IAR Embedded Workbench®

C-SPY® Debugging Guide

for Advanced RISC Machines Ltd’s

ARM® cores

UCSARM-5
COPYRIGHT NOTICE
© 1999–2012 IAR Systems AB.

No part of this document may be reproduced without the prior written consent of IAR Systems AB. The software described in this document is furnished under a license and may only be used or copied in accordance with the terms of such a license.

DISCLAIMER
The information in this document is subject to change without notice and does not represent a commitment on any part of IAR Systems. While the information contained herein is assumed to be accurate, IAR Systems assumes no responsibility for any errors or omissions.

In no event shall IAR Systems, its employees, its contractors, or the authors of this document be liable for special, direct, indirect, or consequential damage, losses, costs, charges, claims, demands, claim for lost profits, fees, or expenses of any nature or kind.

TRADEMARKS
IAR Systems, IAR Embedded Workbench, C-SPY, visualSTATE, The Code to Success, IAR KickStart Kit, I-jet, IAR, and the logotype of IAR Systems are trademarks or registered trademarks owned by IAR Systems AB.

Microsoft and Windows are registered trademarks of Microsoft Corporation.

ARM and Thumb are registered trademarks of Advanced RISC Machines Ltd. EmbeddedICE is a trademark of Advanced RISC Machines Ltd. OCDemon is a trademark of Macraigor Systems LLC. mC/OS-II is a trademark of Micrimm, Inc. CMX-RTX is a trademark of CMX Systems, Inc. ThreadX is a trademark of Express Logic. RTXC is a trademark of Quadros Systems. Fusion is a trademark of Unicoi Systems.

Adobe and Acrobat Reader are registered trademarks of Adobe Systems Incorporated.

All other product names are trademarks or registered trademarks of their respective owners.

EDITION NOTICE
Fifth edition: October 2012
Part number: UCSARM-5

This guide applies to version 6.5x of IAR Embedded Workbench® for Advanced RISC Machines Ltd’s ARM core family.

Internal reference: M12, Too6.5, IMAE.
Brief contents

Tables ......................................................................................................................  21
Figures ....................................................................................................................  23
Preface ....................................................................................................................  29
The IAR C-SPY Debugger ....................................................................................  35
Getting started using C-SPY ..............................................................  51
Executing your application ...........................................................................  73
Variables and expressions ..................................................................................  91
Breakpoints ........................................................................................................  119
Memory and registers .........................................................................................  153
Trace ......................................................................................................................  187
Profiling ...............................................................................................................  239
Power debugging .................................................................................................  251
Code coverage .....................................................................................................  265
Interrupts ..............................................................................................................  271
C-SPY macros .....................................................................................................  293
The C-SPY Command Line Utility—cspybat ...........................................  349
Debugger options .................................................................................................  389
Additional information on C-SPY drivers ....................................................  421
Flash loaders .......................................................................................................  437
Index .....................................................................................................................  443
Contents

Tables ...................................................................................................................... 21
Figures .................................................................................................................... 23
Preface .................................................................................................................... 29

Who should read this guide .............................................................................. 29
  Required knowledge .......................................................................................... 29

What this guide contains ................................................................................. 29

Other documentation .......................................................................................... 30
  User and reference guides .................................................................................. 31
  The online help system ...................................................................................... 31
  Web sites ........................................................................................................... 32

Document conventions ........................................................................................ 32
  Typographic conventions ................................................................................... 32
  Naming conventions .......................................................................................... 33

The IAR C-SPY Debugger ................................................................................... 35

Introduction to C-SPY ....................................................................................... 35
  An integrated environment .................................................................................. 35
  General C-SPY debugger features ..................................................................... 36
  RTOS awareness ................................................................................................. 37

Debugger concepts ............................................................................................... 38
  C-SPY and target systems .................................................................................. 38
  The debugger ...................................................................................................... 39
  The target system ............................................................................................... 39
  The application ................................................................................................... 39
  C-SPY debugger systems .................................................................................. 39
  The ROM-monitor program ............................................................................. 40
  Third-party debuggers ....................................................................................... 40
  C-SPY plugin modules ..................................................................................... 40

C-SPY drivers overview ..................................................................................... 41
  Differences between the C-SPY drivers .......................................................... 42
The IAR C-SPY Simulator ........................................... 43
  Simulator features .............................................. 43

The C-SPY hardware debugger drivers ................... 43
  Communication overview .................................... 44
  Hardware installation .......................................... 46
  USB driver installation ........................................ 47

Getting started using C-SPY ....................................... 51

Setting up C-SPY ..................................................... 51
  Setting up for debugging ..................................... 51
  Executing from reset ......................................... 52
  Using a setup macro file ...................................... 53
  Selecting a device description file ....................... 53
  Loading plugin modules ....................................... 54

Starting C-SPY ......................................................... 54
  Starting a debug session ..................................... 54
  Loading executable files built outside of the IDE .... 54
  Starting a debug session with source files missing ... 55
  Loading multiple images ..................................... 56

Adapting for target hardware .................................. 56
  Memory configuration ....................................... 57
  Modifying a device description file ....................... 57
  Initializing target hardware before C-SPY starts ...... 58
  Remapping memory .......................................... 58

An overview of the debugger startup ....................... 59
  Debugging code in flash ..................................... 60
  Debugging code in RAM ...................................... 61

Running example projects ....................................... 61
  Running an example project ................................. 62

Reference information on starting C-SPY ................. 63
  C-SPY Debugger main window ............................. 63
  Images window ............................................... 69
  Get Alternative File dialog box ......................... 71
Executing your application ................................................................. 73

**Introduction to application execution** ........................................... 73
  Briefly about application execution .............................................. 73
  Source and disassembly mode debugging ...................................... 73
  Single stepping ............................................................................. 74
  Stepping speed ............................................................................. 76
  Running the application ................................................................ 77
  Highlighting .................................................................................. 78
  Call stack information ................................................................. 78
  Terminal input and output ............................................................ 79
  Debug logging .............................................................................. 79

**Reference information on application execution** ......................... 80
  Disassembly window ..................................................................... 80
  Call Stack window ........................................................................ 84
  Terminal I/O window .................................................................... 86
  Terminal I/O Log File dialog box .................................................. 87
  Debug Log window ........................................................................ 88
  Log File dialog box ....................................................................... 89
  Autostep settings dialog box ......................................................... 90

**Variables and expressions** ........................................................... 91

**Introduction to working with variables and expressions** ............... 91
  Briefly about working with variables and expressions ................. 91
  C-SPY expressions ....................................................................... 92
  Limitations on variable information ............................................. 94

**Working with variables and expressions** ..................................... 95
  Using the windows related to variables and expressions ............... 96
  Viewing assembler variables ......................................................... 96
  Getting started using data logging ................................................. 97
  Getting started using event logging .............................................. 98

**Reference information on working with variables and expressions** ................................................................................. 99
  Auto window ................................................................................. 100
  Locals window ............................................................................. 100
Breakpoints

Introduction to setting and using breakpoints

Reasons for using breakpoints

Briefly about setting breakpoints

Breakpoint types

Breakpoint icons

Breakpoints in the C-SPY simulator

Breakpoints in the C-SPY hardware drivers

Breakpoint consumers

Breakpoints options

Breakpoints on exception vectors

Setting breakpoints in __ramfunc declared functions

Setting breakpoints

Various ways to set a breakpoint

Toggling a simple code breakpoint

Setting breakpoints using the dialog box

Setting a data breakpoint in the Memory window

Setting breakpoints using system macros

Setting a breakpoint on an exception vector

Useful breakpoint hints

Reference information on breakpoints

Breakpoints window
Breakpoint Usage window ............................................................... 133
Code breakpoints dialog box ........................................................... 134
JTAG Watchpoints dialog box ........................................................ 136
Log breakpoints dialog box ............................................................ 139
Data breakpoints dialog box .......................................................... 140
Data Log breakpoints dialog box .................................................... 143
Breakpoints options ...................................................................... 145
Immediate breakpoints dialog box ................................................ 147
Vector Catch dialog box ............................................................... 148
Enter Location dialog box .............................................................. 149
Resolve Source Ambiguity dialog box ....................................... 150

Memory and registers ..................................................................... 153

Introduction to monitoring memory and registers ...................... 153
Briefly about monitoring memory and registers .......................... 153
C-SPY memory zones ................................................................. 155
Stack display .............................................................................. 155
Memory access checking ............................................................. 156

Monitoring memory and registers .............................................. 156
Defining application-specific register groups ......................... 157

Reference information on memory and registers ...................... 158
Memory window ................................................................. 159
Memory Save dialog box .......................................................... 163
Memory Restore dialog box ...................................................... 164
Fill dialog box .......................................................................... 164
Symbolic Memory window ......................................................... 166
Stack window ........................................................................... 168
Register window ........................................................................ 172
SFR Setup window ................................................................. 174
Edit SFR dialog box ............................................................... 177
Memory Configuration dialog box ............................................. 179
Edit Memory Range dialog box ................................................ 182
Memory Access Setup dialog box ............................................. 184
Edit Memory Access dialog box ................................................ 186
Introduction to using trace .............................................................. 187
  Reasons for using trace ............................................................... 187
  Briefly about trace ...................................................................... 188
  Requirements for using trace ...................................................... 189

Collecting and using trace data ...................................................... 190
  Getting started with ETM trace .................................................. 190
  Getting started with SWO trace .................................................. 191
  Setting up concurrent use of ETM and SWO .............................. 192
  Trace data collection using breakpoints ..................................... 192
  Searching in trace data .............................................................. 193
  Browsing through trace data ...................................................... 193

Reference information on trace ..................................................... 194
  ETM Trace Settings dialog box (I-jet/JTAGjet) ......................... 195
  ETM Trace Settings dialog box (J-Link/J-Trace) ....................... 197
  SWO Trace Window Settings dialog box ................................. 199
  SWO Configuration dialog box ................................................. 201
  Trace window ........................................................................... 205
  Function Trace window ............................................................. 209
  Timeline window ....................................................................... 209
  Viewing Range dialog box ........................................................ 218
  Trace Start breakpoints dialog box ......................................... 220
  Trace Stop breakpoints dialog box ......................................... 221
  Trace Start breakpoints dialog box (I-jet/JTAGjet) .................... 222
  Trace Stop breakpoints dialog box (I-jet/JTAGjet) .................... 224
  Trace Filter breakpoints dialog box (I-jet/JTAGjet) ................. 226
  Trace Start breakpoints dialog box (J-Link/J-Trace) ............... 227
  Trace Stop breakpoints dialog box (J-Link/J-Trace) ............... 230
  Trace Filter breakpoints dialog box (J-Link/J-Trace) ............... 232
  Trace Expressions window ...................................................... 235
  Find in Trace dialog box ........................................................... 236
  Find in Trace window ............................................................... 237
  Trace Save dialog box ............................................................... 238
Profiling ................................................................................................................ 239

Introduction to the profiler ........................................................................... 239
Reasons for using the profiler ................................................................. 239
Briefly about the profiler ......................................................................... 240
Requirements for using the profiler ....................................................... 241

Using the profiler ...................................................................................... 241
Getting started using the profiler on function level .............................. 242
Getting started using the profiler on instruction level ......................... 242
Selecting a time interval for profiling information .............................. 243

Reference information on the profiler ................................................ 244
Function Profiler window ................................................................. 245

Power debugging ........................................................................................ 251

Introduction to power debugging ................................................... 251
Reasons for using power debugging .................................................. 251
Briefly about power debugging .......................................................... 251
Requirements for power debugging .................................................... 252

Optimizing your source code for power consumption ........................ 253
Waiting for device status ....................................................................... 253
Software delays ..................................................................................... 253
DMA versus polled I/O .......................................................................... 253
Low-power mode diagnostics .............................................................. 254
CPU frequency ....................................................................................... 254
Detecting mistakenly unattended peripherals .................................... 254
Peripheral units in an event-driven system .......................................... 255
Finding conflicting hardware setups ..................................................... 256
Analog interference ................................................................................ 256

Debugging in the power domain ........................................................... 257
Displaying the application’s power profile and analyzing
the result .................................................................................................. 258
Detecting unexpected power usage during application execution .... 259

Reference information on power debugging ................................... 259
Power Log Setup window ................................................................. 260
Power Log window ............................................................................... 262
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code coverage</td>
<td>265</td>
</tr>
<tr>
<td>Introduction to code coverage</td>
<td>265</td>
</tr>
<tr>
<td>Reasons for using code coverage</td>
<td>265</td>
</tr>
<tr>
<td>Briefly about code coverage</td>
<td>265</td>
</tr>
<tr>
<td>Requirements for using code coverage</td>
<td>265</td>
</tr>
<tr>
<td>Reference information on code coverage</td>
<td>266</td>
</tr>
<tr>
<td>Code Coverage window</td>
<td>266</td>
</tr>
<tr>
<td>Interrupts</td>
<td>271</td>
</tr>
<tr>
<td>Introduction to interrupts</td>
<td>271</td>
</tr>
<tr>
<td>Briefly about interrupt logging</td>
<td>271</td>
</tr>
<tr>
<td>Briefly about the interrupt simulation system</td>
<td>272</td>
</tr>
<tr>
<td>Interrupt characteristics</td>
<td>273</td>
</tr>
<tr>
<td>Interrupt simulation states</td>
<td>274</td>
</tr>
<tr>
<td>C-SPY system macros for interrupt simulation</td>
<td>275</td>
</tr>
<tr>
<td>Target-adapting the interrupt simulation system</td>
<td>276</td>
</tr>
<tr>
<td>Using the interrupt system</td>
<td>276</td>
</tr>
<tr>
<td>Simulating a simple interrupt</td>
<td>276</td>
</tr>
<tr>
<td>Simulating an interrupt in a multi-task system</td>
<td>278</td>
</tr>
<tr>
<td>Getting started using interrupt logging</td>
<td>279</td>
</tr>
<tr>
<td>Reference information on interrupts</td>
<td>279</td>
</tr>
<tr>
<td>Interrupt Setup dialog box</td>
<td>280</td>
</tr>
<tr>
<td>Edit Interrupt dialog box</td>
<td>282</td>
</tr>
<tr>
<td>Forced Interrupt window</td>
<td>284</td>
</tr>
<tr>
<td>Interrupt Status window</td>
<td>285</td>
</tr>
<tr>
<td>Interrupt Log window</td>
<td>287</td>
</tr>
<tr>
<td>Interrupt Log Summary window</td>
<td>291</td>
</tr>
<tr>
<td>C-SPY macros</td>
<td>293</td>
</tr>
<tr>
<td>Introduction to C-SPY macros</td>
<td>293</td>
</tr>
<tr>
<td>Reasons for using C-SPY macros</td>
<td>293</td>
</tr>
<tr>
<td>Briefly about using C-SPY macros</td>
<td>294</td>
</tr>
<tr>
<td>Briefly about setup macro functions and files</td>
<td>294</td>
</tr>
<tr>
<td>Briefly about the macro language</td>
<td>295</td>
</tr>
</tbody>
</table>
Using C-SPY macros ........................................................................... 296
  Registering C-SPY macros—an overview ........................................ 296
  Executing C-SPY macros—an overview ........................................... 296
Registering and executing using setup macros and setup files ........ 298
Executing macros using Quick Watch ............................................. 299
Executing a macro by connecting it to a breakpoint ...................... 300
Reference information on the macro language ............................. 301
  Macro functions ........................................................................... 302
  Macro variables ........................................................................... 302
  Macro strings ............................................................................. 303
  Macro statements ........................................................................ 303
  Formatted output ........................................................................ 304
Reference information on reserved setup macro function names .................................................................................................. 306
  Reference information on C-SPY system macros ..................... 307
    __cancelAllInterrupts ............................................................. 310
    __cancelInterrupt ................................................................. 310
    __clearBreak ......................................................................... 311
    __closeFile ........................................................................... 311
    __delay .................................................................................. 311
    __disableInterrupts ............................................................... 312
    __driverType .......................................................................... 312
    __emulatorSpeed ................................................................... 313
    __emulatorStatusCheckOnRead ............................................... 314
    __enableInterrupts ................................................................. 314
    __evaluate ............................................................................... 314
    __gdbserver_exec_command ................................................... 315
    __getTracePortSize ............................................................... 315
    __hwJetResetWithStrategy ....................................................... 316
    __hwReset ............................................................................ 316
    __hwResetRunToBp .................................................................. 317
    __hwResetWithStrategy .......................................................... 318
    __isBatchMode ...................................................................... 319
_jlinkExecCommand ................................................................. 319
__jtagCommand ................................................................. 320
__jtagCP15IsPresent ............................................................ 320
__jtagCP15ReadReg ............................................................. 321
__jtagCP15WriteReg ............................................................ 321
__jtagData ....................................................................... 321
__jtagRawRead ................................................................. 322
__jtagRawSync ................................................................. 323
__jtagRawWrite ................................................................. 324
__jtagResetTRST ............................................................ 325
__loadImage ................................................................. 325
__memoryRestore ............................................................ 326
__memorySave ................................................................. 327
__openFile ................................................................ 327
__orderInterrupt ......................................................... 329
__popSimulatorInterruptExecutingStack ......................... 330
__readFile ................................................................. 330
__readFileByte ............................................................. 331
__readMemory8, __readMemoryByte .................................. 331
__readMemory16 ........................................................... 331
__readMemory32 ........................................................... 332
__registerMacroFile ......................................................... 332
__resetFile ................................................................. 333
__restoreSoftwareBreakpoints ......................................... 333
__setCodeBreak ............................................................ 334
__setDataBreak ............................................................. 335
__setDataLogBreak ........................................................ 337
__setLogBreak ............................................................... 338
__setSimBreak ............................................................... 339
__setTraceStartBreak ................................................... 340
__setTraceStopBreak ..................................................... 341
__sourcePosition .......................................................... 342
__strFind ................................................................ 343
__subString ................................................................ 343
The C-SPY Command Line Utility—cspybat ........................................ 349

Using C-SPY in batch mode .......................................................... 349
- Invocation syntax ................................................................. 349
- Output ................................................................................... 350
- Using an automatically generated batch file .......................... 350

Summary of C-SPY command line options ................................. 351
- General cspybat options ....................................................... 351
- Options available for all C-SPY drivers ................................. 351
- Options available for the simulator driver ............................. 353
- Options available for the C-SPY Angel debug monitor driver .... 353
- Options available for the C-SPY GDB Server driver ............... 353
- Options available for the C-SPY IAR ROM-monitor driver ....... 353
- Options available for the C-SPY J-Link/J-Trace driver .......... 353
- Options available for the C-SPY J-Link/J-Trace driver .......... 353
- Options available for the C-SPY TI Stellaris driver ............... 354
- Options available for the C-SPY TI XDS100 driver ............... 355
- Options available for the C-SPY Macraigor driver ............... 355
- Options available for the C-SPY RDI driver ......................... 355
- Options available for the C-SPY ST-LINK driver ................. 355
- Options available for the C-SPY third-party drivers ............ 356

Reference information on C-SPY command line options ............ 356
- -B ...................................................................................... 356
- --backend ........................................................................... 356
--code_coverage_file ......................................................... 356
--cycles ................................................................. 357
--device .............................................................. 357
--disable_interrupts .................................................. 358
--download_only ...................................................... 358
--drv_catch_exceptions .............................................. 358
--drv_communication .............................................. 360
--drv_communication_log .......................................... 363
--drv_default_breakpoint ........................................... 363
--drv_reset_to_cpu_start ......................................... 364
--drv_restore_breakpoints .......................................... 365
--drv_swo_clock_setup .............................................. 365
--drv_vector_table_base ........................................... 366
--flash_loader .......................................................... 367
--gdbserv_exec_command ............................................. 367
--jet_cpu_clock ......................................................... 367
--jet_interface .......................................................... 368
--jet_ir_length .......................................................... 368
--jet_jtag_speed ........................................................ 369
--jet_power_from_probe ............................................. 369
--jet_script_file ........................................................ 370
--jet_script_reset ..................................................... 370
--jet_standard_reset ................................................. 371
--jet_swo_on_d0 .......................................................... 372
--jet_swo_prescaler ................................................... 372
--jet_swo_protocol .................................................... 373
--jet_tap_position ...................................................... 373
--jlink_device_select ................................................. 374
--jlink_exec_command ................................................ 374
--jlink_initial_speed ................................................ 375
--jlink_interface ....................................................... 375
--jlink_ir_length ....................................................... 375
--jlink_reset_strategy ................................................ 376
--jlink_script_file .................................................... 376
Debugger options .................................................................................. 389

Setting debugger options .................................................................... 389

Reference information on debugger options ..................................... 391

Setup .................................................................................................... 391
Download ............................................................................................ 392
Images .................................................................................................. 394
Plugins ................................................................................................. 395
Extra Options ........................................................................................ 396
Reference information on C-SPY hardware driver options ................................................................. 396
  Angel ........................................................................................................ 397
  GDB Server ............................................................................................ 398
  IAR ROM-monitor ................................................................................ 399
  Setup options for I-jet/JTAGjet .............................................................. 400
  JTAG/SWD options for I-jet/JTAGjet ..................................................... 403
  SWO options for I-jet/JTAGjet ............................................................... 404
  Setup options for J-Link/J-Trace ......................................................... 406
  Connection options for J-Link/J-Trace ................................................ 410
  Macraigor ............................................................................................ 412
  RDI ......................................................................................................... 414
  ST-LINK ............................................................................................... 416
  Setup options for TI Stellaris ................................................................. 417
  Setup options for TI XDS100 ............................................................... 418
  Third-Party Driver options ................................................................. 419

Additional information on C-SPY drivers ................................................................. 421

Reference information on the C-SPY simulator ...................................................... 421
  Simulator menu ........................................................................................ 421

The C-SPY GDB Server driver ........................................................................ 423
  GDB Server menu .................................................................................. 423

The C-SPY I-jet/JTAGjet driver ..................................................................... 423
  I-jet/JTAGjet menu ............................................................................... 424

The C-SPY J-Link/J-Trace driver ..................................................................... 427
  J-Link menu .......................................................................................... 427
  Live watch and use of DCC ................................................................... 429
  Terminal I/O and use of DCC ............................................................... 430

The C-SPY Macraigor driver .......................................................................... 430
  Macraigor JTAG menu .......................................................................... 430

The C-SPY RDI driver ................................................................................... 431
  RDI menu ............................................................................................... 431

The C-SPY ST-LINK driver .......................................................................... 432
  ST-LINK menu ........................................................................................ 432
Contents

19

The C-SPY TI Stellaris driver ..................................................... 433
   TI Stellaris menu .............................................................................. 433
The C-SPY TI XDS100 driver ..................................................... 434
   TI XDS100 menu ............................................................................. 434
Resolving problems ................................................................. 434
   Slow stepping speed ......................................................................... 434
Flash loaders .............................................................................. 437
   Introduction to the flash loader .................................................. 437
   Using flash loaders ....................................................................... 437
      Setting up the flash loader(s) .................................................. 437
      The flash loading mechanism ................................................. 438
Reference information on the flash loader ................................. 438
   Flash Loader Overview dialog box ........................................... 439
   Flash Loader Configuration dialog box ..................................... 440
Index ............................................................................................. 443
# Tables

1: Typographic conventions used in this guide ......................................................... 32  
2: Naming conventions used in this guide ............................................................... 33  
3: Driver differences, I-jet/JTAGjet, J-Link/J-Trace and ST-LINK ......................... 42  
4: Driver differences, other drivers ................................................................. 43  
5: Available quickstart reference information ..................................................... 52  
6: C-SPY assembler symbols expressions ............................................................ 93  
7: Handling name conflicts between hardware registers and assembler labels ...... 94  
8: C-SPY macros for breakpoints ..................................................................... 128  
9: Supported graphs in the Timeline window ..................................................... 211  
10: C-SPY driver profiling support .................................................................. 241  
11: Project options for enabling the profiler ..................................................... 242  
12: Project options for enabling code coverage ............................................... 266  
13: Timer interrupt settings ............................................................................ 278  
14: Examples of C-SPY macro variables ............................................................. 302  
15: C-SPY setup macros .................................................................................. 306  
16: Summary of system macros ....................................................................... 307  
17: __cancelInterrupt return values ............................................................... 310  
18: __disableInterrupts return values ............................................................. 312  
19: __driverType return values ........................................................................ 313  
20: __emulatorSpeed return values .................................................................. 313  
21: __enableInterrupts return values ............................................................... 314  
22: __evaluate return values ........................................................................... 315  
23: __getTracePortSize return values ............................................................. 315  
24: __hwJetResetWithStrategy return values .................................................... 316  
25: __hwReset return values .......................................................................... 317  
26: __hwResetRunToBp return values ............................................................... 317  
27: __hwResetWithStrategy return values .......................................................... 318  
28: __isBatchMode return values .................................................................... 319  
29: __jtagResetTRST return values ................................................................. 325  
30: __loadImage return values ........................................................................ 325  
31: __openFile return values .......................................................................... 328
32: __readFile return values ................................................................. 330
33: __setCodeBreak return values ......................................................... 334
34: __setDataBreak return values ......................................................... 336
35: __setDataLogBreak return values .................................................... 337
36: __setLogBreak return values ......................................................... 338
37: __setSimBreak return values ......................................................... 340
38: __setTraceStartBreak return values ............................................. 341
39: __setTraceStopBreak return values .............................................. 342
40: __sourcePosition return values ................................................... 343
41: __unloadImage return values ....................................................... 346
42: cspybat parameters ................................................................. 349
43: Options specific to the C-SPY drivers you are using .................... 389
44: Catching exceptions ................................................................. 415
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>C-SPY and target systems</td>
<td>39</td>
</tr>
<tr>
<td>2</td>
<td>C-SPY driver communication overview with a debug probe or emulator</td>
<td>44</td>
</tr>
<tr>
<td>3</td>
<td>C-SPY driver communication overview without a debug probe</td>
<td>46</td>
</tr>
<tr>
<td>4</td>
<td>Get Alternative File dialog box</td>
<td>55</td>
</tr>
<tr>
<td>5</td>
<td>Debugger startup when debugging code in flash</td>
<td>60</td>
</tr>
<tr>
<td>6</td>
<td>Debugger startup when debugging code in RAM</td>
<td>61</td>
</tr>
<tr>
<td>7</td>
<td>Example applications</td>
<td>62</td>
</tr>
<tr>
<td>8</td>
<td>Debug menu</td>
<td>65</td>
</tr>
<tr>
<td>9</td>
<td>Disassembly menu</td>
<td>67</td>
</tr>
<tr>
<td>10</td>
<td>Images window</td>
<td>69</td>
</tr>
<tr>
<td>11</td>
<td>Images window context menu</td>
<td>70</td>
</tr>
<tr>
<td>12</td>
<td>Get Alternative File dialog box</td>
<td>71</td>
</tr>
<tr>
<td>13</td>
<td>C-SPY highlighting source location</td>
<td>78</td>
</tr>
<tr>
<td>14</td>
<td>C-SPY Disassembly window</td>
<td>80</td>
</tr>
<tr>
<td>15</td>
<td>Disassembly window context menu</td>
<td>82</td>
</tr>
<tr>
<td>16</td>
<td>Call Stack window</td>
<td>84</td>
</tr>
<tr>
<td>17</td>
<td>Call Stack window context menu</td>
<td>85</td>
</tr>
<tr>
<td>18</td>
<td>Terminal I/O window</td>
<td>86</td>
</tr>
<tr>
<td>19</td>
<td>Ctrl codes menu</td>
<td>86</td>
</tr>
<tr>
<td>20</td>
<td>Input Mode dialog box</td>
<td>87</td>
</tr>
<tr>
<td>21</td>
<td>Terminal I/O Log File dialog box</td>
<td>87</td>
</tr>
<tr>
<td>22</td>
<td>Debug Log window (message window)</td>
<td>88</td>
</tr>
<tr>
<td>23</td>
<td>Debug Log window context menu</td>
<td>88</td>
</tr>
<tr>
<td>24</td>
<td>Log File dialog box</td>
<td>89</td>
</tr>
<tr>
<td>25</td>
<td>Autostep settings dialog box</td>
<td>90</td>
</tr>
<tr>
<td>26</td>
<td>Viewing assembler variables in the Watch window</td>
<td>97</td>
</tr>
<tr>
<td>27</td>
<td>Auto window</td>
<td>100</td>
</tr>
<tr>
<td>28</td>
<td>Locals window</td>
<td>100</td>
</tr>
<tr>
<td>29</td>
<td>Watch window</td>
<td>101</td>
</tr>
<tr>
<td>30</td>
<td>Watch window context menu</td>
<td>102</td>
</tr>
<tr>
<td>31</td>
<td>Live Watch window</td>
<td>103</td>
</tr>
<tr>
<td>32:</td>
<td>Statics window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>33:</td>
<td>Statics window context menu</td>
<td>..........................................................</td>
</tr>
<tr>
<td>34:</td>
<td>Macro Quicklaunch window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>35:</td>
<td>Macro Quicklaunch window context menu</td>
<td>..........................................................</td>
</tr>
<tr>
<td>36:</td>
<td>Quick Watch window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>37:</td>
<td>Watch window context menu</td>
<td>..........................................................</td>
</tr>
<tr>
<td>38:</td>
<td>Symbols window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>39:</td>
<td>Symbols window context menu</td>
<td>..........................................................</td>
</tr>
<tr>
<td>40:</td>
<td>Resolve Symbol Ambiguity dialog box</td>
<td>..........................................................</td>
</tr>
<tr>
<td>41:</td>
<td>Data Log window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>42:</td>
<td>Data Log Summary window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>43:</td>
<td>Event Log window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>44:</td>
<td>Event Log Summary window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>45:</td>
<td>Breakpoint icons</td>
<td>..........................................................</td>
</tr>
<tr>
<td>46:</td>
<td>Modifying breakpoints via the context menu</td>
<td>..........................................................</td>
</tr>
<tr>
<td>47:</td>
<td>Breakpoints window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>48:</td>
<td>Breakpoints window context menu</td>
<td>..........................................................</td>
</tr>
<tr>
<td>49:</td>
<td>Breakpoint Usage window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>50:</td>
<td>Code breakpoints dialog box</td>
<td>..........................................................</td>
</tr>
<tr>
<td>51:</td>
<td>JTAG Watchpoints dialog box</td>
<td>..........................................................</td>
</tr>
<tr>
<td>52:</td>
<td>Log breakpoints dialog box</td>
<td>..........................................................</td>
</tr>
<tr>
<td>53:</td>
<td>Data breakpoints dialog box</td>
<td>..........................................................</td>
</tr>
<tr>
<td>54:</td>
<td>Data Log breakpoints dialog box</td>
<td>..........................................................</td>
</tr>
<tr>
<td>55:</td>
<td>Breakpoints options</td>
<td>..........................................................</td>
</tr>
<tr>
<td>56:</td>
<td>Immediate breakpoints dialog box</td>
<td>..........................................................</td>
</tr>
<tr>
<td>57:</td>
<td>The Vector Catch dialog box—for ARM9/Cortex-R4 versus for Cortex-M3</td>
<td>..........................................................</td>
</tr>
<tr>
<td>58:</td>
<td>Enter Location dialog box</td>
<td>..........................................................</td>
</tr>
<tr>
<td>59:</td>
<td>Resolve Source Ambiguity dialog box</td>
<td>..........................................................</td>
</tr>
<tr>
<td>60:</td>
<td>Zones in C-SPY</td>
<td>..........................................................</td>
</tr>
<tr>
<td>61:</td>
<td>Register Filter options</td>
<td>..........................................................</td>
</tr>
<tr>
<td>62:</td>
<td>Register Filter options</td>
<td>..........................................................</td>
</tr>
<tr>
<td>63:</td>
<td>Memory window</td>
<td>..........................................................</td>
</tr>
<tr>
<td>64:</td>
<td>Memory window context menu</td>
<td>..........................................................</td>
</tr>
<tr>
<td>65:</td>
<td>Memory Save dialog box</td>
<td>..........................................................</td>
</tr>
</tbody>
</table>
Figures

- Memory Restore dialog box ................................................................. 164
- Fill dialog box .................................................................................. 164
- Symbolic Memory window ................................................................. 166
- Symbolic Memory window context menu ......................................... 167
- Stack window .................................................................................... 168
- Stack window context menu .............................................................. 171
- Register window ............................................................................... 172
- SFR Setup window ............................................................................ 174
- SFR Setup window context menu ...................................................... 176
- Edit SFR dialog box .......................................................................... 177
- Memory Configuration dialog box ...................................................... 179
- Edit Memory Range dialog box ......................................................... 182
- Memory Access Setup dialog box ....................................................... 184
- Edit Memory Access dialog box ....................................................... 186
- ETM Trace Settings dialog box (I-jet/JTAGjet) ............................. 195
- ETM Trace Settings dialog box (J-Link/J-Trace) ........................... 197
- SWO Trace Window Settings dialog box ......................................... 199
- SWO Configuration dialog box ......................................................... 201
- The Trace window ........................................................................... 205
- Function Trace window .................................................................. 209
- Timeline window with Call Stack graph .......................................... 211
- Timeline window with Data Log graph .......................................... 212
- Timeline window with Event Log graph .......................................... 213
- Timeline window with Interrupt Log graph ..................................... 214
- Timeline window with Power Log graph .......................................... 215
- Timeline window context menu (for the Call Stack Graph context menu) .... 215
- Viewing Range dialog box ............................................................. 218
- Trace Start breakpoints dialog box ................................................. 220
- Trace Stop breakpoints dialog box .................................................. 221
- Trace Start breakpoints dialog box (I-jet/JTAGjet) .......................... 222
- Trace Stop breakpoints dialog box (I-jet/JTAGjet) .......................... 224
- Trace Filter breakpoints dialog box (I-jet/JTAGjet) .......................... 226
- Trace Start breakpoints dialog box (J-Link/J-Trace) ...................... 227
- Trace Stop breakpoints dialog box (J-Link/J-Trace) ...................... 230
100: Trace Filter breakpoints dialog box .................................................... 232
101: Trace Expressions window .................................................................. 235
102: Find in Trace dialog box ...................................................................... 236
103: Find in Trace window ........................................................................... 237
104: Trace Save dialog box .......................................................................... 238
105: Instruction count in Disassembly window .............................................. 243
106: Power Graph with a selected time interval .............................................. 244
107: Function Profiler window in time-interval mode .................................... 244
108: Function Profiler window ..................................................................... 245
109: Function Profiler window context menu ............................................... 248
110: Power consumption in an event-driven system ....................................... 255
111: A noise spike recorded by an oscilloscope ............................................. 257
112: Power Log Setup window ..................................................................... 260
113: Power Setup window context menu ....................................................... 261
114: Power Log window ................................................................................ 262
115: Power Log window context menu .......................................................... 263
116: Code Coverage window ......................................................................... 266
117: Code coverage window context menu .................................................... 268
118: Simulated interrupt configuration ............................................................ 273
119: Simulation states - example 1 ................................................................. 274
120: Simulation states - example 2 ................................................................. 275
121: Interrupt Setup dialog box ..................................................................... 280
122: Edit Interrupt dialog box ......................................................................... 282
123: Forced Interrupt window ........................................................................ 284
124: Forced Interrupt window context menu ................................................. 284
125: Interrupt Status window .......................................................................... 285
126: Interrupt Log window ............................................................................ 287
127: Interrupt Log window context menu ...................................................... 290
128: Interrupt Log Summary window ............................................................. 291
129: Macro Configuration dialog box .............................................................. 297
130: Quick Watch window ............................................................................. 300
131: Debugger setup options ......................................................................... 391
132: C-SPY Download options ....................................................................... 392
133: Debugger images options ....................................................................... 394
134: Debugger plugin options ................................................................. 395
135: Debugger extra options ................................................................. 396
136: C-SPY Angel options ................................................................. 397
137: GDB Server options ................................................................. 398
138: IAR ROM-monitor options ......................................................... 399
139: I-jet/JTAGjet Setup options .......................................................... 400
140: I-jet/JTAGjet JTAG/SWD options .................................................... 403
141: I-jet/JTAGjet SWO options .......................................................... 404
142: J-Link/J-Trace Setup options ........................................................ 406
143: J-Link/J-Trace Connection options .................................................. 410
144: Macraigor options ................................................................. 412
145: RDI options ................................................................. 414
146: ST-LINK Setup options .............................................................. 416
147: TI Stellaris Setup options ............................................................. 417
148: TI XDS100 Setup options ............................................................. 418
149: C-SPY Third-Party Driver options .................................................. 419
150: Simulator menu ................................................................. 421
151: The GDB Server menu .............................................................. 423
152: The I-jet/JTAGjet menu ............................................................. 424
153: The J-Link menu ................................................................. 427
154: The Macraigor JTAG menu .......................................................... 430
155: The RDI menu ................................................................. 431
156: The ST-LINK menu .............................................................. 432
157: The TI Stellaris menu .............................................................. 433
158: The TI XDS100 menu .............................................................. 434
159: Flash Loader Overview dialog box .................................................. 439
160: Flash Loader Configuration dialog box .................................................. 440
Preface

Welcome to the C-SPY® Debugging Guide for ARM. The purpose of this guide is to help you fully use the features in the IAR C-SPY® Debugger for debugging your application based on the ARM core.

Who should read this guide

Read this guide if you plan to develop an application using IAR Embedded Workbench and want to get the most out of the features available in C-SPY.

REQUIRED KNOWLEDGE

To use the tools in IAR Embedded Workbench, you should have working knowledge of:

- The architecture and instruction set of the ARM core (refer to the chip manufacturer's documentation)
- The C or C++ programming language
- Application development for embedded systems
- The operating system of your host computer.

For more information about the other development tools incorporated in the IDE, refer to their respective documentation, see Other documentation, page 30.

What this guide contains

This is a brief outline and summary of the chapters in this guide:

- The IAR C-SPY Debugger introduces you to the C-SPY debugger and to the concepts that are related to debugging in general and to C-SPY in particular. The chapter also introduces the various C-SPY drivers. The chapter briefly shows the difference in functionality that the various C-SPY drivers provide.
- Getting started using C-SPY helps you get started using C-SPY, which includes setting up, starting, and adapting C-SPY for target hardware.
- Executing your application describes the conceptual differences between source and disassembly mode debugging, the facilities for executing your application, and finally, how you can handle terminal input and output.
- Variables and expressions describes the syntax of the expressions and variables used in C-SPY, as well as the limitations on variable information. The chapter also demonstrates the various methods for monitoring variables and expressions.
Other documentation

- **Breakpoints** describes the breakpoint system and the various ways to set breakpoints.
- **Memory and registers** shows how you can examine memory and registers.
- **Collecting and using trace data** describes how you can inspect the program flow up to a specific state using trace data.
- **Using the profiler** describes how the profiler can help you find the functions in your application source code where the most time is spent during execution.
- **Power debugging** describes techniques for power debugging and how you can use C-SPY to find source code constructions that result in unexpected power consumption.
- **Code coverage** describes how the code coverage functionality can help you verify whether all parts of your code have been executed, thus identifying parts which have not been executed.
- **Interrupts** contains detailed information about the C-SPY interrupt simulation system and how to configure the simulated interrupts to make them reflect the interrupts of your target hardware.
- **Using C-SPY macros** describes the C-SPY macro system, its features, the purposes of these features, and how to use them.
- **The C-SPY Command Line Utility—cspybat** describes how to use C-SPY in batch mode.
- **Debugger options** describes the options you must set before you start the C-SPY debugger.
- **Additional information on C-SPY drivers** describes menus and features provided by the C-SPY drivers not described in any dedicated topics.
- **Flash loaders** describes the flash loader, what it is and how to use it.

Other documentation

User documentation is available as hypertext PDFs and as a context-sensitive online help system in HTML format. You can access the documentation from the Information Center or from the Help menu in the IAR Embedded Workbench IDE. The online help system is also available via the F1 key.
USER AND REFERENCE GUIDES

The complete set of IAR Systems development tools is described in a series of guides. Information about:

- System requirements and information about how to install and register the IAR Systems products, is available in the booklet Quick Reference (available in the product box) and the Installation and Licensing Guide.
- Getting started using IAR Embedded Workbench and the tools it provides, is available in the guide Getting Started with IAR Embedded Workbench®.
- Using the IDE for project management and building, is available in the IDE Project Management and Building Guide for ARM.
- Using the IAR C-SPY® Debugger, is available in the C-SPY® Debugging Guide for ARM.
- Programming for the IAR C/C++ Compiler for ARM and linking using the IAR ILINK Linker, is available in the IAR C/C++ Development Guide for ARM.
- Using the IAR Assembler for ARM, is available in the IAR Assembler Reference Guide for ARM.
- Using the IAR DLIB Library, is available in the DLIB Library Reference information, available in the online help system.
- Porting application code and projects created with a previous version of the IAR Embedded Workbench for ARM, is available in the IAR Embedded Workbench® Migration Guide.
- Using IAR J-Link and IAR J-Trace, refer to the IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores.
- Using I-jet, refer to the I-jet® User Guide for ARM.

Note: Additional documentation might be available depending on your product installation.

THE ONLINE HELP SYSTEM

The context-sensitive online help contains:

- Information about project management and building in the IDE
- Information about debugging using the IAR C-SPY® Debugger
- Information about using the editor
- Reference information about the menus, windows, and dialog boxes in the IDE
- Compiler reference information
Keyword reference information for the DLIB library functions. To obtain reference information for a function, select the function name in the editor window and press F1.

WEB SITES
Recommended web sites:

- The Advanced RISC Machines Ltd web site, www.arm.com, that contains information and news about the ARM cores.
- Finally, the Embedded C++ Technical Committee web site, www.caravan.net/ec2plus, that contains information about the Embedded C++ standard.

Document conventions

When, in the IAR Systems documentation, we refer to the programming language C, the text also applies to C++, unless otherwise stated.

When referring to a directory in your product installation, for example arm\doc, the full path to the location is assumed, for example c:\Program Files\IAR Systems\Embedded Workbench 6.n\arm\doc.

TYPOGRAPHIC CONVENTIONS
The IAR Systems documentation set uses the following typographic conventions:

<table>
<thead>
<tr>
<th>Style</th>
<th>Used for</th>
</tr>
</thead>
<tbody>
<tr>
<td>computer</td>
<td>• Source code examples and file paths.</td>
</tr>
<tr>
<td></td>
<td>• Text on the command line.</td>
</tr>
<tr>
<td></td>
<td>• Binary, hexadecimal, and octal numbers.</td>
</tr>
<tr>
<td>parameter</td>
<td>A placeholder for an actual value used as a parameter, for example filename.h where filename represents the name of the file.</td>
</tr>
<tr>
<td>[option]</td>
<td>An optional part of a linker directive, inline assembler statement, or a stack usage control directive, [ and ] are not part of the actual directive, but any [...], {. or } are part of the directive syntax.</td>
</tr>
</tbody>
</table>

Table 1: Typographic conventions used in this guide
NAMING CONVENTIONS

The following naming conventions are used for the products and tools from IAR Systems®, when referred to in the documentation:

<table>
<thead>
<tr>
<th>Brand name</th>
<th>Generic term</th>
</tr>
</thead>
<tbody>
<tr>
<td>IAR Embedded Workbench® for ARM</td>
<td>IAR Embedded Workbench®</td>
</tr>
<tr>
<td>IAR Embedded Workbench® IDE for ARM</td>
<td>the IDE</td>
</tr>
<tr>
<td>IAR C-SPY® Debugger for ARM</td>
<td>C-SPY, the debugger</td>
</tr>
<tr>
<td>IAR C-SPY® Simulator</td>
<td>the simulator</td>
</tr>
<tr>
<td>IAR C/C++ Compiler™ for ARM</td>
<td>the compiler</td>
</tr>
<tr>
<td>IAR Assembler™ for ARM</td>
<td>the assembler</td>
</tr>
<tr>
<td>IAR ILINK Linker™</td>
<td>ILINK, the linker</td>
</tr>
<tr>
<td>IAR DLIB Library™</td>
<td>the DLIB library</td>
</tr>
</tbody>
</table>

Table 2: Naming conventions used in this guide
Document conventions
The IAR C-SPY Debugger

This chapter introduces you to the IAR C-SPY® Debugger and to the concepts that are related to debugging in general and to C-SPY in particular. The chapter also introduces the various C-SPY drivers. More specifically, this means:

- Introduction to C-SPY
- Debugger concepts
- C-SPY drivers overview
- The IAR C-SPY Simulator
- The C-SPY Hardware debugger drivers

Introduction to C-SPY

This section covers these topics:

- An integrated environment
- General C-SPY debugger features
- RTOS awareness.

AN INTEGRATED ENVIRONMENT

C-SPY is a high-level-language debugger for embedded applications. It is designed for use with the IAR Systems compilers and assemblers, and is completely integrated in the IDE, providing development and debugging within the same application. This will give you possibilities such as:

- Editing while debugging. During a debug session, you can make corrections directly in the same source code window that is used for controlling the debugging. Changes will be included in the next project rebuild.
- Setting breakpoints at any point during the development cycle. You can inspect and modify breakpoint definitions also when the debugger is not running, and breakpoint definitions flow with the text as you edit. Your debug settings, such as watch properties, window layouts, and register groups will be preserved between your debug sessions.
All windows that are open in the Embedded Workbench workspace will stay open when you start the C-SPY Debugger. In addition, a set of C-SPY-specific windows are opened.

**GENERAL C-SPY DEBUGGER FEATURES**

Because IAR Systems provides an entire toolchain, the output from the compiler and linker can include extensive debug information for the debugger, resulting in good debugging possibilities for you.

C-SPY offers these general features:

- **Source and disassembly level debugging**
  C-SPY allows you to switch between source and disassembly debugging as required, for both C or C++ and assembler source code.

- **Single-stepping on a function call level**
  Compared to traditional debuggers, where the finest granularity for source level stepping is line by line, C-SPY provides a finer level of control by identifying every statement and function call as a step point. This means that each function call—inside expressions, and function calls that are part of parameter lists to other functions—can be single-stepped. The latter is especially useful when debugging C++ code, where numerous extra function calls are made, for example to object constructors.

- **Code and data breakpoints**
  The C-SPY breakpoint system lets you set breakpoints of various kinds in the application being debugged, allowing you to stop at locations of particular interest. For example, you set breakpoints to investigate whether your program logic is correct or to investigate how and when the data changes.

- **Monitoring variables and expressions**
  For variables and expressions there is a wide choice of facilities. You can easily monitor values of a specified set of variables and expressions, continuously or on demand. You can also choose to monitor only local variables, static variables, etc.

- **Container awareness**
  When you run your application in C-SPY, you can view the elements of library data types such as STL lists and vectors. This gives you a very good overview and debugging opportunities when you work with C++ STL containers.

- **Call stack information**
  The compiler generates extensive call stack information. This allows the debugger to show, without any runtime penalty, the complete stack of function calls wherever the program counter is. You can select any function in the call stack, and for each function you get valid information for local variables and available registers.
● Powerful macro system
C-SPY includes a powerful internal macro system, to allow you to define complex sets of actions to be performed. C-SPY macros can be used on their own or in conjunction with complex breakpoints and—if you are using the simulator—the interrupt simulation system to perform a wide variety of tasks.

Additional general C-SPY debugger features
This list shows some additional features:
● Threaded execution keeps the IDE responsive while running the target application
● Automatic stepping
● The source browser provides easy navigation to functions, types, and variables
● Extensive type recognition of variables
● Configurable registers (CPU and peripherals) and memory windows
● Graphical stack view with overflow detection
● Support for code coverage and function level profiling
● The target application can access files on the host PC using file I/O
● Optional terminal I/O emulation.

RTOS AWARENESS
C-SPY supports real-time OS aware debugging.
These operating systems are currently supported:
● AVIX-RT
● CMX-RTX
● CMX-Tiny+
● eForce mC3/Compact
● eSysTech X realtime kernel
● Express Logic ThreadX
● FreeRTOS, OpenRTOS, and SafeRTOS
● Freescale MQX
● Micrium uC/OS-II
● Micro Digital SMX
● MISPO NORTi
● OSEK (ORTI)
● RTXC Quadros
RTOS plugin modules can be provided by IAR Systems, and by third-party suppliers. Contact your software distributor or IAR Systems representative, alternatively visit the IAR Systems web site, for information about supported RTOS modules.

A C-SPY RTOS awareness plugin module gives you a high level of control and visibility over an application built on top of an RTOS. It displays RTOS-specific items like task lists, queues, semaphores, mailboxes, and various RTOS system variables. Task-specific breakpoints and task-specific stepping make it easier to debug tasks.

A loaded plugin will add its own menu, set of windows, and buttons when a debug session is started (provided that the RTOS is linked with the application). For information about other RTOS awareness plugin modules, refer to the manufacturer of the plugin module. For links to the RTOS documentation, see the release notes that are available from the Help menu.

Debugger concepts

This section introduces some of the concepts and terms that are related to debugging in general and to C-SPY in particular. This section does not contain specific information related to C-SPY features. Instead, you will find such information in the other chapters of this documentation. The IAR Systems user documentation uses the terms described in this section when referring to these concepts.

C-SPY AND TARGET SYSTEMS

You can use C-SPY to debug either a software target system or a hardware target system.
This figure gives an overview of C-SPY and possible target systems:

THE DEBUGGER
The debugger, for instance C-SPY, is the program that you use for debugging your applications on a target system.

THE TARGET SYSTEM
The target system is the system on which you execute your application when you are debugging it. The target system can consist of hardware, either an evaluation board or your own hardware design. It can also be completely or partially simulated by software. Each type of target system needs a dedicated C-SPY driver.

THE APPLICATION
A user application is the software you have developed and which you want to debug using C-SPY.

C-SPY DEBUGGER SYSTEMS
C-SPY consists of both a general part which provides a basic set of debugger features, and a target-specific back end. The back end consists of two components: a processor module—one for every microcontroller, which defines the properties of the
microcontroller, and a C-SPY driver. The C-SPY driver is the part that provides communication with and control of the target system. The driver also provides the user interface—menus, windows, and dialog boxes—to the functions provided by the target system, for instance, special breakpoints. Typically, there are three main types of C-SPY drivers:

- Simulator driver
- ROM-monitor driver
- Emulator driver.

C-SPY is available with a simulator driver, and depending on your product package, optional drivers for hardware debugger systems. For an overview of the available C-SPY drivers and the functionality provided by each driver, see C-SPY drivers overview, page 41.

THE ROM-MONITOR PROGRAM

The ROM-monitor program is a piece of firmware that is loaded to non-volatile memory on your target hardware; it runs in parallel with your application. The ROM-monitor communicates with the debugger and provides services needed for debugging the application, for instance stepping and breakpoints.

THIRD-PARTY DEBUGGERS

You can use a third-party debugger together with the IAR Systems toolchain as long as the third-party debugger can read ELF/DWARF, Intel-extended, or Motorola. For information about which format to use with a third-party debugger, see the user documentation supplied with that tool.

C-SPY PLUGIN MODULES

C-SPY is designed as a modular architecture with an open SDK that can be used for implementing additional functionality to the debugger in the form of plugin modules. These modules can be seamlessly integrated in the IDE.

Plugin modules are provided by IAR Systems, or can be supplied by third-party vendors. Examples of such modules are:

- Code Coverage, which is integrated in the IDE.
- The various C-SPY drivers for debugging using certain debug systems.
- RTOS plugin modules for support for real-time OS aware debugging.
- Peripheral simulation modules make C-SPY simulate peripheral units. Such plugin modules are not provided by IAR Systems, but can be developed and distributed by third-party suppliers.
The IAR C-SPY Debugger

- C-SPYLink that bridges IAR visualSTATE and IAR Embedded Workbench to make true high-level state machine debugging possible directly in C-SPY, in addition to the normal C level symbolic debugging. For more information, see the documentation provided with IAR visualSTATE.

For more information about the C-SPY SDK, contact IAR Systems.

C-SPY drivers overview

At the time of writing this guide, the IAR C-SPY Debugger for the ARM cores is available with drivers for these target systems and evaluation boards:

- Simulator
- I-jet in-circuit debugging probe / JTAGjet debug probes
- J-Link / J-Trace JTAG/SWD probes
- JTAGjet probes
- RDI (Remote Debug Interface)
- Macraigor JTAG probes
- GDB Server
- ST-LINK JTAG/SWD probe (for ST Cortex-M devices only)
- TI Stellaris JTAG/SWD interface using FTDI or ICDI (for Stellaris Cortex devices only)
- TI XDS100 JTAG interface
- P&E Microcomputer Systems. For information about this driver, see the document Configuring IAR Embedded Workbench for ARM to use a P&E Microcomputer Systems Interface, available in the arm\doc directory.
- Angel debug monitor
- IAR ROM-monitor for Analog Devices ADuC7xxx boards, and IAR Kickstart Card for Philips LPC210x.

Note: In addition to the drivers supplied with IAR Embedded Workbench, you can also load debugger drivers supplied by a third-party vendor; see Third-Party Driver options, page 419.
Differences between the C-SPY Drivers

This table summarizes the key differences between the Simulator, I-jet/JTAGjet, J-Link/J-Trace, and ST-LINK:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Simulator</th>
<th>I-jet/JTAGjet</th>
<th>J-Link/J-Trace</th>
<th>ST-LINK</th>
<th>Comment, for I-jet/JTAGjet, J-Link and ST-LINK</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code breakpoints</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>--</td>
</tr>
<tr>
<td>Data breakpoints</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>--</td>
</tr>
<tr>
<td>Interrupt logging</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Cortex with SWD/SWO</td>
</tr>
<tr>
<td>Data logging</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Cortex with SWD/SWO</td>
</tr>
<tr>
<td>Call stack trace</td>
<td>--</td>
<td>--</td>
<td>x</td>
<td>--</td>
<td>Requires ETM/ETB trace.</td>
</tr>
<tr>
<td>Power logging</td>
<td>--</td>
<td>x</td>
<td>x</td>
<td>--</td>
<td>Requires a Cortex-M device with SWO and a J-Link or J-Link Ultra debug probe.</td>
</tr>
<tr>
<td>Event logging</td>
<td>--</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Cortex with SWD/SWO, using ITM channel</td>
</tr>
<tr>
<td>Live watch</td>
<td>--</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Supported by Cortex devices. For ARM7/9 devices, Live watch is supported if you add a DCC handler to your application. See Live watch and use of DCC, page 429.</td>
</tr>
<tr>
<td>Cycle counter</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>For Cortex-M devices only.</td>
</tr>
<tr>
<td>Code coverage</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Supported by J-Trace and J-link with ETB. For Cortex-M devices, I-jet, J-Link and ST-LINK with SWO support partial code coverage. For more information about code coverage, see Code coverage, page 265.</td>
</tr>
<tr>
<td>Data coverage</td>
<td>x</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Function/instruction profiler</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Requires either SWD/SWO interface or ETM/ETB trace.</td>
</tr>
<tr>
<td>Trace</td>
<td>x</td>
<td>--</td>
<td>x</td>
<td>--</td>
<td>Requires ETM/ETB trace.</td>
</tr>
</tbody>
</table>

Table 3: Driver differences, I-jet/JTAGjet, J-Link/J-Trace and ST-LINK
This table summarizes the key differences between the Simulator and other supported hardware debugger drives:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Simulator</th>
<th>RDI</th>
<th>Macraigor</th>
<th>GDB Server</th>
<th>TI Stellaris</th>
<th>TI XDS100</th>
<th>Angel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code breakpoints</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>Data breakpoints</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Interrupt logging</td>
<td>x</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Cycle counter</td>
<td>x</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Code coverage</td>
<td>x</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Data coverage</td>
<td>x</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Function/instruction profiler</td>
<td>x</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>Trace</td>
<td>x</td>
<td>x</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
</tbody>
</table>

Table 4: Driver differences, other drivers

The IAR C-SPY Simulator

The C-SPY Simulator simulates the functions of the target processor entirely in software, which means that you can debug the program logic long before any hardware is available. Because no hardware is required, it is also the most cost-effective solution for many applications.

SIMULATOR FEATURES

In addition to the general features in C-SPY, the simulator also provides:

- Instruction-level simulation
- Memory configuration and validation
- Interrupt simulation
- Peripheral simulation (using the C-SPY macro system in conjunction with immediate breakpoints).

The C-SPY hardware debugger drivers

C-SPY can connect to a hardware debugger using a C-SPY hardware debugger driver as an interface.

When a debugging session is started, your application is automatically downloaded and programmed into target memory. You can disable this feature, if necessary.
COMMUNICATION OVERVIEW

There are two main communication setups, depending on the type of target system. Many of the ARM cores have built-in, on-chip debug support. Because the hardware debugger logic is built into the core, no ordinary ROM-monitor program or extra specific hardware is needed to make the debugging work, other than the debug probe. For some devices that do not have such built-in, on-chip debug support, there are instead a ROM-monitor debugger solution that can be used.

Overview of a target system with a debug probe or emulator

Most target systems have an emulator, a debug probe or a debug adapter connected between the host computer and the evaluation board:

When USB connection is used, a specific USB driver must be installed before you can use the probe over the USB port. You can find the driver on the IAR Embedded Workbench for ARM installation media.
Overview of a target system using a ROM-monitor

IAR Embedded Workbench comes with two ready-made ROM-monitors:

- Using the IAR Angel debug monitor driver, you can communicate with any device compliant with the Angel debug monitor protocol. In most cases these are evaluation boards.
- Using the IAR ROM-monitor driver, C-SPY can connect to the Analog Devices ADuC7xxx boards and the IAR Kickstart Card for Philips LPC210x. Most ROM-monitors require that the code you want to debug is located in RAM, because the only way you can set breakpoints and step in your application code is to download it to RAM. For some ROM-monitors, for example for Analog Devices ADuC7xxx, the code that you want to debug can be located in flash memory. To maintain debug functionality, the ROM-monitor might simulate some instructions, for example when single stepping.

The boards contain firmware (the ROM-monitor itself) that runs in parallel with your application software. The firmware receives commands from the IAR C-SPY debugger over a serial port, and controls the execution of your application.
Using the C-SPY ROM-monitor driver, C-SPY can connect to a target system equipped with a ROM-monitor located in flash memory. This is an inexpensive solution to debug a target because only a serial cable is needed. All the parts of your code that you want to debug must be located in RAM. The only way you can set breakpoints and step in your application code is to download it into RAM.

For further information, see:
- The `angel_quickstart.html` file, available in the `arm\doc\infocenter` directory, or refer to the manufacturer’s documentation.
- The ` iar_rom_quickstart.html ` file, available in the `arm\doc\infocenter` directory, or refer to the manufacturer’s documentation.

**HARDWARE INSTALLATION**

For information about the hardware installation, see the documentation supplied with the target system from the manufacturer. The following power-up sequence is
The IAR C-SPY Debugger

recommended to ensure proper communication between the target board, the emulator or debug probe, and C-SPY:

1. Connect the probe to the target board.
2. Connect the USB cable to the debug probe.
3. Power up the debug probe, if it is not powered via USB.
4. Power up the target board, if it is not powered by the debug probe.
5. Start the C-SPY debugging session.

USB DRIVER INSTALLATION

A USB driver is also needed. In some cases this driver is automatically installed, but for some probes you need to manually install it.

Installing the I-jet and JTAGjet USB driver

Before you can use the I-jet or the JTAGjet interface over the USB port, the proper USB driver must be installed. Use the USB cable to connect the computer to the I-jet or JTAGjet probe.

Windows 7

1. Start the Windows Device Manager.
2. Select Other devices, right-click on JTAGjet and select Update Driver Software.
3. Click Browse my computer for driver software and browse to the arm/drivers/jet/USB.
4. Click Next and then Install.

Before Windows 7

The first time that the I-jet or JTAGjet interface and the computer are connected, Windows opens a dialog box and asks you to locate the USB driver. The drivers can be found in the product installation in the arm\drivers\jet\USB.

Once the initial setup is completed, you do not need to install the driver again.

Installing the J-Link USB driver

Before you can use the J-Link JTAG probe over the USB port, the Segger J-Link USB driver must be installed.

1. Install IAR Embedded Workbench for ARM.
2 Use the USB cable to connect the computer and J-Link. Do not connect J-Link to the target board yet. The green LED on the front panel of J-Link will blink for a few seconds while Windows searches for a USB driver.

Run the InstDrivers.exe application, which is located in the product installation in the arm\drivers\JLink directory.

Once the initial setup is completed, you will not have to install the driver again.

Note that J-Link will continuously blink until the USB driver has established contact with the J-Link probe. When contact has been established, J-Link will start with a steady light to indicate that it is connected.

**Installing the ST-LINK USB driver for ST-LINK ver. 2**

Before you can use the ST-LINK version 2 JTAG probe over the USB port, the ST-LINK USB driver must be installed.

1 Install IAR Embedded Workbench for ARM.

2 Use the USB cable to connect the computer and ST-LINK. Do not connect ST-LINK to the target board yet.

Because this is the first time ST-LINK and the computer are connected, Windows will open a dialog box and ask you to locate the USB driver. The USB driver can be found in the product installation in the arm\drivers\ST-Link directory: ST-Link_V2_USBdriver.exe.

Once the initial setup is completed, you will not have to install the driver again.

**Installing the TI Stellaris USB driver**

Before you can use the TI Stellaris JTAG interface using FTDI or ICDI over the USB port, the Stellaris USB driver must be installed.

1 Install IAR Embedded Workbench for ARM.

2 Use the USB cable to connect the computer to the TI board.

Because this is the first time the Stellaris JTAG interface and the computer are connected, Windows will open a dialog box and ask you to locate the USB driver. There are different USB drivers for FTDI and ICDI. The drivers can be found in the product installation in the arm\drivers\StellarisFTDI and the arm\drivers\StellarisICDI directories, respectively.

Once the initial setup is completed, you will not have to install the driver again.
Installing the TI XDS100 USB driver

Before you can use the TI XDS100 JTAG interface over the USB port, the TI XDS100 package must be installed.

1. Install IAR Embedded Workbench for ARM.

2. Install the TI XDS100 package which can be found in the `arm\drivers\ti-xds` directory. It is recommended to choose the suggested installation directory. See also Setup options for TI XDS100, page 418.

3. Use the USB cable to connect the computer to the TI board.

Configuring the OpenOCD Server

For further information, see the `gdbserv_quickstart.html` file, available in the `arm\doc\infocenter` directory, or refer to the manufacturer’s documentation.
The C-SPY hardware debugger drivers
Getting started using C-SPY

This chapter helps you get started using C-SPY®. More specifically, this means:

- Setting up C-SPY
- Starting C-SPY
- Adapting for target hardware
- Running example projects
- Reference information on starting C-SPY.

Setting up C-SPY

This section describes how to set up C-SPY.

More specifically, you will get information about:

- Setting up for debugging
- Executing from reset
- Using a setup macro file
- Selecting a device description file
- Loading plugin modules.

SETTING UP FOR DEBUGGING

Install a USB driver if your C-SPY® driver requires it. For more information, see:

- Installing the I-jet and JTAGjet USB driver, page 47
- Installing the J-Link USB driver, page 47
- Installing the ST-LINK USB driver for ST-LINK ver. 2, page 48
- Installing the TI Stellaris USB driver, page 48
- Installing the TI XDS100 USB driver, page 49
- Configuring the OpenOCD Server, page 49
Before you start C-SPY, choose Project>Options>Debugger>Setup and select the C-SPY driver that matches your debugger system: simulator or a hardware debugger system.

**Note:** You can only choose a driver you have installed on your computer.

In the **Category** list, select the appropriate C-SPY driver and make your settings. For information about these options, see *Debugger options*, page 389.

Click **OK**.

Choose **Tools>Options>Debugger** to configure:

- The debugger behavior
- The debugger’s tracking of stack usage.

For more information about these options, see the *IDE Project Management and Building Guide for ARM*.

The following documents containing information about how to set up various debugging systems are available in the arm\doc\infocenter subdirectory:

<table>
<thead>
<tr>
<th>File</th>
<th>Debugger system</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdi_quickstart.html</td>
<td>Quickstart reference for RDI-controlled JTAG debug interfaces</td>
</tr>
<tr>
<td>gdbserver_quickstart.html</td>
<td>Quickstart reference for a GDB Server using OpenOCD together with STR9-comStick</td>
</tr>
<tr>
<td>angel_quickstart.html</td>
<td>Quickstart reference for Angel ROM-monitors and JTAG interfaces</td>
</tr>
<tr>
<td>iar_rom_quickstart.html</td>
<td>Quickstart reference for IAR ROM-monitor</td>
</tr>
</tbody>
</table>

See also *Adapting for target hardware*, page 56.

**EXECUTING FROM RESET**

The **Run to** option—available on the Debugger>Setup page—specifies a location you want C-SPY to run to when you start a debug session as well as after each reset. C-SPY will place a temporary breakpoint at this location and all code up to this point is executed before stopping at the location.

The default location to run to is the **main** function. Type the name of the location if you want C-SPY to run to a different location. You can specify assembler labels or whatever can be evaluated to such, for instance function names.
If you leave the check box empty, the program counter will then contain the regular hardware reset address at each reset. The reset address is set by C-SPY.

If no breakpoints are available when C-SPY starts, a warning message notifies you that single stepping will be required and that this is time-consuming. You can then continue execution in single-step mode or stop at the first instruction. If you choose to stop at the first instruction, the debugger starts executing with the PC (program counter) at the default reset location instead of the location you typed in the Run to box.

**Note:** This message will never be displayed in the C-SPY Simulator, where breakpoints are not limited.

### USING A SETUP MACRO FILE

A setup macro file is a macro file that you choose to load automatically when C-SPY starts. You can define the setup macro file to perform actions according to your needs, using setup macro functions and system macros. Thus, if you load a setup macro file you can initialize C-SPY to perform actions automatically.

For more information about setup macro files and functions, see *Introduction to C-SPY macros*, page 293. For an example of how to use a setup macro file, see the chapter *Initializing target hardware before C-SPY starts*, page 58.

To register a setup macro file:

1. Before you start C-SPY, choose **Project>Options>Debugger>Setup**.
2. Select **Use macro file** and type the path and name of your setup macro file, for example *Setup.mac*. If you do not type a filename extension, the extension *mac* is assumed.

### SELECTING A DEVICE DESCRIPTION FILE

C-SPY uses device description files to handle device-specific information.

A default device description file is automatically used based on your project settings. If you want to override the default file, you must select your device description file. Device description files from IAR Systems are provided in the *arm\config* directory and they have the filename extension *ddf*.

For more information about device description files, see *Adapting for target hardware*, page 56.

To override the default device description file:

1. Before you start C-SPY, choose **Project>Options>Debugger>Setup**.
2. Enable the use of a device description file and select a file using the **Device description file** browse button.
LOADING PLUGIN MODULES

On the Plugins page you can specify C-SPY plugin modules to load and make available during debug sessions. Plugin modules can be provided by IAR Systems, and by third-party suppliers. Contact your software distributor or IAR Systems representative, or visit the IAR Systems web site, for information about available modules.

For more information, see Plugins, page 395.

---

Starting C-SPY

When you have set up the debugger, you are ready to start a debug session; this section describes various ways to start C-SPY.

More specifically, you will get information about:

- Starting a debug session
- Loading executable files built outside of the IDE
- Starting a debug session with source files missing
- Loading multiple images.

STARTING A DEBUG SESSION

You can choose to start a debug session with or without loading the current project.

To start C-SPY and load the current project, click the Download and Debug button. Alternatively, choose Project>Download and Debug.

To start C-SPY without reloading the current project, click the Debug without Downloading button. Alternatively, choose Project>Debug without Downloading.

LOADING EXECUTABLE FILES BUILT OUTSIDE OF THE IDE

You can also load C-SPY with an application that was built outside the IDE, for example applications built on the command line. To load an externally built executable file and to set build options you must first create a project for it in your workspace.

To create a project for an externally built file:

1. Choose Project>Create New Project, and specify a project name.
2. To add the executable file to the project, choose Project>Add Files and make sure to choose All Files in the Files of type drop-down list. Locate the executable file.
3. To start the executable file, click the Download and Debug button. The project can be reused whenever you rebuild your executable file.
The only project options that are meaningful to set for this kind of project are options in the **General Options** and **Debugger** categories. Make sure to set up the general project options in the same way as when the executable file was built.

**STARTING A DEBUG SESSION WITH SOURCE FILES MISSING**

Normally, when you use the IAR Embedded Workbench IDE to edit source files, build your project, and start the debug session, all required files are available and the process works as expected.

However, if C-SPY cannot automatically find the source files, for example if the application was built on another computer, the **Get Alternative File** dialog box is displayed:

Typically, you can use the dialog box like this:

- The source files are not available: Click **If possible, don’t show this dialog again** and then click **Skip**. C-SPY will assume that there simply is no source file available. The dialog box will not appear again, and the debug session will not try to display the source code.

- Alternative source files are available at another location: Specify an alternative source code file, click **If possible, don’t show this dialog again**, and then click **Use this file**. C-SPY will assume that the alternative file should be used. The dialog box will not appear again, unless a file is needed for which there is no alternative file specified and which cannot be located automatically.

If you restart the IAR Embedded Workbench IDE, the **Get Alternative File** dialog box will be displayed again once even if you have clicked **If possible, don’t show this dialog again**. This gives you an opportunity to modify your previous settings.

For more information, see **Get Alternative File dialog box**, page 71.
LOADING MULTIPLE IMAGES

Normally, a debuggable application consists of exactly one file that you debug. However, you can also load additional debug files (images). This means that the complete program consists of several images.

Typically, this is useful if you want to debug your application in combination with a prebuilt ROM image that contains an additional library for some platform-provided features. The ROM image and the application are built using separate projects in the IAR Embedded Workbench IDE and generate separate output files.

If more than one image has been loaded, you will have access to the combined debug information for all the loaded images. In the Images window you can choose whether you want to have access to debug information for one image or for all images.

To load additional images at C-SPY startup:

1. Choose Project>Options>Debugger>Images and specify up to three additional images to be loaded. For more information, see Images, page 394.

2. Start the debug session.

To load additional images at a specific moment:

Use the __loadImage system macro and execute it using either one of the methods described in Using C-SPY macros, page 296.

To display a list of loaded images:

Choose Images from the View menu. The Images window is displayed, see Images window, page 69.

Adapting for target hardware

This section provides information about how to describe the target hardware to C-SPY, and how you can make C-SPY initialize the target hardware before your application is downloaded to memory.

More specifically, you will get information about:

- Memory configuration
- Modifying a device description file
- Initializing target hardware before C-SPY starts
- Remapping memory
MEMORY CONFIGURATION

Providing C-SPY with information about the memory layout of the target system is helpful both in terms of performance and functionality.

- Reading (and writing) memory (if your debug probe is connected for example through a USB port) can be fast, but is usually the limiting factor when C-SPY needs to update many debugger windows. Caching memory can speed up the performance, but then C-SPY needs information about the target memory.
- If C-SPY has been informed that the content of certain memory areas will not change during a debug session, C-SPY can keep a copy of that memory readable even when the target does not normally allow reading (such as when executing).
- C-SPY can prevent accesses to areas without any memory at all, which can be important for certain hardware.

Typically, when you set up your project, a device description file for your particular device is automatically or manually selected. If that file fully specifies the memory range information of your device, you do not have to configure C-SPY in this respect. However, if that file does not specify memory ranges for a specific device, but rather for a family of devices (perhaps with different amounts of on-chip RAM), you will automatically be asked to examine or modify the ranges to fit your specific device.

To fine-tune the information to suit your device, use the Memory Configuration dialog box, see Memory Configuration dialog box, page 179.

MODIFYING A DEVICE DESCRIPTION FILE

C-SPY uses device description files provided with the product to handle several of the target-specific adaptations, see Selecting a device description file, page 53. They contain device-specific information such as:

- Definitions of registers in peripheral units and groups of these.
- Interrupt definitions (for Cortex-M devices only); see Interrupts, page 271.

Normally, you do not need to modify the device description file. However, if the predefined definitions are not sufficient for some reason, you can edit the file. Note, however, that the format of these descriptions might be updated in future upgrade versions of the product.

Make a copy of the device description file that best suits your needs, and modify it according to the description in the file.

The syntax of the device description files is described in the IAR Embedded Workbench for ARM device description file format guide (EWARM_DDFFormat.pdf) located in the arm\doc directory.
For information about how to load a device description file, see Selecting a device description file, page 53.

INITIALIZING TARGET HARDWARE BEFORE C-SPY STARTS

You can use C-SPY macros to initialize target hardware before C-SPY starts. For example, if your hardware uses external memory that must be enabled before code can be downloaded to it, C-SPY needs a macro to perform this action before your application can be downloaded. For example:

1. Create a new text file and define your macro function. For example, a macro that enables external SDRAM might look like this:

   /* Your macro function. */
   enableExternalSDRAM()
   {
      __message "Enabling external SDRAM\n";
      __writeMemory32( /* Place your code here. */ );
      /* And more code here, if needed. */
   }
   /* Setup macro determines time of execution. */
   execUserPreload()
   {
      enableExternalSDRAM();
   }

   Because the built-in execUserPreload setup macro function is used, your macro function will be executed directly after the communication with the target system is established but before C-SPY downloads your application.

2. Save the file with the filename extension .mac.

3. Before you start C-SPY, choose Project>Options>Debugger and click the Setup tab.

4. Select the option Use Setup file and choose the macro file you just created.

   Your setup macro will now be loaded during the C-SPY startup sequence.

REMAPPING MEMORY

A common feature of many ARM-based processors is the ability to remap memory. After a reset, the memory controller typically maps address zero to non-volatile memory, such as flash. By configuring the memory controller, the system memory can be remapped to place RAM at zero and non-volatile memory higher up in the address map. By doing this, the exception table will reside in RAM and can be easily modified when you download code to the target hardware.
You must configure the memory controller before you download your application code. You can do this best by using a C-SPY macro function that is executed before the code download takes place—execUserPreload(). The macro function __writeMemory32() will perform the necessary initialization of the memory controller.

The following example illustrates a macro used for remapping memory on the Atmel AT91SAM7S256 chip, similar mechanisms exist in processors from other ARM vendors.

```c
execUserPreload()
{
    // REMAP command
    // Writing 1 to MC_RCR (MC Remap Control Register)
    // will toggle remap bit.
    __writeMemory32(0x00000001, 0xFFFFFF00, "Memory");
}
```

Note that the setup macro execUserReset() might have to be defined in the same way to reinitialize the memory mapping after a C-SPY reset. This can be needed if you have set up your hardware debugger system to do a hardware reset on C-SPY reset, for example by adding __hwReset() to the execUserReset() macro.

For instructions on how to install a macro file in C-SPY, see Registering and executing using setup macros and setup files, page 298. For information about the macro functions used, see Reference information on C-SPY system macros, page 307.

---

**An overview of the debugger startup**

To make it easier to understand and follow the startup flow, the following figures show the flow of actions performed by C-SPY, and by the target hardware, as well as the execution of any predefined C-SPY setup macros. There is one figure for debugging code located in flash and one for debugging code located in RAM.

For more information about C-SPY system macros, see the chapter C-SPY macros available in this guide.

More specifically, you will get information about:

- Debugging code in flash
- Debugging code in RAM
An overview of the debugger startup

**DEBUGGING CODE IN FLASH**

![Diagram](image)

**Figure 5**: Debugger startup when debugging code in flash

- **C-SPY Debugger**
  - Debugger start
  - JTAG speed is set to the specified frequency, or very low (typically 32 kHz) if auto speed is selected
  - If the option for auto-speed is selected, JTAG speed is set to the maximum reliable speed
  - Flash loader loaded to target RAM
  - If the option Verify download is selected, the flash memory is verified for correct content
  - The debugger is ready to work with the application

- **C-SPY Setup Macro**
  - execUserFlashInit(id)
  - execUserFlashStart()
  - execUserFlashExit()
  - execUserPreload()
  - execUserReset()
  - execUserSetup()

- **Target Hardware**
  - CPU reset
  - CPU halted
  - CPU halted
  - Target flash loader executes from RAM and loads application image from local and writes it to flash
  - CPU reset
  - CPU halted
  - CPU halted

---

**C-SPY® Debugging Guide**

for ARM
Running example projects

IAR Embedded Workbench comes with example applications. You can use these examples to get started using the development tools from IAR Systems or simply to verify that contact has been established with your target board. You can also use the examples as a starting point for your application project.

You can find the examples in the `armexamples` directory. The examples are ready to be used as is. They are supplied with ready-made workspace files, together with source code files and all other related files.
RUNNING AN EXAMPLE PROJECT

To run an example project:

1. Choose Help>Information Center and click EXAMPLE PROJECTS.
2. Browse to the example that matches the specific evaluation board or starter kit you are using.

![Figure 7: Example applications](image)

Click the Open Project button.

3. In the dialog box that appears, choose a destination folder for your project location. Click Select to confirm your choice.

4. The available example projects are displayed in the workspace window. Select one of the projects, and if it is not the active project (highlighted in bold), right-click it and choose Set As Active from the context menu.

5. To view the project settings, select the project and choose Options from the context menu. Verify the settings for General Options>Target>Processor variant and Debugger>Setup>Driver. As for other settings, the project is set up to suit the target system you selected.

For more information about the C-SPY options and how to configure C-SPY to interact with the target board, see Debugger options, page 389.
Click **OK** to close the project **Options** dialog box.

6 To compile and link the application, choose **Project>Make** or click the **Make** button.

7 To start C-SPY, choose **Project>Debug** or click the **Download and Debug** button. If C-SPY fails to establish contact with the target system, see **Resolving problems**, page 434.

8 Choose **Debug>Go** or click the **Go** button to start the application. Click the **Stop** button to stop execution.

---

**Reference information on starting C-SPY**

This section gives reference information about these windows and dialog boxes:

- **C-SPY Debugger main window**, page 63
- **Images window**, page 69
- **Get Alternative File dialog box**, page 71

See also:

- Tools options for the debugger in the **IDE Project Management and Building Guide for ARM**.

---

**C-SPY Debugger main window**

When you start a debug session, these debugger-specific items appear in the main IAR Embedded Workbench IDE window:

- A dedicated **Debug** menu with commands for executing and debugging your application
- Depending on the C-SPY driver you are using, a driver-specific menu, often referred to as the **Driver menu** in this documentation. Typically, this menu contains menu commands for opening driver-specific windows and dialog boxes.
- A special debug toolbar
- Several windows and dialog boxes specific to C-SPY.

The C-SPY main window might look different depending on which components of the product installation you are using.
Menu bar

These menus are available during a debug session:

Debug

Provides commands for executing and debugging the source application. Most of the commands are also available as icon buttons on the debug toolbar.

Simulator

Provides access to the dialog boxes for setting up interrupt simulation and memory access checking. This menu is only available when the C-SPY Simulator is used; see Simulator menu, page 421.

GDB Server

Provides commands specific to the C-SPY GDB Server driver. This menu is only available when the driver is used; see GDB Server menu, page 423.

I-jet/JTAGjet

Provides commands specific to the C-SPY I-jet/JTAGjet driver. This menu is only available when the driver is used; see I-jet/JTAGjet menu, page 424.

J-Link

Provides commands specific to the C-SPY J-Link driver. This menu is only available when the driver is used; see J-Link menu, page 427.

TI Stellaris

Provides commands specific to the C-SPY TI Stellaris driver. This menu is only available when the driver is used; see TI Stellaris menu, page 433.

TI XDS100

Provides commands specific to the TI XDS100 driver. This menu is only available when the driver is used; see TI XDS100 menu, page 434.

JTAG

Provides commands specific to the C-SPY Macraigor driver. This menu is only available when the driver is used; see Macraigor JTAG menu, page 430.

RDI

Provides commands specific to the C-SPY RDI driver. This menu is only available when the driver is used; see RDI menu, page 431.

ST-LINK

Provides commands specific to the C-SPY ST-LINK driver. This menu is only available when the driver is used; see ST-LINK menu, page 432.
Debug menu

The **Debug** menu is available during a debug session. The **Debug** menu provides commands for executing and debugging the source application. Most of the commands are also available as icon buttons on the debug toolbar.

These commands are available:

- **Go F5**
  
  Executes from the current statement or instruction until a breakpoint or program exit is reached.

- **Break**
  
  Stops the application execution.

- **Reset**
  
  Resets the target processor.

- **Stop Debugging (Ctrl+Shift+D)**
  
  Stops the debugging session and returns you to the project manager.

- **Step Over (F10)**
  
  Executes the next statement, function call, or instruction, without entering C or C++ functions or assembler subroutines.

- **Step Into (F11)**
  
  Executes the next statement or instruction, or function call, entering C or C++ functions or assembler subroutines.
Step Out (Shift+F11)
Executes from the current statement up to the statement after the call to the current function.

Next Statement
Executes directly to the next statement without stopping at individual function calls.

Run to Cursor
Executes from the current statement or instruction up to a selected statement or instruction.

Autostep
Displays a dialog box where you can customize and perform autostepping, see Autostep settings dialog box, page 90.

Set Next Statement
Moves the program counter directly to where the cursor is, without executing any source code. Note, however, that this creates an anomaly in the program flow and might have unexpected effects.

C++ Exceptions>
Break on Throw
Specifies that the execution shall break when the target application executes a throw statement.
To use this feature, your application must be built with the option Library low-level interface implementation selected and the language option C++ for Standard C++.

C++ Exceptions>
Break on Uncaught Exception
Specifies that the execution shall break when the target application throws an exception that is not caught by any matching catch statement.
To use this feature, your application must be built with the option Library low-level interface implementation selected and the language option C++ for Standard C++.

Memory>Save
Displays a dialog box where you can save the contents of a specified memory area to a file, see Memory Save dialog box, page 163.
Memory>Restore
Displays a dialog box where you can load the contents of a file in, for example Intel-extended or Motorola s-record format to a specified memory zone, see Memory Restore dialog box, page 164.

Refresh
Refreshes the contents of all debugger windows. Because window updates are automatic, this is needed only in unusual situations, such as when target memory is modified in ways C-SPY cannot detect. It is also useful if code that is displayed in the Disassembly window is changed.

Macros
Displays a dialog box where you can list, register, and edit your macro files and functions, see Using the Macro Configuration dialog box, page 297.

Logging>Set Log file
Displays a dialog box where you can choose to log the contents of the Debug Log window to a file. You can select the type and the location of the log file. You can choose what you want to log: errors, warnings, system information, user messages, or all of these. See Log File dialog box, page 89.

Logging>
Set Terminal I/O Log file
Displays a dialog box where you can choose to log simulated target access communication to a file. You can select the destination of the log file. See Terminal I/O Log File dialog box, page 87.

Disassembly menu

The Disassembly menu is available when C-SPY is running. This menu provides commands for executing and debugging the source application. Most of the commands are also available as icon buttons on the debug toolbar.

Note: After changing disassembly mode, use the Refresh command on the Debug menu to refresh the view of the Disassembly window contents.
These commands are available:

- **Disassemble in Thumb mode** Disassembles your application in Thumb mode.
- **Disassemble in ARM mode** Disassembles your application in ARM mode.
- **Disassemble in Current processor mode** Disassembles your application in the current processor mode.
- **Disassemble in Auto mode** Disassembles your application in automatic mode. This is the default option.

See also *Disassembly window*, page 80.

**C-SPY windows**

Depending on the C-SPY driver you are using, these windows specific to C-SPY are available during a debug session:

- C-SPY Debugger main window
- Disassembly window
- Memory window
- Symbolic Memory window
- Register window
- Watch window
- Locals window
- Auto window
- Live Watch window
- Quick Watch window
- Statics window
- Call Stack window
- Trace window
- Function Trace window
- Timeline window
- Terminal I/O window
- Code Coverage window
- Function Profiler window
- Images window
- Stack window
Symbols window.

Additional windows are available depending on which C-SPY driver you are using.

**Editing in C-SPY windows**

You can edit the contents of the Memory, Symbolic Memory, Register, Auto, Watch, Locals, Statics, Live Watch, and Quick Watch windows.

Use these keyboard keys to edit the contents of these windows:

- **Enter** Makes an item editable and saves the new value.
- **Esc** Cancels a new value.

In windows where you can edit the Expression field, you can specify the number of elements to be displayed in the field by adding a semicolon followed by an integer. For example, to display only the first three elements of an array named `myArray`, or three elements in sequence starting with the element pointed to by a pointer, write:

```
myArray;3
```

Optionally, add a comma and another integer that specifies which element to start with. For example, to display elements 10–14, write:

```
myArray;5,10
```

**Images window**

The Images window is available from the View menu.

<table>
<thead>
<tr>
<th>Name</th>
<th>Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;All images&gt;</td>
<td>[Combines debug information from all images]</td>
</tr>
<tr>
<td>project1</td>
<td>C:\Documents and Settings\My Documents\VAR Embedded Workbench\Debug\Eval\project1.out</td>
</tr>
<tr>
<td>extImage</td>
<td>C:\Documents and Settings\My Documents\VAR Embedded Workbench\Debug\Eval\extImage.out</td>
</tr>
</tbody>
</table>

The Images window lists all currently loaded images (debug files).

Normally, a source application consists of exactly one image that you debug. However, you can also load additional images. This means that the complete debuggable unit consists of several images.

**Requirements**

None; this window is always available.
Reference information on starting C-SPY

Display area

C-SPY can either use debug information from all of the loaded images simultaneously, or from one image at a time. Double-click on a row to show information only for that image. The current choice is highlighted.

This area lists the loaded images in these columns:

- **Name**: The name of the loaded image.
- **Path**: The path to the loaded image.

Context menu

This context menu is available:

- **Show all images**: Shows debug information for all loaded debug images.
- **Show only image**: Shows debug information for the selected debug image.

Related information

For related information, see:

- *Loading multiple images*, page 56
- *Images*, page 394
- *__loadImage*, page 325.
Get Alternative File dialog box

The Get Alternative File dialog box is displayed if C-SPY cannot automatically find the source files to be loaded, for example if the application was built on another computer.

![Get Alternative File dialog box](image)

**Could not find the following source file**

The missing source file.

**Suggested alternative**

Specify an alternative file.

**Use this file**

After you have specified an alternative file, **Use this file** establishes that file as the alias for the requested file. Note that after you have chosen this action, C-SPY will automatically locate other source files if these files reside in a directory structure similar to the first selected alternative file.

The next time you start a debug session, the selected alternative file will be preloaded automatically.

**Skip**

C-SPY will assume that the source file is not available for this debug session.

**If possible, don’t show this dialog again**

Instead of displaying the dialog box again for a missing source file, C-SPY will use the previously supplied response.

**Related information**

For related information, see Starting a debug session with source files missing, page 55.
Reference information on starting C-SPY
Executing your application

This chapter contains information about executing your application in C-SPY®. More specifically, this means:

- Introduction to application execution
- Reference information on application execution.

Introduction to application execution

This section covers these topics:

- Briefly about application execution
- Source and disassembly mode debugging
- Single stepping
- Stepping speed
- Running the application
- Highlighting
- Call stack information
- Terminal input and output
- Debug logging.

BRIEFLY ABOUT APPLICATION EXECUTION

C-SPY allows you to monitor and control the execution of your application. By single-stepping through it, and setting breakpoints, you can examine details about the application execution, for example the values of variables and registers. You can also use the call stack to step back and forth in the function call chain.

The terminal I/O and debug log features let you interact with your application.

You can find commands for execution on the Debug menu and on the toolbar.

SOURCE AND DISASSEMBLY MODE DEBUGGING

C-SPY allows you to switch between source mode and disassembly mode debugging as needed.

Source debugging provides the fastest and easiest way of developing your application, without having to worry about how the compiler or assembler has implemented the
Introduction to application execution

code. In the editor windows you can execute the application one statement at a time while monitoring the values of variables and data structures.

Disassembly mode debugging lets you focus on the critical sections of your application, and provides you with precise control of the application code. You can open a disassembly window which displays a mnemonic assembler listing of your application based on actual memory contents rather than source code, and lets you execute the application exactly one machine instruction at a time.

Regardless of which mode you are debugging in, you can display registers and memory, and change their contents.

**SINGLE STEPPING**

C-SPY allows more stepping precision than most other debuggers because it is not line-oriented but statement-oriented. The compiler generates detailed stepping information in the form of step points at each statement, and at each function call. That is, source code locations where you might consider whether to execute a step into or a step over command. Because the step points are located not only at each statement but also at each function call, the step functionality allows a finer granularity than just stepping on statements.

There are several factors that can slow down the stepping speed. If you find it too slow, see *Slow stepping speed*, page 434 for some tips.

**The step commands**

There are four step commands:

- Step Into
- Step Over
- Next Statement
- Step Out.

Using the Autostep settings dialog box, you can automate the single stepping. For more information, see *Autostep settings dialog box*, page 90.

If your application contains an exception that is caught outside the code which would normally be executed as part of a step, C-SPY terminates the step at the catch statement.
Consider this example and assume that the previous step has taken you to the $f(i)$ function call (highlighted):

```c
extern int g(int);
int f(int n)
{
    value = g(n-1) + g(n-2) + g(n-3);
    return value;
}
int main()
{
    ...
    f(i);
    value ++;
}
```

**Step Into**

While stepping, you typically consider whether to step into a function and continue stepping inside the function or subroutine. The *Step Into* command takes you to the first step point within the subroutine $g(n-1)$:

```c
extern int g(int);
int f(int n)
{
    value = g(n-1) + g(n-2) + g(n-3);
    return value;
}
```

The *Step Into* command executes to the next step point in the normal flow of control, regardless of whether it is in the same or another function.

**Step Over**

The *Step Over* command executes to the next step point in the same function, without stopping inside called functions. The command would take you to the $g(n-2)$ function call, which is not a statement on its own but part of the same statement as $g(n-1)$. Thus, you can skip uninteresting calls which are parts of statements and instead focus on critical parts:

```c
extern int g(int);
int f(int n)
{
    value = g(n-1) + g(n-2) + g(n-3);
    return value;
}
```
Next Statement

The Next Statement command executes directly to the next statement, in this case return value, allowing faster stepping:

```c
extern int g(int);
int f(int n)
{
    value = g(n-1) + g(n-2) + g(n-3);
    return value;
}
```

Step Out

When inside the function, you can—if you wish—use the Step Out command to step out of it before it reaches the exit. This will take you directly to the statement immediately after the function call:

```c
extern int g(int);
int f(int n)
{
    value = g(n-1) + g(n-2) + g(n-3);
    return value;
}
int main()
{
    ... 
    f(i);
    value ++;
}
```

The possibility of stepping into an individual function that is part of a more complex statement is particularly useful when you use C code containing many nested function calls. It is also very useful for C++, which tends to have many implicit function calls, such as constructors, destructors, assignment operators, and other user-defined operators.

This detailed stepping can in some circumstances be either invaluable or unnecessarily slow. For this reason, you can also step only on statements, which means faster stepping.

STEPPING SPEED

Stepping in C-SPY is normally performed using breakpoints. When performing a step command, a breakpoint is set on the next statement and the program executes until reaching this breakpoint. If you are debugging using a hardware debugger system, the number of hardware breakpoints—typically used for setting a stepping breakpoint, at least in code that is located in flash/ROM memory—is limited. If you for example, step into a C switch statement, breakpoints are set on each branch, and hence, this might
consume several hardware breakpoints. If the number of available hardware breakpoints is exceeded, C-SPY switches into single stepping at assembly level, which can be very slow.

For this reason, it can be helpful to keep track of how many hardware breakpoints are used and make sure to some of them are left for stepping. For more information, see and Breakpoint consumers, page 123.

In addition to limited hardware breakpoints, these issues might also affect stepping speed:

- If Trace or Function profiling is enabled. This might slow down stepping because collected Trace data is processed after each step. Note that it is not sufficient to close the corresponding windows to disable Trace data collection. Instead, you must disable the Enable/Disable button in both the Trace and the Function profiling windows.
- If the Register window is open and displays SFR registers. This might slow down stepping because all registers in the selected register group must be read from the hardware after each step. To solve this, you can choose to view only a limited selection of SFR register; you can choose between two alternatives. Either type #SFR_name (where #SFR_name reflects the name of the SFR you want to monitor) in the Watch window, or create your own filter for displaying a limited group of SFRs in the Register window. See Defining application-specific register groups, page 157.
- If any of the Memory or Symbolic memory windows is open. This might slow down stepping because the visible memory must be read after each step.
- If any of the expression related windows such as Watch, Live Watch, Locals, Statics is open. This might slow down stepping speed because all these windows reads memory after each step.
- If the Stack window is open and especially if the option Enable graphical stack display and stack usage tracking option is enabled. To disable this option, choose Tools>Options>Stack and disable it.
- If a too slow communication speed has been set up between C-SPY and the target board/emulator you should consider to increase the speed, if possible.

RUNNING THE APPLICATION

Go

The Go command continues execution from the current position until a breakpoint or program exit is reached.
Run to Cursor

The Run to Cursor command executes to the position in the source code where you have placed the cursor. The Run to Cursor command also works in the Disassembly window and in the Call Stack window.

HIGHLIGHTING

At each stop, C-SPY highlights the corresponding C or C++ source or instruction with a green color, in the editor and the Disassembly window respectively. In addition, a green arrow appears in the editor window when you step on C or C++ source level, and in the Disassembly window when you step on disassembly level. This is determined by which of the windows is the active window. If none of the windows are active, it is determined by which of the windows was last active.

```
void main() {
    int i = 452;
    printf("%d\n", i);
    for (i = 0; i < MAX_SIZE; i++)
    {
```

Figure 13: C-SPY highlighting source location

For simple statements without function calls, the whole statement is typically highlighted. When stopping at a statement with function calls, C-SPY highlights the first call because this illustrates more clearly what Step Into and Step Over would mean at that time.

Occasionally, you will notice that a statement in the source window is highlighted using a pale variant of the normal highlight color. This happens when the program counter is at an assembler instruction which is part of a source statement but not exactly at a step point. This is often the case when stepping in the Disassembly window. Only when the program counter is at the first instruction of the source statement, the ordinary highlight color is used.

CALL STACK INFORMATION

The compiler generates extensive backtrace information. This allows C-SPY to show, without any runtime penalty, the complete function call chain at any time.

Typically, this is useful for two purposes:

- Determining in what context the current function has been called
- Tracing the origin of incorrect values in variables and in parameters, thus locating the function in the call chain where the problem occurred.
The Call Stack window shows a list of function calls, with the current function at the top. When you inspect a function in the call chain, the contents of all affected windows are updated to display the state of that particular call frame. This includes the editor, Locals, Register, Watch and Disassembly windows. A function would normally not make use of all registers, so these registers might have undefined states and be displayed as dashes (---).

In the editor and Disassembly windows, a green highlight indicates the topmost, or current, call frame; a yellow highlight is used when inspecting other frames.

For your convenience, it is possible to select a function in the call stack and click the Run to Cursor command to execute to that function.

Assembler source code does not automatically contain any backtrace information. To see the call chain also for your assembler modules, you can add the appropriate CFI assembler directives to the assembler source code. For further information, see the IAR Assembler Reference Guide for ARM.

TERMINAL INPUT AND OUTPUT
Sometimes you might have to debug constructions in your application that use stdin and stdout without an actual hardware device for input and output. The Terminal I/O window lets you enter input to your application, and display output from it. You can also direct terminal I/O to a file, using the Terminal I/O Log Files dialog box.

This facility is useful in two different contexts:
- If your application uses stdin and stdout
- For producing debug trace printouts.

For more information, see Terminal I/O window, page 86 and Terminal I/O Log File dialog box, page 87.

DEBUG LOGGING
The Debug Log window displays debugger output, such as diagnostic messages, macro-generated output, event log messages, and information about trace.

It can sometimes be convenient to log the information to a file where you can easily inspect it. The two main advantages are:
- The file can be opened in another tool, for instance an editor, so you can navigate and search within the file for particularly interesting parts
- The file provides history about how you have controlled the execution, for instance, which breakpoints that have been triggered etc.
Reference information on application execution

This section gives reference information about these windows and dialog boxes:

- Disassembly window, page 80
- Call Stack window, page 84
- Terminal I/O window, page 86
- Terminal I/O Log File dialog box, page 87
- Debug Log window, page 88
- Log File dialog box, page 89
- Autostep settings dialog box, page 90

See also Terminal I/O options in IDE Project Management and Building Guide for ARM.

Disassembly window

The C-SPY Disassembly window is available from the View menu.

This window shows the application being debugged as disassembled application code.

To change the default color of the source code in the Disassembly window:

1. Choose Tools>Options>Debugger.

Figure 14: C-SPY Disassembly window
2. Set the default color using the **Source code coloring in disassembly window** option.

To view the corresponding assembler code for a function, you can select it in the editor window and drag it to the Disassembly window.

**Requirements**

None; this window is always available.

**Toolbar**

The toolbar contains:

- **Go to**
  The memory location or symbol you want to view.

- **Zone**
  Selects a memory zone, see *C-SPY memory zones*, page 155.

- **Toggle Mixed-Mode**
  Toggles between displaying only disassembled code or disassembled code together with the corresponding source code. Source code requires that the corresponding source file has been compiled with debug information.

**Display area**

The display area shows the disassembled application code.

This area contains these graphic elements:

- **Green highlight**
  Indicates the current position, that is the next assembler instruction to be executed. To move the cursor to any line in the Disassembly window, click the line. Alternatively, move the cursor using the navigation keys.

- **Yellow highlight**
  Indicates a position other than the current position, such as when navigating between frames in the Call Stack window or between items in the Trace window.

- **Red dot**
  Indicates a breakpoint. Double-click in the gray left-side margin of the window to set a breakpoint. For more information, see *Breakpoints*, page 119.

- **Green diamond**
  Indicates code that has been executed—that is, code coverage.

If instruction profiling has been enabled from the context menu, an extra column in the left-side margin appears with information about how many times each instruction has been executed.
Reference information on application execution

Context menu

This context menu is available:

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move to PC</td>
<td>Displays code at the current program counter location.</td>
</tr>
<tr>
<td>Run to Cursor</td>
<td>Executes the application from the current position up to the line containing the cursor.</td>
</tr>
<tr>
<td>Code Coverage</td>
<td>Displays a submenu that provides commands for controlling code coverage. This command is only enabled if the driver you are using supports it.</td>
</tr>
<tr>
<td>Enable</td>
<td>Toggles code coverage on or off.</td>
</tr>
<tr>
<td>Show</td>
<td>Toggles the display of code coverage on or off. Executed code is indicated by a green diamond.</td>
</tr>
<tr>
<td>Clear</td>
<td>Clears all code coverage information.</td>
</tr>
<tr>
<td>Instruction Profiling</td>
<td>Displays a submenu that provides commands for controlling instruction profiling. This command is only enabled if the driver you are using supports it.</td>
</tr>
<tr>
<td>Enable</td>
<td>Toggles instruction profiling on or off.</td>
</tr>
</tbody>
</table>

Note: The contents of this menu are dynamic, which means it might look different depending on your product package.

These commands are available:

Move to PC

Displays code at the current program counter location.

Run to Cursor

Executes the application from the current position up to the line containing the cursor.

Code Coverage

Displays a submenu that provides commands for controlling code coverage. This command is only enabled if the driver you are using supports it.

Enable

Toggles code coverage on or off.

Show

Toggles the display of code coverage on or off. Executed code is indicated by a green diamond.

Clear

Clears all code coverage information.

Instruction Profiling

Displays a submenu that provides commands for controlling instruction profiling. This command is only enabled if the driver you are using supports it.

Enable

Toggles instruction profiling on or off.
Executing your application

Show
Toggles the display of instruction profiling on or off. For each instruction, the left-side margin displays how many times the instruction has been executed.

Clear
Clears all instruction profiling information.

Toggle Breakpoint (Code)
Toggles a code breakpoint. Assembler instructions and any corresponding label at which code breakpoints have been set are highlighted in red. For more information, see Code breakpoints dialog box, page 134.

Toggle Breakpoint (Log)
Toggles a log breakpoint for trace printouts. Assembler instructions at which log breakpoints have been set are highlighted in red. For more information, see Log breakpoints dialog box, page 139.

Toggle Breakpoint (Trace Start)
Toggles a Trace Start breakpoint. When the breakpoint is triggered, the trace data collection starts. Note that this menu command is only available if the C-SPY driver you are using supports trace. For more information, see Trace Start breakpoints dialog box, page 220.

Toggle Breakpoint (Trace Stop)
Toggles a Trace Stop breakpoint. When the breakpoint is triggered, the trace data collection stops. Note that this menu command is only available if the C-SPY driver you are using supports trace. For more information, see Trace Stop breakpoints dialog box, page 221.

Enable/Disable Breakpoint
Enables and Disables a breakpoint. If there is more than one breakpoint at a specific line, all those breakpoints are affected by the Enable/Disable command.

Edit Breakpoint
Displays the breakpoint dialog box to let you edit the currently selected breakpoint. If there is more than one breakpoint on the selected line, a submenu is displayed that lists all available breakpoints on that line.

Set Next Statement
Sets the program counter to the address of the instruction at the insertion point.

Copy Window Contents
Copies the selected contents of the Disassembly window to the clipboard.
Mixed-Mode

Toggles between showing only disassembled code or disassembled code together with the corresponding source code. Source code requires that the corresponding source file has been compiled with debug information.

Call Stack window

The Call stack window is available from the View menu.

Figure 16: Call Stack window

This window displays the C function call stack with the current function at the top. To inspect a function call, double-click it. C-SPY now focuses on that call frame instead.

If the next Step Into command would step to a function call, the name of the function is displayed in the grey bar at the top of the window. This is especially useful for implicit function calls, such as C++ constructors, destructors, and operators.

Requirements

None; this window is always available.

Display area

Provided that the command Show Arguments is enabled, each entry in the display area has the format:

\[ \text{Function(values)} \]

where \( \text{values} \) is a list of the current value of the parameters, or empty if the function does not take any parameters.
Context menu

This context menu is available:

- Go to Source
- Show Arguments
- Run to Cursor
- Toggle Breakpoint (Code)
- Toggle Breakpoint (Log)
- Enable/Disable Breakpoint

Figure 17: Call Stack window context menu

These commands are available:

Go to Source
Displays the selected function in the Disassembly or editor windows.

Show Arguments
Shows function arguments.

Run to Cursor
Executes until return to the function selected in the call stack.

Toggle Breakpoint (Code)
Toggles a code breakpoint.

Toggle Breakpoint (Log)
Toggles a log breakpoint.

Enable/Disable Breakpoint
Enables or disables the selected breakpoint
Terminal I/O window

The Terminal I/O window is available from the View menu.

Figure 18: Terminal I/O window

Use this window to enter input to your application, and display output from it.

To use this window, you must:

1. Link your application with the option **Semihosted** or the **IAR breakpoint**.

C-SPY will then direct stdin, stdout and stderr to this window. If the Terminal I/O window is closed, C-SPY will open it automatically when input is required, but not for output.

Requirements

None; this window is always available.

Input

Type the text that you want to input to your application.

Ctrl codes

Opens a menu for input of special characters, such as **EOF** (end of file) and **NUL**.

Figure 19: Ctrl codes menu
Input Mode

Opens the Input Mode dialog box where you choose whether to input data from the keyboard or from a file.

![Input Mode dialog box](image)

*Figure 20: Input Mode dialog box*

For reference information about the options available in this dialog box, see Terminal I/O options in *IDE Project Management and Building Guide for ARM*.

Terminal I/O Log File dialog box

The Terminal I/O Log File dialog box is available by choosing Debug>Logging>Set Terminal I/O Log File.

![Terminal I/O Log File dialog box](image)

*Figure 21: Terminal I/O Log File dialog box*

Use this dialog box to select a destination log file for terminal I/O from C-SPY.

**Requirements**

None; this dialog box is always available.

**Terminal IO Log Files**

Controls the logging of terminal I/O. To enable logging of terminal I/O to a file, select Enable Terminal IO log file and specify a filename. The default filename extension is *log*. A browse button is available for your convenience.
The Debug Log window is available by choosing View>Messages.

This window displays debugger output, such as diagnostic messages, macro-generated output, event log messages, and information about trace. This output is only available during a debug session. When opened, this window is, by default, grouped together with the other message windows, see IDE Project Management and Building Guide for ARM.

Double-click any rows in one of the following formats to display the corresponding source code in the editor window:

<path> \{<row>:<message>
<path> \{<row>,<column>:<message>

Requirements

None; this window is always available.

Context menu

This context menu is available:

- Copy
- Select All
- Clear All

These commands are available:

Copy
Copies the contents of the window.

Select All
Selects the contents of the window.

Clear All
Clears the contents of the window.
Log File dialog box

The Log File dialog box is available by choosing Debug>Logging>Set Log File.

![Log File dialog box](image)

*Figure 24: Log File dialog box*

Use this dialog box to log output from C-SPY to a file.

**Requirements**

None; this dialog box is always available.

**Enable Log file**

Enables or disables logging to the file.

**Include**

The information printed in the file is, by default, the same as the information listed in the Log window. Use the browse button, to override the default file and location of the log file (the default filename extension is `.log`). To change the information logged, choose between:

**Errors**

C-SPY has failed to perform an operation.

**Warnings**

An error or omission of concern.

**Info**

Progress information about actions C-SPY has performed.

**User**

Messages from C-SPY macros, that is, your messages using the `__message` statement.
Autostep settings dialog box

The Autostep settings dialog box is available from the Debug menu.

![Autostep settings dialog box](image)

Figure 25: Autostep settings dialog box

Use this dialog box to customize autostepping.

The drop-down menu lists the available step commands.

**Requirements**

None; this dialog box is always available.

**Delay**

Specify the delay between each step in milliseconds.
Variables and expressions

This chapter describes how variables and expressions can be used in C-SPY®. More specifically, this means:

- Introduction to working with variables and expressions
- Working with variables and expressions
- Reference information on working with variables and expressions.

Introduction to working with variables and expressions

This section covers these topics:

- Briefly about working with variables and expressions
- C-SPY expressions
- Limitations on variable information.

BRIEFLY ABOUT WORKING WITH VARIABLES AND EXPRESSIONS

There are several methods for looking at variables and calculating their values:

- Tooltip watch—in the editor window—provides the simplest way of viewing the value of a variable or more complex expressions. Just point at the variable with the mouse pointer. The value is displayed next to the variable.
- The Auto window displays a useful selection of variables and expressions in, or near, the current statement. The window is automatically updated when execution stops.
- The Locals window displays the local variables, that is, auto variables and function parameters for the active function. The window is automatically updated when execution stops.
- The Watch window allows you to monitor the values of C-SPY expressions and variables. The window is automatically updated when execution stops.
- The Live Watch window repeatedly samples and displays the values of expressions while your application is executing. Variables in the expressions must be statically located, such as global variables.
- The Statics window displays the values of variables with static storage duration. The window is automatically updated when execution stops.
In Introduction to working with variables and expressions:

- The Macro Quicklaunch window and the Quick Watch window give you precise control over when to evaluate an expression.
- The Symbols window displays all symbols with a static location, that is, C/C++ functions, assembler labels, and variables with static storage duration, including symbols from the runtime library.
- The Data Log window and the Data Log Summary window display logs of accesses up to four different memory locations or areas you choose by setting Data Log breakpoints. Data logging can help you locate frequently accessed data. You can then consider whether you should place that data in more efficient memory.
- The Event Log window and the Event Log Summary window display event logs produced when the execution passes specific positions in your application code. The Timeline window graphically displays these event logs correlated to a common time-axis. Event logging can help you analyze program flow and inspect data correlated to a certain position in your application code. Event logs are supported by the C-SPY J-Link/J-Trace driver.
- The Cortex ITM communication channels are used for passing events from a running application to the C-SPY Event log system. There are predefined preprocessor macros that you can use in your application source code. An Event log will be generated every time such macros are passed during program execution. You can pass a value with each event. Typically, this value can be an identifier or the content of a variable or a register (for example, the stack pointer). The value can be written in 8, 16, or 32-bit format. Using a smaller size will reduce the bandwidth needed on the SWO wire. Events can be generated with or without an associated PC (program counter) value, the PC value makes it possible for the debugger to correlate the event to the executed code.
- The Trace-related windows let you inspect the program flow up to a specific state. For more information, see Trace, page 187.

**C-SPY EXPRESSIONS**

C-SPY expressions can include any type of C expression, except for calls to functions. The following types of symbols can be used in expressions:

- C/C++ symbols
- Assembler symbols (register names and assembler labels)
- C-SPY macro functions
- C-SPY macro variables.
Expressions that are built with these types of symbols are called C-SPY expressions and there are several methods for monitoring these in C-SPY. Examples of valid C-SPY expressions are:

\[
\begin{align*}
i + j \\
i &= 42 \\
myVar &= cVar \\
cVar &= myVar + 2 \\
\text{#asm_label} \\
\text{#R2} \\
\text{#PC} \\
\text{my_macro_func(19)}
\end{align*}
\]

If you have a static variable with the same name declared in several different functions, use the notation `function::variable` to specify which variable to monitor.

**C/C++ symbols**

C symbols are symbols that you have defined in the C source code of your application, for instance variables, constants, and functions (functions can be used as symbols but cannot be executed). C symbols can be referenced by their names. Note that C++ symbols might implicitly contain function calls which are not allowed in C-SPY symbols and expressions.

**Assembler symbols**

Assembler symbols can be assembler labels or registers, for example the program counter, the stack pointer, or other CPU registers. If a device description file is used, all memory-mapped peripheral units, such as I/O ports, can also be used as assembler symbols in the same way as the CPU registers. See *Modifying a device description file*, page 57.

Assembler symbols can be used in C-SPY expressions if they are prefixed by `#`.  

<table>
<thead>
<tr>
<th>Example</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>#PC++</td>
<td>Increments the value of the program counter.</td>
</tr>
<tr>
<td>myVar = #SP</td>
<td>Assigns the current value of the stack pointer register to your C-SPY variable.</td>
</tr>
<tr>
<td>myVar = #label</td>
<td>Sets myVar to the value of an integer at the address of label.</td>
</tr>
<tr>
<td>myptr = #label7</td>
<td>Sets myptr to an int * pointer pointing at label7.</td>
</tr>
</tbody>
</table>

*Table 6: C-SPY assembler symbols expressions*
In case of a name conflict between a hardware register and an assembler label, hardware registers have a higher precedence. To refer to an assembler label in such a case, you must enclose the label in back quotes ` (ASCII character 0x60). For example:

<table>
<thead>
<tr>
<th>Example</th>
<th>What it does</th>
</tr>
</thead>
<tbody>
<tr>
<td>#PC</td>
<td>Refers to the program counter.</td>
</tr>
<tr>
<td>#<code>PC</code></td>
<td>Refers to the assembler label PC.</td>
</tr>
</tbody>
</table>

Which processor-specific symbols are available by default can be seen in the Register window, using the CPU Registers register group. See Register window, page 172.

**C-SPY macro functions**

Macro functions consist of C-SPY macro variable definitions and macro statements which are executed when the macro is called.

For information about C-SPY macro functions and how to use them, see Briefly about the macro language, page 295.

**C-SPY macro variables**

Macro variables are defined and allocated outside your application, and can be used in a C-SPY expression. In case of a name conflict between a C symbol and a C-SPY macro variable, the C-SPY macro variable will have a higher precedence than the C variable. Assignments to a macro variable assign both its value and type.

For information about C-SPY macro variables and how to use them, see Reference information on the macro language, page 301.

**Using sizeof**

According to standard C, there are two syntactical forms of sizeof:

- `sizeof(type)`
- `sizeof(expr)`

The former is for types and the latter for expressions.

**Note:** In C-SPY, do not use parentheses around an expression when you use the `sizeof` operator. For example, use `sizeof x+2` instead of `sizeof (x+2)`.

**LIMITATIONS ON VARIABLE INFORMATION**

The value of a C variable is valid only on step points, that is, the first instruction of a statement and on function calls. This is indicated in the editor window with a bright green highlight color. In practice, the value of the variable is accessible and correct more often than that.
When the program counter is inside a statement, but not at a step point, the statement or part of the statement is highlighted with a pale variant of the ordinary highlight color.

**Effects of optimizations**

The compiler is free to optimize the application software as much as possible, as long as the expected behavior remains. The optimization can affect the code so that debugging might be more difficult because it will be less clear how the generated code relates to the source code. Typically, using a high optimization level can affect the code in a way that will not allow you to view a value of a variable as expected.

Consider this example:

```c
myFunction()
{
  int i = 42;
  ...
  x = computer(i); /* Here, the value of i is known to C-SPY */
  ...
}
```

From the point where the variable `i` is declared until it is actually used, the compiler does not need to waste stack or register space on it. The compiler can optimize the code, which means that C-SPY will not be able to display the value until it is actually used. If you try to view the value of a variable that is temporarily unavailable, C-SPY will display the text:

Unavailable

If you need full information about values of variables during your debugging session, you should make sure to use the lowest optimization level during compilation, that is, **None**.

**Working with variables and expressions**

This section describes various tasks related to working with variables and expressions.

More specifically, you will get information about:

- Using the windows related to variables and expressions
- Viewing assembler variables
- Getting started using data logging
- Getting started using event logging.
USING THE WINDOWS RELATED TO VARIABLES AND EXPRESSIONS

Where applicable, you can add, modify, and remove expressions, and change the display format in the windows related to variables and expressions.

To add a value you can also click in the dotted rectangle and type the expression you want to examine. To modify the value of an expression, click the Value field and modify its content. To remove an expression, select it and press the Delete key.

For text that is too wide to fit in a column—in any of the these windows, except the Trace window—and thus is truncated, just point at the text with the mouse pointer and tooltip information is displayed.

Right-click in any of the windows to access the context menu which contains additional commands. Convenient drag-and-drop between windows is supported, except for in the Locals window, Data logging windows, and the Quick Watch window where it is not relevant.

VIEWING ASSEMBLER VARIABLES

An assembler label does not convey any type information at all, which means C-SPY cannot easily display data located at that label without getting extra information. To view data conveniently, C-SPY by default treats all data located at assembler labels as variables of type int. However, in the Watch, Quick Watch, and Live Watch windows, you can select a different interpretation to better suit the declaration of the variables.
In this figure, you can see four variables in the Watch window and their corresponding declarations in the assembler source file to the left:

![Watch window with variables](image)

**Figure 26: Viewing assembler variables in the Watch window**

Note that `asmvar4` is displayed as an `int`, although the original assembler declaration probably intended for it to be a single byte quantity. From the context menu you can make C-SPY display the variable as, for example, an 8-bit unsigned variable. This has already been specified for the `asmvar3` variable.

### GETTING STARTED USING DATA LOGGING

1. To set up for data logging, choose **C-SPY driver>SWO Configuration**. In the dialog box, set up the serial-wire output communication channel for trace data. Note specifically the **CPU clock** option. For the ST-LINK driver, the CPU clock can also be set up on the **Project>Options>ST-LINK** page. If you are using the C-SPY simulator you can ignore this step.

2. In the Breakpoints or Memory window, right-click and choose **New Breakpoints>Data Log** to open the breakpoints dialog box. Set a Data Log breakpoint on the data you want to collect log information for. You can set up to four Data Log breakpoints.

3. Choose **C-SPY driver>Data Log** to open the Data Log window. Optionally, you can also choose:
   - **C-SPY driver>Data Log Summary** to open the Data Log Summary window
   - **C-SPY driver>Timeline** to open the Timeline window to view the Data Log graph.
4. From the context menu, available in the Data Log window, choose Enable to enable the logging.

5. In the SWO Configuration dialog box, you can notice in the Data Log Evens area that Data Logs are enabled. Choose which level of logging you want:
   - PC only
   - PC + data value + base addr
   - Data value + exact addr

   If you are using the C-SPY simulator you can ignore this step.

6. Start executing your application program to collect the log information.

7. To view the data log information, look in any of the Data Log, Data Log Summary, or the Data graph in the Timeline window.

8. If you want to save the log or summary to a file, choose Save to log file from the context menu in the window in question.

9. To disable data and interrupt logging, choose Disable from the context menu in each window where you have enabled it.

GETTING STARTED USING EVENT LOGGING

1. To specify the position in your application source code that you want to generate event logs for, use the predefined preprocessor macros in arm_itm.h (located in arm/inc/c). In your application source code, write (for example):

   ```c
   #include <arm_itm.h>
   void func(void)
   {
     ITM_EVENT8_WITH_PC(1,25);
     ITM_EVENT32_WITH_PC(2, __get_PSP());
   }
   ```

   The first line sends an event with the value 25 to channel 1. The second line sends an event with the current value of the stack pointer to channel 2, which means that C-SPY can display the stack pointer at a code position of your choice. When these source lines are passed during program execution, events will be generated and visualized by C-SPY, which means that you can further analyze them.

2. To view event log information, you can choose between these alternatives:
   - Choose C-SPY driver>Timeline to open the Timeline window and choose Enable from the context menu. You can now view events for each channel as a graph (Event Log Graph).
Choose C-SPY driver > Event Log to open the Event Log window and choose Enable from the context menu. You can now view the events for each channel as numbers.

Choose C-SPY driver > Event Log Summary to open the Event Log Summary window and choose Enable from the context menu. You will now get a summary of all event logs.

Note: Whenever the Event Log Graph or the Event Log window is enabled, you can at any time enable also the Event Log Summary window to get a summary. However, if you have enabled the Event Log Summary window, but not the Event Log window or the Event Log Graph in the Timeline window, you can get a summary but not detailed information about event logs.

3 To change the display format (you can choose between displaying values in hexadecimal or in decimal format), select the event graph for which you want to change the format in the Timeline window. Right-click and choose the display format of your choice from the context menu. Note that this setting affects also the Event Log window and the Event Log Summary window.

4 Start executing your application program to collect the log information.

5 To view the event log information, look at either the Event Log, the Event Log Summary, or the event graph for the specific channel in the Timeline window.

6 If you want to save the log or summary to a file, choose Save to log file from the context menu in the window.

7 To disable event logging, choose Disable from the context menu in each window where you have enabled it.

Reference information on working with variables and expressions

This section gives reference information about these windows and dialog boxes:

- Auto window, page 100
- Locals window, page 100
- Watch window, page 101
- Live Watch window, page 103
- Statics window, page 104
- Macro Quicklaunch window, page 106
- Quick Watch window, page 108
- Symbols window, page 109
- Resolve Symbol Ambiguity dialog box, page 111
Reference information on working with variables and expressions

- **Data Log window**, page 112
- **Data Log Summary window**, page 114
- **Event Log window**, page 116
- **Event Log Summary window**, page 117.

For trace-related reference information, see **Reference information on trace**, page 194.

**Auto window**

The Auto window is available from the **View** menu.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Location</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>5</td>
<td>R4</td>
<td>short</td>
</tr>
<tr>
<td>Fib[i]</td>
<td>0</td>
<td>0x00102214</td>
<td>unsigned int</td>
</tr>
<tr>
<td>Fb</td>
<td>array</td>
<td>0x00102200</td>
<td>unsigned int[10]</td>
</tr>
<tr>
<td>GooFib</td>
<td>GooFib[i]</td>
<td>0x2</td>
<td>unsigned int[]</td>
</tr>
</tbody>
</table>

*Figure 27: Auto window*

This window displays a useful selection of variables and expressions in, or near, the current statement. Every time execution in C-SPY stops, the values in the Auto window are recalculated. Values that have changed since the last stop are highlighted in red.

**Requirements**

None; this window is always available.

**Context menu**

For more information about the context menu, see **Watch window**, page 101.

**Locals window**

The Locals window is available from the **View** menu.

*Figure 28: Locals window*
This window displays the local variables and parameters for the current function. Every time execution in C-SPY stops, the values in the Locals window are recalculated. Values that have changed since the last stop are highlighted in red.

Requirements

None; this window is always available.

Context menu

For more information about the context menu, see Watch window, page 101.

Watch window

The Watch window is available from the View menu.

![Watch window](image)

Figure 29: Watch window

Use this window to monitor the values of C-SPY expressions or variables. You can open up to four instances of this window, where you can view, add, modify, and remove expressions. Tree structures of arrays, structs, and unions are expandable, which means that you can study each item of these.

Every time execution in C-SPY stops, the values in the Watch window are recalculated. Values that have changed since the last stop are highlighted in red.

Requirements

None; this window is always available.
Context menu

This context menu is available:

![Context menu screenshot](image)

These commands are available:

- **Add**
  Adds an expression.

- **Remove**
  Removes the selected expression.

- **Default Format**,  
  **Binary Format**,  
  **Octal Format**,  
  **Decimal Format**,  
  **Hexadecimal Format**,  
  **Char Format**

These commands are available if a selected line in the window contains a variable.

Changes the display format of expressions. The display format setting affects different types of expressions in different ways. Your selection of display format is saved between debug sessions. These commands are available if a selected line in the window contains a variable.

The display format setting affects different types of expressions in these ways:

- **Variables** The display setting affects only the selected variable, not other variables.
- **Array elements** The display setting affects the complete array, that is, the same display format is used for each array element.
- **Structure fields** All elements with the same definition—the same field name and C declaration type—are affected by the display setting.
Show As
Displays a submenu that provides commands for changing the default type interpretation of variables. The commands on this submenu are mainly useful for assembler variables—data at assembler labels—because these are, by default, displayed as integers. For more information, see Viewing assembler variables, page 96.

Live Watch window
The Live Watch window is available from the View menu.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Location</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>get_file</td>
<td>get_fh(0x1198)</td>
<td>Memory/0x1198</td>
<td>unsigned int[]</td>
</tr>
<tr>
<td>get_fh</td>
<td>get_fh(0x1198)</td>
<td>Memory/0x1198</td>
<td>unsigned int[]</td>
</tr>
</tbody>
</table>

Figure 31: Live Watch window
This window repeatedly samples and displays the value of expressions while your application is executing. Variables in the expressions must be statically located, such as global variables.

This window can only be used for hardware target systems supporting this feature.

Requirements
None; this window is always available.

Context menu
For more information about the context menu, see Watch window, page 101.

In addition, the menu contains the Options command, which opens the Debugger dialog box where you can set the Update interval option. The default value of this option is 1000 milliseconds, which means the Live Watch window will be updated once every second during program execution.
Statics window

The Statics window is available from the View menu.

<table>
<thead>
<tr>
<th>Expression</th>
<th>Value</th>
<th>Location</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>cell_count</td>
<td>0</td>
<td>DATA:0x00000010</td>
<td>r0</td>
</tr>
<tr>
<td>root</td>
<td>1</td>
<td>DATA:0x00000006</td>
<td>unsigned r0</td>
</tr>
<tr>
<td>[0]</td>
<td>2</td>
<td>DATA:0x00000008</td>
<td>unsigned r0</td>
</tr>
<tr>
<td>[1]</td>
<td>0</td>
<td>DATA:0x00000004</td>
<td>unsigned r0</td>
</tr>
<tr>
<td>[2]</td>
<td>0</td>
<td>DATA:0x00000004</td>
<td>unsigned r0</td>
</tr>
<tr>
<td>[3]</td>
<td>0</td>
<td>DATA:0x00000004</td>
<td>unsigned r0</td>
</tr>
<tr>
<td>[4]</td>
<td>0</td>
<td>DATA:0x00000004</td>
<td>unsigned r0</td>
</tr>
<tr>
<td>[5]</td>
<td>0</td>
<td>DATA:0x00000004</td>
<td>unsigned r0</td>
</tr>
<tr>
<td>[6]</td>
<td>0</td>
<td>DATA:0x00000004</td>
<td>unsigned r0</td>
</tr>
<tr>
<td>[7]</td>
<td>0</td>
<td>DATA:0x00000004</td>
<td>unsigned r0</td>
</tr>
<tr>
<td>[8]</td>
<td>0</td>
<td>DATA:0x00000004</td>
<td>unsigned r0</td>
</tr>
<tr>
<td>[9]</td>
<td>0</td>
<td>DATA:0x00000004</td>
<td>unsigned r0</td>
</tr>
</tbody>
</table>

Figure 32: Statics window

This window displays the values of variables with static storage duration that you have selected. Typically, that is variables with file scope but it can also be static variables in functions and classes. Note that volatile declared variables with static storage duration will not be displayed.

Every time execution in C-SPY stops, the values in the Statics window are recalculated. Values that have changed since the last stop are highlighted in red.

To select variables to monitor:

1. In the window, right-click and choose Select statics from the context menu. The window now lists all variables with static storage duration.
2. Either individually select the variables you want to be displayed, or choose Select All or Deselect All from the context menu.
3. When you have made your selections, choose Select statics from the context menu to toggle back to the normal display mode.

Requirements

None; this window is always available.

Display area

This area contains these columns:

Expression

The name of the variable. The base name of the variable is followed by the full name, which includes module, class, or function scope. This column is not editable.
Variables and expressions

Value
The value of the variable. Values that have changed are highlighted in red.
Dragging text or a variable from another window and dropping it on the Value column will assign a new value to the variable in that row.
This column is editable.

Location
The location in memory where this variable is stored.

Type
The data type of the variable.

Context menu
This context menu is available:

Default Format,
Binary Format,
Octal Format,
Decimal Format,
Hexadecimal Format,
Char Format

Changes the display format of expressions. The display format setting affects different types of expressions in different ways. Your selection of display format is saved between debug sessions. These commands are available if a selected line in the window contains a variable.
Reference information on working with variables and expressions

The display format setting affects different types of expressions in these ways:

- **Variables**: The display setting affects only the selected variable, not other variables.
- **Array elements**: The display setting affects the complete array, that is, the same display format is used for each array element.
- **Structure fields**: All elements with the same definition—the same field name and C declaration type—are affected by the display setting.

**Select Statics**
Lists all variables with static storage duration. Select the variables you want to be monitored. When you have made your selections, select this menu command again to toggle back to normal display mode.

- **Select all**: Selects all variables.
- **Deselect all**: Deselects all variables.

**Macro Quicklaunch window**

The Macro Quicklaunch window is available from the View menu.

Use this window to evaluate expressions, typically C-SPY macros.

The Macro Quicklaunch window is similar to the Quick Watch window, but is primarily designed for evaluating C-SPY macros. The window gives you precise control over when to evaluate an expression.
To add an expression:

1. Choose one of these alternatives:
   - Drag the expression to the window
   - In the Expression column, type the expression you want to examine.

If the expression you add and want to evaluate is a C-SPY macro, the macro must first be registered, see Registering C-SPY macros—an overview, page 296.

To evaluate an expression:

Double-click the Recalculate icon to calculate the value of that expression.

Requirements

None; this window is always available.

Display area

This area contains these columns:

- **Recalculate icon**
  - To evaluate the expression, double-click the icon. The latest evaluated expression appears in bold style.

- **Expression**
  - One or several expressions that you want to evaluate. Click <click to add> to add an expression. If the return value has changed since last time, the value will be displayed in red.

- **Result**
  - Shows the return value from the expression evaluation.

Context menu

This context menu is available:

- **Evaluate Now**
- **Remove**
- **Remove All**

Figure 35: Macro Quicklaunch window context menu

These commands are available:

- **Evaluate now**
  - Adds an expression.

- **Remove**
  - Removes the selected expression.
Quick Watch window

The Quick Watch window is available from the View menu and from the context menu in the editor window.

![Quick Watch window](image)

Use this window to watch the value of a variable or expression and evaluate expressions at a specific point in time.

In contrast to the Watch window, the Quick Watch window gives you precise control over when to evaluate the expression. For single variables this might not be necessary, but for expressions with possible side effects, such as assignments and C-SPY macro functions, it allows you to perform evaluations under controlled conditions.

To evaluate an expression:

1. In the editor window, right-click on the expression you want to examine and choose Quick Watch from the context menu that appears.
2. The expression will automatically appear in the Quick Watch window. Alternatively:
3. In the Quick Watch window, type the expression you want to examine in the Expressions text box.
4. Click the Recalculate button to calculate the value of the expression.

For an example, see Executing macros using Quick Watch, page 299.

Requirements

None; this window is always available.
Variables and expressions

Context menu

This context menu is available:

- **Add**
  - Adds an expression.
- **Remove**
  - Removes the selected expression.

In addition, the menu contains the Add to Watch window command, which adds the selected expression to the Watch window.

Symbols window

The Symbols window is available from the View menu.

This window displays all symbols with a static location, that is, C/C++ functions, assembler labels, and variables with static storage duration, including symbols from the runtime library.

Requirements

None; this window is always available.
Display area

This area contains these columns:

Symbol
The symbol name.

Location
The memory address.

Full name
The symbol name; often the same as the contents of the Symbol column but differs for example for C++ member functions.

Click the column headers to sort the list by symbol name, location, or full name.

Context menu

This context menu is available:

<table>
<thead>
<tr>
<th>Functions</th>
<th>Variables</th>
<th>Labels</th>
</tr>
</thead>
</table>

Figure 39: Symbols window context menu

These commands are available:

Functions
Toggles the display of function symbols on or off in the list.

Variables
Toggles the display of variables on or off in the list.

Labels
Toggles the display of labels on or off in the list.
Resolve Symbol Ambiguity dialog box

The Resolve Symbol Ambiguity dialog box appears, for example, when you specify a symbol in the Disassembly window to go to, and there are several instances of the same symbol due to templates or function overloading.

Requirements
None; this window is always available.

Ambiguous symbol
Indicates which symbol that is ambiguous.

Please select one symbol
A list of possible matches for the ambiguous symbol. Select the one you want to use.
Data Log window

The Data Log window is available from the C-SPY driver menu.

![Data Log window](image)

Use this window to log accesses to up to four different memory locations or areas.

See also *Getting started using data logging*, page 97.

Requirements

A Cortex-M device and one of these alternatives:

- The C-SPY simulator
- The C-SPY I-jet/JTAGjet driver and an I-jet in-circuit debugging probe with an SWD interface between the debug probe and the target system
- The C-SPY J-Link/J-Trace driver and a J-Link or J-Trace debug probe with an SWD interface between the debug probe and the target system
- The C-SPY ST-LINK driver and an ST-LINK debug probe with an SWD interface between the debug probe and the target system

For J-Trace, this window is available when ETM trace is disabled. When debugging, this window only displays a limited amount of the collected trace data when ETM is enabled. The entire trace data is displayed when the execution is stopped.
Display area

Each row in the display area shows the time, the program counter, and, for every tracked data object, its value and address in these columns:

**Time**

For the C-SPY I-jet driver, the time for the data access is based on a dedicated 48-MHz clock.

The time for the data access for the C-SPY J-Link driver, the C-SPY ST-LINK driver, and the simulator, based on the clock frequency. For the C-SPY J-Link driver and the C-SPY ST-LINK driver, this is specified in the SWO Configuration dialog box.

If the time is displayed in italics, the target system has not been able to collect a correct time, but instead had to approximate it.

This column is available when you have selected **Show cycles** from the context menu.

**Cycles**

The number of cycles from the start of the execution until the event. This information is cleared at reset.

If a cycle is displayed in italics, the target system has not been able to collect a correct time, but instead had to approximate it.

This column is available when you have selected **Show cycles** from the context menu.

**Program Counter**

Displays one of these:

- An address, which is the content of the PC, that is, the address of the instruction that performed the memory access.
- ---, the target system failed to provide the debugger with any information.
- **Overflow** in red, the communication channel failed to transmit all data from the target system.

**Value**

Displays the access type and the value (using the access size) for the location or area you want to log accesses to. For example, if zero is read using a byte access it will be displayed as 0x00, and for a long access it will be displayed as 0x00000000.

To specify what data you want to log accesses to, use the **Data Log** breakpoint dialog box. See **Data Log breakpoints**, page 121.
Address

The actual memory address that is accessed. For example, if only a byte of a word is accessed, only the address of the byte is displayed. The address is calculated as base address + offset, where the base address is retrieved from the Data Log breakpoint dialog box and the offset is retrieved from the logs. If the log from the target system does not provide the debugger with an offset, the offset contains + ?. If you want the offset to be displayed (for the C-SPY I-jet driver, the C-SPY J-Link driver, and the C-SPY ST-LINK driver), select the Value + exact addr option in the SWO Configuration dialog box.

* You can double-click a line in the display area. If the value of the PC for that line is available in the source code, the editor window displays the corresponding source code (this does not include library source code).

Context menu

Identical to the context menu of the Interrupt Log window, see Interrupt Log window, page 287.

Data Log Summary window

The Data Log Summary window is available from the C-SPY driver menu.

![Data Log Summary window](image)

This window displays a summary of data accesses to specific memory location or areas. See also Getting started using data logging, page 97.

Requirements

A Cortex-M device and one of these alternatives:

- The C-SPY simulator
- The C-SPY I-jet/JTAGjet driver and an I-jet in-circuit debugging probe with an SWD interface between the debug probe and the target system
- The C-SPY J-Link/J-Trace driver and a J-Link or J-Trace debug probe with an SWD interface between the debug probe and the target system
For J-Trace, this window is available when ETM trace is disabled. When debugging, this window only displays a limited amount of the collected trace data when ETM is enabled. The entire trace data is displayed when the execution is stopped.

- The C-SPY ST-LINK driver and a ST-LINK debug probe with an SWD interface between the debug probe and the target system

**Display area**

Each row in this area displays the type and the number of accesses to each memory location or area in these columns:

**Data**

The name of the data object you have selected to log accesses to. To specify what data object you want to log accesses to, use the Data Log breakpoint dialog box. See *Data Log breakpoints*, page 121.

At the bottom of the column Overflow count displays the number of overflows.

**Total accesses**

The number of total accesses.

If the sum of read accesses and write accesses is less than the total accesses, there have been a number of access logs for which the target system for some reason did not provide valid access type information.

**Read accesses**

The number of total read accesses.

**Write accesses**

The number of total write accesses.

**Context menu**

Identical to the context menu of the Interrupt Log window, see *Interrupt Log window*, page 287.
Event Log window

The Event Log window is available from the C-SPY driver menu.

![Figure 43: Event Log window](image)

This window displays event logs produced when the execution passes specific positions in your application code. The Cortex ITM communication channels are used for passing the event logs from a running application to the C-SPY Event Log system.

See also *Getting started using event logging*, page 98.

Requirements

One of these alternatives:

- The C-SPY I-jet/JTAGjet driver and an I-jet in-circuit debugging probe with an SWD interface between the debug probe and the target system
- The C-SPY J-Link/J-Trace driver and a J-Link or J-Trace debug probe with an SWD interface between the debug probe and the target system

Display area

Each row in the display area shows the event logs in these columns:

**Cycles**

The number of cycles from the start of the execution until the event. This information is cleared at reset.

If a cycle is displayed in italics, the target system has not been able to collect a correct time, but instead had to approximate it.

This column is available when you have selected *Show cycles* from the context menu.

**Program Counter**

An address, which is the content of the PC, that is, the address of the instruction that performed the memory access.
---, the target system failed to provide the debugger with any information.

Overflow in red, the communication channel failed to transmit all data from the target system.

ITM1
ITM2
ITM3
ITM4

The Cortex ITM communication channels for which the events are logged. For each event log, the event value is displayed.

Add a preprocessor macro to your application source code where you want event logs to be generated. See Getting started using event logging, page 98.

Context menu

Identical to the context menu of the Interrupt Log window, see Interrupt Log window, page 287.

Event Log Summary window

The Event Log Summary window is available from the C-SPY driver menu.

<table>
<thead>
<tr>
<th>Channel</th>
<th>Count</th>
<th>Average Value</th>
<th>Min Value</th>
<th>Max Value</th>
<th>Average Interval</th>
<th>Min Interval</th>
<th>Max Interval</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ch0</td>
<td>13</td>
<td>0x12</td>
<td>0x12</td>
<td>0x12</td>
<td>506.72us</td>
<td>444.20us</td>
<td>1189.98us</td>
</tr>
<tr>
<td>Ch1</td>
<td>7</td>
<td>0x2</td>
<td>0x2</td>
<td>0x2</td>
<td>889.36us</td>
<td>888.40us</td>
<td>889.62us</td>
</tr>
<tr>
<td>Ch2</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ch3</td>
<td>1</td>
<td>0x4700</td>
<td>0x4700</td>
<td>0x4700</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Approximate time count: 0
Overflow count: 0
Current timer: 0507.38us

This window displays a summary of event logs produced when the execution passes specific positions in your application code. The Cortex ITM communication channels are used for passing the event logs from a running application to the C-SPY Event Log system.

Requirements

One of these alternatives:

- The C-SPY I-jet/JTAGjet driver and an I-jet in-circuit debugging probe with an SWD interface between the debug probe and the target system
Reference information on working with variables and expressions

- The C-SPY J-Link/J-Trace driver and a J-Link or J-Trace debug probe with an SWD interface between the debug probe and the target system

**Display area**

**Channel**
The name of the communication channel for which event logs are generated.
At the bottom of the column, the current time or cycles is displayed—execution time since the start of execution or the number of cycles. Overflow count displays the number of overflows.

**Count**
The number of logged events.

**Average Value**
The average value of all received event values.

**Min Value**
The smallest value of all received event values.

**Max Value**
The largest value of all received event values.

**Average Interval**
The average time (in cycles) between events.

**Min Interval**
The shortest time (in cycles) between two events.

**Max Interval**
The longest time (in cycles) between two events.

**Context menu**
Identical to the context menu of the Interrupt Log window, see *Interrupt Log window*, page 287.
Breakpoints

This chapter describes breakpoints and the various ways to define and monitor them. More specifically, this means:

- Introduction to setting and using breakpoints
- Setting breakpoints
- Reference information on breakpoints.

Introduction to setting and using breakpoints

This section introduces breakpoints. These topics are covered:

- Reasons for using breakpoints
- Briefly about setting breakpoints
- Breakpoint types
- Breakpoint icons
- Breakpoints in the C-SPY simulator
- Breakpoints in the C-SPY hardware drivers
- Breakpoint consumers.

REASONS FOR USING BREAKPOINTS

C-SPY® lets you set various types of breakpoints in the application you are debugging, allowing you to stop at locations of particular interest. You can set a breakpoint at a code location to investigate whether your program logic is correct, or to get trace printouts. In addition to code breakpoints, and depending on what C-SPY driver you are using, additional breakpoint types might be available. For example, you might be able to set a data breakpoint, to investigate how and when the data changes.

You can let the execution stop under certain conditions, which you specify. You can also let the breakpoint trigger a side effect, for instance executing a C-SPY macro function, by transparently stopping the execution and then resuming. The macro function can be defined to perform a wide variety of actions, for instance, simulating hardware behavior.

All these possibilities provide you with a flexible tool for investigating the status of your application.
Introduction to setting and using breakpoints

BRIEFLY ABOUT SETTING BREAKPOINTS
You can set breakpoints in many various ways, allowing for different levels of interaction, precision, timing, and automation. All the breakpoints you define will appear in the Breakpoints window. From this window you can conveniently view all breakpoints, enable and disable breakpoints, and open a dialog box for defining new breakpoints. The Breakpoint Usage window also lists all internally used breakpoints, see Breakpoint consumers, page 123.

Breakpoints are set with a higher precision than single lines, using the same mechanism as when stepping; for more information about the precision, see Single stepping, page 74.

You can set breakpoints while you edit your code even if no debug session is active. The breakpoints will then be validated when the debug session starts. Breakpoints are preserved between debug sessions.

Note: For most hardware debugger systems it is only possible to set breakpoints when the application is not executing.

BREAKPOINT TYPES
Depending on the C-SPY driver you are using, C-SPY supports different types of breakpoints.

Code breakpoints
Code breakpoints are used for code locations to investigate whether your program logic is correct or to get trace printouts. Code breakpoints are triggered when an instruction is fetched from the specified location. If you have set the breakpoint on a specific machine instruction, the breakpoint will be triggered and the execution will stop, before the instruction is executed.

Log breakpoints
Log breakpoints provide a convenient way to add trace printouts without having to add any code to your application source code. Log breakpoints are triggered when an instruction is fetched from the specified location. If you have set the breakpoint on a specific machine instruction, the breakpoint will be triggered and the execution will temporarily stop and print the specified message in the C-SPY Debug Log window.

Trace breakpoints
Trace Start and Stop breakpoints start and stop trace data collection—a convenient way to analyze instructions between two execution points.
Data breakpoints
Data breakpoints are primarily useful for variables that have a fixed address in memory. If you set a breakpoint on an accessible local variable, the breakpoint is set on the corresponding memory location. The validity of this location is only guaranteed for small parts of the code. Data breakpoints are triggered when data is accessed at the specified location. The execution will usually stop directly after the instruction that accessed the data has been executed.

Data Log breakpoints
Data Log breakpoints are triggered when data is accessed at the specified location. If you have set a breakpoint on a specific address or a range, a log message is displayed in the SWO Trace window (Trace window in the simulator) for each access to that location. A log message can also be displayed in the Data Log window. Data logs can also be displayed on the Data Log graph in the Timeline window, if that window is enabled. However, these log messages require that you have set up trace data in the SWO Configuration dialog box, see SWO Configuration dialog box, page 201.

Immediate breakpoints
The C-SPY Simulator lets you set immediate breakpoints, which will halt instruction execution only temporarily. This allows a C-SPY macro function to be called when the simulated processor is about to read data from a location or immediately after it has written data. Instruction execution will resume after the action.

This type of breakpoint is useful for simulating memory-mapped devices of various kinds (for instance serial ports and timers). When the simulated processor reads from a memory-mapped location, a C-SPY macro function can intervene and supply appropriate data. Conversely, when the simulated processor writes to a memory-mapped location, a C-SPY macro function can act on the value that was written.

JTAG watchpoints
The C-SPY J-Link/J-Trace driver and the C-SPY Macraigor driver can take advantage of the JTAG watchpoint mechanism in ARM7/9 cores.

The watchpoints are implemented using the functionality provided by the ARM EmbeddedICE™ macrocell. The macrocell is part of every ARM core that supports the JTAG interface. The EmbeddedICE watchpoint comparator compares the address bus, data bus, CPU control signals and external input signals with the defined watchpoint in real time. When all defined conditions are true, the program will break.

The watchpoints are implicitly used by C-SPY to set code breakpoints or data breakpoints in the application. When setting breakpoints in read/write memory, only one watchpoint is needed by the debugger. When setting breakpoints in read-only memory,
one watchpoint is needed for each breakpoint. Because the macrocell only implements two hardware watchpoints, the maximum number of breakpoints in read-only memory is two.

For a more detailed description of the ARM JTAG watchpoint mechanism, refer to these documents from Advanced RISC Machines Ltd:

- Application Note 28, The ARM7TDMI Debug Architecture.

**BREAKPOINT ICONS**

A breakpoint is marked with an icon in the left margin of the editor window, and the icon varies with the type of breakpoint:

![Figure 45: Breakpoint icons](image)

If the breakpoint icon does not appear, make sure the option **Show bookmarks** is selected, see Editor options in the *IDE Project Management and Building Guide for ARM*.

Just point at the breakpoint icon with the mouse pointer to get detailed tooltip information about all breakpoints set on the same location. The first row gives user breakpoint information, the following rows describe the physical breakpoints used for implementing the user breakpoint. The latter information can also be seen in the Breakpoint Usage window.

**Note:** The breakpoint icons might look different for the C-SPY driver you are using.

**BREAKPOINTS IN THE C-SPY SIMULATOR**

The C-SPY simulator supports all breakpoint types and you can set an unlimited amount of breakpoints.
BREAKPOINTS IN THE C-SPY HARDWARE DRIVERS

Using the C-SPY drivers for hardware debugger systems you can set various breakpoint types. The amount of breakpoints you can set depends on the number of hardware breakpoints available on the target system or whether you have enabled software breakpoints, in which case the number of breakpoints you can set is unlimited.

When software breakpoints are enabled, the debugger will first use any available hardware breakpoints before using software breakpoints. Exceeding the number of available hardware breakpoints, when software breakpoints are not enabled, causes the debugger to single step. This will significantly reduce the execution speed. For this reason you must be aware of the different breakpoint consumers.

For information about the characteristics of breakpoints for the different target systems, see the manufacturer’s documentation.

BREAKPOINT CONSUMERS

A debugger system includes several consumers of breakpoints.

**User breakpoints**

The breakpoints you define in the breakpoint dialog box or by toggling breakpoints in the editor window often consume one physical breakpoint each, but this can vary greatly. Some user breakpoints consume several physical breakpoints and conversely, several user breakpoints can share one physical breakpoint. User breakpoints are displayed in the same way both in the Breakpoint Usage window and in the Breakpoints window, for example Data 0[R] callCount.

**C-SPY itself**

C-SPY itself also consumes breakpoints. C-SPY will set a breakpoint if:

- The debugger option Run To has been selected, and any step command is used. These are temporary breakpoints which are only set during a debug session. This means that they are not visible in the Breakpoints window.
- The linker option Semihosted or IAR breakpoint has been selected.
  In the DLIB runtime environment, C-SPY will set a system breakpoint on the __DebugBreak label.

These types of breakpoint consumers are displayed in the Breakpoint Usage window, for example, C-SPY Terminal I/O & libsupport module.
C-SPY plugin modules

For example, modules for real-time operating systems can consume additional breakpoints. Specifically, by default, the Stack window consumes one physical breakpoint.

To disable the breakpoint used by the Stack window:

1. Choose Tools>Options>Stack.
2. Deselect the Stack pointer(s) not valid until program reaches: label option.

To disable the Stack window entirely, choose Project>Options>Debugger>Plugins and deselect the Stack plugin.

BREAKPOINTS OPTIONS

For the following hardware debugger systems it is possible to set some driver-specific breakpoint options before you start C-SPY:

- GDB Server
- I-jet/JTAGjet
- J-Link/J-Trace JTAG probes
- Macraigor JTAG probes.

For more information, see Breakpoints options, page 145.

BREAKPOINTS ON EXCEPTION VECTORS

You can set breakpoints on exception vectors for ARM9, Cortex-R4, and Cortex-M3 devices. Use the Vector Catch dialog box to set a breakpoint directly on a vector in the interrupt vector table, without using a hardware breakpoint. For more information, see Vector Catch dialog box, page 148.

For the J-Link/J-Trace driver and for RDI drivers, it is also possible to set breakpoints directly on a vector already in the options dialog box, see Setup options for J-Link/J-Trace, page 406 and RDI, page 414.

SETTING BREAKPOINTS IN __ramfunc DECLARED FUNCTIONS

To set a breakpoint in a __ramfunc declared function, the program execution must have reached the main function. The system startup code moves all __ramfunc declared functions from their stored location—normally flash memory—to their RAM location, which means the __ramfunc declared functions are not in their proper place and breakpoints cannot be set until you have executed up to the main function. Use the
**Restore software breakpoints** option to solve this problem, see *Breakpoints options*, page 145, specifically the **Restore software breakpoints** option.

In addition, breakpoints in `__ramfunc` declared functions added from the editor have to be disabled prior to invoking C-SPY and prior to exiting a debug session.

For information about the `__ramfunc` keyword, see the *IAR C/C++ Development Guide for ARM*.

---

**Setting breakpoints**

This section describes various tasks related to setting and using breakpoints.

More specifically, you will get information about:

- Various ways to set a breakpoint
- Toggling a simple code breakpoint
- Setting breakpoints using the dialog box
- Setting a data breakpoint in the Memory window
- Setting breakpoints using system macros
- Useful breakpoint hints.

**VARIOUS WAYS TO SET A BREAKPOINT**

You can set a breakpoint in various ways:

- Toggling a simple code breakpoint.
- Using the New Breakpoints dialog box and the Edit Breakpoints dialog box available from the context menus in the editor window, Breakpoints window, and in the Disassembly window. The dialog boxes give you access to all breakpoint options.
- Setting a data breakpoint on a memory area directly in the Memory window.
- Using predefined system macros for setting breakpoints, which allows automation.

The different methods offer different levels of simplicity, complexity, and automation.

**TOGGLING A SIMPLE CODE BREAKPOINT**

Toggling a code breakpoint is a quick method of setting a breakpoint. The following methods are available both in the editor window and in the Disassembly window:

- Click in the gray left-side margin of the window
- Place the insertion point in the C source statement or assembler instruction where you want the breakpoint, and click the Toggle Breakpoint button in the toolbar
Setting breakpoints

- Choose Edit>Toggle Breakpoint
- Right-click and choose Toggle Breakpoint from the context menu.

SETTING BREAKPOINTS USING THE DIALOG BOX

The advantage of using a breakpoint dialog box is that it provides you with a graphical interface where you can interactively fine-tune the characteristics of the breakpoints. You can set the options and quickly test whether the breakpoint works according to your intentions.

All breakpoints you define using a breakpoint dialog box are preserved between debug sessions.

You can open the dialog box from the context menu available in the editor window, Breakpoints window, and in the Disassembly window.

To set a new breakpoint:

1. Choose View>Breakpoints to open the Breakpoints window.
2. In the Breakpoints window, right-click, and choose New Breakpoint from the context menu.
3. On the submenu, choose the breakpoint type you want to set.
   Depending on the C-SPY driver you are using, different breakpoint types are available.
4. In the breakpoint dialog box that appears, specify the breakpoint settings and click OK.
   The breakpoint is displayed in the Breakpoints window.
To modify an existing breakpoint:

5 In the Breakpoints window, editor window, or in the Disassembly window, select the breakpoint you want to modify and right-click to open the context menu.

6 On the context menu, choose the appropriate command.

7 In the breakpoint dialog box that appears, specify the breakpoint settings and click OK. The breakpoint is displayed in the Breakpoints window.

**SETTING A DATA BREAKPOINT IN THE MEMORY WINDOW**

You can set breakpoints directly on a memory location in the Memory window. Right-click in the window and choose the breakpoint command from the context menu that appears. To set the breakpoint on a range, select a portion of the memory contents.

The breakpoint is not highlighted in the Memory window; instead, you can see, edit, and remove it using the Breakpoints window, which is available from the View menu. The breakpoints you set in the Memory window will be triggered for both read and write accesses. All breakpoints defined in this window are preserved between debug sessions.
Setting breakpoints

**Note:** Setting breakpoints directly in the Memory window is only possible if the driver you use supports this.

### SETTING BREAKPOINTS USING SYSTEM MACROS

You can set breakpoints not only in the breakpoint dialog box but also by using built-in C-SPY system macros. When you use system macros for setting breakpoints, the breakpoint characteristics are specified as macro parameters.

Macros are useful when you have already specified your breakpoints so that they fully meet your requirements. You can define your breakpoints in a macro file, using built-in system macros, and execute the file at C-SPY startup. The breakpoints will then be set automatically each time you start C-SPY. Another advantage is that the debug session will be documented, and that several engineers involved in the development project can share the macro files.

**Note:** If you use system macros for setting breakpoints, you can still view and modify them in the Breakpoints window. In contrast to using the dialog box for defining breakpoints, all breakpoints that are defined using system macros are removed when you exit the debug session.

These breakpoint macros are available:

<table>
<thead>
<tr>
<th>C-SPY macro for breakpoints</th>
<th>Simulator</th>
<th>J-Link/ JTAGjet</th>
<th>J-Trace</th>
<th>GDB Server</th>
<th>ST-Link</th>
<th>TI Stellaris</th>
<th>TI XDS100</th>
<th>Angel / IAR ROM-monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>__setCodeBreak</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>__setDataBreak</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>__setLogBreak</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>__setDataLogBreak</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>__setSimBreak</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>__setTraceStartBreak</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>__setTraceStopBreak</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>__clearBreak</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

*Table 8: C-SPY macros for breakpoints*

For information about each breakpoint macro, see *Reference information on C-SPY system macros*, page 307.
Setting breakpoints at C-SPY startup using a setup macro file

You can use a setup macro file to define breakpoints at C-SPY startup. Follow the procedure described in Registering and executing using setup macros and setup files, page 298.

SETTING A BREAKPOINT ON AN EXCEPTION VECTOR

This procedure applies to I-jet, J-Link/J-Trace and Macraigor.

To set a breakpoint on an exception vector:

1. Select the correct device. Before starting C-SPY, choose Project>Options and select the General Options category. Choose the appropriate core or device from one of the Processor variant drop-down lists available on the Target page.
2. Start C-SPY.
3. Choose C-SPY driver>Vector Catch. By default, vectors are selected according to your settings on the Breakpoints options page, see Breakpoints options, page 145.
4. In the Vector Catch dialog box, select the vector you want to set a breakpoint on, and click OK. The breakpoint will only be triggered at the beginning of the exception.

USEFUL BREAKPOINT HINTS

Below are some useful hints related to setting breakpoints.

Tracing incorrect function arguments

If a function with a pointer argument is sometimes incorrectly called with a NULL argument, you might want to debug that behavior. These methods can be useful:

- Set a breakpoint on the first line of the function with a condition that is true only when the parameter is 0. The breakpoint will then not be triggered until the problematic situation actually occurs. The advantage of this method is that no extra source code is needed. The drawback is that the execution speed might become unacceptably low.
- You can use the assert macro in your problematic function, for example:
  ```c
  int MyFunction(int * MyPtr)
  {
    assert(MyPtr != 0); /* Assert macro added to your source code. */
    /* Here comes the rest of your function. */
  }
  ```
  The execution will break whenever the condition is true. The advantage is that the execution speed is only very slightly affected, but the drawback is that you will get a
Setting breakpoints

You can perform a task when a breakpoint is triggered and then automatically continue execution.

You can use the Action text box to associate an action with the breakpoint, for instance a C-SPY macro function. When the breakpoint is triggered and the execution of your application has stopped, the macro function will be executed. In this case, the execution will not continue automatically.

Instead, you can set a condition which returns 0 (false). When the breakpoint is triggered, the condition—which can be a call to a C-SPY macro that performs a task—is evaluated and because it is not true, execution continues.

Consider this example where the C-SPY macro function performs a simple task:

```
__var my_counter;

count() {
    my_counter += 1;
    return 0;
}
```

To use this function as a condition for the breakpoint, type `count()` in the Expression text box under Conditions. The task will then be performed when the breakpoint is triggered. Because the macro function `count` returns 0, the condition is false and the execution of the program will resume automatically, without any stop.
Reference information on breakpoints

This section gives reference information about these windows and dialog boxes:

- Breakpoints window, page 131
- Breakpoint Usage window, page 133
- Code breakpoints dialog box, page 134
- JTAG Watchpoints dialog box, page 136
- Log breakpoints dialog box, page 139
- Data breakpoints dialog box, page 140
- Data Log breakpoints dialog box, page 143
- Breakpoints options, page 145
- Immediate breakpoints dialog box, page 147
- Vector Catch dialog box, page 148
- Enter Location dialog box, page 149
- Resolve Source Ambiguity dialog box, page 150.

See also:

- Reference information on C-SPY system macros, page 307
- Reference information on trace, page 194.

Breakpoints window

The Breakpoints window is available from the View menu.

![Figure 47: Breakpoints window](image)

The Breakpoints window lists all breakpoints you define.

Use this window to conveniently monitor, enable, and disable breakpoints; you can also define new breakpoints and modify existing breakpoints.
Requirements

None; this window is always available.

Display area

This area lists all breakpoints you define. For each breakpoint, information about the breakpoint type, source file, source line, and source column is provided.

Context menu

This context menu is available:

![Figure 48: Breakpoints window context menu]

These commands are available:

Go to Source

Moves the insertion point to the location of the breakpoint, if the breakpoint has a source location. Double-click a breakpoint in the Breakpoints window to perform the same command.

Edit

Opens the breakpoint dialog box for the breakpoint you selected.

Delete

Deletes the breakpoint. Press the Delete key to perform the same command.

Enable

Enables the breakpoint. The check box at the beginning of the line will be selected. You can also perform the command by manually selecting the check box. This command is only available if the breakpoint is disabled.

Disable

Disables the breakpoint. The check box at the beginning of the line will be deselected. You can also perform this command by manually deselecting the check box. This command is only available if the breakpoint is enabled.

Enable All

Enables all defined breakpoints.
Breakpoints

Disable All
Disables all defined breakpoints.

New Breakpoint
Displays a submenu where you can open the breakpoint dialog box for the available breakpoint types. All breakpoints you define using this dialog box are preserved between debug sessions.

Breakpoint Usage window

The Breakpoint Usage window is available from the menu specific to the C-SPY driver you are using.

The Breakpoint Usage window lists all breakpoints currently set in the target system, both the ones you have defined and the ones used internally by C-SPY. The format of the items in this dialog box depends on the C-SPY driver you are using.

The window gives a low-level view of all breakpoints, related but not identical to the list of breakpoints displayed in the Breakpoints window.

C-SPY uses breakpoints when stepping. If your target system has a limited number of hardware breakpoints and software breakpoints are not enabled, exceeding the number of available hardware breakpoints will cause the debugger to single step. This will significantly reduce the execution speed. Therefore, in a debugger system with a limited amount of hardware breakpoints, you can use the Breakpoint Usage dialog box for:

- Identifying all breakpoint consumers
- Checking that the number of active breakpoints is supported by the target system
- Configuring the debugger to use the available breakpoints in a better way, if possible.

Figure 49: Breakpoint Usage window
Reference information on breakpoints

Requirements

None; this window is always available.

Display area

For each breakpoint in the list, the address and access type are displayed. Each breakpoint in the list can also be expanded to show its originator.

Code breakpoints dialog box

The Code breakpoints dialog box is available from the context menu in the editor window, Breakpoints window, and in the Disassembly window.

![Code breakpoints dialog box]

Use the Code breakpoints dialog box to set a code breakpoint.

Note: The Code breakpoints dialog box depends on the C-SPY driver you are using.

Requirements

None; this dialog box is always available.

Break At

Specify the code location of the breakpoint in the text box. Alternatively, click the Edit button to open the Enter Location dialog box, see Enter Location dialog box, page 149.

Breakpoint type

Overrides the default breakpoint type. Select the Override default check box and choose between the Software and Hardware options.
You can specify the breakpoint type for these C-SPY drivers:

- The C-SPY I-jet/JTAGjet driver
- The C-SPY GDB Server driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY Macraigor JTAG driver.

**Size**

Determines whether there should be a size—in practice, a range—of locations where the breakpoint will trigger. Each fetch access to the specified memory range will trigger the breakpoint. Select how to specify the size:

**Auto**

The size will be set automatically, typically to 1.

**Manual**

Specify the size of the breakpoint range in the text box.

**Action**

Specify a valid C-SPY expression, which is evaluated when the breakpoint is triggered and the condition is true. For more information, see *Useful breakpoint hints*, page 129.

**Conditions**

Specify simple or complex conditions:

**Expression**

Specify a valid C-SPY expression, see *Expressions*, page 303.

**Condition true**

The breakpoint is triggered if the value of the expression is true.

**Condition changed**

The breakpoint is triggered if the value of the expression has changed since it was last evaluated.

**Skip count**

The number of times that the breakpoint condition must be fulfilled before the breakpoint starts triggering. After that, the breakpoint will trigger every time the condition is fulfilled.
JTAG Watchpoints dialog box

The JTAG Watchpoints dialog box is available from the driver-specific menu.

![Figure 51: JTAG Watchpoints dialog box](image)

Use this dialog box to directly control the two hardware watchpoint units. If the number of needed watchpoints (including implicit watchpoints used by the breakpoint system) exceeds two, an error message will be displayed when you click the **OK** button. This check is also performed for the C-SPY **Go** button.

To cause a trigger for accesses in the range 0x20-0xFF:

1. Set Break Condition to Range.
2. Set the address value of watchpoint 0 to 0 and the mask to 0xFF.
3. Set the address value of watchpoint 1 to 0 and the mask to 0x1F.

Requirements

One of these alternatives:

- The J-Link/J-Trace driver
- The Macraigor driver
Breakpoints

Address

Specify the address to watch for.

Value
Specify an address or a C-SPY expression that evaluates to an address. Alternatively, you can select an address you have previously watched for from the drop-down list. For detailed information about C-SPY expressions, see C-SPY expressions, page 92.

Mask
Qualifies each bit in the value. A zero bit in the mask will cause the corresponding bit in the value to be ignored in the comparison. To match any address, enter 0. Note that the mask values are inverted with respect to the notation used in the ARM hardware manuals.

Address Bus Pattern
Shows the bit pattern to be used by the address comparator. Ignored bits as specified in the mask are shown as $x$.

Access Type

Selects the access type of the data to watch for:

Any
Matches any access type.

OP Fetch
Matches an operation code (instruction) fetch.

Read
Reads from location.

Write
Writes to location.

R/W
Reads from or writes to location.

Data

Specifies the data to watch for. For size, choose between:

Any Size
Matches data accesses of any size.

Byte
Matches byte size accesses.

Halfword
Matches halfword size accesses.

Word
Matches word size accesses.
You can specify a value to watch for. Choose between:

**Value**  Specify a value or a C-SPY expression. Alternatively, you can select a value you have previously watched for from the drop-down list. For detailed information about C-SPY expressions, see *C-SPY expressions*, page 92.

**Mask**  Qualifies each bit in the value. A zero bit in the mask will cause the corresponding bit in the value to be ignored in the comparison. To match any address, enter 0. Note that the mask values are inverted with respect to the notation used in the ARM hardware manuals.

**Data Bus Pattern**  Shows the bit pattern to be used by the address comparator. Ignored bits as specified in the mask are shown as $x$.

**Extern**  Defines the state of the external input. Choose between:

- Any  Ignores the state.
- 0  Defines the state as low.
- 1  Defines the state as high.

**Mode**  Selects which CPU mode that must be active for a match. Choose between:

- User  Selects the CPU mode USER.
- Non User  Selects one of the CPU modes SYSTEM SVC, UND, ABORT, IRQ, or FIQ.
- Any  Ignores the CPU mode.

**Break Condition**  Selects how the defined watchpoints will be used. Choose between:

- Normal  Uses the two watchpoints individually (OR).
- Range  Combines both watchpoints to cover a range where watchpoint 0 defines the start of the range and watchpoint 1 the end of the range. Selectable ranges are restricted to being powers of 2.
The Log breakpoints dialog box is available from the context menu in the editor window, Breakpoints window, and in the Disassembly window.

![Log breakpoints dialog box](image)

**Log breakpoints dialog box**

Use the Log breakpoints dialog box to set a log breakpoint.

**Requirements**

None; this dialog box is always available.

**Trigger at**

Specify the code location of the breakpoint. Alternatively, click the **Edit** button to open the **Enter Location** dialog box, see **Enter Location dialog box**, page 149.

**Message**

Specify the message you want to be displayed in the C-SPY Debug Log window. The message can either be plain text, or—if you also select the option **C-SPY macro "__message" style**—a comma-separated list of arguments.

**C-SPY macro "__message" style**

Select this option to make a comma-separated list of arguments specified in the Message text box be treated exactly as the arguments to the C-SPY macro language statement `__message`, see **Formatted output**, page 304.
Conditions

Specify simple or complex conditions:

**Expression**

Specify a valid C-SPY expression, see *Expressions*, page 303.

**Condition true**

The breakpoint is triggered if the value of the expression is true.

**Condition changed**

The breakpoint is triggered if the value of the expression has changed since it was last evaluated.

Data breakpoints dialog box

The **Data** breakpoints dialog box is available from the context menu in the editor window, Breakpoints window, the Memory window, and in the Disassembly window.

![Data breakpoints dialog box](image)

*Figure 53: Data breakpoints dialog box*

Use the **Data** breakpoints dialog box to set a data breakpoint. Data breakpoints never stop execution within a single instruction. They are recorded and reported after the instruction is executed.

**Note:** The **Data** breakpoints dialog box depends on the C-SPY driver you are using.

Requirements

One of these alternatives:

- The C-SPY simulator
- The C-SPY I-jet/JTAGjet driver
Breakpoints

**Break At**
Specify the data location of the breakpoint in the text box. Alternatively, click the **Edit** button to open the **Enter Location** dialog box, see *Enter Location dialog box*, page 149.

**Access Type**
Selects the type of memory access that triggers the breakpoint:
- **Read/Write**
  - Reads from or writes to location.
- **Read**
  - Reads from location.
- **Write**
  - Writes to location.

**Size**
Determines whether there should be a size—in practice, a range—of locations where the breakpoint will trigger. Each fetch access to the specified memory range will trigger the breakpoint. Select how to specify the size:
- **Auto**
  - The size will automatically be based on the type of expression the breakpoint is set on. For example, if you set the breakpoint on a 12-byte structure, the size of the breakpoint will be 12 bytes.
- **Manual**
  - Specify the size of the breakpoint range in the text box.
  
  For data breakpoints, this can be useful if you want the breakpoint to be triggered on accesses to data structures, such as arrays, structs, and unions.

**Action**
Specify a valid C-SPY expression, which is evaluated when the breakpoint is triggered and the condition is true. For more information, see *Useful breakpoint hints*, page 129.

**Conditions**
Specify simple or complex conditions:
- **Expression**
  - Specify a valid C-SPY expression, see *Expressions*, page 303.
- **Condition true**
  - The breakpoint is triggered if the value of the expression is true.
Condition changed
The breakpoint is triggered if the value of the expression has changed since it was last evaluated.

Skip count
The number of times that the breakpoint condition must be fulfilled before the breakpoint starts triggering. After that, the breakpoint will trigger every time the condition is fulfilled.

Trigger range
Shows the requested range and the effective range to be covered by the trace. The range suggested is either within or exactly the area specified by the Break At and the Size options.

Extend to cover requested range
Extends the breakpoint so that a whole data structure is covered. For data structures that do not fit the size of the possible breakpoint ranges supplied by the hardware breakpoint unit, for example three bytes, the breakpoint range will not cover the whole data structure. Note that the breakpoint range will be extended beyond the size of the data structure, which might cause false triggers at adjacent data.

Match data
Enables matching of the accessed data. Use the Match data options in combination with the access types for data. This option can be useful when you want a trigger when a variable has a certain value.

Value
Specify a data value.

Mask
Specify which part of the value to match (word, halfword, or byte).

The Match data options are only available for I-jet, JTAGjet, J-Link/J-Trace and ST-LINK, and when using an ARM7/9 or a Cortex-M device.

Note: For Cortex-M devices, only one breakpoint with Match data can be set. Such a breakpoint uses two hardware breakpoints.
Data Log breakpoints dialog box

The Data Log breakpoints dialog box is available from the context menu in the editor window, Breakpoints window, the Memory window, and in the Disassembly window.

![Data Log breakpoints dialog box](image)

Use the Data Log breakpoints dialog box to set a maximum of four data log breakpoints.

To get started using data logging, see *Getting started using data logging*, page 97

Requirements

One of these alternatives:

- The C-SPY simulator
- The C-SPY J-Link/J-Trace driver, a J-Link debug probe and Cortex-M with SWO
- The C-SPY I-jet/JTAGjet driver and an I-jet in-circuit debugging probe or a JTAG-jet debug probe and Cortex-M with SWO

Trigger at

Specify the data location of the breakpoint. Alternatively, click the *Edit* button to open the *Enter Location* dialog box, see *Enter Location dialog box*, page 149.

Access Type

Selects the type of memory access that triggers the breakpoint:

Read/Write

Reads from or writes to location.
**Read**

Reads from location; except for Cortex-M3, revision 1 devices.

**Write**

Writes to location; except for Cortex-M3, revision 1 devices.

**Size**

Determines whether there should be a size—in practice, a range—of locations where the breakpoint will trigger. Each fetch access to the specified memory range will trigger the breakpoint. Select how to specify the size:

**Auto**

The size will automatically be based on the type of expression the breakpoint is set on. For example, if you set the breakpoint on a 12-byte structure, the size of the breakpoint will be 12 bytes.

**Manual**

Specify the size of the breakpoint range in the text box.

**Trigger range**

Shows the requested range and the effective range to be covered by the trace. The range suggested is either within or exactly the area specified by the Trigger at and the Size options.

**Extend to cover requested range**

Extends the breakpoint so that a whole data structure is covered. For data structures that do not fit the size of the possible breakpoint ranges supplied by the hardware breakpoint unit, for example three bytes, the breakpoint range will not cover the whole data structure. Note that the breakpoint range will be extended beyond the size of the data structure, which might cause false triggers at adjacent data.
Breakpoints options

The Breakpoints option page is available in the Options dialog box. Choose Project>Options, select the category specific to the debugger system you are using, and click the Breakpoints tab.

Requirements

One of these alternatives:

- The C-SPY GDB Server driver
- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY Macraigor driver

Default breakpoint type

Selects the type of breakpoint resource to be used when setting a breakpoint. Choose between:

**Auto**

Uses a software breakpoint. If this is not possible, a hardware breakpoint will be used. The debugger will use read/write sequences to test for RAM; in that case, a software breakpoint will be used. The Auto option works for most applications. However, there are cases when the performed read/write sequence will make the flash memory malfunction. In that case, use the Hardware option.
Hardware

Uses hardware breakpoints. If it is not possible, no breakpoint will be set.

Software

Uses software breakpoints. If it is not possible, no breakpoint will be set.

**Restore software breakpoints at**

Automatically restores any breakpoints that were destroyed during system startup.

This can be useful if you have an application that is copied to RAM during startup and is then executing in RAM. This can, for example, be the case if you use the *initialize by copy* linker directive for code in the linker configuration file or if you have any __ramfunc declared functions in your application.

In this case, all breakpoints will be destroyed during the RAM copying when the C-SPY debugger starts. By using the *Restore software breakpoints at* option, C-SPY will restore the destroyed breakpoints.

Use the text field to specify the location in your application at which point you want C-SPY to restore the breakpoints. The default location is the label _call_main.

**Catch exceptions**

Sets a breakpoint directly on a vector in the interrupt vector table, without using a hardware breakpoint. This option is available for ARM9, Cortex-R4, and Cortex-M3 devices. The settings you make will work as default settings for the project. However, you can override these default settings during the debug session by using the Vector Catch dialog box, see *Breakpoints on exception vectors*, page 124.

The settings you make will be preserved during debug sessions.

This option is supported by the C-SPY J-Link/J-Trace driver only.
Immediate breakpoints dialog box

The Immediate breakpoints dialog box is available from the context menu in the editor window, Breakpoints window, the Memory window, and in the Disassembly window.

![Immediate breakpoints dialog box](image)

Figure 56: Immediate breakpoints dialog box

In the C-SPY simulator, use the Immediate breakpoints dialog box to set an immediate breakpoint. Immediate breakpoints do not stop execution at all; they only suspend it temporarily.

Requirements

The C-SPY simulator.

Trigger at

Specify the data location of the breakpoint. Alternatively, click the Edit button to open the Enter Location dialog box, see Enter Location dialog box, page 149.

Access Type

Selects the type of memory access that triggers the breakpoint:

- **Read**: Reads from location.
- **Write**: Writes to location.

Action

Specify a valid C-SPY expression, which is evaluated when the breakpoint is triggered and the condition is true. For more information, see Useful breakpoint hints, page 129.
Vector Catch dialog box

The Vector Catch dialog box is available from the C-SPY driver menu.

Use this dialog box to set a breakpoint directly on a vector in the interrupt vector table, without using a hardware breakpoint. You can set breakpoints on vectors for ARM9, Cortex-R4, and Cortex-M3 devices. Note that the settings you make here will not be preserved between debug sessions.

Note: For the J-Link/J-Trace driver and for RDI drivers, it is also possible to set breakpoints directly on a vector already in the options dialog box, see Setup options for J-Link/J-Trace, page 406 and RDI, page 414.

Requirements

One of these alternatives:

- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY Macraigor driver
Enter Location dialog box

The Enter Location dialog box is available from the breakpoints dialog box, either when you set a new breakpoint or when you edit a breakpoint.

Use the Enter Location dialog box to specify the location of the breakpoint.

Note: This dialog box looks different depending on the Type you select.

Type

Selects the type of location to be used for the breakpoint, choose between:

Expression

A C-SPY expression, whose value evaluates to a valid code or data location.

A code location, for example the function main, is typically used for code breakpoints.

A data location is the name of a variable and is typically used for data breakpoints. For example, my_var refers to the location of the variable my_var, and arr[3] refers to the location of the fourth element of the array arr. For static variables declared with the same name in several functions, use the syntax my_func::my_static_variable to refer to a specific variable.

For more information about C-SPY expressions, see Expressions, page 303.

Absolute address

An absolute location on the form zone:hexaddress or simply hexaddress (for example Memory:0x42). zone refers to C-SPY memory zones and specifies in which memory the address belongs, see C-SPY memory zones, page 155.

Source location

A location in your C source code using the syntax:

{filename}.row.column.

filename specifies the filename and full path.
Reference information on breakpoints

Row specifies the row in which you want the breakpoint.

Column specifies the column in which you want the breakpoint.

For example, \{(C:\src\prog.c).22.3\} sets a breakpoint on the third character position on row 22 in the source file \prog.c. Note that in quoted form, for example in a C-SPY macro, you must instead write \{(C:\\src\\prog.c).22.3\}.

Note that the Source location type is usually meaningful only for code locations in code breakpoints.

### Resolve Source Ambiguity dialog box

The Resolve Source Ambiguity dialog box appears, for example, when you try to set a breakpoint on templates and the source location corresponds to more than one function.

![Resolve Source Ambiguity dialog box](image)

To resolve a source ambiguity, perform one of these actions:

- In the text box, select one or several of the listed locations and click **Selected**.
- Click **All**.

All

The breakpoint will be set on all listed locations.
Selected

The breakpoint will be set on the source locations that you have selected in the text box.

Cancel

No location will be used.

Automatically choose all

Determines that whenever a specified source location corresponds to more than one function, all locations will be used.

Note that this option can also be specified in the IDE Options dialog box, see Debugger options in the IDE Project Management and Building Guide for ARM.
Memory and registers

This chapter describes how to use the features available in C-SPY® for examining memory and registers. More specifically, this means information about:

- Introduction to monitoring memory and registers
- Monitoring memory and registers
- Reference information on memory and registers.

Introduction to monitoring memory and registers

This section covers these topics:

- Briefly about monitoring memory and registers
- C-SPY memory zones
- Stack display
- Memory access checking.

BRIEFLY ABOUT MONITORING MEMORY AND REGISTERS

C-SPY provides many windows for monitoring memory and registers, each of them available from the View menu:

- The Memory window
  Gives an up-to-date display of a specified area of memory—a memory zone—and allows you to edit it. Different colors are used for indicating data coverage along with execution of your application. You can fill specified areas with specific values and you can set breakpoints directly on a memory location or range. You can open several instances of this window, to monitor different memory areas. The content of the window can be regularly updated while your application is executing.

- The Symbolic memory window
  Displays how variables with static storage duration are laid out in memory. This can be useful for better understanding memory usage or for investigating problems caused by variables being overwritten, for example by buffer overruns.
Introduction to monitoring memory and registers

- **The Stack window**
  Displays the contents of the stack, including how stack variables are laid out in memory. In addition, some integrity checks of the stack can be performed to detect and warn about problems with stack overflow. For example, the Stack window is useful for determining the optimal size of the stack. You can open up to two instances of this window, each showing different stacks or different display modes of the same stack.

- **The Register window**
  Gives an up-to-date display of the contents of the processor registers and SFRs, and allows you to edit them. Except for the hardwired group of CPU registers, additional registers are defined in the device description file. These registers are the device-specific memory-mapped control and status registers for the peripheral units on the ARM devices. Due to the large amount of registers—memory-mapped peripheral unit registers and CPU registers—it is inconvenient to show all registers concurrently in the Register window. Instead you can divide registers into register groups. You can choose to load either predefined register groups or define your own application-specific groups. You can open several instances of this window, each showing a different register group.

- **The SFR Setup window**
  Displays the currently defined SFRs that C-SPY has information about. If required, you can use this window to customize aspects of the SFRs.

To view the memory contents for a specific variable, simply drag the variable to the Memory window or the Symbolic memory window. The memory area where the variable is located will appear.

Reading the value of some registers might influence the runtime behavior of your application. For example, reading the value of a UART status register might reset a pending bit, which leads to the lack of an interrupt that would have processed a received byte. To prevent this from happening, make sure that the Register window containing any such registers is closed when debugging a running application.
C-SPY MEMORY ZONES

In C-SPY, the term *zone* is used for a named memory area. A memory address, or *location*, is a combination of a zone and a numerical offset into that zone. The ARM architecture has only one zone, *Memory*, which covers the whole ARM memory range.

![Figure 60: Zones in C-SPY](image)

Default zone Memory

Memory zones are used in several contexts, most importantly in the Memory and Disassembly windows, and in C-SPY macros. In the windows, use the *Zone* box to choose which memory zone to display.

These zones are available depending on the device description file you are using: Memory, Memory8, Memory16, Memory32, and Memory64.

For normal memory, the default zone Memory can be used, but certain I/O registers might require to be accessed as 8, 16, 32, or 64 bits to give correct results. By using different memory zones, you can control the access width used for reading and writing in, for example, the Memory window.

STACK DISPLAY

The Stack window displays the contents of the stack, overflow warnings, and it has a graphical stack bar. These can be useful in many contexts. Some examples are:

- Investigating the stack usage when assembler modules are called from C modules and vice versa
- Investigating whether the correct elements are located on the stack
- Investigating whether the stack is restored properly
- Determining the optimal stack size
- Detecting stack overflows.

For cores with multiple stacks, you can select which stack to view.
Monitoring memory and registers

Stack usage

When your application is first loaded, and upon each reset, the memory for the stack area is filled with the dedicated byte value \(0xCD\) before the application starts executing. Whenever execution stops, the stack memory is searched from the end of the stack until a byte with a value different from \(0xCD\) is found, which is assumed to be how far the stack has been used. Although this is a reasonably reliable way to track stack usage, there is no guarantee that a stack overflow is detected. For example, a stack can incorrectly grow outside its bounds, and even modify memory outside the stack area, without actually modifying any of the bytes near the stack range. Likewise, your application might modify memory within the stack area by mistake.

The Stack window cannot detect a stack overflow when it happens, but can only detect the signs it leaves behind. However, when the graphical stack bar is enabled, the functionality needed to detect and warn about stack overflows is also enabled.

Note: The size and location of the stack is retrieved from the definition of the section holding the stack, made in the linker configuration file. If you, for some reason, modify the stack initialization made in the system startup code, `cstartup`, you should also change the section definition in the linker configuration file accordingly; otherwise the Stack window cannot track the stack usage. For more information about this, see the IAR C/C++ Development Guide for ARM.

MEMORY ACCESS CHECKING

The C-SPY simulator can simulate various memory access types of the target hardware and detect illegal accesses, for example a read access to write-only memory. If a memory access occurs that does not agree with the access type specified for the specific memory area, C-SPY will regard this as an illegal access. Also, a memory access to memory which is not defined is regarded as an illegal access. The purpose of memory access checking is to help you to identify any memory access violations.

The memory areas can either be the zones predefined in the device description file, or memory areas based on the section information available in the debug file. In addition to these, you can define your own memory areas. The access type can be read and write, read-only, or write-only. You cannot map two different access types to the same memory area. You can check for access type violation and accesses to unspecified ranges. Any violations are logged in the Debug Log window. You can also choose to have the execution halted.

Monitoring memory and registers

This section describes various tasks related to monitoring memory and registers.

- Defining application-specific register groups, page 157.
DEFINING APPLICATION-SPECIFIC REGISTER GROUPS

Defining application-specific register groups minimizes the amount of registers displayed in the Register window and speeds up the debugging.

1. Choose Tools>Options>Register Filter.

![Figure 61: Register Filter options](image)

For information about the register filter options, see the IDE Project Management and Building Guide for ARM.

2. Select Use register filter and specify the filename and destination of the filter file for your new group in the dialog box that appears.

3. Click New Group and specify the name of your group, for example My Timer Group.

![Figure 62: Register Filter options](image)

4. In the register tree view on the Register Filter page, select a register and click the arrow button to add it to your group. Repeat this process for all registers that you want to add to your group.

5. Optionally, select any registers for which you want to change the integer base, and choose a suitable base.
6 When you are done, click OK. Your new group is now available in the Register window.

If you want to add more groups to your filter file, repeat this procedure for each group you want to add.

**Note:** The registers that appear in the list of registers are retrieved from the ddf file that is currently used. If a certain SFR that you need does not appear, you can register your own SFRs. For more information, see *SFR Setup window*, page 174.

---

**Reference information on memory and registers**

This section gives reference information about these windows and dialog boxes:

- Memory window, page 159
- Memory Save dialog box, page 163
- Memory Restore dialog box, page 164
- Fill dialog box, page 164
- Symbolic Memory window, page 166
- Stack window, page 168
- Register window, page 172
- SFR Setup window, page 174
- Edit SFR dialog box, page 177
- Memory Configuration dialog box, page 179
- Edit Memory Range dialog box, page 182
- Memory Access Setup dialog box, page 184
- Edit Memory Access dialog box, page 186.
Memory window

The Memory window is available from the View menu.

To view the memory corresponding to a variable, you can select it in the editor window and drag it to the Memory window.

Requirements

None; this window is always available.

Toolbar

The toolbar contains:

Go to

The memory location or symbol you want to view.

Zone

Selects a memory zone, see C-SPY memory zones, page 155.
Reference information on memory and registers

Context menu button
Displays the context menu.

Update Now
Updates the content of the Memory window while your application is executing. This button is only enabled if the C-SPY driver you are using has access to the target system memory while your application is executing.

Live Update
Updates the contents of the Memory window regularly while your application is executing. This button is only enabled if the C-SPY driver you are using has access to the target system memory while your application is executing. To set the update frequency, specify an appropriate frequency in the IDE Options>Debugger dialog box.

Display area
The display area shows the addresses currently being viewed, the memory contents in the format you have chosen, and—provided that the display mode is set to 1x Units—the memory contents in ASCII format. You can edit the contents of the display area, both in the hexadecimal part and the ASCII part of the area.

Data coverage is displayed with these colors:

- **Yellow**: Indicates data that has been read.
- **Blue**: Indicates data that has been written.
- **Green**: Indicates data that has been both read and written.

**Note**: Data coverage is not supported by all C-SPY drivers. Data coverage is supported by the C-SPY Simulator.
Context menu

This context menu is available:

- **Copy**
- **Paste**
- **Zone**
- **1x Units**
- **2x Units**
- **4x Units**
- **8x Units**
- **Little Endian**
- **Big Endian**
- **Data Coverage**
- **Find...**
- **Replace...**
- **Memory Fill...**
- **Memory Save...**
- **Memory Restore...**
- **Set Data Breakpoint**

*Figure 64: Memory window context menu*

These commands are available:

**Copy, Paste**

Standard editing commands.

**Zone**

Selects a memory zone, see *C-SPY memory zones*, page 155.

**1x Units**

Displays the memory contents as single bytes.

**2x Units**

Displays the memory contents as 2-byte groups.

**4x Units**

Displays the memory contents as 4-byte groups.

**8x Units**

Displays the memory contents as 8-byte groups.

**Little Endian**

Displays the contents in little-endian byte order.
Big Endian
Displays the contents in big-endian byte order.

Data Coverage
Choose between:
- **Enable** toggles data coverage on or off.
- **Show** toggles between showing or hiding data coverage.
- **Clear** clears all data coverage information.
These commands are only available if your C-SPY driver supports data coverage.

Find
Displays a dialog box where you can search for text within the Memory window; read about the **Find** dialog box in the *IDE Project Management and Building Guide for ARM*.

Replace
Displays a dialog box where you can search for a specified string and replace each occurrence with another string; read about the **Replace** dialog box in the *IDE Project Management and Building Guide for ARM*.

Memory Fill
Displays a dialog box, where you can fill a specified area with a value, see **Fill dialog box**, page 164.

Memory Save
Displays a dialog box, where you can save the contents of a specified memory area to a file, see **Memory Save dialog box**, page 163.

Memory Restore
Displays a dialog box, where you can load the contents of a file in Intel-hex or Motorola s-record format to a specified memory zone, see **Memory Restore dialog box**, page 164.

Set Data Breakpoint
Sets breakpoints directly in the Memory window. The breakpoint is not highlighted; you can see, edit, and remove it in the Breakpoints dialog box. The breakpoints you set in this window will be triggered for both read and write access. For more information, see **Setting a data breakpoint in the Memory window**, page 127.
Memory Save dialog box

The Memory Save dialog box is available by choosing Debug>Memory>Save or from the context menu in the Memory window.

![Memory Save dialog box](image)

**Figure 65: Memory Save dialog box**

Use this dialog box to save the contents of a specified memory area to a file.

**Requirements**

None; this dialog box is always available.

**Zone**

Selects a memory zone, see *C-SPY memory zones*, page 155.

**Start address**

Specify the start address of the memory range to be saved.

**End address**

Specify the end address of the memory range to be saved.

**File format**

Selects the file format to be used, which is Intel-extended by default.

**Filename**

Specify the destination file to be used; a browse button is available for your convenience.

**Save**

Saves the selected range of the memory zone to the specified file.
Memory Restore dialog box

The Memory Restore dialog box is available by choosing Debug>Memory>Restore or from the context menu in the Memory window.

Figure 66: Memory Restore dialog box

Use this dialog box to load the contents of a file in Intel-extended or Motorola S-record format to a specified memory zone.

Requirements

None; this dialog box is always available.

Zone

Selects a memory zone, see C-SPY memory zones, page 155.

Filename

Specify the file to be read; a browse button is available for your convenience.

Restore

Loads the contents of the specified file to the selected memory zone.

Fill dialog box

The Fill dialog box is available from the context menu in the Memory window.

Figure 67: Fill dialog box

Use this dialog box to fill a specified area of memory with a value.
Requirements
None; this dialog box is always available.

Start address
Type the start address—in binary, octal, decimal, or hexadecimal notation.

Length
Type the length—in binary, octal, decimal, or hexadecimal notation.

Zone
Selects a memory zone, see C-SPY memory zones, page 155.

Value
Type the 8-bit value to be used for filling each memory location.

Operation
These are the available memory fill operations:

Copy
Value will be copied to the specified memory area.

AND
An AND operation will be performed between Value and the existing contents of memory before writing the result to memory.

XOR
An XOR operation will be performed between Value and the existing contents of memory before writing the result to memory.

OR
An OR operation will be performed between Value and the existing contents of memory before writing the result to memory.
Symbolic Memory window

The Symbolic Memory window is available from the View menu during a debug session.

This window displays how variables with static storage duration, typically variables with file scope but also static variables in functions and classes, are laid out in memory. This can be useful for better understanding memory usage or for investigating problems caused by variables being overwritten, for example buffer overruns. Other areas of use are spotting alignment holes or for understanding problems caused by buffers being overwritten.

To view the memory corresponding to a variable, you can select it in the editor window and drag it to the Symbolic Memory window.

Requirements

None; this window is always available.

Toolbar

The toolbar contains:

Go to
The memory location or symbol you want to view.

Zone
Selects a memory zone, see C-SPY memory zones, page 155.

Previous
Highlights the previous symbol in the display area.

Next
Highlights the next symbol in the display area.
Display area

This area contains these columns:

Location
The memory address.

Data
The memory contents in hexadecimal format. The data is grouped according to the size of the symbol. This column is editable.

Variable
The variable name; requires that the variable has a fixed memory location. Local variables are not displayed.

Value
The value of the variable. This column is editable.

Type
The type of the variable.

There are several different ways to navigate within the memory space:

- Text that is dropped in the window is interpreted as symbols
- The scroll bar at the right-side of the window
- The toolbar buttons Next and Previous
- The toolbar list box Go to can be used for locating specific locations or symbols.

Note: Rows are marked in red when the corresponding value has changed.

Context menu

This context menu is available:

Next Symbol
Highlights the next symbol in the display area.

Previous Symbol
Highlights the previous symbol in the display area.
Reference information on memory and registers

1x Units
Displays the memory contents as single bytes. This applies only to rows which do not contain a variable.

2x Units
Displays the memory contents as 2-byte groups.

4x Units
Displays the memory contents as 4-byte groups.

Add to Watch Window
Adds the selected symbol to the Watch window.

Stack window

The Stack window is available from the View menu.

This window is a memory window that displays the contents of the stack. In addition, some integrity checks of the stack can be performed to detect and warn about problems with stack overflow. For example, the Stack window is useful for determining the optimal size of the stack.

This window retrieves information about the stack size and placement from the definition of the sections holding the stacks made in the linker configuration file. The sections are described in the IAR C/C++ Development Guide for ARM.

For applications that set up the stacks using other mechanisms, it is possible to override the default mechanism. Use one of the C-SPY command line option variants, see --proc_stack_stack, page 383.
To view the graphical stack bar:
1. Choose Tools>Options>Stack.
2. Select the option **Enable graphical stack display and stack usage**.

You can open up to two Stack windows, each showing a different stack—if several stacks are available—or the same stack with different display settings.

**Note:** By default, this window uses one physical breakpoint. For more information, see *Breakpoint consumers*, page 123.

For information about options specific to the Stack window, see the *IDE Project Management and Building Guide for ARM*.

**To view the graphical stack bar:**
1. Choose Tools>Options>Stack.
2. Select the option **Enable graphical stack display and stack usage**.

You can open up to two Stack windows, each showing a different stack—if several stacks are available—or the same stack with different display settings.

**Note:** By default, this window uses one physical breakpoint. For more information, see *Breakpoint consumers*, page 123.

For information about options specific to the Stack window, see the *IDE Project Management and Building Guide for ARM*.

**Requirements**

None; this window is always available.

**Toolbar**

The toolbar contains:

- **Stack**
  
  Selects which stack to view. This applies to cores with multiple stacks.

**The graphical stack bar**

Displays the state of the stack graphically.

The left end of the stack bar represents the bottom of the stack, in other words, the position of the stack pointer when the stack is empty. The right end represents the end of the memory space reserved for the stack. The graphical stack bar turns red when the stack usage exceeds a threshold that you can specify.

When the stack bar is enabled, the functionality needed to detect and warn about stack overflows is also enabled.
Place the mouse pointer over the stack bar to get tooltip information about stack usage.

**Display area**

This area contains these columns:

**Location**
Displays the location in memory. The addresses are displayed in increasing order. If your target system has a stack that grows toward high addresses, the top of the stack will consequently be located at the bottom of the window. The address referenced by the stack pointer, in other words the top of the stack, is highlighted in a green color.

**Data**
Displays the contents of the memory unit at the given location. From the Stack window context menu, you can select how the data should be displayed; as a 1-, 2-, or 4-byte group of data.

**Variable**
Displays the name of a variable, if there is a local variable at the given location. Variables are only displayed if they are declared locally in a function, and located on the stack and not in registers.

**Value**
Displays the value of the variable that is displayed in the Variable column.

**Frame**
Displays the name of the function that the call frame corresponds to.
Context menu

This context menu is available:

- **Show Variables**
- **Show Offsets**
- **1x Units**
- **2x Units**
- **4x Units**
- **Default Format**
- **Binary Format**
- **Octal Format**
- **Decimal Format**
- **Hexadecimal Format**
- **Char Format**

![Figure 71: Stack window context menu](image)

These commands are available:

**Show variables**

Displays separate columns named Variables, Value, and Frame in the Stack window. Variables located at memory addresses listed in the Stack window are displayed in these columns.

**Show offsets**

Displays locations in the Location column as offsets from the stack pointer. When deselected, locations are displayed as absolute addresses.

**1x Units**

Displays the memory contents as single bytes.

**2x Units**

Displays the memory contents as 2-byte groups.

**4x Units**

Displays the memory contents as 4-byte groups.
Default Format,
Binary Format,
Octal Format,
Decimal Format,
Hexadecimal Format,
Char Format

Changes the display format of expressions. The display format setting affects different types of expressions in different ways. Your selection of display format is saved between debug sessions. These commands are available if a selected line in the window contains a variable.

The display format setting affects different types of expressions in these ways:

- **Variables**: The display setting affects only the selected variable, not other variables.
- **Array elements**: The display setting affects the complete array, that is, the same display format is used for each array element.
- **Structure fields**: All elements with the same definition—the same field name and C declaration type—are affected by the display setting.

**Options**
Opens the IDE Options dialog box where you can set options specific to the Stack window, see the IDE Project Management and Building Guide for ARM.

**Register window**

The Register window is available from the View menu.

![Register window](image)

*Figure 72: Register window*
This window gives an up-to-date display of the contents of the processor registers and special function registers, and allows you to edit their contents. Optionally, you can choose to load either predefined register groups or to define your own application-specific groups.

You can open several instances of this window, which is very convenient if you want to keep track of different register groups.

To enable predefined register groups:

1. Select a device description file that suits your device, see Selecting a device description file, page 53.
2. The register groups appear in the Register window, provided that they are defined in the device description file. Note that the available register groups are also listed on the Register Filter page.

To define application-specific register groups:

See Defining application-specific register groups, page 157.

Requirements

None; this window is always available.

Toolbar

The toolbar contains:

CPU Registers

Selects which register group to display, by default CPU Registers. By default, there are two register groups in the debugger: If some of your SFRs are missing, you can register your own SFRs in a Custom group, see SFR Setup window, page 174.

Current CPU Registers contains the registers that are available in the current processor mode.

CPU Registers contains both the current registers and their banked counterparts available in other processor modes.

Additional register groups are predefined in the device description files—available in the arm\config directory—that make all SFR registers available in the register window. The device description file contains a section that defines the special function registers and their groups.
Display area

Displays registers and their values. Every time C-SPY stops, a value that has changed since the last stop is highlighted. To edit the contents of a register, click it, and modify the value.

Some registers are expandable, which means that the register contains interesting bits or subgroups of bits.

To change the display format, change the Base setting on the Register Filter page—available by choosing Tools>Options.

These additional support registers are available in the CPU Registers group (might not be available in the C-SPY driver you are using):

CYCLECOUNTER - Cleared when an application is started or reset and is incremented with the number of used cycles during instruction simulation.

CCSTEP - Shows the number of used cycles during the last performed C/C++ source or assembler step.

CCTIMER 1 and CCTIMER2 - Two trip counts that can be cleared manually at any given time. They are incremented with the number of used cycles during instruction simulation.

SFR Setup window

The SFR Setup window is available from the context menu from the Project menu.

![Figure 73: SFR Setup window](image)

This window displays the currently defined SFRs that C-SPY has information about. You can choose to display only factory-defined or custom-defined SFRs, or both. If required, you can use this window to customize the aspects of the SFRs. For factory-defined SFRs (that is, retrieved from the .ddf file that is currently used), you can only customize the access type.
Any custom-defined SFRs are added to a dedicated register group called Custom, which you can choose to display in the Register window. Your custom-defined SFRs are saved in `projectCustomSFR.sfr`.

You can only add or modify SFRs when the C-SPY debugger is not running.

**Requirements**

None; this window is always available.

**Display area**

This area contains these columns:

**Status**

A character that signals the status of the SFR, which can be one of:
- blank, a factory-defined SFR.
- C, a factory-defined SFR that has been modified.
- +, a custom-defined SFR.
- ?, an SFR that is ignored for some reason. An SFR can be ignored when a factory-defined SFR has been modified, but the SFR is no longer available, or it is located somewhere else or with a different size. Typically, this might happen if you change to another device.

**Name**

A unique name of the SFR.

**Address**

The memory address of the SFR.

**Zone**

Selects a memory zone, see C-SPY memory zones, page 155.

**Size**

The size of the register, which can be any of 8, 16, 32, or 64.

**Access**

The access type of the register, which can be one of Read/Write, Read only, Write only, or None.

You can click a name or an address to change the value. The hexadecimal 0x prefix for the address can be omitted, the value you enter will still be interpreted as hexadecimal. For example, if you enter 4567, you will get 0x4567.

You can click a column header to sort the SFRs according to the column property.
Color coding used in the display area:
- Green, which indicates that the corresponding value has changed
- Red, which indicates an ignored SFR.

**Context menu**

This context menu is available:

<table>
<thead>
<tr>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show All</td>
</tr>
<tr>
<td>Show Custom SFRs only</td>
</tr>
<tr>
<td>Show Factory SFRs only</td>
</tr>
<tr>
<td>Add...</td>
</tr>
<tr>
<td>Edit</td>
</tr>
<tr>
<td>Delete</td>
</tr>
<tr>
<td>Delete/Revert All Custom SFRs</td>
</tr>
<tr>
<td>Save Custom SFRs...</td>
</tr>
</tbody>
</table>

- 8 bits
- 16 bits
- 32 bits
- 64 bits
- Read/Write
- Read only
- Write only
- None

*Figure 74: SFR Setup window context menu*

These commands are available:

**Show All**

Shows all SFR.

**Show Custom SFRs only**

Shows all custom-defined SFRs.

**Show Factory SFRs only**

Shows all factory-defined SFRs retrieved from the ddf file.

**Add**

Displays the Edit SFR dialog box where you can add a new SFR, see *Edit SFR dialog box*, page 177.

**Edit**

Displays the Edit SFR dialog box where you can edit an SFR, see *Edit SFR dialog box*, page 177.
Delete

Deletes an SFR. This command only works on custom-defined SFRs.

Delete/revert All Custom SFRs

Deletes all custom-defined SFRs and reverts all modified factory-defined SFRs to their factory settings.

Save Custom SFRs

Opens a standard save dialog box to save all custom-defined SFRs.

8|16|32|64 bits

Selects display format for the selected SFR, which can be 8, 16, 32, or 64 bits. Note that the display format can only be changed for custom-defined SFRs.

Read/Write|Read only|Write only|None

Selects the access type of the selected SFR, which can be Read/Write, Read only, Write only, or None. Note that for factory-defined SFRs, the default access type is indicated.

Edit SFR dialog box

The Edit SFR dialog box is available from the SFR Setup window.

![Edit SFR dialog box](image)

Figure 75: Edit SFR dialog box

Use this dialog box to define the SFRs.

Requirements

None; this window is always available.
Reference information on memory and registers

Name
Specify the name of the SFR that you want to add or edit.

Address
Specify the address of the SFR that you want to add or edit. The hexadecimal 0x prefix for the address can be omitted, the value you enter will still be interpreted as hexadecimal. For example, if you enter 4567, you will get 0x4567.

Zone
Selects the memory zone for the SFR you want to add or edit. The list of zones is retrieved from the ddf file that is currently used.

Size
Selects the size of the SFR. Choose between 8, 16, 32, or 64 bits. Note that the display format can only be changed for custom-defined SFRs.

Access
Selects the access type of the SFR. Choose between Read/Write, Read only, Write only, or None. Note that for factory-defined SFRs, the default access type is indicated.
Memory Configuration dialog box

The Memory Configuration dialog box is available from the C-SPY driver menu.

C-SPY uses a default memory configuration based on information retrieved from the device description file that you select, or if memory configuration is missing in the device description file, tries to provide a usable factory default. See Selecting a device description file, page 53.

Use this dialog box to verify, and if needed, modify the memory areas so that they match the memory available on your device. C-SPY needs this information to handle memory as efficiently as possible.
This dialog box is automatically displayed the first time you start the C-SPY driver for a given project, unless the device description file contains a memory description which is already specified as correct and complete. Subsequent starts will not display the dialog box unless you have made project changes that might cause the memory configuration to change, for example if you have selected another device description file.

You can only change the memory configuration when C-SPY is not running.

**Requirements**

The C-SPY I-jet/JTAGjet driver.

**Factory ranges**

Identifies which device description file that is currently selected and lists the default memory areas retrieved from the file in these columns:

- **Zone**
  - Selects a memory zone, see *C-SPY memory zones*, page 155.
- **Name**
  - The name of the memory area.
- **Start**
  - The start address for the memory area, in hexadecimal notation.
- **End**
  - The end address for the memory area, in hexadecimal notation.
- **Type**
  - The access type of the memory area.
- **Size**
  - The size of the memory area.

**Used ranges**

These columns lists the memory areas that you have specified manually:

- **Zone**
  - Selects a memory zone, see *C-SPY memory zones*, page 155.
- **Start**
  - The start address for the memory area, in hexadecimal notation.
- **End**
  - The end address for the memory area, in hexadecimal notation.
Memory and registers

Cache Type
The cache type of the memory area.

Size
The size of the memory area.

Comment
Memory area information.

Graphical bar
A graphical bar that visualizes the whole theoretical memory space for the device. Defined areas are highlighted in green.

Buttons
These buttons are available:

New
Opens the Edit Memory Range dialog box, where you can specify a new memory area and attach a cache type to it, see Edit Memory Access dialog box, page 186.

Edit
Opens the Edit Memory Range dialog box, where you can edit the selected memory area. See Edit Memory Access dialog box, page 186.

Remove
Removes the selected memory area definition.

Use Default
Retrieves the memory areas as specified in the selected device description file, or if memory information is missing in the device description file, tries to provide a usable factory default.
Edit Memory Range dialog box

The Edit Memory Range dialog box is available from the Memory Configuration dialog box.

![Edit Memory Range dialog box](image)

Use this dialog box to specify the memory areas, and assign a cache type to each memory range.

Requirements

The C-SPY I-jet/JTAGjet driver.

Memory range

Defines the memory area specific to your device:

Zone

Selects a memory zone, see C-SPY memory zones, page 155.

Start address

Specify the start address for the memory area, in hexadecimal notation.

End address

Specify the end address for the memory area, in hexadecimal notation.
Memory and registers

Cache type

Selects a cache type to the memory area; choose between:

**RAM**

When the target CPU is not executing, all read accesses from memory are loaded into the cache. For example, if two Memory windows show the same part of memory, the actual memory is only read once from the hardware to update both windows. If you modify memory from a C-SPY window, your data is written to cache only. Before any target execution, even stepping a single machine instruction, the RAM cache is flushed so that all modified bytes are written to the memory on your hardware.

**ROM/Flash**

This memory is assumed not to change during a debug session. Any code within such a range that is downloaded when starting a debug session (or technically, any such code that is part of the application being debugged) is stored in the cache and remains there. Other parts of such ranges are loaded into the cache from memory on demand, but are then kept during the debug session. Also, C-SPY will not allow you to modify such memory from C-SPY windows.

Even though flash memory is normally used as a fixed read-only memory, there are applications that use parts of flash memory for modifying storage at runtime. For example, some part of flash memory might be used for a file system or simply to store non-volatile information. To reflect this in C-SPY, you should designate those parts of flash memory as one or more RAM ranges instead. Then C-SPY will assume that those parts can change at any time during execution.

**SFR/Uncached**

A range of this type is completely uncached. All read or write commands from a C-SPY window will access the hardware. Typically, this type is useful for special function registers, which can have all sorts of unusual behavior, such as having different values at every read access, which in turn can have side-effects on other registers when being written, not containing the same value as was previously written, etc.

If you do not have the appropriate information about your device, you can specify an entire memory as **SFR/Uncached**. This is not incorrect, but might make C-SPY slower when updating windows. In fact, this is sometimes the default suggestion when there is no memory range information available.
Memory Access Setup dialog box

The Memory Access Setup dialog box is available from the C-SPY driver menu.

This dialog box lists all defined memory areas, where each column in the list specifies the properties of the area. In other words, the dialog box displays the memory access setup that will be used during the simulation.

Note: If you enable both the Use ranges based on and the Use manual ranges option, memory accesses are checked for all defined ranges.

For information about the columns and the properties displayed, see Edit Memory Access dialog box, page 186.

Requirements

The C-SPY simulator.

Use ranges based on

Selects any of the predefined alternatives for the memory access setup. Choose between:

Device description file
   Loads properties from the device description file.
Debug file segment information

Properties are based on the section information available in the debug file. This information is only available while debugging. The advantage of using this option, is that the simulator can catch memory accesses outside the linked application.

Use manual ranges

Specify your own ranges manually via the Edit Memory Access dialog box. To open this dialog box, choose New to specify a new memory range, or select a memory zone and choose Edit to modify it. For more information, see Edit Memory Access dialog box, page 186.

The ranges you define manually are saved between debug sessions.

Memory access checking

Check for determines what to check for:

- Access type violation
- Access to unspecified ranges.

Action selects the action to be performed if an access violation occurs; choose between:

- Log violations
- Log and stop execution.

Any violations are logged in the Debug Log window.

Buttons

These buttons are available:

New

Opens the Edit Memory Access dialog box, where you can specify a new memory range and attach an access type to it, see Edit Memory Access dialog box, page 186.

Edit

Opens the Edit Memory Access dialog box, where you can edit the selected memory area. See Edit Memory Access dialog box, page 186.

Delete

Deletes the selected memory area definition.

Delete All

Deletes all defined memory area definitions.
Note that except for the OK and Cancel buttons, buttons are only available when the option **Use manual ranges** is selected.

**Edit Memory Access dialog box**

The Edit Memory Access dialog box is available from the Memory Access Setup dialog box.

![Edit Memory Access dialog box](image)

**Requirements**

The C-SPY simulator.

**Memory range**

Defines the memory area specific to your device:

- **Zone**
  Selects a memory zone, see *C-SPY memory zones*, page 155.

- **Start address**
  Specify the start address for the memory area, in hexadecimal notation.

- **End address**
  Specify the end address for the memory area, in hexadecimal notation.

**Access type**

Selects an access type to the memory range; choose between:

- Read and write
- Read only
- Write only.
Trace

This chapter gives you information about collecting and using trace data in C-SPY®. More specifically, this means:

- Introduction to using trace
- Collecting and using trace data
- Reference information on trace.

Introduction to using trace

This section introduces trace.

These topics are covered:

- Reasons for using trace
- Briefly about trace
- Requirements for using trace.

See also:

- Getting started using data logging, page 9
- Getting started using event logging, page 10
- Power debugging, page 2
- Getting started using interrupt logging, page 16
- Profiling, page 2.

REASONS FOR USING TRACE

By using trace, you can inspect the program flow up to a specific state, for instance an application crash, and use the trace data to locate the origin of the problem. Trace data can be useful for locating programming errors that have irregular symptoms and occur sporadically.

Reasons for using the trace triggers and trace filters

By using trace trigger and trace filter conditions, you can select the interesting parts of your source code and use the trace buffer in the J-Trace probe more efficiently. Trace triggers—Trace Start and Trace Stop breakpoints—specify for example a code section
for which you want to collect trace data. A trace filter specifies conditions that, when
fulfilled, activate the trace data collection during execution.

For ARM7/9 devices, you can specify up to 16 trace triggers and trace filters in total, of
which 8 can be trace filters.

For Cortex-M devices, you can specify up to 4 trace triggers and trace filters in total.

BRIEFLY ABOUT TRACE

Your target system must be able to generate trace data. Once generated, C-SPY can
collect it and you can visualize and analyze the data in various windows and dialog
boxes.

C-SPY supports collecting trace data from these target systems:

- Devices with support for ETM (Embedded Trace Macrocell)—ETM trace
- Devices with support for the SWD (Serial Wire Debug) interface using the SWO
  (Serial Wire Output) communication channel—SWO trace
- The C-SPY simulator.

Depending on your target system, different types of trace data can be generated.

ETM trace

ETM trace (also known as full trace) is a continuously collected sequence of every
executed instruction for a selected portion of the execution. It is only possible to collect
as much data as the buffer can hold.

The debug probe contains a trace buffer that collects trace data in real time, but the data
is not displayed in the C-SPY windows until after the execution has stopped.

SWO trace

SWO trace is a sequence of events of various kinds, generated by the on-chip debug
hardware. The events are transmitted in real time from the target system over the SWO
communication channel. This means that the C-SPY windows are continuously updated
while the target system is executing. The most important events are:

- PC sampling
  The hardware can sample and transmit the value of the program counter at regular
  intervals. This is not a continuous sequence of executed instructions (like ETM
  trace), but a sparse regular sampling of the PC. A modern ARM CPU typically
  executes millions of instructions per second, while the PC sampling rate is usually
  counted in thousands per second.
• Interrupt logs
  The hardware can generate and transmit data related to the execution of interrupts, generating events when entering and leaving an interrupt handler routine.

• Data logs
  Using Data Log breakpoints, the hardware can be configured to generate and transmit events whenever a certain variable, or simply an address range, is accessed by the CPU.

The SWO channel has limited throughput, so it is usually not possible to use all the above features at the same time, at least not if either the frequency of PC sampling, of interrupts, or of accesses to the designated variables is high.

If you use the SWO communication channel on a trace probe, the data will be collected in the trace buffer and displayed after the execution has stopped.

**Trace features in C-SPY**

In C-SPY, you can use the trace-related windows Trace, Function Trace, Timeline, and Find in Trace. In the C-SPY simulator, you can also use the Trace Expressions window. Depending on your C-SPY driver, you can set various types of trace breakpoints and triggers to control the collection of trace data.

If you use the C-SPY J-Link/J-Trace driver, you have access to windows such as the Interrupt Log, Interrupt Log Summary, Data Log, and Data Log Summary windows.

When you are debugging, two buttons labeled ETM and SWO, respectively, are visible on the IDE main window toolbar. If any of these buttons is green, it means that the corresponding trace hardware is generating trace data. Just point at the button with the mouse pointer to get detailed tooltip information about which C-SPY features that have requested trace data generation. This is useful, for example, if your SWO communication channel often overflows because too many of the C-SPY features are currently using trace data. Clicking on the buttons opens the corresponding setup dialog boxes.

In addition, several other features in C-SPY also use trace data, features such as the Profiler, Code coverage, and Instruction profiling.

**REQUIREMENTS FOR USING TRACE**

The C-SPY simulator supports trace-related functionality, and there are no specific requirements.

**Note:** The specific set of debug components you are using (hardware, a debug probe, and a C-SPY driver) determine which trace features in C-SPY that are supported.
Requirements for using ETM trace

ETM trace is available for some ARM devices.

To use ETM trace you need one of these combinations:

- An I-jet in-circuit debugging probe and a device that supports ETM via ETB (Embedded Trace Buffer). The JTAGjet debug probe reads ETM data from the ETB buffer. Make sure to use the C-SPY I-jet/JTAGjet driver.
- A J-Trace debug probe and a device that supports ETM. Make sure to use the C-SPY J-Link/J-Trace driver.
- A J-Link debug probe and a device that supports ETM via ETB (Embedded Trace Buffer). The J-Link probe reads ETM data from the ETB buffer. Make sure to use the C-SPY J-Link/J-Trace driver.

Requirements for using SWO trace

To use SWO trace you need an I-jet in-circuit debugging probe, a J-Link, J-Trace, or ST-LINK debug probe that supports the SWO communication channel and a device that supports the SWD/SWO interface.

Requirements for using the trace triggers and trace filters

The trace triggering and trace filtering features are available only for JTAGjet and J-Trace and when using an ARM7/9 or Cortex-M device.

Collecting and using trace data

This section describes various tasks related to collecting and using trace data.

More specifically, you will get information about:

- Getting started with ETM trace
- Trace data collection using breakpoints
- Searching in trace data
- Browsing through trace data.

GETTING STARTED WITH ETM TRACE

Before you start C-SPY:

- For your device, the trace port must be set up. For some devices this is done automatically when the trace logic is enabled. However, for some devices, typically Atmel and ST devices based on ARM 7 or ARM 9, you need to set up the trace port explicitly. You do this by means of a C-SPY macro file. You can find examples of
such files (ETM_init*.mac) in the example projects. To use a macro file, choose
Project>Options>Debugger>Setup>Use macro files. Specify your macro file; a
browse button is available for your convenience.

Note that the pins used on the hardware for the trace signals cannot be used by your
application.

2 Start C-SPY and choose Trace Setup from the C-SPY driver menu. In the Trace
Settings dialog box that appears, check if you need to change any of the default
settings.

3 Open the Trace window—available from the driver-specific menu—and click the
Activate button to enable collecting trace data.

4 Start the execution. When the execution stops, for instance because a breakpoint is
triggered, trace data is displayed in the Trace window. For more information about the
window, see Trace window, page 205.

GETTING STARTED WITH SWO TRACE

To get started using SWO trace:

1 Before you start C-SPY, choose Project>Options>I-jet/JTAGjet for I-jet/JTAGjet,
   Project>Options>J-Link/J-Trace for J-Link/J-Trace, or Project>Options>ST-Link
   for ST-LINK, respectively.
   
   Click the JTAG/SWD tab or the Connection tab, respectively, and choose
   Interface>SWD. Alternatively, for I-jet, choose JTAG and the option SWO>SWO on
   the TraceD0 pin.

2 After you have started C-SPY, choose SWO Trace Windows Settings from the C-SPY
driver menu. In the dialog box that appears, make your settings for controlling the
output in the Trace window.

   To see statistical trace data, select the option Force>PC samples, see SWO Trace
Window Settings dialog box, page 199.

3 To configure the hardware’s generation of trace data, click the SWO Configuration
button available in the SWO Configuration dialog box. For more information, see
SWO Configuration dialog box, page 201.

Note specifically these settings:

- The value of the CPU clock option must reflect the frequency of the CPU clock
  speed at which the application executes. Note also that the settings you make are
  preserved between debug sessions.

- To decrease the amount of transmissions on the communication channel, you can
disable the Timestamp option. Alternatively, set a lower rate for PC Sampling or
use a higher SWO clock frequency.
4 Open the SWO Trace window—available from the C-SPY driver menu—and click the Activate button to enable trace data collection.

5 Start the execution. The Trace window is continuously updated with trace data. For more information about this window, see Trace window, page 205.

SETTING UP CONCURRENT USE OF ETM AND SWO

If you have a JTAGjet or a J-Trace debug probe for Cortex-M3, you can use ETM trace and SWO trace concurrently.

In this case, if you activate the ETM trace and the SWO trace, SWO trace data will also be collected in the ETM trace buffer, instead of being streamed via the SWO channel. This means that the SWO trace data will not be displayed until the execution has stopped, instead of being continuously updated live in the SWO Trace window.

TRACE DATA COLLECTION USING BREAKPOINTS

A convenient way to collect trace data between two execution points is to start and stop the data collection using dedicated breakpoints. Choose between these alternatives:

- In the editor or Disassembly window, position your insertion point, right-click, and toggle a Trace Start or Trace Stop breakpoint from the context menu.
- In the Breakpoints window, choose Trace Start, Trace Stop, or Trace Filter.
- The C-SPY system macros `__setTraceStartBreak` and `__setTraceStopBreak` can also be used.

For more information about these breakpoints, see Trace Start breakpoints dialog box, page 220 and Trace Stop breakpoints dialog box, page 221, respectively.

Using the trace triggers and trace filters:

1 Use the Trace Start dialog box to set a start condition—a start trigger—to start collecting trace data.

2 Use the Trace Stop dialog box to set a stop condition—a stop trigger—to stop collecting trace data.

3 Optionally, set additional conditions for the trace data collection to continue. Then set one or more trace filters, using the Trace Filter dialog box.

4 If needed, set additional trace start or trace stop conditions.

5 Enable the Trace window and start the execution.

6 Stop the execution.
You can view the trace data in the Trace window and in browse mode also in the Disassembly window, where also the trace marks for your trace triggers and trace filters are visible.

If you have set a trace filter, the trace data collection is performed while the condition is true plus some further instructions. When viewing the trace data and looking for a certain data access, remember that the access took place one instruction earlier.

**SEARCHING IN TRACE DATA**

When you have collected trace data, you can perform searches in the collected data to locate the parts of your code or data that you are interested in, for example, a specific interrupt or accesses of a specific variable.

You specify the search criteria in the **Find in Trace** dialog box and view the result in the Find in Trace window.

*Note:* The Find in Trace window is very similar to the Trace window, showing the same columns and data, but only those rows that match the specified search criteria. Double-clicking an item in the Find in Trace window brings up the same item in the Trace window.

**To search in your trace data:**

1. On the Trace window toolbar, click the **Find** button.
2. In the **Find in Trace** dialog box, specify your search criteria.

   Typically, you can choose to search for:
   - A specific piece of text, for which you can apply further search criteria
   - An address range
   - A combination of these, like a specific piece of text within a specific address range.

   For more information about the various options, see **Find in Trace dialog box**, page 236.

3. When you have specified your search criteria, click **Find**. The Find in Trace window is displayed, which means you can start analyzing the trace data. For more information, see **Find in Trace window**, page 237.

**BROWSING THROUGH TRACE DATA**

To follow the execution history, simply look and scroll in the Trace window. Alternatively, you can enter browse mode.

To enter browse mode, double-click an item in the Trace window, or click the **Browse** toolbar button.
The selected item turns yellow and the source and disassembly windows will highlight the corresponding location. You can now move around in the trace data using the up and down arrow keys, or by scrolling and clicking; the source and Disassembly windows will be updated to show the corresponding location. This is like stepping backward and forward through the execution history.

Double-click again to leave browse mode.

Reference information on trace

This section gives reference information about these windows and dialog boxes:

- ETM Trace Settings dialog box (J-Link/J-Trace), page 197
- SWO Trace Window Settings dialog box, page 199
- SWO Configuration dialog box, page 201
- Trace window, page 205
- Function Trace window, page 209
- Trace Expressions window, page 235
- Timeline window, page 209
- Viewing Range dialog box, page 218
- Power Log window, page 14
- Trace Start breakpoints dialog box, page 220 (simulator)
- Trace Stop breakpoints dialog box, page 221 (simulator)
- Trace Start breakpoints dialog box (I-jet/JTAGjet), page 222
- Trace Stop breakpoints dialog box (I-jet/JTAGjet), page 224
- Trace Filter breakpoints dialog box (I-jet/JTAGjet), page 226
- Trace Start breakpoints dialog box (J-Link/J-Trace), page 227
- Trace Stop breakpoints dialog box (J-Link/J-Trace), page 230
- Trace Filter breakpoints dialog box (J-Link/J-Trace), page 232
- Trace Expressions window, page 235
- Find in Trace dialog box, page 236
- Find in Trace window, page 237
- Trace Save dialog box, page 238
ETM Trace Settings dialog box (I-jet/JTAGjet)

The ETM Trace Settings dialog box is available from the C-SPY driver menu.

![ETM Trace Settings dialog box](image)

Use this dialog box to configure ETM trace generation and collection.

See also:
- Requirements for using ETM trace, page 190
- Getting started with ETM trace, page 190.

Requirements

The C-SPY I-jet/JTAGjet driver.

Prefer ETB over ETM

Uses ETB instead of ETM. Typically, this is necessary if you are using the I-jet in-circuit debugging probe and have on-chip ETB.

Port width

Specifies the trace bus width, which can be set to 1, 2, 4, 8, or 16 bits. The value must correspond with what is supported by the hardware and the debug probe.

For the lower values, the risk of FIFO buffer overflow increases, unless you are using the Stall processor on FIFO full option.

Port mode

Specifies the used trace clock rate:
- Normal, full-rate clocking
Reference information on trace

Data Trace

Selects what type of trace data you want C-SPY to collect. Choose between:

- **Collect data values**
  Collects data values.

- **Collect data addresses**
  Collects data addresses.

- **Limit data tracing to address range**
  Collects the specified type of data within the address range you specify in the First and Last text boxes.

Capture

Normally, trace collection starts or stops when execution starts or stops, or when a Trace Start or Trace Stop breakpoint is triggered. To change this, choose between:

- **Stop on buffer full**
  Stops collecting trace data when the probe buffer is full.

- **Stall processor on FIFO full**
  Stalls the processor in case the FIFO buffer fills up. The trace FIFO buffer on the CPU might in some situations become full—FIFO buffer overflow—which means trace data will be lost. This can be the case when the CPU is executing several branch instructions close to each other in time, such as in tight loops.

Status

Shows the ETM status.

Trace clock frequency

Shows the frequency of the trace clock to help you determine if the trace hardware is properly configured. Typically, this depends on the settings of Port Width and Port Mode.

Apply settings

Applies the settings you made in this dialog box. The trace clock frequency will be updated.
ETM Trace Settings dialog box (J-Link/J-Trace)

The ETM Trace Settings dialog box is available from the C-SPY driver menu.

![ETM Trace Settings dialog box (J-Link/J-Trace)](image)

Figure 81: ETM Trace Settings dialog box (J-Link/J-Trace)

Use this dialog box to configure ETM trace generation and collection.

See also:
- Requirements for using ETM trace, page 190
- Getting started with ETM trace, page 190.

Requirements

The C-SPY J-Link/J-Trace driver.

Trace port width

Specifies the trace bus width, which can be set to 1, 2, 4, 8, or 16 bits. The value must correspond with what is supported by the hardware and the debug probe. For Cortex-M3, 1, 2, and 4 bits are supported by the J-Trace debug probe. For ARM7/9, only 4 bits are supported by the J-Trace debug probe.

For the lower values, the risk of FIFO buffer overflow increases, unless you are using the Stall processor on FIFO full option.

Trace port mode

Specifies the used trace clock rate:
- Normal, full-rate clocking
- Normal, half-rate clocking
- Multiplexed
- Demultiplexed
- Demultiplexed, half-rate clocking.
Reference information on trace

Note: For RDI drivers, only the two first alternatives are available. For the J-Trace driver, the available alternatives depend on the device you are using.

**Trace buffer size**

Specify the size of the trace buffer. By default, the number of trace frames is `0xFFFF`. For ARM7/9 the maximum number is `0xFFFFF`, and for Cortex-M3 the maximum number is `0x3FFFFF`.

For ARM7/9, one trace frame corresponds to 2 bytes of the physical J-Trace buffer size. For Cortex-M3, one trace frame corresponds to approximately 1 byte of the buffer size.

Note: The Trace buffer size option is only available for the J-Trace driver.

**Cycle accurate tracing**

Emits trace frames synchronous to the processor clock even when no trace data is available. This makes it possible to use the trace data for real-time timing calculations. However, if you select this option, the risk for FIFO buffer overflow increases.

Note: This option is only available for ARM7/9 devices.

**Broadcast all branches**

Makes the processor send more detailed address trace information. However, if you select this option, the risk for FIFO buffer overflow increases.

Note: This option is only available for ARM7/9 devices. For Cortex, this option is always enabled.

**Stall processor on FIFO full**

Stalls the processor in case the FIFO buffer fills up. The trace FIFO buffer on the CPU might in some situations become full—FIFO buffer overflow—which means trace data will be lost. This can be the case when the CPU is executing several branch instructions close to each other in time, such as in tight loops.

**Show timestamp**

Makes the Trace window display seconds instead of cycles in the Index column. To make this possible you must also specify the appropriate speed for your CPU in the Trace port (CPU core) speed text box.

Note: This option is only available when you use the J-Trace driver with ARM7/9 devices.
SWO Trace Window Settings dialog box

The SWO Trace Window Settings dialog box is available from the I-jet/JTAGjet menu, the J-Link menu or the ST-LINK menu, respectively, alternatively from the SWO Trace window toolbar.

Figure 82: SWO Trace Window Settings dialog box

Use this dialog box to specify what to display in the SWO Trace window.

Note that you also need to configure the generation of trace data, click SWO Configuration. For more information, see SWO Configuration dialog box, page 201.

Requirements

One of these alternatives:

- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY ST-LINK driver.

Force

Enables data generation, if it is not already enabled by other features using SWO trace data. The Trace window displays all generated SWO data. Other features in C-SPY, for example Profiling, can also enable SWO trace data generation. If no other feature has enabled the generation, use the Force options to generate SWO trace data.

The generated data will be displayed in the Trace window. Choose between:

Time Stamps

Enables timestamps for various SWO trace packets, that is sent over the SWO communication channel. Use the resolution drop-down list to choose the resolution of the timestamp value. For example, 1 to count every cycle, or 16 to count every 16th cycle. Note that the lowest resolution is only useful if the time between each event packet is long enough. 16 is useful if using a low SWO clock frequency.

This option does not apply to I-jet.
PC samples
Enables sampling the program counter register, PC, at regular intervals. To choose the sampling rate, see SWO Configuration dialog box, page 201, specifically the option PC Sampling.

Interrupt Logs
Forces the generation of interrupt logs to the SWO Trace window. For information about other C-SPY features that also use trace data for interrupts, see Interrupts, page 271.

ITM Log
Forces the generation of ITM logs to the SWO Trace window.
This option applies to I-jet only.

Generate
Enables trace data generation for these events. The generated data will be displayed in the Trace window. The value of the counters are displayed in the Comment column in the SWO Trace window. Choose between:

CPI
Enables generation of trace data for the CPI counter.

EXC
Enables generation of trace data for the EXC counter.

SLEEP
Enables generation of trace data for the SLEEP counter.

LSU
Enables generation of trace data for the LSU counter.

FOLD
Enables generation of trace data for the FOLD counter.

SWO Configuration
Displays the SWO Configuration dialog box where you can configure the hardware’s generation of trace data. See SWO Configuration dialog box, page 201.
This button is not available when you are using I-jet.
SWO Configuration dialog box

The SWO Configuration dialog box is available from the C-SPY driver menu, alternatively from the SWO Trace Window Settings dialog box.

![SWO Configuration dialog box]

Use this dialog box to configure the serial-wire output communication channel and the hardware’s generation of trace data.

See also *Getting started with SWO trace*, page 191.

Requirements

One of these alternatives:

- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY ST-LINK driver.
PC Sampling

Controls the behavior of the sampling of the program counter. You can specify:

**In use by**

Lists the features in C-SPY that can use trace data for PC Sampling. ON indicates features currently using trace data. OFF indicates features currently not using trace data.

**Rate**

Use the drop-down list to choose the sampling rate, that is, the number of samples per second. The highest possible sampling rate depends on the SWO clock value and on how much other data that is sent over the SWO communication channel. The higher values in the list will not work if the SWO communication channel is not fast enough to handle that much data.

This option does not apply to I-jet.

**Divider**

Select a divider, that, applied to the CPU clock speed, determines the rate of PC samples. The highest possible sampling rate depends on the SWO clock value and on how much other data that is sent over the SWO communication channel. The smaller values in the list will not work if the SWO communication channel is not fast enough to handle that much data.

This option applies to I-jet only.

Data Log Events

Specifies what to log when a Data Log breakpoint is triggered. These items are available:

**In use by**

Lists the features in C-SPY that can use trace data for Data Log Events. ON indicates features currently using trace data. OFF indicates features currently not using trace data.

**PC only**

Logs the value of the program counter.

**PC + data value + base addr**

Logs the value of the program counter, the value of the data object, and its base address.

**Data value + exact addr**

Logs the value of the data object and the exact address of the data object that was accessed.
Interrupt Log

Lists the features in C-SPY that can use trace data for Interrupt Logs. ON indicates features currently using trace data. OFF indicates features currently not using trace data. For more information about interrupt logging, see Interrupts, page 271.

Override project default

Overrides the CPU clock and the SWO clock default values on the Project>Options>J-Link/J-Trace>Setup page for J-Link/J-Trace or on the Project>Options>ST-Link>Setup page for ST-LINK, respectively.

This option does not apply to I-jet.

Override project settings

Overrides the CPU clock and the SWO prescaler default values on the Project>Options>I-jet>Setup page.

This option only applies to I-jet.

CPU clock

Specify the exact clock frequency used by the internal processor clock, HCLK, in MHz. The value can have decimals.

This value is used for configuring the SWO communication speed.

For J-Link and ST-LINK, this value is also used for calculating timestamps.

SWO clock

Specify the clock frequency of the SWO communication channel in kHz. Choose between:

Autodetect

Automatically uses the highest possible frequency that the J-Link debug probe can handle. When it is selected, the Wanted text box displays that frequency.

Wanted

Manually selects the frequency to be used, if Autodetect is not selected. The value can have decimals. Use this option if data packets are lost during transmission.

Actual

Displays the frequency that is actually used. This can differ a little from the wanted frequency.

This option does not apply to I-jet.
**SWO prescaler**

Specify the clock prescaler of the SWO communication channel. The prescaler, in turn, determines the CPU clock frequency. If data packets are lost during transmission, try using a higher prescaler value. Choose between:

**Auto**
- Automatically uses the highest possible frequency that the I-jet debug probe can handle.

1, 2, 5, 10, 20, 50, 100
- The prescaler value.

This option applies to I-jet only.

** Timestamps**

Selects the resolution of the timestamp value. For example, 1 to count every cycle, or 16 to count every 16th cycle. Note that the lowest resolution is only useful if the time between each event packet is long enough.

This option does not apply to I-jet.

**ITM Stimulus Ports**

Selects which ports you want to redirect and to where. The ITM Stimulus Ports are used for sending data from your application to the debugger host without stopping the program execution. There are 32 such ports. Choose between:

**Enabled ports**
- Enables the ports to be used. Only enabled ports will actually send any data over the SWO communication channel to the debugger.
- Port 0 is used by the terminal I/O library functions.
- Ports 1-4 are used by the ITM macros for the Event Log window.
- Port 5 is used for an optional PC value added to the ITM macro.

**To Terminal I/O window**
- Specifies the ports to use for routing data to the Terminal I/O window.

**To Log File**
- Specifies the ports to use for routing data to a log file. To use a different log file than the default one, use the browse button.

The `stdout` and `stderr` of your application can be routed via SWO to the C-SPY Terminal I/O window, instead of via semihosting. To achieve this, choose Project>Options>General Options>Library Configuration>Library low-level
interface implementation>stdout/stderr>Via SWO. This will significantly improve the performance of stdout/stderr, compared to when semihosting is used.

This can be disabled if you deselect the port settings in the Enabled ports and To Terminal I/O options.

Trace window

The Trace window is available from the C-SPY driver menu.

Trace window

![Figure 84: The Trace window](image)

Note: There are three different Trace windows—ETM Trace, SWO Trace, and just Trace for the C-SPY simulator. The windows look slightly different.

This window displays the collected trace data.

The Trace window depends on the C-SPY driver you are using, where the content differs depending on the C-SPY driver you are using and the trace support of your debug probe. This figure reflects the C-SPY simulator.

Requirements

One of these alternatives:

- The C-SPY Simulator.
- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY ST-LINK driver.
Trace toolbar

The toolbar in the Trace window and in the Function trace window contains:

Enable/Disable
Enabled and disables collecting and viewing trace data in this window. This button is not available in the Function trace window.

Clear trace data
Clears the trace buffer. Both the Trace window and the Function trace window are cleared.

Toggle source
Toggles the Trace column between showing only disassembly or disassembly together with the corresponding source code.

Browse
Toggles browse mode on or off for a selected item in the Trace window, see Browsing through trace data, page 193.

Find
Displays a dialog box where you can perform a search, see Find in Trace dialog box, page 236.

Save
In the ETM Trace and SWO Trace windows this button displays the Trace Save dialog box, see Trace Save dialog box, page 238. In the C-SPY simulator this button displays a standard Save As dialog box where you can save the collected trace data to a text file, with tab-separated columns.

Edit Settings
In the C-SPY simulator, this button is not enabled.

In the ETM Trace window this button displays the Trace Settings dialog box, see ETM Trace Settings dialog box (J-Link/J-Trace), page 197 and ETM Trace Settings dialog box (I-jet/JTAGjet), page 195.

In the SWO Trace window this button displays the SWO Trace Window Settings dialog box, see SWO Trace Window Settings dialog box, page 199.

Edit Expressions (C-SPY simulator only)
Opens the Trace Expressions window, see Trace Expressions window, page 235.

Display area (in the C-SPY simulator)
This area displays a collected sequence of executed machine instructions. In addition, the window can display trace data for expressions.
This area contains these columns for the C-SPY simulator:

#  
A serial number for each row in the trace buffer. Simplifies the navigation within the buffer.

Cycles  
The number of cycles elapsed to this point.

Trace  
The collected sequence of executed machine instructions. Optionally, the corresponding source code can also be displayed.

Expression  
Each expression you have defined to be displayed appears in a separate column. Each entry in the expression column displays the value after executing the instruction on the same row. You specify the expressions for which you want to collect trace data in the Trace Expressions window, see Trace Expressions window, page 235.

Display area (for ETM trace)

Index  
A number that corresponds to each packet. Examples of packets are instructions, synchronization points, and exception markers.

Frame/Time  
When collecting trace data in cycle-accurate mode (requires ARM7/9)—enable Cycle accurate tracing in the ETM Trace Settings dialog box—the value corresponds to the number of elapsed cycles since the start of the execution. This column is only available for the J-Link/J-Trace driver.

When collecting trace data in non-cycle-accurate mode, the value corresponds to an approximate amount of cycles. For Cortex-M devices, the value is repeatedly calibrated with the actual number of cycles.

When the Show timestamp option is selected in the ETM Trace Settings dialog box, the value displays the time instead of cycles. To display the value as time requires collecting data in cycle-accurate mode, see ETM Trace Settings dialog box (I-jet/JTAGjet), page 195 and ETM Trace Settings dialog box (J-Link/J-Trace), page 197 (specifically the Cycle accurate tracing option), and the J-Link/J-Trace driver.

Address  
The address of the executed instruction.
Op Code
The operation code of the executed instruction.

Trace
The collected sequence of executed machine instructions. Optionally, the corresponding source code can also be displayed.

Comment
This column is only available for the J-Link/J-Trace driver.

Note: For RDI drivers, this window looks slightly different.

Display area (for SWO trace)
This area contains these columns for SWO trace:

Index
An index number for each row in the trace buffer. Simplifies the navigation within the buffer.

SWO Packet
The contents of the captured SWO packet.

Cycles
The approximate number of cycles from the start of the execution until the event.

Event
The event type of the captured SWO packet. If the column displays *Overflow*, the data packet could not be sent, because too many SWO features use the SWO channel at the same time. To decrease the amount of transmissions on the communication channel, point at the SWO button—on the IDE main window toolbar—with the mouse pointer to get detailed tooltip information about which C-SPY features that have requested trace data generation. Disable some of the features.

Value
The event value, if any.

Trace
If the event is a sampled PC value, the instruction is displayed in this column. Optionally, the corresponding source code can also be displayed.

Comment
Additional information. This includes the values of the selected Trace Events counters, or the number of the comparator (hardware breakpoint) used for the Data Log breakpoint.
If the display area seems to show garbage, make sure you specified a correct value for the CPU clock in the SWO Configuration dialog box.

Function Trace window

The Function Trace window is available from the C-SPY driver menu during a debug session.

![Function Trace window](image)

This window displays a subset of the trace data displayed in the Trace window. Instead of displaying all rows, the Function Trace window only shows trace data corresponding to calls to and returns from functions.

Requirements

One of these alternatives:

- The C-SPY Simulator.
- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY ST-LINK driver.

Toolbar

For information about the toolbar, see Trace window, page 205.

Display area

For information about the columns in the display area, see Trace window, page 205

Timeline window

The Timeline window is available from the C-SPY driver menu during a debug session.
Depending on the abilities in hardware, the debug probe, and the C-SPY driver you are using, this window displays trace data in different graphs in relation to a common time axis:

- Call Stack graph
- Data Log graph
- Event Log graph
- Interrupt Log graph
- Power Log graph.

To display a graph:

1. Choose `C-SPY driver > SWO Configuration` to open the SWO Configuration dialog box. Make sure the CPU clock option is set to the same value as the CPU clock value set by your application. This is necessary to set the SWO clock and to obtain a correct data transfer to the debug probe.
   
   If you are using the C-SPY simulator you can ignore this step.

2. Choose Timeline from the C-SPY driver menu to open the Timeline window.

3. In the Timeline window, click in the graph area and choose Enable from the context menu to enable a specific graph.

4. For the Data Log Graph, you need to set a Data Log breakpoint for each variable you want a graphical representation of in the Timeline window. See `Data Log breakpoints dialog box`, page 34.

5. For the Event Log Graph, you must add a preprocessor macro to your application source code where you want event logs to be generated. See `Getting started using event logging`, page 10.

6. Click Go on the toolbar to start executing your application. The graph appears.

   To navigate in the graph, use any of these alternatives:

   - Right-click and from the context menu choose Zoom In or Zoom Out. Alternatively, use the + and - keys. The graph zooms in or out depending on which command you used.
   - Right-click in the graph and from the context menu choose Navigate and the appropriate command to move backwards and forwards on the graph. Alternatively, use any of the shortcut keys: arrow keys, Home, End, and Ctrl+End.
   - Double-click on a sample of interest and the corresponding source code is highlighted in the editor window and in the Disassembly window.
   - Click on the graph and drag to select a time interval. Press Enter or right-click and from the context menu choose Zoom>Zoom to Selection. The selection zooms in.
Point in the graph with the mouse pointer to get detailed tooltip information for that location.

Requirements

Depending on the abilities in hardware, the debug probe, and the C-SPY driver you are using, the display area can be populated with different graphs:

<table>
<thead>
<tr>
<th>Graphs</th>
<th>Call Stack Graph</th>
<th>Data Log Graph</th>
<th>Event Log Graph</th>
<th>Interrupt Log Graph</th>
<th>Power Log Graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-SPY simulator</td>
<td>X</td>
<td>X</td>
<td>--</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>i-jet</td>
<td>--</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>JTAGGet</td>
<td>X</td>
<td>--</td>
<td>--</td>
<td>--</td>
<td>--</td>
</tr>
<tr>
<td>J-Link</td>
<td>--</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>J-Trace</td>
<td>X</td>
<td>X1</td>
<td>X</td>
<td>X</td>
<td>--</td>
</tr>
<tr>
<td>ST-LINK</td>
<td>--</td>
<td>X</td>
<td>--</td>
<td>X</td>
<td>--</td>
</tr>
</tbody>
</table>

Table 9: Supported graphs in the Timeline window

1 Only available when ETM trace is disabled.

For more information about requirements related to trace data, see Requirements for using trace, page 189.

Display area for the Call Stack Graph

The Call Stack Graph displays the sequence of calls and returns collected by ETM trace.

Figure 86: Timeline window with Call Stack graph
At the bottom of the graph you will usually find `main`, and above it, the functions called from `main`, and so on. The horizontal bars, which represent invocations of functions, use four different colors:

- Medium green for normal C functions with debug information
- Light green for functions known to the debugger only through an assembler label
- Medium or light yellow for interrupt handlers, with the same distinctions as for green.

The numbers represent the number of cycles spent in, or between, the function invocations.

At the bottom of the window, there is a common time axis that uses seconds as the time unit.

**Display area for the Data Log graph**

The Data Log graph displays the data logs generated by SWO trace or by the C-SPY simulator, for up to four different variables or address ranges specified as Data Log breakpoints.

*Figure 87: Timeline window with Data Log graph*
Where:

- Each graph is labeled with—in the left-side area—the variable name or address for which you have specified the Data Log breakpoint.
- The graph itself displays how the value of the variable changes over time. The label area also displays the limits, or range, of the Y-axis for a variable. You can use the context menu to change these limits. The graph is a graphical representation of the information in the Data Log window, see Data Log window, page 33.
- The graph can be displayed either as a thin line or as a color-filled solid graph.
- A red vertical line indicates overflow, which means that the communication channel failed to transmit all data logs from the target system.

At the bottom of the window, there is a common time axis that uses seconds as the time unit.

**Display area for the Event Log graph**

The Event Log Graph displays the event logs produced when the execution passes specific positions in your application code.

Where:

- Each graph is labeled—in the left-side area—with the name of the channel.
- For each channel, there will be a vertical line that indicates when the event occurred. Optionally, you can choose to display the event value that was passed with the event.
A red vertical line indicates overflow, which means that the communication channel failed to transmit all data logs from the target system.

At the bottom of the window, there is a common time axis that uses seconds as the time unit.

**Display area for the Interrupt Log graph**

The Interrupt Log graph displays interrupts reported by SWO trace or by the C-SPY simulator. In other words, the graph provides a graphical view of the interrupt events during the execution of your application.

Where:

- The label area at the left end of the graph shows the names of the interrupts.
- The graph itself shows active interrupts as a thick green horizontal bar where the white figure indicates the time spent in the interrupt. This graph is a graphical representation of the information in the Interrupt Log window, see *Interrupt Log window*, page 30.

At the bottom of the window, there is a common time axis that uses seconds as the time unit.
Display area for the Power Log graph

The Power Log graph displays power measurement samples generated by the debug probe or associated hardware.

Selection and navigation

Click and drag to select. The selection extends vertically over all graphs, but appears highlighted in a darker color for the selected graph. You can navigate backward and forward in the selected graph using the left and right arrow keys. Use the Home and End keys to move to the first or last relevant point, respectively. Use the navigation keys in combination with the Shift key to extend the selection.

Context menu

This context menu is available:

- Navigate
- Auto Scale
- Zoom
- Call Stack
- Enable
- Go To Source
- Select Graphs
- Time Axis Unit

Note: The context menu contains some commands that are common to all graphs and some commands that are specific to each graph. The figure reflects the context menu for the Call Stack Graph, which means that the menu looks slightly different for the other graphs.
These commands are available:

**Navigate (All graphs)**

Commands for navigating over the graph(s); choose between:

- **Next** moves the selection to the next relevant point in the graph. Shortcut key: right arrow.
- **Previous** moves the selection backward to the previous relevant point in the graph. Shortcut key: left arrow.
- **First** moves the selection to the first data entry in the graph. Shortcut key: Home.
- **Last** moves the selection to the last data entry in the graph. Shortcut key: End.
- **End** moves the selection to the last data in any displayed graph, in other words the end of the time axis. Shortcut key: Ctrl+End.

**Auto Scroll (All graphs)**

Toggles auto scrolling on or off. When on, the most recently collected data is automatically displayed if you have executed the command **Navigate>End**.

**Zoom (All graphs)**

Commands for zooming the window, in other words, changing the time scale; choose between:

- **Zoom to Selection** makes the current selection fit the window. Shortcut key: Return.
- **Zoom In** zooms in on the time scale. Shortcut key: +.
- **Zoom Out** zooms out on the time scale. Shortcut key: -.
- **10ns, 100ns, 1us**, etc makes an interval of 10 nanoseconds, 100 nanoseconds, 1 microsecond, respectively, fit the window.
- **1ms, 10ms**, etc makes an interval of 1 millisecond or 10 milliseconds, respectively, fit the window.
- **10m, 1h**, etc makes an interval of 10 minutes or 1 hour, respectively, fit the window.

**Data Log (Data Log Graph)**

A heading that shows that the Data Log-specific commands below are available.

**Event Log (Event Log Graph)**

A heading that shows that the Event Log-specific commands below are available.
Power Log (Power Log Graph)
A heading that shows that the Power Log-specific commands below are available.

Call Stack (Call Stack Graph)
A heading that shows that the Call stack-specific commands below are available.

Interrupt (Interrupt Log Graph)
A heading that shows that the Interrupt Log-specific commands below are available.

Enable (All graphs)
Toggles the display of the graph on or off. If you disable a graph, that graph will be indicated as OFF in the Timeline window. If no trace data has been collected for a graph, no data will appear instead of the graph.

Variable (Data Log Graph)
The name of the variable for which the Data Log-specific commands below apply. This menu command is context-sensitive, which means it reflects the Data Log Graph you selected in the Timeline window (one of up to four).

Variable (Event Log Graph)
The name of the channel for which the Event Log-specific commands below apply. This menu command is context-sensitive, which means it reflects the channel in the Event Log Graph you selected in the Timeline window (one of up to four).

Solid Graph (Data Log Graph)
Displays the graph as a color-filled solid graph instead of as a thin line.

Viewing Range (Data, Event, and Power Log Graph)
Displays a dialog box, see Viewing Range dialog box, page 218.

Size (Data, Event, and Power Log Graph)
Determines the vertical size of the graph; choose between Small, Medium, and Large.

Line Graph
Shows the event logs as a thin line graph instead of as a bar graph.

Show Numerical Value (Data, Event, and Power Log Graph)
Shows the numerical value of the variable, in addition to the graph.

Show Numbers (Event Log Graph)
Shows the value of the event log.
Hexadecimal (Event Log Graph)
Determines the display mode for the value. Choose between hexadecimal or decimal. Note that this setting will also control the display mode for the same channel in the Event Log window and the Event Log Summary window.

Go To Source (Common)
Displays the corresponding source code in an editor window, if applicable.

Select Graphs (Common)
Selects which graphs to be displayed in the Timeline window.

Time Axis Unit (Common)
Selects the unit used in the time axis; choose between Seconds and Cycles.

Profile Selection
Enables profiling time intervals in the Function Profiler window. Note that this command is only available if the C-SPY driver supports PC Sampling.

**Viewing Range dialog box**

The Viewing Range dialog box is available from the context menu that appears when you right-click in the Power Log Graph or the Data Log Graph in the Timeline window.

![Viewing Range dialog box](image)

*Figure 92: Viewing Range dialog box*

Use this dialog box to specify the value range, that is, the range for the Y-axis for the graph.

**Requirements**

One of these alternatives:

- The C-SPY Simulator.
- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link//J-Trace driver
- The C-SPY ST-LINK driver.

Range for ...

Selects the viewing range for the displayed values:

Auto
Uses the range according to the range of the values that are actually collected, continuously keeping track of minimum or maximum values. The currently computed range, if any, is displayed in parentheses. The range is rounded to reasonably even limits.

Factory
For the Data Log Graph: Uses the range according to the value range of the variable, for example 0–65535 for an unsigned 16-bit integer.

For the Power Log Graph: Uses the range according to the properties of the measuring hardware.

Custom
Use the text boxes to specify an explicit range.

Scale

Selects the scale type of the Y-axis:

- Linear
- Logarithmic.
Trace Start breakpoints dialog box

The Trace Start dialog box is available from the context menu that appears when you right-click in the Breakpoints window.

Use this dialog box to set a Trace Start breakpoint where you want to start collecting trace data. If you want to collect trace data only for a specific range, you must also set a Trace Stop breakpoint where you want to stop collecting data.

See also, Trace Stop breakpoints dialog box, page 221.

To set a Trace Start breakpoint:

1. In the editor or Disassembly window, right-click and choose Trace Start from the context menu.
   Alternatively, open the Breakpoints window by choosing View>Breakpoints.

2. In the Breakpoints window, right-click and choose New Breakpoint>Trace Start.
   Alternatively, to modify an existing breakpoint, select a breakpoint in the Breakpoints window and choose Edit on the context menu.

3. In the Trigger At text box, specify an expression, an absolute address, or a source location. Click OK.

4. When the breakpoint is triggered, the trace data collection starts.

Requirements

The C-SPY simulator.
Specify the code location of the breakpoint. Alternatively, click the Edit button to open the Enter Location dialog box, see Enter Location dialog box, page 37.

**Trace Stop breakpoints dialog box**

The Trace Stop dialog box is available from the context menu that appears when you right-click in the Breakpoints window.

![Trace Stop breakpoint dialog box](image)

Use this dialog box to set a Trace Stop breakpoint where you want to stop collecting trace data. If you want to collect trace data only for a specific range, you might also need to set a Trace Start breakpoint where you want to start collecting data.

See also, Trace Start breakpoints dialog box, page 220.

**To set a Trace Stop breakpoint:**

1. In the editor or Disassembly window, right-click and choose Trace Stop from the context menu.

   Alternatively, open the Breakpoints window by choosing View>Breakpoints.

2. In the Breakpoints window, right-click and choose New Breakpoint>Trace Stop.

   Alternatively, to modify an existing breakpoint, select a breakpoint in the Breakpoints window and choose Edit on the context menu.

3. In the Trigger At text box, specify an expression, an absolute address, or a source location. Click OK.

4. When the breakpoint is triggered, the trace data collection stops.
Requirements

The C-SPY simulator.

Trigger at

Specify the code location of the breakpoint. Alternatively, click the Edit button to open the Enter Location dialog box, see Enter Location dialog box, page 37.

Trace Start breakpoints dialog box (I-jet/JTAGjet)

The Trace Start dialog box is available from the context menu that appears when you right-click in the Breakpoints window. You can also right-click in the editor window or the Disassembly window, and then choose Toggle Breakpoint (Trace Start).

![Figure 95: Trace Start breakpoints dialog box (I-jet/JTAGjet)](image)

Use this dialog box to set the conditions that determine when to start collecting trace data. When the trace condition is triggered, the trace data collection is started.

Requirements

The C-SPY I-jet/JTAGjet driver.

Trigger at

Specify the starting point of the code section for which you want to collect trace data. You can specify a variable name, an address, or a cycle counter value.

Access Type

Selects the type of memory access that triggers the breakpoint:

Read/Write

Reads from or writes to location.
Read
Reads from location.

Write
Writes to location.

Fetch
Accesses at execution address.

Any accesses of the specified type will activate the trace data collection.

Match data
Enables matching of the accessed data. Choose between:

Value
Specify a data value.

Mask
Specify which part of the value to match (word, halfword, or byte).

Use the Match data options in combination with the Read/Write, Read, or Write access types for data. This option can be useful when you want a trigger when a variable has a certain value.

Note: The Match data options are only available when using a Cortex-M device. For Cortex-M devices, only one breakpoint with Match data can be set. Such a breakpoint uses two breakpoint resources.

Size
Controls the size of the address range, that when reached, will trigger the start of the trace data collection. Choose between:

Auto
Sets the size automatically. This can be useful if Trigger at contains a variable.

Manual
Specify the size of the breakpoint range manually.

Trigger range
Shows the requested range and the effective range to be covered by the trace data collection. The range suggested is either within or exactly the area specified by the Trigger at and the Size options.
Extend to cover requested range

Extends the range so that a whole data structure is covered. For data structures that do not fit the size of the possible ranges supplied by the hardware breakpoint unit, for example three bytes, the range will not cover the whole data structure. Note that the range will be extended beyond the size of the data structure, which might cause false triggers at adjacent data.

This option is not enabled for ARM7/9 devices because the range for such devices will always cover the whole data structure.

Trace Stop breakpoints dialog box (I-jet/JTAGjet)

The Trace Stop dialog box is available from the context menu that appears when you right-click in the Breakpoints window. You can also right-click in the editor window or the Disassembly window, and then choose Toggle Breakpoint (Trace Start).

![Trace Stop breakpoints dialog box](image)

Use this dialog box to set the conditions that determine when to stop collecting trace data. When the trace condition is triggered, the trace data collection is stopped.

Requirements

The C-SPY I-jet/JTAGjet driver.

Trigger at

Specify the end point of the code section for which you want to collect trace data. You can specify a variable name, an address, or a cycle counter value.
Access Type

Selects the type of memory access that triggers the breakpoint:

Read/Write
Reads from or writes to location.

Read
Reads from location.

Write
Writes to location.

Fetch
Accesses at execution address.

Any accesses of the specified type will activate the trace data collection.

Match data

Enables matching of the accessed data. Choose between:

Value Specify a data value.
Mask Specify which part of the value to match (word, halfword, or byte).

Use the Match data options in combination with the Read/Write, Read, or Write access types for data. This option can be useful when you want a trigger when a variable has a certain value.

Note: The Match data options are only available when using a Cortex-M device. For Cortex-M devices, only one breakpoint with Match data can be set. Such a breakpoint uses two breakpoint resources.

Size

Controls the size of the address range, that when reached, will trigger the start of the trace data collection. Choose between:

Auto
Sets the size automatically. This can be useful if Trigger at contains a variable.

Manual
Specify the size of the breakpoint range manually.
Trigger range

Shows the requested range and the effective range to be covered by the trace data collection. The range suggested is either within or exactly the area specified by the Trigger at and the Size options.

Extend to cover requested range

Extends the range so that a whole data structure is covered. For data structures that do not fit the size of the possible ranges supplied by the hardware breakpoint unit, for example three bytes, the range will not cover the whole data structure. Note that the range will be extended beyond the size of the data structure, which might cause false triggers at adjacent data.

This option is not enabled for ARM7/9 devices because the range for such devices will always cover the whole data structure.

Trace Filter breakpoints dialog box (I-jet/JTAGjet)

The Trace Filter dialog box is available from the context menu that appears when you right-click in the Breakpoints window. You can also right-click in the editor window or the Disassembly window, and then choose Toggle Breakpoint (Trace Start).

![Trace Filter dialog box](image)

Use this dialog box to set the conditions that determine when to start collecting trace data. When the trace condition is triggered, the trace data collection is started.

Requirements

The C-SPY I-jet/JTAGjet driver.
Start

Specify the start location of the code section for which you want to collect trace data. Alternatively, click the Edit button to open the Enter Location dialog box, see Enter Location dialog box, page 149.

End

Specify the end location of the code section for which you want to collect trace data. Alternatively, click the Edit button to open the Enter Location dialog box, see Enter Location dialog box, page 149.

Trace Start breakpoints dialog box (J-Link/J-Trace)

The Trace Start dialog box is available from the context menu that appears when you right-click in the Breakpoints window. You can also right-click in the editor window or the Disassembly window, and then choose Toggle Breakpoint (Trace Start).

![Trace Start breakpoints dialog box (J-Link/J-Trace)](image)

Use this dialog box to set the conditions that determine when to start collecting trace data. When the trace condition is triggered, the trace data collection is started.

Requirements

The C-SPY J-Link/J-Trace driver.
Trigger at

Specify the starting point of the code section for which you want to collect trace data. You can specify a variable name, an address, or a cycle counter value.

Size

Controls the size of the address range, that when reached, will trigger the start of the trace data collection. Choose between:

Auto
Sets the size automatically. This can be useful if Trigger at contains a variable.

Manual
Specify the size of the breakpoint range manually.

Trigger range

Shows the requested range and the effective range to be covered by the trace data collection. The range suggested is either within or exactly the area specified by the Trigger at and the Size options.

Extend to cover requested range

Extends the range so that a whole data structure is covered. For data structures that do not fit the size of the possible ranges supplied by the hardware breakpoint unit, for example three bytes, the range will not cover the whole data structure. Note that the range will be extended beyond the size of the data structure, which might cause false triggers at adjacent data.

This option is not enabled for ARM7/9 devices because the range for such devices will always cover the whole data structure.

Access Type

Selects the type of memory access that triggers the breakpoint:

Read/Write
Reads from or writes to location.

Read
Reads from location.

Write
Writes to location.

OP-fetch
Accesses at execution address.
Cycle
The number of counter cycles at a specific point in time, counted from where the execution started. This option is only available for Cortex-M devices.

Any accesses of the specified type will activate the trace data collection.

Match data
Enables matching of the accessed data. Use the Match data options in combination with the Read/Write, Read, or Write access types for data. This option can be useful when you want a trigger when a variable has a certain value.

Value Specify a data value.
Mask Specify which part of the value to match (word, halfword, or byte).

The Match data options are only available for J-Link/J-Trace and when using a Cortex-M device.

Note: For Cortex-M devices, only one breakpoint with Match data can be set. Such a breakpoint uses two breakpoint resources.

Link condition
Specifies how trace conditions are combined, using AND and OR. When combining a condition that has the link condition AND with a condition that has the link condition OR, AND has precedence. The option Inverse inverts the trace condition and is individual for each trace filter condition. If one trace start or stop condition is inverted, all others will be too. An inverted trace start or stop condition means that the trace data collection is performed everywhere except for this section of the application code.

For ARM7/9 devices, trace filters are combined using the OR algorithm. Use the Inverse option to invert the trace filter; all trace filters are affected. The trace filter will be combined with the start and stop triggers, if any, using the AND algorithm.
Trace Stop breakpoints dialog box (J-Link/J-Trace)

The Trace Stop dialog box is available from the context menu that appears when you right-click in the Breakpoints window. You can also right-click in the editor window or the Disassembly window, and then choose Toggle Breakpoint (Trace Stop).

![Trace Stop breakpoints dialog box (J-Link/J-Trace)](image)

Figure 99: Trace Stop breakpoints dialog box (J-Link/J-Trace)

When the trace condition is triggered, the trace data collection is performed for some further instructions, and then the collection is stopped.

Requirements

The C-SPY J-Link/J-Trace driver.

Trigger at

Specify the stopping point of the code section for which you want to collect trace data. You can specify a variable name, an address, or a cycle counter value.

Size

Controls the size of the address range, that when reached, will trigger the stop of the trace data collection. Choose between:

Auto

Sets the size automatically. This can be useful if Trigger at contains a variable.

Manual

Specify the size of the breakpoint range manually.
**Trigger range**

Shows the requested range and the effective range to be covered by the trace data collection. The range suggested is either within or exactly the area specified by the Trigger at and the Size options.

**Extend to cover requested range**

Extends the range so that a whole data structure is covered. For data structures that do not fit the size of the possible ranges supplied by the hardware breakpoint unit, for example three bytes, the range will not cover the whole data structure. Note that the range will be extended beyond the size of the data structure, which might cause false triggers at adjacent data.

This option is not enabled for ARM7/9 devices because the range for such devices will always cover the whole data structure.

**Access Type**

Selects the type of memory access that triggers the breakpoint:

- **Read/Write**
  - Reads from or writes to location.

- **Read**
  - Reads from location.

- **Write**
  - Writes to location.

- **OP-fetch**
  - Accesses at execution address.

- **Cycle**
  - The number of counter cycles at a specific point in time, counted from where the execution started. This option is only available for Cortex-M devices.

Any accesses of the specified type will stop the trace data collection.

**Match data**

Enables matching of the accessed data. Use the Match data options in combination with the Read/Write, Read, or Write access types for data. This option can be useful when you want a trigger when a variable has a certain value.

- **Value**
  - Specify a data value.
The Match data options are only available for J-Link/J-Trace and when using a Cortex-M device.

**Note:** For Cortex-M devices, only one breakpoint with Match data can be set. Such a breakpoint uses two breakpoint resources.

**Link condition**

Specifies how trace conditions are combined, using AND and OR. When combining a condition that has the link condition AND with a condition that has the link condition OR, AND has precedence. The option Inverse inverts the trace condition and is individual for each trace filter condition. If one trace start or stop condition is inverted, all others will be too. An inverted trace start or stop condition means that the trace data collection is performed everywhere except for this section of the application code.

For ARM7/9 devices, trace filters are combined using the OR algorithm. Use the Inverse option to invert the trace filter; all trace filters are affected. The trace filter will be combined with the start and stop triggers, if any, using the AND algorithm.

**Trace Filter breakpoints dialog box (J-Link/J-Trace)**

The Trace Filter dialog box is available from the context menu that appears when you right-click in the Breakpoints window. You can also right-click in the editor window or the Disassembly window, and then choose Toggle Breakpoint (Trace Filter).
When the trace condition is triggered, the trace data collection is performed for some further instructions, and then the collection is stopped.

**Requirements**

The C-SPY J-Link/J-Trace driver.

**Trigger at**

Specify the code location of the breakpoint. Alternatively, click the **Edit** button to open the **Enter Location** dialog box, see **Enter Location dialog box**, page 37.

**Size**

Controls the size of the address range where filtered trace is active. Choose between:

- **Auto**
  
  Sets the size automatically. This can be useful if **Trigger at** contains a variable.

- **Manual**
  
  Specify the size of the breakpoint range manually.

**Trigger range**

Shows the requested range and the effective range to be covered by the trace data collection. The range suggested is either within or exactly the area specified by the **Trigger at** and the **Size** options.

- **Extend to cover requested range**

  Extends the range so that a whole data structure is covered. For data structures that do not fit the size of the possible ranges supplied by the hardware breakpoint unit, for example three bytes, the range will not cover the whole data structure. Note that the range will be extended beyond the size of the data structure, which might cause false triggers at adjacent data.

  This option is not enabled for ARM7/9 devices because the range for such devices will always cover the whole data structure.

**Access Type**

Selects the type of memory access that triggers the breakpoint:

- **Read/Write**
  
  Reads from or writes to location.

- **Read**
  
  Reads from location.
Write
Writes to location.

OP-fetch
Accesses at execution address.

Cycle
The number of counter cycles at a specific point in time, counted from where the execution started. This option is only available for Cortex-M devices.

Match data
Enables matching of the accessed data. Use the Match data options in combination with the Read/Write, Read, or Write access types for data. This option can be useful when you want a trigger when a variable has a certain value.

Value
Specify a data value.

Mask
Specify which part of the value to match (word, halfword, or byte).

The Match data options are only available for J-Link/J-Trace and when using a Cortex-M device.

Note: For Cortex-M devices, only one breakpoint with Match data can be set. Such a breakpoint uses two breakpoint resources.

Link condition
Specifies how trace conditions are combined, using AND and OR. When combining a condition that has the link condition AND with a condition that has the link condition OR, AND has precedence. The option Inverse inverts the trace condition and is individual for each trace filter condition. If one trace start or stop condition is inverted, all others will be too. An inverted trace start or stop condition means that the trace data collection is performed everywhere except for this section of the application code.

For ARM7/9 devices, trace filters are combined using the OR algorithm. Use the Inverse option to invert the trace filter; all trace filters are affected. The trace filter will be combined with the start and stop triggers, if any, using the AND algorithm.
Trace Expressions window

The Trace Expressions window is available from the Trace window toolbar.

![Figure 101: Trace Expressions window](image)

Use this window to specify, for example, a specific variable (or an expression) for which you want to collect trace data.

Requirements

The C-SPY simulator.

Toolbar

The toolbar buttons change the order between the expressions:

Arrow up

Moves the selected row up.

Arrow down

Moves the selected row down.

Display area

Use the display area to specify expressions for which you want to collect trace data:

Expression

Specify any expression that you want to collect data from. You can specify any expression that can be evaluated, such as variables and registers.

Format

Shows which display format that is used for each expression. Note that you can change display format via the context menu.

Each row in this area will appear as an extra column in the Trace window.
Find in Trace dialog box

The Find in Trace dialog box is available by clicking the Find button on the Trace window toolbar or by choosing Edit>Find and Replace>Find.

Note that the Edit>Find and Replace>Find command is context-dependent. It displays the Find in Trace dialog box if the Trace window is the current window or the Find dialog box if the editor window is the current window.

![Find in Trace dialog box](image)

Use this dialog box to specify the search criteria for advanced searches in the trace data.

The search results are displayed in the Find in Trace window—available by choosing the View>Messages command, see Find in Trace window, page 237.

See also Searching in trace data, page 193.

Requirements

One of these alternatives:

- The C-SPY Simulator.
- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY ST-LINK driver.

Text search

Specify the string you want to search for. To specify the search criteria, choose between:

Match Case

Searches only for occurrences that exactly match the case of the specified text. Otherwise int will also find Int and Int and so on.
**Match whole word**

Searches only for the string when it occurs as a separate word. Otherwise `int` will also find `printf`, `sprintf` and so on.

**Only search in one column**

Searches only in the column you selected from the drop-down list.

**Address Range**

Specify the address range you want to display or search. The trace data within the address range is displayed. If you also have specified a text string in the **Text search** field, the text string is searched for within the address range.

**Find in Trace window**

The Find in Trace window is available from the View>Messages menu. Alternatively, it is automatically displayed when you perform a search using the **Find in Trace** dialog box or perform a search using the **Find in Trace** command available from the context menu in the editor window.

![Figure 103: Find in Trace window](image)

This window displays the result of searches in the trace data. Double-click an item in the Find in Trace window to bring up the same item in the Trace window.

Before you can view any trace data, you must specify the search criteria in the **Find in Trace** dialog box, see **Find in Trace dialog box**, page 236.

For more information, see **Searching in trace data**, page 193.

**Requirements**

One of these alternatives:

- The C-SPY Simulator.
- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
The C-SPY ST-LINK driver.

Display area

The Find in Trace window looks like the Trace window and shows the same columns and data, but only those rows that match the specified search criteria.

Trace Save dialog box

The Trace Save dialog box is available from the driver-specific menu, and from the Trace window and the SWO Trace window.

Requirements

One of these alternatives:

- The C-SPY J-Link/J-Trace driver
- The C-SPY ST-LINK driver

Index Range

Saves a range of frames to a file. Specify a start index and an end index (as numbered in the index column in the Trace window).

Append to file

Appends the trace data to an existing file.

Use tab-separated format

Saves the content in columns that are tab-separated, instead of separated by white spaces.

File

Specify a file for the trace data.
Profiling

This chapter describes how to use the profiler in C-SPY®. More specifically, this means:

- Introduction to the profiler
- Using the profiler
- Reference information on the profiler.

Introduction to the profiler

This section introduces the profiler.

These topics are covered:

- Reasons for using the profiler
- Briefly about the profiler
- Requirements for using the profiler.

REASONS FOR USING THE PROFILER

Function profiling can help you find the functions in your source code where the most time is spent during execution. You should focus on those functions when optimizing your code. A simple method of optimizing a function is to compile it using speed optimization. Alternatively, you can move the data used by the function into more efficient memory. For detailed information about efficient memory usage, see the IAR C/C++ Development Guide for ARM.

Alternatively, you can use filtered profiling, which means that you can exclude, for example, individual functions from being profiled. To profile only a specific part of your code, you can select a time interval—using the Timeline window—for which C-SPY produces profiling information.

Instruction profiling can help you fine-tune your code on a very detailed level, especially for assembler source code. Instruction profiling can also help you to understand where your compiled C/C++ source code spends most of its time, and perhaps give insight into how to rewrite it for better performance.
BRIEFLY ABOUT THE PROFILER

Function profiling information is displayed in the Function Profiler window, that is, timing information for the functions in an application. Profiling must be turned on explicitly using a button on the window’s toolbar, and will stay enabled until it is turned off.

Instruction profiling information is displayed in the Disassembly window, that is, the number of times each instruction has been executed.

Profiling sources

The profiler can use different mechanisms, or sources, to collect profiling information. Depending on the available trace source features, one or more of the sources can be used for profiling:

● Trace (calls)
  The full instruction trace (ETM trace) is analyzed to determine all function calls and returns. When the collected instruction sequence is incomplete or discontinuous, as sometimes happens when using ETM trace, the profiling information is less accurate.

● Trace (flat) / Sampling
  Each instruction in the full instruction trace (ETM trace) or each PC Sample (from SWO trace) is assigned to a corresponding function or code fragment, without regard to function calls or returns. This is most useful when the application does not exhibit normal call/return sequences, such as when you are using an RTOS, or when you are profiling code which does not have full debug information.

● Breakpoints
  The profiler sets a breakpoint on every function entry point. During execution, the profiler collects information about function calls and returns as each breakpoint is hit. This assumes that the hardware supports a large number of breakpoints, and it has a huge impact on execution performance.

Power sampling

Some debug probes support sampling of the power consumption of the development board, or components on the board. Each sample is associated with a PC sample and represents the power consumption (actually, the electrical current) for a small time interval preceding the time of the sample. When the profiler is set to use Power Sampling, additional columns are displayed in the Profiler window. Each power sample is associated with a function or code fragment, just as with regular PC Sampling. Note that this does not imply that all the energy corresponding to a sample can be attributed to that function or code fragment. The time scales of power samples and instruction execution are vastly different; during one power measurement, the CPU has typically executed many thousands of instructions. Power Sampling is a statistics tool.
REQUIREMENTS FOR USING THE PROFILER

The C-SPY simulator supports the profiler; there are no specific requirements.

To use the profiler in your hardware debugger system, you need

- An I-jet in-circuit debugging probe, a JTAGjet, a J-Link, a J-Trace, ST-LINK debug probe with an SWD/SWO interface between the probe and the target system, which must be based on a Cortex-M device
- A J-Trace debug probe and an ARM7/9 device with ETM trace.
- A J-Link or J-Trace Ultra probe.

<table>
<thead>
<tr>
<th>Target system</th>
<th>Trace (calls)</th>
<th>Trace (flat)</th>
<th>Sampling</th>
<th>Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-SPY simulator</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I-jet/JTAGjet</td>
<td>--</td>
<td>--</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>J-Link</td>
<td>--</td>
<td>--</td>
<td>X\textsuperscript{*}</td>
<td>--</td>
</tr>
<tr>
<td>J-Link Ultra</td>
<td>--</td>
<td>--</td>
<td>X\textsuperscript{*}</td>
<td>X</td>
</tr>
<tr>
<td>J-Trace</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RDI</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Macraigor</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GDB Server</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td>--</td>
<td>--</td>
<td>X\textsuperscript{*}</td>
<td>--</td>
</tr>
<tr>
<td>TI Stellaris</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TI XDS100</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Angel</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
<tr>
<td>IAR ROM-monitor</td>
<td>--</td>
<td>--</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 10: C-SPY driver profiling support

* Only for Cortex-M devices.

Using the profiler

This section describes various tasks related to using the profiler.

More specifically, you will get information about:

- Getting started using the profiler on function level
- Getting started using the profiler on instruction level
- Selecting a time interval for profiling information
**GETTING STARTED USING THE PROFILER ON FUNCTION LEVEL**

To display function profiling information in the Function Profiler window:

1. Build your application using these options:

<table>
<thead>
<tr>
<th>Category</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++ Compiler</td>
<td>Output&gt;Generate debug information</td>
</tr>
<tr>
<td>Linker</td>
<td>Output&gt;Include debug information in output</td>
</tr>
</tbody>
</table>

Table 11: Project options for enabling the profiler

2. To set up the profiler for function profiling:
   - If you use ETM trace, make sure that the **Cycle accurate tracing** option is selected in the **Trace Settings** dialog box.
   - If you use the SWD/SWO interface, no specific settings are required.

3. When you have built your application and started C-SPY, choose **Driver-menu>Function Profiler** to open the Function Profiler window, and click the **Enable** button to turn on the profiler. Alternatively, choose **Enable** from the context menu that is available when you right-click in the Function Profiler window.

4. Start executing your application to collect the profiling information.

5. Profiling information is displayed in the Function Profiler window. To sort, click on the relevant column header.

6. When you start a new sampling, you can click the **Clear** button—alternatively, use the context menu—to clear the data.

**GETTING STARTED USING THE PROFILER ON INSTRUCTION LEVEL**

To display instruction profiling information in the Disassembly window:

1. When you have built your application and started C-SPY, choose **View>Disassembly** to open the Disassembly window, and choose **Instruction Profiling>Enable** from the context menu that is available when you right-click in the left-hand margin of the Disassembly window.

2. Make sure that the **Show** command on the context menu is selected, to display the profiling information.

3. Start executing your application to collect the profiling information.
When the execution stops, for instance because the program exit is reached or a breakpoint is triggered, you can view instruction level profiling information in the left-hand margin of the window.

![Figure 105: Instruction count in Disassembly window](image)

For each instruction, the number of times it has been executed is displayed.

Instruction profiling attempts to use the same source as the function profiler. If the function profiler is not on, the instruction profiler will try to use first trace and then PC sampling as source. You can change the source to be used from the context menu that is available in the Function Profiler window.

**SELECTING A TIME INTERVAL FOR PROFILING INFORMATION**

Normally, the profiler computes its information from all PC samples it receives, accumulating more and more information until you explicitly clear the profiling information. However, you can choose a time interval for which the profiler computes the PC samples. This function is supported by the I-jet in-circuit debugging probe, the JTAGjet debug probe, the J-Link probe, the J-Trace probe and the ST-LINK probe.

To select a time interval:

1. Choose **Function Profiler** from the C-SPY driver menu.
2. In the Function Profiler window, right-click and choose **Source: Sampling** from the context menu.
3. Execute your application to collect samples.
4. Choose **View>Timeline**.
Reference information on the profiler

5. In the Timeline window, click and drag to select a time interval.

![A selected time interval](image)

In the selected time interval, right-click and choose **Profile Selection** from the context menu.

The Function Profiler window now displays profiling information for the selected time interval.

![Function Profiler window in time-interval mode](image)

6. Click the **Full/Time-interval profiling** button to toggle the Full profiling view.

Reference information on the profiler

This section gives reference information about these windows and dialog boxes:

- **Function Profiler window**, page 245
- **Disassembly window**, page 80
See also:
- ETM Trace Settings dialog box (J-Link/J-Trace), page 197
- ETM Trace Settings dialog box (I-jet/JTAGjet), page 195
- SWO Trace Window Settings dialog box, page 199
- SWO Configuration dialog box, page 201

Function Profiler window

The Function Profiler window is available from the C-SPY driver menu.

![Function Profiler window](image)

This window displays function profiling information.

When Trace(flat) is selected, a checkbox appears on each line in the left-side margin of the window. Use these checkboxes to include or exclude lines from the profiling. Excluded lines are dimmed but not removed.

Requirements

One of these alternatives:
- The C-SPY simulator
- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY ST-LINK driver.

Toolbar

The toolbar contains:

Enable/Disable

Enables or disables the profiler.
Clear
Clears all profiling data.

Save
Opens a standard Save As dialog box where you can save the contents of the window to a file, with tab-separated columns. Only non-expanded rows are included in the list file.

Graphical view
Overlays the values in the percentage columns with a graphical bar.

Progress bar
Displays a backlog of profiling data that is still being processed. If the rate of incoming data is higher than the rate of the profiler processing the data, a backlog is accumulated. The progress bar indicates that the profiler is still processing data, but also approximately how far the profiler has come in the process. Note that because the profiler consumes data at a certain rate and the target system supplies data at another rate, the amount of data remaining to be processed can both increase and decrease. The progress bar can grow and shrink accordingly.

Time-interval mode
Toggles between profiling a selected time interval or full profiling. This toolbar button is only available if PC Sampling is supported by the debug probe.

For information about which views that are supported in the C-SPY driver you are using, see Requirements for using the profiler, page 241.

Status field
Displays the range of the selected time interval, in other words, the profiled selection. This field is yellow when Time-interval profiling mode is enabled. This field is only available if PC Sampling is supported by the debug probe (SWO trace).

For information about which views that are supported in the C-SPY driver you are using, see Requirements for using the profiler, page 241.

Display area
The content in the display area depends on which source that is used for the profiling information:

- For the Breakpoints and Trace (calls) sources, the display area contains one line for each function compiled with debug information enabled. When some profiling information has been collected, it is possible to expand rows of functions that have
called other functions. The child items for a given function list all the functions that have been called by the parent function and the corresponding statistics.

- For the Sampling and Trace (flat) sources, the display area contains one line for each C function of your application, but also lines for sections of code from the runtime library or from other code without debug information, denoted only by the corresponding assembler labels. Each executed PC address from trace data is treated as a separate sample and is associated with the corresponding line in the Profiling window. Each line contains a count of those samples.

For information about which views that are supported in the C-SPY driver you are using, see Requirements for using the profiler, page 241.

More specifically, the display area provides information in these columns:

**Function (All sources)**
- The name of the profiled C function.
  - For Sampling source, also sections of code from the runtime library or from other code without debug information, denoted only by the corresponding assembler labels, is displayed.

**Calls (Breakpoints and Trace (calls))**
- The number of times the function has been called.

**Flat time (Breakpoints and Trace (calls))**
- The time in cycles spent inside the function.

**Flat time (%) (Breakpoints and Trace (calls))**
- Flat time expressed as a percentage of the total time.

**Acc. time (Breakpoint and Trace (calls))**
- The time in cycles spent in this function and everything called by this function.

**Acc. time (%) (Breakpoints and Trace (calls))**
- Accumulated time expressed as a percentage of the total time.

**PC Samples (Trace (flat) and Sampling)**
- The number of PC samples associated with the function.

**PC Samples (%) (Trace (flat) and Sampling)**
- The number of PC samples associated with the function as a percentage of the total number of samples.

**Power Samples (Power Sampling)**
- The number of power samples associated with that function.
Reference information on the profiler

Energy (%) (Power Sampling)
The accumulated value of all measurements associated with that function, expressed as a percentage of all measurements.

Avg Current [mA] (Power Sampling)
The average measured value for all samples associated with that function.

Min Current [mA] (Power Sampling)
The minimum measured value for all samples associated with that function.

Max Current [mA] (Power Sampling)
The maximum measured value for all samples associated with that function.

Context menu
This context menu is available:

Enable
Enables the profiler. The system will collect information also when the window is closed.

Clear
Clears all profiling data.

Filtering
Selects which part of your code to profile. Choose between:

Check All—Excludes all lines from the profiling.
Uncheck All—Includes all lines in the profiling.
Load—Reads all excluded lines from a saved file.
Save—Saves all excluded lines to a file. Typically, this can be useful if you are a group of engineers and want to share sets of exclusions.
These commands are only available when using one of the modes Trace(flat) or Sampling.

**Source**

Selects which source to be used for the profiling information. Choose between:

- **Sampling**—the instruction count for instruction profiling represents the number of samples for each instruction.

- **Trace (calls)**—the instruction count for instruction profiling is only as complete as the collected trace data.

- **Trace (flat)**—the instruction count for instruction profiling is only as complete as the collected trace data.

**Power Sampling**

Toggles power sampling information on or off. This command is supported by the I-jet in-circuit debugging probe, the JTAGjet, the J-Link, and the J-Trace Ultra debug probes.

* The available sources depend on the C-SPY driver you are using.

For information about which views that are supported in the C-SPY driver you are using, see *Requirements for using the profiler*, page 241.
Reference information on the profiler
Power debugging

This chapter describes techniques for power debugging and how you can use C-SPY® to find source code constructions that result in unexpected power consumption. More specifically, this means:

- Introduction to power debugging
- Optimizing your source code for power consumption
- Debugging in the power domain
- Reference information on power debugging.

Introduction to power debugging

This section covers these topics:

- Reasons for using power debugging
- Briefly about power debugging
- Requirements for power debugging.

Click the arrows below to display more information.

REASONS FOR USING POWER DEBUGGING

Long battery lifetime is a very important factor for many embedded systems in almost any market segment: medical, consumer electronics, home automation, etc. The power consumption in these systems does not only depend on the hardware design, but also on how the hardware is used. The system software controls how it is used.

For examples of when power debugging can be useful, see Optimizing your source code for power consumption, page 253.

BRIEFLY ABOUT POWER DEBUGGING

Power debugging is based on the ability to sample the power consumption—more precisely, the power being consumed by the CPU and the peripheral units—and correlate each sample with the application’s instruction sequence and hence with the source code and various events in the program execution.
Traditionally, the main software design goal has been to use as little memory as possible. However, by correlating your application’s power consumption with its source code you can get insight into how the software affects the power consumption, and thus how it can be minimized.

**Measuring power consumption**

The debug probe measures the voltage drop across a small resistor in series with the supply power to the device. The voltage drop is measured by a differential amplifier and then sampled by an AD converter.

You can specify a threshold and an appropriate action to be executed when the threshold value is reached. This means that you can enable or disable the power measurement or you can stop the application’s execution and determine the cause of unexpected power values.

**Power debugging using C-SPY**

C-SPY provides an interface for configuring your power debugging and a set of windows for viewing the power values:

- The Power Setup window is where you can specify a threshold and an action to be executed when the threshold is reached.
- The Power Log window displays all logged power values. This window can be used for finding peaks in the power logging and because the values are correlated with the executed code, you can double-click on a value in the Power Log window to get the corresponding code. The precision depends on the frequency of the samples, but there is a good chance that you find the source code sequence that caused the peak.
- The Timeline window displays power values on a time scale. This provides a convenient way of viewing the power consumption in relation to the other information displayed in the window. The Timeline window is correlated to both the Power Log window, the source code window, and the Disassembly window, which means you are just a double-click away from the source code that corresponds to the values you see on the timeline.
- The Function Profiler window combines the function profiling with the power logging to display the power consumption per function—power profiling. You will get a list of values per function and also the average values together with max and min values. Thus, you will find the regions in the application that you should focus when optimizing for power consumption.

**REQUIREMENTS FOR POWER DEBUGGING**

To use the features in C-SPY for power debugging, you need:

- A Cortex-M3 device with SWO
Optimizing your source code for power consumption

This section gives some examples where power debugging can be useful and thus hopefully help you identify source code constructions that can be optimized for low power consumption.

WAITING FOR DEVICE STATUS

One common construction that could cause unnecessary power consumption is to use a poll loop for waiting for a status change of, for example a peripheral device. Constructions like this example execute without interruption until the status value changes into the expected state.

```c
while (USBD_GetState() < USBD_STATE_CONFIGURED);
while ((BASE_PMC->PMC_SR & PMC_MCKRDY) != PMC_MCKRDY);
```

To minimize power consumption, rewrite polling of a device status change to use interrupts if possible, or a timer interrupt so that the CPU can sleep between the polls.

SOFTWARE DELAYS

A software delay might be implemented as a `for` or `while` loop like for example:

```c
i = 10000; /* A software delay */
do i--;
while (i != 0);
```

Such software delays will keep the CPU busy with executing instructions performing nothing except to make the time go by. Time delays are much better implemented using a hardware timer. The timer interrupt is set up and after that, the CPU goes down into a low power mode until it is awakened by the interrupt.

DMA VERSUS POLLED I/O

DMA has traditionally been used for increasing transfer speed. For MCUs there are plenty of DMA techniques to increase flexibility, speed, and to lower power consumption. Sometimes, CPUs can even be put into sleep mode during the DMA transfer. Power debugging lets you experiment and see directly in the debugger what effects these DMA techniques will have on power consumption compared to a traditional CPU-driven polled solution.
LOW-POWER MODE DIAGNOSTICS

Many embedded applications spend most of their time waiting for something to happen: receiving data on a serial port, watching an I/O pin change state, or waiting for a time delay to expire. If the processor is still running at full speed when it is idle, battery life is consumed while very little is being accomplished. So in many applications, the microprocessor is only active during a very small amount of the total time, and by placing it in a low-power mode during the idle time, the battery life can be extended considerably.

A good approach is to have a task-oriented design and to use an RTOS. In a task-oriented design, a task can be defined with the lowest priority, and it will only execute when there is no other task that needs to be executed. This idle task is the perfect place to implement power management. In practice, every time the idle task is activated, it sets the microprocessor into a low-power mode. Many microprocessors and other silicon devices have a number of different low-power modes, in which different parts of the microprocessor can be turned off when they are not needed. The oscillator can for example either be turned off or switched to a lower frequency. In addition, individual peripheral units, timers, and the CPU can be stopped. The different low-power modes have different power consumption based on which peripherals are left turned on. A power debugging tool can be very useful when experimenting with different low-level modes.

You can use the Function profiler in C-SPY to compare power measurements for the task or function that sets the system in a low-power mode when different low-power modes are used. Both the mean value and the percentage of the total power consumption can be useful in the comparison.

CPU FREQUENCY

Power consumption in a CMOS MCU is theoretically given by the formula:

\[ P = f \cdot U^2 \cdot k \]

where \( f \) is the clock frequency, \( U \) is the supply voltage, and \( k \) is a constant.

Power debugging lets you verify the power consumption as a factor of the clock frequency. A system that spends very little time in sleep mode at 50 MHz is expected to spend 50% of the time in sleep mode when running at 100 MHz. You can use the power data collected in C-SPY to verify the expected behavior and if there is nonlinear dependency on the clock frequency, make sure to choose the operating frequency that gives the lowest power consumption.

DETECTING MISTAKENLY UNATTENDED PERIPHERALS

Peripheral units can consume much power even when they are not actively in use. If you are designing for low power, it is important that you disable the peripheral units and not
just leave them unattended when they are not in use. But for different reasons, a peripheral unit can be left with its power supply on; it can be a careful and correct design decision, or it can be an inadequate design or just a mistake. If not the first case, then more power than expected will be consumed by your application. This will be easily revealed by the Power graph in the Timeline window. Double-clicking in the Timeline window where the power consumption is unexpectedly high will take you to the corresponding source code and disassembly code. In many cases, it is enough to disable the peripheral unit when it is inactive, for example by turning off its clock which in most cases will shut down its power consumption completely.

However, there are some cases where clock gating will not be enough. Analog peripherals like converters or comparators can consume a substantial amount of power even when the clock is turned off. The Timeline window will reveal that turning off the clock was not enough and that you need to turn off the peripheral completely.

PERIPHERAL UNITS IN AN EVENT-DRIVEN SYSTEM

Consider a system where one task uses an analog comparator while executing, but the task is suspended by a higher-priority task. Ideally, the comparator should be turned off when the task is suspended and then turned on again once the task is resumed. This would minimize the power being consumed during the execution of the high-priority task.

This is a schematic diagram of the power consumption of an assumed event-driven system where the system at the point of time $t_0$ is in an inactive mode and the current is $I_0$.

![Figure 110: Power consumption in an event-driven system](image-url)
At $t_1$, the system is activated whereby the current rises to $I_1$ which is the system’s power consumption in active mode when at least one peripheral device turned on, causing the current to rise to $I_1$. At $t_2$, the execution becomes suspended by an interrupt which is handled with high priority. Peripheral devices that were already active are not turned off, although the task with higher priority is not using them. Instead, more peripheral devices are activated by the new task, resulting in an increased current $I_2$ between $t_2$ and $t_3$ where control is handed back to the task with lower priority.

The functionality of the system could be excellent and it can be optimized in terms of speed and code size. But also in the power domain, more optimizations can be made. The shadowed area represents the energy that could have been saved if the peripheral devices that are not used between $t_2$ and $t_3$ had been turned off, or if the priorities of the two tasks had been changed.

If you use the Timeline window, you can make a closer examination and identify that unused peripheral devices were activated and consumed power for a longer period than necessary. Naturally, you must consider whether it is worth it to spend extra clock cycles to turn peripheral devices on and off in a situation like in the example.

**FINDING CONFLICTING HARDWARE SETUPS**

To avoid floating inputs, it is a common design practice to connect unused MCU I/O pins to ground. If your source code by mistake configures one of the grounded I/O pins as a logical 1 output, a high current might be drained on that pin. This high unexpected current is easily observed by reading the current value from the Power Graph in the Timeline window. It is also possible to find the corresponding erratic initialization code by looking at the Power Graph at application startup.

A similar situation arises if an I/O pin is designed to be an input and is driven by an external circuit, but your code incorrectly configures the input pin as output.

**ANALOG INTERFERENCE**

When mixing analog and digital circuits on the same board, the board layout and routing can affect the analog noise levels. To ensure accurate sampling of low-level analog signals, it is important to keep noise levels low. Obtaining a well-mixed signal design
Power debugging requires careful hardware considerations. Your software design can also affect the quality of the analog measurements.

Performing a lot of I/O activity at the same time as sampling analog signals causes many digital lines to toggle state at the same time, which might introduce extra noise into the AD converter.

![Noise spike recorded by an oscilloscope](image)

Power debugging will help you investigate interference from digital and power supply lines into the analog parts. Power spikes in the vicinity of AD conversions could be the source of noise and should be investigated. All data presented in the Timeline window is correlated to the executed code. Simply double-clicking on a suspicious power value will bring up the corresponding C source code.

**Debugging in the power domain**

This section describes various tasks related to power debugging.

More specifically, you will get information about:

- Displaying the application’s power profile and analyzing the result
- Detecting unexpected power usage during application execution.

See also:

- *Timeline window*, page 209
- *Selecting a time interval for profiling information*, page 243.
DISPLAYING THE APPLICATION’S POWER PROFILE AND ANALYZING THE RESULT

To view the power profile:

1. Choose driver-menu>SWO Configuration to open the SWO Configuration dialog box. Make sure the CPU clock option is set to the same value as the CPU clock value set by your application. This is necessary to set the SWO clock and to obtain a correct data transfer to the debug probe.

   If you are using the C-SPY simulator, you can ignore this step.

2. Choose C-SPY driver>Timeline to open the Timeline window.

3. Right-click in the graph area and choose Enable from the context menu to enable the Power graph.

4. Choose C-SPY driver>Power Log to open the Power Log window.

5. Optionally, if you want to correlate power values to specific interrupts or variables, right-click in the Interrupts or Data Logs graph area, respectively, and choose Enable from the context menu.

   For variables, you also need to set a Data Log breakpoint for each variable you want a graphical representation of in the Timeline window. See Data Log breakpoints dialog box, page 143.

6. Optionally, before you start executing your application you can configure the viewing range of the Y-axis for the graph. See Viewing Range dialog box, page 218.

7. Click Go on the toolbar to start executing your application. In the Power Log window, all power values are displayed. In the Timeline window, you will see a graphical representation of the power values, and of the data and interrupt logs if you enabled these graphs. For information about how to navigate on the graph, see Timeline window, page 209.

8. To analyze power consumption:
   - Double-click on an interesting power value to highlight the corresponding source code is highlighted in the editor window and in the Disassembly window. The corresponding log is highlighted in the Power Log window. For examples of when this can be useful, see Optimizing your source code for power consumption, page 253.
   - You can identify peripheral units that can be disabled when not used. You can detect this by analyzing the Power graph in combination with the other graphs in the Timeline window. See also Detecting mistakenly unattended peripherals, page 254.
● For a specific interrupt, you can see whether the power consumption is changed in an unexpected way after the interrupt exits, for example, if the interrupt enables a power-intensive unit and does not turn it off before exit.

● For function profiling, see Selecting a time interval for profiling information, page 243.

DETECTING UNEXPECTED POWER USAGE DURING APPLICATION EXECUTION

To detect unexpected power consumption:

1. Choose driver-menu>SWO Configuration to open the SWO Configuration dialog box. Make sure these settings are used:
   ● CPU clock must be set to the same value as the CPU clock value set by your application. This is necessary to set the SWO clock and to obtain a correct data transfer to the debug probe.

2. Choose driver-menu>Power Setup to open the Power Setup window.

3. In the Power Setup window, specify a threshold value and the appropriate action, for example Log All and Halt CPU Above Threshold.

4. Choose driver-menu>Power Log to open the Power Log window. If you continuously want to save the power values to a file, choose Choose Live Log File from the context menu. In this case you also need to choose Enable Live Logging to.

5. Start the execution.
   When the power consumption passes the threshold value, the execution will stop and perform the action you specified.
   If you saved your logged power values to a file, you can open that file in an external tool for further analysis.

Reference information on power debugging

This section gives reference information for windows and dialog boxes related to power debugging:

● Power Log Setup window, page 260
● Power Log window, page 262.

See also:

● Trace window, page 205
● Timeline window, page 209
Reference information on power debugging

- Viewing Range dialog box, page 218
- Function Profiler window, page 245.

Power Log Setup window

The Power Log Setup window is available from the C-SPY driver menu during a debug session.

![Power Log Setup window](image)

**Figure 112: Power Log Setup window**

Use this window to configure the power measurement.

**Note:** To enable power logging, choose **Enable** from the context menu in the Power Log window or from the context menu in the Power Log Graph in the Timeline window.

**Requirements**

One of these alternatives:

- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver

**Display area**

This area contains these columns:

**ID**

A unique string that identifies the measurement channel in the probe. Use the check box to activate the channel. If the check box is deselected, logs will not be generated for that channel.

**Name**

Specify a user-defined name.

**Shunt [Ohm]**

This column always contains `--`. 
Threshold
Specify a threshold value in the selected unit. The action you specify will be executed when the threshold value is reached.

Unit
Selects the unit for power display. Choose between: nA, uA, or mA.

Action
Displays which action has been selected for the measurement channel. Choose between: Log All, Log Above Threshold, Log Below Threshold, Log All and Halt CPU Above Threshold, and Log All and Halt CPU Below Threshold.

Context menu
This context menu is available:

<table>
<thead>
<tr>
<th>nA</th>
<th>uA</th>
<th>mA</th>
</tr>
</thead>
</table>

Log All
Logs all values.

Log Above Threshold
Logs all values above the threshold.

Log Below Threshold
Logs all values below the threshold.

Log All and Halt CPU Above Threshold
Logs all values. If a logged value is above the threshold, execution is stopped.

Log All and Halt CPU Below Threshold
Logs all values. If a logged value is below the threshold, execution is stopped.

Figure 113: Power Setup window context menu

These commands are available:

nA, uA, mA
Selects the unit for the power display. These alternatives are available for channels that measure current.

Log All
Logs all values.
Reference information on power debugging

Power Log window

The Power Log window is available from the C-SPY driver menu during a debug session.

![Power Log window](image)

This window displays collected power values.

A row with only the Time/Cycles and Program Counter displayed in grey denotes a logged power value for a channel that was active during the actual collection of data but currently is disabled in the Power Setup window.

**Note:** There is a limit in the number of logged power values. When this limit is exceeded, the entries in the beginning of the buffer are erased.

**Requirements**

One of these alternatives:

- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver

**Display area**

This area contains these columns:

**Time**

The time from the application reset until the event, based on the clock frequency specified in the **SWO Configuration** dialog box.

If the time is displayed in italics, the target system could not collect a correct time, but instead had to approximate it.

This column is available when you have selected **Show Time** from the context menu.

**Cycles**

The number of cycles from the application reset until the event. This information is cleared at reset.
If a cycle is displayed in italics, the target system could not collect a correct time, but instead had to approximate it.

This column is available when you have selected Show Cycles from the context menu.

**Program Counter**

Displays one of these:

An address, which is the content of the PC, that is, the address of an instruction in close to the point where the power value was collected.

---, the target system failed to provide the debugger with any information.

Overflow in red, the communication channel failed to transmit all data from the target system.

Idle, the power value is logged during idle mode.

**Name [unit]**

The power measurement value expressed in the unit you specified in the Power Setup window.

**Context menu**

This context menu is available:

- Enable
- Clear
- Save To Log File...

---

These commands are available:

**Enable**

Enables the logging system, which means power values are saved internally within the IDE. The values are displayed in the Power Log window and in the Power Graph in the Timeline window (if enabled). The system will log information also when the window is closed.

**Clear**

Clears the power values saved internally within the IDE. The values will also be cleared when you reset the debugger, or if you change the execution frequency in the SWO Setup dialog box.
Save to Log File
Displays a standard file selection dialog box where you can choose the destination file for the logged power values. This command then saves the current content of the internal log buffer.

Choose Live Log File
Displays a standard file selection dialog box where you can choose a destination file for the logged power values. The power values are continuously saved to that file during execution. The content of the live log file is never automatically cleared, the logged values are simply appended at the end of the file.

Enable Live Logging to
Toggles live logging on or off. The logs are saved in a to the specified file.

Clear log file
Clears the content of the live log file.

Show Time
Displays the Time column in the Power Log window. This choice is also reflected in the log files.

Show Cycles
Displays the Cycles column in the Power Log window. This choice is also reflected in the log files.

The format of the log file
The log file has a tab-separated format. The entries in the log file are separated by TAB and line feed. The logged power values are displayed in these columns:

- **Time/Cycles**
The time from the application reset until the power value was logged.

- **Approx**
An × in the column indicates that the power value has an approximative value for time/cycle.

- **PC**
The value of the program counter close to the point where the power value was logged.

- **Name[unit]**
The corresponding value from the Power Log window, where Name and unit are according to your settings in the Power Setup window.
Code coverage

This chapter describes the code coverage functionality in C-SPY®, which helps you verify whether all parts of your code have been executed. More specifically, this means:

- Introduction to code coverage
- Reference information on code coverage.

Introduction to code coverage

This section covers these topics:

- Reasons for using code coverage
- Briefly about code coverage
- Requirements for using code coverage.

REASONS FOR USING CODE COVERAGE

The code coverage functionality is useful when you design your test procedure to verify whether all parts of the code have been executed. It also helps you identify parts of your code that are not reachable.

BRIEFLY ABOUT CODE COVERAGE

The Code Coverage window reports the status of the current code coverage analysis. For every program, module, and function, the analysis shows the percentage of code that has been executed since code coverage was turned on up to the point where the application has stopped. In addition, all statements that have not been executed are listed. The analysis will continue until turned off.

REQUIREMENTS FOR USING CODE COVERAGE

For information about the driver you are using, see Differences between the C-SPY drivers, page 42.
Reference information on code coverage

This section gives reference information about these windows and dialog boxes:

- Code Coverage window, page 266.

See also Single stepping, page 74.

Code Coverage window

The Code Coverage window is available from the View menu.

![Figure 116: Code Coverage window](image)

This window reports the status of the current code coverage analysis. For every program, module, and function, the analysis shows the percentage of code that has been executed since code coverage was turned on up to the point where the application has stopped. In addition, all statements that have not been executed are listed. The analysis will continue until turned off.

An asterisk (*) in the title bar indicates that C-SPY has continued to execute, and that the Code Coverage window must be refreshed because the displayed information is no longer up to date. To update the information, use the Refresh command.

To get started using code coverage:

Before using the code coverage functionality you must build your application using these options:

<table>
<thead>
<tr>
<th>Category</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>C/C++ Compiler</td>
<td>Output&gt;Generate debug information</td>
</tr>
<tr>
<td>Linker</td>
<td>Output&gt;Include debug information in output</td>
</tr>
</tbody>
</table>

Table 12: Project options for enabling code coverage
After you have built your application and started C-SPY, choose View>Code Coverage to open the Code Coverage window.

3 Click the Activate button, alternatively choose Activate from the context menu, to switch on code coverage.

4 Start the execution. When the execution stops, for instance because the program exit is reached or a breakpoint is triggered, click the Refresh button to view the code coverage information.

Requirements

One of these alternatives:
- The C-SPY Simulator.
- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY ST-LINK driver.

Display area

The code coverage information is displayed in a tree structure, showing the program, module, function, and statement levels. The window displays only source code that was compiled with debug information. Thus, startup code, exit code, and library code is not displayed in the window. Furthermore, coverage information for statements in inlined functions is not displayed. Only the statement containing the inlined function call is marked as executed. The plus sign and minus sign icons allow you to expand and collapse the structure.
These icons give you an overview of the current status on all levels:

Red diamond  
Signifies that 0% of the modules or functions has been executed.

Green diamond  
Signifies that 100% of the modules or functions has been executed.

Red and green diamond  
Signifies that some of the modules or functions have been executed.

Yellow diamond  
Signifies a statement that has not been executed.

The percentage displayed at the end of every program, module, and function line shows the amount of statements that has been covered so far, that is, the number of executed statements divided with the total number of statements.

For statements that have not been executed (yellow diamond), the information displayed is the column number range and the row number of the statement in the source window, followed by the address of the step point:

<column_start>-<column_end>:row address.

A statement is considered to be executed when one of its instructions has been executed. When a statement has been executed, it is removed from the window and the percentage is increased correspondingly.

Double-clicking a statement or a function in the Code Coverage window displays that statement or function as the current position in the source window, which becomes the active window. Double-clicking a module on the program level expands or collapses the tree structure.

Context menu

This context menu is available:

![Context menu]

*Activate*

Switches code coverage on and off during execution.
Clear
Clears the code coverage information. All step points are marked as not executed.

Refresh
Updates the code coverage information and refreshes the window. All step points that have been executed since the last refresh are removed from the tree.

Auto-refresh
Toggles the automatic reload of code coverage information on and off. When turned on, the code coverage information is reloaded automatically when C-SPY stops at a breakpoint, at a step point, and at program exit.

Save As
Saves the current code coverage result in a text file.

Save session
Saves your code coverage session data to a *.dat file. This is useful if you for some reason must abort your debug session, but want to continue the session later on. This command is available on the toolbar.

Restore session
Restores previously saved code coverage session data. This is useful if you for some reason must abort your debug session, but want to continue the session later on. This command is available on the toolbar.
Reference information on code coverage
Interrupts

This chapter describes how C-SPY® can help you test the logic of your interrupt service routines and debug the interrupt handling in the target system. Interrupt logging provides you with comprehensive information about the interrupt events. More specifically, this chapter gives:

- Introduction to interrupts
- Using the interrupt system
- Reference information on interrupts.

Introduction to interrupts

This section introduces you to interrupt logging and to interrupt simulation. This section covers these topics:

- Briefly about interrupt logging
- Briefly about the interrupt simulation system
- Interrupt characteristics
- Interrupt simulation states
- C-SPY system macros for interrupt simulation
- Target-adapting the interrupt simulation system.

See also:

- Reference information on C-SPY system macros, page 307
- Breakpoints, page 119
- The IAR C/C++ Development Guide for ARM.

BRIEFLY ABOUT INTERRUPT LOGGING

Interrupt logging provides you with comprehensive information about the interrupt events. This might be useful for example, to help you locate which interrupts you can fine-tune to become faster. You can log entrances and exits to and from interrupts. If you are using the C-SPY simulator, you can also log internal interrupt status information, such as triggered, expired, etc. The logs are displayed in the Interrupt Log window and a summary is available in the Interrupt Log Summary window. The Interrupt Graph in
the Timeline window provides a graphical view of the interrupt events during the execution of your application program.

Requirements for interrupt logging

To use interrupt logging you need:

- An I-jet in-circuit debugging probe or a JTAGjet debug probe, and an SWD interface between the debug probe and the target system
- A J-Link or J-Trace debug probe and an SWD interface between the debug probe and the target system
- An ST-LINK debug probe and an SWD interface between the debug probe and the target system
- To enable interrupt logging from the Interrupt Log window, the Interrupt Log Summary window, or the Timeline window

Interrupt logging is supported by the C-SPY simulator.

BRIEFLY ABOUT THE INTERRUPT SIMULATION SYSTEM

By simulating interrupts, you can test the logic of your interrupt service routines and debug the interrupt handling in the target system long before any hardware is available. If you use simulated interrupts in conjunction with C-SPY macros and breakpoints, you can compose a complex simulation of, for instance, interrupt-driven peripheral devices.

The C-SPY Simulator includes an interrupt simulation system where you can simulate the execution of interrupts during debugging. You can configure the interrupt simulation system so that it resembles your hardware interrupt system.

The interrupt system has the following features:

- Simulated interrupt support for the ARM core
- Single-occasion or periodical interrupts based on the cycle counter
- Predefined interrupts for various devices
- Configuration of hold time, probability, and timing variation
- State information for locating timing problems
- Configuration of interrupts using a dialog box or a C-SPY system macro—that is, one interactive and one automating interface. In addition, you can instantly force an interrupt.
- A log window that continuously displays events for each defined interrupt.
- A status window that shows the current interrupt activities.
All interrupts you define using the **Interrupt Setup** dialog box are preserved between debug sessions, unless you remove them. A forced interrupt, on the other hand, exists only until it has been serviced and is not preserved between sessions.

The interrupt simulation system is activated by default, but if not required, you can turn off the interrupt simulation system to speed up the simulation. To turn it off, use either the **Interrupt Setup** dialog box or a system macro.

### INTERRUPT CHARACTERISTICS

The simulated interrupts consist of a set of characteristics which lets you fine-tune each interrupt to make it resemble the real interrupt on your target hardware. You can specify a **first activation time**, a **repeat interval**, a **hold time**, a **variance**, and a **probability**.

![Figure 118: Simulated interrupt configuration](image)

*If probability is less than 100%, some interrupts may be omitted.*

- **A** = Activation time
- **R** = Repeat interval
- **H** = Hold time
- **V** = Variance

The interrupt simulation system uses the cycle counter as a clock to determine when an interrupt should be raised in the simulator. You specify the **first activation time**, which is based on the cycle counter. C-SPY will generate an interrupt when the cycle counter has passed the specified activation time. However, interrupts can only be raised between instructions, which means that a full assembler instruction must have been executed before the interrupt is generated, regardless of how many cycles an instruction takes.

To define the periodicity of the interrupt generation you can specify the **repeat interval** which defines the amount of cycles after which a new interrupt should be generated. In addition to the repeat interval, the periodicity depends on the two options **probability**—the probability, in percent, that the interrupt will actually appear in a period—and **variance**—a time variation range as a percentage of the repeat interval.

These options make it possible to randomize the interrupt simulation. You can also specify a **hold time** which describes how long the interrupt remains pending until removed if it has not been processed. If the hold time is set to **infinite**, the corresponding pending bit will be set until the interrupt is acknowledged or removed.
INTERRUPT SIMULATION STATES

The interrupt simulation system contains status information that you can use for locating timing problems in your application. The Interrupt Status window displays the available status information. For an interrupt, these states can be displayed: Idle, Pending, Executing, or Suspended.

Normally, a repeatable interrupt has a specified repeat interval that is longer than the execution time. In this case, the status information at different times looks like this:

![Diagram of interrupt simulation states](image)

*Figure 119: Simulation states - example 1*

**Note:** The interrupt activation signal—also known as the pending bit—is automatically deactivated the moment the interrupt is acknowledged by the interrupt handler.
However, if the interrupt repeat interval is shorter than the execution time, and the interrupt is reentrant (or non-maskable), the status information at different times looks like this:

![Diagram of interrupt states](image)

An execution time that is longer than the repeat interval might indicate that you should rewrite your interrupt handler and make it faster, or that you should specify a longer repeat interval for the interrupt simulation system.

**C-SPY SYSTEM MACROS FOR INTERRUPT SIMULATION**

Macros are useful when you already have sorted out the details of the simulated interrupt so that it fully meets your requirements. If you write a macro function containing definitions for the simulated interrupts, you can execute the functions automatically when C-SPY starts. Another advantage is that your simulated interrupt definitions will be documented if you use macro files, and if you are several engineers involved in the development project you can share the macro files within the group.

The C-SPY Simulator provides these predefined system macros related to interrupts:

```c
__enableInterrupts
__disableInterrupts
__orderInterrupt
__cancelInterrupt
__cancelAllInterrupts
```
Using the interrupt system

This section describes various tasks related to interrupts.

More specifically, you will get information about:

● Simulating a simple interrupt
● Simulating an interrupt in a multi-task system
● Getting started using interrupt logging.

See also:

● Registering and executing using setup macros and setup files, page 298 for details about how to use a setup file to define simulated interrupts at C-SPY startup
● The tutorial Simulating an interrupt in the Information Center.

SIMULATING A SIMPLE INTERRUPT

This example demonstrates the method for simulating a timer interrupt for OKI ML674001. However, the procedure can also be used for other types of interrupts.
To simulate and debug an interrupt:

1. Assume this simple application which contains an IRQ handler routine that handles system timer interrupts. It increments a tick variable. The main function sets the necessary status registers. The application exits when 100 interrupts have been generated.

```c
/* Enables use of extended keywords */
#pragma language=extended

#include <intrinsics.h>
#include <oki/ioml674001.h>
#include <stdio.h>

unsigned int ticks = 0;

/* IRQ handler */
__irq __arm void IRQ_Handler(void)
{
    /* We use only system timer interrupts, so we do not need
to check the interrupt source. */
ticks += 1;
    TMOVFR_bit.OVF = 1; /* Clear system timer overflow flag */
}

int main( void )
{
    enable_interrupt();
    /* Timer setup code */
    ILC0_bit.ILR0 = 4; /* System timer interrupt priority */
    TMRLR_bit.TMRLR = 1E5; /* System timer reload value */
    TMEN_bit.TCEN = 1; /* Enable system timer */
    while (ticks < 100);
    printf("Done\n");
}
```

2. Add your interrupt service routine to your application source code and add the file to your project.

3. Build your project and start the simulator.
4 Choose Simulator>Interrupt Setup to open the Interrupts Setup dialog box. Select the Enable interrupt simulation option to enable interrupt simulation. Click New to open the Edit Interrupt dialog box. For the Timer example, verify these settings:

<table>
<thead>
<tr>
<th>Option</th>
<th>Settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt</td>
<td>IRQ</td>
</tr>
<tr>
<td>First activation</td>
<td>4000</td>
</tr>
<tr>
<td>Repeat interval</td>
<td>2000</td>
</tr>
<tr>
<td>Hold time</td>
<td>10</td>
</tr>
<tr>
<td>Probability (%)</td>
<td>100</td>
</tr>
<tr>
<td>Variance (%)</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 13: Timer interrupt settings

Click OK.

5 Execute your application. If you have enabled the interrupt properly in your application source code, C-SPY will:

- Generate an interrupt when the cycle counter has passed 4000
- Continuously repeat the interrupt after approximately 2000 cycles.

6 To watch the interrupt in action, choose Simulator>Interrupt Log to open the Interrupt Log window.

7 From the context menu, available in the Interrupt Log window, choose Enable to enable the logging. If you restart program execution, status information about entrances and exits to and from interrupts will now appear in the Interrupt Log window.

For information about how to get a graphical representation of the interrupts correlated with a time axis, see Timeline window, page 209.

SIMULATING AN INTERRUPT IN A MULTI-TASK SYSTEM

If you are using interrupts in such a way that the normal instruction used for returning from an interrupt handler is not used, for example in an operating system with task-switching, the simulator cannot automatically detect that the interrupt has finished executing. The interrupt simulation system will work correctly, but the status information in the Interrupt Setup dialog box might not look as you expect. If too many interrupts are executing simultaneously, a warning might be issued.

To simulate a normal interrupt exit:

1 Set a code breakpoint on the instruction that returns from the interrupt function.

2 Specify the __popSimulatorInterruptExecutingStack macro as a condition to the breakpoint.
When the breakpoint is triggered, the macro is executed and then the application continues to execute automatically.

**GETTING STARTED USING INTERRUPT LOGGING**

1. Choose **C-SPY driver>Interrupt Log** to open the Interrupt Log window. Optionally, you can also choose:
   - **C-SPY driver>Interrupt Log Summary** to open the Interrupt Log Summary window
   - **C-SPY driver>Timeline** to open the Timeline window and view the Interrupt graph.

2. From the context menu in the Interrupt Log window, choose **Enable** to enable the logging.
   
   In the **SWO Configuration** dialog box, you can see in the **Interrupt Log Events** area that interrupt logs are enabled.

3. Start executing your application program to collect the log information.

4. To view the interrupt log information, look in any of the Interrupt Log, Interrupt Log Summary, or the Interrupt graph in the Timeline window.

5. If you want to save the log or summary to a file, choose **Save to log file** from the context menu in the window in question.

6. To disable interrupt logging, from the context menu in the Interrupt Log window, toggle **Enable** off.

**Reference information on interrupts**

This section gives reference information about these windows and dialog boxes:

- **Interrupt Setup dialog box**, page 280
- **Edit Interrupt dialog box**, page 282
- **Forced Interrupt window**, page 284
- **Interrupt Status window**, page 285
- **Interrupt Log window**, page 287
- **Interrupt Log Summary window**, page 291.
Interrupt Setup dialog box

The Interrupt Setup dialog box is available by choosing Simulator>Interrupt Setup.

![Interrupt Setup dialog box]

This dialog box lists all defined interrupts. Use this dialog box to enable or disable the interrupt simulation system, as well as to enable or disable individual interrupts.

Requirements

The C-SPY simulator.

Enable interrupt simulation

Enables or disables interrupt simulation. If the interrupt simulation is disabled, the definitions remain but no interrupts are generated. Note that you can also enable and disable installed interrupts individually by using the check box to the left of the interrupt name in the list of installed interrupts.

Display area

This area contains these columns:

Interrupt

Lists all interrupts. Use the checkbox to enable or disable the interrupt.

ID

A unique interrupt identifier.

Type

Shows the type of the interrupt. The type can be one of:

Forced, a single-occasion interrupt defined in the Forced Interrupt Window.
Interrupts

Single, a single-occasion interrupt.
Repeat, a periodically occurring interrupt.

If the interrupt has been set from a C-SPY macro, the additional part \((\text{macro})\) is added, for example: \(\text{Repeat(}\text{macro})\).

Timing

The timing of the interrupt. For a Single and Forced interrupt, the activation time is displayed. For a Repeat interrupt, the information has the form: \(\text{Activation Time} + n\times\text{Repeat Time}\). For example, \(2000 + n\times2345\). This means that the first time this interrupt is triggered, is at 2000 cycles and after that with an interval of 2345 cycles.

Buttons

These buttons are available:

New
Opens the Edit Interrupt dialog box, see Edit Interrupt dialog box, page 282.

Edit
Opens the Edit Interrupt dialog box, see Edit Interrupt dialog box, page 282.

Delete
Removes the selected interrupt.

Delete All
Removes all interrupts.
Edit Interrupt dialog box

The Edit Interrupt dialog box is available from the Interrupt Setup dialog box.

![Edit Interrupt dialog box](image)

Use this dialog box to interactively fine-tune the interrupt parameters. You can add the parameters and quickly test that the interrupt is generated according to your needs.

**Note:** You can only edit or remove non-forced interrupts.

**Requirements**

The C-SPY simulator.

**Interrupt**

Selects the interrupt that you want to edit. The drop-down list contains all available interrupts. Your selection will automatically update the Description box. The list is, for Cortex-M devices, populated with entries from the device description file that you have selected. For other devices, only two interrupts are available: IRQ and FIQ.

**Description**

A description of the selected interrupt, if available. The description is retrieved from the selected device description file and consists of a string describing the priority, vector offset, enable bit, and pending bit, separated by space characters. The enable bit and pending bit are optional. It is possible to have none, only the enable bit, or both. For interrupts specified using the system macro `__orderInterrupt`, the Description box is empty.

For Cortex-M devices, the description is retrieved from the selected device description file and is editable. Enable bit and pending bit are not available from the ddf file; they must be manually edited if wanted. The priority is as in the hardware: the lower the number, the higher the priority. NMI and HardFault are special, and their descriptions...
Interrupts should not be edited. Cortex-M interrupts are also affected by the PRIMASK, FAULTMASK, and BASEPRI registers, as described in the ARM documentation.

For other devices, the description strings for IRQ and FIQ are hardcoded and cannot be edited. In those descriptions, a higher priority number means a higher priority.

**First activation**
Specify the value of the cycle counter after which the specified type of interrupt will be generated.

**Repeat interval**
Specify the periodicity of the interrupt in cycles.

**Variance %**
Selects a timing variation range, as a percentage of the repeat interval, in which the interrupt might occur for a period. For example, if the repeat interval is 100 and the variance 5%, the interrupt might occur anywhere between $T=95$ and $T=105$, to simulate a variation in the timing.

**Hold time**
Specify how long, in cycles, the interrupt remains pending until removed if it has not been processed. If you select *Infinite*, the corresponding pending bit will be set until the interrupt is acknowledged or removed.

**Probability %**
Selects the probability, in percent, that the interrupt will actually occur within the specified period.
Forced Interrupt window

The Forced Interrupt window is available from the C-SPY driver menu.

![Forced Interrupt window](image)

Use this window to force an interrupt instantly. This is useful when you want to check your interrupt logistics and interrupt routines.

The hold time for a forced interrupt is infinite, and the interrupt exists until it has been serviced or until a reset of the debug session.

**To force an interrupt:**

1. Enable the interrupt simulation system, see *Interrupt Setup dialog box*, page 280.
2. Double-click the interrupt in the Forced Interrupt window, or activate by using the **Force** command available on the context menu.

**Requirements**

The C-SPY simulator.

**Display area**

This area lists all available interrupts and their definitions. This information is retrieved from the selected device description file. See this file for a detailed description.

**Context menu**

This context menu is available:

![Context menu](image)

**Figure 124: Forced Interrupt window context menu**

This command is available:

**Force**

Triggers the interrupt you selected in the display area.
Interrupt Status window

The Interrupt Status window is available from the C-SPY driver menu.

![Interrupt Status window](image)

This window shows the status of all the currently active interrupts, in other words interrupts that are either executing or waiting to be executed.

Requirements

The C-SPY simulator.

Display area

This area contains these columns:

**Interrupt**

Lists all interrupts.

**ID**

A unique interrupt identifier.

**Type**

The type of the interrupt. The type can be one of:

- **Forced**, a single-occasion interrupt defined in the Forced Interrupt window.
- **Single**, a single-occasion interrupt.
- **Repeat**, a periodically occurring interrupt.

If the interrupt has been set from a C-SPY macro, the additional part (macro) is added, for example: Repeat(macro).

**Status**

The state of the interrupt:

- **Idle**, the interrupt activation signal is low (deactivated).
Pending, the interrupt activation signal is active, but the interrupt has not been yet acknowledged by the interrupt handler.

Executing, the interrupt is currently being serviced, that is the interrupt handler function is executing.

Suspended, the interrupt is currently suspended due to execution of an interrupt with a higher priority.

(deleted) is added to Executing and Suspended if you have deleted a currently active interrupt. (deleted) is removed when the interrupt has finished executing.

Next Time
The next time an idle interrupt is triggered. Once a repeatable interrupt stats executing, a copy of the interrupt will appear with the state Idle and the next time set. For interrupts that do not have a next time—that is pending, executing, or suspended—the column will show --.

Timing
The timing of the interrupt. For a Single and Forced interrupt, the activation time is displayed. For a Repeat interrupt, the information has the form: Activation Time + n*Repeat Time. For example, 2000 + n*2345. This means that the first time this interrupt is triggered, is at 2000 cycles and after that with an interval of 2345 cycles.
Interrupt Log window

The Interrupt Log window is available from the C-SPY driver menu.

![Figure 126: Interrupt Log window](image)

This window logs entrances to and exits from interrupts. The C-SPY simulator also logs internal state changes.

The information is useful for debugging the interrupt handling in the target system. When the Interrupt Log window is open, it is updated continuously at runtime.

**Note:** There is a limit on the number of saved logs. When this limit is exceeded, the entries in the beginning of the buffer are erased.

For more information, see *Getting started using interrupt logging*, page 279.

For information about how to get a graphical view of the interrupt events during the execution of your application, see *Timeline window*, page 209.

**Requirements**

One of these alternatives:

- The C-SPY simulator
- An I-jet in-circuit debugging probe or a JTAGjet debug probe, and an SWD interface between the debug probe and the target system
- A J-Link or J-Trace debug probe with an SWD interface between the debug probe and the target system
● An ST-LINK debug probe with an SWD interface between the debug probe and the target system.

**Display area for the C-SPY hardware drivers**

This area contains these columns:

- **Time**
  
  The time for the interrupt entrance, based on the CPU clock frequency specified in the **SWO Configuration** dialog box.
  
  If a time is displayed in italics, the target system has not been able to collect a correct time, but instead had to approximate it.
  
  This column is available when you have selected **Show Time** from the context menu.

- **Cycles**
  
  The number of cycles from the start of the execution until the event.
  
  A cycle count displayed in italics indicates an approximative value. Italics is used when the target system has not been able to collect a correct value, but instead had to approximate it.
  
  This column is available when you have selected **Show Cycles** from the context menu.

- **Interrupt**
  
  The name of the interrupt source where the occurred. If the column displays **Overflow** in red, the communication channel failed to transmit all interrupt logs from the target system.

- **Status**
  
  The event status of the interrupt:
  
  - **Enter**, the interrupt is currently executing.
  
  - **Leave**, the interrupt has finished executing.

- **Program Counter***
  
  The address of the interrupt handler.

- **Execution Time/Cycles**
  
  The time spent in the interrupt, calculated using the Enter and Leave timestamps. This includes time spent in any subroutines or other interrupts that occurred in the specific interrupt.
You can double-click an address. If it is available in the source code, the editor window displays the corresponding source code, for example for the interrupt handler (this does not include library source code).

Display area for the C-SPY simulator

This area contains these columns:

**Time**
The time for the interrupt entrance, based on an internally specified clock frequency.
This column is available when you have selected Show Time from the context menu.

**Cycles**
The number of cycles from the start of the execution until the event.
This column is available when you have selected Show Cycles from the context menu.

**Interrupt**
The interrupt as defined in the device description file.

**Status**
Shows the event status of the interrupt:
- Triggered, the interrupt has passed its activation time.
- Forced, the same as Triggered, but the interrupt was forced from the Forced Interrupt window.
- Enter, the interrupt is currently executing.
- Leave, the interrupt has been executed.
- Expired, the interrupt hold time has expired without the interrupt being executed.
- Rejected, the interrupt has been rejected because the necessary interrupt registers were not set up to accept the interrupt.

**Program Counter**
The value of the program counter when the event occurred.

**Execution Time/Cycles**
The time spent in the interrupt, calculated using the Enter and Leave timestamps. This includes time spent in any subroutines or other interrupts that occurred in the specific interrupt.
Interrupt Log window context menu

This context menu is available in the Interrupt Log window and in the Interrupt Log Summary window:

- **Enable**
  Enables the logging system. The system will log information also when the window is closed.

- **Clear**
  Deletes the log information. Note that this will happen also when you reset the debugger.

- **Save to log file**
  Displays a standard file selection dialog box where you can select the destination file for the log information. The entries in the log file are separated by TAB and LF. An X in the Approx column indicates that the timestamp is an approximation.

- **Show Time**
  Displays the Time column in the Data Log window and in the Interrupt Log window, respectively.

- **Show Cycles**
  Displays the Cycles column in the Data Log window and in the Interrupt Log window, respectively.

Note: The commands are the same in each window, but they only operate on the specific window.

These commands are available:

**Enable**

Enables the logging system. The system will log information also when the window is closed.

**Clear**

Deletes the log information. Note that this will happen also when you reset the debugger.

**Save to log file**

Displays a standard file selection dialog box where you can select the destination file for the log information. The entries in the log file are separated by TAB and LF. An X in the Approx column indicates that the timestamp is an approximation.

**Show Time**

Displays the Time column in the Data Log window and in the Interrupt Log window, respectively.

**Show Cycles**

Displays the Cycles column in the Data Log window and in the Interrupt Log window, respectively.
Interrupt Log Summary window

The Interrupt Log Summary window is available from the C-SPY driver menu.

![Interrupt Log Summary window](image)

This window displays a summary of logs of entrances to and exits from interrupts.

For more information, see *Getting started using interrupt logging*, page 279.

For information about how to get a graphical view of the interrupt events during the execution of your application, see *Timeline window*, page 209.

**Requirements**

One of these alternatives:

- The C-SPY simulator
- An I-jet in-circuit debugging probe or a JTAGjet debug probe, and an SWD interface between the debug probe and the target system
- A J-Link or J-Trace debug probe with an SWD interface between the debug probe and the target system
- An ST-LINK debug probe with an SWD interface between the debug probe and the target system.

**Display area for the C-SPY simulator**

Each row in this area displays statistics about the specific interrupt based on the log information in these columns:

- **Interrupt**
  - The type of interrupt that occurred.
  - At the bottom of the column, the current time or cycles is displayed—the number of cycles or the execution time since the start of execution. Overflow count and approximative time count is always zero.

- **Count**
  - The number of times the interrupt occurred.
**First time**
The first time the interrupt was executed.

**Total time**
The accumulated time spent in the interrupt.

**Fastest**
The fastest execution of a single interrupt of this type.

**Slowest**
The slowest execution of a single interrupt of this type.

**Max interval**
The longest time between two interrupts of this type.
The interval is specified as the time interval between the entry time for two consecutive interrupts.

** Calculated in the same way as for the Execution time/cycles in the Interrupt Log window.

**Context menu**
Identical to the context menu of the Interrupt Log window, see *Interrupt Log window*, page 287.
C-SPY macros

C-SPY® includes a comprehensive macro language which allows you to automate the debugging process and to simulate peripheral devices.

This chapter describes the C-SPY macro language, its features, for what purpose these features can be used, and how to use them. More specifically, this means:

- Introduction to C-SPY macros
- Using C-SPY macros
- Reference information on the macro language
- Reference information on reserved setup macro function names
- Reference information on C-SPY system macros.

Introduction to C-SPY macros

This section covers these topics:

- Reasons for using C-SPY macros
- Briefly about using C-SPY macros
- Briefly about setup macro functions and files
- Briefly about the macro language.

REASONS FOR USING C-SPY MACROS

You can use C-SPY macros either by themselves or in conjunction with complex breakpoints and interrupt simulation to perform a wide variety of tasks. Some examples where macros can be useful:

- Automating the debug session, for instance with trace printouts, printing values of variables, and setting breakpoints.
- Hardware configuring, such as initializing hardware registers.
- Feeding your application with simulated data during runtime.
Introduction to C-SPY macros

- Simulating peripheral devices, see the chapter Interrupts. This only applies if you are using the simulator driver.
- Developing small debug utility functions, for instance calculating the stack depth, see the provided example stack.mac located in the directory \arm\src\.

**BRIEFLY ABOUT USING C-SPY MACROS**

To use C-SPY macros, you should:

- Write your macro variables and functions and collect them in one or several macro files
- Register your macros
- Execute your macros.

For registering and executing macros, there are several methods to choose between. Which method you choose depends on which level of interaction or automation you want, and depending on at which stage you want to register or execute your macro.

**BRIEFLY ABOUT SETUP MACRO FUNCTIONS AND FILES**

There are some reserved setup macro function names that you can use for defining macro functions which will be called at specific times, such as:

- Once after communication with the target system has been established but before downloading the application software
- Once after your application software has been downloaded
- Each time the reset command is issued
- Once when the debug session ends.

To define a macro function to be called at a specific stage, you should define and register a macro function with one of the reserved names. For instance, if you want to clear a specific memory area before you load your application software, the macro setup function execUserPreload should be used. This function is also suitable if you want to initialize some CPU registers or memory-mapped peripheral units before you load your application software.

You should define these functions in a setup macro file, which you can load before C-SPY starts. Your macro functions will then be automatically registered each time you start C-SPY. This is convenient if you want to automate the initialization of C-SPY, or if you want to register multiple setup macros.

For more information about each setup macro function, see Reference information on reserved setup macro function names, page 306.
Remapping memory
A common feature of many ARM-based processors is the ability to remap memory. After a reset, the memory controller typically maps address zero to non-volatile memory, such as Flash. By configuring the memory controller, the system memory can be remapped to place RAM at zero and non-volatile memory higher up in the address map. By doing this, the exception table will reside in RAM and can be easily modified when you download code to the target hardware. To handle this in C-SPY, the setup macro function `execUserPreload()` is suitable. For an example, see Remapping memory, page 58.

BRIEFLY ABOUT THE MACRO LANGUAGE
The syntax of the macro language is very similar to the C language. There are:
- **Macro statements**, which are similar to C statements.
- **Macro functions**, which you can define with or without parameters and return values.
- Predefined built-in **system macros**, similar to C library functions, which perform useful tasks such as opening and closing files, setting breakpoints, and defining simulated interrupts.
- **Macro variables**, which can be global or local, and can be used in C-SPY expressions.
- **Macro strings**, which you can manipulate using predefined system macros.

For more information about the macro language components, see Reference information on the macro language, page 301.

Example
Consider this example of a macro function which illustrates the various components of the macro language:

```c
#define var oldVal;
CheckLatest(val)
{
  if (oldval != val)
  {
    __message "Message: Changed from ", oldval, " to ", val, ":\n";
    oldval = val;
  }
}
```

**Note:** Reserved macro words begin with double underscores to prevent name conflicts.
Using C-SPY macros

This section describes various tasks related to registering and executing C-SPY macros. More specifically, you will get information about:

- Registering C-SPY macros—an overview
- Executing C-SPY macros—an overview
- Registering and executing using setup macros and setup files
- Registering and executing using setup macros and setup files
- Executing macros using Quick Watch
- Executing a macro by connecting it to a breakpoint

For more examples using C-SPY macros, see:

- The tutorial about simulating an interrupt, which you can find in the Information Center
- *Initializing target hardware before C-SPY starts*, page 58.

**REGISTERING C-SPY MACROS—AN OVERVIEW**

C-SPY must know that you intend to use your defined macro functions, and thus you must register your macros. There are various ways to register macro functions:

- You can register macros interactively in the Macro Configuration dialog box, see *Using the Macro Configuration dialog box*, page 297.
- You can register macro functions during the C-SPY startup sequence, see *Registering and executing using setup macros and setup files*, page 298.
- You can register a file containing macro function definitions, using the system macro `__registerMacroFile`. This means that you can dynamically select which macro files to register, depending on the runtime conditions. Using the system macro also lets you register multiple files at the same moment. For information about the system macro, see `__registerMacroFile`, page 332.

Which method you choose depends on which level of interaction or automation you want, and depending on at which stage you want to register your macro.

**EXECUTING C-SPY MACROS—AN OVERVIEW**

There are various ways to execute macro functions:

- You can execute macro functions during the C-SPY startup sequence and at other predefined stages during the debug session by defining setup macro functions in a setup macro file, see *Registering and executing using setup macros and setup files*, page 298.
The Quick Watch window lets you evaluate expressions, and can thus be used for executing macro functions. For an example, see Executing macros using Quick Watch, page 299.

The Macro Quicklaunch window is similar to the Quick Watch window, but is more specified on designed for C-SPY macros. See Macro Quicklaunch window, page 106.

A macro can be connected to a breakpoint; when the breakpoint is triggered the macro is executed. For an example, see Executing a macro by connecting it to a breakpoint, page 300.

Which method you choose depends on which level of interaction or automation you want, and depending on at which stage you want to execute your macro.

**USING THE MACRO CONFIGURATION DIALOG BOX**

The Macro Configuration dialog box is available by choosing Debug > Macros.
Use this dialog box to list, register, and edit your macro files and functions. The dialog box offers you an interactive interface for registering your macro functions which is convenient when you develop macro functions and continuously want to load and test them.

Macro functions that have been registered using the dialog box are deactivated when you exit the debug session, and will not automatically be registered at the next debug session.

**To register a macro file:**

1. Select the macro files you want to register in the file selection list, and click Add or Add All to add them to the Selected Macro Files list. Conversely, you can remove files from the Selected Macro Files list using Remove or Remove All.

2. Click Register to register the macro functions, replacing any previously defined macro functions or variables. Registered macro functions are displayed in the scroll list under Registered Macros.

   *Note:* System macros cannot be removed from the list, they are always registered.

**To list macro functions:**

1. Select All to display all macro functions, select User to display all user-defined macros, or select System to display all system macros.

2. Click either Name or File under Registered Macros to display the column contents sorted by macro names or by file. Clicking a second time sorts the contents in the reverse order.

**To modify a macro file:**

Double-click a user-defined macro function in the Name column to open the file where the function is defined, allowing you to modify it.

**REGISTERING AND EXECUTING USING SETUP MACROS AND SETUP FILES**

It can be convenient to register a macro file during the C-SPY startup sequence. To do this, specify a macro file which you load before starting the debug session. Your macro functions will be automatically registered each time you start the debugger.

If you use the reserved setup macro function names to define the macro functions, you can define exactly at which stage you want the macro function to be executed.

**To define a setup macro function and load it during C-SPY startup:**

1. Create a new text file where you can define your macro function.
For example:

```c
execUserSetup()
{
    ...
    __registerMacroFile("MyMacroUtils.mac");
    __registerMacroFile("MyDeviceSimulation.mac");
}
```

This macro function registers the additional macro files `MyMacroUtils.mac` and `MyDeviceSimulation.mac`. Because the macro function is defined with the function name `execUserSetup`, it will be executed directly after your application has been downloaded.

2 Save the file using the filename extension `.mac`.

3 Before you start C-SPY, choose `Project>Options>Debugger>Setup`. Select `Use Setup file` and choose the macro file you just created.

The macros will now be registered during the C-SPY startup sequence.

**EXECUTING MACROS USING QUICK WATCH**

The Quick Watch window lets you dynamically choose when to execute a macro function.

1 Consider this simple macro function that checks the status of a timer enable bit:

```c
TimerStatus()
{
    if ((TimerStatreg & 0x01) != 0) /* Checks the status of reg */
        return "Timer enabled" /* C-SPY macro string used */
    else
        return "Timer disabled" /* C-SPY macro string used */
}
```

2 Save the macro function using the filename extension `.mac`.

3 To register the macro file, choose `Debug>Macros`. The `Macro Configuration` dialog box appears.

4 Locate the file, click `Add` and then `Register`. The macro function appears in the list of registered macros.

5 Choose `View>Quick Watch` to open the Quick Watch window. Type the macro call `TimerStatus()` in the text field and press Return.
Alternatively, in the macro file editor window, select the macro function name TimerStatus(). Right-click, and choose Quick Watch from the context menu that appears.

![Quick Watch window](image)

The macro will automatically be displayed in the Quick Watch window.

For more information, see Quick Watch window, page 108.

**EXECUTING A MACRO BY CONNECTING IT TO A BREAKPOINT**

You can connect a macro to a breakpoint. The macro will then be executed when the breakpoint is triggered. The advantage is that you can stop the execution at locations of particular interest and perform specific actions there.

For instance, you can easily produce log reports containing information such as how the values of variables, symbols, or registers change. To do this you might set a breakpoint on a suspicious location and connect a log macro to the breakpoint. After the execution you can study how the values of the registers have changed.

To create a log macro and connect it to a breakpoint:

1. Assume this skeleton of a C function in your application source code:
   
   ```c
   int fact(int x)
   {
   ...
   }
   ```

2. Create a simple log macro function like this example:
   
   ```c
   logfact()
   {
   __message "fact(" ,x, ");
   }
   ```

   The __message statement will log messages to the Log window.

   Save the macro function in a macro file, with the filename extension .mac.
3 To register the macro, choose Debug>Macros to open the Macro Configuration dialog box and add your macro file to the list Selected Macro Files. Click Register and your macro function will appear in the list Registered Macros. Close the dialog box.

4 To set a code breakpoint, click the Toggle Breakpoint button on the first statement within the function fact in your application source code. Choose View>Breakpoints to open the Breakpoints window. Select your breakpoint in the list of breakpoints and choose the Edit command from the context menu.

5 To connect the log macro function to the breakpoint, type the name of the macro function, logfact(), in the Action field and click Apply. Close the dialog box.

6 Execute your application source code. When the breakpoint is triggered, the macro function will be executed. You can see the result in the Log window.

   • Note that the expression in the Action field is evaluated only when the breakpoint causes the execution to really stop. If you want to log a value and then automatically continue execution, you can either:
     Use a Log breakpoint, see Log breakpoints dialog box, page 139
   • Use the Condition field instead of the Action field. For an example, see Performing a task and continuing execution, page 130.

7 You can easily enhance the log macro function by, for instance, using the __fmessage statement instead, which will print the log information to a file. For information about the __fmessage statement, see Formatted output, page 304.

   For an example where a serial port input buffer is simulated using the method of connecting a macro to a breakpoint, see the tutorial Simulating an interrupt in the Information Center.

Reference information on the macro language

This section gives reference information on the macro language:

• Macro functions, page 302
• Macro variables, page 302
• Macro strings, page 303
• Macro statements, page 303
• Formatted output, page 304.
**MACRO FUNCTIONS**

C-SPY macro functions consist of C-SPY variable definitions and macro statements which are executed when the macro is called. An unlimited number of parameters can be passed to a macro function, and macro functions can return a value on exit.

A C-SPY macro has this form:

```
macroName (parameterList)
{
  macroBody
}
```

where `parameterList` is a list of macro parameters separated by commas, and `macroBody` is any series of C-SPY variable definitions and C-SPY statements.

Type checking is neither performed on the values passed to the macro functions nor on the return value.

**MACRO VARIABLES**

A macro variable is a variable defined and allocated outside your application. It can then be used in a C-SPY expression, or you can assign application data—values of the variables in your application—to it. For more information about C-SPY expressions, see *Expressions*, page 303.

The syntax for defining one or more macro variables is:

```
__var nameList;
```

where `nameList` is a list of C-SPY variable names separated by commas.

A macro variable defined outside a macro body has global scope, and it exists throughout the whole debugging session. A macro variable defined within a macro body is created when its definition is executed and destroyed on return from the macro.

By default, macro variables are treated as signed integers and initialized to 0. When a C-SPY variable is assigned a value in an expression, it also acquires the type of that expression. For example:

<table>
<thead>
<tr>
<th>Expression</th>
<th>What it means</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>myvar = 3.5;</code></td>
<td><code>myvar</code> is now type <code>float</code>, value 3.5.</td>
</tr>
<tr>
<td><code>myvar = (int*)i;</code></td>
<td><code>myvar</code> is now type <code>pointer to int</code>, and the value is the same as <code>i</code>.</td>
</tr>
</tbody>
</table>

Table 14: Examples of C-SPY macro variables

In case of a name conflict between a C symbol and a C-SPY macro variable, C-SPY macro variables have a higher precedence than C variables. Note that macro variables are allocated on the debugger host and do not affect your application.
MACRO STRINGS

In addition to C types, macro variables can hold values of *macro strings*. Note that macro strings differ from C language strings.

When you write a string literal, such as "Hello!", in a C-SPY expression, the value is a macro string. It is not a C-style character pointer char*, because char* must point to a sequence of characters in target memory and C-SPY cannot expect any string literal to actually exist in target memory.

You can manipulate a macro string using a few built-in macro functions, for example __strFind or __subString. The result can be a new macro string. You can concatenate macro strings using the + operator, for example str + "tail". You can also access individual characters using subscription, for example str[3]. You can get the length of a string using sizeof(str). Note that a macro string is not NULL-terminated.

The macro function __toString is used for converting from a NULL-terminated C string in your application (char* or char[]) to a macro string. For example, assume this definition of a C string in your application:

```c
char const *cstr = "Hello";
```

Then examine these macro examples:

```c
__var str;         /* A macro variable */
str = cstr         /* str is now just a pointer to char */
sizeof str         /* same as sizeof (char*), typically 2 or 4 */
str = __toString(cstr, 512)  /* str is now a macro string */
sizeof str         /* 5, the length of the string */
str[1]             /* 101, the ASCII code for 'e' */
str += " World!"   /* str is now "Hello World!" */
```

See also Formatted output, page 304.

MACRO STATEMENTS

Statements are expected to behave in the same way as the corresponding C statements would do. The following C-SPY macro statements are accepted:

**Expressions**

```c
expression;
```

For more information about C-SPY expressions, see Expressions, page 303.

**Conditional statements**

```c
if (expression)
    statement
```
if (expression)  
  statement  
else  
  statement

**Loop statements**  
for (init_expression; cond_expression; update_expression)  
  statement
while (expression)  
  statement
  do  
  statement  
  while (expression);

**Return statements**  
return;
return expression;
If the return value is not explicitly set, `signed int 0` is returned by default.

**Blocks**  
Statements can be grouped in blocks.

```
{  
  statement1  
  statement2  
  .  
  .  
  .  
  statementN  
}
```

**FORMATTED OUTPUT**  
C-SPY provides various methods for producing formatted output:

```
__message argList;  
__fmessage file, argList;  
__smessage argList;  
```
Prints the output to the Debug Log window.
Prints the output to the designated file.
Returns a string containing the formatted output.
where \( \text{argList} \) is a comma-separated list of C-SPY expressions or strings, and \( \text{file} \) is the result of the \_\_openFile system macro, see \_\_openFile, page 327.

To produce messages in the Debug Log window:

\[
\begin{align*}
\text{var1} &= 42; \\
\text{var2} &= 37; \\
\_\_\_\_\text{message} \quad &\text{"This line prints the values \(, \text{var1}, \) and \(, \text{var2}, \) in the Log window.";}
\end{align*}
\]

This produces this message in the Log window:

This line prints the values 42 and 37 in the Log window.

To write the output to a designated file:

\[
\_\_\_\_\text{fmessage} \quad \text{myfile}, \quad \text{"Result is \(, \text{res}, \) !\n";}
\]

To produce strings:

\[
\text{myMacroVar} = \_\_\_\_\text{smessage} \text{42}, \quad \text{" is the answer.";}
\]

\text{myMacroVar} now contains the string "42 is the answer.".

**Specifying display format of arguments**

To override the default display format of a scalar argument (number or pointer) in \text{argList}, suffix it with a : followed by a format specifier. Available specifiers are:

- \%b for binary scalar arguments
- \%o for octal scalar arguments
- \%d for decimal scalar arguments
- \%x for hexadecimal scalar arguments
- \%c for character scalar arguments

These match the formats available in the Watch and Locals windows, but number prefixes and quotes around strings and characters are not printed. Another example:

\[
\begin{align*}
\_\_\_\_\_\text{message} \quad &\text{"The character \(', \text{cvar}:\%c, \) has the decimal value \(, \text{cvar};\)}
\end{align*}
\]

Depending on the value of the variables, this produces this message:

The character 'A' has the decimal value 65

**Note:** A character enclosed in single quotes (a character literal) is an integer constant and is not automatically formatted as a character. For example:

\[
\begin{align*}
\_\_\_\_\_\text{message} \quad &\text{" is the numeric value of the character \(, \text{\'A\'}:\%c;\)}
\end{align*}
\]
would produce:

65 is the numeric value of the character A

Note: The default format for certain types is primarily designed to be useful in the Watch window and other related windows. For example, a value of type char is formatted as 'A' (0x41), while a pointer to a character (potentially a C string) is formatted as 0x8102 "Hello", where the string part shows the beginning of the string (currently up to 60 characters).

When printing a value of type char*, use the %x format specifier to print just the pointer value in hexadecimal notation, or use the system macro __toString to get the full string value.

Reference information on reserved setup macro function names

There are reserved setup macro function names that you can use for defining your setup macro functions. By using these reserved names, your function will be executed at defined stages during execution. For more information, see Briefly about setup macro functions and files, page 294.

This table summarizes the reserved setup macro function names:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>execUserPreload</td>
<td>Called after communication with the target system is established but before downloading the target application. Implement this macro to initialize memory locations and/or registers which are vital for loading data properly.</td>
</tr>
<tr>
<td>execUserExecutionStarted</td>
<td>Called when the debugger is about to start or resume execution. The macro is not called when performing a one-instruction assembler step, in other words, Step or Step Into in the Disassembly window. This setup macro is supported by all C-SPY drivers.</td>
</tr>
<tr>
<td>execUserExecutionStopped</td>
<td>Called when the debugger has stopped execution. The macro is not called when performing a one-instruction assembler step, in other words, Step or Step Into in the Disassembly window. This setup macro is supported by all C-SPY debugger drivers.</td>
</tr>
<tr>
<td>execUserFlashInit</td>
<td>Called once before the flash loader is downloaded to RAM. Implement this macro typically for setting up the memory map required by the flash loader. This macro is only called when you are programming flash, and it should only be used for flash loader functionality.</td>
</tr>
</tbody>
</table>

Table 15: C-SPY setup macros
If you define interrupts or breakpoints in a macro file that is executed at system start (using `execUserSetup`) we strongly recommend that you also make sure that they are removed at system shutdown (using `execUserExit`). An example is available in `SetupSimple.mac`, see the tutorials in the Information Center.

The reason for this is that the simulator saves interrupt settings between sessions and if they are not removed they will get duplicated every time `execUserSetup` is executed again. This seriously affects the execution speed.

---

**Reference information on C-SPY system macros**

This section gives reference information about each of the C-SPY system macros.

This table summarizes the pre-defined system macros:

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__cancelAllInterrupts</code></td>
<td>Cancels all ordered interrupts</td>
</tr>
<tr>
<td><code>__cancelInterrupt</code></td>
<td>Cancels an interrupt</td>
</tr>
<tr>
<td><code>__clearBreak</code></td>
<td>Clears a breakpoint</td>
</tr>
<tr>
<td><code>__closeFile</code></td>
<td>Closes a file that was opened by <code>__openFile</code></td>
</tr>
<tr>
<td><code>__delay</code></td>
<td>Delays execution</td>
</tr>
</tbody>
</table>

*Table 16: Summary of system macros*
<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__disableInterrupts</td>
<td>Disables generation of interrupts</td>
</tr>
<tr>
<td>__driverType</td>
<td>Verifies the driver type</td>
</tr>
<tr>
<td>__emulatorSpeed</td>
<td>Sets the emulator clock frequency</td>
</tr>
<tr>
<td>__emulatorStatusCheckOnRead</td>
<td>Enables or disables the verification of the CPSR register after each read operation</td>
</tr>
<tr>
<td>__enableInterrupts</td>
<td>Enables generation of interrupts</td>
</tr>
<tr>
<td>__evaluate</td>
<td>Interprets the input string as an expression and evaluates it</td>
</tr>
<tr>
<td>__gdbserver_exec_command</td>
<td>Send strings or commands to the GDB Server</td>
</tr>
<tr>
<td>__getTracePortSize</td>
<td>Returns the width of the trace port</td>
</tr>
<tr>
<td>__hwJetResetWithStrategy</td>
<td>Performs a hardware reset and a halt of the target CPU</td>
</tr>
<tr>
<td>__hwReset</td>
<td>Performs a hardware reset and a halt of the target CPU</td>
</tr>
<tr>
<td>__hwResetRunToBp</td>
<td>Performs a hardware reset and then executes to the specified address</td>
</tr>
<tr>
<td>__hwResetWithStrategy</td>
<td>Performs a hardware reset and halt with delay of the target CPU</td>
</tr>
<tr>
<td>__isBatchMode</td>
<td>Checks if C-SPY is running in batch mode or not.</td>
</tr>
<tr>
<td>__jlinkExecCommand</td>
<td>Sends a low-level command to the J-Link/J-Trace driver</td>
</tr>
<tr>
<td>__jtagCommand</td>
<td>Sends a low-level command to the JTAG instruction register</td>
</tr>
<tr>
<td>__jtagCP15IsPresent</td>
<td>Checks if coprocessor CP15 is available</td>
</tr>
<tr>
<td>__jtagCP15ReadReg</td>
<td>Returns the coprocessor CP15 register value</td>
</tr>
<tr>
<td>__jtagCP15WriteReg</td>
<td>Writes to the coprocessor CP15 register</td>
</tr>
<tr>
<td>__jtagData</td>
<td>Sends a low-level data value to the JTAG data register</td>
</tr>
<tr>
<td>__jtagRawRead</td>
<td>Returns the read data from the JTAG interface</td>
</tr>
<tr>
<td>__jtagRawSync</td>
<td>Writes accumulated data to the JTAG interface</td>
</tr>
<tr>
<td>__jtagRawWrite</td>
<td>Accumulates data to be transferred to the JTAG</td>
</tr>
<tr>
<td>__jtagResetTRST</td>
<td>Resets the ARM TAP controller via the TRST JTAG signal</td>
</tr>
<tr>
<td>__loadImage</td>
<td>Loads an image.</td>
</tr>
</tbody>
</table>

*Table 16: Summary of system macros (Continued)*
<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__memoryRestore</td>
<td>Restores the contents of a file to a specified memory zone</td>
</tr>
<tr>
<td>__memorySave</td>
<td>Saves the contents of a specified memory area to a file</td>
</tr>
<tr>
<td>__openFile</td>
<td>Opens a file for I/O operations</td>
</tr>
<tr>
<td>__orderInterrupt</td>
<td>Generates an interrupt</td>
</tr>
<tr>
<td>__popSimulatorInterruptExecutingStack</td>
<td>Informs the interrupt simulation system that an interrupt handler has finished executing</td>
</tr>
<tr>
<td>__readFile</td>
<td>Reads from the specified file</td>
</tr>
<tr>
<td>__readFileByte</td>
<td>Reads one byte from the specified file</td>
</tr>
<tr>
<td>__readMemoryByte</td>
<td>Reads one byte from the specified memory location</td>
</tr>
<tr>
<td>__readMemory16</td>
<td>Reads two bytes from the specified memory location</td>
</tr>
<tr>
<td>__readMemory32</td>
<td>Reads four bytes from the specified memory location</td>
</tr>
<tr>
<td>__registerMacroFile</td>
<td>Registers macros from the specified file</td>
</tr>
<tr>
<td>__resetFile</td>
<td>Rewinds a file opened by __openFile</td>
</tr>
<tr>
<td>__restoreSoftwareBreakpoints</td>
<td>Restores any breakpoints that were destroyed during system startup.</td>
</tr>
<tr>
<td>__setCodeBreak</td>
<td>Sets a code breakpoint</td>
</tr>
<tr>
<td>__setDataBreak</td>
<td>Sets a data breakpoint</td>
</tr>
<tr>
<td>__setLogBreak</td>
<td>Sets a log breakpoint</td>
</tr>
<tr>
<td>__setSimBreak</td>
<td>Sets a simulation breakpoint</td>
</tr>
<tr>
<td>__setTraceStartBreak</td>
<td>Sets a trace start breakpoint</td>
</tr>
<tr>
<td>__setTraceStopBreak</td>
<td>Sets a trace stop breakpoint</td>
</tr>
<tr>
<td>__sourcePosition</td>
<td>Returns the file name and source location if the current execution location corresponds to a source location</td>
</tr>
<tr>
<td>__strFind</td>
<td>Searches a given string for the occurrence of another string</td>
</tr>
<tr>
<td>__subString</td>
<td>Extracts a substring from another string</td>
</tr>
<tr>
<td>__targetDebuggerVersion</td>
<td>Returns the version of the target debugger</td>
</tr>
</tbody>
</table>

Table 16: Summary of system macros (Continued)
Reference information on C-SPY system macros

Macro Description

<table>
<thead>
<tr>
<th>Macro</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>__toLowerCase</td>
<td>Returns a copy of the parameter string where all the characters have been converted to lower case</td>
</tr>
<tr>
<td>__toString</td>
<td>Prints strings</td>
</tr>
<tr>
<td>__ToUpper</td>
<td>Returns a copy of the parameter string where all the characters have been converted to upper case</td>
</tr>
<tr>
<td>__unloadImage</td>
<td>Unloads a debug image.</td>
</tr>
<tr>
<td>__writeFile</td>
<td>Writes to the specified file</td>
</tr>
<tr>
<td>__writeFileByte</td>
<td>Writes one byte to the specified file</td>
</tr>
<tr>
<td>__writeMemory8,</td>
<td>Writes one byte to the specified memory location</td>
</tr>
<tr>
<td>__writeMemory16</td>
<td>Writes a two-byte word to the specified memory location</td>
</tr>
<tr>
<td>__writeMemory32</td>
<td>Writes a four-byte word to the specified memory location</td>
</tr>
</tbody>
</table>

Table 16: Summary of system macros  (Continued)

__cancelAllInterrupts

Syntax

__cancelAllInterrupts()

Return value

int 0

Applicability

The C-SPY Simulator.

Description

Cancels all ordered interrupts.

__cancelInterrupt

Syntax

__cancelInterrupt(interrupt_id)

Parameters

interrupt_id

The value returned by the corresponding __orderInterrupt macro call (unsigned long).

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>int 0</td>
</tr>
</tbody>
</table>

Table 17: __cancelInterrupt return values

C-SPY® Debugging Guide

for ARM
**C-SPY macros**

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unsuccessful</td>
<td>Non-zero error number</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Applicability</th>
<th>The C-SPY Simulator.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Cancels the specified interrupt.</td>
</tr>
</tbody>
</table>

**__clearBreak**

**Syntax**

```c
__clearBreak(break_id)
```

**Parameters**

- `break_id`
  
  The value returned by any of the set breakpoint macros.

**Return value**

- int 0

**Description**

Clears a user-defined breakpoint.

**See also**

*Breakpoints*, page 119.

**__closeFile**

**Syntax**

```c
__closeFile(fileHandle)
```

**Parameters**

- `fileHandle`
  
  A macro variable used as filehandle by the `__openFile` macro.

**Return value**

- int 0

**Description**

Closes a file previously opened by `__openFile`.

**__delay**

**Syntax**

```c
__delay(value)
```

**Parameters**

- `value`
  
  The number of milliseconds to delay execution.

**Return value**

- int 0

---

Table 17: __cancelInterrupt return values
Reference information on C-SPY system macros

**Description**
Delays execution the specified number of milliseconds.

**__disableInterrupts**

**Syntax**

```
__disableInterrupts()
```

**Return value**

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>int 0</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>Non-zero error number</td>
</tr>
</tbody>
</table>

Table 18: __disableInterrupts return values

**Applicability**
The C-SPY Simulator.

**Description**
Disables the generation of interrupts.

**__driverType**

**Syntax**

```
__driverType(driver_id)
```

**Parameters**

- `driver_id`:
  A string corresponding to the driver you want to check for. Choose one of these:
  - "sim": corresponds to the C-SPY simulator driver
  - "angel": corresponds to the C-SPY Angel driver
  - "gdbserv": corresponds to the C-SPY GDB Server driver
  - "generic": corresponds to third-party drivers
  - "ijet": corresponds to the C-SPY I-jet/JTAGjet driver
  - "jlink": corresponds to the C-SPY J-Link/J-Trace driver
  - "jtag": corresponds to the C-SPY Macraigor driver
  - "lmiftdi": corresponds to the C-SPY TI Stellaris driver
  - "xds100": corresponds to the C-SPY TI XDS100 driver
  - "rdi": corresponds to the C-SPY RDI driver
  - "rom": corresponds to the C-SPY IAR ROM-monitor driver
  - "sim": corresponds to the C-SPY simulator driver
Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>1</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 19: __driverType return values

Applicability
All C-SPY drivers

Description
Checks to see if the current C-SPY driver is identical to the driver type of the driver_id parameter.

Example
__driverType("sim")
If the simulator is the current driver, the value 1 is returned. Otherwise 0 is returned.

__emulatorSpeed

Syntax
__emulatorSpeed(speed)

Parameters

| speed | The emulator speed in Hz. Use 0 (zero) to make the speed automatically detected. Use -1 for adaptive speed (only for emulators supporting adaptive speed). |

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>The previous speed, or 0 (zero) if unknown</td>
</tr>
<tr>
<td>Unsuccessful; the speed is not supported by -1 the emulator</td>
<td>-1</td>
</tr>
</tbody>
</table>

Table 20: __emulatorSpeed return values

Applicability
The C-SPY hardware drivers

Description
Sets the emulator clock frequency. For JTAG interfaces, this is the JTAG clock frequency as seen on the TCK signal.

Example
__emulatorSpeed(0)
Sets the emulator speed to be automatically detected.
__emulatorStatusCheckOnRead

Syntax

__emulatorStatusCheckOnRead(status)

Parameters

status Use 0 to enable checks (default). Use 1 to disable checks.

Return value

int 0

Applicability

The C-SPY J-Link/J-Trace driver

For the C-SPY I-jet/JTAG-jet driver, this macro is recognized, but has no effect.

Description

Enables or disables the driver verification of CPSR (current processor status register) after each read operation. Typically, this macro can be used for initiating JTAG connections on some CPUs, like Texas Instruments’ TMS470R1B1M.

Note: Enabling this verification can cause problems with some CPUs, for example if invalid CPSR values are returned. However, if this verification is disabled (SetCheckModeAfterRead = 0), the success of read operations cannot be verified and possible data aborts are not detected.

Example

__emulatorStatusCheckOnRead(1)

Disables the checks for data aborts on memory reads.

__enableInterrupts

Syntax

__enableInterrupts()

Return value

Result Value
Successful int 0
Unsuccessful Non-zero error number

Table 21: __enableInterrupts return values

Applicability

The C-SPY Simulator.

Description

Enables the generation of interrupts.

__evaluate

Syntax

__evaluate(string, valuePtr)
Parameters

- **string**: Expression string.
- **valuePtr**: Pointer to a macro variable storing the result.

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>int 0</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>int 1</td>
</tr>
</tbody>
</table>

Table 22: __evaluate return values

Description

This macro interprets the input string as an expression and evaluates it. The result is stored in a variable pointed to by `valuePtr`.

Example

This example assumes that the variable `i` is defined and has the value 5:

```c
__evaluate("i + 3", &myVar)
```

The macro variable `myVar` is assigned the value 8.

__gdbserver_exec_command

Syntax

```c
__gdbserver_exec_command("string")
```

Parameters

- **"string"**: String or command sent to the GDB Server; see its documentation for more information.

Applicability

The C-SPY C-SPY GDB Server driver

Description

Use this option to send strings or commands to the GDB Server.

__getTracePortSize

Syntax

```c
__getTracePortSize
```

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>The width of the trace port in bits.</td>
<td>1, 2, 4, 8, or 16.</td>
</tr>
</tbody>
</table>

Table 23: __getTracePortSize return values
Applicability
The C-SPY I-jet/JTAGjet driver
The C-SPY J-Link/J-Trace driver

Description
Returns the width of the trace port.

See also
ETM Trace Settings dialog box (I-jet/JTAGjet), page 195 and ETM Trace Settings dialog box (J-Link/J-Trace), page 197, respectively.

__hwJetResetWithStrategy

Syntax
__hwJetResetWithStrategy(halt_delay, strategy)

Parameters

halt_delay
The delay, in milliseconds, between the end of the reset pulse and the halt of the CPU. Use 0 (zero) to make the CPU halt immediately after reset; only when strategy is set to 0.

strategy
The reset strategy number. For information about supported reset strategies, see --jet_standard_reset, page 371.

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful. The delay feature is not supported by the debugging probe</td>
<td>-1</td>
</tr>
<tr>
<td>Unsuccessful. The reset strategy is not supported by the debugging probe</td>
<td>-3</td>
</tr>
<tr>
<td>Unsuccessful. Other</td>
<td>-4</td>
</tr>
</tbody>
</table>

Table 24: __hwJetResetWithStrategy return values

Applicability
The C-SPY I-jet/JTAGjet driver

Description
Specifies the reset strategy to perform.

Example
__hwJetResetWithStrategy(0,2)
Performs a hardware reset.

__hwReset

Syntax
__hwReset(halt_delay)
**C-SPY macros**

### Parameters

- **halt_delay**

  The delay, in milliseconds, between the end of the reset pulse and the halt of the CPU. Use 0 (zero) to make the CPU halt immediately after reset.

### Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful. The actual delay value implemented by the emulator</td>
<td>&gt;=0</td>
</tr>
<tr>
<td>Successful. The delay feature is not supported by the emulator</td>
<td>-1</td>
</tr>
<tr>
<td>Unsuccessful. Hardware reset is not supported by the emulator</td>
<td>-2</td>
</tr>
</tbody>
</table>

*Table 25: __hwReset return values*

### Applicability

This system macro is available for all JTAG interfaces.

### Description

Performs a hardware reset and halt of the target CPU.

### Example

```
__hwReset(0)
```

Resets the CPU and immediately halts it.

---

**__hwResetRunToBp**

**Syntax**

```
__hwResetRunToBp(strategy, breakpoint_address, timeout)
```

**Parameters**

- **strategy**

  For information about supported reset strategies, see the [IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores](#).

- **breakpoint_address**

  The address of the breakpoint to execute to, specified as an integer value (symbols cannot be used).

- **timeout**

  A time out for the breakpoint, specified in milliseconds. If the breakpoint is not reached within the specified time, the core will be halted.

**Return value**

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&gt;=0</td>
<td>Successful. The approximate execution time in ms until the breakpoint is hit</td>
</tr>
<tr>
<td>-2</td>
<td>Unsuccessful. Hardware reset is not supported by the emulator</td>
</tr>
</tbody>
</table>

*Table 26: __hwResetRunToBp return values*
Reference information on C-SPY system macros

**Applicability**
The C-SPY I-jet/JTAGjet driver

The C-SPY J-Link/J-Trace driver

**Description**
Performs a hardware reset, sets a breakpoint at the specified address, executes to the breakpoint, and then removes it. The breakpoint address should be the start address of the downloaded image after it has been copied to RAM.

This macro is intended for running a boot loader that copies the application image from flash to RAM. The macro should be executed after the image has been downloaded to flash, but before the image is verified. The macro can be run in `execUserFlashExit` or `execUserPreload`.

**Example**

```
__hwResetRunToBp(0, 0x400000, 10000)
```

Resets the CPU with the reset strategy 0 and executes to the address 0x400000. If the breakpoint is not reached within 10 seconds, execution stops in accordance with the specified time out.

**__hwResetWithStrategy**

**Syntax**

```
__hwResetWithStrategy(halt_delay, strategy)
```

**Parameters**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>halt_delay</code></td>
<td>The delay, in milliseconds, between the end of the reset pulse and the halt of the CPU. Use 0 (zero) to make the CPU halt immediately after reset; only when <code>strategy</code> is set to 0.</td>
</tr>
<tr>
<td><code>strategy</code></td>
<td>For information about supported reset strategies, see the IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores.</td>
</tr>
</tbody>
</table>

**Return value**

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful. The actual delay in milliseconds, as implemented by the emulator</td>
<td>&gt;=0</td>
</tr>
<tr>
<td>Successful. The delay feature is not supported by the emulator</td>
<td>-1</td>
</tr>
<tr>
<td>Unsuccessful. Hardware reset is not supported by the emulator</td>
<td>-2</td>
</tr>
</tbody>
</table>

Table 27: __hwResetWithStrategy return values
### __hwResetWithStrategy__

**Syntax**

```c
__hwResetWithStrategy__()
```

**Return value**

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>int 1</td>
</tr>
<tr>
<td>False</td>
<td>int 0</td>
</tr>
</tbody>
</table>

**Description**

This macro returns True if the debugger is running in batch mode, otherwise it returns False.

**Applicability**

The C-SPY I-jet/JTAGjet driver

The C-SPY J-Link/J-Trace driver

This macro exists also in the other C-SPY hardware drivers, but there it has no effect.

**Example**

```c
__hwResetWithStrategy__(0,1)
```

Resets the CPU and halts it using a breakpoint at memory address zero.

### __isBatchMode__

**Syntax**

```c
__isBatchMode__()
```

**Return value**

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>int 1</td>
</tr>
<tr>
<td>False</td>
<td>int 0</td>
</tr>
</tbody>
</table>

**Description**

This macro returns True if the debugger is running in batch mode, otherwise it returns False.

### __jlinkExecCommand__

**Syntax**

```c
__jlinkExecCommand__(cmdstr)
```

**Parameters**

- `cmdstr` J-Link/J-Trace command string

**Return value**

int 0

**Applicability**

The C-SPY J-Link/J-Trace driver

---

Table 27: __hwResetWithStrategy__ return values (Continued)

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>int 1</td>
</tr>
<tr>
<td>False</td>
<td>int 0</td>
</tr>
</tbody>
</table>

Table 28: __isBatchMode__ return values

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>True</td>
<td>int 1</td>
</tr>
<tr>
<td>False</td>
<td>int 0</td>
</tr>
</tbody>
</table>

Unsuccessful. The reset strategy is not supported by the emulator -3

Result Value

<table>
<thead>
<tr>
<th>Table 27: <strong>hwResetWithStrategy</strong> return values (Continued)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>True</td>
</tr>
<tr>
<td>False</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 28: <strong>isBatchMode</strong> return values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>True</td>
</tr>
<tr>
<td>False</td>
</tr>
</tbody>
</table>

| AFE1_AFE2-1:1 |
Description

Sends a low-level command to the J-Link/J-Trace driver. For a list of possible commands, see the IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores.

Example

See the IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores.

See also

--jlink_exec_command, page 374

__jtagCommand

Syntax

__jtagCommand(ir)

Parameters

ir can be one of:

2 SCAN_N
4 RESTART
12 INTEST
14 IDCODE
15 BYPASS

Return value

int 0

Applicability

The C-SPY J-Link/J-Trace driver

Description

Sends a low-level command to the JTAG instruction register IR.

Example

__jtagCommand(14);
Id = __jtagData(0,32);

Returns the JTAG ID of the ARM target device.

__jtagCP15IsPresent

Syntax

__jtagCP15IsPresent()

Return value

1 if CP15 is available, otherwise 0.

Applicability

The C-SPY I-jet/JTAGjet driver

The C-SPY J-Link/J-Trace driver
**__jtagCP15ReadReg**

**Syntax**

```c
__jtagCP15ReadReg(CRn, CRm, op1, op2)
```

**Parameters**

- `CRn`: The register to read from.
- `CRm`: The register to read to.
- `op1`: The operation code, always 0.
- `op2`: Additional operation code.

**Return value**

The register value.

**Applicability**

The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver

**Description**

Reads the value of the CP15 register and returns its value.

---

**__jtagCP15WriteReg**

**Syntax**

```c
__jtagCP15WriteReg(CRn, CRm, op1, op2, value)
```

**Parameters**

- `CRn`: The register to write to.
- `CRm`: The register containing the value to write.
- `op1`: The operation code, always 0.
- `op2`: Additional operation code.
- `value`: The value to write.

**Applicability**

The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver

**Description**

Writes a value to the CP15 register.

---

**__jtagData**

**Syntax**

```c
__jtagData(dr, bits)
```

**Parameters**

- `dr`: 32-bit data register value
- `bits`: Number of valid bits in `dr`, both for the macro parameter and the return value; starting with the least significant bit (1...32).
Reference information on C-SPY system macros

Return value
Returns the result of the operation; the number of bits in the result is given by the bits parameter.

Applicability
The C-SPY J-Link/J-Trace driver

Description
Sends a low-level data value to the JTAG data register DR. The bit shifted out of DR is returned.

Example
__jtagCommand(14);
Id = __jtagData(0,32);

__jtagRawRead

Syntax
__jtagRawRead(bitpos, numbits)

Parameters

- **bitpos**: The start bit position in the returned JTAG bits to return data from
- **numbits**: The number of bits to read. The maximum value is 32.

Applicability
This system macro is available for the J-Link/J-Trace driver.

Description
Returns the data read from the JTAG TDO. Only the least significant bits contain data; the last bit read is from the least significant bit. This function can be called an arbitrary number of times to get all bits returned by an operation. This function also makes an implicit synchronization of any accumulated write bits.

Example
The following piece of pseudocode illustrates how the data is written to the JTAG (on the TMS and TDI pins) and read (from TDO):
__var Id;
__var BitPos;
/***************************************************************************/
*  
* ReadId()  
*  
ReadId() {  
__message "Reading JTAG Id\n";
__jtagRawWrite(0, 0x1f, 6); /* Goto IDLE via RESET state */
__jtagRawWrite(0, 0x1, 3); /* Enter DR scan chain */
BitPos = __jtagRawWrite(0, 0x80000000, 32); /* Shift 32 bits 
    into DR. Remember BitPos for Read operation */
__jtagRawWrite(0, 0x1, 2); /* Goto IDLE */
Id = __jtagRawRead(BitPos, 32); /* Read the Id */
__message "JTAG Id: ", Id:%x, 
}

**__jtagRawSync**

Syntax

__jtagRawSync()

Return value

int 0

Applicability

The C-SPY J-Link/J-Trace driver

Description

Sends arbitrary data to the JTAG interface. All accumulated bits using
__jtagRawWrite will be written to the JTAG scan chain. The data is sent
synchronously with TCK and typically sampled by the device on rising edge of TCK.

Example

The following piece of pseudocode illustrates how the data is written to the JTAG (on
the TMS and TDI pins) and read (from TDO):
int i;
U32 tdo;
for (i = 0; i < numBits; i++) {
    TDI = tdi & 1; /* Set TDI pin */
    TMS = tms & 1; /* Set TMS pin */
    TCK = 0;
    TCK = 1;
    tdo <<= 1;
    if (TDO) {
        tdo |= 1;
    }
    tdi >>= 1;
    tms >>= 1;
}

__jtagRawWrite

Syntax
__jtagRawWrite(tdi, tms, numbits)

Parameters
tdi
The data output to the TDI pin. This data is sent with the least significant bit first.
tms
The data output to the TMS pin. This data is sent with the least significant bit first.
umbits
The number of bits to transfer. Every bit results in a falling and rising edge of the JTAG TCK line. The maximum value is 64.

Return value
Returns the bit position of the data in the accumulated packet. Typically, this value is used when reading data from the JTAG.

Applicability
The C-SPY J-Link/J-Trace driver

Description
Accumulates bits to be transferred to the JTAG. If 32 bits are not enough, this function can be called multiple times. Both data output lines (TMS and TDI) can be controlled separately.

Example
/* Send five 1 bits on TMS to go to TAP-RESET state */
__jtagRawWrite(0x1F, 0, 5); /* Store bits in buffer */
__jtagRawSync(); /* Transfer buffer, writing tms, tdi, reading tdo */

Returns the JTAG ID of the ARM target device.
__jtagResetTRST

Syntax
__jtagResetTRST()

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>int 0</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>Non-zero error number</td>
</tr>
</tbody>
</table>

Table 28: __jtagResetTRST return values

Applicability
The C-SPY J-Link/J-Trace driver

Description
Resets the ARM TAP controller via the TRST JTAG signal.

__loadImage

Syntax
__loadImage(path, offset, debugInfoOnly)

Parameters

<table>
<thead>
<tr>
<th>path</th>
</tr>
</thead>
<tbody>
<tr>
<td>A string that identifies the path to the image to download. The path must either be absolute or use argument variables. For information about argument variables, see the IDE Project Management and Building Guide for ARM.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>An integer that identifies the offset to the destination address for the downloaded image.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>debugInfoOnly</th>
</tr>
</thead>
<tbody>
<tr>
<td>A non-zero integer value if no code or data should be downloaded to the target system, which means that C-SPY will only read the debug information from the debug file. Or, 0 (zero) for download.</td>
</tr>
</tbody>
</table>

Return value

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Non-zero integer number</td>
<td>A unique module identification.</td>
</tr>
<tr>
<td>int 0</td>
<td>Loading failed.</td>
</tr>
</tbody>
</table>

Table 30: __loadImage return values

Description
Loads an image (debug file).

Example 1
Your system consists of a ROM library and an application. The application is your active project, but you have a debug file corresponding to the library. In this case you can add
this macro call in the execUserSetup macro in a C-SPY macro file, which you associate with your project:

```c
__loadImage(ROMfile, 0x8000, 1);
```

This macro call loads the debug information for the ROM library `ROMfile` without downloading its contents (because it is presumably already in ROM). Then you can debug your application together with the library.

**Example 2**

Your system consists of a ROM library and an application, but your main concern is the library. The library needs to be programmed into flash memory before a debug session. While you are developing the library, the library project must be the active project in the IDE. In this case you can add this macro call in the execUserSetup macro in a C-SPY macro file, which you associate with your project:

```c
__loadImage(ApplicationFile, 0x8000, 0);
```

The macro call loads the debug information for the application and downloads its contents (presumably into RAM). Then you can debug your library together with the application.

See also


## __memoryRestore

**Syntax**

```c
__memoryRestore(zone, filename)
```

**Parameters**

- `zone`
  
  A string that specifies the memory zone, see *C-SPY memory zones, page 155.*

- `filename`
  
  A string that specifies the file to be read. The filename must include a path, which must either be absolute or use argument variables. For information about argument variables, see the *IDE Project Management and Building Guide for ARM.*

**Return value**

`int 0`

**Description**

Reads the contents of a file and saves it to the specified memory zone.

**Example**

```c
__memoryRestore("Memory", "c:\temp\saved_memory.hex");
```

See also

*Memory Restore dialog box, page 164.*
__memorySave

Syntax

__memorySave(start, stop, format, filename)

Parameters

start
A string that specifies the first location of the memory area to be saved.

stop
A string that specifies the last location of the memory area to be saved.

format
A string that specifies the format to be used for the saved memory. Choose between:

- intel-extended
- motorola
- motorola-s19
- motorola-s28
- motorola-s37.

filename
A string that specifies the file to write to. The filename must include a path, which must either be absolute or use argument variables. For information about argument variables, see the IDE Project Management and Building Guide for ARM.

Return value

int 0

Description
Saves the contents of a specified memory area to a file.

Example

__memorySave("Memory:0x00", "Memory:0xFF", "intel-extended", "c:\temp\saved_memory.hex");

See also
Memory Save dialog box, page 163.

__openFile

Syntax

__openFile(filename, access)
Parameters

filename
The file to be opened. The filename must include a path, which must either be absolute or use argument variables. For information about argument variables, see the IDE Project Management and Building Guide for ARM.

access
The access type (string).
These are mandatory but mutually exclusive:
"a" append, new data will be appended at the end of the open file
"r" read
"w" write

These are optional and mutually exclusive:
"b" binary, opens the file in binary mode
"t" ASCII text, opens the file in text mode
This access type is optional:
"+" together with r, w, or a; r+ or w+ is read and write, while a+ is read and append

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>The file handle</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>An invalid file handle, which tests as False</td>
</tr>
</tbody>
</table>

Table 31: __openFile return values

Description
Opens a file for I/O operations. The default base directory of this macro is where the currently open project file (*.ewp) is located. The argument to __openFile can specify a location relative to this directory. In addition, you can use argument variables such as $PROJ_DIRS$ and $TOOLKIT_DIRS$ in the path argument.

Example

```c
__var myFileHandle;      /* The macro variable to contain */
/* the file handle */
myFileHandle = __openFile("$PROJ_DIRS\Debug\Exe\test.tst", "r");
if (myFileHandle)
{
    /* successful opening */
}
```
See also
For information about argument variables, see the IDE Project Management and Building Guide for ARM.

__orderInterrupt

Syntax
__orderInterrupt(specification, first_activation, repeat_interval, variance, infinite_hold_time, hold_time, probability)

Parameters

 specification
The interrupt (string). The specification can either be the full specification used in the device description file (ddf) or only the name. In the latter case the interrupt system will automatically get the description from the device description file.

 first_activation
The first activation time in cycles (integer)

 repeat_interval
The periodicity in cycles (integer)

 variance
The timing variation range in percent (integer between 0 and 100)

 infinite_hold_time
1 if infinite, otherwise 0.

 hold_time
The hold time (integer)

 probability
The probability in percent (integer between 0 and 100)

Return value
The macro returns an interrupt identifier (unsigned long). If the syntax of specification is incorrect, it returns -1.

Applicability
The C-SPY Simulator.

Description
Generates an interrupt.

Example
This example generates a repeating interrupt using an infinite hold time first activated after 4000 cycles
__orderInterrupt( 'IRQ', 4000, 2000, 0, 1, 0, 100 );
__popSimulatorInterruptExecutingStack

Syntax
__popSimulatorInterruptExecutingStack(void)

Return value
int 0

Applicability
The C-SPY Simulator.

Description
Informs the interrupt simulation system that an interrupt handler has finished executing, as if the normal instruction used for returning from an interrupt handler was executed. This is useful if you are using interrupts in such a way that the normal instruction for returning from an interrupt handler is not used, for example in an operating system with task-switching. In this case, the interrupt simulation system cannot automatically detect that the interrupt has finished executing.

See also
Simulating an interrupt in a multi-task system, page 278.

__readFile

Syntax
__readFile(fileHandle, valuePtr)

Parameters
fileHandle
A macro variable used as filehandle by the __openFile macro.

valuePtr
A pointer to a variable.

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>0</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>Non-zero error number</td>
</tr>
</tbody>
</table>

Table 32: __readFile return values

Description
Reads a sequence of hexadecimal digits from the given file and converts them to an unsigned long which is assigned to the value parameter, which should be a pointer to a macro variable.

Example

```c
__var number;
if (__readFile(myFileHandle, &number) == 0)
{
    // Do something with number
}
```
**__readFileByte**

**Syntax**

```
__readFileByte(fileHandle)
```

**Parameters**

- **fileHandle**
  
  A macro variable used as filehandle by the **openFile** macro.

**Return value**

-1 upon error or end-of-file, otherwise a value between 0 and 255.

**Description**

Reads one byte from a file.

**Example**

```c
__var byte;
while ( (byte = __readFileByte(myFileHandle)) != -1 )
{
   /* Do something with byte */
}
```

**__readMemory8, __readMemoryByte**

**Syntax**

```
__readMemory8(address, zone)
__readMemoryByte(address, zone)
```

**Parameters**

- **address**
  
  The memory address (integer).

- **zone**
  
  A string that specifies the memory zone, see C-SPY memory zones, page 155.

**Return value**

The macro returns the value from memory.

**Description**

Reads one byte from a given memory location.

**Example**

```c
__readMemory8(0x0108, "Memory");
```

**__readMemory16**

**Syntax**

```
__readMemory16(address, zone)
```

**Parameters**

- **address**
  
  The memory address (integer).
Reference information on C-SPY system macros

zone
   A string that specifies the memory zone, see C-SPY memory zones, page 155.

Return value
   The macro returns the value from memory.

Description
   Reads a two-byte word from a given memory location.

Example
   __readMemory16(0x0108, 'Memory');

__readMemory32

Syntax
   __readMemory32(address, zone)

Parameters
   address
      The memory address (integer).
   zone
      A string that specifies the memory zone, see C-SPY memory zones, page 155.

Return value
   The macro returns the value from memory.

Description
   Reads a four-byte word from a given memory location.

Example
   __readMemory32(0x0108, 'Memory');

__registerMacroFile

Syntax
   __registerMacroFile(filename)

Parameters
   filename
      A file containing the macros to be registered (string). The filename must include
      a path, which must either be absolute or use argument variables. For information
      about argument variables, see the IDE Project Management and Building Guide
      for ARM.

Return value
   int 0

Description
   Registers macros from a setup macro file. With this function you can register multiple
   macro files during C-SPY startup.

Example
   __registerMacroFile("c:\\testdir\\macro.mac");
See also  
*Registering and executing using setup macros and setup files*, page 298.

**__resetFile**

Syntax  

```
__resetFile(fileHandle)
```

Parameters  

`fileHandle`

A macro variable used as filehandle by the `__openFile` macro.

Return value  

`int 0`

Description  

Rewinds a file previously opened by `__openFile`.

**__restoreSoftwareBreakpoints**

Syntax  

```
__restoreSoftwareBreakpoints()
```

Return value  

`int 0`

Applicability  

- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY TI Stellaris driver
- The C-SPY Macraigor driver

Description  

Restores automatically any breakpoints that were destroyed during system startup.

This can be useful if you have an application that is copied to RAM during startup and is then executing in RAM. This can, for example, be the case if you use the `initialize by copy` directive for code in the linker configuration file or if you have any `__ramfunc` declared functions in your application. In this case, any breakpoints will be overwritten during the RAM copying when the application execution starts.

By using this macro, C-SPY will restore the destroyed breakpoints.
__setCodeBreak

Syntax

__setCodeBreak(location, count, condition, cond_type, action)

Parameters

location
A string that defines the code location of the breakpoint, either a valid C-SPY expression whose value evaluates to a valid address, an absolute location, or a source location. For more information about the location types, see Enter Location dialog box, page 25.

count
The number of times that a breakpoint condition must be fulfilled before a break occurs (integer).

condition
The breakpoint condition (string).

cond_type
The condition type; either "CHANGED" or "TRUE" (string).

action
An expression, typically a call to a macro, which is evaluated when the breakpoint is detected.

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>An unsigned integer uniquely identifying the breakpoint. This value must be used to clear the breakpoint.</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 33: __setCodeBreak return values

Applicability

The C-SPY simulator

The C-SPY hardware drivers

Description

Sets a code breakpoint, that is, a breakpoint which is triggered just before the processor fetches an instruction at the specified location.

Examples

__setCodeBreak("{D:\src\prog.c}.12.9", 3, "d>16", "TRUE", "ActionCode()");

This example sets a code breakpoint on the label main in your source:

__setCodeBreak("main", 0, "1", "TRUE", "");
__setDataBreak

Syntax

In the simulator:

__setDataBreak(location, count, condition, cond_type, access, action)

In the I-jet/JTAGjet driver:

__setDataBreak(location, access, extend, match, data, mask)

Parameters

location

A string that defines the data location of the breakpoint, either a valid C-SPY expression whose value evaluates to a valid address or an absolute location. For more information about the location types, see Enter Location dialog box, page 25.

count

The number of times that a breakpoint condition must be fulfilled before a break occurs (integer).

This parameter applies to the simulator only.

condition

The breakpoint condition (string).

This parameter applies to the simulator only.

cond_type

The condition type; either "CHANGED" or "TRUE" (string).

This parameter applies to the simulator only.

access

The memory access type: "R", for read, "W" for write, or "RW" for read/write.

action

An expression, typically a call to a macro, which is evaluated when the breakpoint is detected.

This parameter applies to the simulator only.

See also

Breakpoints, page 119.
**extend**

Extends the breakpoint so that a whole data structure is covered. For data structures that do not fit the size of the possible breakpoint ranges supplied by the hardware breakpoint unit, for example three bytes, the breakpoint range will not cover the whole data structure. Note that the breakpoint range will be extended beyond the size of the data structure, which might cause false triggers at adjacent data. Choose between "TRUE" or "FALSE".

This parameter applies to I-jet/JTAGjet only.

**match**

Enables matching of the accessed data. Choose between "TRUE" or "FALSE".

This parameter applies to I-jet/JTAGjet only.

**data**

A data value to match, in unsigned 32-bit format.

This parameter applies to I-jet/JTAGjet only.

**mask**

Specifies which part of the data value to match (word, halfword, or byte), in unsigned 32-bit format.

This parameter applies to I-jet/JTAGjet only.

---

**Return value**

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>An unsigned integer uniquely identifying the breakpoint. This value must be used to clear the breakpoint.</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>0</td>
</tr>
</tbody>
</table>

*Table 34: __setDataBreak return values*

---

**Applicability**

The C-SPY Simulator.

The I-jet/JTAGjet driver.

**Description**

Sets a data breakpoint, that is, a breakpoint which is triggered directly after the processor has read or written data at the specified location.

**Example**

For the C-SPY simulator:

```c
__var brk;
brk = __setDataBreak("Memory:0x4710", 3, "d>6", "TRUE", "W", "ActionData()");
...
__clearBreak(brk);
```
__setDataLogBreak

Syntax

__setDataLogBreak(location, access, extend)

Parameters

location
A string that defines the data location of the breakpoint, either a valid C-SPY expression whose value evaluates to a valid address or an absolute location. For more information about the location types, see Enter Location dialog box, page 25.

access
The memory access type: "R", for read, "W" for write, or "RW" for read/write.

extend
Extends the breakpoint so that a whole data structure is covered. For data structures that do not fit the size of the possible breakpoint ranges supplied by the hardware breakpoint unit, for example three bytes, the breakpoint range will not cover the whole data structure. Note that the breakpoint range will be extended beyond the size of the data structure, which might cause false triggers at adjacent data. Choose between "TRUE" or "FALSE".

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>An unsigned integer uniquely identifying the breakpoint. This value must be used to clear the breakpoint.</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 35: __setDataLogBreak return values

Applicability
The C-SPY Simulator
The C-SPY J-jet/JTAGjet driver

Description
Sets a data log breakpoint, that is, a breakpoint which is triggered when the processor reads or writes data at the specified location. Note that a data log breakpoint does not stop the execution it just generates a data log.

Example

```c
__var brk;
brk = __setDataLogBreak("Memory:0x4710", "R", "TRUE");
...
__clearBreak(brk);
```
See also Breakpoints, page 119 and Getting started using data logging, page 97.

__setLogBreak

Syntax

__setLogBreak(location, message, msg_type, condition, cond_type)

Parameters

location
A string that defines the code location of the breakpoint, either a valid C-SPY expression whose value evaluates to a valid address, an absolute location, or a source location. For more information about the location types, see Enter Location dialog box, page 25.

message
The message text.

msg_type
The message type; choose between:

TEXT, the message is written word for word.

ARGS, the message is interpreted as a comma-separated list of C-SPY expressions or strings.

condition
The breakpoint condition (string).

cond_type
The condition type; either "CHANGED" or "TRUE" (string).

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>An unsigned integer uniquely identifying the breakpoint. The same value must be used when you want to clear the breakpoint.</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 36: __setLogBreak return values

Applicability

The C-SPY Simulator

The C-SPY hardware drivers

Description

Sets a log breakpoint, that is, a breakpoint which is triggered when an instruction is fetched from the specified location. If you have set the breakpoint on a specific machine
instruction, the breakpoint will be triggered and the execution will temporarily halt and print the specified message in the C-SPY Debug Log window.

Example

```c
__var logBp1;
__var logBp2;

logOn()
{
    logBp1 = __setLogBreak("{C:\temp\Utilities.c}.23.1",
                        "Entering trace zone at ::", #PC:%X",
                        "ARGS", "1", "TRUE");
    logBp2 = __setLogBreak("{C:\temp\Utilities.c}.30.1",
                        "Leaving trace zone...", "TEXT", "1", "TRUE");
}

logOff()
{
    __clearBreak(logBp1);
    __clearBreak(logBp2);
}
```

See also

*Formatted output*, page 304 and *Breakpoints*, page 119.

__setSimBreak

**Syntax**

```c
__setSimBreak(location, access, action)
```

**Parameters**

- **location**
  
  A string that defines the data location of the breakpoint, either a valid C-SPY expression whose value evaluates to a valid address or an absolute location. For more information about the location types, see Enter Location dialog box, page 25.

- **count**
  
  The number of times that a breakpoint condition must be fulfilled before a break occurs (integer).

- **condition**
  
  The breakpoint condition (string).

- **cond_type**
  
  The condition type; either "CHANGED" or "TRUE" (string).

- **action**
  
  An expression, typically a call to a macro, which is evaluated when the breakpoint is detected.
Reference information on C-SPY system macros

Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>An unsigned integer uniquely identifying the breakpoint. This value must be used to clear the breakpoint.</td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 37: __setSimBreak return values

Applicability

The C-SPY Simulator.

Description

Use this system macro to set immediate breakpoints, which will halt instruction execution only temporarily. This allows a C-SPY macro function to be called when the processor is about to read data from a location or immediately after it has written data. Instruction execution will resume after the action.

This type of breakpoint is useful for simulating memory-mapped devices of various kinds (for instance serial ports and timers). When the processor reads at a memory-mapped location, a C-SPY macro function can intervene and supply the appropriate data. Conversely, when the processor writes to a memory-mapped location, a C-SPY macro function can act on the value that was written.

__setTraceStartBreak

Syntax

__setTraceStartBreak(location)

Parameters

location

A string that defines the code location of the breakpoint, either a valid C-SPY expression whose value evaluates to a valid address, an absolute location, or a source location. For more information about the location types, see Enter Location dialog box, page 25.

count

The number of times that a breakpoint condition must be fulfilled before a break occurs (integer)

condition

The breakpoint condition (string)

cond_type

The condition type; either "CHANGED" or "TRUE" (string)

action

An expression, typically a call to a macro, which is evaluated when the breakpoint is detected
Return value

<table>
<thead>
<tr>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>An unsigned integer uniquely identifying the breakpoint. The same</td>
</tr>
<tr>
<td>value must be used when you want to clear the breakpoint.</td>
<td></td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 38: __setTraceStartBreak return values

Applicability

The C-SPY Simulator.

Description

Sets a breakpoint at the specified location. When that breakpoint is triggered, the trace system is started.

Example

```c
__var startTraceBp;
__var stopTraceBp;

traceOn()
{
    startTraceBp = __setTraceStartBreak
        ("{C:\TEMP\Utilities.c}.23.1");
    stopTraceBp = __setTraceStopBreak
        ("{C:\temp\Utilities.c}.30.1");
}

traceOff()
{
    __clearBreak(startTraceBp);
    __clearBreak(stopTraceBp);
}
```

See also

Breakpoints, page 119.

__setTraceStopBreak

Syntax

```
__setTraceStopBreak(location)
```

Parameters

`location`

A string that defines the code location of the breakpoint, either a valid C-SPY expression whose value evaluates to a valid address, an absolute location, or a source location. For more information about the location types, see Enter Location dialog box, page 25.
count
The number of times that a breakpoint condition must be fulfilled before a break occurs (integer)

condition
The breakpoint condition (string)

cond_type
The condition type; either "CHANGED" or "TRUE" (string)

action
An expression, typically a call to a macro, which is evaluated when the breakpoint is detected

<table>
<thead>
<tr>
<th>Return value</th>
<th>Result</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>An unsigned integer uniquely identifying the breakpoint. The same value must be used when you want to clear the breakpoint.</td>
<td></td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>int 0</td>
<td></td>
</tr>
</tbody>
</table>

Table 39: __setTraceStopBreak return values

Applicability
The C-SPY Simulator.

Description
Sets a breakpoint at the specified location. When that breakpoint is triggered, the trace system is stopped.

Example
See __setTraceStartBreak, page 340.

See also
Breakpoints, page 119.

__sourcePosition

Syntax
__sourcePosition(linePtr, colPtr)

Parameters

- **linePtr**
  Pointer to the variable storing the line number

- **colPtr**
  Pointer to the variable storing the column number
Return value | Result | Value
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Successful</td>
<td>Filename string</td>
<td></td>
</tr>
<tr>
<td>Unsuccessful</td>
<td>Empty (&quot;&quot;) string</td>
<td></td>
</tr>
</tbody>
</table>

| Description | If the current execution location corresponds to a source location, this macro returns the filename as a string. It also sets the value of the variables, pointed to by the parameters, to the line and column numbers of the source location. |

__strFind

Syntax

```
__strFind(macroString, pattern, position)
```

Parameters

- `macroString` A macro string.
- `pattern` The string pattern to search for
- `position` The position where to start the search. The first position is 0

<table>
<thead>
<tr>
<th>Return value</th>
<th>The position where the pattern was found or -1 if the string is not found.</th>
</tr>
</thead>
</table>

| Description | This macro searches a given string (`macroString`) for the occurrence of another string (`pattern`). |

<table>
<thead>
<tr>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>__strFind(&quot;Compiler&quot;, &quot;pile&quot;, 0) = 3</code></td>
</tr>
<tr>
<td><code>__strFind(&quot;Compiler&quot;, &quot;foo&quot;, 0) = -1</code></td>
</tr>
</tbody>
</table>

| See also | Macro strings, page 303. |

__subString

Syntax

```
__subString(macroString, position, length)
```

Parameters

- `macroString` A macro string.
- `position` The start position of the substring. The first position is 0.

Table 40: __sourcePosition return values
Reference information on C-SPY system macros

**length**

The length of the substring

**Return value**
A substring extracted from the given macro string.

**Description**
This macro extracts a substring from another string (`macroString`).

**Example**
```c
__subString("Compiler", 0, 2)
```
The resulting macro string contains Co.

```c
__subString("Compiler", 3, 4)
```
The resulting macro string contains pile.

**See also**
*Macro strings*, page 303.

---

**__targetDebuggerVersion**

**Syntax**
```
__targetDebuggerVersion
```

**Return value**
A string that represents the version number of the C-SPY debugger processor module.

**Description**
This macro returns the version number of the C-SPY debugger processor module.

**Example**
```c
__var toolVer;
toolVer = __targetDebuggerVersion();
__message "The target debugger version is, ", toolVer;
```

---

**__toLower**

**Syntax**
```
__toLower(macroString)
```

**Parameters**
- `macroString`
  A macro string.

**Return value**
The converted macro string.

**Description**
This macro returns a copy of the parameter `macroString` where all the characters have been converted to lower case.

**Example**
```c
__toLower("IAR")
```
The resulting macro string contains iar.
__toLower("Mix42")
The resulting macro string contains mix42.

See also
Macro strings, page 303.

__toString

Syntax
__toString(C_string, maxlength)

Parameters
C_string
Any null-terminated C string.

maxlength
The maximum length of the returned macro string.

Return value
Macro string.

Description
This macro is used for converting C strings (char* or char[]) into macro strings.

Example
Assuming your application contains this definition:
char const * hptr = "Hello World!";
this macro call:
__toString(hptr, 5)
would return the macro string containing Hello.

See also
Macro strings, page 303.

__toUpper

Syntax
__toUpper(macroString)

Parameters
macroString
A macro string.

Return value
The converted string.

Description
This macro returns a copy of the parameter macroString where all the characters have been converted to upper case.
Example

```c
__toUpper('string')
```

The resulting macro string contains `STRING`.

See also

Macro strings, page 303.

---

### __unloadImage

**Syntax**

```c
__unloadImage(module_id)
```

**Parameters**

- `module_id`
  
  An integer which represents a unique module identification, which is retrieved as a return value from the corresponding `__loadImage` C-SPY macro.

**Return value**

<table>
<thead>
<tr>
<th>Value</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>module_id</code></td>
<td>A unique module identification (the same as the input parameter).</td>
</tr>
<tr>
<td><code>int 0</code></td>
<td>The unloading failed.</td>
</tr>
</tbody>
</table>

*Table 41: __unloadImage return values*

**Description**

Unloads debug information from an already downloaded image.

See also

Loading multiple images, page 56 and Images, page 394.

---

### __writeFile

**Syntax**

```c
__writeFile(fileHandle, value)
```

**Parameters**

- `fileHandle`
  
  A macro variable used as filehandle by the `__openFile` macro.
- `value`
  
  An integer.

**Return value**

`int 0`

**Description**

Prints the integer value in hexadecimal format (with a trailing space) to the file `file`.

*Note:* The `__fmessage` statement can do the same thing. The `__writeFile` macro is provided for symmetry with `__readFile`. 
C-SPY macros

__writeFileByte
Syntax
__writeFileByte(fileHandle, value)
Parameters
fileHandle
A macro variable used as filehandle by the __openFile macro.
value
An integer.
Return value
int 0
Description
Writes one byte to the file fileHandle.

__writeMemory8, __writeMemoryByte
Syntax
__writeMemory8(value, address, zone)
__writeMemoryByte(value, address, zone)
Parameters
value
An integer.
address
The memory address (integer).
zone
A string that specifies the memory zone, see C-SPY memory zones, page 155.
Return value
int 0
Description
Writes one byte to a given memory location.
Example
__writeMemory8(0x2F, 0x8020, "Memory");

__writeMemory16
Syntax
__writeMemory16(value, address, zone)
Parameters
value
An integer.
Reference information on C-SPY system macros

__writeMemory16

Syntax

__writeMemory16(value, address, zone)

Parameters

value

An integer.

address

The memory address (integer).

zone

A string that specifies the memory zone, see C-SPY memory zones, page 155.

Return value

int 0

Description

Writes two bytes to a given memory location.

Example

__writeMemory16(0x2FFF, 0x8020, "Memory");

__writeMemory32

Syntax

__writeMemory32(value, address, zone)

Parameters

value

An integer.

address

The memory address (integer).

zone

A string that specifies the memory zone, see C-SPY memory zones, page 155.

Return value

int 0

Description

Writes four bytes to a given memory location.

Example

__writeMemory32(0x5555FFFF, 0x8020, "Memory");
The C-SPY Command Line Utility—cspybat

This chapter describes how you can execute C-SPY® in batch mode, using the C-SPY Command Line Utility—cspybat.exe. More specifically, this means:

- Using C-SPY in batch mode
- Summary of C-SPY command line options
- Reference information on C-SPY command line options.

Using C-SPY in batch mode

You can execute C-SPY in batch mode if you use the command line utility cspybat, installed in the directory common\bin.

INVOCATION SYNTAX

The invocation syntax for cspybat is:

```
cspybat processor_DLL driver_DLL debug_file [cspybat_options] --backend driver_options
```

**Note:** In those cases where a filename is required—including the DLL files—you are recommended to give a full path to the filename.

Parameters

The parameters are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>processor_DLL</td>
<td>The processor-specific DLL file; available in arm\bin.</td>
</tr>
<tr>
<td>driver_DLL</td>
<td>The C-SPY driver DLL file; available in arm\bin.</td>
</tr>
<tr>
<td>debug_file</td>
<td>The object file that you want to debug (filename extension .out).</td>
</tr>
<tr>
<td>cspybat_options</td>
<td>The command line options that you want to pass to cspybat. Note that these options are optional. For information about each option, see Reference information on C-SPY command line options, page 356.</td>
</tr>
</tbody>
</table>

Table 42: cspybat parameters
Using C-SPY in batch mode

Example

In the following example, $EW_DIR$ represents the full path of the directory where you have installed IAR Embedded Workbench and $PROJ_DIR$ is the path of the project directory.

Starting cspybat using the simulator driver

```
$EW_DIR$\common\bin\cspybat $EW_DIR$\arm\bin\armproc.dll
$EW_DIR$\arm\bin\armsim.dll $PROJ_DIR$\myproject.out --plugin
$EW_DIR$\arm\bin\armbat.dll --backend sim -B --cpu arm -p
$EW_DIR$\arm\bin\config\devicedescription.ddf
```

OUTPUT

When you run cspybat, these types of output can be produced:

- Terminal output from cspybat itself
  
  All such terminal output is directed to stderr. Note that if you run cspybat from the command line without any arguments, the cspybat version number and all available options including brief descriptions are directed to stdout and displayed on your screen.

- Terminal output from the application you are debugging
  
  All such terminal output is directed to stdout, provided that you have used the --plugin option. See --plugin, page 383.

- Error return codes
  
  cspybat return status information to the host operating system that can be tested in a batch file. For successful, the value int 0 is returned, and for unsuccessful the value int 1 is returned.

USING AN AUTOMATICALLY GENERATED BATCH FILE

When you use C-SPY in the IDE, C-SPY generates a batch file $projectname$.cspy.bat every time C-SPY is initialized. You can find the file in the

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--backend</td>
<td>Marks the beginning of the parameters to the C-SPY driver; all options that follow will be sent to the driver. Note that this option is mandatory.</td>
</tr>
<tr>
<td>driver_options</td>
<td>The command line options that you want to pass to the C-SPY driver. Note that some of these options are mandatory and some are optional. For information about each option, see Reference information on C-SPY command line options, page 356.</td>
</tr>
</tbody>
</table>
The C-SPY Command Line Utility—cspybat

directory $PROJ_DIR\settings$. This batch file contains the same settings as in the IDE, and you can use it from the command line to start cspybat. The file also contains information about required modifications.

### Summary of C-SPY command line options

**GENERAL CSPYBAT OPTIONS**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--backend</td>
<td>Marks the beginning of the parameters to be sent to the C-SPY driver (mandatory).</td>
</tr>
<tr>
<td>--code_coverage_file</td>
<td>Enables the generation of code coverage information and places it in a specified file.</td>
</tr>
<tr>
<td>--cycles</td>
<td>Specifies the maximum number of cycles to run.</td>
</tr>
<tr>
<td>--download_only</td>
<td>Downloads a code image without starting a debug session afterwards.</td>
</tr>
<tr>
<td>--flash_loader</td>
<td>Specifies a flash loader specification XML file.</td>
</tr>
<tr>
<td>--macro</td>
<td>Specifies a macro file to be used.</td>
</tr>
<tr>
<td>--plugin</td>
<td>Specifies a plugin file to be used.</td>
</tr>
<tr>
<td>--silent</td>
<td>Omits the sign-on message.</td>
</tr>
<tr>
<td>--timeout</td>
<td>Limits the maximum allowed execution time.</td>
</tr>
</tbody>
</table>

**OPTIONS AVAILABLE FOR ALL C-SPY DRIVERS**

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-B</td>
<td>Enables batch mode (mandatory).</td>
</tr>
<tr>
<td>--BE8</td>
<td>Uses the big-endian format BE8. For reference information, see the <em>IAR C/C++ Development Guide for ARM</em>.</td>
</tr>
<tr>
<td>--BE32</td>
<td>Uses the big-endian format BE32. For reference information, see the <em>IAR C/C++ Development Guide for ARM</em>.</td>
</tr>
<tr>
<td>--cpu</td>
<td>Specifies a processor variant. For reference information, see the <em>IAR C/C++ Development Guide for ARM</em>.</td>
</tr>
<tr>
<td>--device</td>
<td>Specifies the name of the device.</td>
</tr>
</tbody>
</table>
Summary of C-SPY command line options

- **--drv_attach_to_program**
  Attaches the debugger to a running application at its current location. For reference information, see Download, page 392, specifically the option Attach to program.

- **--drv_catch_exceptions**
  Makes the application stop for certain exceptions.

- **--drv_communication**
  Specifies the communication link to be used.

- **--drv_communication_log**
  Creates a log file.

- **--drv_default_breakpoint**
  Sets the type of breakpoint resource to be used when setting breakpoints.

- **--drv_reset_to_cpu_start**
  Omits setting the PC when starting or resetting the debugger.

- **--drv_restore_breakpoints**
  Restores automatically any breakpoints that were destroyed during system startup.

- **--drv_suppress_download**
  Suppresses download of the executable image. For reference information, see Download, page 392, specifically the option Suppress download.

- **--drv_swo_clock_setup**
  Specifies the CPU clock and the wanted SWO speed.

- **--drv_vector_table_base**
  Specifies the location of the Cortex-M reset vector and the initial stack pointer value.

- **--drv_verify_download**
  Verifies the target program. For reference information, see Download, page 392, specifically the option Verify download.

Available for Angel, GDB Server, IAR ROM-monitor, J-Link/J-Trace, JTAGjet, Macraigor, RDI, ST-LINK, TI Stellaris, and TI XDS100.

- **--endian**
  Specifies the byte order of the generated code and data. For reference information, see the IAR C/C++ Development Guide for ARM.

- **--fpu**
  Selects the type of floating-point unit. For reference information, see the IAR C/C++ Development Guide for ARM.

- **-p**
  Specifies the device description file to be used.
OPTIONS AVAILABLE FOR THE SIMULATOR DRIVER

--disable_interrupts  Disables the interrupt simulation.
--mapu  Activates memory access checking.

OPTIONS AVAILABLE FOR THE C-SPY ANGEL DEBUG MONITOR DRIVER

--rdi_heartbeat  Makes C-SPY poll your target system periodically. For reference information, see Angel, page 397, specifically the option Send heartbeat.
--rdi_step_max_one  Executes one instruction.

OPTIONS AVAILABLE FOR THE C-SPY GDB SERVER DRIVER

--gdbserv_exec_command  Sends a command string to the GDB Server.

OPTIONS AVAILABLE FOR THE C-SPY IAR ROM-MONITOR DRIVER

There are no additional options specific to the C-SPY IAR ROM-monitor driver.

OPTIONS AVAILABLE FOR THE C-SPY I-JET/JTAGJET DRIVER

--jet_cpu_clock  Specifies the frequency of the internal processor clock.
--jet_interface  Selects the communication interface.
--jet_ir_length  Specifies the number of IR bits preceding the ARM core to connect to.
--jet_jtag_speed  Specifies the JTAG and SWD speed.
--jet_power_from_probe  Specifies the power supply from the I-jet probe.
--jet_script_file  Specifies the reset script file.
### OPTIONS AVAILABLE FOR THE C-SPY J-LINK/J-TRACE DRIVER

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--jlink_device_select</td>
<td>Selects a specific device in the JTAG scan chain.</td>
</tr>
<tr>
<td>--jlink_exec_command</td>
<td>Calls the <code>__jlinkExecCommand</code> macro after target connection has been established.</td>
</tr>
<tr>
<td>--jlink_initial_speed</td>
<td>Sets the initial JTAG communication speed in kHz.</td>
</tr>
<tr>
<td>--jlink_interface</td>
<td>Specifies the communication between the J-Link debug probe and the target system.</td>
</tr>
<tr>
<td>--jlink_ir_length</td>
<td>Sets the number of IR bits preceding the ARM core to connect to.</td>
</tr>
<tr>
<td>--jlink_reset_strategy</td>
<td>Selects the reset strategy to be used at debugger startup.</td>
</tr>
<tr>
<td>--jlink_script_file</td>
<td>Specifies the script file for setting up hardware.</td>
</tr>
<tr>
<td>--jlink_speed</td>
<td>Sets the JTAG communication speed in kHz.</td>
</tr>
<tr>
<td>--jlink_trace_source</td>
<td>Selects either ETB or ETM as the trace source.</td>
</tr>
</tbody>
</table>

### OPTIONS AVAILABLE FOR THE C-SPY TI STELLARIS DRIVER

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--lmftdi_speed</td>
<td>Sets the JTAG communication speed in kHz.</td>
</tr>
</tbody>
</table>
OPTIONS AVAILABLE FOR THE C-SPY TI XDS100 DRIVER

--xds_rootdir Specifies the installation directory of the TI XDS100 driver package.

OPTIONS AVAILABLE FOR THE C-SPY MACRAIGOR DRIVER

--mac_handler_address Specifies the location of the debug handler used by Intel XScale devices.

--mac_interface Specifies the communication between the Macraigor debug probe and the target system.

--mac_jtag_device Selects the device corresponding to the hardware interface.

--mac_multiple_targets Specifies the device to connect to, if there are more than one device on the JTAG scan chain.

--mac_reset_pulls_reset Makes C-SPY generate an initial hardware reset.

--mac_set_temp_reg_buffer Provides the driver with a physical RAM address for accessing the coprocessor.

--mac_speed Sets the JTAG speed between the JTAG probe and the ARM JTAG ICE port.

--mac_xscale_ir7 Specifies that the XScale ir7 architecture is used.

OPTIONS AVAILABLE FOR THE C-SPY RDI DRIVER

--rdi_allow_hardware_reset Performs a hardware reset.

--rdi_driver_dll Specifies the path to the driver DLL file.

--rdi_step_max_one Executes one instruction.

OPTIONS AVAILABLE FOR THE C-SPY ST-LINK DRIVER

--stlink_interface Specifies the communication between the ST-LINK debug probe and the target system.

--stlink_reset_strategy Specifies the reset strategy to use.
OPTIONS AVAILABLE FOR THE C-SPY THIRD-PARTY DRIVERS

For information about any options specific to the third-party driver you are using, see its documentation.

Reference information on C-SPY command line options

This section gives detailed reference information about each cspybat option and each option available to the C-SPY drivers.

-B

Syntax

-B

Applicability

All C-SPY drivers.

Description

Use this option to enable batch mode.

--backend

Syntax

--backend {driver options}

Parameters

driver options

Any option available to the C-SPY driver you are using.

Applicability

Sent to cspybat (mandatory).

Description

Use this option to send options to the C-SPY driver. All options that follow --backend will be passed to the C-SPY driver, and will not be processed by cspybat itself.

--code_coverage_file

Syntax

--code_coverage_file file

Parameters

file

The name of the destination file for the code coverage information.

Applicability

Sent to cspybat.
Description
Use this option to enable the generation of code coverage information. The code coverage information will be generated after the execution has completed and you can find it in the specified file.

Note that this option requires that the C-SPY driver you are using supports code coverage. If you try to use this option with a C-SPY driver that does not support code coverage, an error message will be directed to stderr.

See also
Code coverage, page 265.

--cycles

Syntax
--cycles cycles

Parameters
cycles
The number of cycles to run.

Applicability
Sent to cspybat.

Description
Use this option to specify the maximum number of cycles to run. If the target program executes longer than the number of cycles specified, the target program will be aborted.

Using this option requires that the C-SPY driver you are using supports a cycle counter, and that it can be sampled while executing.

--device

Syntax
--device=device_name

Parameters
device_name
The name of the device, for example, ADuC7030, AT91SAM7S256, LPC2378, STR912PM44, or TMS470R1B1M.

Applicability
All C-SPY drivers.

Description
Use this option to specify the name of the device.

To set related option, choose:
Project>Options>General Options>Target>Device
--disable_interrupts

Syntax
--disable_interrupts

Applicability
The C-SPY Simulator driver.

Description
Use this option to disable the interrupt simulation.

To set this option, choose Simulator>Interrupt Setup and deselect the Enable interrupt simulation option.

--download_only

Syntax
--download_only

Applicability
Sent to cspybat.

Description
Use this option to download the code image without starting a debug session afterwards.

To set related options, choose:

Project>Download

--drv_catch_exceptions

Syntax
--drv_catch_exceptions=value
Parameters

value

(value) A value in the range of 0–0x1FF. Each bit specifies which exception to catch:

(Bit 0 = Reset)

(Bit 1 = Undefined instruction)

(Bit 2 = SWI)

(Bit 3 = Not used)

(Bit 4 = Data abort)

(Bit 5 = Prefetch abort)

(Bit 6 = IRQ)

(Bit 7 = FIQ)

(Bit 8 = Other errors)

(value) A value in the range of 0–0x7FF. Each bit specifies which exception to catch:

(Bit 0 = CORERESET - Reset Vector)

(Bit 4 = MMERR - Memory Management Fault)

(Bit 5 = NOCPERR - Coprocessor Access Error)

(Bit 6 = CHKERR - Checking Error)

(Bit 7 = STATERR - State Error)

(Bit 8 = BUSERR - Bus Error)

(Bit 9 = INTERR - Interrupt Service Errors)

(Bit 10 = HARDERR - Hard Fault)

Applicability The C-SPY Angel debug monitor driver.
The C-SPY I-jet/JTAGjet driver
The C-SPY J-Link/J-Trace driver
The C-SPY RDI driver.

Description Use this option to make the application stop when a certain exception occurs.

See also Breakpoints on exception vectors, page 124.
Reference information on C-SPY command line options

For the C-SPY Angel debug monitor driver, use:

**Project>Options>Debugger>Extra Options**

For the C-SPY I-jet/JTAGjet driver and the C-SPY J-Link/J-Trace driver, use:

**Project>Options>Debugger>Driver>Breakpoints>Catch exceptions**

For the C-SPY RDI driver, use:

**Project>Options>Debugger>RDI>Catch exceptions**

---

**--drv_communication**

**Syntax**

--drv_communication=connection

**Parameters**

Where connection is one of these for the C-SPY Angel debug monitor driver:

**Via Ethernet**

UDP:ip_address

UDP:ip_address,port

UDP:hostname

UDP:hostname,port

**Via serial port**

port:baud,parity,stop_bit,handshake

port = COM1-COM256 (default COM1)

baud = 9600, 19200, 38400, 57600, or 115200 (default 9600 baud)

parity = N (no parity)

stop_bit = 1 (one stop bit)

handshake = NONE or RTSCTS (default NONE for no handshaking)

For example, COM1:9600,N,8,1,NONE.

Where connection is one of these for the C-SPY GDB Server driver:
Via Ethernet

TCP/IP: ip_address
TCP/IP: ip_address,port
TCP/IP: hostname
TCP/IP: hostname,port

Note that if no port is specified, port 3333 is used by default.

Where connection is one of these for the C-SPY IAR ROM-monitor driver:

Via serial port

port: baud, parity, stop_bit, handshake
port = COM1–COM256 (default COM1)
baud = 9600, 19200, 38400, 57600, or 115200 (default 9600 baud)
parity = N (no parity)
stop_bit = 1 (one stop bit)
handshake = NONE or RTSCTS (default NONE for no handshaking)

For example, COM1:9600,N,8,1,NONE.

Where connection is one of these for the C-SPY J-Link/J-Trace driver:

Via USB directly to J-Link

USB0–USB3

When using USB0 and if there are more than one J-Link debug probes on the USB connection, a dialog box is displayed when the debug session starts. Use the dialog box to choose which J-Link debug probe to connect to.

USB:#number, connects to the J-Link with the serial number number on the USB connection
Reference information on C-SPY command line options

Via J-Link on LAN

TCP/IP:

When the colon sign is not followed by any address, host name, or serial number, the J-Link driver searches for all J-Link debug probes on the local network and displays them in a dialog box where you can choose which one to connect to (Auto detect).

TCP/IP:ip_address
TCP/IP:ip_address,port
TCP/IP:hostname
TCP/IP:hostname,port
TCP/IP:#{number}, connects to the J-Link with the serial number number on the local network

Note that if no port is specified, port 19020 is used by default.

Where connection is one of these for the C-SPY Macraigor driver:

For mpDemon

port:baud
port = COM1–COM4
baud = 9600, 19200, 38400, 57600, or 115200 (default 9600 baud)

For mpDemon

TCP/IP:ip_address
TCP/IP:ip_address,port
TCP/IP:hostname
TCP/IP:hostname,port

Note that if no port is specified, port 19020 is used by default.

Via USB to usbDemon and usb2Demon

USB ports = USB0–USB3

Applicability

The C-SPY Angel debug monitor driver
The C-SPY GDB Server driver
The C-SPY IAR ROM-monitor driver
The C-SPY J-Link/J-Trace driver
The C-SPY Macraigor driver.

Description
Use this option to choose communication link.

Project>Options>Debugger>Angel>Communication
Project>Options>Debugger>GDB Server>TCP/IP address or hostname [port]
Project>Options>Debugger>IAR ROM-monitor>Communication
Project>Options>Debugger>J-Link/J-Trace>Connection>Communication
To set related options for the C-SPY Macraigor driver, choose:
Project>Options>Debugger>Macraigor

--drv_communication_log

Syntax
--drv_communication_log=filename

Parameters
filename The name of the log file.

Applicability
All C-SPY drivers.

Description
Use this option to log the communication between C-SPY and the target system to a file. To interpret the result, detailed knowledge of the communication protocol is required.

Project>Options>Debugger>Driver>Log communication

--drv_default_breakpoint

Syntax
--drv_default_breakpoint={0|1|2}

Parameters
0 Auto (default)
1 Hardware
2 Software

Applicability
The C-SPY GDB Server driver
Reference information on C-SPY command line options

The C-SPY I-jet/JTAGjet driver
The C-SPY J-Link/J-Trace driver
The C-SPY Macraigor driver.

Description
Use this option to select the type of breakpoint resource to be used when setting a breakpoint.

See also
Default breakpoint type, page 136.

Project>Options>Debugger>Driver>Breakpoints>Default breakpoint type

`--drv_reset_to_cpu_start`

Syntax

`--drv_reset_to_cpu_start`

Applicability

The C-SPY Angel debug monitor driver
The C-SPY GDB Server driver
The C-SPY J-Link/J-Trace driver
The C-SPY TI Stellaris driver
The C-SPY TI XDS100 driver
The C-SPY Macraigor driver
The C-SPY RDI driver
The C-SPY ST-LINK driver.

Description
Use this option to omit setting the PC when starting or resetting the debugger. Instead PC will have the original value set by the CPU, which is the address of the application entry point.

To set this option, use Project>Options>Debugger>Extra Options.
--drv_restore_breakpoints

Syntax
--drv_restore_breakpoints=location

Parameters

location Address or function name label

Applicability
The C-SPY GDB Server driver
The C-SPY I-jet/JTAGjet driver
The C-SPY J-Link/J-Trace driver
The C-SPY Macraigor driver.

Description
Use this option to restore automatically any software breakpoints that were overwritten during system startup.

See also
Restore software breakpoints at, page 136.

Project>Options>Debugger>Driver>Breakpoints>Restore software breakpoints at

--drv_swo_clock_setup

Syntax
--drv_swo_clock_setup=frequency, autodetect, wanted

Parameters

frequency The exact clock frequency used by the internal processor clock, HCLK, in Hz. This value is used for configuring the SWO communication speed and for calculating timestamps.

autodetect 0, Specify the wanted frequency using the parameter wanted. 1, Automatically uses the highest possible frequency that the J-Link debug probe can handle.

wanted The frequency to be used, if autodetect is 0, in Hz. Use wanted if data packets are lost during transmission.

Applicability
The J-Link driver and the ST-LINK driver.
Reference information on C-SPY command line options

**Description**

Use this option to set up the CPU clock. If this option is not used, the CPU clock frequency is by default set to 72 MHz.

- **J-Link>SWO Configuration>CPU clock**
- **J-Link>SWO Configuration>SWO clock>Autodetect**
- **J-Link>SWO Configuration>SWO clock>Wanted**

---

**--drv_vector_table_base**

**Syntax**

```
--drv_vector_table_base=expression
```

**Parameters**

- `expression`: A label or an address

**Applicability**

- The C-SPY GDB Server driver
- The C-SPY I-jet/JTAGjet driver
- The C-SPY J-Link/J-Trace driver
- The C-SPY TI Stellaris driver
- The C-SPY TI XDS100 driver
- The C-SPY Macraigor driver
- The C-SPY RDI driver
- The C-SPY ST-LINK driver
- The C-SPY Simulator driver.

**Description**

Use this option for Cortex-M to specify the location of the reset vector and the initial stack pointer value. This is useful if you want to override the default `__vector_table` label—defined in the system startup code—in the application or if the application lacks this label, which can be the case if you debug code that is built by tools from another vendor.

To set this option, use **Project>Options>Debugger>Extra Options**.
--flash_loader

Syntax
--flash_loader filename

Parameters
filename
The flash loader specification XML file, with the filename extension board.

Applicability
Sent to cspybat.

Description
Use this option to specify a flash loader specification xml file which contains all relevant information about the flash loading. There can be more than one such argument, in which case each argument will be processed in the specified order, resulting in several flash programming passes.

See also
The IAR Embedded Workbench flash loader User Guide.

--gdbserv_exec_command

Syntax
--gdbserv_exec_command="string"

Parameters
"string"
String or command sent to the GDB Server; see its documentation for more information.

Applicability
The C-SPY GDB Server driver.

Description
Use this option to send strings or commands to the GDB Server.

Project>Options>Debugger>Extra Options

--jet_cpu_clock

Syntax
--jet_cpu_clock=frequency

Parameters
frequency
The clock frequency in Hz

Applicability
The C-SPY I-jet/JTAGjet driver.
Description
Use this option to specify the exact clock frequency used by the internal processor clock, HCLK. This value is used for configuring the SWO communication speed and for calculating timestamps.

Note: This option is relevant only when the option --jet_swo_protocol is set to UART.

Project>Options>Debugger>I-jet/JTAGjet>SWO>Clock setup>CPU clock

--jet_interface

Syntax
--jet_interface=JTAG|SWD

Parameters

JTAG
Specifies the JTAG interface

SWD
Specifies the SWD interface

Applicability
The C-SPY I-jet/JTAGjet driver.

Description
Use this option to specify the communication interface between the debug probe and the target system.

The SWD interface uses fewer pins than JTAG. Specify --jet_interface if you want to use the serial-wire output (SWO) communication channel. Alternatively, you can set this option to JTAG and also specify the --jet_swo_on_d0 option.

Note that if you select stdout/stderr via SWO on the General Options>Library Configuration page, SWD is selected automatically.

See also
SWO Trace Window Settings dialog box, page 199 for more information about SWO settings.

Project>Options>Debugger>I-jet/JTAGjet>JTAG/SWD>Interface>SWD

--jet_ir_length

Syntax
--jet_ir_length=length

C-SPY® Debugging Guide
for ARM

368
Parameters

*length* The number of IR bits preceding the ARM core to connect to, for JTAG scan chains that mix ARM devices with other devices.

Applicability The C-SPY I-jet/JTAGjet driver.

Description Use this option to set the number of IR bits preceding the ARM core to connect to.

See also *JTAG scan chain*, page 392

**--jet_jtag_speed**

Syntax

```
--jet_jtag_speed=[auto|adaptive|number]
```

Parameters

*auto* (default) Automatically uses the highest possible frequency for reliable operation (default)

*adaptive* For ARM devices that have the *RTCK* JTAG signal available.

*number* The JTAG and SWD communication speed in kHz.

If there are JTAG communication problems or problems in writing to target memory (for example during program download), these problems might be resolved if the speed is set to a lower frequency.

Applicability The C-SPY I-jet/JTAGjet driver.

Description Use this option to specify the JTAG and SWD communication speed in kHz.

If this option is not specified, the highest possible frequency is automatically used.

**--jet_power_from_probe**

Syntax

```
--jet_power_from_probe=[leave_on|switch_off]
```

Applicability The C-SPY I-jet/JTAGjet driver.

Description Use this option to set the power source for the ARM core.
Reference information on C-SPY command line options

Parameters

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>leave_on</td>
<td>Continues to supply power to the target even after the debug session has been stopped.</td>
</tr>
<tr>
<td>switch_off</td>
<td>Turns off the power to the target when the debug session stops.</td>
</tr>
</tbody>
</table>

Applicability

The C-SPY I-jet/JTAGjet driver.

Description

Use this option to specify the status of the probe power supply after debugging. If this option is not specified, the probe will not supply power to the board.

Syntax

--jet_script_file=path

Parameters

path

The path to the script file.

Applicability

The C-SPY I-jet/JTAGjet driver.

Description

Use this option to specify the device script file that describes the reset to be performed.

Note: This option can only be used in combination with the --jet_script_reset option. You can use either these two options or the --jet_standard_reset option.

See also

--jet_script_reset, page 370 and --jet_standard_reset, page 371.

This option is not available in the IDE.

Syntax

--jet_script_reset=macro

Parameters

macro

The name of a reset macro within the script file that is specified with the option --jet_script_file.
The C-SPY Command Line Utility—cspybat

Applicability
The C-SPY I-jet/JTAGjet driver.

Description
Use this option to specify the name of the reset macro that is used for resetting the target CPU.

Note: This option can only be used in combination with the --jet_script_file option. You can use either these two options or the --jet_standard_reset option.

See also

This option is not available in the IDE.

--jet_standard_reset

Syntax
--jet_standard_reset=type,duration,delay

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>The reset strategy. Choose between:</td>
</tr>
<tr>
<td></td>
<td>0, reset disabled</td>
</tr>
<tr>
<td></td>
<td>1, software reset</td>
</tr>
<tr>
<td></td>
<td>2, hardware reset</td>
</tr>
<tr>
<td></td>
<td>3, core reset</td>
</tr>
<tr>
<td></td>
<td>4, system reset</td>
</tr>
<tr>
<td>duration</td>
<td>The time in milliseconds that the hardware reset asserts the</td>
</tr>
<tr>
<td></td>
<td>reset signal (line nSRST/nRESET) low to reset the device.</td>
</tr>
<tr>
<td></td>
<td>Some devices might require a longer reset signal than the default 200 ms.</td>
</tr>
<tr>
<td></td>
<td>This parameter applies to the hardware reset, and to those</td>
</tr>
<tr>
<td></td>
<td>custom reset strategies that use the hardware reset.</td>
</tr>
<tr>
<td>delay</td>
<td></td>
</tr>
</tbody>
</table>
Reference information on C-SPY command line options

```
delay

The delay time, in milliseconds, after the reset signal has been de-asserted, before the debugger attempts to control the processor.

The processor might be kept internally in reset for some time after the external reset signal has been de-asserted, thus inaccessible for the debugger.

This parameter applies to the Hardware reset, and to those custom reset strategies that use the Hardware reset.
```

**Applicability**

The C-SPY I-jet/JTAGjet driver.

**Description**

Use this option to select the reset strategy to be used when the debugger starts. Note that Cortex-M uses a different set of strategies than other devices.

You can use this option or the two options `--jet_script_file` and `--jet_script_reset`.

**See also**

`--jet_script_file`, page 370 and `--jet_script_reset`, page 370.

Project>Options>Debugger>I-jet/JTAGjet>Setup>Reset

---

**--jet_swo_on_d0**

**Syntax**

`--jet_swo_on_d0`

**Applicability**

The C-SPY I-jet/JTAGjet driver.

**Description**

Use this option to specify that SWO trace data is output on the trace data pin D0. When using this option, both the SWD and the JTAG interface can handle SWO trace data.

Project>Options>Debugger>I-jet/JTAGjet>SWO>SWO on the TraceD0 pin

---

**--jet_swo_prescaler**

**Syntax**

`--jet_swo_prescaler=number`

**Parameters**

`number`

The prescaler value, 1–100, which in turn determines the CPU clock frequency.
The C-SPY Command Line Utility—cspybat

**Applicability**
The C-SPY I-jet/JTAGjet driver.

**Description**
Use this option to specify the prescaler for the SWO clock. The CPU clock frequency is divided by the number specified as the prescaler. If data packets are lost during transmission, try using a higher prescaler value.

If this option is not specified, a prescaler value is set automatically. This automatically set value is the highest possible frequency that the debug probe can handle.

I-jet/JTAGjet>SWO Configuration>SWO prescaler

---

**--jet_swo_protocol**

**Syntax**
```
--jet_swo_protocol={auto|Manchester|UART}
```

**Parameters**
- **auto**
  - Automatically selects the communication protocol.
- **Manchester**
  - Specifies the Manchester protocol.
- **UART**
  - Specifies the UART protocol.

**Applicability**
The C-SPY I-jet/JTAGjet driver.

**Description**
Use this option to specify the communication protocol for the SWO channel. If this option is not specified, `auto` is automatically used.

Project>Options>Debugger>I-jet/JTAGjet>SWO>Protocol

---

**--jet_tap_position**

**Syntax**
```
--jet_tap_position=<tap_number>
```

**Parameters**
- **<tap_number>**
  - The TAP position of the device you want to connect to.

**Applicability**
The C-SPY I-jet/JTAGjet driver.

**Description**
If there is more than one device on the JTAG scan chain, use this option to select a specific device.
Reference information on C-SPY command line options

See also

\textit{JTAG scan chain}, page 392.

----

\textbf{--jlink_device_select}

\textbf{Syntax}

\texttt{--jlink_device_select=\textit{tap_number}}

\textbf{Parameters}

\textit{tap_number}

The TAP position of the device you want to connect to.

\textbf{Applicability}

The C-SPY J-Link/J-Trace driver.

\textbf{Description}

If there is more than one device on the JTAG scan chain, use this option to select a specific device.

\textbf{See also}

\textit{JTAG scan chain}, page 402.

----

\textbf{--jlink_exec_command}

\textbf{Syntax}

\texttt{--jlink_exec_command=cmdstr1; cmdstr2; cmdstr3 ...}

\textbf{Parameters}

\textit{cmdstrn}

\textit{J-Link/J-Trace command string.}

\textbf{Applicability}

The C-SPY J-Link/J-Trace driver.

\textbf{Description}

Use this option to make the debugger call the \texttt{__jlinkExecCommand} macro with one or several command strings, after target connection has been established.

\textbf{See also}

\textit{__jlinkExecCommand}, page 319.

----
---jlink_initial_speed

Syntax
--jlink_initial_speed=speed

Parameters
speed

Description
The initial communication speed in kHz. If no speed is specified, 32 kHz will be used as the initial speed.

See also
JTAG/SWD speed, page 399.
Project>Options>Debugger>J-Link/J-Trace>Setup>JTAG speed>Fixed

---jlink_interface

Syntax
--jlink_interface={JTAG|SWD}

Parameters
JTAG
SWD

Description
Use this option to specify the communication channel between the J-Link debug probe and the target system.

See also
Interface, page 402.
Project>Options>Debugger>J-Link/J-Trace>Connection>Interface

---jlink_ir_length

Syntax
--jlink_ir_length=length
Reference information on C-SPY command line options

Parameters

length

The number of IR bits preceding the ARM core to connect to, for JTAG scan chains that mix ARM devices with other devices.

Applicability

The C-SPY J-Link/J-Trace driver.

Description

Use this option to set the number of IR bits preceding the ARM core to connect to.

See also

JTAG scan chain, page 402.

Project>Options>Debugger>J-Link/J-Trace>Connection>JTAG scan chain>Preceding bits

--jlink_reset_strategy

Syntax

--jlink_reset_strategy=delay,strategy

Parameters

delay

For Cortex-M and ARM 7/9/11 with strategies 1-9, delay should be 0 (ignored). For ARM 7/9/11 with strategy 0, the delay should be one of 0-10000.

strategy

For information about supported reset strategies, see the IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores.

Applicability

The C-SPY J-Link/J-Trace driver.

Description

Use this option to select the reset strategy to be used at debugger startup.

See also

Reset, page 396.

Project>Options>Debugger>J-Link/J-Trace>Setup>Reset

--jlink_script_file

Syntax

--jlink_script_file=filename

Parameters

filename

The name of the J-Link script file.
Applicability: The C-SPY J-Link/J-Trace driver.

Description: Use this option to specify the J-Link script file to be used.

J-Link has a script language that can be used for setting up hardware. For certain targets, ready-made script files are automatically pointed out by IAR Embedded Workbench. In command line mode, the script file needs to be manually specified by using this option.

See also: The J-Link/J-Trace ARM User Guide (JLinkARM.pdf, document number UM08001), section 5.10, for a detailed description of the script language.

To set this option using a non-predefined script file, use Project>Options>Debugger>Extra Options.

--jlink_speed

Syntax: --jlink_speed={fixed|auto|adaptive}

Parameters:
- **fixed**: 1–12000
- **auto**: The highest possible frequency for reliable operation (default)
- **adaptive**: For ARM devices that have the RTCK JTAG signal available

Applicability: The C-SPY J-Link/J-Trace driver.

Description: Use this option to set the JTAG communication speed in kHz.

See also: JTAG/SWD speed, page 414.

Project>Options>Debugger>J-Link/J-Trace>Setup>JTAG speed

--jlink_trace_source

Syntax: --jlink_trace_source={ETB|ETM}

Parameters:
- **ETB**: Selects ETB trace.
- **ETM**: Selects ETM trace.

Applicability: The C-SPY J-Link/J-Trace driver.
Reference information on C-SPY command line options

**Description**

Use this option to select either ETB or ETM as the trace source.

**Note:** This option applies only to J-Trace.

**See also**

*ETM/ETB*, page 400.

Project>Options>Debugger>J-Link/J-Trace>Setup>ETM/ETB

---

**--lmiftdi_speed**

**Syntax**

```
--lmiftdi_speed=frequency
```

**Parameters**

frequency  The frequency in kHz.

**Applicability**

The C-SPY TI Stellaris driver.

**Description**

Use this option to set the JTAG communication speed in kHz.

**See also**

*JTAG/SWD speed*, page 414.

Project>Options>Debugger>TI Stellaris>Setup>JTAG speed

---

**--macro**

**Syntax**

```
--macro filename
```

**Parameters**

filename  The C-SPY macro file to be used (filename extension `.mac`).

**Applicability**

Sent to `cspybat`.

**Description**

Use this option to specify a C-SPY macro file to be loaded before executing the target application. This option can be used more than once on the command line.

**See also**

*Briefly about using C-SPY macros*, page 294.

---

**--mac_handler_address**

**Syntax**

```
--mac_handler_address=address
```

---
Parameters

address  The start address of the memory area for the debug handler.

Applicability  The C-SPY Macraigor driver.

Description  Use this option to specify the location—the memory address—of the debug handler used by Intel XScale devices.

See also  Debug handler address, page 409.

Project>Options>Debugger>Macraigor>Debug handler address

--mac_interface

Syntax  --mac_interface={JTAG|SWD}

Parameters

JTAG  Uses JTAG communication with the target system (default).

SWD  Uses SWD communication with the target system (Cortex-M only); uses fewer pins than JTAG communication.

Applicability  The C-SPY Macraigor driver.

Description  Use this option to specify the communication channel between the Macraigor debug probe and the target system.

Project>Options>Debugger>Macraigor>Interface

--mac_jtag_device

Syntax  --mac_jtag_device=device

Parameters

device  The device corresponding to the hardware interface that is used. Choose between Macraigor mpDemon, usbDemon, and usb2Demon.

Applicability  The C-SPY Macraigor driver.
Reference information on C-SPY command line options

--mac_multiple_targets

Description
Use this option to select the device corresponding to the hardware interface that is used.

See also
OCD interface device, page 407.

Project>Options>Debugger>Macraigor>OCD interface device

Syntax
--mac_multiple_targets=<tap-no>@dev0,dev1,dev2,dev3,...

Parameters

tap-no The TAP number of the device to connect to, where 0 connects to the first device, 1 to the second, and so on.

dev0–devn The nearest TDO pin on the Macraigor JTAG probe.

Applicability
The C-SPY Macraigor driver.

Description
If there is more than one device on the JTAG scan chain, each device must be defined. Use this option to specify which device you want to connect to.

Example
--mac_multiple_targets=0@ARM7TDMI,ARM7TDMI

See also
JTAG scan chain with multiple targets, page 408.

Project>Options>Debugger>Macraigor>JTAG scan chain with multiple targets

--mac_reset_pulls_reset

Syntax
--mac_reset_pulls_reset=time

Parameters

time 0–2000 which is the delay in milliseconds after reset.

Applicability
The C-SPY Macraigor driver.

Description
Use this option to make C-SPY perform an initial hardware reset when the debugger is started, and to specify the delay for the reset.

See also
Hardware reset, page 408.
The C-SPY Command Line Utility—cspybat

--mac_set_temp_reg_buffer

Syntax

```
--mac_set_temp_reg_buffer=address
```

Parameters

- `address`: The start address of the RAM area.

Applicability

The C-SPY Macraigor driver.

Description

Use this option to specify the start address of the RAM area that is used for controlling the MMU and caching via the CP15 coprocessor.

To set this option, use `Project>Options>Debugger>Extra Options`.

--mac_speed

Syntax

```
--mac_speed={factor}
```

Parameters

- `factor`: The factor by which the JTAG probe clock is divided when generating the scan clock. The number must be in the range 1–8 where 1 is the fastest.

Applicability

The C-SPY Macraigor driver.

Description

Use this option to set the JTAG speed between the JTAG probe and the ARM JTAG ICE port.

See also

*JTAG speed*, page 407.

To set this option, use `Project>Options>Debugger>Macraigor>JTAG speed`.

--mac_xscale_ir7

Syntax

```
--mac_xscale_ir7
```
Applicability: The C-SPY Macraigor driver.

Description: Use this option to specify that the XScale ir7 core is used, instead of XScale ir5. Note that this option is mandatory when using the XScale ir7 core.

These XScale cores are supported by the C-SPY Macraigor driver:
- Intel XScale Core 1 (5-bit instruction register—ir5)
- Intel XScale Core 2 (7-bit instruction register—ir7)

To set this option, use Project>Options>Debugger>Extra Options.

**--mapu**

Syntax: --mapu

Applicability: The C-SPY simulator driver.

Description: Specify this option to use the section information in the debug file for memory access checking. During the execution, the simulator will then check for accesses to unspecified memory ranges. If any such access is found, the C function call stack and a message will be printed on stderr and the execution will stop.

See also: Memory access checking, page 156.

To set related options, choose: Simulator>Memory Access Setup

**-p**

Syntax: -p filename

Parameters:
- **filename**
  - The device description file to be used.

Applicability: All C-SPY drivers.

Description: Use this option to specify the device description file to be used.

See also: Selecting a device description file, page 53.
**--plugin**

**Syntax**

\[ \text{--plugin filename} \]

**Parameters**

`filename`

The plugin file to be used (filename extension `.dll`).

**Applicability**

Sent to \texttt{cspybat}.

**Description**

Certain C/C++ standard library functions, for example \texttt{printf}, can be supported by C-SPY—for example, the C-SPY Terminal I/O window—instead of by real hardware devices. To enable such support in \texttt{cspybat}, a dedicated plugin module called located in the `arm\bin` directory must be used.

Use this option to include this plugin during the debug session. This option can be used more than once on the command line.

**Note:** You can use this option to include also other plugin modules, but in that case the module must be able to work with \texttt{cspybat} specifically. This means that the C-SPY plugin modules located in the `common\plugin` directory cannot normally be used with \texttt{cspybat}.

**--proc_stack_stack**

**Syntax**

\[ \text{--proc_stack_stack=startaddress, endaddress} \]

where `stack` is one of `main` or `proc` for Cortex-M and

where `stack` is one of `usr`, `svc`, `irq`, `fig`, `und`, or `abt` for other ARM cores

**Parameters**

`startaddress`

The start address of the stack, specified either as a value or as an expression.

`endaddress`

The end address of the stack, specified either as a value or as an expression.

**Applicability**

All C-SPY drivers. Note that this command line option is only available when using C-SPY from the IDE; not in batch mode using \texttt{cspybat}.

**Description**

Use this option to provide information to the C-SPY stack plugin module about reserved stacks. By default, C-SPY receives this information from the system startup code, but if you for some reason want to override the default values, this option can be useful.
Example:  
--proc_stack_irq=0x8000,0x80FF

To set this option, use Project>Options>Debugger>Extra Options.

--rdi_allow_hardware_reset
Syntax:  
--rdi_allow_hardware_reset
Applicability:  
The C-SPY RDI driver.
Description:  
Use this option to allow the emulator to perform a hardware reset of the target. Requires support by the emulator.
See also:  
Allow hardware reset, page 409.

--rdi_driver_dll
Syntax:  
--rdi_driver_dll filename
Parameters:  
filename:  
The file or path to the driver DLL file.
Applicability:  
The C-SPY RDI driver
Description:  
Use this option to specify the path to the driver DLL file provided with the JTAG pod.
See also:  
Manufacturer RDI driver, page 409.

--rdi_step_max_one
Syntax:  
--rdi_step_max_one
Applicability:  
The C-SPY Angel debug monitor driver
The C-SPY RDI driver.

C-SPY® Debugging Guide
for ARM

AFE1_AFE2-1:1
Description

Use this option to execute only one instruction. The debugger will turn off interrupts while stepping and, if necessary, simulate the instruction instead of executing it.

To set this option, use Project>Options>Debugger>Extra Options.

--semihosting

Syntax

--semihosting={none|iar_breakpoint}

Parameters

No parameter Use standard semihosting.
none Does not use semihosted I/O.
iar_breakpoint Uses the IAR proprietary semihosting variant.

Applicability

All C-SPY drivers.

Description

Use this option to enable semihosted I/O and to choose the kind of semihosting interface to use. Note that if this option is not used, semihosting will by default be enabled and C-SPY will try to choose the correct semihosting mode automatically. This means that normally you do not have to use this option if your application is linked with semihosting.

To make semihosting work, your application must be linked with a semihosting library.

See also

The IAR C/C++ Development Guide for ARM for more information about linking with semihosting.

Project>Options>General Options>Library Configuration

--silent

Syntax

--silent

Applicability

Sent to cspybat.

Description

Use this option to omit the sign-on message.
Reference information on C-SPY command line options

--stlink_interface

Syntax

--stlink_interface={JTAG|SWD}

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTAG</td>
<td>Uses JTAG communication with the target system (default).</td>
</tr>
<tr>
<td>SWD</td>
<td>Uses SWD communication with the target system.</td>
</tr>
</tbody>
</table>

Applicability

The C-SPY ST-LINK driver.

Description

Use this option to specify the communication channel between the ST-LINK debug probe and the target system.

See also

Interface, page 411.

Project>Options>Debugger>ST-LINK>ST-LINK>Interface

--stlink_reset_strategy

Syntax

--stlink_reset_strategy=delay, strategy

Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>delay</td>
<td>The delay time measured in milliseconds. delay is ignored and should be 0.</td>
</tr>
<tr>
<td>strategy</td>
<td>The reset strategy.</td>
</tr>
<tr>
<td>0</td>
<td>(Normal) performs the standard reset procedure.</td>
</tr>
<tr>
<td>1</td>
<td>(Reset Pin) uses the reset pin to perform a hardware reset. Only available for ST-LINK version 2.</td>
</tr>
<tr>
<td>2</td>
<td>(Connect during reset) ST-LINK connects to the target while keeping Reset active (Reset is pulled low and remains low while connecting to the target). Only available for ST-LINK version 2.</td>
</tr>
</tbody>
</table>

Applicability

The C-SPY ST-LINK driver.

Description

Use this option to select the reset strategy to be used at debugger startup.

See also

Reset, page 411
--timeout

Syntax

--timeout milliseconds

Parameters

milliseconds

The number of milliseconds before the execution stops.

Applicability

Sent to cspybat.

Description

Use this option to limit the maximum allowed execution time.

This option is not available in the IDE.

--xds_rootdir

Syntax

--xds_rootdir=path

Applicability

The C-SPY TI XDS100 driver

Description

Use this option to specify the path to the directory where the TI XDS100 driver package is installed.

To set this option, use Project>Options>Debugger>Extra Options.
Reference information on C-SPY command line options
Debugger options

This chapter describes the C-SPY® options available in the IAR Embedded Workbench® IDE. More specifically, this means:

- Setting debugger options
- Reference information on debugger options
- Reference information on C-SPY hardware driver options.

Setting debugger options

Before you start the C-SPY debugger you might need to set some options—both C-SPY generic options and options required for the target system (C-SPY driver-specific options). This section gives detailed information about the options in the Debugger category.

To set debugger options in the IDE:

1. Choose Project>Options to display the Options dialog box.
2. Select Debugger in the Category list.
   For more information about the generic options, see Reference information on debugger options, page 391.
3. On the Setup page, select the appropriate C-SPY driver from the Driver drop-down list.
4. To set the driver-specific options, select the appropriate driver from the Category list. Depending on which C-SPY driver you are using, different options are available.

<table>
<thead>
<tr>
<th>C-SPY driver</th>
<th>Available options pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>C-SPY Angel debug monitor driver</td>
<td>Angel, page 397</td>
</tr>
<tr>
<td>C-SPY GDB Server driver</td>
<td>GDB Server, page 398</td>
</tr>
<tr>
<td></td>
<td>Breakpoints options, page 145</td>
</tr>
<tr>
<td>C-SPY IAR ROM-monitor driver</td>
<td>IAR ROM-monitor, page 399</td>
</tr>
</tbody>
</table>

Table 43: Options specific to the C-SPY drivers you are using
To restore all settings to the default factory settings, click the Factory Settings button.

When you have set all the required options, click OK in the Options dialog box.
Reference information on debugger options

This section gives reference information on C-SPY debugger options.

Setup

The Setup options select the C-SPY driver, the setup macro file, and device description file to use, and specify which default source code location to run to.

Driver

Selects the C-SPY driver for the target system you have.

Run to

Specifies the location C-SPY runs to when the debugger starts after a reset. By default, C-SPY runs to the main function.

To override the default location, specify the name of a different location you want C-SPY to run to. You can specify assembler labels or whatever can be evaluated as such, for example function names.

If the option is deselected, the program counter will contain the regular hardware reset address at each reset.

Setup macros

Registers the contents of a setup macro file in the C-SPY startup sequence. Select Use macro file and specify the path and name of the setup file, for example SetupSimple.mac. If no extension is specified, the extension .mac is assumed. A browse button is available for your convenience.

It is possible to specify up to two different macro files.
Device description file

A default device description file—either an IAR-specific ddf file or a CMSIS System View Description file—is selected automatically based on your project settings. To override the default file, select **Override default** and specify an alternative file. A browse button is available for your convenience.

For information about the device description file, see *Modifying a device description file*, page 57.

IAR-specific device description files for each arm device are provided in the directory arm\config and have the filename extension ddf.

Download

By default, C-SPY downloads the application to RAM or flash when a debug session starts. The **Download** options let you modify the behavior of the download.

![Figure 132: C-SPY Download options](image)

**Attach to program**

Makes the debugger attach to a running application at its current location, without resetting or halting (for J-Link only) the target system. To avoid unexpected behavior when using this option, the **Debugger>Setup** option **Run to** should be deselected.

**Verify download**

Verifies that the downloaded code image can be read back from target memory with the correct contents.
**Suppress download**

Disables the downloading of code, while preserving the present content of the flash. This command is useful if you want to debug an application that already resides in target memory.

If this option is combined with the *Verify download* option, the debugger will read back the code image from non-volatile memory and verify that it is identical to the debugged application.

**Note:** It is important that the image that resides in target memory is linked consistently with how you use C-SPY for debugging. This applies, for example, if you first link your application using an output format without debug information, such as Intel-hex, and then load the application separately from C-SPY. If you then use C-SPY only for debugging without downloading, you cannot build the debugged application with any of the options *Semihosted* or *IAR breakpoint*—on the General Options>Library Configuration page—as that would add extra code, resulting in two different code images.

**Use flash loader(s)**

Use this option to use one or several flash loaders for downloading your application to flash memory. If a flash loader is available for the selected chip, it is used by default. Press the *Edit* button to display the Flash Loader Overview dialog box.

For more information about flash loaders, see *Flash loaders*, page 437.

**Override default .board file**

A default flash loader is selected based on your choice of device on the General Options>Target page. To override the default flash loader, select *Override default .board file* and specify the path to the flash loader you want to use. A browse button is available for your convenience. Click *Edit* to display the Flash Loader Overview dialog box. For more information, see *Flash Loader Overview dialog box*, page 439.
Images

The Images options control the use of additional debug files to be downloaded.

![Image Options](image)

**Note:** Flash loading will not be performed; you can only download images to RAM using the Images options.

**Download extra Images**

Controls the use of additional debug files to be downloaded:

- **Path**
  Specify the debug file to be downloaded. A browse button is available for your convenience.

- **Offset**
  Specify an integer that determines the destination address for the downloaded debug file.

- **Debug info only**
  Makes the debugger download only debug information, and not the complete debug file.

If you want to download more than three images, use the related C-SPY macro, see `__loadImage`, page 325.

For more information, see *Loading multiple images*, page 56.
Plugins

The Plugins options select the C-SPY plugin modules to be loaded and made available during debug sessions.

**Select plugins to load**

Selects the plugin modules to be loaded and made available during debug sessions. The list contains the plugin modules delivered with the product installation.

**Description**

Describes the plugin module.

**Location**

Informs about the location of the plugin module.

Generic plugin modules are stored in the `common\plugins` directory. Target-specific plugin modules are stored in the `arm\plugins` directory.

**Originator**

Informs about the originator of the plugin module, which can be modules provided by IAR Systems or by third-party vendors.

**Version**

Informs about the version number.
Extra Options

The Extra Options page provides you with a command line interface to C-SPY.

Figure 135: Debugger extra options

Use command line options

Specify additional command line arguments to be passed to C-SPY (not supported by the GUI).

Reference information on C-SPY hardware driver options

This section gives reference information on C-SPY hardware driver options.

- Angel, page 397
- GDB Server, page 398
- IAR ROM-monitor, page 399
- Setup options for I-jet/JTAGjet, page 400
- JTAG/SWD options for I-jet/JTAGjet, page 403
- SWO options for I-jet/JTAGjet, page 404
- Setup options for J-Link/J-Trace, page 406
- Connection options for J-Link/J-Trace, page 410
- Macraigor, page 412
- RDI, page 414
- ST-LINK, page 416
- Setup options for TI Stellaris, page 417
**Angel**

The Angel options control the C-SPY Angel debug monitor driver.

![C-SPY Angel options](image)

**Send heartbeat**

Makes C-SPY poll the target system periodically while your application is running. That way, the debugger can detect if the target application is still running or has terminated abnormally. Enabling the heartbeat will consume some extra CPU cycles from the running program.

**Communication**

Selects the Angel communication link. RS232 serial port connection and TCP/IP via an Ethernet connection are supported.

**TCP/IP**

Specify the IP address of the target device in the text box.

**Serial port settings**

Configures the serial port. You can specify

- **Port**: Selects which port on the host computer to use as the Angel communication link.
- **Baud rate**: Sets the communication speed.

The initial Angel serial speed is always 9600 baud. After the initial handshake, the link speed is changed to the specified speed. Communication problems can occur at very high speeds; some Angel-based evaluation boards will not work above 38400 baud.
Log communication

Logs the communication between C-SPY and the target system to a file. To interpret the result, a detailed knowledge of the Angel monitor protocol is required.

GDB Server

The GDB Server options control the C-SPY GDB Server for the STR9-comStick evaluation board.

TCP/IP address or hostname

Specify the IP address and port number of a GDB server; by default the port number 3333 is used. The TCP/IP connection is used for connecting to a J-Link server running on a remote computer.

Log communication

Logs the communication between C-SPY and the target system to a file. To interpret the result, a detailed knowledge of the JTAG interface is required.
IAR ROM-monitor

The IAR ROM-monitor options control the C-SPY IAR ROM-monitor interface.

![IAR ROM-monitor options](image)

**Serial port settings**

Configures the serial port. You can specify

- **Port**: Selects which port on the host computer to use as the ROM-monitor communication link.
- **Baud rate**: Sets the communication speed. The serial port communication link speed must match the speed selected on the target board.

**Log communication**

Logs the communication between C-SPY and the target system to a file. To interpret the result, a detailed knowledge of the ROM-monitor protocol is required.
Setup options for I-jet/JTAGjet

The Setup options control the I-jet in-circuit debugging probe and the JTAGjet debug probe.

**Figure 139: I-jet/JTAGjet Setup options**

**Reset**

Selects the reset strategy to be used when the debugger starts. Note that Cortex-M uses a different set of strategies than other devices. Based on your hardware, one of the strategies is the default. Choose between:

- **Disabled (no reset)**: No reset is performed.
- **Software**
  - Sets PC to the program entry address.
  - This is a software reset.
- **Hardware**
  - The probe toggles the nSRST/nRESET line on the JTAG connector to reset the device. This reset usually also resets the peripherals. The reset pulse timing is controlled by the **Duration** and **Delay after** options.
  - The processor should stop at the reset handler before executing any instruction. Some processors might not stop at the reset vector, but will be halted soon after, executing some instructions.
- **Core**
  - Resets the core via the VECTRESET bit; the peripheral units are not affected. For Cortex-M devices only.
- **System**
  - Resets the core and peripherals. For Cortex-M devices only.
Debugger options

All of these strategies are available for both the JTAG and the SWD interface, and all strategies halt the CPU after the reset.

A software reset of the target does not change the settings of the target system; it only resets the program counter and the mode register CPSR to its reset state. For some ARM9, ARM11, and Cortex-A devices, it also resets the CP15 system control coprocessor, effectively disabling the virtual memory (MMU), caches and memory protection.

 Normally, a C-SPY reset is a software reset only. If you use the Hardware option, C-SPY will generate an initial hardware reset when the debugger is started. This is performed
Reference information on C-SPY hardware driver options

once before download, and if the option Use flash loader(s) is selected, also once after flash download, see Debugging code in flash, page 60, and Debugging code in RAM, page 61.

Hardware resets can be a problem if the low-level setup of your application is not complete. If the low-level setup does not set up memory configuration and clocks, the application will not work after a hardware reset. To handle this in C-SPY, the setup macro function execUserReset() is suitable. For a similar example where execUserPreload() is used, see Remapping memory, page 58.

Duration

The time in milliseconds that the hardware reset asserts the reset signal (line nSRST/nRESET) low to reset the device.

Some devices might require a longer reset signal than the default 200 ms.

This option applies to the hardware reset, and to those custom reset strategies that use the hardware reset.

Delay after

The delay time, in milliseconds, after the reset signal has been de-asserted, before the debugger attempts to control the processor.

The processor might be kept internally in reset for some time after the external reset signal has been de-asserted, thus inaccessible for the debugger.

This option applies to the hardware reset, and to those custom reset styles that use the hardware reset.

Target power supply

If power for the target system is supplied from the probe, this option specifies the status of the power supply after debugging. Choose between:

- **Leave on after debugging** Continues to supply power to the target even after the debug session has been stopped.
- **Switch off after debugging** Turns off the power to the target when the debug session stops.

Log communication

Logs trace output, that is a sequence of internal communication activities, between C-SPY and its lower-level interfaces to hardware. This log is primarily useful as a troubleshooting help when contacting IAR Systems support.
JTAG/SWD options for I-jet/JTAGjet

The JTAG/SWD options specify the interface between I-jet or JTAGjet and the target system.

![Figure 140: I-jet/JTAGjet JTAG/SWD options](image)

### Interface

Selects the communication interface between the debug probe and the target system. Choose between:

- **JTAG**
  - Uses the JTAG interface.

- **SWD**
  - Uses the SWO interface, which uses fewer pins than JTAG. Select SWD if you want to use the serial-wire output (SWO) communication channel. Note that if you select `stdout/stderr via SWO` on the General Options>Library Configuration page, SWD is selected automatically. For more information about SWO settings, see `SWO Trace Window Settings dialog box`, page 199.

### JTAG scan chain

Specifies the JTAG scan chain. Choose between:

- **JTAG scan chain with multiple targets**
  - Specifies that there is more than one device on the JTAG scan chain.

- **TAP number**
  - Specify the TAP (Test Access Port) position of the device you want to connect to. The TAP numbers start from zero.

- **Scan chain contains non-ARM devices**
  - Enables JTAG scan chains that mix ARM devices with other devices like, for example, FPGA.
Reference information on C-SPY hardware driver options

**Preceding bits**
Specifying the number of IR bits before the ARM device to be debugged.

**JTAG/SWD speed**
Specify the JTAG and SWD communication speed. Choose between:
- **Auto detect**
  Automatically uses the highest possible frequency for reliable operation.
- **Adaptive**
  Synchronizes the clock to the processor clock outside the core. Works only with ARM devices that have the R[CK] JTAG signal available.
- **n MHz**
  Sets the JTAG and SWD communication speed to the selected frequency.
  If there are JTAG communication problems or problems in writing to target memory (for example during program download), these problems might be resolved if the speed is set to a lower frequency.

**Probe configuration file**
Specify a probe configuration file to use instead of the default probe configuration file that comes with your product package.

**SWO options for I-jet/JTAGjet**
The SWO options specify the serial-wire output for I-jet/JTAGjet.

![SWO options for I-jet/JTAGjet](image)

*Figure 141: I-jet/JTAGjet SWO options*
Debugger options

Protocol
Specifies the communication protocol for the SWO channel. Choose between:

- **Auto**: Automatically selects the best possible protocol and speed, depending on the device you are using.
- **Manchester**: Specifies the Manchester protocol.
- **UART**: Specifies the UART protocol.

Clock setup
Specifies the CPU clock. Choose between:

- **CPU clock**: Specifies the exact clock frequency used by the internal processor clock, HCLK, in MHz. The value can have decimals. This value is used for configuring the SWO communication speed.
- **SWO prescaler**: Specifies the clock prescaler of the SWO communication channel in KHz. The prescaler, in turn, determines the CPU clock frequency.
- **Auto**: Automatically uses the highest possible frequency that the J-Link debugging probe can handle. Use this option if data packets are lost during transmission.

To override the Clock setup options, use the Override project default option in the SWO Configuration dialog box, see Override project default, page 182.

SWO on the TraceD0 pin
Specifies that SWO trace data is output on the trace data D0 pin. When using this option, both the SWD and the JTAG interface can handle SWO trace data.

Note that both the device and the board you are using must support this pin.
Setup options for J-Link/J-Trace

The Setup options specify the J-Link/J-Trace probe.

**Reset**

Selects the reset strategy to be used when the debugger starts. Note that Cortex-M uses a different set of strategies than other devices. The actual reset strategy type number is specified for each available choice. Choose between:

- **Normal** (0, default) Tries to reset the core via the reset strategy Core and peripherals first. If this fails, the reset strategy Core only is used. It is recommended that you use this strategy to reset the target.

- **Core** (1) Resets the core via the VECTRESET bit; the peripheral units are not affected.

- **Core and peripherals** (8) Resets the core and the peripherals.

- **Reset Pin** (2) J-Link pulls its RESET pin low to reset the core and the peripheral units. Normally, this causes the CPU RESET pin of the target device to go low as well, which results in a reset of both the CPU and the peripheral units.

- **Connect during reset** (3) J-Link connects to the target while keeping Reset active (reset is pulled low and remains low while connecting to the target). This is the recommended reset strategy for STM32 devices. This strategy is available for STM32 devices only.
Debugger options

Halt after bootloader (4 or 7) NXP Cortex-M0 devices. This is the same strategy as the Normal strategy, but the target is halted when the bootloader has finished executing. This is the recommended reset strategy for LPC11xx and LPC13xx devices.

Analog Devices Cortex-M3 devices (7), Resets the core and peripheral units by setting the SYSRESETREQ bit in the AIRCR. The core is allowed to perform the ADI kernel (which enables the debug interface), but the core is halted before the first instruction after the kernel is executed to guarantee that no user application code is performed after reset.

Halt before bootloader (5) This is the same strategy as the Normal strategy, but the target is halted before the bootloader has started executing. This strategy is normally not used, except in situations where the bootloader needs to be debugged. This strategy is available for LPC11xx and LPC13xx devices only.

Normal, disable watchdog (6, 9, or 10) First performs a Normal reset, to reset the core and peripheral units and halt the CPU immediately after reset. After the CPU is halted, the watchdog is disabled, because the watchdog is by default running after reset. If the target application does not feed the watchdog, J-Link loses connection to the device because it is permanently reset. This strategy is available for Freescale Kinetis devices (6), for NXP LPC 1200 devices (9), and for Samsung S3FN60D devices (10).

All of these strategies are available for both the JTAG and the SWD interface, and all strategies halt the CPU after the reset.

For other cores, choose between these strategies:

Hardware, halt after delay (ms) (0) Specify the delay between the hardware reset and the halt of the processor. This is used for making sure that the chip is in a fully operational state when C-SPY starts to access it. By default, the delay is set to zero to halt the processor as quickly as possible.

This is a hardware reset.
### Reference information on C-SPY hardware driver options

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hardware, halt using</td>
<td><strong>Breakpoint</strong> (1) After reset, J-Link continuously tries to halt the CPU using a breakpoint. Typically, this halts the CPU shortly after reset; the CPU can in most systems execute some instructions before it is halted. This is a hardware reset.</td>
</tr>
<tr>
<td></td>
<td><strong>Hardware, halt at 0</strong> (4) Halts the processor by placing a breakpoint at the address zero. Note that this is not supported by all ARM microcontrollers. This is a hardware reset.</td>
</tr>
<tr>
<td></td>
<td><strong>Hardware, halt using DBGRQ</strong> (5) After reset, J-Link continuously tries to halt the CPU using DBGRQ. Typically, this halts the CPU shortly after reset; the CPU can in most systems execute some instructions before it is halted. This is a hardware reset.</td>
</tr>
<tr>
<td></td>
<td><strong>Software</strong> (-) Sets PC to the program entry address. This is a software reset.</td>
</tr>
<tr>
<td></td>
<td><strong>Software, Analog devices</strong> (2) Uses a reset sequence specific for the Analog Devices ADuC7xxx family. This strategy is only available if you have selected such a device from the Device drop-down list on the General Options&gt;Target page. This is a software reset.</td>
</tr>
<tr>
<td></td>
<td><strong>Hardware, NXP LPC</strong> (9) This strategy is only available if you have selected such a device from the Device drop-down list on the General Options&gt;Target page. This is a hardware reset specific to NXP LPC devices.</td>
</tr>
<tr>
<td></td>
<td><strong>Hardware, Atmel AT91SAM7</strong> (8) This strategy is only available if you have selected such a device from the Device drop-down list on the General Options&gt;Target page. This is a hardware reset specific for the Atmel AT91SAM7 family.</td>
</tr>
</tbody>
</table>

For more details about the different reset strategies, see the *IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores* available in the arm\doc directory.

A software reset of the target does not change the settings of the target system; it only resets the program counter and the mode register CPSR to its reset state. Normally, a
C-SPY reset is a software reset only. If you use the Hardware reset option, C-SPY will generate an initial hardware reset when the debugger is started. This is performed once before download, and if the option Use flash loader(s) is selected, also once after flash download, see Debugging code in flash, page 60, and Debugging code in RAM, page 61.

Hardware resets can be a problem if the low-level setup of your application is not complete. If the low-level setup does not set up memory configuration and clocks, the application will not work after a hardware reset. To handle this in C-SPY, the setup macro function execUserReset() is suitable. For a similar example where execUserPreload() is used, see Remapping memory, page 58.

**JTAG/SWD speed**

Specify the JTAG communication speed in kHz. Choose between:

- **Auto**
  - Automatically uses the highest possible frequency for reliable operation. The initial speed is the fixed frequency used until the highest possible frequency is found. The default initial frequency—32 kHz—can normally be used, but in cases where it is necessary to halt the CPU after the initial reset, in as short time as possible, the initial frequency should be increased.

  A high initial speed is necessary, for example, when the CPU starts to execute unwanted instructions—for example power down instructions—from flash or RAM after a reset. A high initial speed would in such cases ensure that the debugger can quickly halt the CPU after the reset.

  The initial value must be in the range 1–12000 kHz.

- **Fixed**
  - Sets the JTAG communication speed in kHz. The value must be in the range 1–12000 kHz.

  If there are JTAG communication problems or problems in writing to target memory (for example during program download), these problems might be resolved if the speed is set to a lower frequency.

- **Adaptive**
  - Synchronizes the clock to the processor clock outside the core. Works only with ARM devices that have the RTCK JTAG signal available. For more information about adaptive speed, see the IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores available in the arm\doc directory.
Reference information on C-SPY hardware driver options

Clock setup

Specifies the CPU clock. Choose between:

**CPU clock**

Specifies the exact clock frequency used by the internal processor clock, HCLK, in MHz. The value can have decimals. This value is used for configuring the SWO communication speed and for calculating timestamps.

**SWO clock**

Specifies the clock frequency of the SWO communication channel in KHz.

**Auto**

Automatically uses the highest possible frequency that the J-Link debug probe can handle. If Auto is not selected, the wanted SWO clock value can be input in the text box. The value can have decimals. Use this option if data packets are lost during transmission.

To override the Clock setup options, use the Override project default option in the SWO Configuration dialog box, see Override project default, page 182.

**ETM/ETB**

The Prefer ETB option selects ETB trace instead of ETM trace, which is the default.

**Note:** This option applies only to J-Trace.

**Connection options for J-Link/J-Trace**

The Connection options specify the connection with the J-Link/J-Trace probe.

![Connection options](image)

Figure 143: J-Link/J-Trace Connection options
Debugger options

Communication
Selects the communication channel between C-SPY and the J-Link debug probe. Choose between:

- **USB**: Selects the USB connection. If Serial number is selected in the drop-down list, the J-Link debug probe with the specified serial number is chosen.

- **TCP/IP**: Specify the IP address of a J-Link server. The TCP/IP connection is used for connecting to a J-Link server running on a remote computer.
  - **IP address**: specify the IP address of a J-Link probe connected to LAN.
  - **Auto detect**: automatically scans the network for J-Link probes. Use the dialog box to choose among the detected J-Link probes.
  - **Serial number**: connects to the J-Link probe on the network with the serial number that you specify.

Interface
Selects the communication interface between the J-Link debug probe and the target system. Choose between:

- **JTAG** (default): Uses the JTAG interface.

- **SWD**: Uses fewer pins than JTAG. Select SWD if you want to use the serial-wire output (SWO) communication channel. Note that if you select stdout/stderr via SWO on the General Options>Library Configuration page, SWD is selected automatically. For more information about SWO settings, see SWO Trace Window Settings dialog box, page 199.

JTAG scan chain
Specifies the JTAG scan chain. Choose between:

- **JTAG scan chain with multiple targets**: Specifies that there is more than one device on the JTAG scan chain.

- **TAP number**: Specify the TAP (Test Access Port) position of the device you want to connect to. The TAP numbers start from zero.
Reference information on C-SPY hardware driver options

Scan chain contains non-ARM devices

Enables JTAG scan chains that mix ARM devices with other devices like, for example, FPGA.

Preceding bits

Specify the number of IR bits before the ARM device to be debugged.

Log communication

Logs the communication between C-SPY and the target system to a file. To interpret the result, a detailed knowledge of the JTAG interface is required.

Macraigor

The Macraigor options specify the Macraigor interface.

**OCD interface device**

Selects the device corresponding to the hardware interface you are using. Supported Macraigor JTAG probes is Macraigor mpDemon.

**Interface**

Selects the communication interface between the J-Link debug probe and the target system. Choose between:

- **JTAG (default)**: Uses the JTAG interface.
- **Scan chain contains non-ARM devices**
- **Preceding bits**
Debugger options

SWD
Uses fewer pins than JTAG. Select SWD if you want to use the serial-wire output (SWO) communication channel. Note that if you select stdout/stderr via SWO on the General Options>Library Configuration page, SWD is selected automatically. For more information about SWO settings, see SWO Trace Window Settings dialog box, page 199.

JTAG speed
Specify the speed between the JTAG probe and the ARM JTAG ICE port. The number must be in the range 1–8 and sets the factor by which the JTAG probe clock is divided when generating the scan clock.

The mpDemon interface might require a higher setting such as 2 or 3, that is, a lower speed.

TCP/IP
Specify the IP address of a JTAG probe connected to the Ethernet/LAN port.

Port
Selects which serial port or parallel port on the host computer to use as communication link. Select the host port to which the JTAG probe is connected.

In the case of parallel ports, you should normally use LPT1 if the computer is equipped with a single parallel port. Note that a laptop computer might in some cases map its single parallel port to LPT2 or LPT3. If possible, configure the parallel port in EPP mode because this mode is fastest; bidirectional and compatible modes will work but are slower.

Baud rate
Selects the serial communication speed.

Hardware reset
Generates an initial hardware reset when the debugger is started. This is performed once before download, and if the option Use flash loader(s) is selected, also once after flash download, see Debugger startup when debugging code in flash, page 53, and Figure 6, Debugger startup when debugging code in RAM.

A software reset of the target does not change the settings of the target system; it only resets the program counter to its reset state. Normally, a C-SPY reset is a software reset only.

Hardware resets can be a problem if the low-level setup of your application is not complete. If low-level setup does not set up memory configuration and clocks, the
application will not work after a hardware reset. To handle this in C-SPY, the setup macro function `execUserReset()` is suitable. For a similar example where `execUserPreload()` is used, see Remapping memory, page 58.

**JTAG scan chain with multiple targets**
Defines each device on the JTAG scan chain, if there is more than one. Also, you must state which device you want to connect to. The syntax is:

```
<0> @dev0, dev1, dev2, dev3, ...
```

where 0 is the TAP number of the device to connect to, and dev0 is the nearest TDO pin on the Macraigor JTAG probe.

**Debug handler address**
Specify the location—the memory address—of the debug handler used by Intel XScale devices. To save memory space, you should specify an address where a small portion of cache RAM can be mapped, which means the location should not contain any physical memory. Preferably, find an unused area in the lower 16-Mbyte memory and place the handler address there.

**Log communication**
Logs the communication between C-SPY and the target system to a file. To interpret the result, a detailed knowledge of the JTAG interface is required.

**RDI**
With the RDI options you can use JTAG interfaces compliant with the ARM Ltd. RDI 1.5.1 specification. One example of such an interface is the ARM RealView Multi-ICE JTAG interface.

![RDI options](image)

Figure 145: RDI options
Manufacturer RDI driver

Specify the file path to the RDI driver DLL file provided with the JTAG pod.

Allow hardware reset

Allows the emulator to perform a hardware reset of the target.

A software reset of the target does not change the settings of the target system; it only resets the program counter to its reset state.

You should only allow hardware resets if the low-level setup of your application is complete. If the low-level setup does not set up memory configuration and clocks, the application will not work after a hardware reset. To handle this in C-SPY, the setup macro function execUserReset() is suitable. For a similar example where execUserPreload() is used, see Remapping memory, page 58.

Note: This option requires that hardware resets are supported by the RDI driver you are using.

Catch exceptions

Causes exceptions to be treated as breakpoints. Instead of handling the exception as defined by the running program, the debugger will stop.

The ARM core exceptions that can be caught are:

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reset</td>
<td>Reset</td>
</tr>
<tr>
<td>Undef</td>
<td>Undefined instruction</td>
</tr>
<tr>
<td>SWI</td>
<td>Software interrupt</td>
</tr>
<tr>
<td>Data</td>
<td>Data abort (data access memory fault)</td>
</tr>
<tr>
<td>Prefetch</td>
<td>Prefetch abort (instruction fetch memory fault)</td>
</tr>
<tr>
<td>IRQ</td>
<td>Normal interrupt</td>
</tr>
<tr>
<td>FIQ</td>
<td>Fast interrupt</td>
</tr>
</tbody>
</table>

Table 44: Catching exceptions

Log RDI communication

Logs the communication between C-SPY and the target system to a file. To interpret the result, detailed knowledge of the RDI interface is required.
ST-LINK

The ST-LINK page contains options for the ST-LINK probe.

Reset

Selects the reset strategy to be used when the debugger starts. The actual reset strategy type number is specified for each available choice. Choose between:

- **Normal** (0) Performs the standard reset procedure.
- **Reset Pin** (1) Uses the reset pin to perform a hardware reset. Only available for ST-LINK version 2.
- **Connect during reset** (2) ST-LINK connects to the target while keeping the reset pin active (the reset pin is pulled low and remains low while connecting to the target). Only available for ST-LINK version 2.

Interface

Selects the communication interface between the ST-LINK debug probe and the target system. Choose between:

- **JTAG** (default) Uses the JTAG interface.
- **SWD** Uses fewer pins than JTAG.
Clock setup

Specifies the CPU clock. Choose between:

**CPU clock**

Specifies the exact clock frequency used by the internal processor clock, $h_{CLK}$, in MHz. The value can have decimals. This value is used for configuring the SWO communication speed and for calculating timestamps.

**SWO clock**

Specifies the clock frequency of the SWO communication channel in KHz.

**Auto**

Automatically uses the highest possible frequency that the J-Link debug probe can handle. If Auto is not selected, the wanted SWO clock value can be input in the text box. The value can have decimals. Use this option if data packets are lost during transmission.

To override the Clock setup options, use the Override project default option in the SWO Configuration dialog box, see Override project default, page 197.

Setup options for TI Stellaris

The Setup options specify the TI Stellaris interface.

**Interface**

Selects the communication interface between the J-Link debug probe and the target system. Choose between:

**JTAG** (default) Uses the JTAG interface.

**SWO**

Specifies the clock frequency of the SWO communication channel in KHz.

**Auto**

Automatically uses the highest possible frequency that the J-Link debug probe can handle. If Auto is not selected, the wanted SWO clock value can be input in the text box. The value can have decimals. Use this option if data packets are lost during transmission.
Reference information on C-SPY hardware driver options

**JTAG/SWD speed**
Specify the JTAG communication speed in kHz.

**Log communication**
Logs the communication between C-SPY and the target system to a file. To interpret the result, a detailed knowledge about the communication protocol is required.

**Setup options for TI XDS100**
The **Setup** options control the TI XDS100 interface.

![TI XDS100 Setup options](image)

**Emulator**
Specify the emulator you are using. To override the default board file, choose **Custom board file** and specify a board file.

**TI emulation package installation path**
Select **Override default** to override the default installation path of the Texas Instruments emulation package.

**SWD**
Uses fewer pins than JTAG. Select SWD if you want to use the serial-wire output (SWO) communication channel. Note that if you select stdout/stderr via SWO on the General Options>Library Configuration page, SWD is selected automatically. For more information about SWO settings, see SWO Trace Window Settings dialog box, page 199.
Third-Party Driver options

The Third-Party Driver options are used for loading any driver plugin provided by a third-party vendor. These drivers must be compatible with the C-SPY debugger driver specification.

![Figure 149: C-SPY Third-Party Driver options](image)

In addition to the options you can set here, you can set options for the third-party driver using the **Project>Options>Debugger>Extra Options** page.

**IAR debugger driver plugin**

Specify the file path to the third-party driver plugin DLL file. A browse button is available for your convenience.

**Suppress download**

Disables the downloading of code, while preserving the present content of the flash. This command is useful if you want to debug an application that already resides in target memory.

If this option is combined with the **Verify download** option, the debugger will read back the code image from non-volatile memory and verify that it is identical to the debugged application.

**Note:** It is important that the image that resides in target memory is linked consistently with how you use C-SPY for debugging. This applies, for example, if you first link your application using an output format without debug information, such as Intel-hex, and then load the application separately from C-SPY. If you then use C-SPY only for debugging without downloading, you cannot build the debugged application with any of the options **Semihosted** or **IAR breakpoint**—on the **General Options>Library Configuration** page—as that would add extra code, resulting in two different code images.
Reference information on C-SPY hardware driver options

Verify all
Verifies that the downloaded code image can be read back from target memory with the correct contents.

Log communication
Logs the communication between C-SPY and the target system to a file. To interpret the result, detailed knowledge of the interface is required. This option can be used if is supported by the third-party driver.
Additional information on C-SPY drivers

This chapter describes the additional menus and features provided by the C-SPY® drivers. You will also find some useful hints about resolving problems.

Reference information on the C-SPY simulator

This section gives additional reference information the C-SPY simulator, reference information not provided elsewhere in this documentation.

More specifically, this means:

- Simulator menu, page 421

Simulator menu

When you use the simulator driver, the Simulator menu is added to the menu bar.

*Figure 150: Simulator menu*
Menu commands

These commands are available on the menu:

**Interrupt Setup**
Displays a dialog box where you can configure C-SPY interrupt simulation, see *Interrupt Setup dialog box*, page 280.

**Forced Interrupts**
Opens a window from where you can instantly trigger an interrupt, see *Forced Interrupt window*, page 284.

**Interrupt Status**
Opens a window from where you can instantly trigger an interrupt, see *Interrupt Status window*, page 285.

**Interrupt Log**
Opens a window which displays the status of all defined interrupts, see *Interrupt Log window*, page 287.

**Interrupt Log Summary**
Opens a window which displays a summary of the status of all defined interrupts, see *Interrupt Log Summary window*, page 291.

**Data Log**
Opens a window which logs accesses to up to four different memory locations or areas, see *Data Log window*, page 112.

**Data Log Summary**
Opens a window which displays a summary of data accesses to specific memory location or areas, see *Data Log Summary window*, page 114.

**Memory Access Setup**
Displays a dialog box to simulate memory access checking by specifying memory areas with different access types, see *Memory Access Setup dialog box*, page 184.

**Trace**
Opens a window which displays the collected trace data, see *Trace window*, page 205.

**Function Trace**
Opens a window which displays the trace data for function calls and function returns, see *Function Trace window*, page 209.
Additional information on C-SPY drivers

Function Profiler
Opens a window which shows timing information for the functions, see Function Profiler window, page 245.

Timeline
Opens a window which shows trace data for interrupt logs and for the call stack, see Timeline window, page 209.

Breakpoint Usage
Displays a window which lists all active breakpoints, see Breakpoint Usage window, page 133.

The C-SPY GDB Server driver
This section gives additional reference information on the C-SPY GDB Server driver, reference information not provided elsewhere in this documentation.

More specifically, this means:
● GDB Server menu, page 423.

GDB Server menu
When you are using the C-SPY GDB Server driver, the GDB Server menu is added to the menu bar.

Menu commands
These commands are available on the menu:

Breakpoint Usage
Opens a window which lists all active breakpoints; see Breakpoint Usage window, page 133.

The C-SPY I-jet/JTAGjet driver
This section gives additional reference information on the C-SPY I-jet/JTAGjet driver, reference information not provided elsewhere in this documentation.
More specifically, this means:

- **I-jet/JTAGjet menu**, page 424.

**I-jet/JTAGjet menu**

When you are using the C-SPY I-jet/JTAGjet driver, the **I-jet/JTAGjet menu** is added to the menu bar.

![Menu commands](image)

**Menu commands**

These commands are available on the menu:

**Memory Configuration**

- Displays a dialog box; see **Memory Configuration dialog box**, page 179.
Disable Debugger Cache
Disables memory caching and memory range checking in C-SPY.

Disable Interrupts When Stepping
Ensures that only the stepped instructions will be executed. Interrupts will not be executed. This command can be used when not running at full speed and some interrupts interfere with the debugging process.

ETM Trace Settings
Displays a dialog box; see ETM Trace Settings dialog box (I-jet/JTAGjet), page 195.

ETM Trace Save
Displays a dialog box; see Trace Save dialog box, page 238.

ETM Trace
Opens the ETM Trace window; see Trace window, page 205.

Function Trace
Opens a window; see Function Trace window, page 209.

SWO Configuration
Displays a dialog box; see SWO Configuration dialog box, page 201.

SWO Trace Window Settings
Displays a dialog box; see SWO Trace Window Settings dialog box, page 199.

SWO Trace
Opens the SWO Trace window to display the collected trace data; see Trace window, page 205.

Interrupt Log
Opens a window; see Interrupt Log window, page 287.

Interrupt Log Summary
Opens a window; see Interrupt Log Summary window, page 291.

Data Log
Opens a window; see Data Log window, page 112.

Data Log Summary
Opens a window; see Data Log Summary window, page 114.

Event Log
Opens a window; see Event Log window, page 116.
Event Log Summary
Opens a window; see Event Log Summary window, page 117.

Power Log Setup
Opens a window; see Power Log Setup window, page 260.

Power Log
Opens a window; see Power Log window, page 262.

Vector Catch
Displays a dialog box for setting a breakpoint directly on a vector in the interrupt vector table, see Breakpoints on exception vectors, page 124. Note that this command is not available for all ARM cores.

Timeline¹
Opens a window; see Timeline window, page 209.

Function Profiler
Opens a window which shows timing information for the functions; see Function Profiler window, page 245.

Session Overview
Displays a window that lists information about the debug session, such as details about project settings, session settings, and the session state. To save the contents of the window to a file, choose Save As from the context menu.

Breakpoint Usage
Opens a window which lists all active breakpoints; see Breakpoint Usage window, page 133.

EmuDiag
Starts the EmuDiag application where you can diagnose the connection between the host computer, the probe, and the board.

¹ Only available when the SWD/SWO interface is used.

2 Normally, C-SPY uses the memory range information in the Memory Configuration dialog box both to restrict access to certain parts of target memory and to cache target memory contents for improved C-SPY performance. Under certain rare circumstances, this is not appropriate, and you can choose Disable Debugger Cache to turn off the caching and memory range checking completely. All accesses from C-SPY will then result in corresponding accesses to the target system. Some of those circumstances are:

- When memory is remapped at runtime and cannot be specified as a fixed set of ranges.
- When the memory range setup is incorrect or incomplete.
The C-SPY J-Link/J-Trace driver

This section gives additional reference information for the C-SPY J-Link/J-Trace driver, reference information not provided elsewhere in this documentation.

More specifically, this means:

- **Live watch and use of DCC**, page 429
- **Terminal I/O and use of DCC**, page 430

### J-Link menu

When you are using the C-SPY J-Link driver, the **J-Link** menu is added to the menu bar.

![J-Link menu](image)

**Figure 153: The J-Link menu**

### Menu commands

These commands are available on the menu:

**Watchpoints**

Displays a dialog box for setting watchpoints, see *Code breakpoints dialog box*, page 134.
Vector Catch
Displays a dialog box for setting a breakpoint directly on a vector in the interrupt vector table, see Breakpoints on exception vectors, page 124. Note that this command is not available for all ARM cores.

Disable Interrupts When Stepping
Ensures that only the stepped instructions will be executed. Interrupts will not be executed. This command can be used when not running at full speed and some interrupts interfere with the debugging process.

ETM Trace Settings
Displays a dialog box to configure ETM trace data generation and collection; see ETM Trace Settings dialog box (J-Link/J-Trace), page 197.

ETM Trace Save
Displays a dialog box to save the collected trace data to a file; see Trace Save dialog box, page 238.

ETM Trace
Opens the ETM Trace window to display the collected trace data; see Trace window, page 205.

Function Trace
Opens a window which displays the trace data for function calls and function returns; see Function Trace window, page 209.

SWO Configuration
Displays a dialog box; see SWO Configuration dialog box, page 201.

SWO Trace Window Settings
Displays a dialog box; see SWO Trace Window Settings dialog box, page 199.

SWO Trace Save
Displays a dialog box to save the collected trace data to a file; see Trace Save dialog box, page 238.

SWO Trace
Opens the SWO Trace window to display the collected trace data; see Trace window, page 205.

Interrupt Log
Opens a window; see Interrupt Log window, page 287.

Interrupt Log Summary
Opens a window; see Interrupt Log Summary window, page 291.
Additional information on C-SPY drivers

Data Log
Opens a window; see Data Log window, page 112.

Data Log Summary
Opens a window; see Data Log Summary window, page 114.

Power Log Setup
Opens a window; see Power Log Setup window, page 260.

Power Log
Opens a window; see Power Log window, page 262.

Timeline
Opens a window; see Timeline window, page 209.

Function Profiler
Opens a window which shows timing information for the functions; see Function Profiler window, page 245.

Breakpoint Usage
Opens a window which lists all active breakpoints; see Breakpoint Usage window, page 133.

1 Only available when using either ETM or J-Link with ETB.
2 Only available when the SWD/SWO interface is used.
3 Available when using either ETM or SWD/SWO.

LIVE WATCH AND USE OF DCC
The following possibilities for using live watch apply:

For Cortex-M
Access to memory or setting breakpoints is always possible during execution. The DCC (Debug Communications Channel) unit is not available.

For ARMxxx-S devices
Setting hardware breakpoints is always possible during execution.

For ARM7/ARM9 devices, including ARMxxx-S
Memory accesses must be made by your application. By adding a small program—a DCC handler—that communicates with the debugger through the DCC unit to your application, memory can be read/written during execution. Software breakpoints can also be set by the DCC handler.
Just add the files `JLINKDCC_Process.c` and `JLINKDCC_HandleDataAbort.s` located in `arm\src\debugger\dcc` to your project and call the `JLINKDCC_Process` function regularly, for example every millisecond.

In your local copy of the `cstartup` file, modify the interrupt vector table so that data aborts will call the `JLINKDCC_HandleDataAbort` handler.

**TERMINAL I/O AND USE OF DCC**

The following possibilities for using Terminal I/O apply:

**For Cortex-M**

See *ITM Stimulus Ports*, page 198.

**For ARM7/ARM9 devices, including ARMxxx-S**

DCC can be used for Terminal I/O output by adding the file `arm\src\debugger\dcc\DCC_Write.c` to your project. `DCC_write.c` overrides the library function `write`. Functions such as `printf` can then be used to output text in real time to the C-SPY Terminal I/O window.

In this case, you can disable semihosting which means that the breakpoint it uses is freed for other purposes. To disable semihosting, choose **General Options>Library Configuration>Library low-level interface implementation>None.**

---

**The C-SPY Macraigor driver**

This section gives additional reference information on the C-SPY Macraigor driver, reference information not provided elsewhere in this documentation.

More specifically, this means:

- **Macraigor JTAG menu**, page 430.

**Macraigor JTAG menu**

When you are using the C-SPY Macraigor driver, the JTAG menu is added to the menu bar.

![Figure 154: The Macraigor JTAG menu](image-url)
These commands are available on the menu:

**Watchpoints**
Opens a dialog box for setting watchpoints, see *Code breakpoints dialog box*, page 134.

**Vector Catch**
Opens a dialog box for setting a breakpoint directly on a vector in the interrupt vector table, see *Breakpoints on exception vectors*, page 124. Note that this command is not available for all ARM cores.

**Breakpoint Usage**
Opens a window which lists all active breakpoints; see *Breakpoint Usage window*, page 133.

---

**The C-SPY RDI driver**

This section gives additional reference information on the C-SPY RDI driver, reference information not provided elsewhere in this documentation.

More specifically, this means:

- **RDI menu**, page 431.

---

**RDI menu**

When you are using the C-SPY RDI driver, the RDI menu is added to the menu bar.

![The RDI menu](image)

*Figure 135: The RDI menu*

These commands are available on the menu:

**Configure**
Opens a dialog box that originates from the RDI driver vendor. For information about details in this dialog box, refer to the driver documentation.

**Trace Settings**
Displays a dialog box to configure the ETM trace; see *ETM Trace Settings dialog box (J-Link/J-Trace)*, page 197.

**Trace Save**
Displays a dialog box to save the collected trace data to a file; see *Trace Save dialog box*, page 238.
The C-SPY ST-LINK driver

This section gives additional reference information on the C-SPY ST-LINK driver, reference information not provided elsewhere in this documentation.

More specifically, this means:

- ST-LINK menu, page 432.

ST-LINK menu

When you are using the C-SPY ST-LINK driver, the ST-LINK menu is added to the menu bar.

These commands are available on the menu:

- **SWO Configuration**
  Displays a dialog box; see SWO Configuration dialog box, page 201.

- **SWO Trace Window Settings**
  Displays a dialog box; see SWO Trace Window Settings dialog box, page 199.

- **SWO Trace Save**
  Displays a dialog box to save the collected trace data to a file; see Trace Save dialog box, page 238.
The C-SPY TI Stellaris driver

This section gives additional reference information on the C-SPY TI Stellaris driver, reference information not provided elsewhere in this documentation.

More specifically, this means:

- **TI Stellaris menu**, page 433.

**TI Stellaris menu**

When you are using the C-SPY TI Stellaris driver, the **TI Stellaris** menu is added to the menu bar.

1 Only available when the SWD/SWO interface is used.
2 Available when using either ETM or SWD/SWO.

---

**SWO Trace**

Opens the SWO Trace window to display the collected trace data; see **Trace window**, page 205.

**Interrupt Log**

Opens a window; see **Interrupt Log window**, page 287.

**Interrupt Log Summary**

Opens a window; see **Interrupt Log Summary window**, page 291.

**Data Log**

Opens a window; see **Data Log window**, page 112.

**Data Log Summary**

Opens a window; see **Data Log Summary window**, page 114.

**Timeline**

Opens a window; see **Timeline window**, page 209.

**Function Profiler**

Opens a window which shows timing information for the functions; see **Function Profiler window**, page 245.

**Breakpoint Usage**

Opens a window which lists all active breakpoints; see **Breakpoint Usage window**, page 133.
The C-SPY TI XDS100 driver

This section gives additional reference information on the C-SPY TI XDS100 driver, reference information not provided elsewhere in this documentation.

More specifically, this means:

- TI XDS100 menu, page 434.

TI XDS100 menu

When you are using the C-SPY TI XDS100 driver, the TI XDS100 menu is added to the menu bar.

Figure 158: The TI XDS100 menu

This command is available on the menu:

Breakpoint Usage... Opens a window which lists all active breakpoints; see Breakpoint Usage window, page 133.

Resolving problems

Debugging using the C-SPY hardware debugger systems requires interaction between many systems, independent from each other. For this reason, setting up this debug system can be a complex task. If something goes wrong, it might be difficult to locate the cause of the problem.

For information about the current debug session, choose Session Overview from the driver menu. Note that this window might not be supported by the C-SPY driver you are using.

This section includes suggestions for resolving the most common problems that can occur when debugging with the C-SPY hardware debugger systems.

For problems concerning the operation of the evaluation board, refer to the documentation supplied with it, or contact your hardware distributor.

SLOW STEPPING SPEED

If you find that the stepping speed is slow, these troubleshooting tips might speed up stepping:

- If you are using a hardware debugger system, keep track of how many hardware breakpoints that are used and make sure some of them are left for stepping.
Stepping in C-SPY is normally performed using breakpoints. When C-SPY performs a step command, a breakpoint is set on the next statement and the application executes until it reaches this breakpoint. If you are using a hardware debugger system, the number of hardware breakpoints—typically used for setting a stepping breakpoint in code that is located in flash/ROM memory—is limited. If you, for example, step into a `switch` statement, breakpoints are set on each branch; this might consume several hardware breakpoints. If the number of available hardware breakpoints is exceeded, C-SPY switches into single stepping on assembly level, which can be very slow.

For more information, see and *Breakpoint consumers*, page 123.

- Disable trace data collection, using the *Enable/Disable* button in both the Trace and the Function Profiling windows. Trace data collection might slow down stepping because the collected trace data is processed after each step. Note that it is not sufficient to just close the corresponding windows to disable trace data collection.

- Choose to view only a limited selection of SFR registers. You can choose between two alternatives. Either type `$SFR_name` (where `SFR_name` reflects the name of the SFR you want to monitor) in the Watch window, or create your own filter for displaying a limited group of SFRs in the Register window. Displaying many SFR registers might slow down stepping because all registers must be read from the hardware after each step. See *Defining application-specific register groups*, page 157.

- Close the Memory and Symbolic Memory windows if they are open, because the visible memory must be read after each step and that might slow down stepping.

- Close any window that displays expressions such as Watch, Live Watch, Locals, Statics if it is open, because all these windows read memory after each step and that might slow down stepping.

- Close the Stack window if it is open. Choose *Tools>*Options>*Stack and disable the *Enable graphical stack display and stack usage tracking* option if it is enabled.

- If possible, increase the communication speed between C-SPY and the target board/emulator.
Flash loaders

This chapter describes the flash loader, what it is and how to use it. More specifically, this means:

- Introduction to the flash loader
- Introduction to the flash loader
- Reference information on the flash loader.

Introduction to the flash loader

A flash loader is an agent that is downloaded to the target. It fetches your application from the debugger and programs it into flash memory. The flash loader uses the file I/O mechanism to read the application program from the host. You can select one or several flash loaders, where each flash loader loads a selected part of your application. This means that you can use different flash loaders for loading different parts of your application.

Flash loaders for various microcontrollers is provided with IAR Embedded Workbench for arm. In addition to these, more flash loaders are provided by chip manufacturers and third-party vendors. The flash loader API, documentation, and several implementation examples are available to make it possible for you to implement your own flash loader.

Using flash loaders

This section describes various tasks related to using flash loaders. More specifically, you will get information about:

- Setting up the flash loader(s)
- The flash loading mechanism

SETTING UP THE FLASH LOADER(S)

To use a flash loader for downloading your application:

1. Choose Project>Options.
2. Choose the Debugger category and click the Download tab.
3 Select the **Use Flash loader(s)** option. A default flash loader configured for the device you have specified will be used. The configuration is specified in a preconfigured board file.

4 To override the default flash loader or to modify the behavior of the default flash loader to suit your board, select the **Override default. board file** option, and **Edit** to open the **Flash Loader Configuration** dialog box. A copy of the *.board file will be created in your project directory and the path to the *.board file will be updated accordingly.

5 The **Flash Loader Overview** dialog box lists all currently configured flash loaders, see *Flash Loader Overview dialog box*, page 439. You can either select a flash loader or open the **Flash Loader Configuration** dialog box.

In the **Flash Loader Configuration** dialog box, you can configure the download. For more information about the various flash loader options, see *Flash Loader Configuration dialog box*, page 440.

**THE FLASH LOADING MECHANISM**

When the **Use flash loader(s)** option is selected and one or several flash loaders have been configured, these steps are performed when the debug session starts.

Steps 1 to 4 are performed for each flash loader in the flash loader configuration.

1 C-SPY downloads the flash loader into target RAM.

Steps 2 to 4 are performed one or more times depending on the size of the RAM and the size of the application image.

2 C-SPY writes code/data from the application image into target RAM (RAM buffer).

3 C-SPY starts execution of the flash loader.

4 The flash loader reads data from the RAM buffer and programs the flash memory.

5 The application image now resides in flash memory and can be started. The flash loader and the RAM buffer are no longer needed, so RAM is fully available to the application in the flash memory.

---

**Reference information on the flash loader**

This section gives reference information about these windows and dialog boxes:

- **Flash Loader Overview dialog box**, page 439
- **Flash Loader Configuration dialog box**, page 440.
Flash Loader Overview dialog box

The Flash Loader Overview dialog box is available from the Debugger>Download page.

![Flash Loader Overview dialog box](image)

This dialog box lists all defined flash loaders. If you have selected a device on the General Options>Target page for which there is a flash loader, this flash loader is by default listed in the Flash Loader Overview dialog box.

The display area

Each row in the display area shows how you have set up one flash loader for flashing a specific part of memory:

**Range**

The part of your application to be programmed by the selected flash loader.

**Offset/Address**

The start of the memory where your application will be flashed. If the address is preceded with a, the address is absolute. Otherwise, it is a relative offset to the start of the memory.

**Loader Path**

The path to the flash loader *.flash file to be used (*.out for old-style flash loaders).

**Extra Parameters**

List of extra parameters that will be passed to the flash loader.

Click on the column headers to sort the list by range, offset/address, etc.
Function buttons

These function buttons are available:

OK
The selected flash loader(s) will be used for downloading your application to memory.

Cancel
Standard cancel.

New
Displays a dialog box where you can specify what flash loader to use, see Flash Loader Configuration dialog box, page 440.

Edit
Displays a dialog box where you can modify the settings for the selected flash loader, see Flash Loader Configuration dialog box, page 440.

Delete
Deletes the selected flash loader configuration.

Flash Loader Configuration dialog box

The Flash Loader Configuration dialog box is available from the Flash Loader Overview dialog box.

![Flash Loader Configuration dialog box](image)

Figure 160: Flash Loader Configuration dialog box
Use the **Flash Loader Configuration** dialog box to configure the download to suit your board. A copy of the default board file will be created in your project directory.

**Memory range**

Specify the part of your application to be downloaded to flash memory. Choose between:

- **All**
  - The whole application is downloaded using this flash loader.
- **Start/End**
  - Specify the start and the end of the memory area for which part of the application will be downloaded.

**Relocate**

Overrides the default flash base address, in other words, relocates the location of the application in memory. This means that you can flash your application to a different location from where it was linked. Choose between:

- **Offset**
  - A numeric value for a relative offset. This offset will be added to the addresses in the application file.

- **Absolute address**
  - A numeric value for an absolute base address where the application will be flashed. The lowest address in the application will be placed on this address. Note that you can only use one flash loader for your application when you specify an absolute address.

You can use these numeric formats:

- 123456, decimal numbers
- 0x123456, hexadecimal numbers
- 0123456, octal numbers

The default base address used for writing the first byte—the lowest address—to flash is specified in the linker configuration file used for your application. However, it can sometimes be necessary to override the flash base address and start at a different location in the address space. This can, for example, be necessary for devices that remap the location of the flash memory.

**Flash loader path**

Use the text box to specify the path to the flash loader file (*.flash) to be used by your board configuration.
Extra parameters

Some flash loaders define their own set of specific options. Use this text box to specify options to control the flash loader. For information about available flash loader options, see the Parameter descriptions field.

Parameter descriptions

Displays a description of the extra parameters specified in the Extra parameters text box.
<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>absolute location, specifying for a breakpoint</td>
<td>-B (C-SPY command line option)</td>
</tr>
<tr>
<td>Access Type (Data breakpoints option)</td>
<td>--backend (C-SPY command line option)</td>
</tr>
<tr>
<td>Access type (Edit Memory Access option)</td>
<td>allowance information</td>
</tr>
<tr>
<td>Access type (Edit SFR option)</td>
<td>generated by compiler</td>
</tr>
<tr>
<td>Action (Code breakpoints option)</td>
<td>viewing in Call Stack window</td>
</tr>
<tr>
<td>Actual (SWO clock setting)</td>
<td>batch mode, using C-SPY in</td>
</tr>
<tr>
<td>Adaptive (JTAG/SWD speed setting)</td>
<td>Baud rate (Macraigor option)</td>
</tr>
<tr>
<td>Add to Watch Window (Symbolic Memory window context menu)</td>
<td>Baud rate (Serial port settings option)</td>
</tr>
<tr>
<td>Add (SFR Setup window context menu)</td>
<td>--BE32 (C-SPY command line option)</td>
</tr>
<tr>
<td>Address Bus Pattern (Address setting)</td>
<td>--BE8 (C-SPY command line option)</td>
</tr>
<tr>
<td>Address (Edit SFR option)</td>
<td>Big Endian (Memory window context menu)</td>
</tr>
<tr>
<td>Address (JTAG Watchpoints option)</td>
<td>blocks, in C-SPY macros</td>
</tr>
<tr>
<td>Allow hardware reset (RDI option)</td>
<td>bold style, in this guide</td>
</tr>
<tr>
<td>Ambiguous symbol (Resolve Symbol Ambiguity option)</td>
<td>Break At (breakpoints option)</td>
</tr>
<tr>
<td>Any Size (Data setting)</td>
<td>Break Condition (JTAG Watchpoints option)</td>
</tr>
<tr>
<td>Any (Access Type setting)</td>
<td>Break on Throw (Debug menu)</td>
</tr>
<tr>
<td>Any (Extern setting)</td>
<td>Break on Uncaught Exception (Debug menu)</td>
</tr>
<tr>
<td>Any (Mode setting)</td>
<td>Break (Debug menu)</td>
</tr>
<tr>
<td>Append to file (Trace Save option)</td>
<td>breakpoint condition, example</td>
</tr>
<tr>
<td>application, built outside the IDE</td>
<td>Breakpoint type (Code breakpoints option)</td>
</tr>
<tr>
<td>assembler labels, viewing</td>
<td>Breakpoint Usage window</td>
</tr>
<tr>
<td>assembler source code, fine-tuning</td>
<td>Breakpoint Usage (GDB Server menu)</td>
</tr>
<tr>
<td>assembler symbols, using in C-SPY expressions</td>
<td>Breakpoint Usage (I-jetJTAGjet menu)</td>
</tr>
<tr>
<td>assembler variables, viewing</td>
<td>Breakpoint Usage (J-Link menu)</td>
</tr>
<tr>
<td>assumptions, programming experience</td>
<td>Breakpoint Usage (Macraigor JTAG menu)</td>
</tr>
<tr>
<td>Attach to program (debugger option)</td>
<td>Breakpoint Usage (RDI menu)</td>
</tr>
<tr>
<td>Auto Scroll (Timeline window context menu)</td>
<td>Breakpoint Usage (Simulator menu)</td>
</tr>
<tr>
<td>Auto window</td>
<td>Breakpoint Usage (ST-LINK menu)</td>
</tr>
<tr>
<td>Auto (Clock setup setting)</td>
<td>Breakpoint Usage (TI Stellaris menu)</td>
</tr>
<tr>
<td>Auto (Default breakpoint type setting)</td>
<td>Breakpoint Usage (TI XDS100 menu)</td>
</tr>
<tr>
<td>Auto (JTAG/SWD speed setting)</td>
<td>breakpoints code, example</td>
</tr>
<tr>
<td>Auto (Protocol setting)</td>
<td>Autostep (Debug menu)</td>
</tr>
<tr>
<td>Auto (SWO setting)</td>
<td>Autodetect (SWO clock setting)</td>
</tr>
<tr>
<td>Autodetect (SWO prescaler setting)</td>
<td>204</td>
</tr>
<tr>
<td>Autodetect (SWO clock setting)</td>
<td>203</td>
</tr>
<tr>
<td>Autostep settings dialog box</td>
<td>90</td>
</tr>
</tbody>
</table>
Connecting a C-SPY macro ........................................ 300
Consumers of .................................................. 123
Data ............................................................. 140
data ............................................................ 140
data log ......................................................... 143
description of .................................................. 120
disabling used by Stack window ......................... 124
Icons for in the IDE ............................................ 122
in Memory window .......................................... 127
Listing all ....................................................... 133
profiling source .............................................. 240, 246
reasons for using ............................................ 119
Setting
    in Memory window ....................................... 127
    using system macros .................................... 128
    using the dialog box .................................... 126
Single-stepping if not available ....................... 53
Toggling ....................................................... 125
Types of ....................................................... 120
Useful tips ................................................... 129
Breakpoints dialog box
Code .......................................................... 134
Data .......................................................... 140
Data Log ..................................................... 143
Immediate .................................................... 147
Log .......................................................... 139
Trace Filter (J-Link) ....................................... 232
Trace Start .................................................. 220, 222, 224, 226–227
Trace Stop .................................................... 221
Breakpoints options (C-SPY options) .................. 145
Breakpoints window ......................................... 131
Broadcast all branches (ETM Trace Settings option) 198
Browse (Trace toolbar) ...................................... 206
Byte order, setting in Memory window ................ 161
Byte (Data setting) .......................................... 137
C
C function information, in C-SPY ....................... 78
C symbols, using in C-SPY expressions ............... 93
C variables, using in C-SPY expressions ............. 93
Cache type (Edit Memory Range option) ............. 183
call chain, displaying in C-SPY ......................... 78
Call stack information ..................................... 78
Call Stack window .......................................... 84
for backtrace information .................................. 79
Call Stack (Timeline window context menu) ......... 217
__cancelAllInterrupts (C-SPY system macro) ....... 310
__cancelInterrupt (C-SPY system macro) .......... 310
Catch exceptions (Breakpoints option) ............... 146
Catch exceptions (RDI option) ......................... 415
Clear All (Debug Log window context menu) ....... 88
Clear trace data (Trace toolbar) ....................... 206
Clear (Interrupt Log window context menu) ....... 290
Clear (Power Log window context menu) ........... 263
__clearBreak (C-SPY system macro) ................. 311
Clock setup (I-jet/JTAGjet option) .................. 405
Clock setup (J-Link/J-Trace option) ................. 410
Clock setup (ST-LINK option) ......................... 417
__closeFile (C-SPY system macro) .................. 311
code breakpoints
    overview ................................................ 120
toggling ................................................... 125
Code Coverage window ................................... 266
Code Coverage (Disassembly window context menu) 82
--code_coverage_file (C-SPY command line option) 356
code, covering execution of ............................ 266
command line options ..................................... 356
typographic convention .................................. 33
command prompt icon, in this guide ................. 33
communication problem, I-jet ......................... 404
communication problem, J-Link ....................... 369, 409
Communication (Angel option) ....................... 397
Communication (J-Link/J-Trace option) ............ 411
computer style, typographic convention ............. 32
conditional statements, in C-SPY macros ............ 303
Conditions (Code breakpoints option) .............. 135, 140–141
Configure (RDI menu) ..................................... 431

C-SPY® Debugging Guide
for ARM

444

UCSARM-4.3
Connect during reset (Reset setting) .......... 406, 416
context menu, in windows .................. 96
conventions, used in this guide ............ 32
Copy Window Contents (Disassembly
window context menu) ..................... 83
Copy (Debug Log window context menu) .... 88
copyright notice ................................ 2
Core and peripherals (Reset setting) ....... 406
Core (Reset setting) ......................... 400, 406
CPI (Generate setting) ....................... 200
--cpu (C-SPY command line option) ...... 351
CPU clock (Clock setup setting) ........... 405
CPU clock (SWO Configuration option) .... 203
CPU clock (SWO setting) ................. 410, 417
cspybat ....................................... 349
current position, in C-SPY Disassembly window ..... 81
cursor, in C-SPY Disassembly window ....... 81
Custom (Reset setting) ..................... 401
Cycle accurate tracing (ETM Trace Settings option) .. 198
--cycles (C-SPY command line option) .... 357
C-SPY
batch mode, using in ....................... 349
debugger systems, overview of ............ 39
differences between drivers ............... 42
environment overview ..................... 35
plugin modules, loading .................... 54
setting up .................................... 52–53
starting the debugger ....................... 54
C-SPY drivers
overview ...................................... 41
specifying .................................... 391
C-SPY expressions ......................... 92
evaluating, using Macro Quicklaunch window .. 106
evaluating, using Quick Watch window ...... 108
in C-SPY macros ............................. 303
Tooltip watch, using ....................... 91
Watch window, using ....................... 91
C-SPY hardware drivers, hardware installation .. 46
C-SPY macro "]_message"
style (Log breakpoints option) ............ 139
C-SPY macros
blocks ......................................... 304
conditional statements ..................... 303
C-SPY expressions .......................... 303
dialog box, using ........................... 297
examples ..................................... 295
checking status of register .................. 299
creating a log macro ....................... 300
execUserPreload, using ..................... 59
remapping memory before download ........ 59
executing ..................................... 296
connecting to a breakpoint .................. 300
using Quick Watch ......................... 299
using setup macro and setup file .......... 298
functions .................................... 94, 302
loop statements .............................. 304
macro statements ........................... 303
setup macro file ............................. 294
execting ..................................... 298
setup macro functions ...................... 294
summary ...................................... 306
system macros, summary of ................ 307
using .......................................... 293
variables ..................................... 94, 302
C-SPY options ............................... 389
Extra Options ................................ 396
Images ........................................ 394
Plugins ........................................ 395
Setup .......................................... 391
C-SPYLink ................................... 41
C++ exceptions
debugging ..................................... 66
single stepping ............................ 74
C++ terminology ............................. 32

D
data breakpoints, overview .................. 121
Data Bus Pattern (Data setting) ............ 138
Edit Expressions (Trace toolbar) ........................................... 206
Edit Interrupt dialog box .................................................. 282
Edit Memory Access dialog box ......................................... 186
Edit Memory Range dialog box .......................................... 177, 182
Edit Settings (Trace toolbar) ................................................. 206
Edit (Breakpoints window context menu) ............................... 132
Edit (SFR Setup window context menu) ................................. 176
edition, of this guide .......................................................... 2
Embedded C++ Technical Committee .................................. 32
EmbeddedICE macrocell .................................................... 121
EmuDiag (driver menu) ......................................................... 426
Emulator (TI XDS100 option) .................................................. 418
__emulatorSpeed (C-SPY system macro) ................................. 313
__emulatorStatusCheckOnRead (C-SPY system macro) ............. 314
Enable All (Breakpoints window context menu) ....................... 132
Enable interrupt simulation (Interrupt Setup option) ............... 280
Enable Log File (Log File option) .......................................... 89
Enable (Breakpoints window context menu) ............................. 132
Enable (Interrupt Log window context menu) ............................ 290
Enable (Power Log window context menu) .............................. 263
Enable (Timeline window context menu) ............................... 217
Enabled ports (ITM Stimulus Ports setting) ............................. 204
__enableInterrupts (C-SPY system macro) .............................. 314
Enable/Disable Breakpoint (Call Stack window context menu) .... 85
Enable/Disable Breakpoint (Disassembly window context menu) ... 83
Enable/Disable (Trace toolbar) ............................................. 206
End address (Edit Memory Access option) ............................. 182, 186
–endian (C-SPY command line option) ................................. 352
endianness. See byte order .................................................. 149
Enter Location dialog box ................................................... 149
ETM Trace Save (I-jet/JTAGjet menu) .................................... 425
ETM Trace Save (J-Link menu) ............................................. 428
ETM Trace Settings dialog box ............................................. 195, 197
ETM Trace Settings (I-jet/JTAGjet menu) ................................. 425
ETM Trace Settings (J-Link menu) ......................................... 428
ETM Trace (I-jet/JTAGjet menu) ............................................ 425
ETM Trace (J-Link menu) ..................................................... 428
ETM/ETB (J-Link/J-Trace option) .......................................... 410
__evaluate (C-SPY system macro) ......................................... 314
Evaluate Now (Macro Quicklauncher window context menu) ....... 107
Event Log Summary window ............................................... 117
Event Log Summary (I-jet/JTAGjet menu) ................................. 426
Event Log window ............................................................. 116
Event Log (I-jet/JTAGjet menu) ............................................ 425
Event Log (Timeline window context menu) ............................ 216
examples
C-SPY macros ................................................................. 295
interrupts
interrupt logging .............................................................. 279
timer .............................................................................. 276
macros
checking status of register ................................................. 299
creating a log macro .......................................................... 300
using Quick Watch ............................................................ 299
performing tasks and continue execution ............................... 130
tracing incorrect function arguments .................................. 129
EXC (Generate setting) .......................................................... 200
execUserExit (C-SPY setup macro) ......................................... 307
execUserFlashExit (C-SPY setup macro) ................................. 307
execUserFlashInit (C-SPY setup macro) ................................. 306
execUserFlashReset (C-SPY setup macro) ............................... 307
execUserPreload (C-SPY setup macro) .................................... 306
execUserPreReset (C-SPY setup macro) ................................. 307
execUserReset (C-SPY setup macro) ....................................... 307
execUserSetup (C-SPY setup macro) ...................................... 307
executed code, covering .................................................... 266
execution history, tracing ................................................... 193
expressions. See C-SPY expressions
Extend to cover requested range (Trigger range setting) .......... 144
Extend to cover requested range (Trigger range setting) . . . 142, 224, 226, 228, 231, 233
Extern (JTAG Watchpoints option) .......................... 138
Extra Options, for C-SPY ................................. 396

F

Factory ranges (Memory Configuration option) ........ 180
File format (Memory Save option) ......................... 163
file types
device description, specifying in IDE ................. 53
macro ................................................. 53, 391
File (Trace Save option) ........................... 238
filename extensions
ddf, selecting device description file ............. 53
mac, using macro file ................................ 53
Filename (Memory Restore option) .................. 164
Filename (Memory Save option) ....................... 163
Fill dialog box ........................................ 164
Find in Trace dialog box .............................. 236
Find in Trace window ................................. 237
Find (Memory window context menu) ............... 162
Find (Trace toolbar) ................................ 206
first activation time (interrupt property)
definition of ......................................... 273
First activation (Edit Interrupt option) ............ 283
Fixed (JTAG/SWD speed setting) ...................... 409
flash loader
parameters to control ................................. 442
specifying the path to ................................ 441
using .................................................... 437
Flash Loader Overview dialog box ................. 439
flash memory, load library module to ............ 326
--flash_loader (C-SPY command line option) ...... 367
FOLD (Generate setting) ................................. 200
for (macro statement) ................................ 304
Force (SWO Trace Window Settings option) ..... 199
Forced Interrupt window ............................. 284
Forced Interrupts (Simulator menu) ............... 422
--fpu (C-SPY command line option) ................. 352
Function Profiler window .............................. 245
Function Profiler (I-jet/JTAGjet menu) .......... 426
Function Profiler (J-Link menu) ................... 429
Function Profiler (Simulator menu) ............... 423
Function Profiler (ST-LINK menu) .................. 433
Function Trace window ............................... 209
Function Trace (I-jet/JTAGjet menu) ............ 425
Function Trace (J-Link menu) ...................... 428
Function Trace (Simulator menu) ................... 422
functions
call stack information for ............................ 78
C-SPY running to when starting ................ 52, 391
most time spent in, locating ......................... 239

G

GDB Server (C-SPY driver), menu .................... 423
__gdbserver_exec_command (C-SPY system macro) . 315
--gdbserver_exec_command (C-SPY command line option) 367
Generate (SWO Trace Window Settings option) .... 200
Go to Source (Breakpoints window context menu) 132
Go to Source (Call Stack window context menu) . 85
Go To Source (Timeline window context menu) ... 218
Go (Debug menu) .................................... 65, 77
Graphical bar (Memory Configuration dialog box) 181

H

Halfword (Data setting) ............................... 137
Halt after bootloader (Reset setting) ............ 407
Halt before bootloader (Reset setting) ........... 407
Hardware reset (Macraigor option) ............... 413
hardware setup, power consumption because of . 256
Hardware (Default breakpoint type setting) .... 146
Hardware (Reset setting) ........................... 400
Hardware, Atmel AT91SAM7 (Reset setting) .... 408
Hardware, halt after delay (ms) (Reset setting) . 407
Hardware, halt at 0 (Reset setting) ............... 408
--jet_standard_reset (C-SPY command line option) . . . 371
--jet_swo_on_d0 (C-SPY command line option) . . . . . 372
--jet_swo_prescaler (C-SPY command line option) . . . . 373
--jet_swo_protocol (C-SPY command line option) . . . 373
--jet_tap_position (C-SPY command line option) . . . 373
__jlinkExecCommand (C-SPY system macro) . . . . . . . . 319
--jlink_device_select (C-SPY command line option) . 374
--jlink_exec_command (C-SPY command line option) . 374
--jlink_initial_speed (C-SPY command line option) . . 375
--jlink_interface (C-SPY command line option) . . . . . 375
--jlink_ir_length (C-SPY command line option) . . . . . . 375
--jlink_reset_strategy (C-SPY command line option) . 376
--jlink_script_file (C-SPY command line option) . . . . 376
--jlink_speed (C-SPY command line option) . . . . . . . . 377
--jlink_trace_source (C-SPY command line option) . . . 377
JTAG interfaces
  I-jet ........................................... 400
  J-Link ....................................... 406, 410
I-jet scan chain with multiple targets (Macraigor option) 414
I-jet scan chain with multiple targets (I-jet option) ....... 403
I-jet scan chain with multiple targets (J-Link/J-Trace option) . 411
JTAG scan chain (I-jet/JTAGjet option) ................... 403
JTAG scan chain (J-Link/J-Trace option) .................. 411
JTAG speed (Macraigor option) ......................... 413
JTAG Watchpoints dialog box .................................. 136
JTAG watchpoints, overview .................................. 121
JTAG (Interface setting) ................................200
  __jtagCommand (C-SPY system macro) .................. 320
  __jtagCP15IsPresent (C-SPY system macro) .......... 320
  __jtagCP15ReadReg (C-SPY system macro) ............ 321
  __jtagCP15WriteReg (C-SPY system macro) .......... 321
  __jtagData (C-SPY system macro) .................... 321
  __jtagRawRead (C-SPY system macro) ................. 322
  __jtagRawSync (C-SPY system macro) ................. 323
  __jtagRawWrite (C-SPY system macro) ............... 324
  __jtagResetTRST (C-SPY system macro) ............ 325
JTAG/SWD speed (I-jet/JTAGjet option) ................... 404
JTAG/SWD speed (J-Link/J-Trace option) ................. 409

JTAG/SWD speed (TI Stellaris option) .................... 418
J-Link communication problem ............................ 369, 409
J-Link JTAG interface .................................... 406, 410
J-Link (C-SPY driver), menu ................................ 427

L
labels (assembler), viewing .................................. 96
Leave on after debugging (Target power supply setting) . 402
Length (Fill option) .................................. 165
lightbulb icon, in this guide ................................ 33
Link condition (Trace Start option) ....................... 229
Link condition (Trace Stop option) ...................... 232, 234
linker options
  typographic convention .................................. 32
Little Endian (Memory window context menu) ............ 161
Live Watch window ..................................... 103
--lmiftdi_speed (C-SPY command line option) .......... 378
__loadImage (C-SPY system macro) ...................... 325
loading multiple debug files, list currently loaded .... 69
loading multiple images ................................ 56
Locals window ......................................... 100
log breakpoints, overview ................................ 120
Log communication (Angel option) ....................... 398
Log communication (debugger option) .................... 420
Log communication (IAR ROM-monitor option) .......... 399
Log communication (I-jet/JTAGjet option) ............. 402
Log communication (J-Link/J-Trace option) ............ 412
Log communication (Macraigor option) ................. 414
Log communication (TI Stellaris option) ............. 418
Log File dialog box .................................... 89
Log RDI communication (RDI option) ................... 415
Logging>Set Log file (Debug menu) ..................... 67
Logging>Set Terminal I/O Log file (Debug menu) .... 67
loop statements, in C-SPY macros ....................... 304
low-power mode, power consumption during ............. 254
LSU (Generate setting) .................................. 200

C-SPY® Debugging Guide
for ARM

450

UCSARM-4:3
mac (filename extension), using a macro file .......................... 53
Macraigor (C-SPY driver), menu ................................ 430
--macro (C-SPY command line option) .................. 378
Macro Configuration dialog box .......................... 297
macro files, specifying ............................................ 53, 391
Macro Quicklaunch window .................................. 106
macro statements .................................................... 303
macros
  executing ......................................................... 296
  using .......................................................... 293
Macros (Debug menu) .............................................. 67
--mac_handler_address (C-SPY command line option) .... 378
--mac_jtag_device (C-SPY command line option) ....... 379
--mac_multiple_targets (C-SPY command line option) .. 380
--mac_reset_pulls_reset (C-SPY command line option) .... 380
--mac_set_temp_reg_buffer (C-SPY command line option) .... 381
--mac_speed (C-SPY command line option) ............. 381
main function, C-SPY running to when starting ....... 52, 391
Manchester (Protocol setting) .......................... 405
Manufacturer RDI driver (RDI option) .................. 415
--mapu (C-SPY command line option) .................. 382
Mask (Address setting) ........................................... 137
Mask (Data setting) .............................................. 138
Mask (Match data setting) .................................... 142, 223, 225, 229, 232, 234
Match data (Data breakpoints option) ................ 142
Match data (Trace Start option) ......................... 223, 225, 229
Match data (Trace Stop option) ......................... 231, 234
memory access checking ......................................... 156
Memory access checking (Memory Access Setup option) 185
Memory Access Setup dialog box ......................... 184
Memory Access Setup (Simulator menu) .................. 422
memory accesses, illegal ....................................... 156
Memory Configuration dialog box ................. 179
Memory Configuration (I-jet/JTAGjet menu) .......... 424
Memory Fill (Memory window context menu) .......... 162
memory map ....................................................... 184
Memory range (Edit Memory Access option) .......... 182, 186
Memory Restore dialog box .................................. 164
Memory Restore (Memory window context menu) ...... 162
Memory Save dialog box ...................................... 163
Memory Save (Memory window context menu) ...... 162
Memory window ................................................ 159
memory zones .................................................. 155
__memoryRestore (C-SPY system macro) .............. 326
__memorySave (C-SPY system macro) .................. 327
Memory>Restore (Debug menu) ............................ 67
Memory>Save (Debug menu) .................................. 66
menu bar, C-SPY-specific ...................................... 64
Message (Log breakpoints option) ....................... 139
migration, from earlier IAR compilers ................. 31
MISRA C, documentation ....................................... 31
Mixed Mode (Disassembly window context menu) ..... 84
Mode (JTAG Watchpoints option) ......................... 138
Motorola, C-SPY output format .......................... 40
Move to PC (Disassembly window context menu) .... 82
Multi-ICE interface. See RealView Multi-ICE interface

n MHz (JTAG/SWD speed setting) ......................... 404
Name (Edit SFR option) ...................................... 178
naming conventions .............................................. 33
Navigate (Timeline window context menu) ............ 216
New Breakpoint (Breakpoints window context menu) .. 133
Next Statement (Debug menu) ........................... 66
Next Symbol (Symbolic Memory window context menu) 167
Non User (Mode setting) .................................... 138
Normal (Break Condition setting) ....................... 138
Normal (Reset setting) ....................................... 406, 416
Normal, disable watchdog (Reset setting) ............. 407
OCD interface device (Macraigor option) .......................... 412
OP Fetch (Access Type setting) ................................. 137
__openFile (C-SPY system macro) ................................ 327
Operation (Fill option) ............................................ 165
operators, sizeof in C-SPY ........................................ 94
optimizations, effects on variables .............................. 95
options
  in the IDE ......................................................... 389
  on the command line ......................................... 356, 396
Options (Stack window context menu) .......................... 172
__orderInterrupt (C-SPY system macro) ....................... 329
Originator (debugger option) ................................. 395
Override default board file (debugger option) ............... 393
Override project default (SWO Configuration option) ...... 203
Override project setting (SWO Configuration option) ....... 203
overriding the default stack setup ............................ 168

P

-p (C-SPY command line option) ................................. 382
parameters
  list of passed to the flash loader .......................... 439
  tracing incorrect values of ................................. 78
typographic convention ........................................ 32
part number, of this guide .................................... 2
PC only (Data Log Events setting) .............................. 202
PC samples (Force setting) ...................................... 200
PC Sampling (SWO Configuration option) ..................... 202
PC + data value + base addr (Data Log Events setting) ... 202
peripheral units
  detecting unattended ......................................... 254
device-specific .................................................. 57
peripheral units, in Register window .......................... 154
Please select one symbol
(Resolve Symbol Ambiguity option) ......................... 111

--plugin (C-SPY command line option) ....................... 383

Plugins (C-SPY options) ........................................ 395
__popSimulatorInterruptExecutingStack (C-SPY system macro) ........................................ 330

pop-up menu. See context menu
Port (Macraigor option) ........................................ 413
Port (Serial port settings option) ............................. 397, 399
power consumption, measuring ................................ 240, 251
Power Log Setup (I-jet/JTAGjet menu) ....................... 426
Power Log Setup (J-Link menu) ............................... 429
Power Log window ................................................ 262
Power Log (I-jet/JTAGjet menu) ............................... 426
Power Log (J-Link menu) ........................................ 429
Power Log (Timeline window context menu) ................. 217
definition of ..................................................... 273
Probabilty % (Edit Interrupt option) ......................... 283
 Probe configuration file (I-jet/JTAGjet option) ............ 404
--proc_stack_xxx (C-SPY command line option) ............ 383
Profile Selection (Timeline window context menu) ........ 218

profiling
  on function level ............................................. 242
  on instruction level ......................................... 242

profiling information, on functions and instructions ....... 240
profiling sources
  breakpoints .................................................... 240, 246
 sampling ......................................................... 240, 247
trace (calls) ..................................................... 240, 246
trace (flat) ....................................................... 240, 247

program execution, in C-SPY .................................. 240, 247
program, see also application ................................. 29

C-SPY® Debugging Guide
for ARM

452
projects, for debugging externally built applications .... 54
Protocol (I-jet/JTAGjet option) ......................... 405
publication date, of this guide ......................... 2

Q
Quick Watch window .................................. 108
executing C-SPY macros .............................. 299

R
RAM (Edit Memory Access option) ................. 183
Range for (Viewing Range option) ................. 219
Range (Break Condition setting) ...................... 138
Rate (PC Sampling setting) ........................... 202
RDI (C-SPY driver), menu ............................ 431
--rdi_allow_hardware_reset (C-SPY command line option) .... 384
--rdi_driver_dll (C-SPY command line option) .... 384
--rdi_heartbeat (C-SPY command line option) ...... 353
--rdi_step_max_one (C-SPY command line option) .... 384
Read (Access Type setting) ......................... 137
__readFile (C-SPY system macro) .................. 330
__readFileByte (C-SPY system macro) .............. 331
__readMemoryByte (C-SPY system macro) .......... 331
__readMemory8 (C-SPY system macro) .............. 331
__readMemory16 (C-SPY system macro) ............ 331
__readMemory32 (C-SPY system macro) ............ 332
RealView Multi-ICE interface ....................... 414
reference information, typographic convention ...... 33
Refresh (Debug menu) ............................... 67
register groups ....................................... 154
    predefined, enabling ................................ 173
Register window ..................................... 172
registered trademarks ................................ 2
__registerMacroFile (C-SPY system macro) ........ 332
registers, displayed in Register window .......... 173
Remove All (Macro Quicklauncher window context menu) ........ 108
Remove (Macro Quicklauncher window context menu) .... 107
Remove (Watch window context menu) ............... 102, 109
Repeat interval (Edit Interrupt option) ............ 283
repeat interval (interrupt property), definition of . 273
Replace (Memory window context menu) ............. 162
Reset and halt after bootloader (Reset setting) .... 401
Reset by watchdog or reset register (Reset setting) .... 401
Reset Pin (Reset setting) ............................ 406, 416
Reset (Debug menu) .................................. 65
Reset (I-jet/JTAGjet option) ......................... 400
Reset (J-Link/J-Trace option) ....................... 406, 416
__resetFile (C-SPY system macro) .................. 333
Resolve Source Ambiguity dialog box ............... 150
Restore software breakpoints at (Breakpoints option) .... 146
Restore (Memory Restore option) .................... 164
__restoreSoftwareBreakpoints (C-SPY system macro) .... 333
return (macro statement) ............................ 304
ROM-monitor, definition of .......................... 40
ROM/Flash (Edit Memory Access option) .......... 183
RTOS awareness debugging ........................... 37
RTOS awareness (C-SPY plugin module) ............. 38
Run to Cursor (Call Stack window context menu) .... 85
Run to Cursor (Debug menu) ........................ 66
Run to Cursor (Disassembly window context menu) .... 82
Run to Cursor, command for executing .......... 78
Run to (C-SPY option) ............................... 52
Run to (debugger option) ............................ 391
R/W (Access Type setting) ......................... 137

S
sampling, profiling source .......................... 240, 247
Save Custom SFRs (SFR Setup window context menu) .... 177
Save to log file (Interrupt Log window context menu) .... 290
Save to log file (Power Log window context menu) .... 264
Save (Memory Save option) .......................... 163
Save (Trace toolbar) ................................. 206
Scale (Viewing Range option) ....................... 219
Scan chain contains non-ARM devices
(JTAG scan chain setting) .......................... 403, 412
Select All (Debug Log window context menu) .......... 88
Select Graphs (Timeline window context menu) ...... 218
Select plugins to load (debugger option) ............... 395
--semihosting (C-SPY command line option) .......... 385
Send heartbeat (Angel option) ........................ 397
Serial port settings (Angel option) ..................... 397
Serial port settings (IAR ROM-monitor option) ....... 399
Session Overview (driver menu) ......................... 426
Set Data Breakpoint (Memory window context menu) . 162
Set Next Statement (Disassembly menu) ................. 66
_SetCodeBreak (C-SPY system macro) .................. 334
_SetDataBreak (C-SPY system macro) .................. 335
_SetDataLogBreak (C-SPY system macro) ............... 337
_SetLogBreak (C-SPY system macro) ................. 338
_SetSimBreak (C-SPY system macro) ..................... 339
_SetTraceStartBreak (C-SPY system macro) ............ 340
_SetTraceStopBreak (C-SPY system macro) ............. 341
setup macro functions ................................ 294
reserved names ........................................ 306
Setup macros (debugger option) ......................... 391
Setup (C-SPY options) ................................ 391
SFR in Register window .................................. 173
using as assembler symbols ............................. 93
SFR Setup window ........................................ 174
SFR/Uncached (Edit Memory Access option) ............ 183
shortcut menu, See context menu
Show all images (Images window context menu) ....... 70
Show All (SFR Setup window context menu) ............ 176
Show Arguments (Call Stack window context menu) ... 85
Show As (Watch window context menu) ................. 103
Show Custom SFRs only (SFR Setup window context menu) ........ 176
Show Cycles (Interrupt Log window context menu) .... 290
Show Cycles (Power Log window context menu) ....... 264
Show Factory SFRs only (SFR Setup window context menu) .......... 176
Show Factory SFRs only (SFR Setup window context menu) .......... 176
Show Numbers (Timeline window context menu) ....... 217
Show Numbers (Timeline window context menu) ....... 217
Show offsets (Stack window context menu) ............. 171
Show only (Image window context menu) ............... 170
Show Time (Interrupt Log window context menu) ....... 290
Show Time (Power Log window context menu) .......... 264
Show timestamp (ETM Trace Settings option) .......... 198
Show variables (Stack window context menu) ........... 171
--silent (C-SPY command line option) .................. 385
simulating interrupts, enabling/disabling ............... 280
Simulator menu ......................................... 412
Size (breakpoints option) .............................. 135, 141, 144
Size (Edit SFR option) .................................. 178
Size (Timeline window context menu) .................. 217
Size (Trace Start option) .............................. 223, 225, 228, 230, 233
Sizeof .................................................... 94
SLEEP (Generate setting) .................................. 200
software delay, power consumption during ............... 253
Software (Default breakpoint type setting) ............. 146
Software (Reset setting) ................................ 400, 408
Software, Analog devices (Reset setting) ............... 408
Solid Graph (Timeline window context menu) ........... 217
_sourcePosition (C-SPY system macro) .................. 342
special function registers (SFR) ........................
in Register window ...................................... 173
using as assembler symbols ............................. 93
stack usage, computing .................................. 156
Stack window ............................................. 168
stack.mac ................................................. 294
Stall processor on FIFO full
(ETM Trace Settings option) ............................ 198
standard C, sizeof operator in C-SPY ..................... 94
Start address (Edit Memory Access option) ............ 182, 186
Start address (Fill option) .............................. 165
Start address (Memory Save option) ..................... 163
Statics window .......................................... 104
Step Into (Debug menu) .................................. 65
Step Into, description .................................... 75
Step Out (Debug menu) ................................... 66

C-SPY® Debugging Guide
for ARM

454

UCSARM-4:3
Step Out, description. ........................................ 76
Step Over (Debug menu) ....................................... 65
Step Over, description ....................................... 75
step points, definition of .................................. 74
--stlink_interface (C-SPY command line option) .... 386
--stlink_reset_strategy (C-SPY command line option) .... 386
Stop address (Memory Save option) ....................... 163
Stop Debugging (Debug menu) ............................. 65
__strFind (C-SPY system macro) ........................ 343
ST-LINK (C-SPY driver), menu ............................... 432
__subString (C-SPY system macro) ....................... 343
Suppress download (debugger option) ................. 393, 419
SWD interface, information in Trace window ........ 190
SWD (Interface setting) ..................................... 403, 411, 413, 416, 418
Switch off after debugging (Target power supply setting) .... 402
SWO clock (SWO Configuration option) .................. 203
SWO communication channel enabling .................. 368, 403, 411, 413, 418
for timestamps in trace .................................. 199
SWO Configuration dialog box ............................. 201
SWO Configuration (I-jet/JTAGjet menu) ................ 425
SWO Configuration (J-Link menu) ......................... 428
SWO Configuration (ST-LINK menu) ...................... 432
SWO on the TraceDO pin (I-jet/JTAGjet option) ...... 405
SWO prescaler (Clock setup setting) .................... 405
SWO prescaler (SWO Configuration option (I-jet)) .... 204
SWO Trace Save (J-Link menu) ............................ 428
SWO Trace Save (ST-LINK menu) ......................... 432
SWO Trace Settings dialog box ......................... 199
SWO Trace Settings dialog box (I-jet) .................. 199
SWO Trace Window Settings (I-jet/JTAGjet menu) .... 425
SWO Trace Window Settings (J-Link menu) ............. 428
SWO Trace Window Settings (ST-LINK menu) .......... 432
SWO Trace (I-jet/JTAGjet menu) ......................... 425
SWO Trace (J-Link menu) .................................. 428
SWO Trace (ST-LINK menu) ................................. 433
Symbolic Memory window .................................. 166
Symbols window ............................................. 109
symbols, using in C-SPY expressions .................. 92
System (Reset setting) ...................................... 400

T

TAP number (JTAG scan chain setting) .................. 403, 411
Target power supply (I-jet/JTAGjet option) .......... 402
target system, definition of ............................. 39
__targetDebuggerVersion (C-SPY system macro) .... 344
TCP/IP address or hostname (GDB Server option) ... 398
TCP/IP Macraigor option .................................. 413
TCP/IP (Angel option) ...................................... 397
TCP/IP (Communication setting) ......................... 411
Terminal IO Log Files (Terminal IO Log Files option) .... 87
Terminal I/O Log Files dialog box ....................... 87
Terminal I/O window ........................................ 79, 86
terminology .................................................. 32
Text search (Find in Trace option) ....................... 236
Third-Party Driver (debugger options) .................. 419
Ti emulation package installation path
(TI XDS100 option) .......................................... 418
Ti Stellaris (C-SPY driver), menu ......................... 433
Ti XDS100 (C-SPY driver), menu ......................... 434
Time Axis Unit (Timeline window context menu) ...... 218
time interval, in Timeline window ....................... 243
Time Stamps (Force setting) .............................. 199
timeline window .............................................. 209
Timeline window ............................................. 420
Timeline (I-jet/JTAGjet menu) ............................ 426
Timeline (J-Link menu) ..................................... 429
Timeline (Simulator menu) ................................. 423
Timeline (ST-LINK menu) .................................. 433
--timeout (C-SPY command line option) ............... 387
timer interrupt, example ................................. 276
timestamps in SWO trace ................................ 199
Timesteps (SWO Configuration option) .................. 204
To Log File (ITM Stimulus Ports setting) ................ 204
To Terminal I/O window (ITM Stimulus Ports setting) ... 204
Toggle Breakpoint (Code) (Call) ......................... 85
Stack window context menu ............................... 85
Toggle Breakpoint (Code) (Disassembly window context menu) ........................................ 83
Toggle Breakpoint (Log) (Call Stack window context menu) ........................................ 85
Toggle Breakpoint (Log) (Disassembly window context menu) ........................................ 83
Toggle Breakpoint (Trace Start) (Disassembly window context menu) ........................................ 83
Toggle Breakpoint (Trace Stop) (Disassembly window context menu) ........................................ 83
Toggle source (Trace toolbar) .................................................. 206
_toLower (C-SPY system macro) ........................................ 344
tools icon, in this guide .................................................. 33
_toString (C-SPY system macro) ........................................ 345
__toUpper (C-SPY system macro) ........................................ 345
Trace buffer size (Trace Settings option) ........................................ 196
Trace Expressions window .................................................. 235
Trace Filter breakpoints dialog box (J-Link) .................................................. 232
Trace port mode (Trace Settings option) ........................................ 195, 197
Trace port width (Trace Settings option) ........................................ 195, 197
Trace Save dialog box .................................................. 238
Trace Save (RDI menu) .................................................. 431
Trace Settings (RDI menu) .................................................. 431
trace start and stop breakpoints, overview ........................................ 120
Trace Start breakpoints dialog box ........................................ 220, 222, 224, 226–227
Trace Stop breakpoints dialog box .................................................. 221
Trace window .................................................. 205
trace (calls), profiling source ........................................ 240, 246
trace (flat), profiling source ........................................ 240, 247
Trace (Simulator menu) .................................................. 422
trace, in Timeline window .................................................. 209
trademarks .................................................. 2
Trigger at (breakpoints option) ........................................ 139, 143, 147, 221–222, 233
Trigger at (Trace Start option) .................................................. 222, 224, 227–228
Trigger at (Trace Stop option) .................................................. 230
Trigger range (Data breakpoints option) ........................................ 142
Trigger range (Data Log breakpoints option) .................................................. 144
Trigger range (Trace Start option) ........................................ 223, 226, 228, 231, 233
Trigger (Forced Interrupt window context menu) ........................................ 284
typographic conventions .................................................. 32

U
UART (Protocol setting) ........................................ 405
Unavailable, C-SPY message ........................................ 95
__unloadImage (C-SPY system macro) ........................................ 346
USB (Communication setting) ........................................ 411
Use command line options (debugger option) ........................................ 396
Use Extra Images (debugger option) ........................................ 394
Use flash loader (debugger option) ........................................ 393
Use manual ranges (Memory Access Setup option) ........................................ 185
Use ranges based on (Memory Access Setup option) ........................................ 184
Use tab-separated format (Trace Save option) ........................................ 238
Used ranges (Memory Configuration option) ........................................ 180
user application, definition of ........................................ 39
User (Mode setting) ........................................ 138

V
Value (Address setting) ........................................ 137
Value (Data setting) ........................................ 138
Value (Fill option) ........................................ 165
Value (Match data setting) ........................................ 142, 223, 225, 229, 231, 234
variables
effects of optimizations ........................................ 95
information, limitation on ........................................ 94
using in C-SPY expressions ........................................ 93
variance (interrupt property), definition of ........................................ 273
Variance % (Edit Interrupt option) ........................................ 283
Vector Catch dialog box ........................................ 148
Vector Catch (I-jet/JTAGjet menu) ........................................ 426
Vector Catch (J-Link menu) ........................................ 428
Vector Catch (Macraigor JTAG menu) ........................................ 431
Verify all (debugger option) ........................................ 420
Verify download (debugger option) ........................................ 392
tversion number of this guide ........................................ 2
Viewing Range dialog box ........................................ 218
Viewing Range (Timeline window context menu) ........................................ 217
visualSTATE, C-SPY plugin module for ........................................ 41
### Symbols

- `_cancelAllInterrupts` (C-SPY system macro) .............. 310
- `_cancelInterrupt` (C-SPY system macro) .............. 310
- `_clearBreak` (C-SPY system macro) .................. 311
- `_closeFile` (C-SPY system macro) .................. 311
- `_delay` (C-SPY system macro) ..................... 311
- `_disableInterrupts` (C-SPY system macro) ............ 312
- `_driverType` (C-SPY system macro) .................. 312
- `_emulatorSpeed` (C-SPY system macro) ............. 313
- `_emulatorStatusCheckOnRead` (C-SPY system macro) . 314
- `_enableInterrupts` (C-SPY system macro) ............ 314
- `_evaluate` (C-SPY system macro) .................. 314
- `_fmessage` (C-SPY macro statement) .................. 304
- `_gdbserver_exec_command` (C-SPY system macro) ..... 315
- `_hwJetResetWithStrategy` (C-SPY system macro) .... 316
- `_hwReset` (C-SPY system macro) .................... 316
- `_hwResetRunToBp` (C-SPY system macro) ............ 317
- `_hwResetWithStrategy` (C-SPY system macro) ....... 318
- `_isBatchMode` (C-SPY system macro) ............... 319
- `_jlinkExecCommand` (C-SPY system macro) ........... 319
- `_jtagCommand` (C-SPY system macro) ............... 320
- `_jtagCP15IsPresent` (C-SPY system macro) ........ 320
- `_jtagCP15ReadReg` (C-SPY system macro) ........... 321
- `_jtagCP15WriteReg` (C-SPY system macro) ........... 321
- `_jtagData` (C-SPY system macro) ................... 321
- `_jtagRawRead` (C-SPY system macro) ................. 322
- `_jtagRawSync` (C-SPY system macro) ................. 323
- `_jtagRawWrite` (C-SPY system macro) ............... 324
- `_jtagResetTRST` (C-SPY system macro) .............. 325
- `_loadImage` (C-SPY system macro) .................. 325
- `_memoryRestore` (C-SPY system macro) .............. 326
- `_memorySave` (C-SPY system macro) ................. 327
- `_message` (C-SPY macro statement) ................. 304
- `_openFile` (C-SPY system macro) .................. 327
- `_orderInterrupt` (C-SPY system macro) ............. 329
- `_popSimulatorInterruptExecutingStack` (C-SPY system macro) ............... 329
- `_readFile` (C-SPY system macro) .................. 330
- `_readFileByte` (C-SPY system macro) ............... 331
- `_readMemoryByte` (C-SPY system macro) ............ 331
- `_readMemory8` (C-SPY system macro) ............... 331
- `_readMemory16` (C-SPY system macro) .............. 331
- `_readMemory32` (C-SPY system macro) .............. 331
- `_registerMacroFile` (C-SPY system macro) ............ 332
- `_resetFile` (C-SPY system macro) .................. 333
- `_writeFile` (C-SPY system macro) .................. 346
- `_writeFileByte` (C-SPY system macro) ............... 347
- `_writeMemoryByte` (C-SPY system macro) ............ 347
- `_writeMemory8` (C-SPY system macro) ............... 347
- `_writeMemory16` (C-SPY system macro) ............. 347
- `_writeMemory32` (C-SPY system macro) ............. 348
- `_writeMemory64` (C-SPY system macro) ............. 348

### W

- waiting for device, power consumption during .......... 253
- Wanted (SWO clock setting) ......................... 203
- warnings icon, in this guide ......................... 33
- Watch window ........................................ 101
- using .............................................. 91
- Watchpoints (J-Link menu) .......................... 427
- Watchpoints (Macraigor JTAG menu) .................. 431
- web sites, recommended ................................ 32
- while (macro statement) ................................ 304
- windows, specific to C-SPY ......................... 68
- Word (Data setting) .................................. 137
- Write (Access Type setting) .......................... 137
- `__writeFile` (C-SPY system macro) .................. 346
- `__writeFileByte` (C-SPY system macro) ............ 347
- `__writeMemoryByte` (C-SPY system macro) ......... 347
- `__writeMemory8` (C-SPY system macro) ............. 347
- `__writeMemory16` (C-SPY system macro) ........... 347
- `__writeMemory32` (C-SPY system macro) ........... 348

### X

- `--xds_rootdir` (C-SPY command line option) ........ 387

### Z

- `zone` in C-SPY .................................. 155
- Zone (Edit SFR option) ................................ 178
- Zone (Memory Save option) ............................ 163–165
- Zoom (Timeline window context menu) ............... 216
__restoreSoftwareBreakpoints (C-SPY system macro) ............... 333
__setCodeBreak (C-SPY system macro) .......................... 334
__setDataBreak (C-SPY system macro) ......................... 335
__setDataLogBreak (C-SPY system macro) ..................... 337
__setLogBreak (C-SPY system macro) .......................... 338
__setSimBreak (C-SPY system macro) .......................... 339
__setTraceStartBreak (C-SPY system macro) ................... 340
__setTraceStopBreak (C-SPY system macro) ..................... 341
__smessage (C-SPY macro statement) .......................... 304
__sourcePosition (C-SPY system macro) ....................... 342
__strFind (C-SPY system macro) .............................. 343
__subString (C-SPY system macro) .............................. 343
__targetDebuggerVersion (C-SPY system macro) ............... 344
__toLower (C-SPY system macro) ............................... 344
__toString (C-SPY system macro) .............................. 345
__toUpper (C-SPY system macro) ............................... 345
__unloadImage (C-SPY system macro) .......................... 346
__writeFile (C-SPY system macro) ............................. 346
__writeFileByte (C-SPY system macro) ......................... 347
__writeMemoryByte (C-SPY system macro) ...................... 347
__writeMemory8 (C-SPY system macro) ........................ 348
__writeMemory16 (C-SPY system macro) ....................... 349
__writeMemory32 (C-SPY system macro) ....................... 349
-B (C-SPY command line option) .............................. 356
-p (C-SPY command line option) ............................... 382
--backend (C-SPY command line option) ....................... 356
--BE32 (C-SPY command line option) ......................... 351
--BE8 (C-SPY command line option) .......................... 351
-code_coverage_file (C-SPY command line option) ........... 356
--cpu (C-SPY command line option) ........................... 351
--cycles (C-SPY command line option) ....................... 357
--device (C-SPY command line option) ......................... 357
--disable_interrupts (C-SPY command line option) .......... 358
--download_only (C-SPY command line option) ............... 358
--drv_attach_to_program (C-SPY command line option) ....... 352
--drv_catch_exceptions (C-SPY command line option) ........ 358
--drv_communication (C-SPY command line option) .......... 360
--drv_communication_log (C-SPY command line option) ....... 363
--drv_default_breakpoint (C-SPY command line option) ...... 363
--drv_reset_to_cpu_start (C-SPY command line option) ..... 364
--drv_restore_breakpoints (C-SPY command line option) ..... 365
--drv_suppress_download (C-SPY command line option) ..... 352
--drv_swo_clock_setup (C-SPY command line option) ....... 365
--drv_vector_table_base (C-SPY command line option) ..... 366
--drv_verify_download (C-SPY command line option) ......... 352
--endian (C-SPY command line option) ....................... 352
--flash_loader (C-SPY command line option) ................ 367
--fp (C-SPY command line option) ............................ 352
--gdbserv_exec_command (C-SPY command line option) ....... 367
--jet_cpu_clock (C-SPY command line option) ............... 367
--jet_interface (C-SPY command line option) ................ 368
--jet_ir_length (C-SPY command line option) ................ 368
--jet_jtag_speed (C-SPY command line option) ............... 369
--jet_power_from_probe (C-SPY command line option) ........ 369
--jet_script_file (C-SPY command line option) ............... 370
--jet_script_reset (C-SPY command line option) .............. 370
--jet_standard_reset (C-SPY command line option) .......... 371
--jet_swo_on_d0 (C-SPY command line option) ................. 372
--jet_swo_prescaler (C-SPY command line option) .......... 372
--jet_swo_protocol (C-SPY command line option) ............. 373
--jet_tap_position (C-SPY command line option) ............. 373
--jlink_device_select (C-SPY command line option) ........ 374
--jlink_exec_command (C-SPY command line option) .......... 374
--jlink_initial_speed (C-SPY command line option) .......... 375
--jlink_interface (C-SPY command line option) ............... 375
--jlink_ir_length (C-SPY command line option) ............... 375
--jlink_reset_strategy (C-SPY command line option) ......... 376
--jlink_script_file (C-SPY command line option) ............ 376
--jlink_speed (C-SPY command line option) ................... 377
--jlink_trace_source (C-SPY command line option) .......... 377
--lmiftdi_speed (C-SPY command line option) ................. 378
--macro (C-SPY command line option) ........................ 378
--mac_handler_address (C-SPY command line option) ......... 378
--mac_interface (C-SPY command line option) ................ 379
Index

--mac_jtag_device (C-SPY command line option) . . . . . . 379
--mac_multiple_targets (C-SPY command line option) . . 380
--mac_reset_pulls_reset (C-SPY command line option) . . 380
--mac_set_temp_reg_buffer (C-SPY command line option) . 381
--mac_speed (C-SPY command line option). . . . . . . . . 381
--mac_xscale_ir7 (C-SPY command line option) . . . . . . . 381
--mapu (C-SPY command line option) . . . . . . . . . . . . . . 382
--plugin (C-SPY command line option) . . . . . . . . . . . . 383
--proc_stack_xxx (C-SPY command line option) . . . . . . 383
--rdi_allow_hardware_reset (C-SPY command line option) . 384
--rdi_driver_dll (C-SPY command line option). . . . . . . 384
--rdi_heartbeat (C-SPY command line option) . . . . . . . 353
--rdi_step_max_one (C-SPY command line option) . . . . . 384
--semihosting (C-SPY command line option) . . . . . . . . 385
--silent (C-SPY command line option) . . . . . . . . . . . . . 385
--stlink_interface (C-SPY command line option) . . . . . 386
--stlink_reset_strategy (C-SPY command line option) . . . 386
--timeout (C-SPY command line option) . . . . . . . . . . . 387
--xds_rootdir (C-SPY command line option) . . . . . . . 387

Numerics

1x Units (Memory window context menu) ......... 161, 171
1x Units (Symbolic Memory window context menu) .. 168
2x Units (Memory window context menu) ......... 161, 168, 171
4x Units (Memory window context menu) ......... 161, 168, 171
8x Units (Memory window context menu) .......... 161