specify the MCDS clock

Format: MCDS.CLOCK Frequency.McdsClock <frequency>

Default: 0.Hz

Specifies the frequency of the MCDS clock $f_{MCDS}$. This is required for decoding relative timestamps.

See also
- MCDS.CLOCK Frequency

MCDS.CLOCK Frequency.ReferenceClock Specify the reference clock

Format: MCDS.CLOCK Frequency.McdsClock <frequency>

Default: 0.Hz

Specifies the frequency of the reference clock $f_{REF}$. This is required for decoding absolute timestamps and for configuring the periodic trigger event MCDS.CLOCK TIMER.

See also
- MCDS.CLOCK Frequency
MCDS.CLOCK MCDSDIV

Set divider for generating the MCDS clock

Format:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCDS.CLOCK MCDSDIV &lt;divider&gt;</td>
<td>(deprecated for TriCore, PCP)</td>
</tr>
<tr>
<td>MCDS.CLOCK SYStemDIV &lt;divider&gt;</td>
<td>(deprecated)</td>
</tr>
</tbody>
</table>

Default: Device dependent, minimum possible value

Configures the divider for generating the MCDS clock. The legal divider values are dependent on the device. TRACE32 knows about the limitations and auto-adjusts the user value in case the specified setting is not applicable in the current context.

For TriCore and PCP devices this command has to be enabled using MCDS.CLOCK DEPRECATED.

For more information on the MCDS specific clock generation of your device, see "Device Specific Details" (mcds_user.pdf).

See also

- MCDS.CLOCK

MCDS.CLOCK REFDIV

Set divider for generating the reference clock

Format:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MCDS.CLOCK REFDIV &lt;divider&gt;</td>
<td></td>
</tr>
<tr>
<td>MCDS.CLOCK EXTernDIV &lt;divider&gt;</td>
<td>(deprecated)</td>
</tr>
</tbody>
</table>

Default: Device dependent, minimum possible value

Configures the divider for generating the reference clock. The legal divider values are dependent on the device. TRACE32 knows about the limitations and auto-adjusts the user value in case the specified setting is not applicable in the current context.

For more information on the MCDS specific clock generation of your device, see "Device Specific Details" (mcds_user.pdf).

See also

- MCDS.CLOCK
**MCDS.CLOCK REFERENCE**

Select the reference clock source

| Format: | MCDS.CLOCK REFERENCE [USB | PLL | ERAY | BACKUP] |

Default: Device dependent

Selects which clock is input for the reference clock $f_{REF}$. **USB** and **ERAY** is the external clock $f_{EXT}$, **PLL** is the System Clock $f_{SYS}$ and **BACKUP** the internal Backup Clock $f_{BACK}$.

For more information on the MCDS specific clock generation of your device, see "Device Specific Details" (mcds_user.pdf).

**See also**
- MCDS.CLOCK

**MCDS.CLOCK SYStem**

Set the system clock frequency

| Format: | MCDS.CLOCK SYStem <frequency> (deprecated for TriCore, PCP) |

Default: Device dependent

Specifies the frequency of the system clock $f_{SYS}$. This is required for calculating the MCDS clock $f_{MCDS}$. $f_{MCDS}$ is used for sampling the trace data generated by the cores and buses. All relative timestamp messages, including TICK messages are generated depending on $f_{MCDS}$.

For TriCore and PCP devices this command has to be enabled using **MCDS.CLOCK DEPRECATED**.

**See also**
- MCDS.CLOCK

©1989-2015 Lauterbach GmbH
MCDS.CLOCK TIMER
Setup timer for periodic trigger event

Format:  
MCDS.CLOCK TIMER [<frequency> | <period>]

| <frequency> | 1.0…<maximum frequency> |
| <period>    | 1.0…<maximum period>    |

Default: 0. (disabled)

MCDS has a timer driven by the reference clock. It can be used to generate a periodic trigger signal. Periods from micro seconds up to minutes are possible depending on the available clock source.

Not all values can be entered as a frequency or period. The time base for the period is seconds, but the unit “s” must not be specified on the command line. Not all frequencies are possible, an appropriate one is chosen. Entering a frequency higher than the reference clock disables the trigger generation.

The trigger signal is not available immediately. It must be connected to an event for becoming effective. For more information, refer to “Periodic Trigger” (mcds_user.pdf).

See also
- MCDS.CLOCK
MCDS.CLOCK TimeStamp

Force decoding of timestamp messages

Format:  
MCDS.CLOCK TimeStamp [AUTO | OFF | Relative | Absolute]

**AUTO** (default)  
Decode timestamps according to configuration made by TRACE32. The settings of **MCDS.TimeStamp** and an OCTL trigger program (see **MCDS.Program** command) is evaluated for determining whether timestamps have to be decoded or not.

**OFF**  
Do not decode any timestamps, even if generated. This option is useful to increase the decoding speed in case of big trace recordings.

**Relative**  
Decode relative timestamps based on the MCDS clock.

**Absolute**  
Decode absolute timestamps based on the reference clock.

This command controls the decoding of timestamps. It only makes sense to change the default setting **AUTO** in a few cases only:

- Avoid long processing times by disabling the timestamp decoding. Timestamp decoding can be re-enabled at any point of time if necessary.

- When timestamps are generated manually, TRACE32 does not know that there are timestamps to be generated. Use **Relative** and **Absolute** for telling TRACE32 which timestamps to decode.

Absolute and relative timestamps can be generated simultaneously, but only one kind of them can be displayed at a time. Switching between both methods is possible, there is no need to perform a new recording.

TRACE32 only configures relative timestamps. Absolute timestamps are required for special use cases only and require manual configuration. Manual configuration requires expert knowledge. See "Guarded MCDS Programming" (mcds_user.pdf) for more information.

See also

- MCDS.CLOCK
**MCDS.DIAG**

Enable diagnostic output

```markdown
Format:  MCDS.DIAG [ON | OFF]
```

**OFF** (default)  Disable diagnostic MCDS-related output.

**ON**  Enable MCDS-related diagnostic output for trace, trigger and filter programming. The output contains internal information and can only be interpreted with expert knowledge. Only enable when requested by Lauterbach support engineer.

**See also**
- MCDS
- MCDS.state

---

**MCDS.INFO**

Information on MCDS and usage

```markdown
Format:  MCDS.INFO
```

Opens a window to provide detailed information about the MCDS of the current device:

- MCDS ID and module version.
- Emulation memory usage.
- Which MCDS features, e.g. actions, watchpoints or cross-triggers, are available and how many of them are already in use. For example, this supports an advanced or expert user writing trigger programs.

**See also**
- MCDS
- MCDS.state

---

©1989-2015 Lauterbach GmbH
The **MCDS.Init** command performs the following:

- Enable MCDS (perform **MCDS.ON**).
- Initializes all counters e.g., used by the **BMC** commands.
- Reprogram the entire MCDS breakpoint, trigger and trace configuration. All MCDS registers are re-written to ensure coherency between the setting assumed by TRACE32 and the target.

**See also**
- MCDS
- MCDS.state

**MCDS.LOG**

The MCDS logging feature provides a mechanism for the experienced MCDS user to keep track of which MCDS registers are accessed by the MCDS subsystem of the TRACE32 PowerView software.

The TRACE32 breakpoint feature is designed in a way that it automatically performs the trigger programming depending on the device's capabilities as well as the actual MCDS resource usage. In case a potential configuration is already in use by some other feature, e.g. the **MCDS.Set** command, it searches for another valid programming. Also in case the user wishes to use a resource already in use by the Simple Trigger, PowerView will try to find an alternative programming for the Break Actions, enabling the user to use the requested resource.

So with every issued command, the MCDS resource usage can change completely. The MCDS logging feature tracks these changes for enabling the user to keep track of all changes issues with every command.

Please note that for a complete understanding of the information provided by the MCDS logging feature a deeper knowledge of the MCDS implementation is mandatory. The PowerView normally works in cached mode, so the target hardware is only updated when necessary, e.g. when restarting target execution.

**NOTE:** **MCDS.LOG** is only available for C166 and XC2000ED.

**See also**
- MCDS
- MCDS.LOG OWner
- MCDS.LOG CLEAR
- MCDS.LOG ON
- MCDS.LOG OFF
- MCDS.LOG view
- MCDS.state
MCDS.LOG CLEAR

Clears all MCDS register access log entries

Format: **MCDS.LOG CLEAR** (C166, XC2000ED only)

Deletes all existing log entries. **MCDS.LOG.view** will then show an empty window.

**See also**
- **MCDS.LOG**

MCDS.LOG OFF

Disable MCDS register accesses logging

Format: **MCDS.LOG OFF** (C166, XC2000ED only)

Default: Disabled.

Disables the MCDS register access logging. Required for debug purpose only.

**See also**
- **MCDS.LOG**

MCDS.LOG ON

Enable MCDS register accesses logging

Format: **MCDS.LOG ON** (C166, XC2000ED only)

Default: Disabled.

Enables the MCDS register access logging. Required for debug purpose only.

**See also**
- **MCDS.LOG**
Select MCDS owners for logging

Format: \texttt{MCDS.LOG OWner <owner> [ON | OFF]} (C166, XC2000ED only)

\texttt{<owner>} Defines which owner's accesses to log:

- \textbf{All} (default) \quad Set logging for all owners.
- \textbf{NONE} \quad Set logging for all owners.
- \textbf{BMC} \quad Set logging for Benchmark Counter owners.
- \textbf{CTRL} \quad Set logging for trace control owners.
- \textbf{OCTL} \quad Set logging for \texttt{OCTL} owners.
- \textbf{SET} \quad Set logging for \texttt{MCDS.Set} command owners.
- \textbf{ST} \quad Set logging for \texttt{Break.Set} owners.

See also

\texttt{MCDS.LOG}

View MCDS register accesses

Format: \texttt{MCDS.LOG view [/<option>] (C166, XC2000ED only)}

\texttt{<option>} \quad \texttt{HighLight}

Displays the performed MCDS register accesses:

<table>
<thead>
<tr>
<th>address</th>
<th>Register address of resource</th>
</tr>
</thead>
<tbody>
<tr>
<td>rw</td>
<td>Access type: read or write</td>
</tr>
<tr>
<td>name</td>
<td>Register name as in peripheral file or documentation</td>
</tr>
<tr>
<td>cTarget</td>
<td>Register content as stored in target, if already read from target</td>
</tr>
<tr>
<td>cHostval</td>
<td>Register content as stored in host</td>
</tr>
<tr>
<td>cHostmask</td>
<td>Which bits of cHostval are valid</td>
</tr>
<tr>
<td>owner</td>
<td>Owner of a register content modification</td>
</tr>
</tbody>
</table>
The HighLight option allows to easily distinguish between the displayed lines.

**See also**
- MCDS.LOG

---

### MCDS.OFF

**Disable MCDS programming**

Format: `MCDS.OFF`

Default: MCDS enabled

Disables all MCDS related debugger functionality. TRACE32 will stop programming MCDS registers. When there are no other GUIs attached to the same Emulation Device that have the MCDS enabled, TRACE32 will disable the MCDS hardware.

**See also**
- MCDS
- MCDS.state

---

### MCDS.ON

**Enable MCDS programming**

Format: `MCDS.ON`

Default: MCDS enabled

Enables all MCDS related debugger functionality, such as onchip trace, additional breakpoints, …

**See also**
- MCDS
- MCDS.state
### MCDS.Option AddressBreak

Use MCDS for address breakpoints

**Format:**

```
MCDS.Option AddressBreak [ON | OFF] (TriCore and PCP only)
MCDS.Option WriteAddressBreak [ON | OFF] (deprecated)
```

Default: see table

When enabled, MCDS can be used to implement address breakpoints using the **Break.Set** command. The number of additional breakpoints is device dependent.

<table>
<thead>
<tr>
<th>TriCore architecture version</th>
<th>supported address breakpoints</th>
<th>default</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1.6 (AUDO) and older</td>
<td>write address</td>
<td>OFF</td>
</tr>
<tr>
<td>v1.6.1 (AURIX) and later</td>
<td>read/write address</td>
<td>ON</td>
</tr>
</tbody>
</table>

For further restrictions when using MCDS breakpoints, see the **MCDS.Option ProgramBreak** command.

**See also**

- MCDS
- MCDS.Option AddressBreak
- MCDS.Option DataAssign
- MCDS.Option DataBreak
- MCDS.Option eXception
- MCDS.Option ProgramBreak
- MCDS.Option QuickOFF
- MCDS.Option TTRESet
- MCDS.state

With the **MCDS.Option** commands, the user can control the behavior of the MCDS programming.
**MCDS.Option DataAssign**  
Data assignment in trace listing

| Format: | MCDS.Option DataAssign [ON | OFF] (XC2000ED only) |
|---------|-----------------------------------------------|

Default: ON

When enabled, the debugger tries to assign the data cycles to the associated program cycles. The not assigned data cycles are shown in red in the trace listing. When disabled, no data assignment is used.

**See also**
- MCDS.Option

---

**MCDS.Option DataBreak**  
Use MCDS for data breakpoints

| Format: | MCDS.Option DataBreak [ON | OFF] (TriCore and PCP only) |
|---------|-----------------------------------------------|
|         | MCDS.Option WriteDataBreak [ON | OFF] (deprecated) |

Default: OFF

When enabled, MCDS can be used to implement data breakpoints (address and value) using the Break.Set command. The number of additional breakpoints is device dependent.

Note that enabling this feature automatically disables the Run-stop data breakpoint feature of the TriCore OCDS breakpoints and only MCDS write data value breakpoints are available.

For further restrictions when using MCDS breakpoints see the MCDS.Option ProgramBreak command.

**See also**
- MCDS.Option
**MCDS.Option eXception**  
Exception identification in trace decoder

```
Format:  
MCDS.Option eXception.DCU [ON | OFF] (TriCore only)  
MCDS.Option eXception.TABLE <table_config> (TriCore only)
```

```
<table_config>:
[OFF | AUTO | Interrupt {<range> [<size>]} | Trap {<range> [<size>]}]
```

```
<size>:
Size of an exception handler entry, default is 32 B.
```

Default: DCU OFF, TABLE AUTO

MCDS.Option eXception is a command group that configures how the MCDS trace decoder identifies the occurrence of interrupt and trap events. For TriCore it replaces the obsolete SYStem.Option [INTSTART | INTUSE | TRAPSTART] commands.

In case of multi-core up to three address ranges can be specified, one for each core starting with core 0.

Looking up the exception handler entries in the table can be disabled using OFF. It is not possible to detect only parts of the configuration automatically.

**See also**
- MCDS.Option

---

**MCDS.Option ProgramBreak**  
Use MCDS for program breakpoints

```
Format:  
MCDS.Option ProgramBreak [ON | OFF] (TriCore and PCP only)
```

Default: OFF

When enabled, MCDS can be used to implement program stop breakpoints when using the Break.Set command. The number of additional breakpoints is device dependent.

Restrictions when using MCDS breakpoints:

- MCDS program breakpoints are only used when no OCDS breakpoints are available any more.
- MCDS breakpoints are inaccurate and will not stop on the instruction they were programmed.
- Although it is possible to determine whether a break was triggered by MCDS, it is not possible to determine which MCDS breakpoint finally triggered.
Use a properly configured MCDS on-chip trace to get more information on the trigger cause. It is not possible to select which breakpoint is configured by OCDS or MCDS.

See also
- MCDS.Option

**MCDS.Option QuickOFF**  
Disable trace recording by hardware

| Format: | MCDS.Option QuickOFF [ON | OFF] (TriCore and PCP only) |

Default: OFF

When enabled, the debugger uses a hardware signal to disable the trace recording in case the CPU stops the application execution. This avoids the generation of additional messages, e.g. timestamp messages, and improves the trace buffer usage.

**NOTE:**  
MCDS.Option QuickOFF:
- Only has an effect when <trace>.AutoArm is enabled.
- Is disabled when the Break Action WATCH is selected.
- Uses many trigger resources, especially in multi-core scenarios. Enabling this option will reduce the number of available triggers.

See also
- MCDS.Option
Enable generation of reset information in trace

```
Format: MCDS.Option TTRESet [ON | OFF]
```

Default: ON

Required for trace-through-reset scenarios. Generates additional trace messages in order to detect the occurrence of a reset and to find the start for decoding the trace messages.

The option can be disabled to avoid the generation of these trace messages. This may be useful in special trigger setups only. The generated trace messages are selective relative timestamps for the reconstruction of the trace buffer context information and DCU messages for the identifying the occurrence of the reset.

**See also**
- MCDS.Option
MCDS.PortSIZE
Set number of used Aurora lanes

| Format: | MCDS.PortSIZE [<lanes>] |
| <lanes>: | 1Lane |

Default: 1Lane

The Aurora serial trace protocol supports the use of several serial data streams (lanes) in parallel. This command allows to select how many lanes are to be used in this setup. Changing the port size will result in a Trace.Init.

This command is only available for the AGBT (Aurora GigaBit Trace) off-chip trace feature.

See also
- MCDS
- MCDS.state

MCDS.PortSPEED
Set Aurora lane speed

| Format: | MCDS.PortSPEED [<speed>] |
| <lanes>: | 625Mbps | 1250Mbps | 2500 Mbps |

Default: 2500Mbps

This command defines the transfer rate of one Aurora lane. In case more than one lane is used, all lanes will operate with the same transfer rate. Using more than one lane at the same time may require reducing the port speed. Changing the port speed will result in a Trace.Init.

This command is only available for the AGBT (Aurora GigaBit Trace) off-chip trace feature.

See also
- MCDS
- MCDS.state
This command opens a trigger programming window.

The programming is guided by softkeys.

Pushing the Compile button programs the on-chip trigger unit. If the programming was successful, the following message is displayed in the softkey line:

![compiled successfully]

The default extension for an on-chip trigger program is ".octl". If no file name is specified the name t32.octl is used.

For more information on the built-in editor and the save options refer to the EDIT.file command in "IDE Reference Guide" (ide_ref.pdf).

See also
- MCDS
- MCDS.state
## MCDS.Register

Open window with MCDS registers

### Format:
```
MCDS.Register [<file> [/<options>]]
```

- **<file>**: name of the register file or comma for default
- **<options>**: `SpotLight | DualPort | Core <core number>`

Open a peripheral window showing all MCDS related registers. By default, the register file of the currently selected devices is opened.

### See also
- MCDS
- MCDS.state

## MCDS.ReProgram

Load and execute an OCTL trigger program

### Format:
```
MCDS.ReProgram [<file>]
```

- **<file>**: name of the trigger program or comma for default

Load and executes an OCTL trigger program.

The default extension for an on-chip trigger program is ".octl". If no file name is specified the name t32.octl is used.

### See also
- MCDS
- MCDS.state
The **MCDS.RESet** command performs the following:

- Reset all MCDS settings to their defaults
- Clears the MCDS related traces (**Trace.Init**)
- Clears all settings of an OCTL program
- Clears all settings done by the command **MCDS.Set**
- Resets all counters e.g., used by the **BMC** commands.
- All MCDS registers are re-written to ensure coherency between the setting assumed by TRACE32 and the target.

See also
- **MCDS**
- **MCDS.state**
MCDS.RM

MCDS resource management commands

Commands for controlling the MCDS Resource Management. These commands are mainly for diagnostic purpose and not necessary for normal operation.

The MCDS Resource Management is a data structure containing the MCDS register configuration for maintaining coherency between multiple PowerView instances connected to the same Emulation Device and the register programming of the Emulation Device itself. This avoids conflicting register accesses and trigger setups.

The MCDS Resource Management also acts as cache to improve performance.

See also
- MCDS
- MCDS.RM ReSTore
- MCDS.RM WriteTarget
- MCDS.state

MCDS.RM ReSTore

Restore MCDS registers.

Format: MCDS.RM ReSTore

Re-write all MCDS registers.

All modified MCDS registers are re-written, overwriting any manual change by the user. If an MCDS register has an internal reset value, the register will be reset to this value.

See also
- MCDS.RM

MCDS.RM WriteTarget

Flush MCDS register cache

Format: MCDS.RM WriteTarget

Write internally cached MCDS register settings to target.

All modified MCDS registers are re-written. If a register was changed by the user but not by TRACE32, the user’s setting will not be overwritten. MCDS.Set modifications are considered to be TRACE32 related.

See also
- MCDS.RM
Provide a 64 bit MCDS session key for unlocking the MCDS in case it is locked by the application. This is normally only required in very late stages of the development phase.

Default: 0x0000000000000000.

See also

- MCDS
- MCDS.state
The **MCDS.Set** commands provide an interface to program an MCDS feature from a logical point of view. Although the commands are quite comfortable and more or less self-explaining a detailed understanding of the MCDS implementation is mandatory. See the Infineon MCDS documentation for details.

**<unit>** is a core, bus or the MCX:

- **MCX**
  Program a feature of the Multi-core Cross-connnect.
- **CpuMux0**
  Program a feature of the processor connected to CPU multiplexer 0.
- **CpuMux1**
  Program a feature of the processor or the OTGM connected to CPU multiplexer 1.
- **TriCore**
  Program a feature of the TriCore processor.
- **PCP**
  Program a feature of the PCP processor.
- **C166**
  Program a feature of the C166 processor.
- **SPB**
  Program a feature of the System Peripheral Bus.
- **RPB**
  Program a feature of the Remote Peripheral Bus.
- **LMB**
  Program a feature of the Local Memory Bus.
- **SRI**
  Program a feature of the Shared Resource Interconnect.

An **MCDS.Set** command programs all registers belonging to the selected feature, e.g. an IP pretrigger programs the bound and the ranges value at the same time. Implicit information is added automatically.
The following example enables the Program Flow Trace of a TriCore AUDO as long as the CPU executes code from the function `sieve()`. This example is equivalent to the Break Action TraceEnable.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>MCDS.Set TriCore IP0 Var.RANGE(sieve)</code></td>
<td>Pretrigger IP0 is active as long as TriCore executes code within the function <code>sieve()</code></td>
</tr>
<tr>
<td><code>MCDS.Set TriCore EVT0 IP0</code></td>
<td>Enable event EVT0 as long as pretrigger IP is active</td>
</tr>
<tr>
<td><code>MCDS.Set TriCore ACT.PTU_EN 0. EVT0 /Normal /High</code></td>
<td>Enable TriCore Program Flow Trace as long as event EVT0 is active</td>
</tr>
</tbody>
</table>

All common MCDS use cases are available as Trace Triggers and Filters via the Break.Set command and the OCTL trigger programming. The MCDS.Set commands allow setting up filters and triggers for special use cases. They can be used stand alone, in parallel or as an extension to the Trace Triggers and Filters. The last one of course requires a detailed knowledge of how the debugger programs the MCDS. This knowledge is not documented and may change without prior notice.

For performance reason all MCDS register accesses are cached by the TRACE32 software and written to the hardware when necessary, e.g. when resuming program execution, see the MCDS.RM command. As this can impact a currently active trigger configuration, the user can specify whether the MCDS.Set command is to be executed immediately (option WriteThru) or delayed (option Cached) until the next automatic write back. Default is the standard behavior (Cached for TriCore and C166/ XC2000).

For more information see "Guarded MCDS Programming" (mcds_user.pdf).

See also
- MCDS
- MCDS.state

©1989-2015 Lauterbach GmbH
The **MCDS.SOURCE** command group controls which on-chip modules (sources) generate which kind of trace data. In general, there are three basic kinds of trace sources:

- **Core trace**: Trace data generated by an execution unit, e.g. program flow but also memory accesses.
- **Bus trace**: Trace data generated by a bus unit, e.g. destination address, data and meta information.
- **Peripheral trace**: Trace data generated by peripherals, e.g. DMA or Interrupt Router, or special execution units, e.g. GTM.

### See also
- **MCDS**
- **MCDS.SOURCE.ALL**
- **MCDS.SOURCE.DEFaul**
- **MCDS.SOURCE.NONE**
- **MCDS.SOURCE.Set**
- **MCDS.state**
- **MCDS.SOURCE.RESet** (deprecated)

## MCDS.SOURCE.ALL

Enable all MCDS trace sources

**Format:**

```
MCDS.SOURCE.ALL
```

The virtual trace sources **ALL** enables all available trace sources and types in one step.

### See also
- **MCDS.SOURCE**

## MCDS.SOURCE.DEFaul

Set default MCDS trace sources.

**Format:**

```
MCDS.SOURCE.DEFaul
```

Sets all **MCDS.SOURCE** configurations to their default values.

### See also
- **MCDS.SOURCE**
### MCDS.SOURCE.Set

Set individual MCDS trace sources

**Format 1:**

```
MCDS.SOURCE.Set <cpu>.<parameter>
```

**<cpu>:**

CPU source

- CpuMux0
- CpuMux1
- TriCore
- PCP
- C166

**<parameter>:**

MUX source

- Core NONE
- Core TriCore0
- Core TriCore1
- Core TriCore2
- Core OTGM

**<parameter>:**

CPU trace messages

- Program [ON | OFF]
- FlowTrace [ON | OFF] (deprecated)
- ReadAddr [ON | OFF]
- ReadData [ON | OFF]
- WriteAddr [ON | OFF]
- WriteData [ON | OFF]
- OwnerShip [ON | OFF]
- Debug [ON | OFF]

**<parameter>:**

program trace mode

- PTMode FlowTrace
- PTMode SyncTrace
- PTMode CFT

**Format 2:**

```
MCDS.SOURCE.<bus>.<parameter>
```

**<bus>:**

bus sources

- SPB
- RPB
- LMB

**<parameter>:**

bus trace messages

- ReadAddr [ON | OFF]
- ReadData [ON | OFF]
- WriteAddr [ON | OFF]
- WriteData [ON | OFF]
- Debug [ON | OFF]
| Format 3: | MCDS.SOURCE.Set SRI.(1 | 2).
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>SRI trace</td>
<td>MCDS.SOURCE.Set SRI.Debug [ON</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>&lt;parameter&gt;:</th>
<th>SLAVE CPU0</th>
<th>CPU1</th>
<th>CPU2</th>
</tr>
</thead>
<tbody>
<tr>
<td>SRI slave</td>
<td>SLAVE CPU1_PMI</td>
<td>CPU1_DMI</td>
<td>CPU2_PMI</td>
</tr>
<tr>
<td></td>
<td>SLAVE PMI</td>
<td>DMI</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLAVE PMU0</td>
<td>PMU1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLAVE PMU0_PFLASH0</td>
<td>PMU0_PFLASH1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SLAVE PMU0_PFLASH2</td>
<td>PMU0_PFLASH3</td>
<td>PMU0_DFLASH</td>
</tr>
<tr>
<td></td>
<td>SLAVE EBU</td>
<td>LMU</td>
<td>SFI</td>
</tr>
</tbody>
</table>

| <parameter>: | ReadAddr [ON | OFF] |
|-------------|----------------------|
| SRI messages | ReadData [ON | OFF] |
|             | WriteAddr [ON | OFF] |
|             | WriteData [ON | OFF] |

©1989-2015 Lauterbach GmbH
Trace Source Configuration for Cores

<cpu> defines the core for which type of trace <message> is to be generated. For single-core systems and TriCore AUDO, <cpu> directly addresses the core. For TriCore AURIX <cpu> addresses a multiplexer which allows to choose a core or the OTGM. For TriCore AURIX only up to two cores or up to one core and the OTGM can be traced.

Available <cpu> sources:

<table>
<thead>
<tr>
<th>&lt;cpu&gt;</th>
<th>Description</th>
<th>Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>CpuMux0</td>
<td>Generate trace data for the core selected by CPU multiplexer 0. Required for TriCore AURIX and later.</td>
<td></td>
</tr>
<tr>
<td>CpuMux1</td>
<td>Generate trace data for the core or the OTGM selected by CPU multiplexer 1. Required for TriCore AURIX and later.</td>
<td></td>
</tr>
<tr>
<td>TriCore</td>
<td>Generate trace data for the TriCore core. Required for TriCore AUDO only.</td>
<td></td>
</tr>
<tr>
<td>PCP</td>
<td>Generate trace data for the PCP core. Required for TriCore AUDO only.</td>
<td></td>
</tr>
<tr>
<td>C166</td>
<td>Generate trace data for the C166 core. Required for C166 and XC2000ED only.</td>
<td></td>
</tr>
<tr>
<td>NONE</td>
<td>Disable trace data generation for this multiplexer. This will also disable the trigger generation (Break.Set, OCTL) for this multiplexer.</td>
<td></td>
</tr>
<tr>
<td>TriCore0</td>
<td>TriCore1</td>
<td>TriCore2</td>
</tr>
<tr>
<td>OTGM</td>
<td>Select the peripheral trace as input for CpuMux1. Only write data trace is available for this &lt;source&gt;. OTGM is not only used for tracing dedicated peripherals, e.g. DMA or Interrupt Router, but also the core trace of GTM.</td>
<td></td>
</tr>
</tbody>
</table>

For more information on GTM and peripheral trace refer to:

- "Special Trace Sources via OTGM" (mcds_user.pdf)
- "GTM Debugger and Trace" (debugger_gtm.pdf)
Available CPU trace message types:

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Program</td>
<td>Configure generation of program trace messages. Different modes of generating the program trace are available, see the description of the PTrace option below.</td>
</tr>
<tr>
<td>FlowTrace</td>
<td>Configure generation of program flow trace messages. When enabling, program trace is configured with PTrace set to FlowTrace. For backward compatibility only, do not use any more.</td>
</tr>
<tr>
<td>ReadAddr</td>
<td>Configure generation of trace messages (data address) on read accesses. Not on all architectures and chips.</td>
</tr>
<tr>
<td>ReadData</td>
<td>Configure generation of trace messages (data value) on read accesses. Not on all architectures and chips.</td>
</tr>
<tr>
<td>WriteAddr</td>
<td>Configure generation of trace messages (data address) on write accesses.</td>
</tr>
<tr>
<td>WriteData</td>
<td>Configure generation of trace messages (data value) on write accesses.</td>
</tr>
<tr>
<td>Ownership</td>
<td>Configure generation of ownership trace messages. Depending on the core, different type of information is generated:</td>
</tr>
<tr>
<td></td>
<td>• PCP: current channel ID.</td>
</tr>
<tr>
<td></td>
<td>• TriCore: active memory protection set. The availability of this trace type depends on the architecture and the chip.</td>
</tr>
<tr>
<td>Debug</td>
<td>Configure generation of debug and status related trace messages. Generated trace messages provide additional information, e.g. halted or whether the exception handler is active. If not explicitly supported by TRACE32, e.g. for exceptions, this information can only be displayed as a value.</td>
</tr>
</tbody>
</table>

The default core trace source is program flow trace for the first core of the chip.

Any trace filters programmed using the Break.Set command only have an effect on the enabled trace sources.
Example: The following configuration will only generate write trace messages if TriCore core 0 writes to the variable magic, but not if TriCore core 1 accesses it:

```
Break.Set magic /Write /TraceEnable
MCDS.SOURCE.Set CpuMux0.Core TriCore0
MCDS.SOURCE.Set CpuMux0.WriteAddr ON
MCDS.SOURCE.Set CpuMux0.WriteData ON

MCDS.SOURCE.Set CpuMux1.Core TriCore1
MCDS.SOURCE.Set CpuMux1.WriteAddr OFF
MCDS.SOURCE.Set CpuMux1.WriteData OFF
```

Available program trace modes for PTMode:

- **FlowTrace**: Program Flow Trace. Trace information is only generated on a discontinuity of the program flow, e.g. branch or jump instructions. FlowTrace offers the best trace buffer usage but does not provide timestamp information on every executed instruction.

- **SyncTrace**: SYNC Trace mode. Trace information is generated on every MCDS clock cycle. Timestamps are generated for almost all instructions.

- **CFT**: Compact Function Trace. Trace information is only generated for call and return instructions. Information about function call hierarchy may be lost with advanced compiler optimization.

**Trace Source Configuration for Buses**

`<bus>` defines the bus system for which generation of trace data of `<type>` is to be configured. Bus trace is not available for C166 and XC2000ED.

Available `<bus>` sources:

- **SPB**: Configure trace data generation for the System Peripheral Bus.
- **LMB**: Configure trace data generation for the Local Memory Bus. LMB bus trace is not available for AUDO-NG devices.
Available bus trace message types:

- **ReadAddr**: Configure generation of trace messages (data address and meta information) on read accesses.
- **ReadData**: Configure generation of trace messages (data value) on read accesses.
- **WriteAddr**: Configure generation of trace messages (data address and meta information) on write accesses.
- **WriteData**: Configure generation of trace messages (data value) on write accesses.
- **Debug**: Configure generation of debug and status related trace messages. Generated trace messages provide additional information on the bus, e.g., sleeping, reset, error. If not explicitly supported by TRACE32, e.g., for reset, this information can only be displayed as a value.

Meta information provides information, e.g., on the bus master, DMA channel or on the access mode.

### Trace Source Configuration for SRI

The **SRI** is a fabric that connects the cores and on-chip memories on recent TriCore devices (TriCore core architecture v1.6 and later). **SRI** can handle multiple transactions in parallel. The SRI trace can only observe the transactions to up to two bus slaves (the destination of the data transfer). The availability of these slaves is device dependent.

The debug trace messages for SRI are generated for the entire SRI and not independently for each slave.

#### Available `<SRI>` sources:

- **SRI**: Configure trace data generation for the Shared Resource Interconnect.

#### Observable SRI slaves:

The availability of the SRI slaves which can be observed is device and slave dependent.

#### Available SRI trace message types:

- **ReadAddr**: Configure generation of trace messages (data address and meta information) on read accesses.
- **ReadData**: Configure generation of trace messages (data value) on read accesses.
- **WriteAddr**: Configure generation of trace messages (data address and meta information) on write accesses.
**WriteData**

Configure generation of trace messages (data value) on write accesses.

**Debug**

Configure generation of debug and status related trace messages. Generated trace messages provide additional information on the bus, e.g. sleeping, reset, error. If not explicitly supported by TRACE32, e.g. for reset, this information can only be displayed as a value. Debug message generation is independent of the slaves.

Meta information provides information, e.g. on the bus master, DMA channel or on the access mode.

---

**See also**
- [MCDS.SOURCE](#)

## MCDS.SOURCE.NONE

**Disable all MCDS trace sources.**

**Format:**

```
MCDS.SOURCE.NONE
```

The virtual trace sources **NONE** disables all available trace sources and types in one step.

---

**See also**
- [MCDS.SOURCE](#)
**MCDS.state**

Display MCDS configuration window

Opens the MCDS configuration window.

### Format:

```
MCDS.state
```

**See also**

- MCDS
- MCDS.INFO
- MCDS.ON
- MCDS.Program
- MCDS.RM
- MCDS.TimeStamp
- MCDS.CLEAR
- MCDS.Init
- MCDS.Option
- MCDS.Register
- MCDS.SessionKEY
- MCDS.Set
- MCDS.CLOCK
- MCDS.LOG
- MCDS.PortSIZE
- MCDS.ReProgram
- MCDS.RESet
- MCDS.DIAG
- MCDS.OFF
- MCDS.PortSPEED
- MCDS.RESet
- MCDS.SOURCE

▲ 'TRACE32 Support for Emulation Devices' in 'MCDS User's Guide'

---

**MCDS.TimeStamp**

Enable MCDS trace sources

### Format:

```
MCDS.TimeStamp [<timestamp>] [ON | OFF]
```

- **<timestamp>:**
  - ON | OFF
  - Tick | Absolute | Relative | RelTick (C166 only)
  - AutoCONFig (deprecated alias for ON)

Default: OFF
Enable or disable timestamp generation. C166 allows to manually select the timestamp type. Recommendation is to use either OFF or ON which automatically selects the best method depending on the Break Action configuration.

For more information on MCDS timestamps see “MCDS Timestamps” (mcds_user.pdf).

See also

- MCDS
- MCDS.state
The **MCDS.TraceBuffer** commands allow to configure the EMEM for being used as on-chip trace buffer or AGBT FIFO. A correct setup is not only required for the operation of the trace modes but also for cooperation with third-party applications such as calibration tools, or when using parts of the EMEM for application.

TriCore miniMCDS, C166 and XC2000ED do not allow to configure the trace buffer.

### NOTE:

- All **MCDS.TraceBuffer** commands influence each other. Especially pay attention to the **MCDS.TraceBuffer NoStealing** setting.
- When switching the trace method, the current trace buffer configuration (array, size, lower and upper gap) will be remembered when switching back to this method.
- When switching the memory arrays within the same trace method, the trace buffer configuration (size and gap) will be reset to the default values according to the newly selected trace method.
- Always check the results of your configuration to avoid unwanted effects.


### See also

- **MCDS**
- **MCDS.TraceBuffer ARRAY**
- **MCDS.TraceBuffer LowerGAP**
- **MCDS.TraceBuffer SIZE**
- **MCDS.state**
- **MCDS.TraceBuffer DETECT**
- **MCDS.TraceBuffer NoStealing**
- **MCDS.TraceBuffer UpperGAP**

---

### MCDS.TraceBuffer ARRAY

**Select MCDS trace buffer array**

Format:  

```
MCDS.TraceBuffer ARRAY [TCM | XTM]
```

Selects which memory array is to be used as on-chip trace buffer or memory array. Not all memory arrays are available for all devices. Memory arrays that can not be used as trace buffer, e.g. XCM (calibration only), can not be configured.

- **TCM**
  
  Use TCM (Trace- and Calibration Memory) as trace buffer.
  
  Huge trace tiles. Default for all onchip traces.

- **XTM**
  
  Use XTM (Extended Trace Memory) as trace buffer.
  
  Small trace tiles for use as FIFO. Default for all off-chip traces, if available.

### See also

- **MCDS.TraceBuffer**

©1989-2015 Lauterbach GmbH
MCDS.TraceBuffer DETECT  
Auto-detect MCDS trace buffer configuration

Format:  

| MCDS.TraceBuffer DETECT |

Reads the EMEM configuration from the device and tries to detect which memory array and which tiles can be used as trace buffer. This feature is useful if a third-party tool or the application also uses the emulation memory to allow a concurrent use.

NOTE:  
Ensure that the third-party tool or application already configured the EMEM for its purpose when using this command. Strange effects will occur when the EMEM configuration is changed by application while tracing. TRACE32 will not be able to access the EMEM, also the trace recording is stopped and the trace data stored in the on-chip trace buffer will be destroyed.

The first suitable trace buffer configuration found will be used for tracing. For on-chip trace, TCM is the preferred memory array, for off-chip trace XTM is preferred. If the preferred memory array does not contain a suitable trace buffer configuration, another array is selected. If no array contains a suitable configuration, the trace buffer size is set to zero.

NOTE:  
Always check the results of the detection to avoid any unwanted setup.

The first suitable trace buffer configuration is not necessarily the largest possible configuration. The search for the trace buffer array starts at tile 0 and stops, when after the first range of suitable tiles a non-suitable tile is found.

See also  
■ MCDS.TraceBuffer

MCDS.TraceBuffer LowerGAP  
Set MCDS trace buffer lower gap

Format:  

| MCDS.TraceBuffer LowerGAP <size> |

Default: 0 bytes

Configures which EMEM tiles at the lower boundary of the currently selected memory array are not used as trace buffer, starting with tile 0. Some devices, e.g. TriCore AUDO-NG and XC2000, do not support configuration of a lower gap.
The value of the trace buffer lower gap can be entered in Bytes, KB or MB and will be rounded up by the software to match a multiple of the tile size. The configuration of the upper gap will be adjusted accordingly. The trace buffer size is only adjusted in case a further reduction of the upper gap is not possible or the device does not support an upper gap.

See also
- MCDS.TraceBuffer

**MCDS.TraceBuffer NoStealing**

Prevent conflicts with third-party tools

**Format:**

```
MCDS.TraceBuffer NoStealing [ON | OFF]
```

**ON (default)**

Force using EMEM tiles for tracing even if already assigned to a third-party tool or application. This allows TRACE32 to destroy a configuration of another tool or an application in order to use the assigned memory tiles (all or some) for tracing. A warning message is printed in this case.

**OFF**

Do not destroy the EMEM configuration of another tool or application. Instead, TRACE32 tries to find another suitable configuration. If this is not possible, the size of the trace buffer is either set to zero (on-chip trace) or the trace method is disabled at all (off-chip trace).

Always check the result of your trace buffer configuration to avoid any unwanted setup.

**NOTE:**

Devices that do not support the unused mode for trace tiles should be handled with care. For these devices unused mode is identical with calibration mode, so NoStealing should only be enabled in case a third-party tool or the application always maps tiles not used for their purpose to trace mode.

See also
- MCDS.TraceBuffer
MCDS.TraceBuffer SIZE

Set MCDS trace buffer size

Format:

```
MCDS.TraceBuffer SIZE <size>
MCDS.SIZE <size> (deprecated)
```

Default: maximum possible trace buffer size

Configures how many EMEM tiles of the currently selected memory array are used as trace buffer. The value of the trace buffer size can be entered in Bytes, KB or MB, the debugger automatically adjusts to a possible value. Depending on the device, not all Emulation Memory can be used as trace buffer.

Some devices, e.g. XC2000, do not support configuration of the trace buffer size. When using off-chip trace, the trace buffer is used as AGBT FIFO. In this case the trace buffer size can not be changed.

The values of the lower and upper gap are adjusted accordingly. Use MCDS.TraceBuffer LowerGAP and MCDS.TraceBuffer UpperGAP to align the trace buffer within the EMEM. Always check the result of your trace buffer configuration to avoid any unwanted setup.

See also

- MCDS.TraceBuffer
- ‘Trace Configuration within TRACE32’ in ‘AURIX Trace Training’

MCDS.TraceBuffer UpperGAP

Set MCDS trace buffer upper gap

Format:

```
MCDS.TraceBuffer UpperGAP <size>
MCDS.GAP <size> (deprecated)
```

Default: 0 bytes

Configures which EMEM tiles at the upper boundary of the currently selected memory array are not used as trace buffer, starting with the highest tile. Some devices, e.g. XC2000, do not support configuration of an upper gap.

The value of the trace buffer upper gap can be entered in Bytes, KB or MB and will be rounded up by the software to match a multiple of the tile size. The configuration of the lower gap will be adjusted accordingly. The trace buffer size is only adjusted in case a further reduction of the lower gap is not possible or the device does not support a lower gap.

See also

- MCDS.TraceBuffer
**MCDSBase<trace>**

Non-optimized trace MCDS trace processing

| Format: | MCDSBase<trace> (diagnostic use only) |
| <trace>: | Analyzer | Onchip |

MCDSBaseAnalyzer and MCDSBaseOnchip process the MCDS trace data recorded by the Analyzer or Onchip trace without any optimization or fine tuning. The purpose of this command is to find issues related to trace decoder optimizations.

MCDSBaseAnalyzer and MCDSBaseOnchip are used as <trace> aliases.

Example:

```
MCDSBaseAnalyzer.List ; display non-optimized trace content
```

**NOTE:** TRACE32 automatically detects which optimizations are necessary.

---

**MCDSDCA<trace>**

MCDS trace processing with data cycle assignment

| Format: | MCDSDCA<trace> (diagnostic use only) |
| <trace>: | Analyzer | Onchip |

MCDSDCAAnalyzer and MCDSDCAOnchip process the MCDS trace data recorded by the Analyzer or Onchip trace after DDTU reordering and data cycle assignment optimizations. The purpose of this command is to find issues related to trace decoder optimizations.

Data cycle assignment is an optimization where TRACE32 assigns recorded core data cycles (read, write) to the corresponding recorded program cycles. A requirement is the correct order of the core’s data cycles.

MCDSDCAAnalyzer and MCDSDCAOnchip are used as <trace> aliases.

Example:

```
MCDSDCAAnalyzer.List ; display trace content after data cycle assignment
```

**NOTE:** TRACE32 automatically detects which optimizations are necessary.
MCDSDDTU<trace>  MCDS trace processing with DDTU reordering

<table>
<thead>
<tr>
<th>Format:</th>
<th>MCDSDDTU&lt;trace&gt; (diagnostic use only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;trace&gt;:</td>
<td>Analyzer</td>
</tr>
</tbody>
</table>

**MCDSDDTUAnalyzer** and **MCDSDDTUOnchip** process the MCDS trace data recorded by the **Analyzer** or **Onchip** trace after DDTU reordering. The purpose of this command is to find issues related to trace decoder optimizations.

DDTU (Duplex Data Trace Unit) reordering is an optimization where TRACE32 reorders core- and bus data cycles into their correct temporal order. Timestamps is a requirement for DDTU reordering.

**MCDSDDTUAnalyzer** and **MCDSDDTUOnchip** are used as <trace> aliases.

**Example:**

```
MCDSDDTUnalyzer.List ; display trace content after DDTU reordering
```

**NOTE:** TRACE32 automatically detects which optimizations are necessary.
The command **MergeAnalyzer** is used for several purposes:

- Merge the trace information of all cores of an SMP system to display them together in one trace listing.
- Merge a program-flow-only ETM and an HTM data access trace to display them together in one trace listing.
- Merge a program-flow-only ETM and ITM trace on selected data accesses to display them in one trace listing.
Merging a program-flow-only ETM and HTM information requires the following settings for the HTM:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTM.ON</td>
<td>enable HTM functionality</td>
</tr>
<tr>
<td>HTM.DataTrace ON</td>
<td>the HTM emits information on all accesses on the AHB bus</td>
</tr>
</tbody>
</table>

No filters for the HTM are allowed.

Example:

Analyzer.List; display ETM trace information
HTMAnalyzer.List; display HTM trace information
MergedAnalyzer.List; display merged information
Merging a program-flow-only ETM and DataPC trace requires the following settings for the ITM:

```
ITM.ON ; enable ITM functionality
ITM.DataTrace DataPC ; advise the ITM to emit data
                      ; address, data value and
                      ; instruction on a specified data
                      ; access

Var.Break.Set flags[3] /TraceData ; the option /TraceData advises
                      ; TRACE32 to program a DTW
                      ; comparator

Analyzer.List ; display ETM trace information
ITMAnalyzer.List ; display iTM trace information
MergedAnalyzer.List ; display merged information
```
The **MIPS** command group can be used to analyze the workload (MIPS) of your systems. The source for this analysis is the trace information recorded into the selected trace sink (**Trace.METHOD** command).

**Basics**

The system can be analyzed under different aspects: workload per task, workload per high-level language line, workload per specified functional group etc.

The following results are provided (workload per task as example):

```plaintext
; the trace information recorded to the PowerTrace is analyzed by the
; MIPS commands
MIPS.STATistic.TASK /InterVal 10.ms
MIPS.PROfileChart.TASK /InterVal 10.ms
MIPS.PROfileSTATistic.TASK /InterVal 10.ms
```

Numeric statistical analysis of the MIPS per task for the recording time.

©1989-2015 Lauterbach GmbH
Graphical analysis of the MIPS per task for the recording time.

The observation time is cut into segments in order to provide the graphical analysis. The default segment size is 10.us. The /\texttt{InterVal} <\textit{time}> option allows to specify a different segment time.

For each segment the MIPS per task are calculated. Based on this calculation the workload for the different tasks is displayed in a graphic.

The command subgroup \texttt{MIPS.PROfileSTATistic.<item>} is provided to enable the export of the results as CSV (Comma Separated Value).

\begin{verbatim}
; select Comma Separated Values as output format
PRinTer.FileType CSV

; specify the file name
PRinTer.FILE Mips

; send the result to the file
WinPrint.MIPS.PROfileSTATistic.TASK /InterVal 10.ms
\end{verbatim}
In order to see the color assignment for all tasks drag the slider.

The color assignment is also displayed in the tool tip.
<table>
<thead>
<tr>
<th><strong>Fine</strong></th>
<th>Decrease segment size by factor 10.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Coarse</strong></td>
<td>Increase segment size by factor 10.</td>
</tr>
</tbody>
</table>

The current interval is displayed in the tool tip.
See also

- MIPS.PROfileChart.GROUP
- MIPS.PROfileChart.LINE
- MIPS.PROfileChart.RWINST
- MIPS.PROfileChart.sYmbol
- MIPS.PROfileChart.TASK
- MIPS.PROfileSTATistic
- MIPS.STATistic
Draw MIPS profile charts for **groups**.

```
; create group "INPUT", add the modules \jquant2 \jdinput \jidctred and
; assign the color AQUA to this group
GROUP.CreateModules "INPUT" \jquant2 \jdinput \jidctred /AQUA

; create another group
GROUP.CreateModules "JPEG" ...

; start and stop the program execution

; display the result
MIPS.PROfileChart.GROUP
```

See also
- MIPS.PROfileChart
MIPS.PROfileChart.Line

MIPS per high-level language line graphically

Format: MIPS.PROfileChart.Line

Draw MIPS graph for all executed hll lines.

See also
- MIPS.PROfileChart

MIPS.PROfileChart.RWINST

MIPS per cycle type graphically

Format: MIPS.PROfileChart.RWINST

Draw MIPS graph for read, write, modify instruction and all others (none) instructions.

See also
- MIPS.PROfileChart
Format: MIPS.PROfileChart.sYmbol

Draw MIPS graph for all program symbols.

See also
- MIPS.PROfileChart
Draw MIPS graph for all tasks.

See also
- MIPS.PROfileChart
MIPS.PROfileSTATistic
Profile statistics for MIPS

See also
- MIPS.PROfileChart
- MIPS.PROfileSTATistic.Line
- MIPS.PROfileSTATistic.sYmbol
- MIPS.STATistic

MIPS.PROfileSTATistic.GROUP
MIPS per GROUP as profile chart

Format: MIPS.PROfileSTATistic.GROUP

Display MIPS as profile statistic for groups.

See also
- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.Line
MIPS per high-level language line as table

Format: MIPS.PROfileSTATistic.Line

MIPS per high-level language line as table.

See also
- MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.RWINST
MIPS per cycle type as table

Format: MIPS.PROfileSTATistic.RWINST

MIPS for read, write, modify instruction and all others (none) instructions as table.

See also
- MIPS.PROfileSTATistic
MIPS.PROfileSTATistic.sYmbol

MIPS for all program symbols as table.

See also
■ MIPS.PROfileSTATistic

MIPS.PROfileSTATistic.TASK

MIPS per task as table.

See also
■ MIPS.PROfileSTATistic
MIPS.STATistic

Statistical analysis for MIPS

MIPS.STATistic.GROUP
MIPS statistic for groups

Format: **MIPS.STATistic.GROUP**

MIPS statistic for groups.

See also
- MIPS.PROfileChart
- MIPS.PROfileSTATistic
- MIPS.STATistic.GROUP
- MIPS.STATistic.RWINST

MIPS.STATistic.RWINST
MIPS per cycle type numerically

Format: **MIPS.STATistic.RWINST**

MIPS per read, write, modify instruction and all others (none) instructions in a numerical statistic.

See also
- MIPS.STATistic

©1989-2015 Lauterbach GmbH

General Commands Reference Guide M 115 MIPS
MIPS.STATistic.sYmbol

MIPS for all program symbol numerically.

See also
- MIPS.STATistic

MIPS.STATistic.TASK

MIPS per task numerically.

See also
- MIPS.STATistic
MMU

Memory management unit

See also
- MMU.DUMP
- MMU.ITALBSCAN
- MMU.SCAN
- MMU.TLB
- MMU.TLBSET
- MMU.UTLBCSCAN
- MMU.DEFAULTPT()
- MMU.PHYSICAL()
- MMU.DEFAULTPT()
- MMU.FORMAT()
- MMU.LINEAR()
- MMU.PHYSICAL()
- MMU.PHYSICAL()
- MMU.DEFAULTPT()
- MMU.FORMAT()
- MMU.LINEAR()
- MMU.PHYSICAL()
- MMU.DEFAULTPT()
- MMU.FORMAT()
- MMU.LINEAR()
- MMU.PHYSICAL()

▲ 'Using the MMU for Banked Target Systems' in 'ICE Emulator for Z80 and Z180'
▲ 'Release Information' in 'Release History'

**MMU.DUMP**

Dump MMU tables

**Format:**

```
MMU.DUMP <table> [<range>] | <addr> | <range> | <root> | <addr> | <root>
```

MMU.<table>.DUMP (deprecated)

**<table>:**

- **PageTable**
- **KernelPageTable**
- **TaskPageTable**

```
<table> <task_name> | <magic> | <space_id> | <space_id>:0x0
```

and CPU specific tables

```
<table>:
  ITLB | DTLB | TLB (e.g. for ARM, MIPS)
  (CPU specific) PTE | BAT (e.g. for MPC8260, MPC750)
  TLB0 | TLB1 (e.g. for MPC54xx, MPC85xx)
```

Displays the contents of the MMU translation table or a CPU specific TLB table.

- If the command is called without parameters, the complete current page table will be displayed; i.e., in this case **MMU.DUMP** is equivalent to **MMU.DUMP PageTable**.
- If the command is called with either an address range or an explicit address, table entries will only be displayed, if their **logical** address matches with the given parameter.

The optional **<root>** argument can be used to specify a page table base address deviating from the default page table base address. This allows to display a page table located anywhere in memory.

tbd.
Description of Columns in the MMU.DUMP.PageTable Window

These columns are available for all architectures.

<table>
<thead>
<tr>
<th>Column name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>logical</td>
<td>Logical page address range</td>
</tr>
<tr>
<td>physical</td>
<td>Physical page address range</td>
</tr>
<tr>
<td>size</td>
<td>Size of mapped page in bytes</td>
</tr>
<tr>
<td>tablewalk</td>
<td>Details of table walk for logical page address (one sub column for each table level, showing the table base address, entry index, entry width in bytes and value of table entry) tbd.</td>
</tr>
</tbody>
</table>

All other columns in the MMU.DUMP PageTable window are architecture specific.

Examples:

; dump entries of current page table in specified range
; <table> <range>
MMU.DUMP PageTable 0xC0000000--0xDFFFFFFF

; display PT with physical base address A:0x00402100
; dump entries with logical address >= 0xC0000000
; <table> <start address> <root>
MMU.DUMP PageTable 0xC0000000 A:0x00402100

; open the TASK.List window to display task name, task magic number, and space ID
TASK.List

; The format of the MMU.DUMP.TaskPageTable argument governs how it is interpreted:

MMU.DUMP.TaskPageTable "ash" ; task name
MMU.DUMP.TaskPageTable 0xC707AE20 ; task magic number in hex
MMU.DUMP.TaskPageTable 673. ; space ID in decimal

; the same space ID in hex, specified as an extended address.
; Only the space ID part of the address is relevant for this command.
MMU.DUMP.TaskPageTable 0x2A1:0x0

The command is target processor specific. For details, see “Processor Architecture Manuals” listed in the See also block below.
See also

- MMU
- MMU.view
- TRANSlation.TlbAutoScan

- 'CPU specific MMU Commands' in 'CPU32 and ColdFire Debugger and Trace'
- 'CPU specific MMU Commands' in 'Andes Debugger'
- 'CPU specific MMU Commands' in 'ARC Debugger'
- 'CPU specific MMU Commands' in 'ARM Debugger'
- 'CPU specific MMU Commands' in 'ARMv8-A Debugger'
- 'CPU specific MMU Commands' in 'Beyond Debugger and Trace'
- 'CPU specific MMU Commands' in 'Hexagon Debugger'
- 'CPU specific MMU Commands' in 'MicroBlaze Debugger and Trace'
- 'CPU specific MMU Commands' in 'MIPS Debugger and Trace'
- 'Command Reference: MMU Commands' in 'Qorivva MPC5xxx/SPC5xx Debugger and NEXUS Trace'
- 'CPU specific MMU Commands' in 'NIOS II Debugger and Trace'
- 'CPU specific MMU Commands' in 'PPC400/PPC440 Debugger and Trace'
- 'CPU specific MMU Commands' in 'PPC600 Family Debugger'
- 'CPU specific MMU Commands' in 'MPC5xx/8xx Debugger and Trace'
- 'CPU specific MMU Commands' in 'PQIII Debugger'
- 'CPU specific MMU Commands' in 'QorIQ Debugger and NEXUS Trace'
- 'CPU specific MMU Commands' in 'SH2 Debugger'
- 'CPU specific MMU Commands' in 'SH2, SH3 and SH4 Debugger'
- 'CPU specific MMU Commands' in 'StarCore Debugger and Trace'
- 'CPU specific MMU Commands' in 'Intel x86/x64 Debugger'
- 'MMU' in 'ICE Emulator for MC68020/30'
- 'MMU' in 'ICE Emulator for MC68040/60'
- 'MMU' in 'ICE Emulator for 386/486'
- 'CPU specific MMU Commands' in 'ARM and XSCALE Monitor'
- 'CPU specific MMU Commands' in 'TRACE32 pdebug Target Server for ARM'
- 'CPU specific MMU Commands' in 'SH2 Monitor'
- 'CPU specific MMU Commands' in 'x386 and x486 Monitor'
- 'CPU specific MMU Commands' in 'MPC56x NEXUS Debugger and Trace'
- 'Release Information' in 'Release History'
- 'CPU specific MMU Commands' in 'Simulator for 68K/ColdFire'
- 'CPU specific MMU Commands' in 'Simulator for ARM and XSCALE'
- 'CPU specific MMU Commands' in 'Simulator for MIPS'
- 'CPU specific MMU Commands' in 'Simulator for NIOS-II'
- 'CPU specific MMU Commands' in 'Simulator for PowerPC'
- 'CPU specific MMU Commands' in 'Simulator for SuperH'
MMU.FORMAT  Define MMU table structure

Format:  

| MMU.FORMAT | <format> [ <base_address> [ <logical_kernel_address_range> <physical_kernel_address> ] ] |

Defines the information needed for the page table walks, which are performed by TRACE32 for debugger address translation, page table dumps, or page table scans.

<format>

<format> is to be replaced with a CPU architecture specific keyword which defines the structure of the MMU page tables used by the kernel. The MMU format STD is used when MMU.FORMAT is not specified at all.

- In addition to STD, some CPU architectures have further <format> keywords. For an overview of these keywords, please refer to “Appendix - <format> Options of MMU.FORMAT”, page 136.
- For more information about the declaration of the debugger address translation, refer to the “RTOS Debugger Manuals”. These manuals also list the OS specific <format> keywords broken down by CPU architecture.

<base_address>

<base_address> defines the default page table which is usually the kernel page table containing translations for mapped address ranges owned by the kernel.

The debugger address translation uses the default page table if no process specific page table (task page table) is available to translate an address.

<base_address> can be left empty by typing a comma or set to zero if there is no default page table available in the system.

<logical_kernel_address_range> and <physical_kernel_address>

The arguments <logical_kernel_address_range> and <physical_kernel_address> define a linear logical-to-physical address translation for the kernel addresses, called kernel translation or default translation. This translation should cover all statically mapped logical address ranges of kernel code or kernel data.

For the <physical_kernel_address> you just need to specify the start address.

NOTE: If no kernel translation is specified for a given memory access, TRACE32 tries to use static address translations defined by the command TRANSLation.Create. The kernel translation is shown in the TRANSLation.List window.
Example

To enable the debugger address translation with page table walk, please use `TRANSlation.ON` and `TRANSlation.TableWalk ON` after specifying `MMU.FORMAT`. This example shows up a typical `MMU.FORMAT` declaration for Linux:

```plaintext
; enable space ID usage, needed for Linux
SYStem.Option.MMUSpaces ON

; these are the arguments used for the MMU.FORMAT example:
; 1. use MMU page table format LINUX for page table walks
; 2. use symbol swapper_pg_dir to define the base address of the
;    kernel page table
; 3. define the kernel translation:
;      translate the logical kernel address range 0xc0000000--0xdfffffff
;      to the physical address range 0x0--0x1fffffff
MMU.FORMAT LINUX swapper_pg_dir 0xc0000000--0xdfffffff 0x0

; define the common kernel address range
TRANSlation.COMMON 0xc0000000--0xffffffff

; enable the table walk and the debugger address translation
TRANSlation.TableWalk ON
TRANSlation.ON
```

The known page tables can be viewed with commands `TRANSlation.List.<page_table>` or `TRANSlation.DUMP.<page_table>` where `<page_table>` specifies the page table to be viewed - please see the architecture specific documentation of these two commands in the “Processor Architecture Manuals”.

**NOTE:** The error message "INVALID COMBINATION" will be shown if a Linux-related MMU format such as LINUX, LINUXBIG, ... is specified without a previous `SYStem.Option.MMUSpaces ON` command. Linux page table handling requires space IDs to be enabled in TRACE32.

**See also**
- MMU
- TRANSlation.COMMON
- TRANSlation.TlbAutoScan
- TRANSlation.Create
- TRANSlation.TlbAutoScan
- MMU.DEFAULTPT()
- MMU.FORMAT()
- SYStem.Option MMUSPACES
- TRANSlation.COMMON
- TRANSlation.TableWalk
- MMU.view
- TRANSlation.Create
- TRANSlation.List
- MMU.FORMAT()
View all translation information related to an address

Format:  **MMU.INFO <address>**

Displays all translation information related to a physical address. If the address is a logical address, TRACE32 first translates it into a physical address. The information contains:

- All cache lines that cache the physical address, including both instruction and data cache.
- All TLB entries that contain translation rules for the physical address.
- All mmu entries that contain translation rules for the physical address (or all pages mapped to the given physical address), including both the task and kernel MMU entries.

See also
- **MMU**
- **MMU.view**
- **sYmbol.INFO**

▲ ‘Release Information’ in ‘Release History’

---

**MMU.ITLB**

Format:  **MMU.ITLB <address> <address>**

tbd.

See also
- **MMU**
- **MMU.view**

▲ ‘Release Information’ in ‘Release History’
**MMU.ITLBSCAN**

Format:  
```
MMU.ITLBSCAN <address> <address>
```

tbd.

**See also**
- MMU
- MMU.view

**MMU.PageTable**

Handles MMU table for the current process

Format:  
```
MMU.PageTable.dump [<address> | <range>]
MMU.PageTable.List [<address> | <range>]
MMU.PageTable.SCAN [<address> | <range>]
```

- **dump**  
  Show the current processor/target MMU table.

- **List**  
  Show the current debugger MMU table.

- **SCAN**  
  Scan the current processor/target MMU table into the debugger MMU table.

**See also**
- MMU
- MMU.view

**MMU.PTE**

Display MMU PTE entries

Format:  
```
MMU.PTE
MMU.PTE <range>
MMU.PTE <address>
```

Display MMU page table entries. If called without parameters, the complete page table will be displayed.

If the command is called with either an address range or an explicit address, page table entries will only be displayed, if their **logical** address matches with the given parameter.

**See also**
- MMU
- MMU.view
**MMU.PTESCAN**

Load MMU PTE table from probe to the debugger's internal MMU translation table. If called without parameters, the complete page table will be loaded.

If the command is called with either an address range or an explicit address, page table entries will only be loaded, if their *logical* address matches with the given parameter.

---

**MMU.SCAN**

Scans the entries of the specified page translation table or scans the translation look-aside buffer entries (such as ITLB) into the static translation list. The result is a snapshot of the scanned table and does not reflect the fact that the table may be dynamically modified by the target's OS.

The static translation list can be displayed with **TRANSLation.List**.
Based on this information, address translation rules for TRACE32's internal debugger MMU are created. The command obviates the need to create the logical to physical address translations using the `TRANSlation.Create` command.

**NOTE:** Page tables are dynamic structures and are frequently modified by the OS. Instead of `MMU.SCAN`, use `TRANSlation.TableWalk ON` to enable the debugger table walk. This method dynamically parses the page tables on demand for every debugger address translation. The debugger table walk is faster than repetitive `MMU.SCAN` calls and ensures that the debugger address translations correspond to the current OS address translations.

The command is target processor specific. For details, see “Processor Architecture Manuals” listed in the See also block below.

---

<table>
<thead>
<tr>
<th>See also</th>
<th>See also</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>MMU</strong></td>
<td><strong>MMU.view</strong></td>
</tr>
<tr>
<td><strong>TRANSlation.TlbAutoScan</strong></td>
<td><strong>CPU specific MMU Commands</strong> in 'CPU32 and ColdFire Debugger and Trace'</td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'Andes Debugger'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'ARC Debugger'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'ARM Debugger'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'ARMv8-A Debugger'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'Beyond Debugger and Trace'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'Hexagon Debugger'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'MicroBlaze Debugger and Trace'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'MIPS Debugger and Trace'</td>
<td></td>
</tr>
<tr>
<td><strong>Command Reference: MMU Commands</strong> in 'Qoriva MPC5xxx/SPC5xx Debugger and NEXUS Trace'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'NIOS II Debugger and Trace'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'PPC400/PPC440 Debugger and Trace'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'PPC600 Family Debugger'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'MPC5xx/8xx Debugger and Trace'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'PQIII Debugger'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'QorIQ Debugger and NEXUS Trace'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'SH2, SH3 and SH4 Debugger'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'Intel x86/x64 Debugger'</td>
<td></td>
</tr>
<tr>
<td><strong>MMU</strong> in 'ICE Emulator for MC68020/30'</td>
<td></td>
</tr>
<tr>
<td><strong>MMU</strong> in 'ICE Emulator for MC68040/60'</td>
<td></td>
</tr>
<tr>
<td><strong>MMU</strong> in 'ICE Emulator for 386/486'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'ARM and XSCALE Monitor'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'TRACE32 pdebug Target Server for ARM'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'SH2 Monitor'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'x386 and x486 Monitor'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'MPC56x NEXUS Debugger and Trace'</td>
<td></td>
</tr>
<tr>
<td><strong>Release Information</strong> in 'Release History'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'Simulator for 68K/ColdFire'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'Simulator for ARM and XSCALE'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'Simulator for MIPS'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'Simulator for NIOS-II'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'Simulator for PowerPC'</td>
<td></td>
</tr>
<tr>
<td><strong>CPU specific MMU Commands</strong> in 'Simulator for SuperH'</td>
<td></td>
</tr>
</tbody>
</table>

©1989-2015 Lauterbach GmbH
MMU.Set

Set MMU registers or tables

Sets a physical MMU register or table. Note that this command is not available on all probes.
tbd.

See also

- MMU
- MMU.view
- 'CPU specific MMU Commands' in 'Hexagon Debugger'
- 'CPU specific MMU Commands' in 'PPC600 Family Debugger'
- 'MMU' in 'ICE Emulator for MC68020/30'
- 'MMU' in 'ICE Emulator for MC68040/60'

Format 1:  MMU.Set [<registername> [<value>]]

Format 2:  MMU.Set <table>
            MMU.<table>.Set (deprecated)

<table>  TLB (e.g. MIPS)
          TLB0 | TLB1 (e.g. for MPC54xx, MPC85xx)
MMU.TaskPageTable

Handles MMU table for a specific process

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMU.TaskPageTable.dump [&lt;address&gt;</td>
<td>&lt;range&gt;]</td>
</tr>
<tr>
<td>MMU.TaskPageTable.List [&lt;address&gt;</td>
<td>&lt;range&gt;]</td>
</tr>
<tr>
<td>MMU.TaskPageTable.SCAN [&lt;address&gt;</td>
<td>&lt;range&gt;]</td>
</tr>
</tbody>
</table>

**dump**

Show the MMU table for the defined process based on the translation tables hold by the operating system.

**List**

Show the MMU table for the defined process based on the information in the debugger MMU.

**SCAN**

Scan the MMU table of the defined process based on the translation tables hold by the operating system into the debugger MMU table.

```plaintext
MMU.TaskPageTable.SCAN "hello" ; scan the MMU table for the process "hello"
MMU.TaskPageTable.SCAN 0x12:0 ; the process is specified by the space-id
```

**See also**

- MMU
- MMU.view

---

**MMU.TDUMP**

tbd.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMU.TDUMP &lt;address&gt; &lt;address&gt;</td>
<td>tbd.</td>
</tr>
</tbody>
</table>

tbd.

**See also**

- MMU
- MMU.view

©1989-2015 Lauterbach GmbH
Display MMU TLB entries

**Format:**

<table>
<thead>
<tr>
<th>MMU.TLB</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMU.TLB.TLB0</td>
</tr>
<tr>
<td>MMU.TLB.TLB1</td>
</tr>
<tr>
<td>MMU.TLB ITLB</td>
</tr>
</tbody>
</table>

Displays a table of the CPU's TLB entries.

**See also**

- [MMU](#) [MMU.view](#)
- 'CPU specific MMU commands' in 'MPC5xx/8xx Debugger and Trace'
- 'CPU specific MMU Commands' in 'PWRefficient Debugger'
- 'CPU specific MMU commands' in 'ICE Emulator for PowerPC'

### MMU.TLBINIT

**tbd.**

**Format:**

`MMU.TLBINIT <address> <address>`

**tbd.**

**See also**

- [MMU](#) [MMU.view](#)

### MMU.TLBRERESET

**tbd.**

**Format:**

`MMU.TLBRERESET <address> <address>`

**tbd.**

**See also**

- [MMU.view](#)
**MMU.TLBSCAN**

Load MMU TLB table from probe

Format:

- `MMU.TLBSCAN`
- `MMU.TLBSCAN.TLB0`
- `MMU.TLBSCAN.TLB1`

Loads the CPU's TLB table into the debugger and uses this data to configure the internal address translation.

See also

- MMU
- MMU.view
- 'CPU specific MMU commands' in 'MPC5xx/8xx Debugger and Trace'
- 'CPU specific MMU commands' in 'ICE Emulator for PowerPC'

---

**MMU.TLBSET**

Set MMU TLB entry

Format:

- `MMU.TLBSET <index> <value1> <value2> <value2>`
- `MMU.TLBSET.TLB0 <index> <value1> <value2> <value2>`
- `MMU.TLBSET.TLB1 <index> <value1> <value2> <value2>`

Sets an MMU TLB entry.

See also

- MMU
- MMU.view
**Transparent banking area**

The current bank (in the BNK register) is used, when an access is made inside this address range. The following example show the usage for a banked system:

```plaintext
sYmbol.RESet
MMU.RESet

; bank 0
TRANSLation.Create 0x1000000--0x100ffff a:0x00000--0x0ffff

; bank 1
TRANSLation.Create 0x1010000--0x101ffff a:0x10000--0x1ffff
MMU.TRANS 0x0--0x0ffff
TRANSLation.ON
Data.LOAD.u example.dbg /NosYmbol

; shift symbols to logical address at 1000000
sYmbol.RELOCate c:0x1000000
```

Any access to locations 0..0ffffH will access the same data as the CPU in the current bank, i.e. a variable pointing to a location, while bank1 is active, will show the data in this bank. To separate the logical addresses in bank 0 from the transparent range, all addresses get an offset of 1000000H. After load the symbols are shifted by 1000000.

### See also
- **MMU**
- **MMU.view**

---

**MMU.TSCAN**

tbd.

**Format:**

```
MMU.TSCAN <address> <address>
```

tbd.

### See also
- **MMU**
- **MMU.view**
MMU.UTLB

Format: **MMU.UTLB** `<address> <address>`

tbd.

See also
- MMU
- MMU.view

---

MMU.UTLBSCAN

Format: **MMU.UTLBSCAN** `<address> <address>`

tbd.

See also
- MMU
- MMU.view

---

MMU.view

View MMU registers

Format: **MMU.view**

This command displays all MMU registers (Not available for all probes).

See also
- MMU
- MMU.DUMP
- MMU.UTLB
- MMU.UTLBSCAN
- MMU.LINEAR()
- MMU.INFO
- MMU.PTESCAN
- MMU.PAGESCAN
- MMU.UTLB
- MMU.UTLBSCAN
- MMU.LINEAR()
- MMU.INFO
- MMU.PTESCAN
- MMU.PAGESCAN
- MMU.UTLB
- MMU.UTLBSCAN
- MMU.LINEAR()
- MMU.INFO
- MMU.PTESCAN
- MMU.PAGESCAN
- MMU.UTLB
- MMU.UTLBSCAN
- MMU.LINEAR()
- MMU.INFO
- MMU.PTESCAN
- MMU.PAGESCAN
- MMU.UTLB
- MMU.UTLBSCAN
- MMU.LINEAR()
- MMU.INFO
- MMU.PTESCAN
- MMU.PAGESCAN
- MMU.UTLB
- MMU.UTLBSCAN
- MMU.LINEAR()
- MMU.INFO
- MMU.PTESCAN
- MMU.PAGESCAN
- MMU.UTLB
- MMU.UTLBSCAN
- MMU.LINEAR()
- MMU.INFO
- MMU.PTESCAN
- MMU.PAGESCAN
- MMU.UTLB
- MMU.UTLBSCAN
- MMU.LINEAR()
- MMU.INFO
- MMU.PTESCAN
- MMU.PAGESCAN
- MMU.UTLB
- MMU.UTLBSCAN
- MMU.LINEAR()
MMX registers (MultiMedia eXtension)

See also
- MMX.Init
- MMX.OFF
- MMX.ON
- MMX.Set
- MMX.view

'MMx()'

▲ ‘Release Information’ in ‘Release History’

---

**MMX.Init**

Initialize MMX registers

Format:

```
MMX.Init
MMX.RESet (deprecated)
```

Sets all MMX registers to zero.

See also
- MMX
- MMX.view

---

**MMX.OFF**

Inhibit MMX accesses by the debugger

Format:

```
MMX.OFF
```

Inhibits accesses to the MMX by the debugger. Usually required until the MMX is on.

See also
- MMX
- MMX.view
**MMX.ON**

Permit MMX accesses by the debugger (default).

**MMX.Set**

Modify MMX registers.

**Format:**

```
MMX.ON
```

Permits accesses to the MMX by the debugger (default).

**See also**

- MMX
- MMX.view

**Format:**

```
MMX.Set <register> <value> [/<option>]
```

Modifies the MMX registers. For a description of the <options>, see Register.view.

**See also**

- MMX
- MMX.view
Opens an MMX register window. For a description of the <options>, see Register.view.

Format: **MMX.view** [/<options>]

See also

- MMX
- MMX.Init
- MMX.OFF
- MMX.ON
- MMX.Set
- 'Release Information' in 'Release History'
Switching the display mode of the sources.

**Mode**

### Emulation mode

**Format:**  
```
Mode[.<mode>]
```

**<mode>:**  
```
switch | Asm | Hll | Mix
```

Realtime emulation hardware support executes assembler steps in assembler mode and mixed mode; in high-level language mode only high-level language steps are executed.

- **switch**  
  Toggles only between **Mix** and **Hll**.
  Using just `Mode` without `.switch` also toggles only between **Mix** and **Hll**.

- **Asm**  
  In assembler mode the disassembled memory contents is displayed only without the source test.

- **Hll**  
  In HLL mode the source text is displayed without memory dump. Emulation takes place in HLL mode. Asynchronous breaks stop at the next HLL line.

- **Mix**  
  In mixed mode both information types are displayed back-to-back ([Data.List command]). Emulation takes place at assembler level.

**See also**

- [List.Asm](#)  
- [List.Hll](#)  
- [List.Mix](#)  
- [DEBUGMODE()](#)

- ‘Basics’ in ‘ICE Emulator for MC68040/60’
- ‘Basics’ in ‘ICE Emulator for 68HC11’
- ‘Program and Data Memory’ in ‘ICE User's Guide’
- ‘Realtime Emulation’ in ‘ICE User's Guide’
### Appendix - `<format>` Options of MMU.FORMAT

#### `<format>` Options for Andes:

<table>
<thead>
<tr>
<th><code>&lt;format&gt;</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard format defined by the CPU</td>
</tr>
<tr>
<td>LINUX</td>
<td>Standard format used by Linux</td>
</tr>
<tr>
<td>LINUX26</td>
<td>tbd.</td>
</tr>
</tbody>
</table>

#### `<format>` Options for ARC:

<table>
<thead>
<tr>
<th><code>&lt;format&gt;</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard format defined by the CPU</td>
</tr>
<tr>
<td>LINUX</td>
<td>Standard format used by Linux</td>
</tr>
<tr>
<td>LINUX26</td>
<td>tbd.</td>
</tr>
</tbody>
</table>

#### `<format>` Options for ARM:

<table>
<thead>
<tr>
<th><code>&lt;format&gt;</code></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard format defined by the CPU</td>
</tr>
<tr>
<td>LINUX</td>
<td>Standard format used by Linux</td>
</tr>
<tr>
<td>LINUX64</td>
<td>tbd.</td>
</tr>
<tr>
<td>LINUX26</td>
<td>tbd.</td>
</tr>
<tr>
<td>LINUXEXT</td>
<td>tbd.</td>
</tr>
<tr>
<td>LINUXSWAP</td>
<td>Linux &lt;= 2.6.37 with configured swap space</td>
</tr>
<tr>
<td>LINUXSWAP3</td>
<td>Linux &gt;= 2.6.38 with configured swap space</td>
</tr>
<tr>
<td>TINY</td>
<td>tbd.</td>
</tr>
<tr>
<td>QNX</td>
<td>QNX standard format</td>
</tr>
<tr>
<td>WINCE5</td>
<td>Format used by Windows CE5</td>
</tr>
<tr>
<td>WINCE6</td>
<td>Format used by Windows CE6 / EC7 / EC2013</td>
</tr>
<tr>
<td>SYMBIAN</td>
<td>Format used by Symbian</td>
</tr>
</tbody>
</table>
### <format> Options for BEYOND:

<table>
<thead>
<tr>
<th>&lt;format&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard format defined by the CPU</td>
</tr>
<tr>
<td>LINUX</td>
<td>Standard format used by Linux</td>
</tr>
<tr>
<td>LINUX26</td>
<td>tbd.</td>
</tr>
</tbody>
</table>

### <format> Options for x86:

<table>
<thead>
<tr>
<th>&lt;format&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard format derived from the CPU state</td>
</tr>
<tr>
<td>P32</td>
<td>32bit format with 2 page table levels</td>
</tr>
<tr>
<td>PAE</td>
<td>Format with 3 page table levels</td>
</tr>
<tr>
<td>PAE64</td>
<td>64bit format with 3 page table levels</td>
</tr>
<tr>
<td>LINUX64</td>
<td>tbd.</td>
</tr>
</tbody>
</table>

### <format> Options for MicroBlaze:

<table>
<thead>
<tr>
<th>&lt;format&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard format defined by the CPU</td>
</tr>
<tr>
<td>LINUX</td>
<td>Standard format used by Linux</td>
</tr>
<tr>
<td>LINUX26</td>
<td>tbd.</td>
</tr>
</tbody>
</table>

### <format> Options for Motorola Coldfire/68k:

<table>
<thead>
<tr>
<th>&lt;format&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard format defined by the CPU</td>
</tr>
<tr>
<td>LINUX</td>
<td>Standard format used by Linux</td>
</tr>
</tbody>
</table>
### <format> Options for MIPS:

<table>
<thead>
<tr>
<th>&lt;format&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard format defined by the CPU</td>
</tr>
<tr>
<td>LINUX32</td>
<td>Linux 32bit</td>
</tr>
<tr>
<td>LINUX32RIXI</td>
<td>Linux 32bit with RI/XI bits</td>
</tr>
<tr>
<td>LINUX32R4K</td>
<td>tbd.</td>
</tr>
<tr>
<td>LINUX32P16</td>
<td>Linux 32bit, pagesize 16kB</td>
</tr>
<tr>
<td>LINUX32P16R41</td>
<td>Linux 32bit, pagesize 16kB, used on MIPS32 R2 or R6</td>
</tr>
<tr>
<td>LINUX64</td>
<td>Linux 64bit</td>
</tr>
<tr>
<td>LINUX64P64</td>
<td>tbd.</td>
</tr>
<tr>
<td>LINUX64P64LT</td>
<td>tbd.</td>
</tr>
<tr>
<td>LINUX64RIXI</td>
<td>Linux 64bit with RI/XI bits</td>
</tr>
<tr>
<td>LINUX64HTLB</td>
<td>tbd.</td>
</tr>
<tr>
<td>LINUX64HTLBP16</td>
<td>tbd.</td>
</tr>
<tr>
<td>LINUXBIG</td>
<td>Linux 32bit with 64bit PTEs on MIPS32</td>
</tr>
<tr>
<td>LINUXBIG64</td>
<td>Linux 32bit with 64bit PTEs on MIPS64</td>
</tr>
<tr>
<td>WINCE6</td>
<td>Format used by Windows CE6</td>
</tr>
</tbody>
</table>

### <format> Options for NIOS:

<table>
<thead>
<tr>
<th>&lt;format&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard format defined by the CPU</td>
</tr>
<tr>
<td>LINUX</td>
<td>Standard format used by Linux</td>
</tr>
<tr>
<td>LINUX26</td>
<td>tbd.</td>
</tr>
</tbody>
</table>

### <format> Options for PowerPC:

<table>
<thead>
<tr>
<th>&lt;format&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>VXWORKS.E500</td>
<td>VxWorks specific format for PowerPC e500 core</td>
</tr>
<tr>
<td>VXWORKS.E6500</td>
<td>VxWorks specific format for PowerPC e6500 core</td>
</tr>
<tr>
<td>STD</td>
<td>Standard format defined by the CPU</td>
</tr>
<tr>
<td>LINUX</td>
<td>Standard format used by Linux</td>
</tr>
<tr>
<td>&lt;format&gt;</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>LINUX26</td>
<td>Linux format with physical table pointers</td>
</tr>
<tr>
<td>LINUXEXT</td>
<td>Linux with 64bit PTEs, no e500 core</td>
</tr>
<tr>
<td>LINUXE5</td>
<td>Linux with 64bit PTEs, e500 core</td>
</tr>
<tr>
<td>LINUX64_E6</td>
<td>tbd.</td>
</tr>
<tr>
<td>LYNXOS</td>
<td>LynxOS format, virtual table pointers</td>
</tr>
<tr>
<td>LYNXOSPHYS</td>
<td>LynxOS format, physical table pointers</td>
</tr>
<tr>
<td>QNX</td>
<td>QNX standard format</td>
</tr>
<tr>
<td>QNXBIG</td>
<td>QNX format with 64bit table entries</td>
</tr>
<tr>
<td>DEOS</td>
<td>tbd.</td>
</tr>
<tr>
<td>DEOS64</td>
<td>tbd.</td>
</tr>
<tr>
<td>PIKEOS</td>
<td>PikeOS standard format</td>
</tr>
<tr>
<td>PIKEOSE5</td>
<td>PikeOS format on e500 cores</td>
</tr>
<tr>
<td>OSE</td>
<td>OSE format for load modules</td>
</tr>
<tr>
<td>VX653</td>
<td>tbd.</td>
</tr>
</tbody>
</table>
### <format> Options for Hexagon QDSP6:

<table>
<thead>
<tr>
<th>&lt;format&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TLB</td>
<td>Reads TLB entries to translate logical addresses</td>
</tr>
<tr>
<td>L4</td>
<td>tbd.</td>
</tr>
<tr>
<td>BLAST</td>
<td>QURT Page Table format</td>
</tr>
<tr>
<td>QURT V2</td>
<td>QuRT Page Table version 2</td>
</tr>
</tbody>
</table>

### <format> Options for SH4:

<table>
<thead>
<tr>
<th>&lt;format&gt;</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>STD</td>
<td>Standard format defined by the CPU</td>
</tr>
<tr>
<td>LINUX</td>
<td>Standard format used by Linux</td>
</tr>
<tr>
<td>LINUX 26</td>
<td>tbd.</td>
</tr>
<tr>
<td>LINUX EXTP64</td>
<td>Linux with extended TLBs (3 page table levels, 64bit PTEs)</td>
</tr>
<tr>
<td>QNX</td>
<td>QNX standard format</td>
</tr>
<tr>
<td>WINCE 6</td>
<td>Format used by Windows CE6</td>
</tr>
</tbody>
</table>