

IAR Embedded Workbench®

C-SPY® デバッガガイド

Advanced RISC Machines Ltd

ARM マイクロプロセッサファミリ



UCSARM-2-J

 **IAR**
SYSTEMS

著作権事項

Copyright © 1999–2011 IAR Systems AB.

IAR Systems AB が事前に書面で同意した場合を除き、このドキュメントを複製することはできません。このドキュメントに記載するソフトウェアは、正当な権限の範囲内でインストール、使用、およびコピーすることができます。

免責事項

このドキュメントの内容は、予告なく変更されることがあります。また、IAR Systems 社では、このドキュメントの内容に関して一切責任を負いません。記載内容には万全を期していますが、万一、誤りや不備がある場合でも IAR Systems 社はその責任を負いません。

IAR Systems 社、その従業員、その下請企業、またはこのドキュメントの作成者は、特殊な状況で、直接的、間接的、または結果的に発生した損害、損失、費用、課金、権利、請求、逸失利益、料金、またはその他の経費に対して一切責任を負いません。

商標

IAR Systems、IAR Embedded Workbench、C-SPY、visualSTATE、From Idea to Target、IAR KickStart Kit、IAR PowerPac、IAR YellowSuite、IAR Advanced Development Kit、IAR、および IAR Systems のロゴタイプは、IAR Systems AB が所有権を有する商標または登録商標です。J-Link は IAR Systems AB がライセンスを受けた商標です。

Microsoft および Windows は、Microsoft Corporation の登録商標です。

ARM および Thumb は、Advanced RISC Machines Ltd. の登録商標です。EmbeddedICE は、Advanced RISC Machines Ltd. の商標です。OCDemon は Macraigor Systems LLC の商標です。μC/OS-II は Micrium, Inc. の商標です。CMX-RTX は CMX Systems, Inc. の商標です。ThreadX は Express Logic の商標です。RTXC は、Quadros Systems の商標です。Fusion は、Unicoi Systems の商標です。

Adobe および Acrobat Reader は、Adobe Systems Incorporated の登録商標です。

その他のすべての製品名は、その所有者の商標または登録商標です。

改版情報

第 2 版 : 2011 年 4 月

部品番号 : UCSARM-2-J

本ガイドは、Advanced RISC Machines Ltd の ARM コアファミリ用 IAR Embedded Workbench® のバージョン 6.2x に適用する。

内部参照 : M10、Too6.3、IMAE。

目次（章）

表	17
図	19
はじめに	25
IAR C-SPY デバッガ	31
C-SPY を使用するにあたって	57
アプリケーションの実行	79
変数と式の扱い	97
ブレークポイントの使用	119
メモリとレジスタのモニタ	155
トレースデータの収集と使用	177
プロファイラの使用	223
Power ドメインのデバッグ	235
コードカバレッジ	251
割込み	255
C-SPY マクロの使用	277
C-SPY コマンドラインユーティリティ — cspybat	331
デバッガオプション	363
C-SPY ドライバについての追加情報	391
フラッシュローダの使用	403
索引	409

目次

表	17
図	19
はじめに	25
本ガイドの対象者	25
このガイドの使用方法	25
このガイドの概要	26
その他のドキュメント	27
ユーザガイドおよびリファレンスガイド	27
オンラインヘルプシステムを参照	28
Web サイト	28
表記規則	29
表記規則	29
命名規約	30
IAR C-SPY デバッガ	31
C-SPY の概要	31
統合環境	32
C-SPY デバッガの特長	32
RTOS 認識	34
デバッガの概念	35
C-SPY とターゲットシステム	35
デバッガ	35
ターゲットシステム	36
アプリケーション	36
C-SPY デバッガシステム	36
ROM モニタプログラム	36
サードパーティ製デバッガ	37
C-SPY プラグインモジュール	37
C-SPY の概要	37
C-SPY ドライバ間の差異	38

IAR C-SPY シミュレータ	39
特長	39
シミュレータドライバの選択	39
C-SPY J-Link ドライバ	39
特長	40
通信の概要	41
J-Link USB ドライバのインストール	41
C-SPY RDI ドライバ	42
特長	42
通信の概要	43
C-SPY Macraigor ドライバ	44
特長	44
通信の概要	45
C-SPY GDB サーバドライバ	45
特長	46
通信の概要	47
OpenOCD サーバの設定	47
C-SPY ST-LINK ドライバ	47
特長	48
通信の概要	49
ST-LINK バージョン 2 の ST-LINK USB ドライバの インストール	49
C-SPY TI Stellaris FTDI ドライバ	50
特長	50
通信の概要	51
FTDI USB ドライバのインストール	51
C-SPY Angel デバッグモニタドライバ	52
特長	52
通信の概要	52
C-SPY IAR ROM モニタドライバ	54
Analog Devices 評価ボードの特長	54
Philips LPC210x 用 IAR Kickstart Card の特長	54
通信の概要	55

C-SPY を使用するにあたって	57
C-SPY の設定	57
デバッグの設定	57
リセットからの実行	58
セットアップマクロファイルの使用	59
デバイス記述ファイルの選択	59
プラグインモジュールのロード	60
C-SPY の起動	60
デバッグの起動	61
IDE 外部でビルドされた実行可能ファイルのロード	61
ソースファイルがない状態でデバッグセッションを 開始する	61
複数イメージのロード	62
ターゲットハードウェアへの適合	63
デバイス記述ファイルの修正	63
C-SPY の起動前にターゲットハードウェアを初期化する	64
メモリの再配置	65
デバッグ起動の概要	66
フラッシュのコードのデバッグ	66
RAM のコードのデバッグ	67
サンプルプロジェクトの実行	67
サンプルプロジェクトの実行	68
C-SPY の起動についてのリファレンス情報	69
C-SPY デバッグメインウィンドウ	69
[イメージ] ウィンドウ	75
[別のファイルを取得] ダイアログボックス	77
アプリケーションの実行	79
アプリケーション実行の概要	79
アプリケーション実行の概要について	79
ソースモードと逆アセンブリモードのデバッグ	79
ステップ実行	80
アプリケーションの実行	83
強調表示	83

呼出しスタック情報	84
ターミナル I/O	84
デバッグログ	85
アプリケーション実行についてのリファレンス情報	85
[逆アセンブリ] ウィンドウ	86
[呼出しスタック] ウィンドウ	90
[ターミナル I/O] ウィンドウ	92
[ターミナル I/O ログファイル] ダイアログボックス	93
[デバッグログ] ウィンドウ	94
[ログファイル] ダイアログボックス	95
[自動ステップの設定] ダイアログボックス	96
変数と式の扱い	97
変数と式の扱いの概要	97
変数と式の扱いの概要について	97
C-SPY 式	98
変数情報の制限	100
アセンブラ変数の表示	101
変数と式の扱いの手順について	102
変数と式に関連するウィンドウの使用	102
アセンブラ変数の表示	102
データロギングを開始するには	103
変数と式の扱いについてのリファレンス情報	104
[自動] ウィンドウ	105
[ローカル] ウィンドウ	105
[ウォッチ] ウィンドウ	106
[ライブウォッチ] ウィンドウ	108
[静的] ウィンドウ	109
[静的] ダイアログボックスの選択	111
[クイック ウォッチ] ウィンドウ	112
[シンボル] ウィンドウ	113
[シンボルの曖昧さの解決] ダイアログボックス	114
[データログ] ウィンドウ	115
[データログ概要] ウィンドウ	117

ブレイクポイントの使用	119
ブレイクポイントの設定と使用の概要	119
ブレイクポイントを使用する理由	119
ブレイクポイントの設定の概略	120
ブレイクポイントの種類	120
ブレイクポイントアイコン	123
C-SPY シミュレータのブレイクポイント	123
C-SPY ハードウェアドライバのブレイクポイント	123
ブレイクポイントの設定元	124
[ブレイクポイント] オプション	125
例外ベクタのブレイクポイント	125
__ramfunc 宣言関数のブレイクポイントの設定	125
ブレイクポイントの設定の手順	126
ブレイクポイントのさまざまな設定方法	126
シンプルなコードブレイクポイントトグル	127
ダイアログボックスを使用したブレイクポイントの設定	127
[メモリ] ウィンドウでのデータブレイクポイントの設定	128
システムマクロを使用したブレイクポイントの設定	129
例外ベクタ上へのブレイクポイントの設定	130
ブレイクポイントのヒント	130
ブレイクポイントのリファレンス情報	132
[ブレイクポイント] ウィンドウ	133
[ブレイクポイントの使用] ダイアログボックス	135
[コード] ブレイクポイントダイアログボックス	136
[JTAG ウォッチポイント] ダイアログボックス	138
[ログ] ブレイクポイントダイアログボックス	141
[データ] ブレイクポイントダイアログボックス	143
[データログ] ブレイクポイントダイアログボックス	145
[ブレイクポイント] オプション	147
[イミディエイトブレイクポイント] ダイアログボックス	149
[ベクタキャッチ] ダイアログボックス	150
[位置入力] ダイアログボックス	150
[ソースの曖昧さの解決] ダイアログボックス	152

メモリとレジスタのモニタ	155
メモリとレジスタのモニタの概要	155
メモリとレジスタのモニタの概要について	155
C-SPY メモリゾーン	156
スタック表示	157
メモリアクセスチェック	158
メモリとレジスタについてのリファレンス情報	159
[メモリ] ウィンドウ	159
[メモリ保存] ダイアログボックス	163
[メモリ復元] ダイアログボックス	164
[フィル] ダイアログボックス	164
[シンボルメモリ] ウィンドウ	166
[スタック] ウィンドウ	168
[レジスタ] ウィンドウ	171
[メモリアクセス設定] ダイアログボックス	173
[メモリアクセスの編集] ダイアログボックス	175
トレースデータの収集と使用	177
トレースの使用の概要	177
トレースを使用する理由	177
トレースの概要	178
トレースを使用するための条件	179
トレースの使用の手順	180
C-SPY シミュレータでトレースを開始するには	180
ETM トレースを開始するには	181
SWO トレースを開始するには	182
ETM および SWO の並列使用の設定	182
ブレイクポイントを使用したトレースデータの収集	183
トレースデータの検索	184
トレースデータの参照	184
トレースのリファレンス情報	185
[ETM トレース設定] ダイアログボックス	186
[SWO トレースウィンドウ設定] ダイアログボックス	188
[SWO 設定] ダイアログボックス	190

[トレース] ウィンドウ	193
[トレースの保存] ダイアログボックス	198
[関数トレース] ウィンドウ	199
[タイムライン] ウィンドウ	200
[表示範囲] ダイアログボックス	206
[トレース開始ブレークポイント] ダイアログボックス (シミュレータ)	207
[トレース停止ブレークポイント] ダイアログボックス (シミュレータ)	208
[トレース開始] ブレークポイントダイアログボックス	209
[トレース停止] ブレークポイントダイアログボックス	213
[トレースフィルタ] ブレークポイントダイアログボックス	216
[トレース式] ウィンドウ	218
[トレースを検索] ダイアログボックス	219
[トレースを検索] ウィンドウ	221
プロファイラの使用	223
プロファイラの概要	223
プロファイラの用途	223
プロファイラの概要について	223
プロファイラの使用に関する要件	225
プロファイラの使用手順	225
関数レベルでプロファイラを使用するにあたって	226
命令レベルでプロファイラを使用するにあたって	226
プロファイリング情報の間隔を選択する	227
プロファイラについてのリファレンス情報	229
[関数プロファイラ] ウィンドウ	229
Power ドメインのデバッグ	235
Power デバッグの概要	235
Power デバッグを使用する理由	235
Power デバッグの概要	235
Power デバッグの要件	237

電力消費のソースコードの最適化	237
デバイスのステータスの待機	237
ソフトウェア遅延	237
DMA とポーリングされた I/O の比較	237
低電力モードの診断	238
CPU 周波数	238
誤ってアンアテンディングになっている周辺ユニットの検出	239
イベント駆動型システムでの周辺ユニット	239
衝突するハードウェア設定の検出	241
アナログ干渉	241
Power デバッグの手順	242
電力プロファイルの表示と結果の分析	242
アプリケーション実行中の予想外の電力消費の検出	243
Power デバッグのリファレンス情報	244
[Power ログ設定] ウィンドウ	244
[Power ログ] ウィンドウ	246
コードカバレッジ	251
コードカバレッジの概要	251
コードカバレッジを使用する理由	251
コードカバレッジの概要	251
コードカバレッジを使用するための要件	251
コードカバレッジについてのリファレンス情報	252
[コードカバレッジ] ウィンドウ	252
割込み	255
割込みの概要	255
割込みロギングの概要	255
割込みシミュレーションシステムの概要について	256
割込み特性	257
割込みシミュレーションの状態	258
割込みシミュレーションの C-SPY システムマクロ	259
ターゲットに合せた割込みシミュレーションシステムの調整	260

割込みの手順	260
シンプルな割込みシミュレーション	261
マルチタスクシステムでの割込みシミュレーション	262
C-SPY J-Link ドライバで割込みロギングを 使用するにあたって	263
割込みのリファレンス情報	264
[割込み設定] ダイアログボックス	264
[割込みの編集] ダイアログボックス	266
[強制割込み] ウィンドウ	268
[割込みステータス] ウィンドウ	269
[割込みログ] ウィンドウ	271
[割込みログ概要] ウィンドウ	274
C-SPY マクロの使用	277
C-SPY マクロの概要	277
C-SPY マクロを使用する理由	277
C-SPY マクロの使用の概要	278
セットアップマクロ関数およびファイルの概要	278
マクロ言語の概要	279
C-SPY マクロの使用手順	280
C-SPY マクロの登録 — 概要	280
C-SPY マクロの実行 — 概要	280
[マクロ設定] ダイアログボックスの使用	281
セットアップマクロとセットアップファイルによる 登録と実行	282
[クイックウォッチ] によるマクロの実行	283
ブレークポイントにマクロを接続して実行	284
マクロ言語についてのリファレンス情報	286
マクロ関数	286
マクロ変数	286
マクロ文字列	287
マクロ文	288
フォーマットした出力	289

予約済みのセットアップマクロ関数名についての リファレンス情報	291
C-SPY システムマクロについてのリファレンス情報	292
C-SPY コマンドラインユーティリティ — cspybat	331
C-SPY をバッチモードで使用	331
呼出し構文	331
出力	332
自動生成されたバッチファイルの使用	333
C-SPY コマンドラインオプションの概要	333
cspybat の一般オプション	333
すべての C-SPY ドライバで使用可能なオプション	334
シミュレータドライバで使用可能なオプション	335
C-SPY Angel デバッグモニタドライバで 使用可能なオプション	336
C-SPY GDB サーバドライバで使用可能なオプション	336
C-SPY IAR ROM-monitor ドライバで使用可能なオプション	336
C-SPY J-Link/J-Trace ドライバで使用可能なオプション	336
C-SPY TI Stellaris FTDI ドライバで使用可能なオプション	336
C-SPY Macraigor ドライバで使用可能なオプション	337
C-SPY RDI ドライバで使用可能なオプション	337
C-SPY ST-LINK ドライバで使用可能なオプション	337
C-SPY サードパーティ製ドライバで使用可能なオプション	337
C-SPY コマンドラインオプションについてのリファレンス 情報	338
デバッグオプション	363
デバッグオプションの設定	363
デバッグオプションについてのリファレンス情報	365
設定	365
ダウンロード	367
追加オプション	368
イメージ	369
プラグイン	370

C-SPY ドライバオプションのリファレンス情報	371
Angel	371
GDB サーバ	372
IAR ROM モニタ	373
J-Link/J-Trace の設定オプション	374
J-Link/J-Trace 接続オプション	379
TI Stellaris FTDI の設定オプション	381
Macraigor	382
RDI	385
ST-LINK	386
サードパーティ製ドライバのオプション	388
C-SPY ドライバについての追加情報	391
C-SPY シミュレータ	391
シミュレータのメニュー	392
C-SPY GDB サーバドライバ	393
[GDB サーバ] メニュー	393
C-SPY J-Link/J-Trace ドライバ	394
J-Link メニュー	394
ライブウォッチおよび DCC の使用	396
ターミナル I/O および DCC の使用	397
C-SPY TI Stellaris FTDI ドライバ	397
TI Stellaris FTDI メニュー	397
C-SPY Macraigor ドライバ	398
Macraigor の [JTAG] メニュー	398
C-SPY RDI ドライバ	398
RDI メニュー	399
C-SPY ST-LINK ドライバ	399
ST-LINK メニュー	400
フラッシュローダの使用	403
フラッシュローダの概要	403
フラッシュローダの概要について	403
フラッシュローダの設定	403
フラッシュローディング機構	404

フラッシュローダについてのリファレンス情報	405
[フラッシュローダの概要] ダイアログボックス	405
[フラッシュローダの構成] ダイアログボックス	407
索引	409

表

1: このガイドの表記規則	29
2: このガイドで使用されている命名規約	30
3: ドライバ間の差異	38
4: 利用可能なクイックスタートリファレンス情報	58
5: C-SPY アセンブラシンボル式	99
6: ハードウェアレジスタとアセンブララベルの 名前が衝突する場合の処理	99
7: 式の種類による表示フォーマット設定の影響	107
8: 式の種類による表示フォーマット設定の影響	110
9: ブレークポイント用の C-SPY マクロ	129
10: [タイムライン] ウィンドウでサポートされているグラフ	201
11: C-SPY ドライバのプロファイリングのサポート	225
12: プロファイラを有効にするためのプロジェクトオプション	226
13: コードカバレッジを有効にするためのプロジェクトオプション	253
14: タイマ割り込み設定	262
15: C-SPY マクロ変数の例	287
16: C-SPY セットアップマクロ	291
17: システムマクロのまとめ	292
18: __cancelInterrupt のリターン値	295
19: __disableInterrupts のリターン値	296
20: __driverType のリターン値	297
21: __emulatorSpeed のリターン値	298
22: __enableInterrupts のリターン値	299
23: __evaluate リターン値	299
24: __hwReset のリターン値	300
25: __hwResetRunToBp のリターン値	301
26: __hwResetWithStrategy のリターン値	302
27: __isBatchMode のリターン値	302
28: __jtagResetTRST のリターン値	308
29: __loadImage のリターン値	308
30: __openFile のリターン値	311

31: __readFile リターン値	313
32: __setCodeBreak のリターン値	317
33: __setDataBreak のリターン値	319
34: __setLogBreak のリターン値	320
35: __setSimBreak のリターン値	321
36: __setTraceStartBreak のリターン値	322
37: __setTraceStopBreak のリターン値	324
38: __sourcePosition リターン値	324
39: __unloadImage のリターン値	327
40: cspybat のパラメータ	331
41: 使用する C-SPY ドライバに固有のオプション	364
42: 例外の取得	386



1: C-SPY とターゲットシステム	35
2: C-SPY J-Link 通信の概要	41
3: C-SPY RDI 通信の概要	43
4: C-SPY Macraigor 通信の概要	45
5: C-SPY GDB サーバ通信の概要	47
6: C-SPY ST-LINK 通信の概要	49
7: C-SPY TI Stellaris FTDI の通信の概要	51
8: C-SPY Angel デバッグモニタ通信の概要	53
9: C-SPY ROM モニタ通信の概要	55
10: [別のファイルを取得] ダイアログボックス	62
11: フラッシュのコードをデバッグする場合のデバッガの起動	66
12: RAM のコードをデバッグする場合のデバッガの起動	67
13: サンプルアプリケーション	68
14: [デバッグ] メニュー	71
15: [逆アセンブリ] メニュー	73
16: [イメージ] ウィンドウ	75
17: [イメージ] ウィンドウのコンテキストメニュー	76
18: [別のファイルを取得] ダイアログボックス	77
19: C-SPY で強調表示されるソース位置	83
20: C-SPY の [逆アセンブリ] ウィンドウ	86
21: [逆アセンブリ] ウィンドウのコンテキストメニュー	88
22: [呼出しスタック] ウィンドウ	90
23: [呼出しスタック] ウィンドウのコンテキストメニュー	91
24: [ターミナル I/O] ウィンドウ	92
25: [制御コード] メニュー	92
26: [入力モード] ダイアログボックス	93
27: [ターミナル I/O ログファイル] ダイアログボックス	93
28: [デバッグログ] ウィンドウ (メッセージウィンドウ)	94
29: [デバッグログ] ウィンドウのコンテキストメニュー	94
30: [ログファイル] ダイアログボックス	95
31: [自動ステップの設定] ダイアログボックス	96

32:	[ウォッチ] ウィンドウでのアセンブラ変数の表示	101
33:	[ウォッチ] ウィンドウでのアセンブラ変数の表示	103
34:	[自動] ウィンドウ	105
35:	[ローカル] ウィンドウ	105
36:	[ウォッチ] ウィンドウ	106
37:	[ウォッチ] ウィンドウのコンテキストメニュー	107
38:	[ライブウォッチ] ウィンドウ	108
39:	[静的] ウィンドウ	109
40:	[静的] ウィンドウのコンテキストメニュー	110
41:	[静的変数の選択] ダイアログボックス	111
42:	[クイックウォッチ] ウィンドウ	112
43:	[シンボル] ウィンドウ	113
44:	[シンボル] ウィンドウのコンテキストメニュー	113
45:	[シンボルの曖昧さの解決] ダイアログボックス	114
46:	[データログ] ウィンドウ	115
47:	[データログ概要] ウィンドウ	117
48:	ブレークポイントアイコン	123
49:	コンテキストメニューからのブレークポイントの変更	128
50:	[ブレークポイント] ウィンドウ	133
51:	[ブレークポイント] ウィンドウのコンテキストメニュー	133
52:	[ブレークポイントの使用] ダイアログボックス	135
53:	[コード] ブレークポイントダイアログボックス	136
54:	[JTAG ウォッチポイント] ダイアログボックス	138
55:	[ログ] ブレークポイントダイアログボックス	141
56:	[データブレークポイント] ダイアログボックス	143
57:	[データログ] ブレークポイントダイアログボックス	145
58:	[ブレークポイント] オプション	147
59:	[イミディエイトブレークポイント] ダイアログボックス	149
60:	[ベクタキャッチ] ダイアログボックス (ARM9/Cortex-R4 と Cortex-M3 との比較)	150
61:	[位置入力] ダイアログボックス	150
62:	[ソースの曖昧さの解決] ダイアログボックス	152
63:	C-SPY のゾーン	157
64:	[メモリ] ウィンドウ	159

65: [メモリ] ウィンドウのコンテキストメニュー	161
66: [メモリ保存] ダイアログボックス	163
67: [メモリ復元] ダイアログボックス	164
68: [フィル] ダイアログボックス	164
69: [シンボルメモリ] ウィンドウ	166
70: [シンボルメモリ] ウィンドウのコンテキストメニュー	167
71: [スタック] ウィンドウ	168
72: [スタック] ウィンドウのコンテキストメニュー	170
73: [レジスタ] ウィンドウ	171
74: [メモリアクセス設定] ダイアログボックス	173
75: [メモリアクセスの編集] ダイアログボックス	175
76: [ETM トレース設定] ダイアログボックス	186
77: [SWO トレースウィンドウ設定] ダイアログボックス	188
78: [SWO 設定] ダイアログボックス	190
79: シミュレータの [トレース] ウィンドウ	193
80: [トレースの保存] ダイアログボックス	198
81: [関数トレース] ウィンドウ	199
82: [タイムライン] ウィンドウ	200
83: 呼出しスタックグラフの [タイムライン] ウィンドウの コンテキストメニュー	203
84: [表示範囲] ダイアログボックス	206
85: [トレース開始] ブレークポイントダイアログボックス (シミュレータ)	207
86: [トレース停止] ブレークポイントダイアログボックス (シミュレータ)	208
87: [トレース開始] ブレークポイントダイアログボックス (J-Link/J-Trace)	210
88: [トレース停止] ブレークポイントダイアログボックス (J-Link/J-Trace)	213
89: [トレースフィルタ] ブレークポイントダイアログボックス	216
90: [トレース式] ウィンドウ	218
91: [トレースを検索] ダイアログボックス	219
92: [トレースを検索] ウィンドウ	221
93: [逆アセンブリ] ウィンドウの命令カウント	227

94: Power グラフで選択された間隔	228
95: 間隔モードの [関数プロファイラ] ウィンドウ	228
96: [関数プロファイラ] ウィンドウ	229
97: [関数プロファイラ] ウィンドウのコンテキストメニュー	232
98: イベント駆動型システムでの電力消費	240
99: オシロスコープにより記録されたノイズの上昇	241
100: [Power 設定] ウィンドウ	244
101: [Power 設定] ウィンドウのコンテキストメニュー	245
102: [Power ログ] ウィンドウ	246
103: [Power ログ] ウィンドウのコンテキストメニュー	248
104: [コードカバレッジ] ウィンドウ	252
105: [コードカバレッジ] ウィンドウのコンテキストメニュー	254
106: 擬似割込みの構成	257
107: シミュレーション状態 — 例 1	258
108: シミュレーション状態 — 例 2	259
109: [割込み設定] ダイアログボックス	264
110: [割込みの編集] ダイアログボックス	266
111: [強制割込み] ウィンドウ	268
112: [強制割込み] ウィンドウのコンテキストメニュー	268
113: [割込みステータス] ウィンドウ	269
114: [割込みログ] ウィンドウ	271
115: [割込みログ] ウィンドウのコンテキストメニュー	273
116: [割込みログ概要] ウィンドウ	274
117: [マクロ設定] ダイアログボックス	281
118: [クイックウォッチ] ウィンドウ	284
119: [デバッガ] 設定オプション	365
120: C-SPY ダウンロードオプション	367
121: デバッガのその他のオプション	368
122: デバッガのイメージオプション	369
123: デバッガのプラグインオプション	370
124: C-SPY Angel オプション	371
125: [GDB サーバ] のオプション	372
126: IAR ROM モニタオプション	373
127: J-Link/J-Trace 設定オプション	374

128: J-Link/J-Trace 接続オプション	379
129: TI Stellaris FTDI 設定オプション	381
130: Macraigor オプション	382
131: RDI オプション	385
132: ST-LINK (設定オプション)	386
133: C-SPY サードパーティ製ドライバオプション	388
134: [シミュレータ] メニュー	392
135: [GDB サーバ] メニュー	393
136: J-Link メニュー	394
137: [TI Stellaris FTDI] メニュー	397
138: Macraigor の [JTAG] メニュー	398
139: RDI メニュー	399
140: [ST-LINK] メニュー	400
141: [フラッシュローダの概要] ダイアログボックス	405
142: [フラッシュローダの構成] ダイアログボックス	407

はじめに

『ARM 用 C-SPY® デバッガガイド』へようこそ。本書の目的は、IAR C-SPY® デバッガの機能を十分に理解して、ARM コアに基づいたアプリケーションのデバッグに役立てることです。

本ガイドの対象者

本ガイドは、C-SPY で利用可能なすべての機能を活用する場合に利用してください。また、以下について十分な知識があるユーザを対象としています。

- C/C++ プログラミング言語
- 組込みシステム用アプリケーションの開発
- ARM コアのアーキテクチャ、命令セット（チップメーカーのドキュメントを参照）
- ホストコンピュータのオペレーティングシステム

IDE に統合されている他の開発ツールの詳細は、それぞれのドキュメントを参照（27 ページの *その他のドキュメント* を参照）してください。

このガイドの使用方法

IAR Embedded Workbench を初めて使用する場合は、まずガイド *『IAR Embedded Workbench® の使用開始の手順』* で、IDE で利用可能なツールおよび機能の概要を確認することをお勧めします。

IAR Embedded Workbench の使用経験者で、IAR システムズの開発ツールの使用方法を再確認する必要がある場合は、IAR インフォメーションセンタのチュートリアルから始めることをお勧めします。プロジェクト管理、ビルド、編集手順については、*『ARM 用 IDE プロジェクト管理およびビルドガイド』* で説明しています。C-SPY を使用したデバッグの方法については、本書に説明があります。

本書では多くのトピックをカバーしており、各トピックのセクションにはそのトピックに関する概念を説明した概要が含まれます。これに目を通せば、C-SPY の機能が良く理解できます。さらに、トピックのセクションではステップごとの手順が説明されており、機能を使用する上で役に立ちます。最後に、各トピックのセクションには適切なリファレンス情報がすべて記載されています。

また、IAR システムズのユーザガイドおよびリファレンスガイドで不明な用語がある場合は、『*ARM 用 IDE プロジェクト管理およびビルドガイド*』の用語集も推奨します。

このガイドの概要

本ガイドの構成および各章の概要を以下に示します。

- 「*IAR C-SPY デバッガ*」は、**C-SPY** デバッガおよび一般的なデバッグと **C-SPY** に関する概念について説明します。また、さまざまな **C-SPY** ドライバについても解説します。本章では、さまざまな **C-SPY** ドライバ機能の違いについて簡単に説明します。
- 「*C-SPY を使用するにあたって*」は、**C-SPY** の使用を開始するための準備（設定や起動、ターゲットハードウェアに合わせた **C-SPY** の調整など）について説明します。
- 「*アプリケーションの実行*」では、ソースモードと逆アセンブリモードのデバッグの違い、アプリケーションの実行機能、ターミナルの **I/O** の操作方法について説明します。
- 「*変数と式の扱い*」では、**C-SPY** で使用する式や変数の構文、および変数の制限について説明します。また、変数や式のモニタ方法についても説明します。
- 「*ブレークポイントの使用*」では、ブレークポイントシステムとブレークポイントの設定方法について説明します。
- 「*メモリとレジスタのモニタ*」では、メモリやレジスタの内容を調べる方法について説明します。
- 「*トレースデータの収集と使用*」は、トレースデータを使用して特定の状態までプログラムのフローを点検する方法について説明します。
- 「*プロファイラの使用*」では、実行中に最も多くの時間を費やす、プロファイラを使用したアプリケーションソースコードでの関数の検索方法について説明します。
- 「*Power ドメインのデバッグ*」では、**Power** デバッグのテクニックと、**C-SPY** を使用して予想外の電力消費につながるソースコード構造体を探す方法について説明します。
- 「*コードカバレッジ*」では、コードカバレッジ機能を使用して、コードのすべての部分が実行されたか検証し、それによって実行されていない部分を特定する方法について説明します。
- 「*割込み*」では、**C-SPY** の割込みシミュレーションシステムの詳細と、ターゲットハードウェアの割込みに応じて割込みシミュレーションを設定する方法について説明します。

- 「*C-SPY* マクロの使用」では、**C-SPY** のマクロシステムとその機能、マクロの用途、マクロの使用方法について説明します。
- 「*C-SPY* コマンドラインユーティリティ *cspybat*」では、バッチモードでの **C-SPY** の使用方法について説明します。
- 「デバッグオプション」では、**C-SPY** デバッグを起動する前に設定しなければならないオプションについて説明します。
- 「*C-SPY* ドライバについての追加情報」では、他のトピックに記載されていない、**C-SPY** ドライバのメニューおよび機能について説明します。
- 「フラッシュローダの使用」では、フラッシュローダの機能と利用方法について説明します。

その他のドキュメント

ユーザドキュメンテーションは、ハイパーテキスト PDF 形式、およびコンテキスト依存のオンラインヘルプシステム（HTML フォーマット）があります。ドキュメンテーションには、インフォメーションセンタあるいは **IAR Embedded Workbench IDE** の **【ヘルプ】** メニューからアクセスできます。オンラインヘルプシステムは、F1 キーを押しても使用できます。

ユーザガイドおよびリファレンスガイド

IAR システムズの各開発ツールについては、一連のガイドで説明しています。知りたい情報に対応するドキュメントを以下に示します。

- **IAR** システムズの製品のインストールおよび登録の要件と詳細については、同梱されているクイックレファレンスのブックレットおよび『インストールとライセンス登録ガイド』をご覧ください。
- **IAR Embedded Workbench** および提供されるツールの利用にあたっては、『**IAR Embedded Workbench®** の使用開始の手順』を参照してください。
- プロジェクト管理とビルドでの **IDE** の使用については、『**ARM** 用 **IDE** プロジェクト管理およびビルドガイド』を参照してください。
- **ARM** 用 **IAR C/C++** コンパイラのプログラミングおよび **IAR ILINK** リンカを使用したリンクについては、『**ARM** 用 **IAR C/C++** 開発ガイド』を参照してください。
- **ARM** 用 **IAR** アセンブラを使用したプログラミングについては、『**ARM** 用 **IAR** アセンブラリファレンスガイド』を参照してください。
- **IAR DLIB** ライブラリの使用については、オンラインヘルプで利用できる **DLIB** ライブラリリファレンス情報を参照してください。

- ARM 用 IAR Embedded Workbench の旧バージョンで開発したアプリケーションコードやプロジェクトの移植については、『*ARM 用 IAR Embedded Workbench® 移行ガイド*』を参照してください。
- MISRA-C ガイドラインを使用して、安全性を最重要視したアプリケーションを開発する方法については、『*IAR Embedded Workbench® MISRA-C:2004 リファレンスガイド*』または『*IAR Embedded Workbench® MISRA-C:1998 リファレンスガイド*』を参照してください。
- IAR J-Link と IAR J-Trace については、『*IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores*』（ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド）を参照してください。

注：製品のインストール内容によっては、他のドキュメントも提供される場合があります。

オンラインヘルプシステムを参照

コンテキスト依存のオンラインヘルプの内容は以下のとおりです。

- IAR C-SPY® デバッガを使用したデバッグについての包括的な情報
- IDE のメニューやウィンドウ、ダイアログボックスに関するリファレンス情報
- コンパイラのリファレンス情報
- DLIB ライブラリ関数のキーワードリファレンス情報 関数のリファレンス情報を確認するには、エディタウィンドウで関数名を選択し、F1 キーを押します

WEB サイト

推奨 Web サイト：

- Advanced RISC Machines Ltd の Web サイト (www.arm.com) には、the ARM コアに関する情報とニュースが記載されています。
- IAR システムズの Web サイト (www.iar.com/jp) では、アプリケーションノートおよびその他の製品情報を公開しています。
- C 標準化作業グループの Web サイト、www.open-std.org/jtc1/sc22/wg14。
- C++ Standards Committee の Web サイト、www.open-std.org/jtc1/sc22/wg21。
- Embedded C++ Technical Committee の Web サイト (www.caravan.net/ec2plus) には、Embedded C++ 規格についての情報が公開されています。

表記規則

本ガイドでプログラミング言語 C と記述されている場合、特に記述がない限り C++ も含まれます。

たとえば `arm¥doc` のように、製品のインストール先のディレクトリを指す場合、その場所のフルパスが想定されます。たとえば、`c:¥Program Files¥IAR Systems¥Embedded Workbench 6.n¥arm¥doc` のようになります。

表記規則

このガイドでは、次の表記規則を使用します。





スタイル	用途
コンピュータ	<ul style="list-style-type: none">ソースコードの例、ファイルパス。コマンドライン上のテキスト。2 進数、16 進数、8 進数。
パラメータ	パラメータとして使用される実際の値を表すプレースホルダ。 たとえば、 <code>filename.h</code> の場合、 <code>filename</code> はファイルの名前を表します。
[オプション]	コマンドのオプション部分
[a b c]	代替の選択肢を持つコマンドのオプション部分
{a b c}	コマンドの必須部分に選択肢があることを示します。
太字	画面に表示されるメニュー名、メニューコマンド、ボタン、およびダイアログボックス
斜体	<ul style="list-style-type: none">本ガイドや他のガイドへのクロスリファレンスを示します。強調。 <p>3 点リーダーは、その前の項目を任意の回数繰り返せることを示します。</p>
	IAR Embedded Workbench IDE 固有の内容を示します。
	コマンドラインインタフェース固有の内容を示します。
	開発やプログラミングについてのヒントを示します。
	ワーニングを示します。

表 1: このガイドの表記規則

命名規約

以下の命名規約は、このガイドに記述されている IAR システムズの製品およびツールで使用されています。

ブランド名	一般名称
ARM 用 IAR Embedded Workbench®	IAR Embedded Workbench®
ARM 用 IAR Embedded Workbench® IDE	IDE
ARM 用 IAR C-SPY® デバッガ	C-SPY、デバッガ
IAR C-SPY® シミュレータ	シミュレータ
ARM 用 IAR C/C++ コンパイラ	コンパイラ
ARM 用 IAR アセンブラ	アセンブラ
IAR ILINK リンカ	ILINK、リンカ
IAR DLIB ライブラリ	DLIB ライブラリ

表 2: このガイドで使用されている命名規約

IAR C-SPY デバッガ

この章では、IAR C-SPY® デバッガおよび一般的なデバッグと C-SPY に関する概念について説明します。また、さまざまな C-SPY ドライバについても解説します。具体的には以下の項目を解説します。

- C-SPY の概要
- デバッガの概念
- C-SPY の概要
- IAR C-SPY シミュレータ
- C-SPY J-Link ドライバ
- C-SPY RDI ドライバ
- C-SPY Macraigor ドライバ
- C-SPY GDB サーバドライバ
- C-SPY ST-LINK ドライバ
- C-SPY TI Stellaris FTDI ドライバ
- C-SPY Angel デバッグモニタドライバ
- C-SPY IAR ROM モニタドライバ

C-SPY の概要

このセクションでは、以下のトピックについて説明します。

- 統合環境
- C-SPY デバッガの特長
- RTOS 認識

統合環境

C-SPY は、組み込みアプリケーション用の高級言語デバッガです。IAR システムズのコンパイラおよびアセンブラとともに使用することを目的としており、IDE に完全に統合されているので、同一アプリケーション内での開発とデバッグが可能になります。このため、以下のような操作が可能です。

- デバッグ中の編集。デバッグセッション中に、デバッグの制御に使用するものと同じソースコードウィンドウで直接修正を行うことができます。変更内容は、次にプロジェクトをリビルドしたときに有効になります。
- 開発サイクル中の任意の位置へのブレークポイントの設定。デバッガを実行していないときもブレークポイント定義を確認および修正することができます。ブレークポイント定義フローを編集中のテキストで確認および修正できます。ウォッチプロパティ、ウィンドウレイアウト、レジスタグループなどのデバッグ設定は、デバッグセッションが終わるまで保持されます。

Embedded Workbench のワークスペースで開いているウィンドウはすべて、C-SPY デバッガを起動してもそのまま開いています。それ以外に、C-SPY 固有の複数のウィンドウが開かれます。

C-SPY デバッガの特長

IAR システムズはツールチェーン全体を提供しているため、コンパイラやリンカの出力にデバッガ用の詳細なデバッグ情報を含めることができ、デバッグ時に非常に役立ちます。

C-SPY は以下のような一般的特長があります。

- ソースおよび逆アセンブリレベルのデバッグ
C-SPY では、C/C++ とアセンブラの両方のソースコードで、ソースと逆アセンブリのデバッグを必要に応じて切り替えることができます。
- 関数呼出しのステップ実行
ソースレベルのステップ実行の最小単位が行単位である従来のデバッガとは異なり、C-SPY ではすべての文および関数呼出しがステップポイントとして認識されるため、より詳細な制御が可能です。つまり、式に含まれる呼出しと、他の関数への呼出しのパラメータとして記述されている呼出しの両方をステップ実行できます。パラメータ中の呼出しのステップ実行は、オブジェクトのコンストラクタなどのように、多数の追加関数呼出しが行われる C++ コードで特に便利です。
- コード/データブレークポイント
C-SPY のブレークポイントシステムでは、デバッグ対象のアプリケーションでさまざまなブレークポイントを設定し、目的箇所を実行を停止することができます。たとえば、ブレークポイントを設定して、プログラムロジックが正しいかどうかや、データアクセスに設定してデータが変更されたタイミングと方法を調べたりできます。

- 変数および式のモニタ

変数および式の場合、さまざまな機能から選択できます。任意の変数および式をポップアップ表示で評価できます。指定した式の値をより長時間モニタおよび記録することができます。ローカル変数をその場で制御でき、処理の中断なくリアルタイムでデータが表示されます。さらに、最後に参照した変数が自動的に表示されます。

- コンテナ認識

C-SPY でアプリケーションを実行すると、STL の list や vector などのライブラリデータ型のエレメントを表示することができます。これにより、C++ STL コンテナの使用時に内容を簡単に把握してデバッグすることができます。

- 呼出しスタック情報

コンパイラは、詳細な呼出しスタック情報を生成します。これにより、実行に影響することなく、プログラムカウンタがどこを指していても、関数呼出しの完全な呼出しスタックをデバッガで表示できます。呼出しスタックで任意の関数を選択し、その関数についてローカル変数やレジスタの情報を表示することができます。

- 強力なマクロシステム

C-SPY は強力な内蔵マクロシステムを装備しており、複雑なアクションを定義して実行することができます。C-SPY マクロは、単独で、あるいはブレークポイントや割込みシミュレーションシステム（シミュレータ使用時）と組み合わせて使用し、さまざまなタスクを実行できます。

C-SPY デバッガのその他の特長

他にも、以下のような特長があります。

- スレッド実行により、ターゲットアプリケーションの実行中も IDE の応答性を維持
- 自動ステップ実行
- ソースブラウザで、関数、型、変数を簡単に表示可能
- ささまざまな変数の型を認識
- 設定可能なレジスタ（CPU、周辺）、メモリウィンドウ
- オーバフローの検出機能を持った、グラフィック表示のスタックビュー
- コードカバレッジ、関数レベルのプロファイルをサポート
- ターゲットアプリケーションは、ファイル I/O
- オプションのターミナル I/O エミュレーション

RTOS 認識

C-SPY は、リアルタイム OS 対応のデバッグをサポートしています。以下のオペレーティングシステムが現在サポートされています。

- CMX-RTX
- CMX-Tiny+
- eForce μ C3/Compact
- eSysTech X realtime kernel
- Express Logic ThreadX
- FreeRTOS、OpenRTOS、SafeRTOS
- Freescale MQX
- Micrium μ C/OS-II
- Micro Digital SMX
- MISPO NORTi
- OSEK (ORTI)
- RTXC Quadros
- Segger embOS
- unicon Fusion.

IAR システムズかサードパーティ製の RTOS プラグインモジュールを使用できます。サポートされている RTOS モジュールについては、ソフトウェア販売代理店か IAR システムズの担当者までお問い合わせください。また、IAR システムズの Web サイトでも情報を提供しています。

C-SPY RTOS 認識プラグインモジュールを使用すると、RTOS 上で構築されたアプリケーションを高度に制御し、モニタできます。モニタできるのは、タスクリスト、キュー、セマフォ、メールボックス、さまざまな RTOS システム変数などの RTOS 固有の項目です。タスク固有のブレークポイントとタスク固有のステップ実行により、タスクを簡単にデバッグできます。

デバッグセッションが開始されると、ロードされたプラグイン独自のメニューとウィンドウ、ボタンが追加されます (RTOS がアプリケーションとリンクされている場合)。他の RTOS 認識プラグインモジュールについては、メーカーのプラグインモジュールを参照してください。RTOS 文書へのリンクについては、[ヘルプ] メニューから入手可能なリリースノートを参照してください。

デバッガの概念

このセクションでは、デバッグの一般的な概念と用語、特に C-SPY に関連する内容について説明します。このセクションは、C-SPY の機能に関する具体的な情報を含んでいません。それらの情報については、このパートの各章で説明します。IAR システムズのユーザドキュメントでは、デバッガの概念を説明するときに、このセクションで説明されている用語を使用します。

C-SPY とターゲットシステム

C-SPY は、ソフトウェアターゲットシステムとハードウェアターゲットシステムの両方のデバッグに使用できます。

以下の図は、C-SPY およびターゲットシステムの概要を示します。

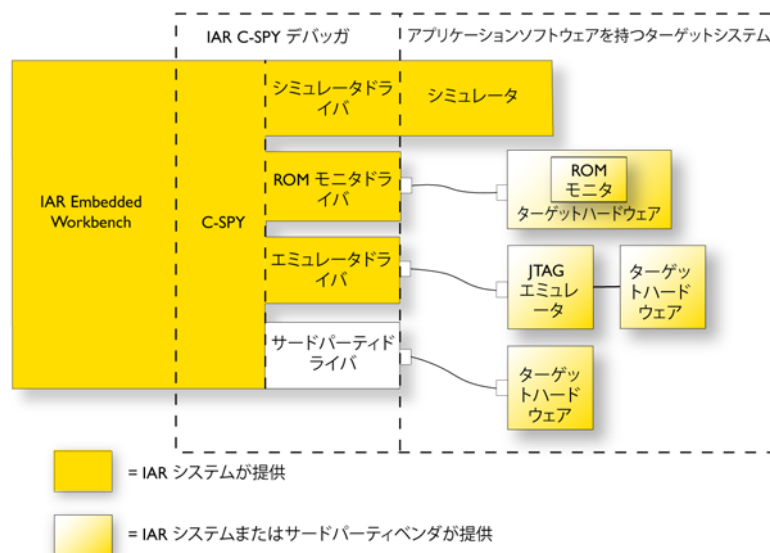


図 1: C-SPY とターゲットシステム

デバッガ

C-SPY のようなデバッガは、ターゲットシステム上でアプリケーションをデバッグするためのプログラムです。

ターゲットシステム

ターゲットシステムは、アプリケーションをデバッグするときに、それを実行するシステムです。ターゲットシステムは、評価ボードか独自に設計したハードウェアから構成されます。また、その全体または一部をソフトウェアでシミュレーションすることもできます。ターゲットシステムは、その種類ごとに専用の C-SPY ドライバを必要とします。

アプリケーション

ユーザアプリケーションは、ユーザが開発し、C-SPY を使用してデバッグを行うソフトウェアです。

C-SPY デバッガシステム

C-SPY は、デバッガの基本機能セットを提供する部分とターゲット固有のバックエンドで構成されています。バックエンドは、各マイクロコントローラに 1 つずつあってマイクロコントローラのプロパティを定義するプロセッサモジュールと、C-SPY ドライバの 1 つから構成されています。C-SPY ドライバは、ターゲットシステムとの通信およびターゲットシステムの管理を提供します。また、特殊ブレークポイントなど、ターゲットシステムが提供する機能へのユーザインタフェースとして、メニュー、ウィンドウ、ダイアログボックスを提供します。一般的に、C-SPY ドライバには、大きく分けて以下の 3 つの種類があります。

- シミュレータドライバ
- ROM モニタドライバ
- エミュレータドライバ

C-SPY にはシミュレータドライバのほか、製品パッケージによっては、ハードウェアデバッガシステム用の追加ドライバもインストールされます。C-SPY ドライバおよび各ドライバの機能については、37 ページの C-SPY の概要を参照してください。

ROM モニタプログラム

ROM モニタプログラムは、ターゲットハードウェアの不揮発性メモリにロードされるファームウェアで、アプリケーションと並行して動作します。ROM モニタは、デバッガと通信して、ステップ実行やブレークポイントなど、アプリケーションのデバッグに必要なサービスを提供します。

サードパーティ製デバッガ

サードパーティ製デバッガも IAR システムズのツールチェーンに組み込むことができます。その場合、サードパーティ製デバッガは、ELF/DWARF、Intel-extended、または Motorola の読み込みが可能であることが必要です。サードパーティ製デバッガで使用するフォーマットについては、そのデバッガに付属のユーザドキュメントを参照してください。

C-SPY プラグインモジュール

C-SPY は、プラグインモジュールとしてデバッガの追加機能を実装するとき使用できる、オープン SDK で構築したモジュール化構造アーキテクチャとして設計されています。これらのモジュールは、IDE にシームレスに統合できます。

IAR システムズがサードパーティ製のプラグインモジュールを使用できます。このようなモジュールの例を以下に示します。

- [コードカバレッジ]、[スタック] ウィンドウ（これらはすべて IDE に統合されます）
- 特定のデバッグシステムを使用するための様々な C-SPY ドライバ
- リアルタイム OS 対応のデバッグをサポートする RTOS プラグインモジュール
- C-SPY が周辺ユニットをシミュレートするための周辺デバイスシミュレーションモジュール。このようなプラグインモジュールは、IAR システムズでは提供されていませんが、サードパーティサプライヤが開発および提供できます
- IAR visualSTATE と IAR Embedded Workbench 間のインタフェースとして機能する C-SPYLink。これは、標準 C レベルシンボリックデバッグのほか、真の意味での高水準ステートマシンデバッグを C-SPY で直接可能にします。詳細については、IAR visualSTATE の付属ドキュメントを参照してください

C-SPY SDK の詳細については、IAR システムズまでお問い合わせください。

C-SPY の概要

本書の執筆時点では、ARM コアの IAR C-SPY デバッガでは、以下に示すターゲットシステムのドライバと評価ボードを使用できます。

- シミュレータ
- J-Link/J-Trace JTAG プロローブ
- RDI (Remote Debug Interface)
- Macraigor JTAG プロローブ
- GDB サーバ
- ST-LINK JTAG プロローブ (ST Cortex-M デバイス専用)

- TI Stellaris FTDI JTAG インタフェース（Cortex デバイス専用）
- P&E Microcomputer システム。このドライバの詳細は、arm¥doc ディレクトリ
のドキュメント、「*P&E Microcomputer システムインタフェースでARM
用 IAR Workbench デバッグを使用するための設定*」を参照してください
- Angel デバッグモニタ
- Analog Devices ADuC7xxx ボード用 IAR ROM モニタ、Philips LPC210x 用 IAR
Kickstart Card

注：IAR Embedded Workbench で提供するドライバに加えて、サードパーティ
ベンダ製のデバッグドライバもロードできます。388 ページの サードパー
ティ製ドライバのオプションを参照してください。

C-SPY ドライバ間の差異

以下の表に、C-SPY ドライバ間の主な差異の概要を示します。

機能	シミュ レータ	J-Link/ J-Trace	RDI	Mac- raigor	GDB サーバ	ST- LINK	TI Stellaris FTDI	Angel	IAR ROM- モニタ
コードブレークポ イント	無制限	x	x	x	x	x	x	x	x
データブレークポ イント	x	x	x	x	x	x	x	--	--
割込みロギング	x	x 4)	--	--	--	x 4)	--	--	--
データロギング	--	x 4)	--	--	--	x 4)	--	--	--
ライブウォッチ	--	x 2)	--	--	--	x 2)	--	--	--
サイクルカウンタ	x	x 5)	--	--	--	x 5)	--	--	--
コードカバレッジ	x	x 1)	--	--	--	x 1)	--	--	--
データカバレッジ	x	--	--	--	--	--	--	--	--
関数 / 命令プロ ファイラ	x	x 3)	--	--	--	x 3)	--	--	--
トレース	x	x 3)	x	--	--	--	--	--	--

表 3: ドライバ間の差異

1) J-Trace および ETB 付属の J-link で使用可能。Cortex-M デバイスの場合、SWO 付きの J-Link
および ST-LINK は部分的コードカバレッジをサポートします。コードカバレッジの情報については、
「251 ページの コードカバレッジ」を参照してください。

2) Cortex デバイスでサポートされています。ARM7/9 デバイスについては、ライブウォッチは、
DCC ハンドラをアプリケーションに追加した場合にサポートされます。396 ページの ライブ
ウォッチおよび DCC の使用を参照してください。

3) SWD/SWO インタフェースまたは ETM トレースに必要です。

4) SWD/SWO を持つ Cortex

5) Cortex-M デバイスの場合のみ。

IAR C-SPY シミュレータ

C-SPY シミュレータは、ターゲットプロセッサの機能をソフトウェアで完全にシミュレーションするため、ハードウェアがすべて揃ってなくてもプログラムロジックをデバッグできます。ハードウェアが不要であるため、多くのアプリケーションにとって最も費用効果の高いソリューションでもあります。

特長

C-SPY の一般的な特長に加えて、シミュレータには以下のような特長があります。

- 命令レベルのシミュレーション
- メモリの構成、検証
- 割込みシミュレーション
- イミディエイトブレークポイントと C-SPY マクロシステムを使用した周辺シミュレーション

シミュレータドライバの選択

C-SPY を起動する前に、シミュレータドライバを選択する必要があります。

- 1 IDE で、[プロジェクト] > [オプション] を選択して、[デバッガ] カテゴリの [設定] タブをクリックします。
- 2 [ドライバ] ドロップダウンリストで [シミュレータ] を選択します。

C-SPY J-Link ドライバ

ARM IAR J-Link/J-Trace ドライバを使用すると、C-SPY では IAR J-Link JTAG デバッグプローブおよび IAR J-Trace JTAG デバッグプローブに接続できます。JTAG は、ほとんどの ARM プロセッサで利用可能な、標準オンチップデバッグ接続です。

USB ポート経由で J-Link/J-Trace JTAG プローブを使用するには、まず Segger J-Link/J-Trace USB ドライバをインストールする必要があります。41 ページの *J-Link USB ドライバのインストール* を参照してください。ドライバは IAR Embedded Workbench for ARM のインストール CD にあります。

J-Link ドライバでデバッグセッションを開始すると、[J-Link] メニューがデバッガメニューバーに追加されます。メニューコマンドの詳細については、394 ページの *J-Link メニュー* を参照してください。

特長

C-SPY の一般的な特長に加えて、J-Link ドライバには以下のような特長があります。

- リアルタイム実行
- USB 経由の通信
- ターゲットシステムのゼロメモリフットプリント
- ARM コアの利用可能な2つのハードウェアブレークポイントを利用すると、フラッシュなどの不揮発性メモリのコードがデバッグ可能。Cortex デバイスでは、6 つのハードウェアブレークポイントをサポート
- ARM コアウォッチポイントレジスタに直接アクセス
- ブレークポイント数の制限なし (RAM でコードをデバッグ時)
- デバッガでは、ターゲットシステムをリセットしたり停止せずに実行中のアプリケーションに接続が可能

注：コードカバレッジは、J-Trace および ETB 付属の J-Link で使用可能です。Cortex-M デバイスの場合、SWO 付きの J-Link は部分的コードカバレッジをサポートします。Cortex デバイス用の C-SPY J-Link/J-Trace ドライバは、ライブウォッチをサポートしています。ARM7/9 デバイスについては、DCC ハンドラがご利用のアプリケーションに追加されている場合にサポートされます。

通信の概要

C-SPY J-Link ドライバでは、USB 接続経由で JTAG プロブと通信します。続いて、その JTAG プロブはハードウェアの JTAG モジュールと通信します。

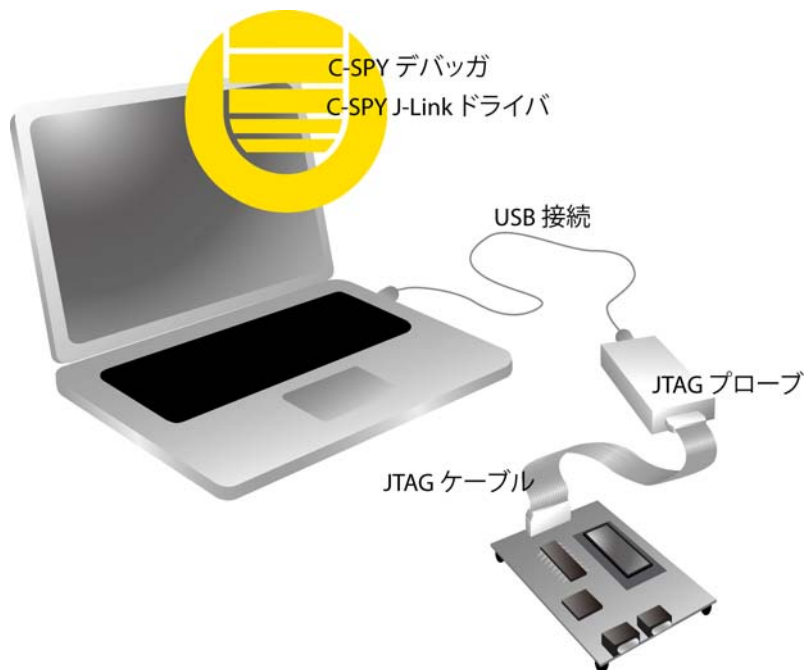


図 2: C-SPY J-Link 通信の概要

J-LINK USB ドライバのインストール

USB ポート経由で J-Link JTAG プロブを使用するには、まず Segger J-Link USB ドライバをインストールする必要があります。

- 1 ARM 用 IAR Embedded Workbench をインストールします。
- 2 USB ケーブルを使用してコンピュータと J-Link を接続します。まだターゲットボードには J-Link を接続してはいけません。Windows が USB ドライバを検索している間、J-Link の前面パネルにある緑の LED が数秒間点滅します。

J-Link とコンピュータの接続は初めてであるため、Windows ではダイアログボックスを開き、USB ドライバがある場所を尋ねます。USB ドライバは、`arm\drivers\JLink` ディレクトリの IAR Embedded Workbench 製品インストール内にあります。

```
x86¥JLink.inf、x64¥JLinkx64.inf
x86¥JLink.sys、x64¥JLinkx64.sys
```

初期設定が終了すると、再度ドライバをインストールする必要はありません。

USB ドライバが J-Link プローブとの接続を確立するまで、J-Link では点滅が継続することに注意してください。接続が確立されると、J-Link では接続されていることを示すために、安定して点灯し始めます。

C-SPY RDI ドライバ

IAR C-SPY RDI ドライバを使用すると、C-SPY では RDI に準拠したデバッグシステムに接続できます。たとえば、シミュレータ、ROM モニタ、JTAG プローブ、エミュレータなどです。IAR C-SPY RDI ドライバは RDI 仕様 1.5.1 に準拠しています。

このセクションでは、RDI に準拠した JTAG プローブへの接続を前提とします。JTAG は、ほとんどの ARM プロセッサで利用可能な、標準オンチップデバッグ接続です。

RDI 準拠の JTAG プローブを使用するには、まず JTAG プローブベンダが提供する RDI ドライバ DLL をインストールする必要があります。

次に Embedded Workbench IDE では、RDI ドライバ DLL ファイルの位置を特定する必要があります。この位置を特定するには、[プロジェクト] > [オプション] を選択し、[C-SPY デバッガ] カテゴリを選択します。[設定] ページで、[ドライバ] ドロップダウンリストから [RDI] を選択します。[RDI] ページでは、[メーカ供給 RDI ドライバ] ブラウズボタンを使用して、RDI ドライバ DLL ファイルの位置を特定します。使用可能なその他のオプションの詳細については、385 ページの *RDI* を参照してください。RDI ドライバ DLL をロードすると、[RDI] メニューが Embedded Workbench IDE のメニューバーに表示されます。このメニューには、選択した RDI ドライバ DLL に関連する設定ダイアログボックスが表示されます。なお、このダイアログボックスは各 RDI ドライバ DLL によって異なります。

特長

IAR C-SPY デバッガの一般的な特長に加えて、RDI ドライバには以下のような特長があります。

- リアルタイム実行
- USB、イーサネット、またはパラレルポート経由の高速通信（RDI 互換の JTAG プローブを使用する場合）
- ターゲットシステムのゼロメモリフットプリント

- ARMコアの利用可能な2つのハードウェアブレイクポイントを利用すると、フラッシュなどの不揮発性メモリのコードがデバッグ可能
- ブレイクポイント数の制限なし (RAM でコードをデバッグ時)
- デバッガでは、ターゲットシステムをリセットせずに実行中のアプリケーションに接続が可能

通信の概要

RDI ドライバ DLL では、パラレル、シリアル、イーサネット、または USB の接続で、JTAG プロブと通信します。続いて、その JTAG プロブはハードウェアの JTAG モジュールと通信します。

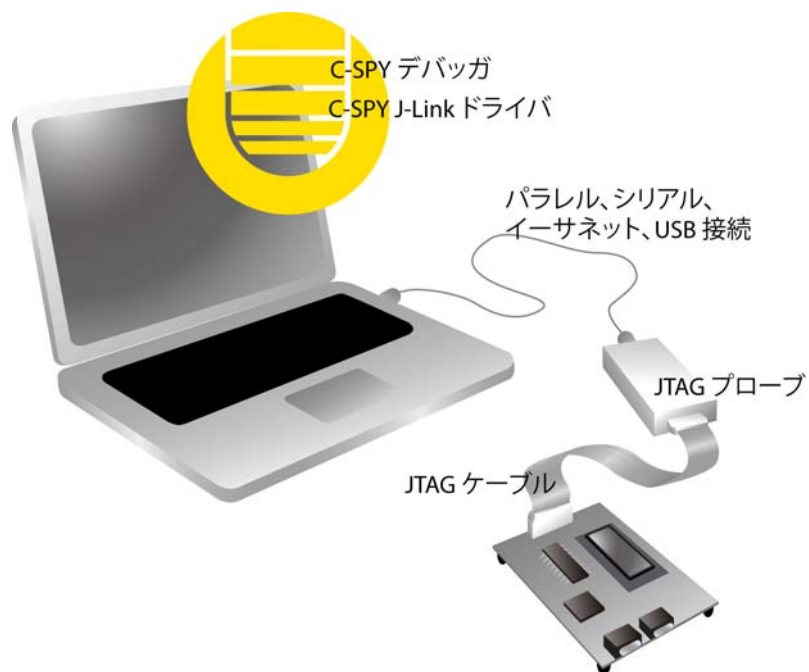


図3: C-SPY RDI 通信の概要

詳細については、`arm\doc\infocenter` ディレクトリの `rdi_quickstart.html` ファイルを参照するか、メーカーの資料を参照してください。

C-SPY Macraigor ドライバ

IAR C-SPY Macraigor ドライバを使用すると、C-SPY では Macraigor mpDemon、USB2 Demon、USB2 Sprite JTAG プローブに接続できます。JTAG は、ほとんどの ARM プロセッサで利用可能な、標準オンチップデバッグ接続です。

パラレルポートまたは USB ポート経由で Macraigor JTAG プローブを使用するには、まず Macraigor OCDemon ドライバをインストールする必要があります。ドライバは、ARM の IAR Embedded Workbench CD にあります。このドライバはシリアルおよびイーサネットの接続には必要ありません。

Macraigor ドライバでデバッグセッションを開始すると、**[JTAG]** メニューがデバッガのメニューバーに追加されます。このメニューには、JTAG ウォッチポイントを設定するためのコマンドと、例外ベクタ（別名はベクタキャッチ）にブレークポイントを設定するためのコマンドが表示されます。メニューコマンドの詳細については、398 ページの *Macraigor の [JTAG] メニュー* を参照してください。

特長

IAR C-SPY デバッガの一般的な特長に加えて、Macraigor JTAG ドライバには以下のような特長があります。

- リアルタイム実行
- イーサネットまたは USB 経由の通信
- ターゲットシステムのゼロメモリフットプリント
- ARM コアの利用可能な2つのハードウェアブレークポイントを利用すると、フラッシュなどの不揮発性メモリのコードがデバッグ可能
- ARM コアウォッチポイントレジスタに直接アクセス
- ブレークポイント数の制限なし（RAM でコードをデバッグ時）
- デバッガでは、ターゲットシステムをリセットせずに実行中のアプリケーションに接続が可能

通信の概要

C-SPY Macraigor ドライバでは、USB またはイーサネットの接続で、JTAG プロブと通信します。続いて、その JTAG プロブはハードウェアの JTAG モジュールと通信します。

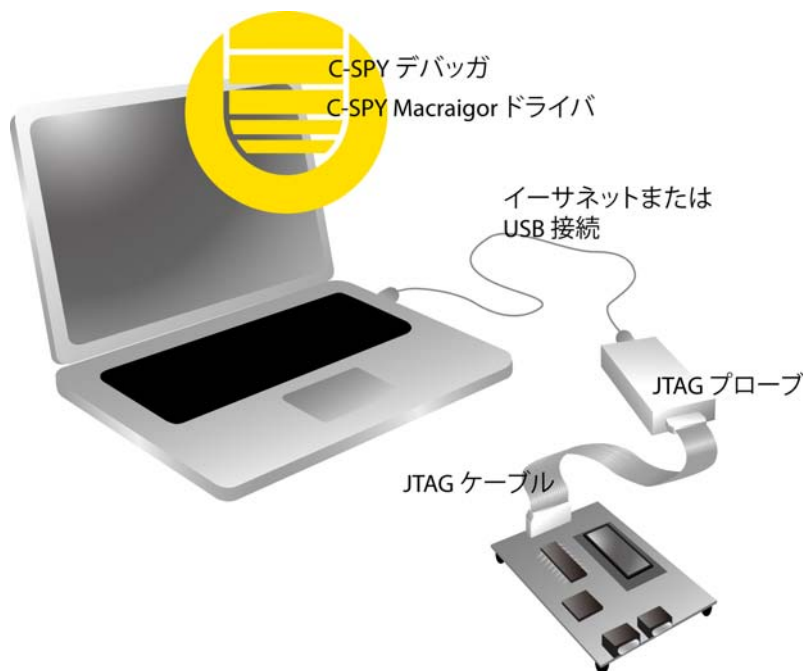


図 4: C-SPY Macraigor 通信の概要

C-SPY GDB サーバドライバ

IAR GDB サーバドライバを使用すると、C-SPY では、GDB サーバベースの使用可能な JTAG ソリューション（現在は STR9-comStick の OpenOCD）に接続できます。JTAG は、ほとんどの ARM プロセッサで利用可能な、標準オンチップデバッグ接続です。

GDB サーバベースの JTAG ソリューションを使用するには、対象のハードウェアとソフトウェアドライバを設定する必要があります。47 ページの *OpenOCD サーバの設定* を参照してください。

C-SPY GDB サーバドライバでデバッグセッションを開始すると、[GDB サーバ] メニューがデバッガメニューバーに追加されます。メニューコマンドの詳細については、393 ページの [GDB サーバ] メニューを参照してください。

特長

IAR C-SPY デバッガの一般的な特長に加えて、GDB サーバドライバには (OpenOCD を介して) 以下のような特長があります。

- STR9、Cortex-M および他のデバイスのサポート
- リアルタイム実行
- USB 経由の通信
- ターゲットシステムのゼロメモリフットプリント
- ARM7/9 デバイスおよび Cortex デバイスで、それぞれ 2 つまたは 6 つのハードウェアブレークポイントを使用
- ブレークポイント数の制限なし (RAM でコードをデバッグ時)

通信の概要

C-SPY GDB サーバドライバは、イーサネット接続経由で GDB サーバと通信し、GDB サーバは USB 接続経由で JTAG プロブと通信します。続いて、その JTAG プロブはハードウェアの JTAG モジュールと通信します。

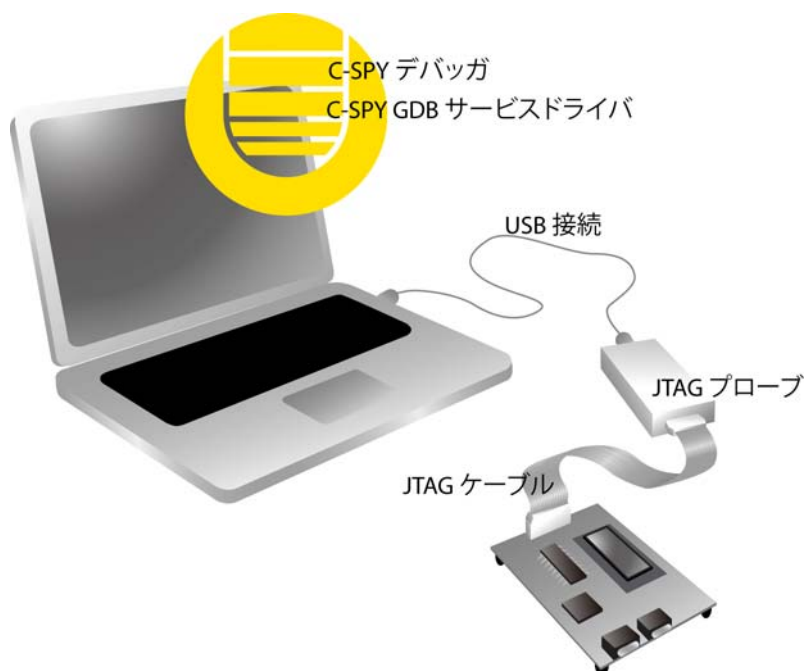


図5: C-SPY GDB サーバ通信の概要

OPENOCD サーバの設定

詳細については、`arm¥doc¥infocenter` ディレクトリの `gdbserv_quickstart.html` ファイルを参照するか、メーカーの資料を参照してください。

C-SPY ST-LINK ドライバ

IAR C-SPY ST-LINK ドライバを使用すると、C-SPY では ST-LINK JTAG プロブに接続できます。プロブの両方のバージョンがサポートされています。JTAG は、ほとんどの ARM プロセッサで利用可能な、標準オンチップデバッグ接続です。

USB ポート経由で ST-LINK JTAG プローブを使用するには、まず ST-LINK USB ドライバをインストールする必要があります (49 ページの *ST-LINK* バージョン2 の *ST-LINK USB* ドライバのインストールを参照)。

特長

IAR C-SPY デバッガの一般的な特長に加えて、ST-LINK ドライバには以下のような特長があります。

- ST Cortex-M デバイスのサポート
- リアルタイム実行
- USB 経由の通信
- ターゲットシステムのゼロメモリフットプリント
- 6 つの使用可能なハードウェアブレークポイントを使用
- 部分的コードカバレッジ
- ブレークポイント数の制限なし (RAM でコードをデバッグ時)

通信の概要

C-SPY ST-LINK ドライバでは、USB 接続経由で JTAG プローブと通信します。続いて、その JTAG プローブはハードウェアの JTAG モジュールと通信します。

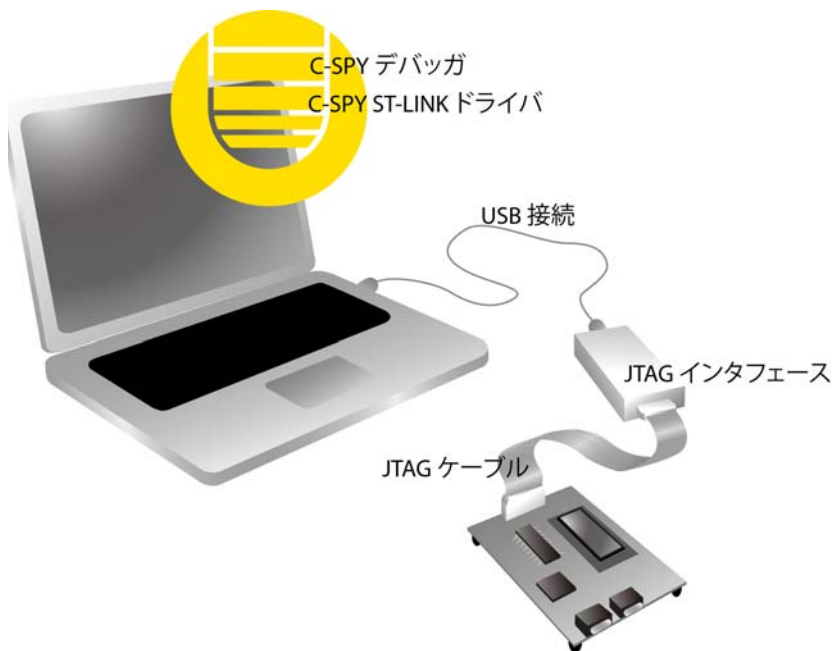


図 6: C-SPY ST-LINK 通信の概要

ST-LINK ドライバを使用する場合の C-SPY 環境の詳細については、400 ページの *ST-LINK* メニューを参照してください。

ST-LINK バージョン 2 の ST-LINK USB ドライバのインストール

USB ポート経由で ST-LINK バージョン 2 の JTAG プローブを使用するには、まず ST-LINK USB ドライバをインストールする必要があります。

- 1 ARM 用 IAR Embedded Workbench をインストールします。
- 2 USB ケーブルを使用してコンピュータと ST-LINK を接続します。まだターゲットボードには ST-LINK を接続してはいけません。

ST-LINK とコンピュータの接続は初めてであるため、Windows ではダイアログボックスを開き、USB ドライバがある場所を尋ねます。USB ドライバは、`arm¥drivers¥ST-Link` ディレクトリの IAR Embedded Workbench 製品インストール内にあります: `ST-Link_V2_USBdriver.exe`。

初期設定が終了すると、再度ドライバをインストールする必要はありません。

C-SPY TI Stellaris FTDI ドライバ

IAR C-SPY TI Stellaris FTDI ドライバを使用すると、C-SPY では Cortex-M デバイス用の TI Stellaris FTDI 搭載 JTAG インタフェースに接続できます。

USB ポート経由で FTDI JTAG インタフェースを使用するには、まず FTDI USB ドライバをインストールする必要があります。ドライバは IAR Embedded Workbench for ARM のインストール CD にあります。

FTDI ドライバでデバッグセッションを開始すると、**[TI Stellaris FTDI]** メニューがデバッガのメニューバーに追加されます。メニューコマンドの詳細については、397 ページの *TI Stellaris FTDI* メニューを参照してください。

特長

IAR C-SPY デバッガの一般的な特長に加えて、TI Stellaris FTDI ドライバには以下のような特長があります。

- TI Stellaris FTDI Cortex-M デバイスのサポート
- リアルタイム実行
- USB 経由の通信
- ターゲットシステムのゼロメモリフットプリント
- 6 つの使用可能なハードウェアブレイクポイントを使用
- ブレイクポイント数の制限なし (RAM でコードをデバッグ時)

通信の概要

C-SPY TI Stellaris FTDI ドライバは、USB 経由でハードウェア上の FTDI JTAG インタフェースチップと通信します。FTDI JTAG インタフェースは、マイクロコントローラの JTAG ポートに接続します。

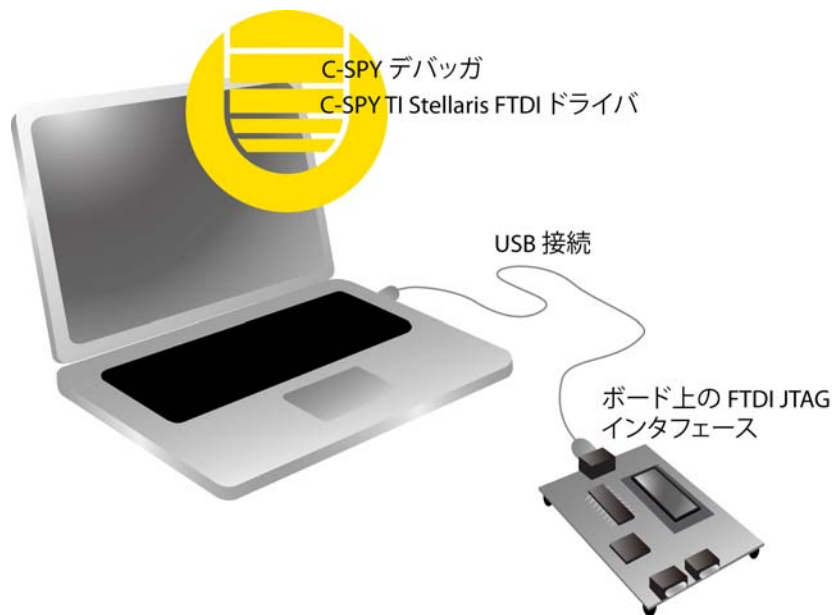


図7: C-SPY TI Stellaris FTDI の通信の概要

FTDI USB ドライバのインストール

USB ポート経由で TI Stellaris FTDI JTAG インタフェースを使用するには、まず FTDI USB ドライバをインストールする必要があります。

- 1 ARM 用 IAR Embedded Workbench をインストールします。
- 2 USB ケーブルを使用してコンピュータと TI ボードを接続します。

FTDI とコンピュータの接続は初めてであるため、Windows ではダイアログボックスを開き、USB ドライバがある場所を尋ねます。USB ドライバは、製品をインストールした `arm\drivers\StellarisFTDI` ディレクトリにあります。

初期設定が終了すると、再度ドライバをインストールする必要はありません。

C-SPY Angel デバッグモニタドライバ

IAR Angel デバッグモニタドライバを使用すると、Angel デバッグモニタプロトコルに準拠するすべてのデバイスと通信できます。ほとんどの場合、これらは評価ボードです。評価ボードに接続すると、Angel ファームウェアはご利用のアプリケーションと同時に実行します。

以後このセクションでは、Angel が評価ボードと接続されているものと仮定します。

特長

IAR C-SPY デバッガの一般的な特長に加えて、Angel デバッグモニタドライバには以下のような特長があります。

- リアルタイム実行
- シリアルポートまたはイーサネット経由の通信
- Angel が搭載されているすべての評価ボードのサポート

通信の概要

この評価ボードには、アプリケーションソフトウェアと同時に実行するファームウェア（Angel デバッグモニタ自体）が搭載されています。このファームウェアでは、シリアルポートまたはイーサネット接続経由で IAR C-SPY デバッガからコマンドを受け取り、アプリケーションの実行を制御します。

Angel プロトコルを使用すると、C-SPY ではフラッシュに Angel モニタを備えるターゲットシステムに接続できます。このプロトコルは、必要なものはシリアルケーブルだけであるため、ターゲットのデバッグに費用のかからない方法です。

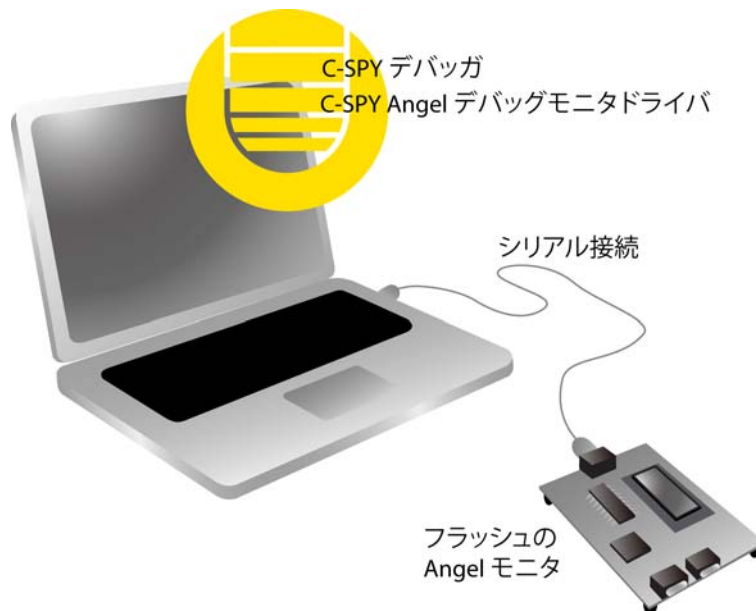


図 8: C-SPY Angel デバッグモニタ通信の概要

デバッグしたいソースコードのすべての部分を RAM で探す必要があります。ブレークポイントの設定やアプリケーションのステップインを行う唯一の方法が、コードを RAM にダウンロードすることです。

詳細については、`arm\doc\infocenter` ディレクトリの `angel_quickstart.html` ファイルを参照するか、メーカーの資料を参照してください。

C-SPY IAR ROM モニタ ドライバ

IAR ROM モニタドライバを使用すると、C-SPY では Analog Devices ADuC7xxx ボード、Philips LPC210x 用 IAR Kickstart Card に接続できます。ほとんどの ROM モニタでは、デバッグするコードを RAM に置く必要があります。これは、ブレークポイントの設定やアプリケーションコードのステップインを行う唯一の方法が、コードを RAM にダウンロードすることであるためです。ROM モニタによっては（Analog Devices ADuC7xxx の場合など）、デバッグするコードはフラッシュメモリにあってもかまいません。デバッグ機能を維持するために、ROM モニタではいくつかの命令をシミュレートする場合（シングルステップを行うときなど）があります。

ANALOG DEVICES 評価ボードの特長

IAR C-SPY デバッガの一般的な特長に加えて、ROM モニタドライバには以下のような特長があります。

- リアルタイム実行
- シリアルポート経由の通信
- Analog Devices ADuC7xxx 評価ボードのサポート
- フラッシュメモリでのダウンロードおよびデバッグ
- フラッシュメモリと RAM のブレークポイント数が無制限

PHILIPS LPC210X 用 IAR KICKSTART CARD の特長

IAR C-SPY デバッガの一般的な特長に加えて、ROM モニタドライバには以下のような特長があります。

- リアルタイム実行
- RS232 シリアルポート経由の通信
- Philips LPC210x 用 IAR Kickstart Card のサポート

通信の概要

このボードには、アプリケーションソフトウェアと同時に実行するファームウェア（ROM モニタ自体）が搭載されています。このファームウェアでは、シリアルポート接続経由で IAR C-SPY デバッガからコマンドを受け取り、アプリケーションの実行を制御します。

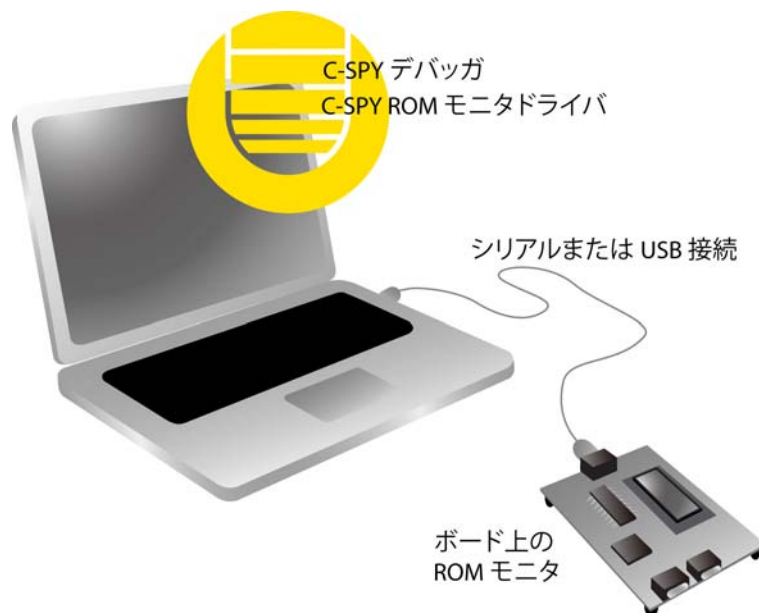


図9: C-SPY ROM モニタ通信の概要

詳細については、[arm¥doc¥infocenter](#) ディレクトリの [iar_rom_quickstart.html](#) ファイルを参照するか、メーカーの資料を参照してください。

C-SPY を使用するにあたって

この章では、C-SPY® の使用に必要な基本事項について説明します。
具体的には以下の項目を解説します。

- C-SPY の設定
- C-SPY の起動
- ターゲットハードウェアへの適合
- デバッグ起動の概要
- サンプルプロジェクトの実行
- C-SPY の起動についてのリファレンス情報

C-SPY の設定

このセクションでは、C-SPY の設定に必要な手順について説明します。
具体的には、以下の項目について説明します。

- デバッグの設定
- リセットからの実行
- セットアップマクロファイルの使用
- デバイス記述ファイルの選択
- プラグインモジュールのロード

デバッグの設定

- I C-SPY ドライバで必要とする場合、USB ドライバをインストールしてください。詳細については、以下を参照してください。

- 41 ページの *J-Link USB* ドライバのインストール
- 49 ページの *ST-LINK* バージョン2 の *ST-LINK USB* ドライバのインストール
- 51 ページの *FTDI USB* ドライバのインストール

- 2 C-SPY を起動する前に、[プロジェクト] > [オプション] > [デバッガ] > [設定] を選択し、シミュレータやハードウェアデバッグシステムの中から使用するデバッグシステムに合った C-SPY ドライバを選んでください。

注：選択できるのは、コンピュータにインストール済みのドライバのみです。

- 3 [カテゴリ] リストで、適切な C-SPY ドライバを選択して設定を行います。
これらのオプションについては、363 ページの デバッガオプションを参照してください。
- 4 [OK] をクリックします。
- 5 [ツール] > [オプション] > [デバッガ] を選択して以下を設定します。

- デバッガの動作
- デバッガによるスタック使用率のトラッキング

これらのオプションについては、『ARM 用 IDE プロジェクト管理およびビルドガイド』を参照してください。

さまざまなデバッグシステムの設定方法が記載された以下のドキュメントが、arm\doc\infocenter サブディレクトリから入手できます。

ファイル	デバッグシステム
rdi_quickstart.html	RDI コントロール JTAG デバッグインタフェース用 クイックスタートリファレンス
gdbserver_quickstart.html	STR9-comStick とともに OpenOCD を使用した GDB サーバ用クイックスタートリファレンス
angel_quickstart.html	Angel ROM モニタおよび JTAG インタフェース用ク イックスタートリファレンス
iar_rom_quickstart.html	IAR ROM モニタ用クイックスタートリファレンス

表 4: 利用可能なクイックスタートリファレンス情報

63 ページの ターゲットハードウェアへの適合も参照してください。

リセットからの実行

[指定位置まで実行] オプション（[デバッガ] > [設定] ページ）では、デバッガの起動時やリセット時の C-SPY の実行先の位置を指定します。C-SPY は指定された位置に一時的なブレークポイントを配置して、そこまでのすべてのコードを実行してから、その位置で停止します。

デフォルトでは、main 関数まで実行します。他の位置まで実行する場合は、その位置の名前を入力します。アセンブララベルかそれに相当するもの（関数名など）を指定できます。

チェックボックスを選択していない場合は、リセットごとにプログラムカウンタに通常のハードウェアリセットアドレスが格納されます。C-SPY によりリセットアドレスが設定されます。

C-SPY の起動時にブレークポイントが設定されていない場合は、ステップ実行をする必要があることと、それには非常に時間がかかることを通知する警告メッセージが表示されます。そこでステップ実行を継続するか、最初の命令で停止するかを選択できます。最初の命令で停止することを選択した場合、デバッガは **【指定位置まで実行】** ボックスで入力した位置ではなく、PC（プログラムカウンタ）に格納されているデフォルトリセット位置から実行を開始します。

注：C-SPY シミュレータではブレークポイントに制限がないので、このメッセージは表示されません。

セットアップマクロファイルの使用

セットアップマクロファイルは、C-SPY の起動時に自動的にロードするように指定するマクロファイルです。セットアップマクロ関数とシステムマクロを使用することにより、ニーズに合せたアクションを実行するセットアップマクロファイルを定義できます。したがって、セットアップマクロファイルをロードすることによって、アクションを自動的に実行するように C-SPY を初期化できます。

セットアップマクロファイルと関数の詳細については、278 ページの *セットアップマクロ関数およびファイルの概要* を参照してください。セットアップマクロファイルの使用例については、64 ページの *C-SPY の起動前にターゲットハードウェアを初期化する* を参照してください。

セットアップマクロファイルを登録するには、以下の手順に従います。

- 1 C-SPY を起動する前に、**【プロジェクト】** > **【オプション】** > **【デバッガ】** > **【設定】** を選択します。
- 2 **【マクロファイルの使用】** を選択して、Setup.mac というようにセットアップマクロファイルのパスと名前を入力します。ファイル名拡張子の入力を省略した場合、拡張子は mac とみなされます。

デバイス記述ファイルの選択

C-SPY はデバイス記述ファイルを使用して、デバイス固有の情報を処理します。デバイス記述ファイルは、IAR 固有のデバイス記述ファイルか CMSIS システムビュー記述ファイル (SVD) のどちらかのフォーマットです。

プロジェクト設定に応じて、デフォルトのデバイス記述ファイルが自動的に使用されます。デフォルトのファイルをオーバーライドする場合、デバイス記述ファイルを選択する必要があります。IAR 固有のデバイス記述ファイルは、armvconfig ディレクトリにあり、ファイル名の拡張子は ddf です。

デバイス記述ファイルの詳細については、63 ページの *ターゲットハードウェアへの適合* を参照してください。

デフォルトのデバイス記述ファイルをオーバーライドするには、次の手順に従います。

- 1 C-SPY を起動する前に、[プロジェクト] > [オプション] > [デバッガ] > [設定] を選択します。
- 2 デバイス記述ファイルの使用を有効にし、[デバイス記述ファイル] 参照ボタンを使用してファイルを選択します。

プラグインモジュールのロード

[プラグイン] ページでは、デバッグセッションでロードして使用する C-SPY プラグインモジュールを指定します。IAR システムズやサードパーティ製のプラグインモジュールを使用できます。使用可能なモジュールについては、ソフトウェア販売代理店または IAR システムズの担当者までお問い合わせください。また、IAR システムズの Web サイトでも情報を提供しています。

詳しくは、370 ページの *プラグイン* を参照してください。

C-SPY の起動

デバッガを設定したら、デバッグセッションを開始する準備は終わりです。このセクションでは、必要な手順について説明します。

具体的には、以下の項目について説明します。

- デバッガの起動
- IDE 外部でビルドされた実行可能ファイルのロード
- ソースファイルがない状態でデバッグセッションを開始する
- 複数イメージのロード

デバッガの起動

デバッガは、現在のプロジェクトをロードしてもしなくても起動できます。



C-SPY を起動して現在のプロジェクトをロードするには、[ダウンロードしてデバッグ] ボタンをクリックします。または、[プロジェクト] > [ダウンロードしてデバッグ] を選択してください。



現在のプロジェクトをリロードせずに C-SPY を起動するには、[ダウンロードせずにデバッグ] ボタンをクリックしてください。または、[プロジェクト] > [ダウンロードせずにデバッグ] を選択してください。

IDE 外部でビルドされた実行可能ファイルのロード

IDE 外でビルドされたアプリケーション、たとえばコマンドラインでビルドされたアプリケーションとともに C-SPY をロードすることもできます。外部でビルドされた実行可能ファイルをロードしてビルドのオプションを設定するには、最初にワークスペース内にそのファイル用のプロジェクトを作成する必要があります。

外部でビルドされたファイルにプロジェクトを作成するには、次の手順に従います。

- 1 [プロジェクト] > [新規プロジェクトの作成] を選択して、プロジェクト名を指定します。
- 2 プロジェクトに実行可能ファイルを追加するには、[プロジェクト] > [ファイルの追加] を選択して、[ファイルの種類] ドロップダウンリストで [すべてのファイル] を選択します。実行可能ファイルを指定します。



- 3 実行可能ファイルを起動するには、[ダウンロードしてデバッグ] ボタンをクリックします。プロジェクトは、実行可能ファイルをリビルドするたびに再利用できます。

このようなプロジェクトで設定する意味があるプロジェクトオプションは、[一般オプション] カテゴリと [デバッガ] カテゴリにだけあります。プロジェクトの一般オプションは、実行可能ファイルをビルドしたときと同様に設定するようにしてください。

ソースファイルがない状態でデバッグセッションを開始する

通常は、IAR Embedded Workbench IDE を使用してソースファイルを編集したり、プロジェクトのビルド、デバッグセッションを開始する場合、必要なすべてのファイルが入手できてプロセスは想定したとおりに機能します。

ただし、C-SPY は自動的にソースファイルを見つけることはできません。たとえば、アプリケーションが別のコンピュータでビルドされた場合、**「別のファイルを取得」** ダイアログボックスが表示されます。

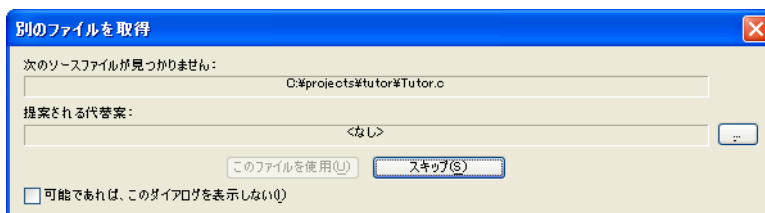


図 10: 「別のファイルを取得」ダイアログボックス

通常は以下のような場合にこのダイアログボックスを使用できます。

- ソースファイルが使用できない場合。**「可能であれば、このダイアログを表示しない」** に続いて **「スキップ」** をクリックします。C-SPY は、単に入手可能なソースファイルがないと見なします。ダイアログボックスは再び表示されず、デバッグセッションではソースコードは表示されません。
- 代替のソースファイルが別の場所にある場合。代替のソースコードを指定して、**「可能であれば、このダイアログを表示しない」** をクリックし、**「このファイルを使用」** をクリックします。C-SPY は代替ファイルを使用するものと想定します。代替ファイルが指定されておらず、自動的に検索されないファイルが必要な場合を除いて、ダイアログボックスは再び表示されません。

IAR Embedded Workbench IDE を再起動すると、**「可能であれば、このダイアログを表示しない」** をクリックした場合でも、**「別のファイルを取得」** ダイアログボックスが再び表示されます。これによって、以前の設定を変更する機会が提供されます。

詳細については、77 ページの **「別のファイルを取得」** ダイアログボックスを参照してください。

複数イメージのロード

通常、デバッグ可能なアプリケーションは、デバッグ対象の 1 ファイルのみで構成されます。ただし、追加のデバッグファイル（イメージ）をロードすることもできます。すなわち、プログラム全体は複数のイメージで構成されることになります。

この機能は、プラットフォーム提供の機能に関する追加ライブラリが含まれるビルド済 ROM イメージと一緒にアプリケーションをデバッグする場合に便利です。ROM イメージとアプリケーションは、別々のプロジェクトを使用して IAR Embedded Workbench IDE でビルドされ、別々の出力ファイルを生成します。

複数のイメージがロードされている場合、ロードされたすべてのイメージを合わせたデバッグ情報が表示されます。[イメージ] ウィンドウでは、1 つのイメージまたは全部のイメージのデバッグ情報を表示するかを選択できます。

C-SPY の起動時に追加のイメージをロードするには、以下の手順に従います。

- 1 [プロジェクト] > [オプション] > [デバッグ] > [イメージ] を選択して、ロードするイメージを最高 3 つまで指定します。詳細については、369 ページのイメージを参照してください。
- 2 デバッグセッションを開始します。

特定の瞬間で追加のイメージをロードするには、以下の手順に従います。

__loadImage システムマクロを使用し、280 ページの C-SPY マクロの使用手順にある方法のいずれかによって実行します。

ロードされたイメージのリストを表示するには、次の手順に従います。

[表示] メニューから [イメージ] を選択します。イメージのウィンドウが表示されます (75 ページの [イメージ] ウィンドウを参照)。

ターゲットハードウェアへの適合

このセクションでは、C-SPY に対してターゲットハードウェアを記述する方法や、アプリケーションがメモリにダウンロードされる前に C-SPY でターゲットハードウェアを初期化する方法について説明します。

具体的には、以下の項目について説明します。

- デバイス記述ファイルの修正
- C-SPY の起動前にターゲットハードウェアを初期化する
- メモリの再配置

デバイス記述ファイルの修正

C-SPY は、製品に付属のデバイス記述ファイルを使用して、さまざまなターゲット固有の調整を行います (59 ページの *デバイス記述ファイルの選択* を参照)。これらには以下のようなデバイス固有の情報が含まれています。

- 周辺ユニットおよびそれらのグループにおけるレジスタの定義
- 割込み定義 (Cortex-M デバイス専用)

通常はデバイス記述ファイルを修正する必要はありません。ただし、なんらかの理由によって事前定義が十分でない場合、ファイルを編集できます。しかし、これらの記述構文の形式は、製品の今後のアップグレード版で更新される可能性がある点に注意してください。

ニーズに最も適したデバイス記述ファイルをコピーし、ファイル内の記述に合わせて修正します。

デバイス記述ファイルの構文については、arm¥doc ディレクトリの『*ARM 用 IAR Embedded Workbench デバイス記述ファイルのフォーマット*』ガイド (EWARM_DDFFormat.pdf) に説明があります。

デバイス記述ファイルのロード方法に関する情報については、59 ページの *デバイス記述ファイルの選択* を参照してください。

C-SPY の起動前にターゲットハードウェアを初期化する

ハードウェアで、コードをダウンロードする前に有効化しなければならない外部メモリを使用する場合、アプリケーションをダウンロードする前にこの処理を実行するためには C-SPY にマクロが必要です。次に例を示します。

- 1 新しいテキストファイルを作成して、マクロ関数を定義します。たとえば、外部の SDRAM を有効にするマクロは以下ようになります：

```
/* 使用するマクロ関数 */
enableExternalSDRAM()
{
    __message "Enabling external SDRAM\n";
    __writeMemory32 ( /* ここにコードを配置 */ );
    /* 必要に応じてここにコードを追加 */
}

/* セットアップマクロが実行時間を決定 */
execUserPreload()
{
    enableExternalSDRAM();
}
```

組込みセットアップマクロ関数の execUserPreload が使用されるため、ターゲットシステムとの通信が確立されてから C-SPY がアプリケーションをダウンロードするまでにマクロ関数が実行されます。

- 2 ファイル名拡張子を mac としてこのファイルを保存します。
- 3 C-SPY を起動する前に、[プロジェクト] > [オプション] > [デバッガ] を選択して、[設定] タブをクリックします。
- 4 オプション [セットアップファイルの使用] を選択して、作成したマクロファイルを選択します。

これで、セットアップマクロが C-SPY の起動シーケンス中にロードされるようになります。

メモリの再配置

ARM を基本とする多くのプロセッサに共通する機能は、メモリの再配置機能です。メモリコントローラでは、リセット後に、フラッシュのような不揮発性メモリにアドレスのゼロをマッピングするのが一般的です。メモリコントローラを構成することで、RAM をアドレスマップのゼロに配置し、不揮発性メモリをアドレスマップの上位に配置するように、システムメモリを再配置することができます。再配置することで、例外テーブルは RAM に置かれ、ターゲットハードウェアにコードをダウンロードしたときに簡単に修正できます。

メモリコントローラを設定してからアプリケーションコードをダウンロードする必要があります。これを実行する最も良い方法は、コードのダウンロードを行う前に C-SPY マクロ関数の `execUserPreload()` を実行することです。マクロ関数 `__writeMemory32()` では、メモリコントローラに必要な初期化を実行します。

以下の例では、Atmel AT91SAM7S256 チップでメモリを再配置するために使用するマクロについて示します。同様なしくみは他の ARM ベンダのプロセッサに存在します。

```
execUserPreload()
{
    // REMAP コマンド
    // 1 を MC_RCR (MC 再配置制御レジスタ) に書き込む
    // 再配置ビットの切替え
    __writeMemory32(0x00000001, 0xFFFFFFFF00, "Memory");
}
```

設定マクロ `execUserReset()` は、C-SPY リセット後にマッピングするメモリの再初期化と同じように定義する必要があることに注意してください。これは、ハードウェアデバッグシステムを設定して C-SPY リセットでハードウェアリセットを行う場合に（たとえば `__hwReset()` を `execUserReset()` マクロに追加）、必要となることがあります。

C-SPY にマクロファイルをインストールする方法についての詳細は、282 ページの **セットアップマクロとセットアップファイルによる登録と実行**を参照してください。使用するマクロ関数の詳細については、292 ページの **C-SPY システムマクロについてのリファレンス情報**を参照してください。

デバッグ起動の概要

起動フローの理解と実施を簡単に行うため、以下の図で、C-SPY およびターゲットハードウェアが実行する処理フローと、事前定義された C-SPY 設定マクロの実行について説明します。1 つめの図がフラッシュ内のデバッグコードであり、もう 1 つの図が RAM 内のデバッグコードです。

C-SPY システムマクロの詳細については、本ガイドの *C-SPY マクロの使用* の章を参照してください。

フラッシュのコードのデバッグ

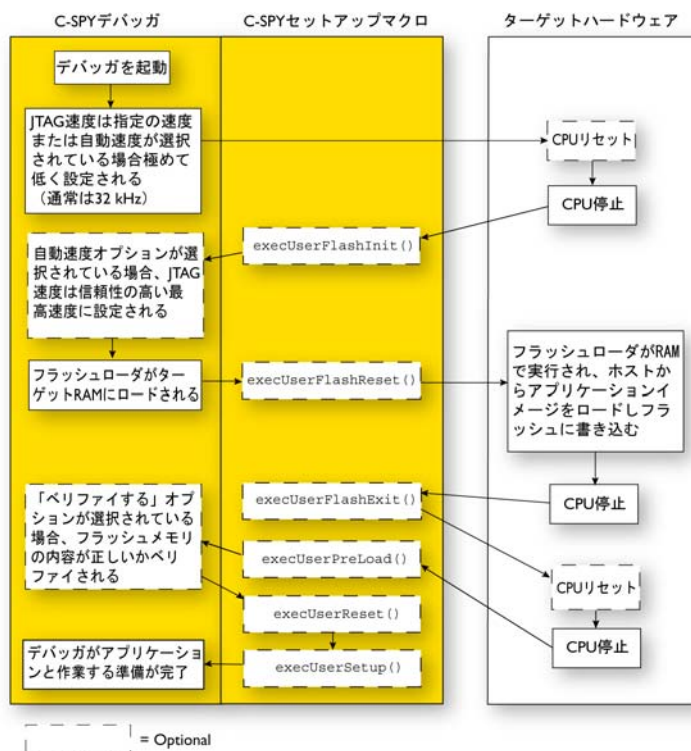


図 11: フラッシュのコードをデバッグする場合のデバッグの起動

RAM のコードのデバッグ

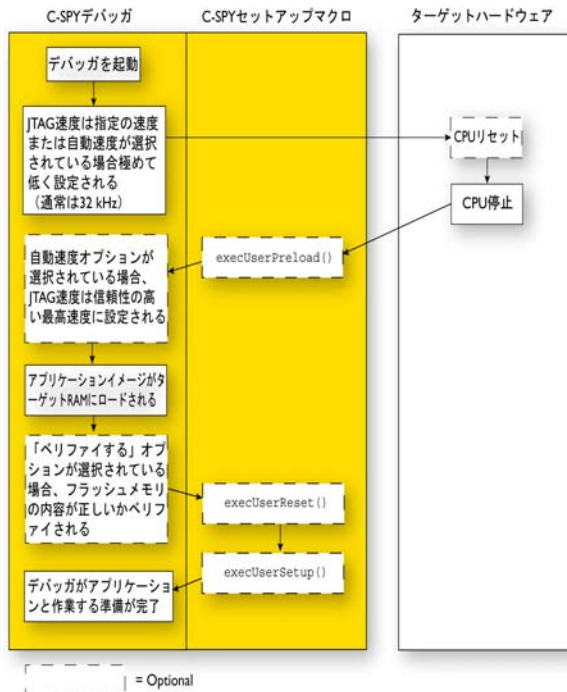


図 12: RAM のコードをデバッグする場合のデバッガの起動

サンプルプロジェクトの実行

IAR Embedded Workbench には、サンプルアプリケーションが付属しています。これらのサンプルを使用して、IAR システムズの開発ツールの基本的な使用方法を理解したり、ターゲットボードで確立した接続を検証することができます。また、これらのサンプルを基にして、アプリケーションプロジェクトを開始することもできます。

これらのサンプルは、arm\examples ディレクトリにあります。サンプルはすぐ使用できるようになっています。既製のワークスペースファイルが、ソースコードファイルおよび関連する他のすべてのファイルとともに提供されます。

サンプルプロジェクトの実行

サンプルプロジェクトを実行するには、次の手順に従います。

- 1 [ヘルプ] > [インフォメーションセンタ] を選択して、[プロジェクト例] をクリックします。
- 2 使用する特定の評価ボードまたはスターターキットに合う例を探します。



図 13: サンプルアプリケーション

[プロジェクトを開く] ボタンをクリックします。

- 3 表示されたダイアログボックスで、プロジェクトの配置先フォルダを選択します。[選択] をクリックして、フォルダの選択を確認します。
- 4 [ワークスペース] ウィンドウに、使用可能なサンプルプロジェクトが表示されます。プロジェクトを 1 つ選択します。アクティブなプロジェクト（太字で強調表示）でない場合は、そのプロジェクトを右クリックして、コンテキストメニューから [アクティブに設定] を選択します。
- 5 プロジェクト設定を表示するには、プロジェクトを選択して、コンテキストメニューから [オプション] を選択します。[一般オプション] > [ターゲット] > [派生プロセッサ] および [デバッガ] > [設定] > [ドライバ] の設定を確認します。プロジェクトのその他の設定については、選択したターゲットシステムに合わせて設定されます。

C-SPY オプションの詳細と、ターゲットボード操作用に C-SPY を設定する方法については、363 ページの **デバッガオプション** を参照してください。

[OK] をクリックして、プロジェクトの **[オプション]** ダイアログボックスを閉じます。



6 アプリケーションをコンパイルしてリンクするには、**[プロジェクト]** > **[作成]** を選択するか、**[作成]** ボタンをクリックします。

7 C-SPY を起動するには、**[プロジェクト]** > **[デバッグ]** を選択するか、**[ダウンロードしてデバッグ]** ボタンをクリックします。



8 **[デバッグ]** > **[移動]** を選択するか、**[移動]** ボタンをクリックしてアプリケーションを起動します。

実行を停止するには、**[停止]** ボタンをクリックします。

C-SPY の起動についてのリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 69 ページの *C-SPY* デバッガメインウィンドウ
- 75 ページの **[イメージ]** ウィンドウ
- 77 ページの **[別のファイルを取得]** ダイアログボックス

以下も参照してください。

- 『*ARM 用 IDE プロジェクト管理およびビルドガイド*』のデバッガのツールオプション

C-SPY デバッガメインウィンドウ

C-SPY デバッガを起動すると、IAR Embedded Workbench IDE のメインウィンドウに、以下のデバッガ専用項目が表示されます。

- アプリケーションの実行、デバッグ用コマンドが含まれる **[デバッグ]** 専用メニュー
- 使用する C-SPY ドライバに応じた、ドライバ固有のメニュー。本書では通常 **ドライバメニュー** と表記します。通常は、このメニューには、ドライバ専用のウィンドウやダイアログボックスを表示するためのメニューコマンドが表示されます。
- デバッグ用ツールバー
- C-SPY 専用のウィンドウ、ダイアログボックス

C-SPY メインウィンドウは、製品インストールのどのコンポーネントを使用するかによって外観が異なる場合があります。

メニューバー

以下のメニューは C-SPY の実行中に使用できます。

デバッグ	ソースアプリケーションを実行およびデバッグするコマンドを提供します (71 ページの <i>[デバッグ]</i> メニューを参照)。ほとんどのコマンドは、デバッグツールバーのアイコンボタンからも実行できます。
逆アセンブリ	逆アセンブリプロセッサモードを制御するコマンドを提供します (73 ページの <i>[逆アセンブリ]</i> メニューを参照)。
シミュレータ	割込みシミュレーションとメモリアクセスチェックングを設定するためのダイアログボックスへのアクセスを提供します。C-SPY シミュレータが使用されている場合にのみ、このメニューは利用可能です (392 ページの <i>シミュレータ</i> のメニューを参照)。
GDB サーバ	C-SPY GDB サーバドライバに固有のコマンドを提供します。このメニューはドライバの使用時にのみ利用可能です (393 ページの <i>[GDB サーバ]</i> メニューを参照)。
J-Link	C-SPY J-Link ドライバに固有のコマンドを提供します。このメニューはドライバの使用時にのみ利用可能です (394 ページの <i>J-Link</i> メニューを参照)。
TI Stellaris FTDI	C-SPY TI Stellaris FTDI ドライバに固有のコマンドを提供します。このメニューはドライバの使用時にのみ利用可能です (397 ページの <i>TI Stellaris FTDI</i> メニューを参照)。
JTAG	C-SPY Macraigor ドライバに固有のコマンドを提供します。このメニューはドライバの使用時にのみ利用可能です (398 ページの <i>Macraigor</i> の <i>[JTAG]</i> メニューを参照)。
RDI	C-SPY RDI ドライバに固有のコマンドを提供します。このメニューはドライバの使用時にのみ利用可能です (399 ページの <i>RDI</i> メニューを参照)。
ST-LINK	C-SPY ST-LINK ドライバに固有のコマンドを提供します。このメニューはドライバの使用時にのみ利用可能です (400 ページの <i>ST-LINK</i> メニューを参照)。









【デバッグ】メニュー

【デバッグ】メニューは、C-SPY の実行中のみ使用できます。【デバッグ】メニューには、ソースアプリケーションの実行 / デバッグ用コマンドが表示されます。ほとんどのコマンドは、デバッグツールバーのアイコンボタンから実行できます。

実行 (G)	F5
ブレーク (B)	
リセット (R)	
デバッグの停止 (S)	Ctrl+Shift+D
ステップオーバー (O)	F10
ステップイン (I)	F11
ステップアウト (U)	Shift+F11
次のステートメント (N)	
カーソル位置まで実行 (C)	
自動ステップ (T)...	
カーソルの位置に PC を設定 (E)	
C++ 例外 (X)	▶
メモリ (M)	▶
更新 (U)	
マクロ (M)...	
ログ (L)	▶

図 14: 【デバッグ】メニュー

以下のコマンドがあります。

	Go F5	現在の文 / 命令から、ブレークポイントかプログラム終了までコードを実行します。
	ブレーク	アプリケーション実行を停止します。
	リセット	ターゲットプロセッサをリセットします。
	デバッグの停止 Ctrl+Shift+D	デバッグセッションを停止し、プロジェクトマネージャに戻ります。
	ステップオーバー F10	C/C++ 関数やアセンブラサブルーチンに入らずに、次の文 / 関数呼出し / 命令を実行します。
	ステップイン F11	C/C++ 関数やアセンブラサブルーチンに入って、次の文 / 命令 / 関数呼出しを実行します。
	ステップアウト Shift+F11	現在の文から、現在の関数の呼出し後の文までを実行します。
	次の実行文	各関数呼び出しを停止せずに次の文を直接実行します。

**カーソルまで実行**

現在の文 / 命令から、選択した文 / 命令までコードを実行します。

自動ステップ

自動ステップをカスタマイズしたり実行できるダイアログボックスを表示します (96 ページの [自動ステップの設定] ダイアログボックスを参照)。

次の実行文の設定

プログラムカウンタをカーソル位置に直接移動します。ソースコードは実行しません。ただし、プログラムフローに異常が生じ、予期しない効果が発生することがあります。

**C++ 例外 >
スロー時にブレイク**

ターゲットアプリケーションが throw 文を実行したときに、実行が停止するように指定します。

この機能を使用するには、オプション [ライブラリ低レベルインタフェースの実装] を選択し、C++ 規格の言語オプションを [C++] にしてアプリケーションをビルドする必要があります。

**C++ 例外 >
例外が取得されなかったときにブレイク**

catch 文を照合して取得されない例外をターゲットアプリケーションがスローしたときに、実行が停止するように指定します。

この機能を使用するには、オプション [ライブラリ低レベルインタフェースの実装] を選択し、C++ 規格の言語オプションを [C++] にしてアプリケーションをビルドする必要があります。

メモリ > 保存

特定のメモリエリアの内容をファイルに保存できるダイアログボックスを表示します (163 ページの [メモリ保存] ダイアログボックスを参照)。

メモリ > 復元

特定のメモリゾーンにファイルの内容を Intel-extended または Motorola s-record フォーマットでロードできるダイアログボックスを表示します (164 ページの [メモリ復元] ダイアログボックスを参照)。

更新

すべてのデバッグウィンドウの内容を更新します。ウィンドウの更新は自動的に行われるため、C-SPY が検出できない方法でターゲットメモリが修正された場合など、通常の状態でない場合にのみ更新が必要です。[逆アセンブリ] ウィンドウに表示されたコードが変更された場合にも有効です。

マクロ	マクロファイルと関数を一覧表示、登録、編集するためのダイアログボックスを表示します (281 ページの [マクロ設定] ダイアログボックスの使用を参照)。
ログ > ログファイルの設定	[デバッグログ] ウィンドウの内容をファイルに記録することもできるダイアログボックスを表示します。ログファイルの種類と場所を選択できます。以下の項目から選択して記録できます。エラー、ワーニング、システム情報、ユーザーメッセージ、またはすべて。95 ページの [ログファイル] ダイアログボックスを参照してください。
ログ > ターミナル I/O ログファイルの設定	シミュレーションされたターゲットアクセス通信をファイルに記録することもできるダイアログボックスを表示します。ログファイルの場所を選択できます。93 ページの [ターミナル I/O ログファイル] ダイアログボックスを参照してください。

[逆アセンブリ] メニュー

[逆アセンブリ] メニューは、C-SPY の実行中のみ使用できます。このメニューには、ソースアプリケーションの実行 / デバッグ用コマンドが表示されます。ほとんどのコマンドは、デバッグツールバーのアイコンボタンから実行できます。

Thumbモードでの逆アセンブル(T)
 ARMモードでの逆アセンブル(A)
 現在のプロセッサモードでの逆アセンブル(C)
 ・自動モードでの逆アセンブル(U)

図 15: [逆アセンブリ] メニュー

メニューのコマンドを使用して、使用する逆アセンブリモードを選択します。

注: 逆アセンブリモードを変更した後は、[デバッグ] メニューの [更新] コマンドを使用して [逆アセンブリ] ウィンドウの表示内容を更新してください。

以下のコマンドがあります。

Thumb モードでの逆アセンブル	アプリケーションを Thumb モードで逆アセンブルします。
ARM モードでの逆アセンブル	アプリケーションを ARM モードで逆アセンブルします。

現在のプロセッサモードでの逆アセンブル	アプリケーションを現在のプロセッサモードで逆アセンブルします。
自動モードでの逆アセンブル	アプリケーションを自動モードで逆アセンブルします。デフォルトのオプションです。

86 ページの [逆アセンブリ] ウィンドウも参照してください。

C-SPY のウィンドウ

使用する C-SPY ドライバに応じて、C-SPY の実行中に C-SPY に固有の以下のウィンドウが使用できます。

- C-SPY デバッガメインウィンドウ
- [逆アセンブリ] ウィンドウ
- [メモリ] ウィンドウ
- [シンボルメモリ] ウィンドウ
- [レジスタ] ウィンドウ
- [ウォッチ] ウィンドウ
- [ローカル] ウィンドウ
- [自動] ウィンドウ
- [ライブウォッチ] ウィンドウ
- [クイック ウォッチ] ウィンドウ
- [静的] ウィンドウ
- [呼出しスタック] ウィンドウ
- [トレース] ウィンドウ
- [関数トレース] ウィンドウ
- [タイムライン] ウィンドウ
- [ターミナル I/O] ウィンドウ
- [コードカバレッジ] ウィンドウ
- [関数プロファイラ] ウィンドウ
- [イメージ] ウィンドウ
- [スタック] ウィンドウ
- [シンボル] ウィンドウ

使用する C-SPY ドライバに応じて、他のウィンドウも使用可能です。

C-SPY ウィンドウでの編集

[メモリ]、[シンボルメモリ]、[レジスタ]、[自動]、[ウォッチ]、[ローカル]、[静的]、[ライブウォッチ]、[クイックウォッチ] の各ウィンドウの内容を編集できます。

これらのウィンドウ内容の編集には、以下のキー操作を使用します。

Enter 項目を編集可能にします。また、新しい値を保存します。

Esc 新しい値を取り消します。

[式] フィールドが編集可能なウィンドウでは、整数の後にセミコロンを追加すると表示されるエレメントの数を指定できます。たとえば、myArray という配列の最初の 3 つのエレメントのみ、またはポインタによって示されるエレメントから続いて 3 つのエレメントを表示するには、次のように記述します。

```
myArray;3
```

または、最初のエレメントを指定するコンマと整数を 1 つずつ追加します。たとえば、10 から 14 のエレメントを表示するには、次のように記述します。

```
myArray;5,10
```

[イメージ] ウィンドウ

[イメージ] ウィンドウは [表示] メニューから利用できます。



図 16: [イメージ] ウィンドウ

[イメージ] ウィンドウには、現在ロードされたすべてのイメージ（デバッグファイル）が一覧表示されます。

通常、ソースアプリケーションは、デバッグ対象の 1 つのイメージのみで構成されます。ただし、追加のイメージをロードすることもできます。すなわち、デバッグ対象のユニット全体は複数のイメージで構成されることになります。

表示エリア

このエリアには、以下の列にロードされたイメージが一覧表示されます。

名称	ロードされているイメージ名。
パス	ロードされているイメージのパス。

C-SPY はロードされているすべてのイメージからのデバッグ情報を同時に使用するか、または 1 回ごとに 1 つのイメージから使用できます。そのイメージだけの情報を表示するには、行をダブルクリックします。現在の選択内容が強調表示されます。

コンテキストメニュー

以下のコンテキストメニューがあります。

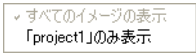


図 17: [イメージ] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

すべてのイメージの表示	ロードされたすべてのデバッグイメージのデバッグ情報を表示します。
イメージのみを表示	選択されたデバッグイメージのデバッグ情報を表示します。

関連情報

関連情報については、以下を参照してください。

- 62 ページの *複数イメージのロード*
- 369 ページの *イメージ*
- 308 ページの *__loadImage*

【別のファイルを取得】ダイアログボックス

C-SPY がロードするソースファイルを自動的に見つけられない場合（アプリケーションが別のコンピュータでビルドされた場合など）、**【別のファイルを取得】** ダイアログボックスが表示されます。

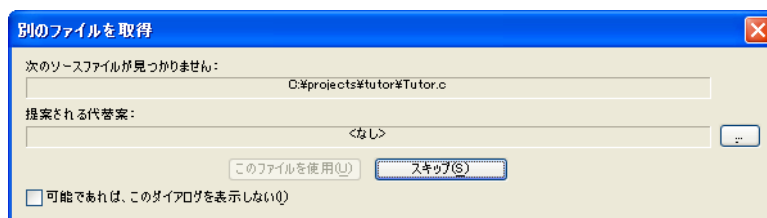


図 18: **【別のファイルを取得】** ダイアログボックス

次のソースファイルが見つかりません

見つからないソースファイル。

提案される代替案

代替りのファイルを指定します。

このファイルを使用

代替ファイルを指定した後に、**【このファイルを使用】** によって、そのファイルを要求したファイルのエイリアスとして確定します。このアクションを選択した後は、これらのファイルが最初に選択された代替ファイルと似たようなディレクトリ構造にある場合、C-SPY は自動的に他のソースファイルを検索します。

次にデバッグセッションを開始するときに、選択した代替ファイルが自動的にあらかじめロードされます。

スキップ

このデバッグセッションについて、C-SPY はソースファイルが入手できないと想定します。

可能であれば、このダイアログを表示しない

不足しているソースファイルについてダイアログボックスを表示する代わりに、C-SPY は以前の応答を使用します。

関連情報

関連情報については、61 ページの **ソースファイルがない状態でデバッグセッションを開始する**を参照してください。

アプリケーションの実行

この章には、C-SPY® でのアプリケーション実行に関する情報が記載されています。具体的には以下の項目を解説します。

- アプリケーション実行の概要
- アプリケーション実行についてのリファレンス情報

アプリケーション実行の概要

このセクションでは、以下のトピックについて説明します。

- アプリケーション実行の概要について
- ソースモードと逆アセンブリモードのデバッグ
- ステップ実行
- アプリケーションの実行
- 強調表示
- 呼出しスタック情報
- ターミナル I/O
- デバッグログ

アプリケーション実行の概要について

C-SPY では、アプリケーションの実行をモニタおよび制御することができます。シングルステップでアプリケーションを実行し、ブレークポイントを設定することにより、変数やレジスタの値など、アプリケーション実行の詳細を調べることができます。また、呼出しスタックを使用して、関数の呼出しチェーン内でステップを前後に実行することもできます。

ターミナル I/O およびデバッグログ機能を使用すると、アプリケーションを操作できます。

そのようなコマンドは、[デバッグ] メニューとツールバーにあります。

ソースモードと逆アセンブリモードのデバッグ

C-SPY では、必要に応じてソースモードと逆アセンブリモードを切り替えてデバッグできます。

ソースモードのデバッグは、アプリケーションを短期間で簡単に開発するための手段であり、コンパイラやアセンブラがどのようにコードを実装したかを知る必要はありません。エディタウィンドウでは、一度に 1 文ずつアプリケーションを実行しながら、変数やデータ構造体の値をモニタできます。

逆アセンブリモードのデバッグでは、アプリケーションの重要なセクションに焦点を当てて、アプリケーションコードを高い精度で制御することができます。逆アセンブリウィンドウを開くと、アプリケーションが、ソースコードではなく実際のメモリの内容に基づいて、ニーモニックアセンブリリストとして表示され、一度にひとつのマシン命令をアプリケーションで実行できます。

どちらのモードでデバッグしている場合でも、レジスタとメモリを表示したり、その内容を書き換えたりすることができます。

ステップ実行

C-SPY は文単位でステップを実行できるため、行単位でステップを実行する他の多くのデバッガよりも高い精度でステップ実行できます。コンパイラは、文や関数呼出しごとにステップポイントという形式で詳細なステップ実行情報を生成します。すなわち、コマンドでステップインするか、ステップオーバーするかを選択する可能性のあるソースコード上の位置に、ステップポイントを生成します。ステップポイントは文だけでなく、関数呼出しの位置にも生成されるので、単純に文をステップ実行するよりも詳細にステップ実行できます。ステップコマンドには、以下の 4 つのコマンドがあります。

- ステップイン
- ステップオーバー
- 次の実行文
- ステップアウト

[自動ステップの設定] ダイアログボックスを使用して、シングルステップの実行を自動化できます。詳細については、96 ページの *[自動ステップの設定]* ダイアログボックスを参照してください。

コード外部で検出される例外（通常はステップの一部として実行されます）がアプリケーションに含まれる場合、C-SPY は catch 文の位置でステップを終了します。

以下に示す例で、1 つ前のステップ実行の結果、現在 `f(i)` 関数呼出し（強調表示）の位置にいるものとします。

```
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 ++;
}
```



ステップイン

ステップ実行中は、通常は関数にステップインして関数かサブルーチンの内部でステップ実行を継続することを考えます。[ステップイン] コマンドを実行すると、サブルーチン `g(n-1)` 内部の最初のステップポイントに移動します。

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

[ステップイン] コマンドは、通常の制御の流れに従い、次のステップポイントが同じであるか別の関数であるかどうかに関わらず、次のステップポイントまで実行します。



ステップオーバ

[ステップオーバ] コマンドは、同じ関数の次のステップポイントまで実行します。呼び出された関数の内部にステップインすることはありません。上の例でステップオーバを実行すると、`g(n-2)` 関数呼出しまで実行します。この関数呼出しは独立した文ではなく、`g(n-1)` と同じ文にあります。このようにして、文の一部に興味のない関数呼出しがある場合にそこをスキップして、重要な部分だけをデバッグすることができます。

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



次の実行文

[次の実行文] コマンドは、この場合の次の文である `return value` まで一気に実行します。これによって、高速にステップを進めることができます。

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



ステップアウト

関数内部では、必要に応じて、[ステップアウト] コマンドを実行すると、関数の最後に到達する前に、ステップアウトすることができます。これによって、関数呼出しの直後の文まで一気に実行します。

```
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 ++;
}
```

複雑な文の一部である個別の関数にステップインできる機能は、複数回ネストしている関数呼出しを含む C ソースコードを使用している場合に、特に役に立ちます。また、コンストラクタ、デストラクタ、代入演算子、その他のユーザ定義演算子など、多くの間接的な関数呼出しを使用する傾向がある C++ でも、ステップイン機能が非常に役に立ちます。

細かいステップ実行は、状況によっては役立つ場合もありますが、不必要に処理を遅くすることもあります。そのため、文単位でステップ実行する機能があり、これによって高速なステップ実行が可能になります。

アプリケーションの実行



Go

[Go] コマンドは、現在の位置からブレークポイントまたはプログラムの最後で到達するまで、続けて実行します。



カーソルまで実行

[カーソルまで実行] コマンドは、ソースコードの現在カーソルのある位置まで実行します。[カーソルまで実行] コマンドは、[逆アセンブリ] ウィンドウと[呼出しスタック] ウィンドウでも使用できます。

強調表示

エディタウィンドウと[逆アセンブリ] ウィンドウの両方で実行を停止するたびに、C-SPY は対応する C/C++ ソースまたは命令を緑色で強調表示します。さらに緑色の矢印が、C/C++ ソースレベルでステップする場合はエディタウィンドウに、逆アセンブリレベルでステップする場合は[逆アセンブリ] ウィンドウに表示されます。これは、どのウィンドウがアクティブであるかによって決まります。ウィンドウがどれもアクティブでない場合、最後にアクティブだったウィンドウによって決まります。

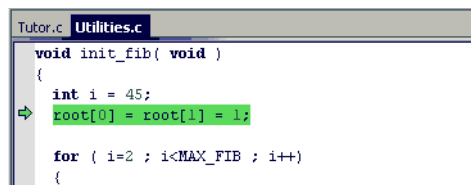


図 19: C-SPY で強調表示されるソース位置

関数呼出しのない単純な文の場合、通常は文全体が強調表示されます。複数の関数呼出しを含む文で停止した場合、[ステップイン] と [ステップオーバー] の違いを明確に表すために、最初の関数呼出しが強調表示されます。

ときどき、ソースウィンドウの文が通常の強調表示よりも薄い色で強調表示される場合があります。これは、プログラムカウンタはソース文を構成するアセンブラ命令を指している、そこが正確なステップポイントではないことを意味します。これは、[逆アセンブリ] ウィンドウでステップ実行しているときによく発生します。プログラムカウンタがソース文の最初の命令を指している場合は、通常の強調表示の色が使用されます。

呼出しスタック情報

コンパイラは、大量のバックトレース情報を生成します。これにより、C-SPY は、実行に影響を及ぼすことなく、いつでも関数呼出しチェーン全体を表示できます。



この機能は通常、以下の 2 つの目的で使用します。

- 現在の関数の呼出し元のコンテキストを決定する場合
- 不正な変数やパラメータの値を検出したときに、その発生元をトレースして、呼出しチェーン内で問題が発生した関数を特定する場合

[呼出しスタック] ウィンドウには関数呼出しのリストが表示され、現在の関数が一番上になります。呼出しチェーンの関数を調べるときに、影響を受けたすべてのウィンドウの内容が更新され、特定の呼出しフレームの状態が表示されます。影響を受けるウィンドウには、エディタ、[ローカル]、[レジスタ]、[ウォッチ]、[逆アセンブリ] の各ウィンドウがあります。通常、関数がすべてのレジスタを使用することはないため、一部のレジスタは状態が未定義であり、そのときはダッシュ記号 (---) で表示されます。

エディタウィンドウと [逆アセンブリ] ウィンドウでは、最上位または現在の呼出しフレームは緑色で強調表示され、他のフレームを検証しているときは黄色で強調表示されます。

呼出しスタックで関数を選択し、[カーソル位置まで実行] コマンドをクリックしてその関数を実行すれば便利です。

アセンブラソースコードには、自動的にバックトレース情報が挿入されることはありません。適切な CFI アセンブラディレクティブをアセンブラソースコードに追加すると、アセンブラモジュールの呼出しチェーンも表示できるようになります。詳細については、『ARM 用 IAR アセンブラリファレンスガイド』を参照してください。

ターミナル I/O

場合によっては、stdin や stdout を使用するアプリケーションの構文を、実際のハードウェアデバイスを入出力として使用しないでデバッグする必要があります。[ターミナル I/O] ウィンドウでは、アプリケーションへ入力したり、アプリケーションからの出力を表示したりできます。また、[ターミナル I/O ログファイル] ダイアログボックスを使用して、ターミナル I/O をファイルに出力することもできます。



この機能は、以下の2つのコンテキストで使用します。

- アプリケーションが `stdin` と `stdout` を使用している場合
- デバッグトレース出力を生成する場合

詳細については、92 ページの [ターミナルI/O] ウィンドウ、93 ページの [ターミナルI/O ログファイル] ダイアログボックスを参照してください。

デバッグログ

[デバッグログ] ウィンドウには、診断メッセージやマクロにより生成された出力、イベントログメッセージ、トレースについての情報など、デバッグの出力が表示されます。



これらの情報は、簡単に検証できるファイルにログとして記録しておくことが便利です。デバッグの出力をファイルに記録することで、主に2つのメリットがあります。

- ファイルをエディタなどの別のツールで開くことができるので、ファイルの中で特に興味のある部分だけ調べることが可能
- ファイルには、どこでブレークポイントがトリガされたかなど、プログラム実行がどのように制御されたかの履歴が残る

アプリケーション実行についてのリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 86 ページの [逆アセンブリ] ウィンドウ
- 90 ページの [呼出しスタック] ウィンドウ
- 92 ページの [ターミナルI/O] ウィンドウ
- 93 ページの [ターミナルI/O ログファイル] ダイアログボックス
- 94 ページの [デバッグログ] ウィンドウ
- 95 ページの [ログファイル] ダイアログボックス
- 96 ページの [自動ステップの設定] ダイアログボックス

『ARM 用 IDE プロジェクト管理およびビルドガイド』のターミナル I/O オプションも参照してください。

【逆アセンブリ】ウィンドウ

【C-SPY 逆アセンブリ】ウィンドウは【表示】メニューから利用できます。

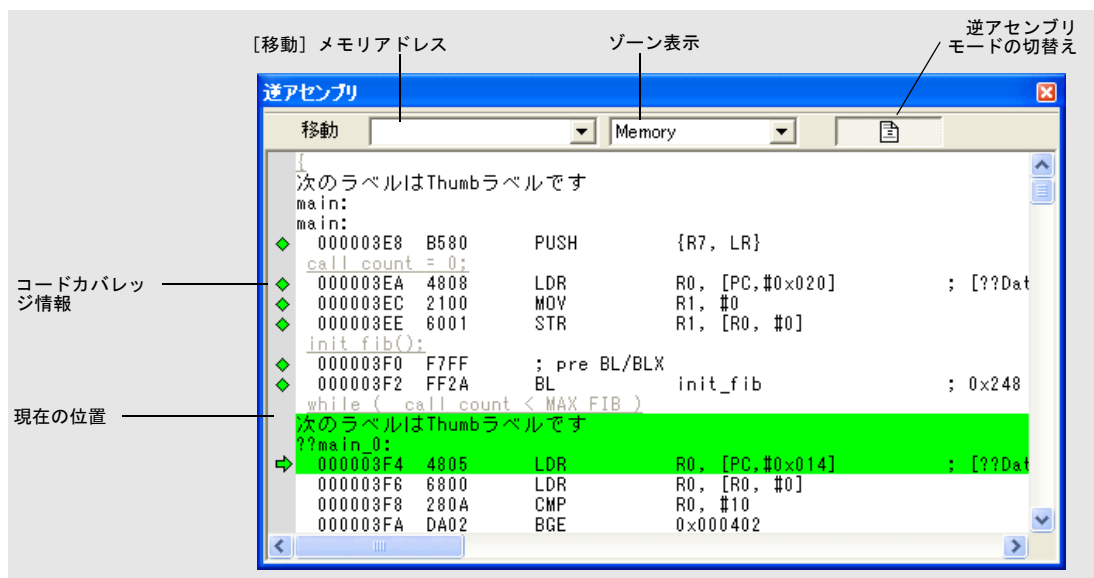


図20: C-SPY の【逆アセンブリ】ウィンドウ

このウィンドウには、デバッグ中のアプリケーションが、逆アセンブリされたアプリケーションコードとして表示されます。

【逆アセンブリ】ウィンドウでソースコードのデフォルト色を変更するには、次の手順に従います。

- 1 【ツール】 > 【オプション】 > 【デバッグ】 を選択します。
- 2 デフォルトの色を設定するには、[[逆アセンブリ] ウィンドウのソースコード色] オプションを使用します。



関数に対応するアセンブラコードを表示するには、エディタウィンドウでその関数を選択し、【逆アセンブリ】ウィンドウにドラッグします。

ツールバー

ツールバーの内容は以下のとおりです。

移動	表示するロケーションを指定できます。それには、メモリアドレス、または変数、機能、ラベルの名前を指定できます。
ゾーン表示	表示可能なメモリゾーンのリストを示します (156 ページの <i>C-SPY</i> メモリゾーンを参照)。
混在モードのトグル	逆アセンブリしたコードだけを表示するか、対応するソースコードも併せて表示するかを選択します。ソースコードを表示するには、デバッグ情報付きでソースファイルをコンパイルしておく必要があります。

表示エリア

表示エリアには、逆アセンブリしたアプリケーションコードが表示されます。このエリアには以下のグラフィック要素が含まれます。

緑の強調表示	現在の位置は、次に実行されるアセンブラ命令を示します。[逆アセンブリ] ウィンドウの任意の行をクリックすると、その行にカーソルが移動します。また、移動キーを使用してカーソルを移動することもできます。
黄色の強調表示	[呼出しスタック] ウィンドウのフレーム間を移動したり、[トレース] ウィンドウでアイテム間を移動するときなど、現在の位置以外の位置を示します。
赤の点	ブレークポイントを示します。ブレークポイントを設定するには、ウィンドウの左側の灰色で表示された余白部分をダブルクリックします。詳細については、119 ページの <i>ブレークポイントの使用</i> を参照してください。
緑色のひし形	実行済みのコードを示します。すなわち、コードカバレッジです。

命令プロファイリングが有効化（コンテキストメニューで設定）されている場合、それぞれの命令が実行された回数に関する情報を表示する列が左側の余白部に追加されます。

コンテキストメニュー

以下のコンテキストメニューがあります。

PCへ移動(M) カーソル位置まで実行(C)	
コードカバレッジ(Q)	▶
命令プロファイリング (P)	▶
ブレークポイントの切り替え(A) (コード)	
ブレークポイントの切り替え(A) (ログ)	
ブレークポイントの切り替え(A) (トレース開始)	
ブレークポイントの切り替え(A) (トレース停止)	
ブレークポイントの有効化/無効化(U)	
次の文の設定	
ウインドウ内容のコピー	
▼ 混在モード(O)	

図 21: [逆アセンブリ] ウィンドウのコンテキストメニュー

注：このメニューの内容は動的です。つまり、製品パッケージによって外観が変わることがあります。

以下のコマンドがあります。

PC へ移動	現在のプログラムカウンタ位置のコードを表示します。
カーソルまで実行	現在の位置からカーソルを含む行まで、アプリケーションを実行します。
コードカバレッジ	コードカバレッジ制御用コマンドのサブメニューを表示します。このコマンドは、使用しているドライバがサポートする場合のみ使用できます。 [有効化] は、コードカバレッジの有効 / 無効を切り替えます。 [表示] は、コードカバレッジの有効 / 無効の表示を切り替えます。実行されるコードは、緑色のひし形で示されます。 [クリア] は、すべてのコードカバレッジ情報を消去します。

命令プロファイリング	<p>命令プロファイリング制御用コマンドのサブメニューを表示します。このコマンドは、使用しているドライバがサポートする場合のみ使用できます。</p> <p>【有効化】は、命令プロファイリングの有効 / 無効を切り替えます。</p> <p>【表示】は、命令プロファイリングの有効 / 無効の表示を切り替えます。それぞれの命令ごとに、命令の実行回数が左側の余白部に表示されます。</p> <p>【クリア】は、すべての命令プロファイリング情報を消去します。</p>
ブレークポイントの切替え (コード)	<p>コードブレークポイントを設定 / 解除します。コードブレークポイントが設定されたアセンブラ命令および対応するすべてのラベルは、赤色で強調表示されます。詳細については、136 ページの [コード] ブレークポイントダイアログボックスを参照してください。</p>
ブレークポイントの切替え (ログ)	<p>ログでトレースを出力するためのブレークポイントを設定 / 解除します。ログブレークポイントが設定されたアセンブラ命令は赤色で強調表示されます。詳細については、141 ページの [ログ] ブレークポイントダイアログボックスを参照してください。</p>
ブレークポイントの切替え (トレース開始)	<p>トレース開始ブレークポイントを切替えます。ブレークポイントがトリガされると、トレースデータの収集が始まります。このメニューコマンドは、使用している C-SPY ドライバでトレースがサポートされている場合にのみ使用できます。詳細については、207 ページの [トレース開始ブレークポイント] ダイアログボックス (シミュレータ) を参照してください。</p>
ブレークポイントの切替え (トレース停止)	<p>トレース停止ブレークポイントを切替えます。ブレークポイントがトリガされると、トレースデータの収集が終了します。このメニューコマンドは、使用している C-SPY ドライバでトレースがサポートされている場合にのみ使用できます。詳細については、208 ページの [トレース停止ブレークポイント] ダイアログボックス (シミュレータ) を参照してください。</p>
有効化 / 無効ブレークポイント	<p>ブレークポイントを有効 / 無効にします。特定の行に複数のブレークポイントがある場合、【有効 / 無効】コマンドによってすべてのブレークポイントが影響を受けます。</p>

ブレークポイントの編集	[ブレークポイントの編集] ダイアログボックスを表示します。このダイアログボックスを使用して、現在選択しているブレークポイントの編集ができます。複数のブレークポイントが選択行にある場合、使用可能なすべてのブレークポイントの一覧を示すサブメニューがその行に表示されます。
次の実行文の設定	プログラムカウンタを挿入ポイントの命令のアドレスに設定します。
ウィンドウの内容のコピー	[逆アセンブリ] ウィンドウで選択した内容をクリップボードにコピーします。
混在モード	逆アセンブリしたコードだけを表示するか、対応するソースコードも併せて表示するかを選択します。ソースコードを表示するには、デバッグ情報付きでソースファイルをコンパイルしておく必要があります。

〔呼出しスタック〕ウィンドウ

〔呼出しスタック〕ウィンドウは〔表示〕メニューから利用できます。

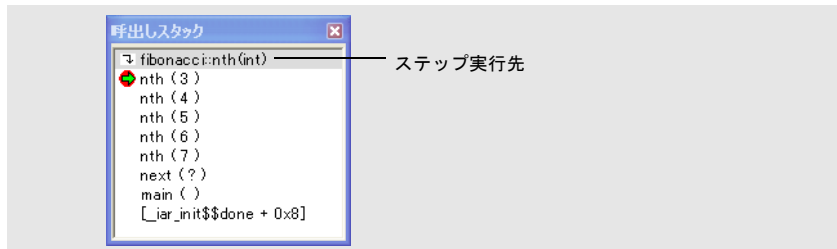


図22: 〔呼出しスタック〕ウィンドウ

C 関数呼出しスタックが、現在の関数とともに上部に表示されます。関数呼出しを調べるには、ダブルクリックします。C-SPY でその呼出しフレームが代わりに表示されます。

次の〔ステップイン〕コマンドが関数呼出しに移動してステップ実行する場合は、ウィンドウ上部の灰色部分に関数名が表示されます。これは、C++ のコンストラクタ、デストラクタ、演算子のような暗黙的な関数呼出しの場合に特に便利です。

表示エリア

コマンド **【引数の表示】** が有効な場合、表示エリアにあるそれぞれのエンタリは次のフォーマットになります。

```
function(values)
```

ここで、*(values)* はパラメータの現在値のリストを示します。関数がパラメータを取らない場合は、空白になります。

コンテキストメニュー

以下のコンテキストメニューがあります。

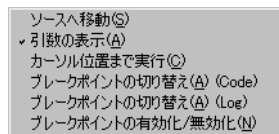


図 23: **【呼出しスタック】** ウィンドウのコンテキストメニュー

以下のコマンドがあります。

ソースへ移動	選択した関数を 【逆アセンブリ】 ウィンドウかエディタウィンドウで表示します。
引数の表示	関数の引数を表示します。
カーソルまで実行	呼出しスタックで選択した関数に戻るまで実行します。
ブレークポイントの切替え (コード)	コードブレークポイントを設定 / 解除します。
ブレークポイントの切替え (ログ)	ログブレークポイントを設定 / 解除します。
有効化 / 無効ブレークポイント	選択したブレークポイントを有効 / 無効にします。

【ターミナル I/O】 ウィンドウ

【表示】 メニューを使用して、【ターミナル I/O】 ウィンドウを開きます。

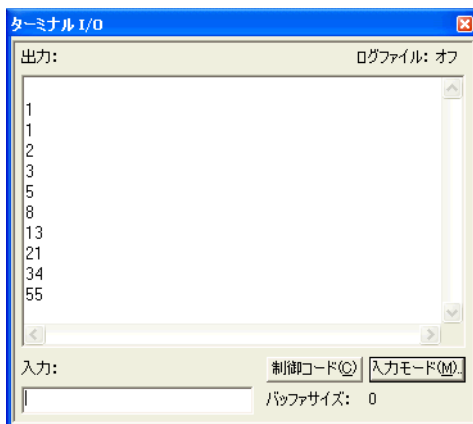


図 24: 【ターミナル I/O】 ウィンドウ

このウィンドウを使用してアプリケーションにデータを入力し、その出力を表示します。

このウィンドウを使用するには、次のことを行う必要があります。

- 1 オプション【セミホスティング】または【IAR ブレークポイント】オプションのどちらかを使用してアプリケーションをビルドします。

それによって、C-SPY は `stdin`、`stdout`、`stderr` をこのウィンドウに転送します。【ターミナル I/O】ウィンドウが表示されていない場合は、入力が必要な場合に自動的に表示されます。出力の場合は自動表示されません。

入力

アプリケーションに入力するテキストを入力します。

制御コード

EOF（ファイル終端）や NUL などの特殊文字の入力メニューを開きます。

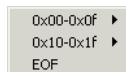


図 25: 【制御コード】メニュー

入力モード

【入力モード】 ダイアログボックスを開きます。ここではキーボードとファイルのどちらからデータを入力するかを選択します。

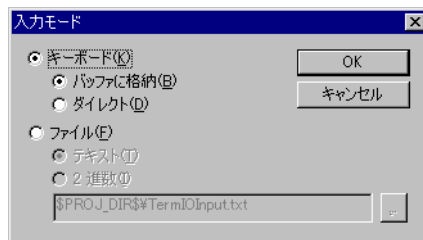


図26: 【入力モード】ダイアログボックス

このダイアログボックスで使用可能なオプションのリファレンス情報については、『*ARM 用 IDE プロジェクト管理およびビルドガイド*』の「ターミナル I/O オプション」の項を参照してください。

【ターミナル I/O ログファイル】ダイアログボックス

【ターミナル I/O ログファイル】ダイアログボックスは、[デバッグ] > [ログ] > [ターミナル I/O ログファイルの設定] を選択して使用します。

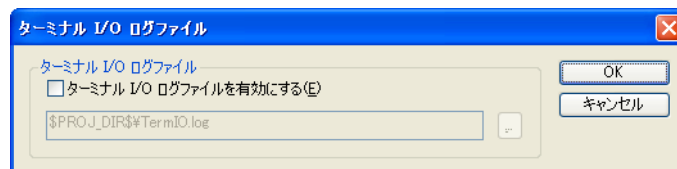


図27: 【ターミナル I/O ログファイル】ダイアログボックス

このダイアログボックスを使用して、C-SPY からターミナル I/O の記録先ログファイルを選択します。

ターミナル I/O ログファイル

ターミナル I/O のログを制御します。ターミナル I/O のファイルへのログを有効にするには、[ターミナル I/O ログファイルの有効化] を選択してファイル名を指定します。デフォルトのファイル名拡張子は log です。参照ボタンを使用して選択することもできます。

【デバッグログ】 ウィンドウ

【デバッグログ】 ウィンドウは、**【表示】 > 【メッセージ】** を選択すれば使用できます。



図28: 【デバッグログ】 ウィンドウ (メッセージウィンドウ)

このウィンドウには、診断メッセージやマクロにより生成された出力、イベントログメッセージ、トレースについての情報など、デバッガの出力が表示されます。この出力は、**C-SPY** の実行中のみ使用できます。デフォルトでは、このウィンドウは他のメッセージウィンドウとグループ化されて表示されず (*ARM 用 IDE プロジェクト管理およびビルドガイド*を参照)。

以下のフォーマットのいずれかの行をダブルクリックすると、対応するソースコードが **【エディタ】** ウィンドウに表示されます。

```
<path> (<row>):<message>
<path> (<row>,<column>):<message>
```

コンテキストメニュー

以下のコンテキストメニューがあります。

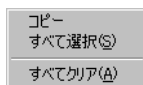


図29: 【デバッグログ】 ウィンドウのコンテキストメニュー

以下のコマンドがあります。

コピー	ウィンドウの内容をコピーします。
すべて選択	ウィンドウの内容を選択します。
すべてをクリア	ウィンドウの内容を消去します。

「ログファイル」ダイアログボックス

「ログファイル」ダイアログボックスは、「デバッグ」>「ログ」>「ログファイルの設定」を選択して使用します。



図 30: 「ログファイル」ダイアログボックス

このダイアログボックスを使用して、C-SPY からの出力をファイルに記録します。

ログの有効化

ファイルへのログを有効 / 無効にします。

含める内容

ファイルに出力される情報は、デフォルトでは「ログ」ウィンドウに表示される内容と同一です。ログに記録する情報を変更するには、以下から選択します。

エラー	C-SPY が処理の実行に失敗したことを示します。
ワーニング	問題となるエラーや漏れ。
情報	C-SPY が実行したアクションの進行状況を示します。
ユーザ	C-SPY マクロからのメッセージ。つまり、__message 文を使用した自分のメッセージ。

参照ボタンを使用して、デフォルトファイルとログファイルの場所をオーバーライドします（デフォルトのファイル名拡張子は log）です。

〔自動ステップの設定〕 ダイアログボックス

〔自動ステップの設定〕 ダイアログボックスは、〔デバッグ〕メニューから表示します。

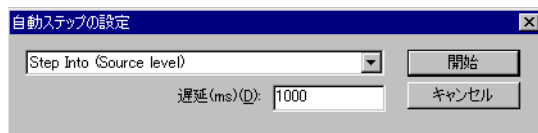


図31: 〔自動ステップの設定〕 ダイアログボックス

このダイアログボックスを使用して、自動ステップをカスタマイズします。
ドロップダウンメニューに、使用可能なステップコマンドが表示されます。

遅延

各ステップ間の遅延をミリ秒単位で指定します。

変数と式の扱い

この章では、変数や式を C-SPY® でどのように使用するかについて説明します。具体的には以下の項目を解説します。

- 変数と式の扱いの概要
- 変数と式の扱いの手順について
- 変数と式の扱いについてのリファレンス情報

変数と式の扱いの概要

このセクションでは、以下のトピックについて説明します。

- 変数と式の扱いの概要について
- C-SPY 式
- 変数情報の制限
- アセンブラ変数の表示

変数と式の扱いの概要について

変数の値を参照、計算する方法はいろいろあります。

- ツールチップウォッチは、エディタウィンドウで表示され、変数や、より複雑な式の値を表示するための最も簡単な方法です。マウスポインタで変数を指すだけで、変数の横にその値が表示されます。
- [自動] ウィンドウには、現在の文やその近くにある文の変数や式が自動的に表示されます。実行が停止すると、ウィンドウは自動的に更新されます。
- [ローカル] ウィンドウには、ローカル変数、つまりアクティブな関数の自動変数と関数パラメータが表示されます。実行が停止すると、ウィンドウは自動的に更新されます。
- [ウォッチ] ウィンドウを使用すると、C-SPY 式および変数の値をモニターできます。実行が停止すると、ウィンドウは自動的に更新されます。
- [ライブウォッチ] ウィンドウは繰り返しサンプリングを行い、アプリケーションの実行中に式の値を表示します。式の変数は、グローバル変数のように、静的に特定する必要があります。
- [統計] ウィンドウには、静的記憶寿命を持つ変数の値が表示されます。実行が停止すると、ウィンドウは自動的に更新されます。

- [クイックウォッチ] ウィンドウでは、式を評価するタイミングを正確に制御することができます。
- [シンボル] ウィンドウには、ランタイムライブラリのシンボルを含め、静的な位置を持つすべてのシンボル（すなわち、C/C++ 関数、アセンブララベル、静的記憶寿命変数）が表示されます。
- [データログ] ウィンドウと [データログ概要] ウィンドウには、最大4つの異なるメモリ位置、またはデータログブレイクポイントを設定して選択するエリアへのアクセスのログが表示されます。データロギングを使用すると、頻繁にアクセスするデータを容易に検索できます。データを特定したら、より効率的なメモリに配置するかどうかを検討できます。データログは、C-SPY J-Link/J-Trace ドライバと C-SPY ST-LINK ドライバでサポートされています。
- トレース関連のウィンドウでは、プログラムの流れを特定の状態まで調べることができます。詳細については、177 ページの *トレースデータの収集と使用* を参照してください。

C-SPY 式

C-SPY 式には、関数呼出しを除く任意の種類の C 言語の式を使用できます。また、以下に示すシンボルも使用できます。

- C/C++ シンボル
- アセンブラシンボル（レジスタ名、アセンブララベル）
- C-SPY マクロ関数
- C-SPY マクロ変数

これらのシンボルから構成された式を C-SPY 式と呼び、さまざまな方法で C-SPY 式をモニタできます。有効な C-SPY 式の例を以下に示します。

```
i + j
i = 42
#asm_label
#R2
#PC
my_macro_func(19)
```

C/C++ シンボル

C シンボルは、アプリケーションの C ソースコードで定義したシンボルです。たとえば、変数や定数、関数（関数はシンボルとして使用できますが、実行はできません）。C 言語のシンボルは、その名前でも参照できます。C++ シンボルに、C-SPY シンボルや式で有効でない関数呼出しが暗黙的に含まれる場合があります。

アセンブラシンボル

アセンブラシンボルは、アセンブララベルかレジスタ名です。つまり、R4-R15 のような汎用レジスタと、プログラムカウンタやステータスレジスタなどの特殊機能レジスタがあります。デバイス記述ファイルを使用する場合は、I/O ポートなど、メモリにマッピングされたすべての周辺ユニットも、CPU レジスタと同様にアセンブラシンボルとして使用できます。63 ページの *デバイス記述ファイルの修正* を参照してください。

アセンブラシンボルの頭に # が付いている場合は C-SPY 式で使用できます。

例	実行される内容
#PC++	プログラムカウンタ値をインクリメント
myptr = #label7	myptr にそのゾーン内の label7 のアドレスを設定

表 5: C-SPY アセンブラシンボル式

ハードウェアレジスタとアセンブララベルの名前が衝突する場合、ハードウェアレジスタが優先的に選択されます。そのような場合にアセンブララベルを参照するには、ラベルをバッククォート（ASCII 文字 0x60）で囲む必要があります。以下に例を示します。

例	実行される内容
#PC	プログラムカウンタを参照
#`PC`	アセンブララベル PC を参照

表 6: ハードウェアレジスタとアセンブララベルの名前が衝突する場合の処理

[レジスタ] ウィンドウには、デフォルトで使用するプロセッサ固有のシンボルが、CPU レジスタレジスタグループに基づいて表示されます。171 ページの [レジスタ] ウィンドウを参照してください。

C-SPY マクロ関数

マクロ関数は、C-SPY マクロ変数定義と、マクロが呼び出されたときに実行されるマクロ文から構成されます。

C-SPY マクロ関数とその使用方法の詳細については、279 ページの *マクロ言語の概要* を参照してください。

C-SPY マクロ変数

マクロ変数の定義と配置はアプリケーションの外部で行われ、C-SPY 式で使用できます。C 言語のシンボルと C-SPY マクロ変数の名前が衝突する場合、C-SPY マクロ変数が C 言語の変数よりも優先的に選択されます。マクロ変数に代入すると、その値と型の両方に代入が行われます。

C-SPY マクロ変数とその使用方法の詳細については、286 ページの *マクロ言語についてのリファレンス情報* を参照してください。

sizeof の使用

C 規格に準じて、2 つの sizeof の構文形式をとります。

```
sizeof (タイプ)  
sizeof 式
```

前者がタイプ用で、後者が式用です。

注：C-SPY では、sizeof 演算子の使用時に式を括弧で囲いません。たとえば、sizeof (x+2) ではなく、sizeof x+2 を使用します。

変数情報の制限

C 言語の変数の値は、ステップポイント、すなわち文の先頭の命令と関数呼出しの位置でのみ有効です。その場合、エディタウィンドウで淡い緑色で強調表示されます。実際には、変数の値はそれ以外の位置でもアクセスでき、正しい値を示すことがほとんどです。

プログラムカウンタが文の内部でステップポイント以外の位置を指している場合は、その文または文の一部は通常の強調表示色よりも薄い色で強調表示されます。

最適化の影響

コンパイラは、予想される動作が実行できる範囲内であれば、可能な限り自由にアプリケーションソフトウェアを最適化します。最適化によって、コードが影響を受けます。生成されたコードがソースコードにどう関連するかが明確でなくなり、デバッグがより困難になるためです。通常は、高いレベルで最適化を行うと、コードに変更が生じ、予想と違って、変数の値を参照できなくなります。

1 つ例を示します。

```
myFunction()  
{  
    int i = 42;  
    ...  
    x = computer(i); /* i の値は C-SPY に認識されています */  
    ...  
}
```

変数 `i` が宣言されてから実際に使用されるまで、コンパイラはその変数用のスペースをスタックやレジスタに確保する必要はありません。コンパイラはこのコードを最適化できますが、C-SPY はその変数が実際に使用されるまでその値を表示できなくなります。一時的に使用できない状態にある変数の値を表示しようとする、C-SPY は以下のメッセージを表示します。

使用不可能

デバッグセッションで変数の値を常に把握する必要がある場合は、コンパイル時に一番低い最適化レベル、[なし] を指定する必要があります。

アセンブラ変数の表示

アセンブララベルは、型に関する情報を何も持たないため、C-SPY はそれ以外の情報が与えられなければ、そのラベルの位置にあるデータを簡単に表示することはできません。データを簡単に表示する方法として、デフォルトでは、C-SPY はアセンブララベルの位置にあるすべてのデータを int 型の変数として処理します。ただし、[ウォッチ]、[クイックウォッチ]、[ライブウォッチ] の各ウィンドウでは、変数宣言に合わせて別の解釈を選択することができます。

以下に示す図では、4 つの変数が [ウォッチ] ウィンドウに表示され、対応する宣言が左側のアセンブラソースファイルで示されています。

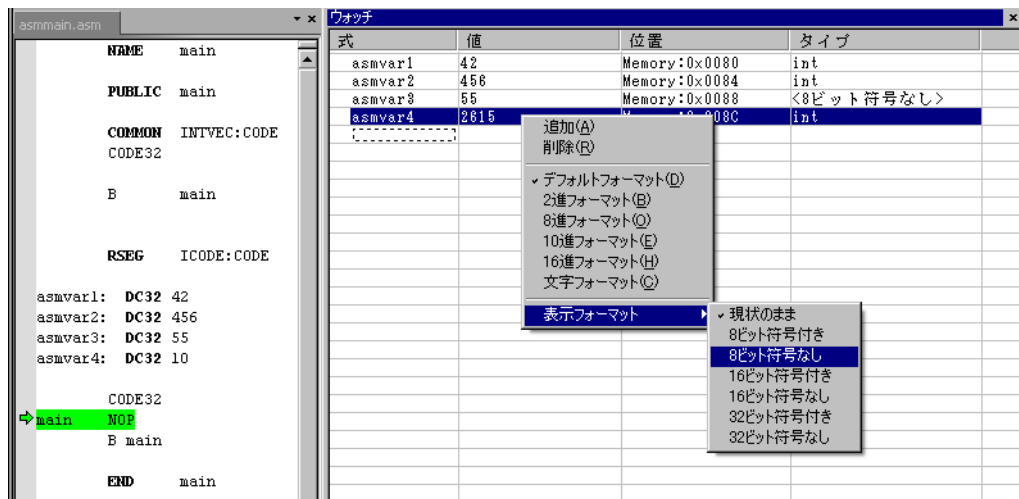


図 32: [ウォッチ] ウィンドウでのアセンブラ変数の表示

asmvar4 は、元のアセンブラ宣言では 1 バイトデータとして意図していたのに対し、[ウォッチ] ウィンドウでは int 型として表示されていることに注意してください。コンテキストメニューを使用すると、たとえば 8 ビットの符号なし変数として表示するように指定できます。asmvar3 変数は、すでにそのように指定されています。

変数と式の扱いの手順について

このセクションでは、変数および式の使用方法についてステップごとに説明します。

具体的には、以下の項目について説明します。

- 変数と式に関連するウィンドウの使用
- アセンブラ変数の表示
- データロギングを開始するには

変数と式に関連するウィンドウの使用

該当する場合、式の追加や修正、削除を行ったり、変数や式に関連するウィンドウの表示フォーマットを変更できます。

値を追加するには、点線の長方形をクリックして、調べる式を入力します。式の値を変更するには、**[値]** フィールドをクリックして、その内容を変更します。式を削除するには、式を選択して、**Delete** キーを押します。



列の長さ以上の長さのテキストは、**[トレース]** ウィンドウ以外のこれらのすべてのウィンドウで切り詰められており、該当するテキストにマウスポインタを置くと、ツールチップ情報が表示されます。

ウィンドウのいずれかで右クリックして、追加のコマンドを含むコンテキストメニューにアクセスします。**[ローカル]** ウィンドウと**データロギング**のウィンドウ、**[クイックウォッチ]** ウィンドウ以外のウィンドウでは、ウィンドウ間のドラッグアンドドロップもサポートされています。

アセンブラ変数の表示

アセンブララベルは、型に関する情報を何も持たないため、**C-SPY** はそれ以外の情報が与えられなければ、そのラベルの位置にあるデータを簡単に表示することはできません。データを簡単に表示する方法として、デフォルトでは、**C-SPY** はアセンブララベルの位置にあるすべてのデータを **int** 型の変数として処理します。ただし、**[ウォッチ]**、**[クイックウォッチ]**、**[ライブウォッチ]** の各ウィンドウでは、変数宣言に合せて別の解釈を選択することができます。

以下に示す図では、4 つの変数が [ウォッチ] ウィンドウに表示され、対応する宣言が左側のアセンブラソースファイルで示されています。

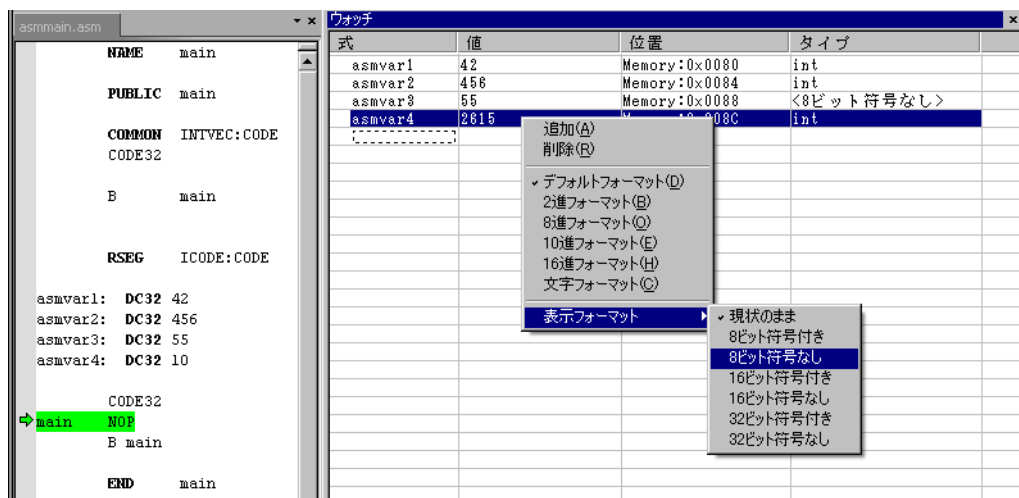


図33: [ウォッチ] ウィンドウでのアセンブラ変数の表示

asmvar4 は、元のアセンブラ宣言では 1 バイトデータとして意図していたのに対し、[ウォッチ] ウィンドウでは int 型として表示されていることに注意してください。コンテキストメニューを使用すると、たとえば 8 ビットの符号なし変数として表示するように指定できます。asmvar3 変数は、すでにそのように指定されています。

データロギングを開始するには

- 1 データロギングを設定するには、[J-Link] > [SWO 設定] または [ST-LINK] > [SWO 設定] をそれぞれ選択します。ダイアログボックスで、トレースデータの serial-wire output 通信チャンネルを設定します。特に [CPU クロック] オプションに注意してください。CPU クロックは、[プロジェクト] > [オプション] > [ST-LINK] ページでも設定できます。
- 2 [ブレークポイント] または [メモリ] ウィンドウで右クリックし、[新規ブレークポイント] > [データログ] を選択して、[ブレークポイント] ダイアログボックスを開きます。ログ情報を収集するデータにデータログブレークポイントを設定します。

- 3 C-SPY ドライバのメニューから **［データログ］** を選択して、[データログ] ウィンドウを開きます。または、以下のように選択することもできます。
 - C-SPY ドライバのメニューから **［データログ概要］** を選択して、[データログ概要] ウィンドウを開きます。
 - C-SPY ドライバのメニューから **［タイムライン］** を選択して [タイムライン] ウィンドウを開き、データロググラフを表示します。
- 4 [データログ] ウィンドウのコンテキストメニューから、**［有効化］** を選択してロギングを有効にします。
- 5 **［SWO 設定］** ダイアログボックスの **［データログイベント］** エリアで、データログが有効になっていることが確認できます。必要なロギングのレベルを以下から選択します。
 - PC のみ
 - PC + データ値 + ベースアドレス
 - データ値 + 正確なアドレス
- 6 アプリケーションプログラムの実行を開始して、ログ情報を収集します。
- 7 データログ情報を表示するには、[データログ]、[データログ概要]、[タイムライン] ウィンドウのデータグラフを参照します。
- 8 ログまたは概要をファイルに保存する場合は、対象のウィンドウのコンテキストメニューから **［ログファイルを保存］** を選択します。
- 9 データおよび割込みロギングを無効にするには、有効になっている対象のウィンドウのコンテキストメニューで **［無効］** を選択します。

注： データロギングは、J-Link/J-Trace ドライバと ST-LINK ドライバでサポートされています。

変数と式の扱いについてのリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 105 ページの **［自動］** ウィンドウ
- 105 ページの **［ローカル］** ウィンドウ
- 106 ページの **［ウォッチ］** ウィンドウ
- 108 ページの **［ライブウォッチ］** ウィンドウ
- 109 ページの **［静的］** ウィンドウ
- 111 ページの **［静的］** ダイアログボックスの選択
- 112 ページの **［クイック ウォッチ］** ウィンドウ

- 113 ページの [シンボル] ウィンドウ
- 114 ページの [シンボルの曖昧さの解決] ダイアログボックス
- 115 ページの [データログ] ウィンドウ
- 117 ページの [データログ概要] ウィンドウ

トレース関連のリファレンス情報については、185 ページの *トレースのリファレンス情報* を参照してください。

[自動] ウィンドウ

[自動] ウィンドウは [表示] メニューから利用できます。

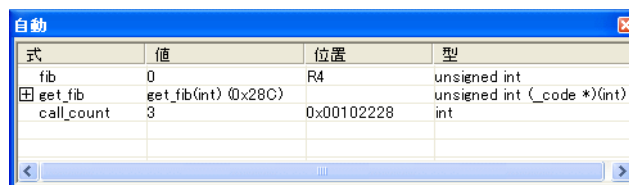


図 34: [自動] ウィンドウ

このウィンドウには、現在の文やその近くにある文の変数や式が自動的に表示されます。C-SPY で実行が停止するたびに、[自動] ウィンドウの値が再計算されます。前回停止した後に変更になった値は赤色で強調表示されます。

コンテキストメニュー

コンテキストメニューの詳細については、106 ページの [ウォッチ] ウィンドウを参照してください。

[ローカル] ウィンドウ

[ローカル] ウィンドウは [表示] メニューから利用できます。



図 35: [ローカル] ウィンドウ

このウィンドウには、現在の関数のローカル変数およびパラメータが表示されます。**C-SPY** で実行が停止するたびに、[ローカル] ウィンドウの値が再計算されます。前回停止した後に変更になった値は赤色で強調表示されます。

コンテキストメニュー

コンテキストメニューの詳細については、106 ページの [ウォッチ] ウィンドウを参照してください。

[ウォッチ] ウィンドウ

[ウォッチ] ウィンドウは [表示] メニューから利用できます。

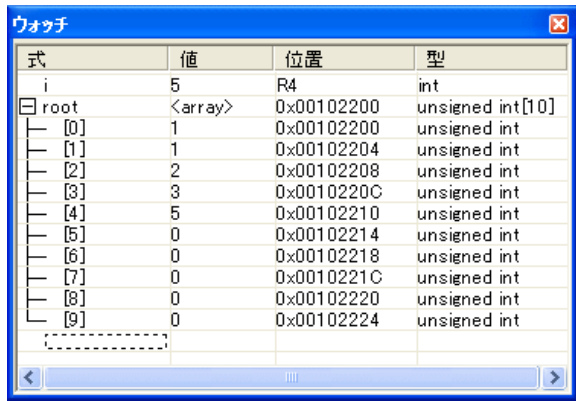


図 36: [ウォッチ] ウィンドウ

このウィンドウを使用して、**C-SPY** の式や変数の値をモニタします。式の表示や追加、修正、削除を行えます。配列、構造体、共用体は展開可能です。つまり、各エレメントの値をモニタすることができます。

C-SPY で実行が停止するたびに、[ウォッチ] ウィンドウの値が再計算されます。前回停止した後に変更になった値は赤色で強調表示されます。

コンテキストメニュー

以下のコンテキストメニューがあります。

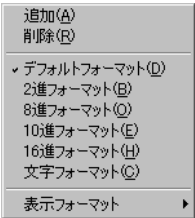


図 37: 「ウォッチ」ウィンドウのコンテキストメニュー

以下のコマンドがあります。

追加	式を追加します。
削除	選択した式を削除します。
デフォルトフォーマット	式の表示フォーマットを変更します。表示フォーマット設定は、式の種類によって適用対象が異なります。表 7 「式の種類による表示フォーマット設定の影響」を参照してください。表示フォーマットの選択は、デバッグセッションの終了後も保持されます。
2 進フォーマット	
8 進フォーマット	
10 進フォーマット	
16 進フォーマット	
文字フォーマット	
表示フォーマット	変数のデフォルトの型解釈を変更するコマンドをサブメニューで表示します。このサブメニューのコマンドは、デフォルトで整数として表示されるアセンブラの変数（アセンブララベルでのデータ）に主に使用します。詳細については、101 ページの <i>アセンブラ変数の表示</i> を参照してください。

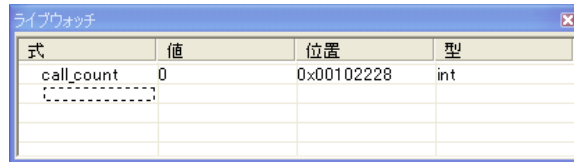
表示フォーマット設定は、式の種類によって以下のように適用対象が異なります。

式の種類	表示フォーマット設定の影響
変数	表示設定は、選択した変数だけに適用されます。他の変数には適用されません。
配列エレメント	表示設定は配列全体に適用されます。つまり、配列の各エレメントに同一の表示フォーマットが使用されます。
構造体のフィールド	定義が同一のエレメント（フィールド名、C の宣言型）に表示設定が適用されます。

表 7: 式の種類による表示フォーマット設定の影響

【ライブウォッチ】ウィンドウ

【ライブウォッチ】ウィンドウは【表示】メニューから利用できます。



式	値	位置	型
call_count	0	0x00102228	int

図38: 【ライブウォッチ】ウィンドウ

このウィンドウは繰り返しサンプリングを行い、アプリケーションの実行中に式の変数の値を表示します。式の変数は、グローバル変数のように、静的に特定できる必要があります。

このウィンドウは、この機能をサポートするハードウェアターゲットシステムでのみ使用できます。

コンテキストメニュー

コンテキストメニューの詳細については、106 ページの【ウォッチ】ウィンドウを参照してください。

また、このメニューには【オプション】コマンドが含まれています。このコマンドを選択すると、【デバッガ】ダイアログボックスが表示され、【更新間隔】オプションを設定することができます。このオプションのデフォルト値は 1000 ミリ秒です。つまり、プログラム実行中に、【ライブウォッチ】ウィンドウが 1 秒に 1 回更新されます。

[静的] ウィンドウ

[静的] ウィンドウは [表示] メニューから利用できます。



式	値	位置	型
call_count <Tutor#call_count>	0	0x00102228	int
root <Utilities#root>	<array>	0x00102200	unsigned int[10]
[0]	1	0x00102200	unsigned int
[1]	1	0x00102204	unsigned int
[2]	2	0x00102208	unsigned int
[3]	0	0x0010220C	unsigned int
[4]	0	0x00102210	unsigned int
[5]	0	0x00102214	unsigned int
[6]	0	0x00102218	unsigned int
[7]	0	0x0010221C	unsigned int
[8]	0	0x00102220	unsigned int
[9]	0	0x00102224	unsigned int

図 39: [静的] ウィンドウ

このウィンドウには、静的記憶寿命変数（通常はファイルスコープ付き変数、および関数とクラスの静的変数）の値が表示されます。volatile として宣言された静的記憶寿命変数は表示されないことに注意してください。

C-SPY で実行が停止するたびに、[静的] ウィンドウの値が再計算されます。前回停止した後に変更になった値は赤色で強調表示されます。

表示エリア

このエリアには以下の列が含まれます。

- 式

変数名。変数のベース名に続いて、モジュール、クラス、または関数スコープを含むフルネームが表示されます。この列は編集できません。
- 値

変数の値。変更された値は赤色で強調表示されます。この列は編集できます。
- 位置

この変数が格納されているメモリの場所。
- 型

変数のデータ型。

コンテキストメニュー

以下のコンテキストメニューがあります。

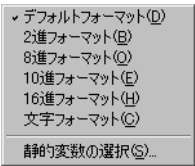


図40: [静的] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

- デフォルトフォーマット

2進フォーマット

8進フォーマット

10進フォーマット

16進フォーマット

文字フォーマット
- 式の表示フォーマットを変更します。表示フォーマット設定は、式の種類によって適用対象が異なります。表7「式の種類による表示フォーマット設定の影響」を参照してください。表示フォーマットの選択は、デバッグセッションの終了後も保持されます。

[静的] ウィンドウに表示する変数のサブセットを選択するダイアログボックスを、表示します (111 ページの [静的] ダイアログボックスの選択を参照)。

表示フォーマット設定は、式の種類によって以下のように適用対象が異なります。

式の種類	表示フォーマット設定の影響
変数	表示設定は、選択した変数だけに適用されます。他の変数には適用されません。
配列エレメント	表示設定は配列全体に適用されます。つまり、配列の各エレメントに同一の表示フォーマットが使用されます。
構造体のフィールド	定義が同一のエレメント（フィールド名、Cの宣言型）に表示設定が適用されます。

表8: 式の種類による表示フォーマット設定の影響

〔静的〕 ダイアログボックスの選択

〔静的変数の選択〕 ダイアログボックスは、〔静的〕 ウィンドウのコンテキストメニューから使用できます。

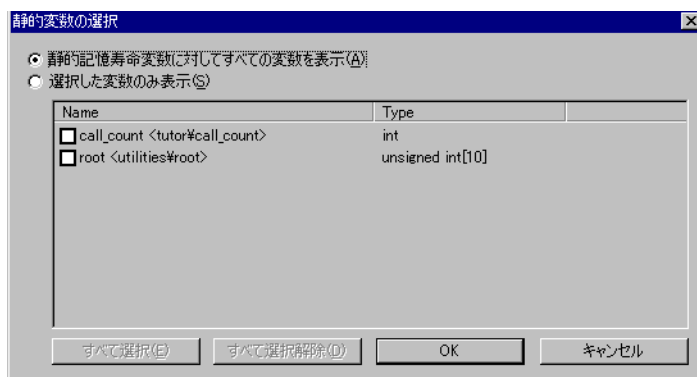


図41: 〔静的変数の選択〕 ダイアログボックス

このダイアログボックスを使用して、〔静的〕 ウィンドウで表示する変数を選択します。

静的記憶寿命変数に対してすべての変数を表示

デバッグセッションの終了後にアプリケーションに追加された新規変数を含めたすべての変数を、〔静的〕 ウィンドウに表示します。

選択した変数のみ表示

〔静的〕 ウィンドウで表示する変数を選択します。2回のデバッグセッションの間にアプリケーションに追加された新規変数は、〔静的〕 ウィンドウに自動的に表示されない点に注意してください。この変数を表示するには、変数の横にあるチェックボックスをチェックします。または、〔すべて選択〕 をクリックします。

【クイック ウォッチ】 ウィンドウ

【クイックウォッチ】ウィンドウは、**【表示】**メニューおよびエディタウィンドウのコンテキストメニューから使用できます。



図42: 【クイックウォッチ】ウィンドウ

このウィンドウを使用して、変数や式の値を監視するほか、特定の時点で式を評価します。

【ウォッチ】ウィンドウと違って、【クイックウォッチ】ウィンドウでは式を評価するタイミングを細かく制御できます。代入や C-SPY マクロ関数などアクションのある式の場合は、条件を制御しながら評価することができます（単一変数では必要ない場合が多い）。

式を評価するには、以下の手順に従います。

- 1 エディタウィンドウで評価式を右クリックして、表示されるコンテキストメニューで【クイックウォッチ】を選択します。
- 2 【クイックウォッチ】ウィンドウに式が自動的に表示されます。

別の方法は、

- 1 【クイックウォッチ】ウィンドウで、確認するファイル名を**【式】**テキストボックスに入力します。
- 2 **【再計算】**ボタンをクリックすると、式の値が計算されます。

例については、283 ページの【クイックウォッチ】によるマクロの実行を参照してください。

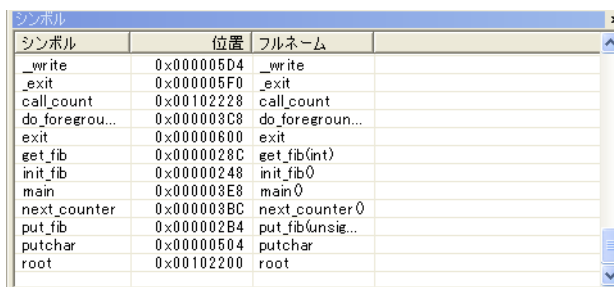
コンテキストメニュー

コンテキストメニューの詳細については、106 ページの【ウォッチ】ウィンドウを参照してください。

また、このメニューには**[[ウォッチ] ウィンドウに追加]** コマンドも表示されます。このコマンドは、選択した式を【ウォッチ】ウィンドウに追加します。

【シンボル】 ウィンドウ

【シンボル】 ウィンドウは 【表示】 メニューから利用できます。



シンボル	位置	フルネーム
_write	0x000005D4	_write
_exit	0x000005F0	_exit
call_count	0x00102228	call_count
do_foregrou...	0x000003C8	do_foregrou...
exit	0x00000600	exit
get_fib	0x0000028C	get_fib(int)
init_fib	0x00000248	init_fib()
main	0x000003E8	main()
next_counter	0x000003BC	next_counter()
put_fib	0x000002B4	put_fib(unsig...
putchar	0x00000504	putchar
root	0x00102200	root

図43: 【シンボル】 ウィンドウ

このウィンドウには、ランタイムライブラリのシンボルを含め、静的な位置を持つすべてのシンボル（すなわち、C/C++ 関数、アセンブララベル、静的記憶寿命変数）が表示されます。

表示エリア

このエリアには以下の列が含まれます。

シンボル	シンボル名。
位置	メモリアドレス。
フルネーム	シンボル名。通常は【シンボル】列の内容と同じですが、C++ メンバ関数などでは異なります。

列の見出しをクリックすると、リストがシンボル名、位置、またはフルネームによってソートされます。

コンテキストメニュー

以下のコンテキストメニューがあります。

関数
▼ 変数
▼ ラベル

図44: 【シンボル】 ウィンドウのコンテキストメニュー

以下のコマンドがあります。

関数	リスト内の関数シンボルの表示を切り替えます。
変数	リスト内の変数の表示を切り替えます。
ラベル	リスト内のラベルの表示を切り替えます。

【シンボルの曖昧さの解決】ダイアログボックス

【シンボルの曖昧さの解決】ダイアログボックスは、たとえば [逆アセンブリ] ウィンドウで移動先のシンボルを指定して、テンプレートや関数のオーバーロードのために同じシンボルが複数ある場合などに表示されます。

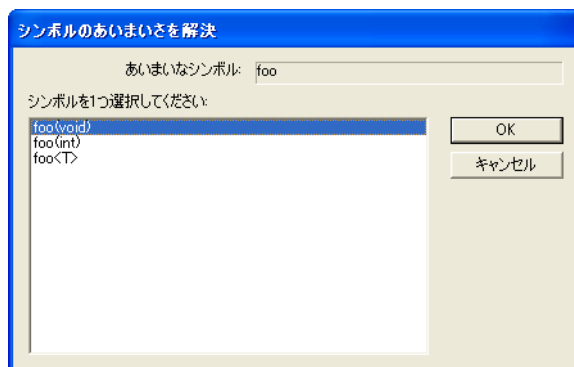


図 45: 【シンボルの曖昧さの解決】ダイアログボックス

曖昧なシンボル

曖昧なシンボルを指定します。

シンボルを1つ選択してください

曖昧なシンボルに一致する項目の一覧。使用するものを1つ選択します。

[データログ] ウィンドウ

[データログ] ウィンドウは、[J-Link] メニューまたは [ST-LINK] メニューから使用できます。

時間	プログラムカウンタ	II	アドレス	s2	アドレス
0.160us	---			W 0x0000	@ 0x2004
0.160us	0xFFE00049	-	@ 0x2000		
24.480us	0xFFE000B5			R 0x0000	@ 0x2006
24.720us	0xFFE000BF			W 0x0042	@ 0x2004
24.760us	0xFFE000C6			R 0x0042	@ 0x2006
24.960us	0xFFE000E4	W 0x00004444	@ 0x2000		
78.760us	0xFFE00104			R 0x0042	@ 0x2004+?
79.000us	---			W 0x0084	@ 0x2004
100.800us	0xFFE00104			R 0x0084	@ 0x2006
101.040us	0xFFE0010E			W 0x00C6	@ 0x2004
136.640us	Overflow				
136.880us	0xFFE0010E				@ 0x2004

白色の行はリードアクセスを示します

灰色の行はライトアクセスを示します

図46: [データログ] ウィンドウ

[データログ] ウィンドウを使用するには、以下が必要です。

- J-Link デバッグプロブ、ST-LINK デバッグプロブ、または J-Trace デバッグプロブ。J-Trace の場合は、[データログ] ウィンドウは ETM トレースが無効なときに使用可能です。ETM が有効なときは、[データログ] ウィンドウには何もデータが表示されません。
- デバッグプロブとターゲットシステム間の SWD インタフェース。

このウィンドウを使用して、最大 4 つの異なるメモリ位置またはエリアへのアクセスを記録します。

103 ページの データロギングを開始するにはも参照してください。

表示エリア

表示エリアの各行には、時刻、プログラムカウンタが表示されます。また、追跡されるデータオブジェクトごとに、その値とアドレスがこれらの列に表示されます。

時間	<p>[SWO 設定] ダイアログボックスで指定したクロック周波数に基づく、データアクセスの時間。</p> <p>ターゲットシステムが正確な時間を収集できなかった場合は、おおよその時刻が斜体で表示されます。</p> <p>この列は、コンテキストメニューから [サイクル表示] を選択した場合に有効になります。</p>
サイクル	<p>実行の開始からイベントまでのサイクルの数。この情報は、リセットでクリアされます。</p> <p>ターゲットシステムが正確な数を収集できなかった場合は、おおよその値が斜体で表示されます。</p> <p>この列は、コンテキストメニューから [サイクル表示] を選択した場合に有効になります。</p>
プログラムカウンタ *	<p>PC の内容です。メモリアクセスを実行した命令のアドレスです。</p> <p>--- が表示されている場合、ターゲットシステムがデバッガに情報を提供できなかったことを示しています。赤色で Overflow と表示されている場合、通信チャンネルがすべてのデータをターゲットシステムから送信することを示します。</p>
値	<p>アクセスタイプと、アクセスの記録対象の位置またはエリアの値（アクセスサイズを使用）を表示します。たとえば、バイトアクセスでゼロがリードされると、0x00 と表示され、ロングアクセスの場合は 0x00000000 と表示されます。</p> <p>アクセスの記録対象のデータを指定するには、[データログ] ブレークポイントダイアログボックスを使用します。143 ページの [データ] ブレークポイントダイアログボックスを参照してください。</p>

アドレス

アクセスされた実際のメモリアドレス。たとえば、1つのワードの1バイトだけアクセスされた場合は、そのバイトのアドレスだけが表示されます。アドレスは、ベースアドレス + オフセットによって算出されます。このベースアドレスは、[データログ] ブレークポイントダイアログボックスから取得され、オフセットはログから取得されます。ターゲットシステムからのログによってデバッガにオフセットが提供されなかった場合、オフセットには+?が含まれます。オフセットが表示されるようにするには、[SWO 設定] ダイアログボックスの [Value + exact addr] オプションを選択します。

* 表示エリアの行をダブルクリックできます。そのラインの PC の値がソースコードで使用可能な場合、エディタウィンドウに、対応するソースコードが表示されます（ライブラリソースコードは除く）。

コンテキストメニュー

273 ページの [割込みログ] ウィンドウのコンテキストメニューを参照してください。

[データログ概要] ウィンドウ

[データログ概要] ウィンドウは、[J-Link] メニューまたは [ST-LINK] メニューから使用できます。

Data	全てのアクセス	読み込みアクセス	書き込みアクセス
l1	2	0	1
s2	20	9	9

オーバーフローカウンタ...

図 47: [データログ概要] ウィンドウ

[データログ概要] ウィンドウを使用するには、以下が必要です。

- J-Link デバッグプローブ、ST-LINK デバッグプローブ、または J-Trace デバッグプローブ。J-Trace の場合は、[データログ概要] ウィンドウは ETM トレースが無効なときに使用可能です。ETM が有効なときは、[データログ概要] ウィンドウには何もデータが表示されません。
- デバッグプローブとターゲットシステム間の SWD インタフェース。

このウィンドウには、特定のメモリ位置またはメモリエリアへのデータアクセスの概要が表示されます。

103 ページの データロギングを開始するにはも参照してください。

表示エリア

このエリアの各行の以下の列には、各メモリ位置またはメモリエリアへのアクセスのタイプと回数が表示されます。

データ *	アクセスの記録対象のデータオブジェクトの名前。 アクセスの記録対象のデータオブジェクトを指定するには、 [データログ] ブレークポイントダイアログボックスを使用します。143 ページの [データ] ブレークポイントダイアログボックスを参照してください。
合計アクセス数 **	合計アクセス数。
リードアクセス数	合計リードアクセス数。
ライトアクセス数	合計ライトアクセス数。

* 列の最下部に、オーバーフローの数が表示されます。
** リードアクセスとライトアクセスの合計が、**[合計アクセス]** の値以下の場合、何らかの理由によりターゲットシステムが有効なアクセスタイプ情報を提供しなかったアクセスログが存在します。

コンテキストメニュー

273 ページの **[割込みログ]** ウィンドウのコンテキストメニューを参照してください。

ブレークポイントの使用

この章では、ブレークポイントおよびそれらを定義してモニタするさまざまな方法について説明します。具体的には以下の項目を解説します。

- ブレークポイントの設定と使用の概要
- ブレークポイントの設定の手順
- ブレークポイントのリファレンス情報

ブレークポイントの設定と使用の概要

このセクションではブレークポイントの概要を説明します。

以下のトピックを解説します。

- ブレークポイントを使用する理由
- ブレークポイントの設定の概略
- ブレークポイントの種類
- ブレークポイントアイコン
- C-SPY シミュレータのブレークポイント
- C-SPY ハードウェアドライバのブレークポイント
- ブレークポイントの設定元
- [ブレークポイント] オプション
- 例外ベクタのブレークポイント
- `_ramfunc` 宣言関数のブレークポイントの設定

ブレークポイントを使用する理由

C-SPY® を使用すると、デバッグ中のアプリケーションでさまざまな種類のブレークポイントを設定して、必要な位置で実行を停止することができます。ブレークポイントをコード部分に設定すると、プログラムロジックが正しいかどうかを調べたり、トレースを出力したりできます。コードブレークポイントの他に、使用している C-SPY ドライバによっては、別の種類のブレークポイントを使用できる場合があります。たとえば、データブレークポイントを設定すると、データがいつどのように変更されるかを調べることができます。

ユーザが指定した特定の条件が成立したときに実行を停止させることができます。また、非表示で実行を停止してから再開することにより、C-SPY マクロ関数の実行などの 2 次アクションをブレイクポイントからトリガすることもできます。マクロ関数を定義すると、ハードウェアの動作のシミュレーションなど、さまざまなアクションを実行できます。

このようにさまざまな使い方ができるため、アプリケーションのステータスを検証するための柔軟なツールとして使用できます。

ブレイクポイントの設定の概略

ブレイクポイントは異なるレベルの相互作用や精度、タイミング、自動化に合わせて、さまざまな数多くの方法で設定できます。定義したすべてのブレイクポイントが [ブレイクポイント] ウィンドウに表示されます。このウィンドウでは、すべてのブレイクポイントの表示、ブレイクポイントの有効化 / 無効化、新しいブレイクポイントを定義するためのダイアログボックスの表示を実行できます。[ブレイクポイントの使用] ダイアログボックスには、内部的に使用されるすべてのブレイクポイントもリストされます (124 ページの *ブレイクポイントの設定元* を参照)。

ブレイクポイントは、ステップ動作と同じメカニズムを使用して、行単位よりも細かい精度で設定されます。精度に関する詳細については、80 ページの *ステップ実行* を参照してください。

デバッグセッションがアクティブでなくても、コードを編集しながらブレイクポイントを設定できます。設定したブレイクポイントは、デバッグセッションを開始するときに検証されます。ブレイクポイントはデバッグセッション終了後も保持されます。

注：ほとんどのハードウェアデバッグシステムでは、アプリケーションが実行中でないときにだけブレイクポイントを設定できます。

ブレイクポイントの種類

使用している C-SPY ドライバによっては、C-SPY で別の種類のブレイクポイントを使用できる場合があります。

コードブレイクポイント

コードブレイクポイントは、プログラムロジックが正しいかどうかや、トレースの出力を取得するためにコードの位置を探すときに使用します。コードブレイクポイントは、指定位置から命令をフェッチした時にトリガされます。特定のマシン命令にブレイクポイントを設定した場合は、命令の実行前にブレイクポイントがトリガされ、実行が停止します。

ログブレークポイント

ログブレークポイントは、アプリケーションのソースコードにコードを追加することなく、トレース出力を追加する便利な方法です。ログブレークポイントは、指定位置から命令をフェッチ時にトリガされます。特定のマシン命令にブレークポイントを設定した場合は、命令の実行前にブレークポイントがトリガされ、実行が一時停止し、指定したメッセージが [C-SPY デバッグ ログ] ウィンドウに出力されます。

トレースブレークポイント

トレース開始および停止ブレークポイントは、トレースデータの収集を開始および停止します。これは、2つの実行ポイント間で命令を解析する便利な方法です。

データブレークポイント

主にメモリ上の固定アドレスに割り当てられた変数に使用します。アクセス可能なローカル変数にブレークポイントを設定した場合、実際には対応するメモリアドレス（ロケーション）に設定されます。この位置の妥当性が保証されるのは、コードの一部だけです。データブレークポイントは、指定された位置のデータがアクセスされたときにトリガされます。通常は、データにアクセスする命令が実行された直後に、実行が停止します。

イミディエイトブレークポイント

C-SPY シミュレータでは、イミディエイトブレークポイントを設定できます。これによって、命令の実行が一時的に停止します。このブレークポイントを使用すると、シミュレーションされたプロセッサがある位置からデータを読み込む直前かある位置にデータを書き込んだ直後に、C-SPY マクロ関数を呼び出すことができます。アクションが終了すると、命令の実行が再開されます。

イミディエイトブレークポイントは、メモリにマッピングされたさまざまな種類のデバイス（シリアルポートやタイマなど）をシミュレーションする場合に便利です。シミュレーションされたプロセッサがデバイスがメモリマッピングされた位置から読み込むと、C-SPY マクロ関数が実行されて適切なデータを供給します。逆に、デバイスがメモリマッピングされた位置にシミュレーションされたプロセッサが書き込むと、C-SPY マクロ関数が実行されて、書き込まれた値に応じた適切な動作を実行します。

データログブレークポイント

データログブレークポイントは、Cortex-M デバイスを使用する際に J-Link/J-Trace ドライバと ST-LINK ドライバで使用可能です。

データログブレークポイントは、指定された位置のデータがアクセスされたときにトリガされます。特定のアドレスまたは範囲にログブレークポイントを設定した場合は、その位置へのアクセスが発生するたびに、ログメッセージが [SWO トレース] ウィンドウに表示されます。ログメッセージは [データログ] ウィンドウにも表示されます（ウィンドウが有効な場合）。ただし、これらのログメッセージを使用するには、[SWO 設定] ダイアログボックスでトレースデータを設定しておく必要があります。190 ページの [SWO 設定] ダイアログボックスを参照してください。

JTAG ウォッチポイント

C-SPY J-Link/J-Trace ドライバおよび C-SPY Macraigor ドライバでは、ARM7/9 コアの JTAG ウォッチポイント機構を活用できます。

ウォッチポイントは、ARM EmbeddedICE のマクロセルが提供する機能を使用して実装されます。このマクロセルは、JTAG インタフェースをサポートするすべての ARM コアの一部です。EmbeddedICE ウォッチポイントコンパレータでは、アドレスバス、データバス、CPU 制御信号、および外部入力信号と、定義されたウォッチポイントをリアルタイムで比較します。定義された条件がすべて真の場合、プログラムが中断します。

ウォッチポイントは、C-SPY で暗黙的に使用して、アプリケーションのコードブレークポイントやデータブレークポイントを設定します。リード/ライトメモリにブレークポイントを設定する場合、デバッガではウォッチポイントが 1 つのみ必要です。リードオンリーメモリにブレークポイントを設定する場合、各ブレークポイントには 1 つのウォッチポイントが必要です。マクロセルでは 2 つのハードウェアウォッチポイントしか実装しないため、リードオンリーメモリのブレークポイントの最大値は 2 つです。

ARM JTAG ウォッチポイント機構の詳細については、Advanced RISC Machines Ltd の以下の資料を参照してください。

- 『ARM7TDMI (rev 3) Technical Reference Manual』の第 5 章「Debug Interface」および付録 B の「Debug in Depth」
- アプリケーションノート 28 の「The ARM7TDMI Debug Architecture」

ブレイクポイントアイコン

ブレイクポイントはエディタウィンドウの左余白にあるアイコンでマークを付け、コードブレイクポイント用とログブレイクポイント用で違うアイコンを使用します。

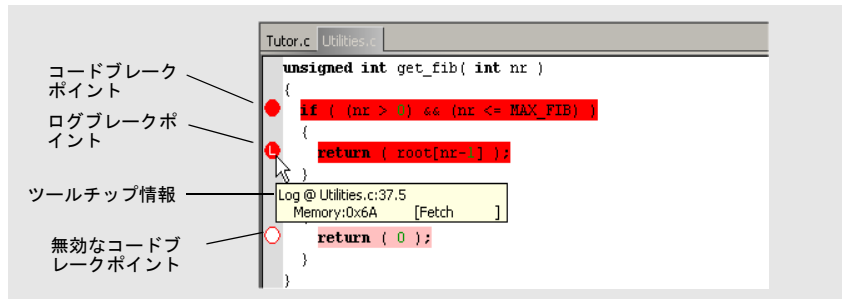


図48: ブレイクポイントアイコン



ブレイクポイントアイコンが表示されない場合は、[ブックマークの表示] オプションが選択されていることを確認します (『ARM 用 IDE プロジェクト管理およびビルドガイド』のエディタオプションを参照)。



マウスポインタをブレイクポイントアイコンに置くだけで、同じ位置に設定したすべてのブレイクポイントに関する詳細なツールチップ情報を取得できます。最初の行がユーザブレイクポイント情報を、後続する行が、ユーザブレイクポイントの実装に使用する物理ブレイクポイントを説明します。後者の情報は、[ブレイクポイントの使用] (ブレイクポイントの使用) ダイアログボックスでも表示されます。

注: ブレイクポイントアイコンは、使用している C-SPY ドライバによって異なった外観になります。

C-SPY シミュレータのブレイクポイント

C-SPY シミュレータは全種類のブレイクポイントをサポートしており、ブレイクポイントを無制限に設定することができます。

C-SPY ハードウェアドライバのブレイクポイント

ハードウェアデバッグシステムに C-SPY ドライバを使用して、さまざまな種類のブレイクポイントを設定できます。設定可能なブレイクポイントの数は、ターゲットシステム上で使用できるハードウェアブレイクポイントの数や、ソフトウェアブレイクポイントが有効かどうかによって変わります (有効な場合は、設定できるブレイクポイントの数は限られます)。

ソフトウェアブレークポイントが有効な場合、デバッガはソフトウェアブレークポイントより先に、まず利用可能なハードウェアブレークポイントを使用します。ソフトウェアブレークポイントが有効でない場合に、使用可能なハードウェアブレークポイントの数を超えると、デバッガがシングルステップで実行するようになります。この場合、大幅に実行速度が低下します。このため、異なるブレークポイントの設定元に注意する必要があります。

異なるターゲットシステムのブレークポイントの特徴については、メーカーのドキュメントを参照してください。

ブレークポイントの設定元

デバッガシステムには複数のブレークポイントの設定元が存在します。

ユーザブレークポイント

[ブレークポイント] ダイアログボックスで定義したり、[エディタ] ウィンドウでブレークポイントを切り替えると、通常は物理的なブレークポイントが1つ使用されますが、これは状況に応じて大きく異なります。一部のユーザブレークポイントは複数の物理的ブレークポイントを使用します。逆に複数のユーザブレークポイントが1つの物理的ブレークポイントを共有することもできます。ユーザブレークポイントは、たとえば「Data @[R] callCount」のように、[ブレークポイントの使用] ダイアログボックスと [ブレークポイント] ウィンドウに同じように表示されます。

C-SPY 自身

C-SPY 自身もブレークポイントを使用します。C-SPY は以下の場合にブレークポイントを設定します。

- デバッガオプション **[実行]** が選択され、いずれかのステップコマンドが使用されている場合。これらはデバッガシステムの実行時にのみ設定される一時的なブレークポイントです。したがって、[ブレークポイントの使用] ウィンドウにはこれらのブレークポイントは表示されません。
- **[セミホスティング]** または **[IAR ブレークポイント]** オプションが選択されている。

これらのブレークポイントの設定元は、[ブレークポイントの使用] ダイアログボックスに、たとえば「C-SPY Terminal I/O & libsupport module」のように表示されます。

C-SPY プラグインモジュール

たとえば、リアルタイムオペレーティングシステム用のモジュールは、追加のブレイクポイントを使用します。特にデフォルトでは、[スタック] ウィンドウで物理的ブレイクポイントを1つ使用します。

[スタック] ウィンドウで使用するブレイクポイントを無効にするには、以下の操作を行います。

- 1 [ツール] > [オプション] > [スタック] を選択します。
- 2 [プログラム開始までスタックポインタを無効にする] の [ラベル] オプションの選択を解除します。

[ブレイクポイント] オプション

以下のハードウェアデバッガシステムでは、C-SPY を起動する前にドライバ固有のブレイクポイントオプションをいくつか設定できます。

- GDB サーバ
- J-Link/J-Trace JTAG プロローブ
- Macraigor JTAG プロローブ

詳細については、147 ページの [ブレイクポイント] オプションを参照してください。

例外ベクタのブレイクポイント

ARM9、Cortex-R4、Cortex-M3 の各デバイスで例外ベクタにブレイクポイントを設定できます。[ベクタキャッチ] ダイアログボックスを使用すると、ハードウェアのブレイクポイントを使用せずに、割込みベクタテーブルのベクタにブレイクポイントを直接設定することができます。詳細については、150 ページの [ベクタキャッチ] ダイアログボックスを参照してください。

J-Link/J-Trace ドライバおよび RDI ドライバの場合、オプションダイアログボックスでベクタに直接ブレイクポイントを設定できます。詳細については、374 ページの J-Link/J-Trace の設定オプションおよび 385 ページの RDI を参照してください。

__RAMFUNC 宣言関数のブレイクポイントの設定

__ramfunc 宣言関数にブレイクポイントを設定するには、プログラムの実行が main 関数に達する必要があります。システム起動コードでは、すべての __ramfunc 宣言関数を、コードが格納されている場所（一般的にはフラッシュメモリ）から RAM まで移動します。つまり、__ramfunc 宣言関数は適切な場所にはないため、main 関数を実行するまでブレイクポイントを設定できません。この問題を回避するには、[ソフトウェアブレイクポイント復元位

置] オプションを使用します。148 ページの ソフトウェアブレイクポイント 復元位置を参照してください。

また、エディタから追加された `__ramfunc` 宣言関数のブレイクポイントは、C-SPY の起動前およびデバッグセッションの終了前に無効にする必要があります。

`__ramfunc` キーワードについては、*ARM 用 IAR C/C++ 開発ガイド*を参照してください。

ブレイクポイントの設定の手順

このセクションでは、ブレイクポイントの設定と使用方法についてステップごとに説明します。

具体的には、以下の項目について説明します。

- ブレイクポイントのさまざまな設定方法
- シンプルなコードブレイクポイントトグル
- ダイアログボックスを使用したブレイクポイントの設定
- [メモリ] ウィンドウでのデータブレイクポイントの設定
- システムマクロを使用したブレイクポイントの設定
- ブレイクポイントのヒント

ブレイクポイントのさまざまな設定方法

ブレイクポイントは、さまざまな方法で設定できます。

- [ブレイクポイントの切替え] コマンドを使用すると、コードブレイクポイントが切り替わります。このコマンドは、[ツール] メニューのほか、エディタウィンドウや [逆アセンブリ] ウィンドウでのコンテキストメニューからも使用できます。
- エディタウィンドウまたは [逆アセンブリ] ウィンドウの左側の余白部分でダブルクリックすると、コードブレイクポイントが切り替わります。
- エディタウィンドウ、[ブレイクポイント] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから [新規ブレイクポイント] ダイアログボックスおよび [ブレイクポイントの編集] ダイアログボックスを使用する方法。これらのダイアログボックスでは、すべてのブレイクポイントオプションにアクセスできます。
- [メモリ] ウィンドウでメモリエリアに直接データブレイクポイントを設定する方法。
- 定義済システムマクロを使用してブレイクポイントを設定する方法。自動化が可能になります。

方法によって簡単さ、複雑さ、自動化のレベルが異なります。

シンプルなコードブレイクポイントトグル

コードブレイクポイントのトグルは、簡単にブレイクポイントを設定するための方法です。エディタウィンドウと [逆アセンブリ] ウィンドウの両方で、次の方法を使用できます。



- ウィンドウの左側の灰色で表示された余白部分をダブルクリック
- ブレイクポイントを設定する C 言語のソース文、アセンブラ命令に挿入ポイントを配置して、ツールバーの [ブレイクポイントの切替え] ボタンをクリック
- [編集] > [ブレイクポイントの切替え] を選択
- 右クリックして、表示されるコンテキストメニューで [ブレイクポイントの切替え] を選択

ダイアログボックスを使用したブレイクポイントの設定

ブレイクポイントダイアログボックスを使用する利点は、グラフィカルインタフェースで対話的にブレイクポイントの特性を微調整できるということです。この方法では、オプションを設定した後、すぐにブレイクポイントが意図したとおりに動作するかどうかをテストできます。

ブレイクポイントダイアログボックスで定義したブレイクポイントはすべて、デバッグセッションが終了後も保持されます。

新しいブレイクポイントを設定するには：

ブレイクポイントダイアログボックスは、エディタウィンドウ、[ブレイクポイント] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから開くことができます。

- 1 [表示] > [ブレイクポイント] を選択して、[ブレイクポイント] ウィンドウを開きます。
- 2 [ブレイクポイント] ウィンドウで右クリックし、コンテキストメニューで [新規ブレイクポイント] を選択します。
- 3 サブメニューで、設定するブレイクポイントの種類を選択します。

使用している C-SPY ドライバによっては、別の種類のブレイクポイントを使用できる場合があります。

- 4 表示される [ブレイクポイント] ダイアログボックスで、ブレイクポイントの設定を指定して [OK] をクリックします。

ブレイクポイントが [ブレイクポイント] ウィンドウに表示されます。

既存のブレイクポイントを変更するには：

- 1 [ブレイクポイント] ウィンドウ、エディタウィンドウ、または [逆アセンブリ] ウィンドウで変更するブレイクポイントを選択し、右クリックしてコンテキストメニューを開きます。

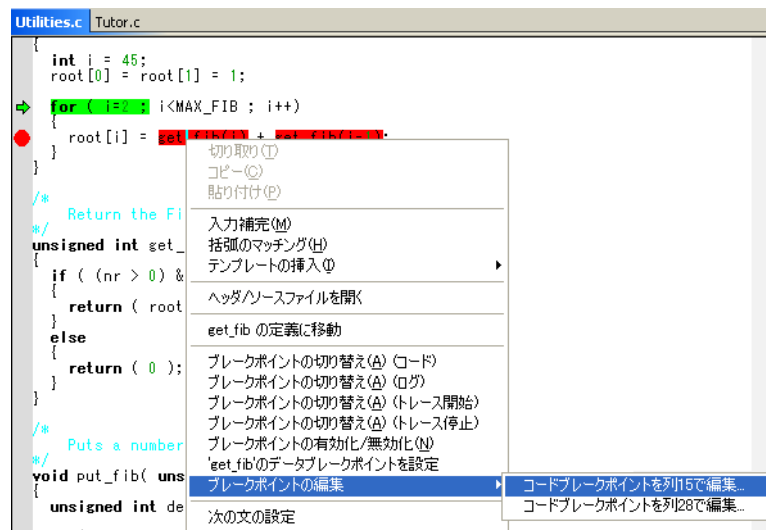


図 49: コンテキストメニューからのブレイクポイントの変更

同じソースコードの行に複数のブレイクポイントが設定されている場合、それらはサブメニューに一覧表示されます。

- 2 コンテキストメニューで、目的のコマンドを選択します。
- 3 表示される [ブレイクポイント] ダイアログボックスで、ブレイクポイントの設定をしてして [OK] をクリックします。

ブレイクポイントが [ブレイクポイント] ウィンドウに表示されます。

[メモリ] ウィンドウでのデータブレイクポイントの設定

[メモリ] ウィンドウでメモリロケーションにブレイクポイントを直接設定することができます。ウィンドウを右クリックして、表示されるコンテキストメニューからブレイクポイントコマンドを選択します。範囲にブレイクポイントを設定するには、メモリの該当領域を選択します。

ブレイクポイントは [メモリ] ウィンドウでは強調表示されません。代わりに、[表示] メニューの [ブレイクポイント] ウィンドウを使用して、確認や編集、削除ができます。[メモリ] ウィンドウで設定したブレイクポイント

は、リードとライトの両方のアクセスでトリガされます。このウィンドウで定義したブレークポイントはすべて、デバッグセッションが終了後も保持されます。

注：[メモリ] ウィンドウで直接ブレークポイントを設定するには、使用するドライバでそれがサポートされている必要があります。

システムマクロを使用したブレークポイントの設定

ブレークポイントの設定は、[ブレークポイント] ダイアログボックス以外に、C-SPY の組込みシステムマクロでも行えます。ブレークポイントの設定にシステムマクロを使用する場合、ブレークポイントの特徴をマクロのパラメータとして指定します。

マクロによる定義は、要求どおりのブレークポイント設定ができない場合に便利です。組込みのシステムマクロを使用してブレークポイントをマクロファイルに定義し、C-SPY の起動時にマクロファイルを実行することができます。これにより、ブレークポイントは、C-SPY を起動するたびに自動的に設定されます。他にも、デバッグセッションがドキュメント化される、開発プロジェクトに携わる複数のエンジニア間でマクロファイルを共有できるといった長所があります。

注：システムマクロを使用して設定されたブレークポイントも、[ブレークポイント] ウィンドウで表示や変更を行えます。ダイアログボックスを使用して定義されたブレークポイントと異なり、システムマクロを使用して定義されたブレークポイントはデバッグセッションを終了するとすべて削除されます。

以下のブレークポイントマクロが使用できます。

ブレークポイント用の C-SPY マクロ	シミュ レータ	J-Link	RDI	Mac- raigor	GDB サーバ	ST- Link	LMI FTDI	Angel	IAR ROM- モニタ
__setCodeBreak	X	X	X	X	X	X	X	X	X
__setDataBreak	X	--	--	--	--	--	--	--	--
__setLogBreak	X	X	X	X	X	X	X	X	X
__setSimBreak	X	--	--	--	--	--	--	--	--
__setTraceStartBreak	X	--	--	--	--	--	--	--	--
__setTraceStopBreak	X	--	--	--	--	--	--	--	--
__clearBreak	X	X	X	X	X	X	X	X	X

表9: ブレークポイント用の C-SPY マクロ

各ブレークポイントマクロの詳細については、「292 ページの C-SPY システムマクロについてのリファレンス情報」を参照してください。

セットアップマクロファイルを使用して C-SPY 起動時にブレークポイントを設定

セットアップマクロファイルを使用して C-SPY の起動時にブレークポイントを定義できます。手順の詳細については、282 ページの *セットアップマクロとセットアップファイルによる登録と実行* を参照してください。

例外ベクタ上へのブレークポイントの設定

この手順は、J-Link/J-Trace および Macraigor に該当します。

例外ベクタ上にブレークポイントを設定するには：

- 1 正しいデバイスを選択します。C-SPY を起動する前に、[プロジェクト] > [オプション] を選択して、[一般オプション] カテゴリを選択します。[ターゲット] ページで使用可能な [派生プロセッサ] ドロップダウンリストから、該当するコアまたはデバイスを選択します。
- 2 C-SPY を起動します。
- 3 [J-Link]> [ベクタキャッチ] を選択します。デフォルトでは、ベクタはブレークポイントオプションページの設定に基づいて選択されます (147 ページの [ブレークポイント] オプションを参照)。
- 4 [ベクタキャッチ] ダイアログボックスで、ブレークポイントを設定するベクタを選択し、[OK] をクリックします。ブレークポイントは、例外の開始時のみトリガされます。

ブレークポイントのヒント

以下は、ブレークポイントの設定に関連して役に立つヒントです。



不正な関数引数のトレース

ポインタ引数を持つ関数が時々 NULL 引数によって誤って呼び出される場合、その動作をデバッグした方がよいときがあります。以下の方法が役に立ちます。

- 関数の最初の行にブレークポイントを設定して、パラメータが 0 のときにだけ条件が真となるようにします。このブレークポイントは、問題となる状況が実際に発生するまでトリガされません。この方法の利点は、余分なソースコードが必要ないことです。欠点は、実行速度が極端に低下する可能性があることです。
- 問題のある関数で `assert` マクロを使用できます。たとえば、次のようになります。

```
int MyFunction(int * MyPtr)
```

```
{
    assert(MyPtr != 0); /* アサートマクロがソースコードに追加されます。*/
    /* 関数の残りがここに入ります */
}
```

条件が真のときは必ず実行が中断します。利点は、実行速度がわずかにしか影響を受けないことですが、欠点はソースコードに小さいフットプリントが追加されることです。また、実行の停止を除去する唯一の方法は、マクロを削除してソースコードをリビルドすることです。

- assert マクロを使用する代わりに、次のように関数を修正できます。

```
int MyFunction(int * MyPtr)
{
    if(MyPtr == 0)
        MyDummyStatement; /* ブレイクポイントを設定するダミーの文 */
    /* 関数の残りがここに入ります */
}
```

また、条件が真のときに常に実行が中断するように、追加のダミー文にブレイクポイントを設定する必要があります。利点は、実行速度がわずかにしか影響を受けないことですが、欠点はソースコードに小さいフットプリントが追加されることです。ただし、この方法ではブレイクポイントを削除するだけで、実行の停止を除去することができます。



タスクを処理して実行を継続する

ブレイクポイントがトリガされたらタスクを処理して、自動的に実行を継続することができます。

[アクション] テキストボックスを使用すると、C-SPY マクロ関数などのアクションをブレイクポイントに関連付けることができます。ブレイクポイントがトリガされ、アプリケーションの実行が停止すると、マクロ関数が実行されます。この場合は、実行は自動的に継続されません。

代わりに、0（偽）を返す条件を設定できます。ブレイクポイントがトリガされると、条件（タスクを実行する C-SPY マクロの呼出しなど）が評価され、真ではないために実行が継続します。

C-SPY マクロ関数が単純なタスクを実行する例を考えます。

```
__var my_counter;

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

この関数をブレークポイントの条件として使用するには、[条件] の [式] テキストボックスに「count()」と入力します。これにより、ブレークポイントがトリガされると、タスクが実行されます。マクロ関数 count は常に 0 を返すため、条件は偽であり、プログラムは停止することなく自動的に再開されます。

ブレークポイントのリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 133 ページの [ブレークポイント] ウィンドウ
- 135 ページの [ブレークポイントの使用] ダイアログボックス
- 136 ページの [コード] ブレークポイントダイアログボックス
- 138 ページの [JTAG ウォッチポイント] ダイアログボックス
- 141 ページの [ログ] ブレークポイントダイアログボックス
- 143 ページの [データ] ブレークポイントダイアログボックス
- 145 ページの [データログ] ブレークポイントダイアログボックス
- 147 ページの [ブレークポイント] オプション
- 149 ページの [イミディエイトブレークポイント] ダイアログボックス
- 150 ページの [ベクタキャッチ] ダイアログボックス
- 150 ページの [位置入力] ダイアログボックス
- 152 ページの [ソースの曖昧さの解決] ダイアログボックス

以下も参照してください。

- 292 ページの C-SPY システムマクロについてのリファレンス情報
- 185 ページの トレースのリファレンス情報

「ブレイクポイント」ウィンドウ

「ブレイクポイント」ウィンドウは「表示」メニューから利用できます。

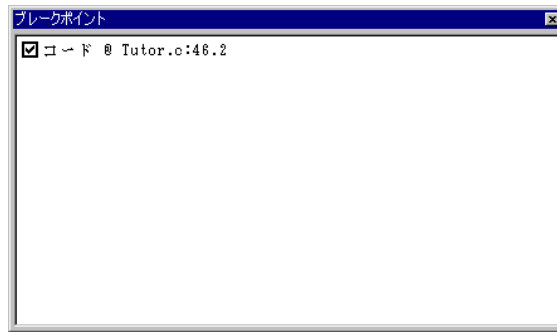


図 50: 「ブレイクポイント」ウィンドウ

「ブレイクポイント」ウィンドウには、定義するすべてのブレイクポイントが一覧表示されます。

このウィンドウでは、ブレイクポイントのモニタや有効/無効の切替えを簡単に行うことができます。また、新しいブレイクポイントの定義や、既存のブレイクポイントの修正も行うことができます。

表示エリア

このエリアには、定義するすべてのブレイクポイントが一覧表示されます。それぞれのブレイクポイントについて、ブレイクポイントの種類、ソースファイル、ソース行、ソース列の情報が表示されます。

コンテキストメニュー

以下のコンテキストメニューがあります。

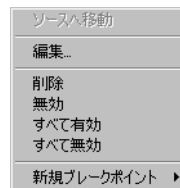


図 51: 「ブレイクポイント」ウィンドウのコンテキストメニュー

以下のコマンドがあります。

ソースへ移動	ブレイクポイントに対応する位置がソースにある場合に、挿入ポイントをブレイクポイント位置に移動します。[ブレイクポイント] ウィンドウでブレイクポイントをダブルクリックした場合も、同一の操作が実行されます。
編集	選択したブレイクポイントについて、[ブレイクポイント] ダイアログボックスを表示します。
削除	ブレイクポイントを削除します。Delete キーを押した場合も、同一の操作が実行されます。
有効化	ブレイクポイントを有効にします。行の最初にあるチェックボックスが選択されます。チェックボックスを手動で選択しても、同一の結果になります。このコマンドは、ブレイクポイントが無効になっている場合にだけ使用できます。
無効	ブレイクポイントを無効にします。行の最初にあるチェックボックスが選択解除されます。チェックボックスを手動で選択解除しても、このコマンドを実行できます。このコマンドは、ブレイクポイントが有効になっている場合にだけ使用できます。
すべて有効	定義されたすべてのブレイクポイントを有効にします。
すべて無効	定義されたすべてのブレイクポイントを無効にします。
新規ブレイクポイント	[ブレイクポイント] ダイアログボックスを開くためのサブメニューを表示します。ここで、使用可能な種類のブレイクポイントを定義できます。このダイアログボックスを使用して定義したブレイクポイントはすべて、デバッグセッションが終了後も保持されます。

【ブレイクポイントの使用】 ダイアログボックス

【ブレイクポイントの使用】 ダイアログボックスは、使用する C-SPY ドライバに固有のメニューから利用できます。

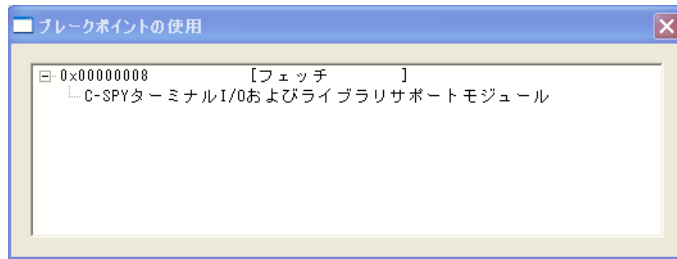


図 52: 【ブレイクポイントの使用】 ダイアログボックス

【ブレイクポイントの使用】 ダイアログボックスには、ターゲットシステムで現在設定されているすべてのブレイクポイントのリストが表示されます。これらのブレイクポイントには、ユーザ定義によるブレイクポイントと C-SPY が内部的に使用しているブレイクポイントが含まれます。このダイアログボックスの項目のフォーマットは、使用している C-SPY ドライバによって異なります。

このダイアログボックスでは、すべてのブレイクポイントの概要が表示されます。これらの項目は、【ブレイクポイント】 ダイアログボックスで表示されるブレイクポイントのリストと関連はありますが、同一ではありません。

C-SPY はステップの実行時にブレイクポイントを使用します。ターゲットシステムでハードウェアブレイクポイントの数量に上限があり、ソフトウェアブレイクポイントが有効でない場合、使用可能なハードウェアブレイクポイントの数を超えると、デバッガがシングルステップで実行するようになります。この場合、大幅に実行速度が低下します。そのため、ハードウェアブレイクポイントの数が限られているデバッガシステムでは、以下の目的で【ブレイクポイントの使用】 ダイアログボックスを使用すると便利です。

- すべてのブレイクポイント設定元の特定
- ターゲットシステムでサポートされているアクティブなブレイクポイントの数をチェック
- 可能であれば、使用できるブレイクポイントを効率よく利用できるようにデバッガを設定

表示エリア

リスト内の各ブレイクポイントについて、アドレスとアクセスタイプが表示されます。また、リストの各ブレイクポイントを拡張すると、その発生元が表示されます。

【コード】 ブレイクポイントダイアログボックス

【コード】 ブレイクポイントダイアログボックスは、エディタウィンドウ、[ブレイクポイント] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから開くことができます。

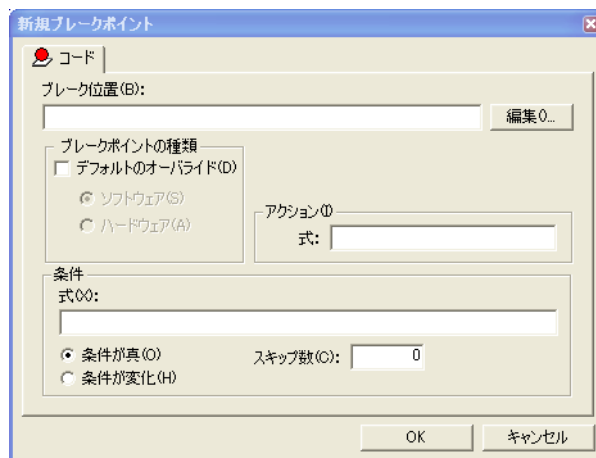


図 53: 【コード】 ブレイクポイントダイアログボックス

【コード】 ブレイクポイントダイアログボックスを使用して、コードブレイクポイントを設定します。

注: 【コード】 ブレイクポイントダイアログボックスは、使用する C-SPY ドライバによって異なります。使用する C-SPY ドライバでのブレイクポイントのサポートについては、123 ページの *C-SPY* ハードウェアドライバのブレイクポイントを参照してください。

ブレイク位置

【ブレイク位置】 テキストボックスでブレイクポイントの位置を指定します。または、【編集】 ボタンをクリックして【位置入力】ダイアログボックスを表示します（150 ページの【位置入力】ダイアログボックスを参照）。

ブレイクポイントの種類

デフォルトのブレイクポイントの種類をオーバーライドします。[デフォルトのオーバーライド] チェックボックスを選択し、[ソフトウェア] と [ハードウェア] オプションから選択します。

以下の C-SPY ドライバについて、ブレイクポイントタイプを指定できます。

- GDB サーバ
- J-Link/J-Trace JTAG プロローブ
- Macraigor JTAG プロローブ

サイズ

ブレイクポイントがトリガされる位置にサイズ（特に範囲）があるべきかを指定します。指定したメモリ範囲に対してフェッチアクセスが発生するごとに、ブレイクポイントがトリガされます。サイズの指定方法を選択します。

自動	サイズが自動設定される（通常は 1）。
手動	テキストボックスでブレイクポイント範囲のサイズを指定します。

アクション

ブレイクポイントに関連するアクションがあるかどうかを決定します。C-SPY マクロ関数などの式を指定すると、ブレイクポイントのトリガ時に条件が真であるときに評価されます。

条件

単純または複雑な条件を指定します。

式	C-SPY 式の構文に準拠する有効な式を指定します。
条件が真	式の値が真の場合に、ブレイクポイントがトリガされます。
条件（変更）	最後の評価時以降に式の値が変化した場合に、ブレイクポイントがトリガされます。
スキップ数	ブレイクポイントがトリガを開始するまでにブレイクポイント条件が真となる回数を指定します。この回数に達すると、条件が満たされるたびにブレイクポイントがトリガされます。

【JTAG ウォッチポイント】 ダイアログボックス

【JTAG ウォッチポイント】 ダイアログボックスは、ドライバ個別のメニューから開きます。

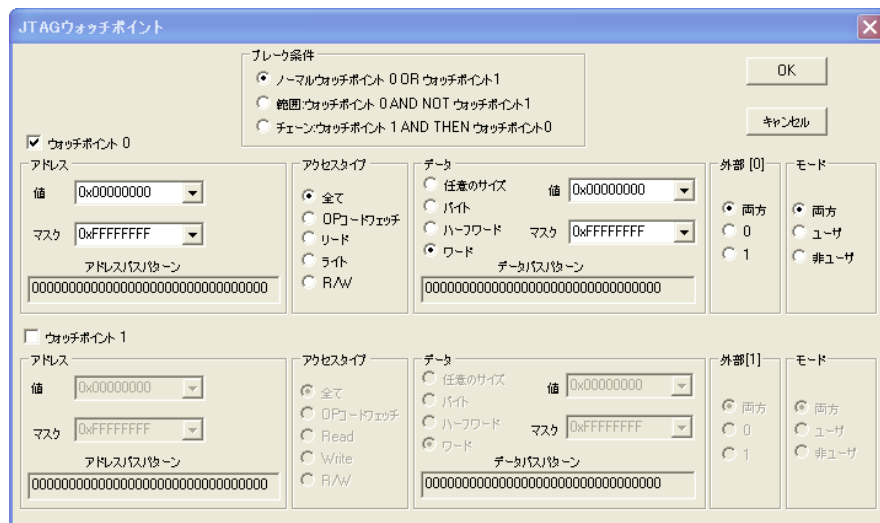


図 54: 【JTAG ウォッチポイント】 ダイアログボックス

このダイアログボックスを使用して、2つのハードウェアウォッチポイント装置を直接制御します。必要なウォッチポイント数（ブレイクポイントシステムが使用する暗黙的なウォッチポイントを含む）が2を超えると、**[OK]** ボタンをクリックしたときにエラーメッセージが表示されます。このチェックはC-SPYの**[実行]** ボタンをクリックした場合も行われます。

このダイアログボックスは以下で使用できます。

- J-Link/J-Trace ドライバ
- Macraigor ドライバ

0x20-0xFF の範囲でアクセスをトリガするには、以下の手順を行います。

- 1 **【ブレーク条件】** を **【範囲】** に設定します。
- 2 ウォッチポイント 0 のアドレス値を 0 に設定し、0xFF にマスクします。
- 3 ウォッチポイント 1 のアドレス値を 0 に設定し、0x1F にマスクします。

アドレス

モニタするアドレスを指定します。

値	アドレスまたは評価の結果がアドレスになる C-SPY 式を指定します。また、以前に監視したアドレスをドロップダウンリストから選択することができます。C-SPY 式の詳細については、98 ページの C-SPY 式を参照してください
マスク	値の各ビットを制限します。マスクのビットがゼロである場合、値に対応するビットが比較のときに無視されます。どのアドレスとも一致させるには、0 を入力します。なお、マスクの値は ARM ハードウェアマニュアルで使用する表記法に基づいて変換されます
アドレスバスパターン	アドレスコンパレータで使用するビットパターンが表示されます。マスクで指定したように無視されたビットは x と表示されます

アクセスタイプ

モニタするデータのアクセスタイプを選択します。

全て	どのアクセスタイプにも一致
OP フェッチ	演算コード（命令）フェッチに一致
リード	指定された位置から読み取ります
ライト	指定の位置に書き込みます
R/W	指定された位置から読み取り / 書き込みを行います

データ

モニタするデータを指定します。サイズについては、以下から選択します。

任意のサイズ	あらゆるサイズのデータアクセスに一致
バイト	バイトサイズのアクセスに一致
ハーフワード	ハーフワードのサイズのアクセスに一致
ワード	ワードサイズのアクセスに一致

モニタする値を指定します。以下から選択します。

値	値または C-SPY 式を指定します。また、以前に監視した値をドロップダウンリストから選択することができます。 C-SPY 式の詳細については、98 ページの C-SPY 式を参照してください
マスク	値の各ビットを制限します。マスクのビットがゼロである場合、値に対応するビットが比較のときに無視されます。どのアドレスとも一致させるには、0 を入力します。なお、マスクの値は ARM ハードウェアマニュアルで使用する表記法に基づいて変換されます
データバスパターン	アドレスコンパレータで使用するビットパターンが表示されます。マスクで指定したように無視されたビットは x と表示されます

外部

外部入力の状態を定義します。以下から選択します。

全て	状態を無視します
0	状態を「低」として定義します
1	状態を「高」として定義します

モード

一致するためにアクティブでなければならない CPU モードを選択します。以下から選択します。

ユーザ	CPU モードの USER を選択します
非ユーザ	CPU モードの SYSTEM SVC 、 UND 、 ABORT 、 IRQ または FIQ を選択します
全て	CPU モードを無視します

ブレイク条件

定義されたウォッチポイントをどう使用するかを選択します。以下から選択します。

通常	2つのウォッチポイントを個別に使用します (OR)。
範囲	両方のウォッチポイントを組み合わせて対象の範囲とします。ウォッチポイント 0 で定義する開始点と、ウォッチポイント 1 で定義する終了点の範囲です。選択可能な範囲は 2 の累乗で制限されます。
チェーン	ウォッチポイント 1 がトリガされるとウォッチポイント 0 が準備されます。次にウォッチポイント 0 がトリガされると、プログラムが中止されます。

[ログ] ブレイクポイントダイアログボックス

[ログ] ブレイクポイントダイアログボックスは、エディタウィンドウ、[ブレイクポイント] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから開くことができます。

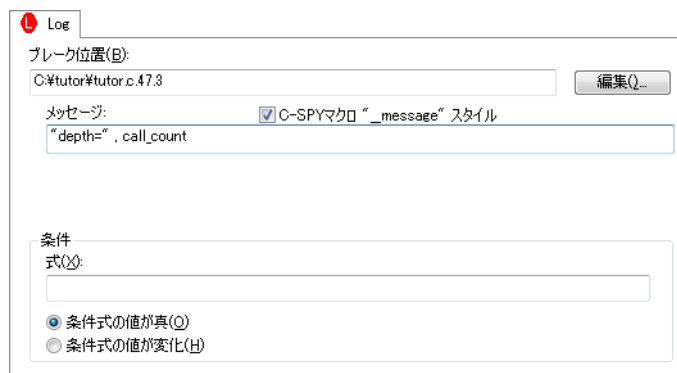


図 55: [ログ] ブレイクポイントダイアログボックス

[ログ] ブレイクポイントダイアログボックスを使用して、ログブレイクポイントを設定します。

注: [ログ] ブレイクポイントダイアログボックスは、使用する C-SPY ドライバによって異なります。この図は C-SPY シミュレータを示します。使用する C-SPY ドライバでのブレイクポイントのサポートについては、123 ページの C-SPY ハードウェアドライバのブレイクポイントを参照してください。

ブレイク位置

ブレイクポイントの位置を指定します。または、**[編集]** ボタンをクリックして **[位置入力]** ダイアログボックスを表示します (150 ページの **[位置入力]** ダイアログボックスを参照)。

メッセージ

[C-SPY デバッグログ] ウィンドウで表示するメッセージを指定します。通常のテキストか、コンマ区切りの引数リスト ([C-SPY マクロ **"__message" style**] オプションも選択している場合) を入力します。

C-SPY マクロ **"__message" style**

[メッセージ] テキストボックスで指定したコンマ区切りの引数リストを C-SPY マクロ言語文 **__message** の引数として使用する場合は、このオプションを選択します (289 ページの **フォーマットした出力**を参照)。

条件

単純または複雑な条件を指定します。

式	C-SPY 式の構文に準拠する有効な式を指定します。
条件が真	式の値が真の場合に、ブレイクポイントがトリガされます。
条件 (変更)	最後の評価時以降に式の値が変化した場合に、ブレイクポイントがトリガされます。

【データ】 ブレイクポイントダイアログボックス

【データ】 ブレイクポイントダイアログボックスは、エディタウィンドウ、[ブレイクポイント] ウィンドウ、[メモリ] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから開くことができます。

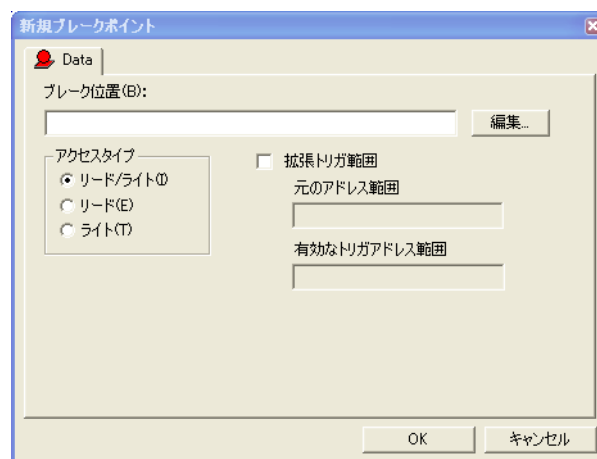


図 56: 【データブレイクポイント】 ダイアログボックス

【データ】 ブレイクポイントダイアログボックスを使用して、データブレイクポイントを設定します。データブレイクポイントによって単一命令内で実行が停止することはありません。ブレイクポイントは命令の実行後に記録、レポートされます。

注: 【データ】 ブレイクポイントダイアログボックスは、使用する C-SPY ドライバによって異なります。使用する C-SPY ドライバでのブレイクポイントのサポートについては、123 ページの C-SPY ハードウェアドライバのブレイクポイントを参照してください。

ブレイク位置

【ブレイク位置】 テキストボックスでブレイクポイントの位置を指定します。または、【編集】 ボタンをクリックして【位置入力】ダイアログボックスを表示します (150 ページの【位置入力】ダイアログボックスを参照)。

アクセスタイプ

データブレイクポイントをトリガするメモリアクセスの種類を選択します。

リード/ライト	指定された位置から読み取り / 書き込みを行います。
リード	指定された位置から読み取ります。
ライト	指定の位置に書き込みます。

トリガ範囲

要求された範囲とトレースでカバーする有効範囲が表示されます。推奨される範囲は、**「ブレイク位置」**と**「サイズ」** オプションによって指定された領域とまったく同じか、その内側です。

拡張して要求された範囲をカバー	データ構造体がカバーされるようにブレイクポイントを拡張します。ハードウェアブレイクポイント装置で提供できるブレイクポイント範囲のサイズと合わないデータ構造（たとえば3 バイト）の場合、ブレイクポイントの範囲はデータ構造全体を対象としません。ブレイクポイントの範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。
-----------------	---

データ照合

アクセスされるデータの照合を有効にします。**「データ照合」** オプションとデータのアクセスタイプを組み合わせて使用します。このオプションは、変数が特定の値を持つときにトリガが必要な場合に便利です。

値	データ値を指定します。
マスク	値のどの部分（ワード、ハーフワード、バイト）を照合するか指定します。

「データ照合」 オプションは、J-Link/J-Trace と ST-LINK でのみ使用可能です（ARM7/9 または Cortex-M デバイスを使用中のみ）。

注： Cortex-M デバイスについては、1 つのブレイクポイントにのみ **「データ照合」** を設定できます。このようなブレイクポイントでは、2 つのハードウェアブレイクポイントを使用します。

【データログ】 ブレイクポイントダイアログボックス

【データログ】 ブレイクポイントダイアログボックスは、エディタウィンドウ、[ブレイクポイント] ウィンドウ、[メモリ] ウィンドウ、[逆アセンブリ] ウィンドウのコンテキストメニューから開くことができます。

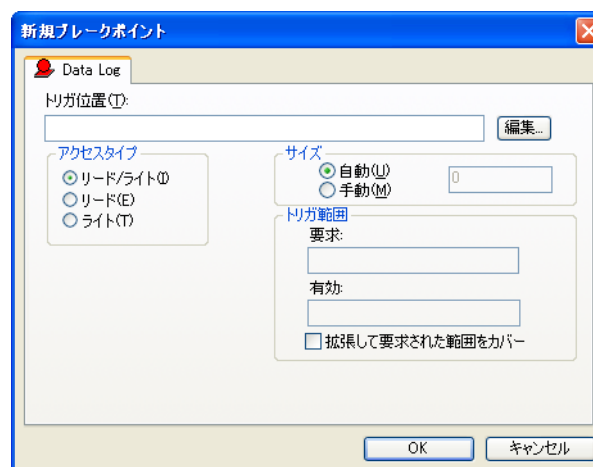


図 57: 【データログ】 ブレイクポイントダイアログボックス

データログブレイクポイントは、指定された位置のデータがアクセスされたときにトリガされます。特定のアドレスまたは範囲にログブレイクポイントを設定した場合は、その位置へのアクセスが発生するたびに、ログメッセージが [SWO トレース] ウィンドウに表示されます。ログメッセージは [データログ] ウィンドウにも表示されます（ウィンドウが有効な場合）。データログは [タイムライン] ウィンドウのデータロググラフにも表示することができます（そのウィンドウが有効な場合）。ログメッセージを使用するには、[SWO 設定] ダイアログボックスでトレースデータを設定しておく必要があります（190 ページの [SWO 設定] ダイアログボックスを参照）。

注：データログブレイクポイントの設定は、J-Link デバッグプロープまたは ST-LINK デバッグプロープを使用する SWO の Cortex-M に対してのみサポートされています。

トリガ位置

【トリガ位置】 テキストボックスでブレイクポイントの位置を指定します。または、[編集] ボタンをクリックして [位置入力] ダイアログボックスを表示します（150 ページの [位置入力] ダイアログボックスを参照）。

アクセスタイプ

データブレイクポイントをトリガするメモリアクセスの種類を選択します。

リード/ライト	指定された位置から読み取り / 書き込みを行います。
リード	指定位置からのリード。Cortex-M3 のレビジョン 2 のデバイスでのみ機能します。
ライト	指定位置でのライト。Cortex-M3 のレビジョン 2 のデバイスでのみ機能します。

サイズ

ブレイクポイントがトリガされる位置にサイズ（特に範囲）があるべきかを指定します。指定したメモリ範囲に対してフェッチアクセスが発生するごとに、ブレイクポイントがトリガされます。配列、構造体、共用体などのデータ構造へのアクセスによってデータブレイクポイントをトリガする必要がある場合に、この機能は便利です。サイズの指定方法は 2 つの中から選択します。

自動	ブレイクポイントが設定されている式のタイプに基づいて、サイズが自動的に決まります。これは、 〔トリガ位置〕 に変数が含まれる場合に便利です。
手動	テキストボックスでブレイクポイント範囲のサイズを指定します。

トリガ範囲

要求された範囲とトレースでカバーする有効範囲が表示されます。推奨される範囲は、**〔トリガ位置〕** と **〔サイズ〕** オプションによって指定された領域とまったく同じか、その内側です。

拡張して要求された範囲をカバー	データ構造体がカバーされるようにブレイクポイントを拡張します。ハードウェアブレイクポイント装置で提供できるブレイクポイント範囲のサイズと合わないデータ構造（たとえば 3 バイト）の場合、ブレイクポイントの範囲はデータ構造全体を対象としません。ブレイクポイントの範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。
-----------------	--

【ブレイクポイント】 オプション

【ブレイクポイント】 オプションページは、【オプション】 ダイアログボックスから使用できます。【プロジェクト】 > 【オプション】 を選択して、使用するデバッガシステムに固有のカテゴリを選び、【ブレイクポイント】 タブをクリックします。

The screenshot shows the 'Breakpoint' tab in a software configuration window. It contains three main sections: 'Default Breakpoint Type' with radio buttons for 'Automatic (A)', 'Hardware (H)', and 'Software (S)'; 'Software Breakpoint Reset Location (R)' with a text field containing 'main'; and 'Exceptions to Catch' with a grid of checkboxes for various system events like 'Reset (R)', 'Prefetch (P)', 'CoreReset', 'StatErr', 'Undefined (U)', 'IRQ (I)', 'MMERR', 'BusErr', 'SWT (S)', 'FIQ (F)', 'NCPERR', 'IntErr', 'Data (D)', and 'HRRERR'.

図 58: 【ブレイクポイント】 オプション

以下のハードウェアデバッガシステムでは、C-SPY を起動する前にドライバ固有のブレイクポイントオプションをいくつか設定できます。

- GDB サーバ
- J-Link/J-Trace JTAG プロローブ
- Macraigor JTAG プロローブ

デフォルトのブレイクポイントタイプ

ブレイクポイントの設定時に使用するブレイクポイントリソースの種類を選択します。以下から選択します。

自動

ソフトウェアブレイクポイントを使用します。不可能な場合は、ハードウェアブレイクポイントが使用されます。RAM のテストにはリード/ライトシーケンスを使用します。この場合は、ソフトウェアブレイクポイントが使用されます。【自動】 オプションは多くのアプリケーションに有効です。ただし、実行されたリード/ライトシーケンスによってフラッシュメモリが誤動作をする場合があります。この場合、【ハードウェア】 オプションを使用します。

ハードウェア	ハードウェアブレイクポイントを使用します。不可能な場合は、ブレイクポイントは設定されません。
ソフトウェア	ソフトウェアブレイクポイントを使用します。不可能な場合は、ブレイクポイントは設定されません。

ソフトウェアブレイクポイント復元位置

システム起動中に破壊されたブレイクポイントを自動的に復元します。

起動中に RAM にコピーしてから RAM で実行しているアプリケーションの場合に有効です。たとえば、リンカ構成ファイルのコードに `initialize by copy` リンカディレクティブを使用する場合、あるいはアプリケーションに `__ramfunc` 宣言関数がある場合に有効となることがあります。

この場合、C-SPY デバッガが起動すると、すべてのブレイクポイントは RAM のコピー中に破棄されます。C-SPY では、[ソフトウェアブレイクポイント復元位置] オプションを使用して、破棄されたブレイクポイントを復元します。

このテキストフィールドでは、C-SPY がブレイクポイントを復元する地点の、アプリケーションの位置を指定します。デフォルトの位置は、ラベル `-call_main` です。

例外の取得

ハードウェアのブレイクポイントを使用せずに、割込みベクタテーブルのベクタにブレイクポイントを直接設定します。このオプションは、ARM9、Cortex-R4、Cortex-M3 のデバイスで使用可能です。この設定は、プロジェクトのデフォルト設定として機能します。ただし、これらのデフォルト設定は、デバッグセッション中に [ベクタキャッチ] ダイアログボックスを使用してオーバーライドできます (125 ページの *例外ベクタのブレイクポイント* を参照)。

これらの設定は、デバッグセッション中は保持されます。

このオプションは、C-SPY J-Link/J-Trace ドライバのみでサポートされています。

【イミディエイトブレイクポイント】ダイアログボックス

【イミディエイト】ブレイクポイントダイアログボックスは、エディタウィンドウ、【ブレイクポイント】ウィンドウ、【メモリ】ウィンドウ、【逆アセンブリ】ウィンドウのコンテキストメニューから開くことができます。



図 59: 【イミディエイトブレイクポイント】ダイアログボックス

C-SPY シミュレータで、【イミディエイト】ブレイクポイントダイアログボックスを使用してイミディエイトブレイクポイントを設定します。イミディエイトブレイクポイントは一時的に命令の実行を停止するだけで、すぐに実行を再開します。

ブレイク位置

【ブレイク位置】テキストボックスでブレイクポイントの位置を指定します。または、【編集】ボタンをクリックして【位置入力】ダイアログボックスを表示します（150 ページの【位置入力】ダイアログボックスを参照）。

アクセスタイプ

イミディエイトブレイクポイントをトリガするメモリアクセスの種類を選択します。

リード	指定された位置から読み取ります。
ライト	指定の位置に書き込みます。

アクション

ブレイクポイントに関連するアクションがあるかどうかを決定します。C-SPY マクロ関数などの式を指定すると、ブレイクポイントのトリガ時に条件が真であるときに評価されます。

〔ベクタキャッチ〕 ダイアログボックス

〔ベクタキャッチ〕 ダイアログボックスは、J-Link/J-Trace と Macraigor の [J-Link] メニューから利用できます。

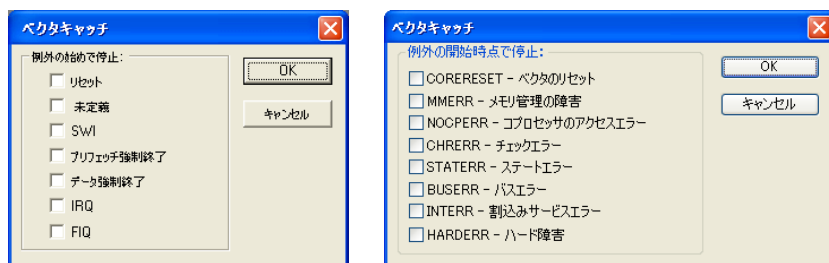


図 60: 〔ベクタキャッチ〕 ダイアログボックス (ARM9/Cortex-R4 と Cortex-M3 との比較)

このダイアログボックスを使用すると、ハードウェアのブレイクポイントを使用せずに、割り込みベクタテーブルのベクタにブレイクポイントを直接設定できます。ARM9、Cortex-R4、Cortex-M3 の各デバイスでベクタにブレイクポイントを設定できます。ここでの設定はデバッグセッションが終了した後は保持されません。

注：J-Link/J-Trace ドライバおよび RDI ドライバの場合、オプションダイアログボックスでベクタに直接ブレイクポイントを設定できます。詳細については、374 ページの *J-Link/J-Trace の設定オプション* および 385 ページの *RDI* を参照してください。

〔位置入力〕 ダイアログボックス

〔位置入力〕 ダイアログボックスは、新しいブレイクポイントを設定するかブレイクポイントを編集するときにブレイクポイントダイアログボックスから利用できます。

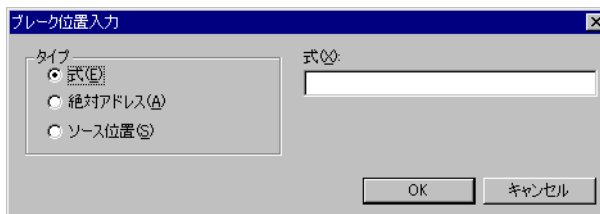


図 61: 〔位置入力〕 ダイアログボックス

[位置入力] ダイアログボックスを使用して、ブレイクポイントの位置を指定します。

注：このダイアログボックスは、選択する **[タイプ]** に応じて外観が変わります。

タイプ

ブレイクポイントで使用する位置のタイプを選択します。

式

C-SPY 式。 値は、関数や変数名など有効なアドレスに評価されます。

コードブレイクポイントは、たとえば `main` などの関数に設定されます。データブレイクポイントは、変数名に設定されます。たとえば、`my_var` は変数 `my_var` の位置を、`arr[3]` は配列 `arr` の 3 番目のエレメントの位置をそれぞれ参照します。

いくつかの関数で同じ名前で宣言された静的変数については、特定の変数を参照するために構文

`my_func::my_static_variable` を使用してください。

C-SPY 式の詳細については、98 ページの *C-SPY 式* を参照してください。

絶対アドレス

フォーム `zone:hexaddress` または単に `hexaddress` (`Memory:0x42` など) の絶対位置。 `zone` は **C-SPY** メモリゾーンを参照し、アドレスがどのメモリに属するかを指定します。

ソース位置

次の構文を使用して示した C ソースコード内の位置。
`{filename}.row.column.`

`filename` には、ファイル名およびフルパスを指定します。
`row` には、ブレイクポイントを設定する行を指定します。
`column` には、ブレイクポイントを設定する列を指定します。

たとえば、`{C:\src\prog.c}.22.3` は、ソースファイル `Utilities.c` の 22 行目の 3 文字目の位置にブレイクポイントを設定します。

[ソース位置] のタイプは、通常はコードのブレイクポイントだけに使用します。

【ソースの曖昧さの解決】 ダイアログボックス

【ソースの曖昧さの解決】 ダイアログボックスは、たとえばインライン関数やテンプレート上にブレイクポイントを設定しようとして、ソース位置が複数の関数に対応するような場合に表示されます。

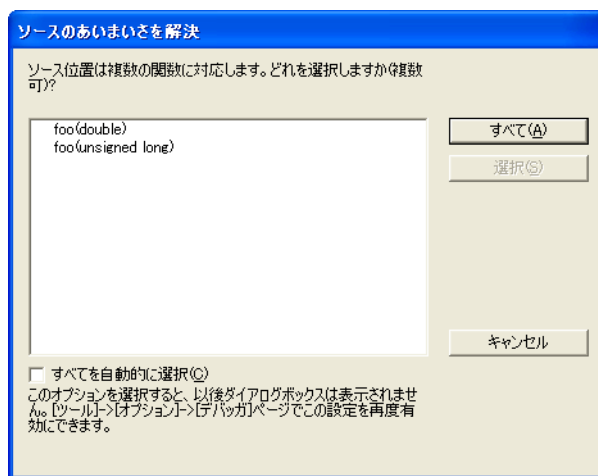


図 62: 【ソースの曖昧さの解決】 ダイアログボックス

ソースの曖昧さを解決するには、次のアクションのいずれかを実行します。

- テキストボックスで、リストされた位置を選択して（複数可）**【選択】**をクリックします。
- **【すべて】** をクリックします。

すべて

ブレイクポイントは、リストされたすべての位置に設定されます。

選択

ブレイクポイントは、テキストボックスで選択したソース位置に設定されます。

キャンセル

どの場所も使用されません。

自動的にすべて選択

指定されたソース位置が複数の関数と一致する場合にすべての場所を使用するかどうかを決定します。

このオプションは、**[IDE オプション]** ダイアログボックスでも指定できます (『*ARM 用 IDE プロジェクト管理およびビルドガイド*』のデバッガオプションを参照)。

メモリとレジスタのモニタ

この章では、メモリとレジスタを調査するために C-SPY® で使用できる機能の使用方法について説明します。具体的には以下の項目を解説します。

- メモリとレジスタのモニタの概要
- メモリとレジスタについてのリファレンス情報

メモリとレジスタのモニタの概要

このセクションでは、以下のトピックについて説明します。

- メモリとレジスタのモニタの概要について
- C-SPY メモリゾーン
- スタック表示
- メモリアクセスチェック

メモリとレジスタのモニタの概要について

C-SPY には、メモリとレジスタをモニタするウィンドウが多数あり、それらは個々に **[表示]** メニューから表示できます。

- **[メモリ]** ウィンドウ
指定メモリエリアであるメモリゾーンの最新状態を表示して、編集できます。様々な色を使用して、アプリケーションの実行に伴うデータカバレッジを示します。指定エリアに特定の値を設定して、メモリ位置と範囲に直接ブレークポイントを設定できます。このウィンドウで数個のインスタンスを開き、様々なメモリエリアをモニタできます。ウィンドウの内容は、アプリケーションの実行中に定期的に更新されます。
- **[シンボルメモリ]** ウィンドウ
静的記憶寿命変数がメモリ内でどのように配置されるかを表示します。これにより、メモリの使用が理解し易くなり、バッファオーバーランなど上書きされた変数に起因して発生した問題の調査に役立ちます。
- **[スタック]** ウィンドウ
メモリ内でのスタック変数の配置を含むスタック内容を表示します。また、スタックの整合性チェックを実行し、スタックオーバフローを検出してワーニングすることもできます。たとえば、**[スタック]** ウィンドウを

使用して、スタックの最適サイズを特定できます。このウィンドウのインスタンスを複数開いて、それぞれに異なるスタックを表示したり、同じスタックを異なる表示モードで表示したりできます。

- [レジスタ] ウィンドウ

プロセッサレジスタと SFR の内容の最新状態を表示し、それを編集できます。固定グループの CPU レジスタを除いて、追加のレジスタはデバイス記述ファイルで定義します。これらのレジスタとしては、ARM デバイスの周辺ユニットに対する、メモリにマッピングされたデバイス固有の制御レジスタとステータスレジスタがあります。

多くのレジスタがあるため、[レジスタ] ウィンドウに同時にすべてのレジスタを表示するのは不便です。その場合は、レジスタをレジスタグループに分割する方法があります。デバイス記述ファイルでは、デバイスの各周辺ユニットに 1 つのグループを定義します。[ツール] > [オプション] > [レジスタフィルタ] を選択すれば、独自のグループを定義することもできます。このウィンドウのインスタンスを複数開いて、それぞれに異なるレジスタグループを表示することができます。

特定の変数の内容を表示するには、単純にその変数を [メモリ] ウィンドウまたは [シンボルメモリ] ウィンドウにドラッグします。変数が配置されているメモリエリアが表示されます。

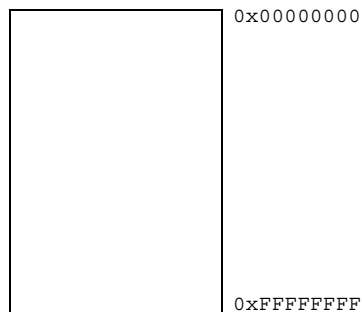


一部のレジスタの値を読み取ると、アプリケーションの実行時の動作に影響することがあります。たとえば、UART ステータスレジスタの値を読み取ると、保持ビットがリセットされて、受け取ったバイトを処理するはずの割込みがない状態になることがあります。こうならないようにするには、実行中のアプリケーションをデバッグする際には、このようなレジスタを含む [レジスタ] ウィンドウを必ず閉じてください。

C-SPY メモリゾーン

C-SPY では、ゾーンは名前付きメモリエリアを表します。メモリアドレス、またはメモリアドレス (ロケーション) は、ゾーンとそのゾーン内のオフ

セット値の組合せです。ARM アーキテクチャには、ARM メモリ範囲全体をカバーするメモリというゾーンが 1 つだけあります。



デフォルトゾーンメモリ

図 63: C-SPY のゾーン

メモリゾーンはさまざまなコンテキストで使用されますが、[メモリ] ウィンドウと [逆アセンブリ] ウィンドウで最も重要な役割を果たします。これらのウィンドウにある [ゾーン] ボックスでは、表示するメモリゾーンを選択します。

ゾーンには、Memory、Memory8、Memory16、Memory32、Memory64 があります。

一般的なメモリの場合、デフォルトゾーンメモリを使用できます。しかし、正しい結果を得るには、いくつかの I/O レジスタには 8、16、32 または 64 ビットとしてのアクセスが必要な場合があります。さまざまなメモリゾーンを使用することで、[メモリ] ウィンドウなどの読み込み/書き込みを使用するアクセス幅を制御することができます。

スタック表示

[スタック] ウィンドウにはグラフィカルスタックバーがあり、スタックの内容とオーバーフローのワーニングが表示されます。これらは多くの場面で役に立ちます。以下に例を示します。

- C モジュールからアセンブラモジュールを呼び出すか、その逆のときに、スタック使用量を調べる場合
- 適切なエレメントがスタック上に配置されているかどうかを調べる場合
- スタックが正しく復元されているかどうかを調べる場合
- 最適なスタックサイズの判定
- スタックオーバーフローの検出

複数のスタックを持つコアの場合は、表示するスタックを選択できます。

スタックの使用

アプリケーションを最初にロードするときや、リセットごとに、スタックエリアのメモリに 0xCD というバイト値が設定され、その後でアプリケーションの実行が開始されます。実行が停止すると、スタックの最後から、値が 0xCD でないバイトの位置（スタック中で使用された最も上の位置）までの範囲のスタックメモリが検索されます。これはスタック使用率のトレース方法としては信頼性の高い方法ですが、スタックオーバーフローが検出されるという保証はありません。たとえば、スタックが範囲を超えて誤って拡張され、スタック上限近辺のバイトは変更されることなく、スタックエリア外のメモリが変更される *可能性があります*。同様に、アプリケーションがスタックエリア内のメモリを誤って修正する可能性もあります。



さらに、[スタック] ウィンドウで検出できるのはスタックオーバーフローの痕跡だけで、スタックオーバーフローを発生時点で検出することはできません。ただし、グラフィカルスタックバーを有効にすると、スタックオーバーフローの検出とワーニングに必要な機能も有効になります。

注：スタックのサイズと場所は、リンカ構成ファイルで作成されるスタックを保持する section の定義から読み込まれます。何らかの理由で、システム起動コード (cstartup) によるスタック初期化を変更する場合は、その変更に応じて、リンカ構成ファイルの section 定義も変更する必要があります。これを行わないと、[スタック] ウィンドウでスタック使用率を追跡できません。この詳細については、*ARM 用 IAR C/C++ 開発ガイド*を参照してください。

メモリアクセスチェック

C-SPY シミュレータはターゲットハードウェアの様々なメモリアクセスタイプをシミュレーションして、ライト専用メモリにリードアクセスするなどの不正なアクセスを検出します。特定のメモリエリアに対して指定されたアクセスタイプに従わないメモリアクセスが発生した場合、C-SPY はそれを不正なアクセスと認識します。また、定義されていないメモリへのメモリアクセスは、不正なアクセスと見なされます。メモリアクセスチェック機能によって、ユーザはメモリアクセス違反を特定しやすくなります。

メモリエリアは、デバイス記述ファイルで定義済みのゾーンか、デバッグファイルの section 情報に基づいています。これら以外に、ユーザが独自のメモリエリアを定義できます。アクセスタイプには、読み取り、書き込み、読み取りのみ、書き込みのみがあります。2つの異なるアクセスタイプを同一のメモリエリアに割り当てることはできません。アクセスタイプの違反および未指定の範囲へのアクセスをチェックすることができます。違反が検出された場合は、[デバッグログ] ウィンドウにロギングされます。実行を停止するかどうかを選択することもできます。

メモリとレジスタについてのリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 159 ページの [メモリ] ウィンドウ
- 163 ページの [メモリ保存] ダイアログボックス
- 164 ページの [メモリ復元] ダイアログボックス
- 164 ページの [フィル] ダイアログボックス
- 166 ページの [シンボルメモリ] ウィンドウ
- 168 ページの [スタック] ウィンドウ
- 171 ページの [レジスタ] ウィンドウ
- 173 ページの [メモリアクセス設定] ダイアログボックス
- 175 ページの [メモリアクセスの編集] ダイアログボックス

[メモリ] ウィンドウ

[メモリ] ウィンドウは [表示] メニューから利用できます。

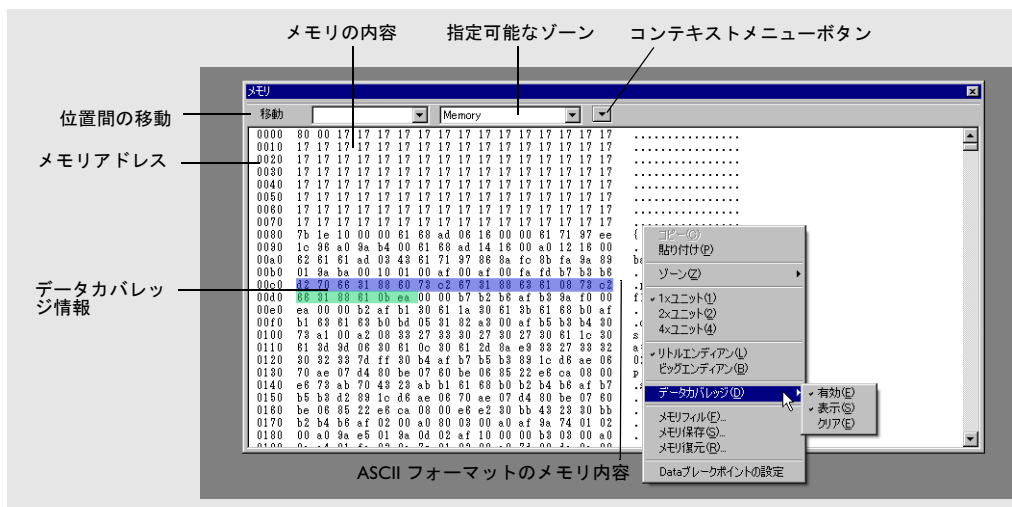


図 64: [メモリ] ウィンドウ

このウィンドウでは、指定メモリエリアであるメモリゾーンの最新状態を表示して、編集できます。このウィンドウは複数表示でき、メモリやレジスタの複数のゾーンをトレースする場合や、メモリのさまざまな部分をモニタする場合に非常に便利です。



変数に対応するメモリを表示するには、エディタウィンドウでその変数を選択し、[メモリ] ウィンドウにドラッグします。

ツールバー

ツールバーの内容は以下のとおりです。

移動	表示するロケーションを指定できます。それには、メモリアドレス、または変数、機能、ラベルの名前を指定できます。
ゾーン表示	表示するメモリゾーンを選択します (156 ページの <i>C-SPY</i> メモリゾーンを参照)。
コンテキストメニューボタン	コンテキストメニューを表示するには、「161 ページの コンテキストメニュー」を参照してください。
今すぐ更新	アプリケーションの実行中に [メモリ] ウィンドウの内容を更新します。このボタンは、使用している C-SPY ドライバがアプリケーションの実行中にターゲットのシステムメモリにアクセス可能な場合のみ有効化されます。
ライブ更新	アプリケーションの実行中に [メモリ] ウィンドウの内容を定期的に更新します。このボタンは、使用している C-SPY ドライバがアプリケーションの実行中にターゲットのシステムメモリにアクセス可能な場合のみ有効化されます。更新頻度を設定するには、 [IDE オプション] > [デバッグ] ダイアログボックスに適切な頻度を指定します。

表示エリア

表示エリアには現在表示しているアドレスとメモリの内容が選択したフォーマットで表示されるほか、表示モードが **1x ユニット** に設定されていれば、メモリの内容が **ASCII** フォーマットで表示されます。表示エリアの内容は、16 進表示と **ASCII** 表示のどちらの部分でも編集できます。

データカバレッジは、以下の色で表示されます。

黄色	データのリードが行われたことを示します。
青	データのライトが実行されたことを示します。
緑	データのリードとライトの両方が実行されたことを示します。

注：一部の C-SPY ドライバは、データカバレッジをサポートしていません。
C-SPY シミュレータは、データカバレッジをサポートしています。

コンテキストメニュー

以下のコンテキストメニューがあります。

コピー(C)
貼り付け(P)
ゾーン(Z) ▶
✓ 1xユニット(U)
2xユニット(O)
4xユニット(Q)
✓ リトルエンディアン(L)
ビッグエンディアン(B)
データカバレッジ(D) ▶
検索(F)...
置換(R)...
メモリフィル(F)...
メモリ保存(S)...
メモリストア(T)...
データブレイクポイントの設定

図 65: [メモリ] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

コピー、貼り付け	標準の編集コマンド
ゾーン	表示するメモリゾーンを選択します (156 ページの C-SPY メモリゾーンを参照)。
1x ユニット	メモリの内容を 8 ビット単位で表示します。
2x ユニット	メモリの内容を 16 ビット単位で表示します。
4x ユニット	メモリの内容を 32 ビット単位で表示します。
リトルエンディアン	リトルエンディアンのバイトオーダで内容を表示します。
ビッグエンディアン	ビッグエンディアンのバイトオーダで内容を表示します。

データカバレッジ 以下から選択します。

[有効化] は、データカバレッジの有効 / 無効を切り替えます。

[表示] は、データカバレッジの表示 / 非表示を切り替えます。

[クリア] は、すべてのデータカバレッジ情報を消去します。

これらのコマンドは、C-SPY ドライバがデータカバレッジをサポートする場合のみ使用できます。

検索 [メモリ] ウィンドウ内でテキストを検索するダイアログボックスを表示します。**[検索]** ダイアログボックスについて詳しくは、『*ARM 用 IDE プロジェクト管理およびビルドガイド*』を参照してください。

置換 指定した文字列を検索して、該当する項目を別の文字列に置換するダイアログボックスを表示します。**[置換]** ダイアログボックスについては、『*ARM 用 IDE プロジェクト管理およびビルドガイド*』を参照してください。

メモリフィル 指定エリアに値を設定できるダイアログボックスを開きます (164 ページの **[フィル]** ダイアログボックスを参照)。

メモリ保存 特定のメモリエリアの内容をファイルに保存できるダイアログボックスを表示します (163 ページの **[メモリ保存]** ダイアログボックスを参照)。

メモリ復元 Intex-hex や Motorola s-record フォーマットでファイルの内容を指定したメモリゾーンにロードできるダイアログボックスを表示します (164 ページの **[メモリ復元]** ダイアログボックスを参照)。

データブレイクポイントの設定 [メモリ] ウィンドウでブレイクポイントを直接設定します。ブレイクポイントが強調表示されていない場合は、**[ブレイクポイント]** ダイアログボックスでブレイクポイントを表示、編集、削除することができます。このウィンドウで設定したブレイクポイントは、リードとライトの両方のアクセスでトリガされます。詳細については、128 ページの **[メモリ] ウィンドウでのデータブレイクポイントの設定**を参照してください。

「メモリ保存」ダイアログボックス

「メモリ保存」ダイアログボックスは、「デバッグ」>「メモリ」>「保存」を選択するか、「メモリ」ウィンドウのコンテキストメニューから使用できます。

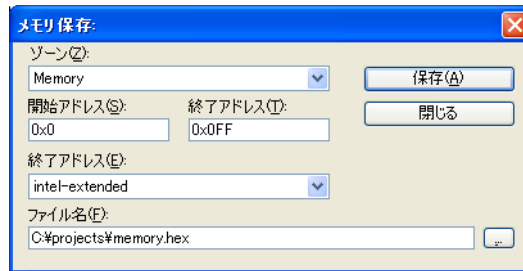


図 66: 「メモリ保存」ダイアログボックス

このダイアログボックスを使用して、指定したメモリエリアの内容をファイルに保存します。

ゾーン

メモリゾーンを選択します。

開始アドレス

保存するメモリ範囲の開始アドレスを指定します。

終了アドレス

保存するメモリ範囲の停止アドレスを指定します。

ファイルフォーマット

使用するファイルフォーマットを選択します。デフォルトでは Intel-extended です。

ファイル名

使用する対象ファイルを指定します。参照ボタンを使用して選択すると便利です。

保存

選択したメモリゾーン範囲を指定ファイルに保存します。

【メモリ復元】 ダイアログボックス

【メモリ復元】 ダイアログボックスは、[デバッグ] > [メモリ] > [復元] を選択するか、[メモリ] ウィンドウのコンテキストメニューから使用できます。

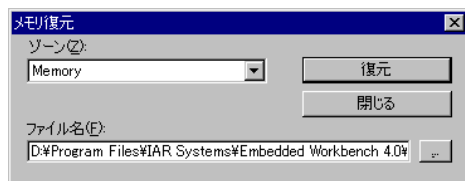


図 67: 【メモリ復元】 ダイアログボックス

このダイアログボックスを使用して、特定のメモリゾーンにファイルの内容を Intel-extended または Motorola s-record フォーマットでロードします。

ゾーン

メモリゾーンを選択します。

ファイル名

リード対象ファイルを指定します。参照ボタンを使用して選択すると便利です。

元に戻す

指定ファイルの内容を選択したメモリゾーンにロードします。

【フィル】 ダイアログボックス

【フィル】 ダイアログボックスは、[メモリ] ウィンドウのコンテキストメニューから使用できます。



図 68: 【フィル】 ダイアログボックス

このダイアログボックスを使用して、指定したメモリエリアに値を設定できます。

開始アドレス

2進数、8進数、10進数、16進数のいずれかの表記法で開始アドレスを入力します。

長さ

2進数、8進数、10進数、16進数のいずれかの表記法でデータ長を入力します。

ゾーン

メモリゾーンを選択します。

値

各メモリアドレス（ロケーション）に設定する8ビット値を入力します。

操作

以下のメモリ操作を使用できます。

コピー

【値】に入力した値が指定したメモリエリアにコピーされます。

AND

【AND】を指定すると、【値】の値とメモリの既存値の論理積がメモリに書き込まれます。

XOR

【XOR】を指定すると、【値】の値とメモリの既存値の論理積がメモリに書き込まれます。

OR

【OR】を指定すると、【値】の値とメモリの既存値の論理積がメモリに書き込まれます。

【シンボルメモリ】 ウィンドウ

【シンボルメモリ】 ウィンドウは、デバッガの実行中に **【表示】** メニューから使用できます。

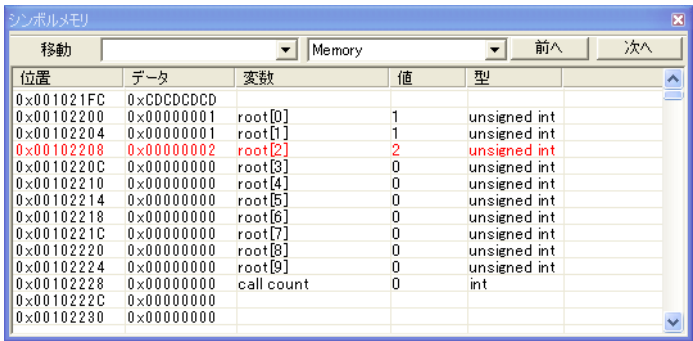


図 69: 【シンボルメモリ】 ウィンドウ

このウィンドウには、静的記憶寿命を持つ変数や、通常はファイルスコープのほかに静的変数を関数およびクラスに持つ変数が、メモリ内でどのように配置されるかが表示されます。これにより、メモリの使用が理解し易くなり、バッファオーバーランなど上書きされた変数に起因して発生した問題の調査に役立ちます。アラインメントの穴の検出や上書きしたバッファに起因する問題の理解にも役立ちます。



変数に対応するメモリを表示するには、エディタウィンドウでその変数を選択し、【シンボルメモリ】 ウィンドウにドラッグします。

ツールバー

ツールバーの内容は以下のとおりです。

- 移動** 表示するメモリアドレス（ロケーション）またはシンボルを指定できます。
- ゾーン表示** 表示するメモリゾーンを選択します（156 ページの *C-SPY* メモリゾーンを参照）。
- 前へ** 表示エリアで前のシンボルを強調表示します。
- 次へ** 表示エリアで次のシンボルを強調表示します。

表示エリア

このエリアには以下の列が含まれます。

位置	メモリアドレス。
データ	16 進フォーマットのメモリ内容。データはシンボルサイズに従ってグループ化されます。この列は編集できます。
変数	変数名。変数には固定されたメモリ位置が必要です。ローカル変数は表示されません。
値	変数の値。この列は編集できます。
型	変数のタイプ。

いくつかの方法でメモリ空間内を移動できます。

- ウィンドウでドロップされるテキストがシンボルと解釈されます。
- ウィンドウ右側のスクロールバー
- ツールバーボタン **〔次へ〕** と **〔前のエラー〕**
- ツールバーリストボックス **〔移動〕** は、特定の場所やシンボルの検出に使用できます

注： 対応する値を変更すると行に赤色のマークが付けられます。

コンテキストメニュー

以下のコンテキストメニューがあります。

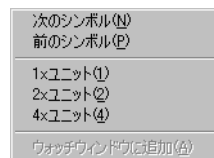


図 70: [シンボルメモリ] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

次のシンボル	表示エリアで次のシンボルを強調表示します。
前のシンボル	表示エリアで前のシンボルを強調表示します。
1x ユニット	メモリの内容を 8 ビット単位で表示します。これは変数を含まない行にのみ適用します。
2x ユニット	メモリの内容を 16 ビット単位で表示します。

メモリの内容を 32 ビット単位で表示します。

選択したシンボルを [ウォッチ] ウィンドウに追加します。

「スタック」ウィンドウは「表示」メニューから利用できます。



このウィンドウには、スタックの内容が表示されます。また、スタックの整合性チェックを実行し、スタックオーバフローを検出してワーニングすることもできます。たとえば、[スタック] ウィンドウを使用して、スタックの最適サイズを特定できます。

[スタック] ウィンドウでは、リンカ設定ファイルに実施されたスタックを保持する **section** の定義からスタックのサイズと場所に関する情報を取得します。このセクションは『**ARM 用 IAR C/C++ 開発ガイド**』に記述されています。

スタックを他の方法で設定するアプリケーションに対しては、デフォルトの方法をオーバーライドできます。**C-SPY** コマンドラインの派生オプションのいずれかを使用します (357 ページの `--proc stack stack` を参照)。

グラフィカルスタックバーを表示するには、以下の手順に従います。

- 1 [ツール] > [オプション] > [スタック] を選択します。
- 2 オプション [グラフィカルスタック表示とスタック使用トラッキングを有効にする] を選択します。

複数の[スタック]ウィンドウを表示し、それぞれ異なるスタックを表示する(スタックが複数ある場合)か、同一のスタックを異なる表示設定で、表示することができます。

注: デフォルトでは、このウィンドウは物理的ブレークポイントを1つ使用します。詳細については、124 ページの *ブレークポイントの設定元* を参照してください。

[スタック] ウィンドウに固有の情報については、『*ARM 用 IDE プロジェクト管理およびビルドガイド*』を参照してください。

ツールバー

スタック 表示するスタックを選択します。これは複数のスタックを持つコアに適用されます。

グラフィカルスタックバー

スタックの状態をグラフィックを使用して表示します。

スタックバーの左端はスタックの底、つまりスタックが空白のときのスタックポインタの位置を示します。右端は、スタック用に予約されているメモリエリアの最後を示します。スタック使用率が指定のしきい値を超えると、スタックバーは赤色に変化します。

スタックバーを有効にすると、スタックオーバフローの検出とワーニングに必要な機能も有効になります。



マウスポインタをスタックバーの上に移動すると、スタック使用量に関するツールチップ情報が表示されます。

表示エリア

このエリアには以下の列が含まれます。

位置	メモリでの位置を示します。アドレスは小さい方から順に表示されます。スタックポインタが参照するアドレス、つまりスタック最上部は、緑色で強調表示されます。
データ	その位置のメモリユニットの内容を表示します。[スタック] ウィンドウのコンテキストメニューで、データの表示方法 (1 バイト、2 バイト、4 バイトのいずれかの単位) を選択できます。
変数	その位置にローカル変数がある場合に、その変数名を表示します。変数は、関数内でローカルに宣言されていて、レジスタではなくスタックにある場合にだけ表示されます。

値	[変数] 列に表示されている変数の値を示します。
フレーム	呼出しフレームが対応する関数の名前を示します。

コンテキストメニュー

以下のコンテキストメニューがあります。

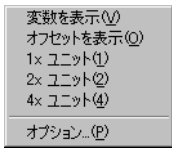


図 72: [スタック] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

変数を表示	変数、値、フレームという別々の列を [スタック] ウィンドウに表示します。[スタック] ウィンドウで表示されるメモリ位置にある変数が、これらの列に表示されます。
オフセットを表示	[位置] 列にスタックポインタからのオフセットとして場所を表示します。選択を解除すると、位置は絶対アドレスとして表示されます。
1x ユニット	[データ] 列に 1 バイトとしてデータを表示します。
2x ユニット	[データ] 列に 2 バイトグループとしてデータを表示します。
4x ユニット	[データ] 列に 4 バイトグループとしてデータを表示します。
オプション	[IDE オプション] ダイアログボックスを表示します。このダイアログボックスで、[スタック] ウィンドウ専用オプションを設定できます (『ARM 用 IDE プロジェクト管理およびビルドガイド』を参照)。

「レジスタ」ウィンドウ

「レジスタ」ウィンドウは「表示」メニューから利用できます。

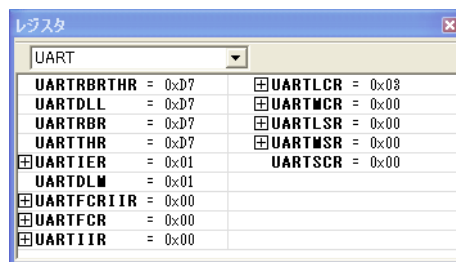


図 73: 「レジスタ」ウィンドウ

このウィンドウにはプロセッサレジスタと特殊機能レジスタ (SFR) の最新状態が表示されており、これらの内容の編集が可能です。オプションで、定義済みのレジスタグループをロードするか、アプリケーションに固有な独自のグループを定義することもできます。

このウィンドウは複数表示でき、複数のレジスタグループをトレースするのに非常に便利です。

定義済レジスタグループを有効にするには、次の手順にしたがいます。

- 1 デバイスに適合するデバイス記述ファイルを選択します (59 ページの *デバイス記述ファイルの選択* を参照)。
- 2 デバイス記述ファイルで定義されていれば、レジスタグループは「レジスタ」ウィンドウに表示されます。使用可能なレジスタグループは、「レジスタフィルタ」ページにも一覧表示されます。

アプリケーション固有のレジスタグループを定義するには、『*ARM 用 IDE プロジェクト管理およびビルドガイド*』のレジスタフィルタのオプションの項を参照してください。

ツールバー

ドロップダウンリスト 表示するレジスタグループを選択します。デフォルトでは、デバッグには次の2つのレジスタグループがあります。

【現在の CPU レジスタ】には、現在のプロセッサモードで使用可能なレジスタが含まれます。

【CPU レジスタ】には、現在のレジスタと、他のプロセッサモードで使用可能なバンクレジスタの両方が含まれます。

追加のレジスタグループは、デバイス記述ファイル (arm¥config ディレクトリ) に定義されます。これらによって、すべての SFR レジスタが [レジスタ] ウィンドウで使用可能になります。デバイス記述ファイルには、特殊機能レジスタとそのグループを定義するセクションがあります。

表示エリア

レジスタとその値を表示します。C-SPY が停止するたびに、前回停止したときから変更された値が強調表示されます。レジスタの内容を編集するには、レジスタをクリックして、値を変更します。

一部のレジスタは展開可能です。つまり、関連するビットやビットのサブグループが含まれています。

表示フォーマットを変更するには、[レジスタフィルタ] ページで [ベース] 設定を変更します。このページにアクセスするには、[ツール] > [オプション] を選択します。

〔メモリアクセス設定〕ダイアログボックス

〔メモリアクセス設定〕ダイアログボックスは、〔シミュレータ〕メニューから使用できます。



図 74: 〔メモリアクセス設定〕ダイアログボックス

このダイアログボックスには定義済みのすべてのメモリエリアが一覧表示され、リストのそれぞれの列ではエリアのプロパティが指定されます。つまり、このダイアログボックスには、シミュレーション中に使用されるメモリアクセス設定が表示されます。

注: 〔使用範囲の設定基準ファイル〕オプションと 〔使用範囲を手動で設定〕オプションを両方とも有効にすると、定義されているすべての範囲へのメモリアクセスがチェックされます。

表示される列とプロパティについては、175 ページの 〔メモリアクセスの編集〕ダイアログボックスを参照してください。

使用範囲の設定基準ファイル

定義済メモリアクセス設定を選択します。以下から選択します。

デバイス記述ファ デバイス記述ファイルからプロパティをロードします。
イル

デバッグファイルセグメント情報 プロパティは、デバッグファイルで使用する section 情報に基づいています。この情報は、デバッグ中のみ使用できます。このオプションの利点は、シミュレータが、リンクされているアプリケーション以外からのメモリアクセスをキャプチャできることです。

使用範囲を手動で設定

［メモリアクセスの編集］ ダイアログボックスで独自の範囲を手動で指定します。このダイアログボックスを開くには、**［新規作成］** を選択して新しいメモリ範囲を指定するか、メモリゾーンを選択してから **［編集］** を選択してそのメモリゾーンを変更します。詳細については、175 ページの **［メモリアクセスの編集］** ダイアログボックスを参照してください。

手動で定義した範囲は、デバッグセッション終了後も保持されます。

メモリアクセスチェック

［チェック対象］ では、チェックの対象を指定します。

- アクセスタイプ違反
- 指定範囲外へのアクセス

［アクション］ では、アクセス違反が発生したときに実行するアクションを以下から選択します。

- 違反のログ
- ログと実行停止

違反が検出された場合は、**［デバッグログ］** ウィンドウにロギングされます。

ボタン

以下のボタンを選択できます。

- | | |
|-------------|--|
| 新規 | ［メモリアクセスの編集］ ダイアログボックスを開きます。新しいメモリ範囲を指定したり、アクセスタイプを割り当てたりすることができます (175 ページの ［メモリアクセスの編集］ ダイアログボックスを参照)。 |
| 編集 | ［メモリアクセスの編集］ ダイアログボックスを開きます。選択されたメモリエリアを編集できます。175 ページの ［メモリアクセスの編集］ ダイアログボックスを参照してください。 |
| 削除 | 選択されたメモリエリア定義を削除します。 |
| 全て削除 | 定義されているすべてのメモリエリア定義を削除します。 |

注：[OK] ボタンと [キャンセル] ボタン以外のボタンは、[使用範囲を手動で設定] オプションが選択されているときだけ使用できます。

[メモリアクセスの編集] ダイアログボックス

[メモリアクセスの編集] ダイアログボックスは、[メモリアクセスの設定] ダイアログボックスから使用できます。



図75: [メモリアクセスの編集] ダイアログボックス

このダイアログボックスを使用してメモリ範囲を指定し、各メモリ範囲にアクセスタイプを割り当てます、これに対して、シミュレーション中に不正なアクセスを検出します。

メモリ範囲

メモリアクセスをチェックするメモリエリアを定義します。

ゾーン	メモリゾーンを選択します (156 ページの <i>C-SPY</i> メモリゾーンを参照)。
開始アドレス	アドレス範囲の開始アドレスを 16 進表記で指定します。
終了アドレス	アドレス範囲の終了アドレスを 16 進表記で指定します。

アクセスタイプ

メモリ範囲へのアクセスタイプを以下から選択します。

- リード と ライト
- リード オンリー
- ライト のみ

トレースデータの収集と使用

この章では、C-SPY® でのトレースデータ収集と使用について説明します。具体的には以下の項目を解説します。

- トレースの使用の概要
- トレースの使用の手順
- トレースのリファレンス情報

トレースの使用の概要

このセクションではトレースの概要を説明します。

以下のトピックを解説します。

- トレースを使用する理由
- トレースの概要
- トレースを使用するための条件

以下も参照してください。

- 255 ページの *割込み*
- 223 ページの *プロファイラの使用*

トレースを使用する理由

トレースを使用することで、特定の状態（アプリケーションのクラッシュなど）になるまでのプログラムの流れを調べたり、トレースデータを使用して問題の原因を特定したりすることができます。トレースデータは、不規則な症状が散発的に発生するようなプログラミングエラーを特定する際に役立ちます。

トレーストリガとトレースフィルタを使用する理由

トレーストリガとトレースフィルタ条件を使用することで、ソースコードの関心を持つ部分を選択し、J-Trace プローブのトレースバッファをより効率的に使用できます。トレーストリガ（トレース開始ブレークポイントとトレース停止ブレークポイント）は、たとえばトレースデータを収集する対象のコードセクションを指定します。トレースフィルタは、実行中に満たされたときにトレースデータの収集を有効にする条件を指定します。

ARM7/9 デバイスの場合、合計で最大 16 のトレーストリガおよびトレースフィルタを指定でき、うち 8 つはトレースフィルタとすることができます。

Cortex-M デバイスの場合、合計で最大 4 つのトレーストリガおよびトレースフィルタを指定できます。

トレースの概要

ターゲットシステムは、トレースデータを生成できる必要があります。データが生成されると、C-SPY でそれを収集し、さまざまなウィンドウやダイアログボックスでそのデータを視覚化および分析することができます。

C-SPY は、以下のターゲットシステムからのトレースデータの収集をサポートしています。

- ETM (Embedded Trace Macrocell) をサポートするデバイス — ETM トレース
- SWO (Serial Wire Output) 通信チャンネルを使用してSWD (Serial Wire Debug) インタフェースをサポートするデバイス — SWO トレース
- C-SPY シミュレータ

使用しているターゲットシステムに応じて、異なるタイプのトレースデータが生成できます。

ETM トレース

ETM トレース（別名フルトレース）は、選択された実行の一部について実行されたすべての命令を連続して収集したシーケンスです。バッファが保持できるだけのデータしか収集できません。

デバッグプローブには、トレースデータをリアルタイムで収集するトレースバッファが含まれていますが、データは実行が停止するまで C-SPY のウィンドウに表示されません。

SWO トレース

SWO トレースは、オンチップのデバッグハードウェアによって生成されるさまざまな種類のイベントのシーケンスです。イベントは SWO 通信チャンネルを介して、ターゲットシステムからリアルタイムで送信されます。つまり、C-SPY のウィンドウはターゲットシステムの実行中に連続して更新されます。最も重要なイベントは以下のとおりです。

- PC サンプルリング

ハードウェアは、プログラムカウンタの値を一定の間隔でサンプリングおよび送信することができます。これは（ETM トレースのような）実行済み命令の連続シーケンスではなく、PC の散発的かつ定期的なサンプリングです。現在の ARM CPU は通常、毎秒数百万の命令を実行しますが、PC サンプリングレートは一般的に毎秒数千の単位でカウントされます。

- 割込みログ

ハードウェアは割込みの実行に関連するデータを生成して送信できます。割込みハンドラルーチンの投入および終了時にイベントが生成されます。

- データログ

データログブレイクポイントを使用して、ある変数または単にアドレス範囲が CPU によってアクセスされたときにイベントを生成して送信するようハードウェアを設定できます。

SWO チャンネルの処理量には制限があります。よって、PC サンプリング、割込み、指定した変数へのアクセスの頻度のいずれかが高い場合、通常は上記のすべての機能を同時に使用することはできません。

C-SPY のトレース機能

C-SPY では、トレース関連のウィンドウとして [トレース]、[関数トレース]、[タイムライン]、[トレースを検索] が使用できます。C-SPY シミュレータでは、[トレース式] ウィンドウも使用できます。使用する C-SPY ドライバに応じて、さまざまなタイプのトレースブレイクポイントやトリガを設定してトレースデータの収集を制御することができます。

C-SPY J-Link/J-Trace ドライバまたは ST-LINK ドライバを使用する場合、[割込みログ]、[ログサマリの割込み]、[データログ]、[データログサマリ] などのウィンドウを使用できます。



デバッグ時には、ETM および SWO という 2 つのボタンが IDE のメインウィンドウに表示されます。これらのボタンのどれかが緑の場合、対応するトレースハードウェアがトレースデータを生成中であることを意味します。C-SPY のどの機能がトレースデータの生成を要求しているのかについて詳細なツールチップ情報を得るには、マウスのポインタをボタンにあわせてください。この情報は、多くの C-SPY 機能が現在トレースデータを使用しているために SWO 通信チャンネルがよくオーバーフローする場合などに便利です。ボタンをクリックすると、対応する設定ダイアログボックスが開きます。

また、プロファイラやコードカバレッジ、命令プロファイリングなど、C-SPY の他のいくつかの機能でもトレースデータを使用します。

トレースを使用するための条件

C-SPY のシミュレータはトレース関連の機能をサポートしており、特定の要件はありません。

ハードウェアデバッグシステムでトレース関連の機能を使用するには、トレースをサポートするデバッグコンポーネント（ハードウェア、デバッグプローブ、C-SPY ドライバ）が必要です。

注：使用するデバッグコンポーネントの特定のセットによって、サポートされる C-SPY のトレース機能が決まります。

ETM トレースを使用するための条件

ETM トレースは一部の ARM デバイスで使用可能です。

ETM トレースを使用するには、以下の組合せのいずれかが必要です。

- J-Trace デバッグプローブおよび ETM をサポートするデバイス。必ず C-SPY J-Link/J-Trace ドライバを使用してください。
- J-Link デバッグプローブおよび ETB (Embedded Trace Buffer) によって ETM をサポートするデバイス。J-Link プローブは、ETB バッファから ETM データを読み込みます。必ず C-SPY J-Link/J-Trace ドライバを使用してください。

SWO トレースを使用するための条件

SWO トレースを使用するには、SWO 通信チャンネルをサポートする J-Link または J-Trace または ST-LINK デバッグプローブ、および SWD/SWO インタフェースをサポートするデバイスが必要です。

トレーストリガとトレースフィルタを使用する要件

トレーストリガとトレースフィルタ機能は J-Trace だけに対応しており、ARM7/9 または Cortex-M デバイスの使用時のみ利用できます。

トレースの使用の手順

このセクションでは、トレースデータの収集と使用の方法についてステップごとに説明します。

具体的には、以下の項目について説明します。

- C-SPY シミュレータでトレースを開始するには
- ETM トレースを開始するには
- SWO トレースを開始するには
- ETM および SWO の並列使用の設定
- ブレークポイントを使用したトレースデータの収集
- トレースデータの検索
- トレースデータの参照

C-SPY シミュレータでトレースを開始するには

C-SPY シミュレータを使用してトレースデータを収集するには、特定のビルド設定は必要がありません。

トレースを開始するには：



- 1 アプリケーションをビルドして C-SPY を起動したら、[シミュレータ] > [トレース] を選択して [トレース] ウィンドウを開き、[有効化] ボタンをクリックしてトレースデータの収集を有効にします。
- 2 実行を開始します。ブレークポイントがトリガされたなどの理由で実行が停止したときは、[トレース] ウィンドウにトレースデータが表示されます。ウィンドウの情報については、「193 ページの [トレース] ウィンドウ」を参照してください。

ETM トレースを開始するには

ETM トレースを開始するには：

- 1 C-SPY を起動する前に：
 - J-Trace の場合、C-SPY を起動する上で特定の設定は必要ありません。
 - デバイスにトレースポートを設定する必要があります。一部のデバイスでは、これはトレースロジックを有効にすると自動的に行われます。ただし、ARM 7 または ARM 9 に基づいた Atmel および ST デバイスなど一部のデバイスでは、トレースポートを明示的に設定する必要があります。これは、C-SPY マクロファイルを介して行います。こうしたファイル (ETM_init*.mac) の例は、サンプルプロジェクトにあります。マクロファイルを使用するには、[プロジェクト] > [オプション] > [デバッグ] > [設定] > [マクロファイルの使用] を選択します。マクロファイルを指定します。参照ボタンを使用すると便利です。

ハードウェアでトレースシグナルに使用されたピンは、アプリケーションでは使用できません。
- 2 C-SPY を起動したら、C-SPY ドライバのメニューから [トレース設定] を選択します。表示される [トレース設定] ダイアログボックスで、デフォルト設定を変更する必要があるか確認します。詳細については、186 ページの [ETM トレース設定] ダイアログボックスを参照してください。
- 3 [トレース] ウィンドウ（ドライバ固有のメニューからアクセス）を開き、[有効化] ボタンをクリックしてトレースデータの収集を有効にします。
- 4 実行を開始します。ブレークポイントがトリガされたなどの理由で実行が停止したときは、[トレース] ウィンドウにトレースデータが表示されます。ウィンドウの情報については、「193 ページの [トレース] ウィンドウ」を参照してください。

SWO トレースを開始するには

SWO トレースを開始するには：

- 1 C-SPY を起動する前に、J-Link/J-Trace の場合は [プロジェクト] > [オプション] > [J-Link/J-Trace] を、ST-LINK の場合は [プロジェクト] > [オプション] > [ST-Link] をそれぞれ選択します。[接続] タブをクリックして、[インタフェース] > [SWD] を選択します。
- 2 C-SPY を起動したら、[J-Link] メニューまたは [ST-LINK] メニューから [SWO トレースウィンドウ設定] を選択します。表示される [SWO トレースウィンドウ設定] ダイアログボックスで、[トレース] ウィンドウの出力を制御する設定を行います。統計的なトレースデータを見るには、オプション [強制] > [PC サンプル] を選択します (188 ページの [SWO トレースウィンドウ設定] ダイアログボックスを参照)。
- 3 ハードウェアによるトレースデータの生成を設定するには、[SWO 設定] ダイアログボックスの [SWO 設定] ボタンをクリックします。詳細については、190 ページの [SWO 設定] ダイアログボックスを参照してください。

特に以下の設定に注意してください。



- [CPU クロック] オプションの値は、アプリケーションを実行する CPU クロックのスピードを反映する必要があります。また、設定はデバッグセッション間で保持されます。
- 通信チャンネルの送信量を減らすには、[タイムスタンプ] オプションを無効にします。または、[PC サンプリング] のレートを低くするか、SWO クロック周波数を高く設定します。



- 4 [SWO トレース] ウィンドウ ([J-Link/J-Trace] メニューまたは [ST-LINK] メニューからアクセス) を開いて、[有効化] ボタンをクリックしてトレースデータの収集を有効にします。
- 5 実行を開始します。[トレース] ウィンドウは、トレースデータによって継続的に更新されます。このウィンドウの情報については、「193 ページの [トレース] ウィンドウ」を参照してください。

ETM および SWO の並列使用の設定

Cortex-M3 用の J-Trace デバッグプロブがある場合、ETM トレースと SWO トレースを同時に使用できます。

この場合、ETM トレースと SWO トレースを有効にすると、SWO トレースは SWO チャンネルを介してストリーム送信される代わりに、ETM トレースバッファでも収集されます。つまり、SWO トレースデータは [SWO トレース] ウィンドウで連続してリアルタイムで更新されるのではなく、実行が停止するまで表示されません。

ブレイクポイントを使用したトレースデータの収集

2つの実行ポイント間でトレースデータを収集する簡単な方法は、専用のブレイクポイントを使用してデータ収集を開始および停止することです。以下から選択します。

- エディタか「逆アセンブリ」ウィンドウで、挿入ポイントを配置して右クリックし、コンテキストメニューから「トレース開始」または「トレース停止」ブレイクポイントを切り替えます。
- 「ブレイクポイント」ウィンドウで、「トレース開始」、「トレース停止」、「トレースフィルタ」を選択します。
- C-SPY システムマクロ `__setTraceStartBreak` と `__setTraceStopBreak` も使用できます。

これらのブレイクポイントについて詳しくは、「207 ページの「トレース開始ブレイクポイント」ダイアログボックス（シミュレータ）」と「208 ページの「トレース停止ブレイクポイント」ダイアログボックス（シミュレータ）」をそれぞれ参照してください。

トレーストリガとトレースフィルタの使用：

- 1 「トレース開始」ダイアログボックスを使用して、トレースデータの収集を開始するための開始条件（開始トリガ）を設定します。
- 2 「トレース停止」ダイアログボックスでは、トレースデータの収集を停止するための停止条件（停止トリガ）を設定します。
- 3 オプションで、トレースデータの収集を続行するための追加条件を設定します。さらに、「トレースフィルタ」ダイアログボックスを使用して、1つまたは複数のトレースフィルタを設定します。
- 4 必要ならば、追加のトレース条件またはトレース停止条件を設定してください。
- 5 「トレース」ウィンドウを有効にして、実行を開始します。
- 6 実行を停止します。
- 7 トレースデータは、「トレース」ウィンドウおよび「逆アセンブリ」ウィンドウのブラウズモードで参照できます。ここでは、トレーストリガおよびトレースフィルタのトレース跡も見ることができます。
- 8 トレースフィルタを設定した場合、条件が真で何らかの命令がある場合にトレースデータの収集が実行されます。トレースデータを参照して特定のデータアクセスを探すときには、アクセスが命令1つ前に発生していることに注意してください。

トレースデータの検索

トレースデータを収集したら、収集したデータに対して検索を実行し、関心のあるコードまたはデータの一部を見つけることができます。たとえば、特定の命令や特定の変数へのアクセスなどです。

【トレースを検索】ダイアログボックスで検索基準を指定すると、【トレースを検索】ウィンドウに結果が表示されます。

【トレースを検索】ウィンドウは【トレース】ウィンドウと非常によく似ており、表示される列とデータは同じですが、表示される行は、指定された検索基準に一致した行だけです。【トレースを検索】ウィンドウで項目をダブルクリックすると、【トレース】ウィンドウに同じ項目が表示されます。

トレースデータ内を検索するには：



1 トレースウィンドウツールバーで、**【検索】** ボタンをクリックします。

2 **【トレースを検索】** ダイアログボックスで、検索基準を指定します。

通常は以下を対象に検索基準を選択します。

- 文字の一部。検索基準を追加して適用することができます
- アドレスの範囲
- 特定のアドレス範囲における特定の文字の一部というように、上記を組み合わせたもの

さまざまなオプションの詳細については、219 ページの **【トレースを検索】ダイアログボックス**を参照してください。

3 検索基準を指定したら、**【検索】** をクリックします。【トレースを検索】ウィンドウが表示され、トレースデータを分析できるようになります。詳細については、221 ページの **【トレースを検索】ウィンドウ**を参照してください。

トレースデータの参照

【トレース】ウィンドウを表示してスクロールするだけで、実行履歴をたどることができます。別の方法として、**ブラウズモード**に入ることができます。



ブラウズモードに入るには、【トレース】ウィンドウで項目をダブルクリックするか、ツールバーの **【ブラウズ】** ボタンをクリックします。

選択された項目が黄色で表示され、ソースウィンドウと逆アセンブリウィンドウの対応する位置が強調表示されます。ここで、上向き矢印キーと下向き矢印キーを使用するか、スクロールしてクリックすることで、トレースデータ内を移動できます。ソースウィンドウと逆アセンブリウィンドウも、対応する位置が表示されるように更新されます。これは、実行履歴を前後に移動するのと似ています。

もう一度ダブルクリックすると、ブラウズモードが終了します。

トレースのリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 186 ページの [ETM トレース設定] ダイアログボックス
- 188 ページの [SWO トレースウィンドウ設定] ダイアログボックス
- 190 ページの [SWO 設定] ダイアログボックス
- 193 ページの [トレース] ウィンドウ
- 198 ページの [トレースの保存] ダイアログボックス
- 199 ページの [関数トレース] ウィンドウ
- 200 ページの [タイムライン] ウィンドウ
- 246 ページの [Power ログ] ウィンドウ
- 207 ページの [トレース開始ブレークポイント] ダイアログボックス (シミュレータ)
- 208 ページの [トレース停止ブレークポイント] ダイアログボックス (シミュレータ)
- 209 ページの [トレース開始] ブレークポイントダイアログボックス
- 213 ページの [トレース停止] ブレークポイントダイアログボックス
- 216 ページの [トレースフィルタ] ブレークポイントダイアログボックス
- 218 ページの [トレース式] ウィンドウ
- 219 ページの [トレースを検索] ダイアログボックス
- 221 ページの [トレースを検索] ウィンドウ

【ETM トレース設定】 ダイアログボックス

【ETM トレース設定】 ダイアログボックスは、C-SPY ドライバのメニューから使用できます。

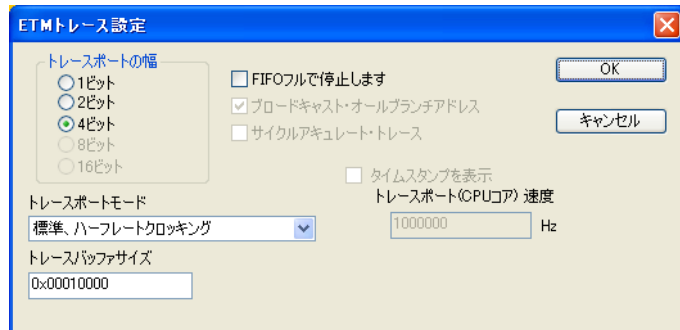


図 76: 【ETM トレース設定】 ダイアログボックス

このダイアログボックスは以下で使用できます。

- J-Link/J-Trace ドライバ

このダイアログボックスを使用して、ETM トレースの生成と収集を設定します。

関連項目：

- 180 ページの ETM トレースを使用するための条件
- 181 ページの ETM トレースを開始するには

トレースポートの幅

トレースのバス幅に 1、2、4、8、16 ビットを指定します。この値はハードウェアおよびデバッグプロンプがサポートするバス幅に合わせる必要があります。Cortex-M3 の場合、J-Trace デバッグプロンプで 1、2、4 ビットがサポートされています。ARM7/9 の場合、J-Trace デバッグプロンプで 4 ビットのみがサポートされています。

トレースポートモード

使用されたトレースのクロックレートを指定します。

- 標準、フルレートクロッキング
- 標準、ハーフレートクロッキング
- 多重化

- 逆多重化
- 逆多重化、ハーフレートクロッキング

注：RDI ドライバの場合、最初の 2 つの選択肢のみ使用できます。J-Trace ドライバの場合、使用可能な選択肢は使用するデバイスによって異なります。

トレースバッファサイズ

トレースバッファのサイズを指定します。デフォルトでは、トレースフレームの数は 0xFFFF です。ARM7/9 では最大数は 0xFFFFF で、Cortex-M3 では 0x3FFFFFF です。

ARM7/9 の場合、1 つのトレースフレームは J-Trace の物理的バッファサイズの 2 バイトに相当します。Cortex-M3 では、1 つのトレースフレームはバッファサイズの約 1 バイトになります。

注： [トレースバッファサイズ] オプションは J-Trace ドライバでのみ使用できます。

サイクルアキュレート・トレース

トレースデータが使用できない場合でも、プロセッサのクロックに同期するトレースフレームを出力します。このため、リアルタイムのタイミング計算にトレースデータを利用できます。ただしこのオプションを選択すると、FIFO バッファオーバーフローの危険性が増大します。

注： このオプションは、ARM7/9 デバイスでのみ使用できます。

ブロードキャスト・オールブランチ

プロセッサからより詳細なアドレステレース情報が送信されます。ただしこのオプションを選択すると、FIFO バッファオーバーフローの危険性が増大します。

注： このオプションは、ARM7/9 デバイスでのみ使用できます。Cortex では、このオプションは常に有効になっています。

FIFO フルで停止

FIFO のバッファが満杯になったときにプロセッサを停止します。トレース FIFO バッファは、条件によっては満杯になる (FIFO バッファオーバーフロー) 可能性があるため、トレースデータが失われる場合があります。

タイムスタンプを表示

[トレース] ウィンドウの [インデックス] 列にサイクルではなく、秒を表示します。このオプションを使用するには、[トレースポート (CPU コア) 速度] テキストボックスに、使用する CPU に適した速度を設定することも必要です。

注： このオプションは、J-Trace ドライバを ARM7/9 デバイスで使用する場合のみ利用できます。

[SWO トレースウィンドウ設定] ダイアログボックス

[SWO トレースウィンドウ設定] ダイアログボックスは、[J-Link] メニューまたは [ST-LINK] メニュー、あるいは [SWO トレース] ウィンドウのツールバーから利用できます。

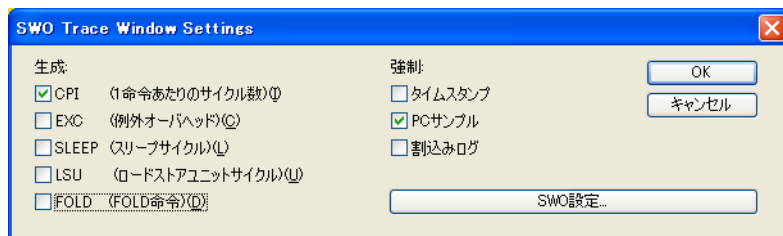


図 77: [SWO トレースウィンドウ設定] ダイアログボックス

このダイアログボックスを使用して、[SWO トレース] ウィンドウに表示する内容を指定します。

トレースデータの生成も設定する必要があります。[SWO 設定] をクリックしてください。詳細については、190 ページの [SWO 設定] ダイアログボックスを参照してください。

強制

SWO トレースデータを使用して他の機能により有効になっていない場合、データ生成を有効にします。[トレース] ウィンドウには、生成されたすべての SWO データが表示されます。プロファイリングなど C-SPY の他の機能では、SWO トレースデータの生成を有効にすることもできます。他の機能によって生成が有効化されていない場合、[強制] オプションを使用して SWO トレースデータを生成します。

生成されたデータは、[トレース] ウィンドウに表示されます。以下から選択します。

タイムスタンプ	SWO 通信チャンネルを介して送信される、さまざまな SWO トレースパケットについてタイムスタンプを有効にします。タイムスタンプ値の間隔をドロップダウンリストから選択します。たとえば、1 サイクルごとにカウントする場合は 1 を、16 サイクルごとにカウントする場合は 16 を選択します。最小値は、各イベントパケットの間隔が十分に長い場合にのみ有益であることに注意してください。16 は低い SWO クロック周波数を使用している場合に便利です。
PC サンプル	プログラムカウンタレジスタ (PC) の定期的なサンプリングを有効にします。サンプリングレートを選択するには、191 ページの <i>PC サンプリング</i> を参照してください。
割込みログ	割込みログの生成を有効にします。割込みにトレースデータを使用する他の C-SPY 機能については、255 ページの <i>割込み</i> を参照してください。

生成

以下のイベントでトレースデータの生成を有効にします。生成されたデータは、[トレース] ウィンドウに表示されます。カウンタの値は、[SWO トレース] ウィンドウの [コメント] 列に表示されます。以下から選択します。

CPI	CPI カウンタについてトレースデータの生成を有効にします。
EXC	EXC カウンタについてトレースデータの生成を有効にします。
SLEEP	SLEEP カウンタについてトレースデータの生成を有効にします。
LSU	LSU カウンタについてトレースデータの生成を有効にします。
FOLD	FOLD カウンタについてトレースデータの生成を有効にします。

SWO 設定

[SWO 設定] ダイアログボックスを表示します。ここでは、ハードウェアのトレースデータの生成を設定することができます。190 ページの [SWO 設定] ダイアログボックスを参照してください。

【SWO 設定】 ダイアログボックス

【SWO 設定】 ダイアログボックスは、[J-Link] メニューまたは [ST-LINK] メニュー、あるいは【SWO トレースウィンドウ設定】 ダイアログボックスから利用できます。

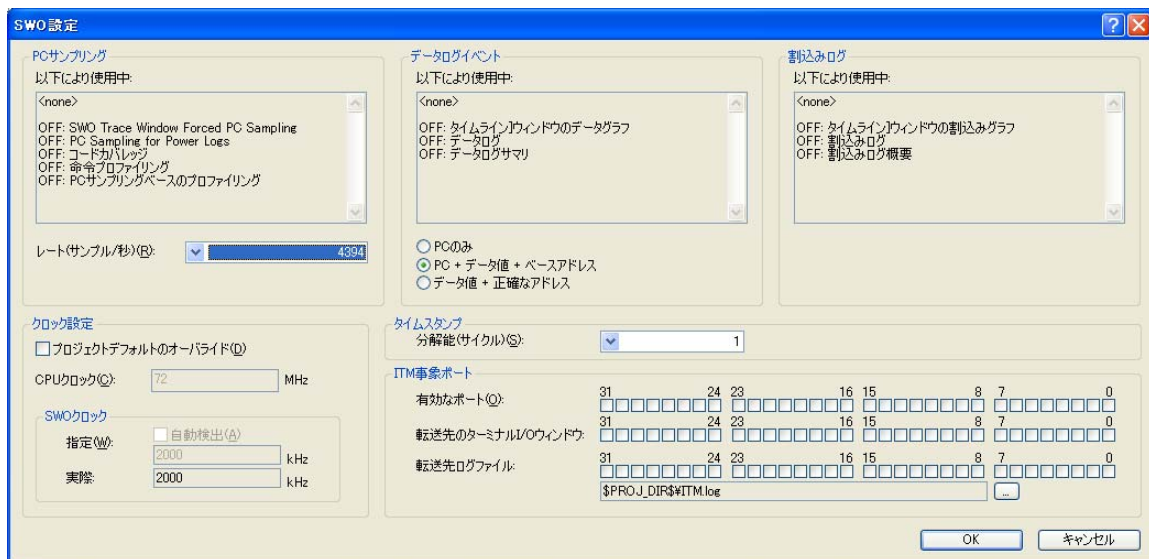


図 78: 【SWO 設定】 ダイアログボックス

このダイアログボックスは、serial-wire output 通信チャンネルおよびハードウェアによるトレースデータの生成を設定するときに使用します。

SWO トレースを開始するには 182 ページの SWO トレースを開始するにはも参照してください。

PC サンプリング

プログラムカウンタのサンプリング動作を制御します。以下を指定できます。

以下により 使用中	PC サンプリングでトレースデータを使用可能な C-SPY の機能を一覧表示します。ON は、現在トレースデータを使用中の機能を示します。OFF は、現在トレースデータを使用中でない機能を示します。
レート	サンプリングレート（1 秒あたりのサンプル数）をドロップダウンリストから選択します。最大可能サンプリングレートは、SWO クロック値と、SWO 通信チャンネル経由で送信される他のデータ量により決まります。SWO 通信チャンネルが大量のデータを処理できるほど高速でない場合、リストで最大値を指定すると適切に機能しません。

データログイベント

データログブレイクポイントがトリガされたときにログに記録する対象を指定します。以下の項目を選択できます。

以下により 使用中	データログイベントでトレースデータを使用可能な C-SPY の機能を一覧表示します。ON は、現在トレースデータを使用中の機能を示します。OFF は、現在トレースデータを使用中でない機能を示します。
PC のみ	プログラムカウンタの値をロギングします。
PC + データ値 + ベースアドレス	プログラムカウンタの値、データオブジェクトの値、およびそのベースアドレスをロギングします。
データ値 + 正確 なアドレス	データオブジェクトの値、およびアクセスされたデータオブジェクトの正確なアドレスを記録します。

割込みログ

割込みログでトレースデータを使用可能な C-SPY の機能を一覧表示します。ON は、現在トレースデータを使用中の機能を示します。OFF は、現在トレースデータを使用中でない機能を示します。

割込みロギングの詳細については、255 ページの *割込み* を参照してください。

プロジェクトデフォルトのオーバーライド

J-Link/J-Trace の [プロジェクト] > [オプション] > [J-Link/J-Trace] > [設定] ページの、または ST-LINK の [プロジェクト] > [オプション] > [ST-Link] > [設定] ページの [CPU クロック] および [SWO クロック] のデフォルト値を、それぞれオーバーライドします。

CPU クロック

内部プロセッサクロック HCLK の正確なクロック周波数を指定します (MHz)。10 進数で指定できます。

この値は、SWO の通信速度の設定およびタイムスタンプの計算に使用します。

SWO クロック

SWO 通信チャンネルのクロック周波数を指定します (kHz)。以下から選択します。

自動検出	J-Link デバッグプローブで使用できる最大可能周波数を自動的に使用します。選択された場合、[指定] (最大) テキストボックスにその周波数が表示されます。
指定	[自動検出] が選択されていない場合に、使用する周波数を手動で選択します。10 進数で指定できます。このオプションは、送信中にデータパケットが失われる場合に使用します。
実際	実際に使用される周波数を表示します。最大周波数とは微妙に異なる場合があります。

タイムスタンプ

タイムスタンプ値の分解能を選択します。たとえば、1 サイクルごとにカウントする場合は 1 を、16 サイクルごとにカウントする場合は 16 を選択します。最小値は、各イベントパケットの間隔が十分に長い場合にのみ有益であることに注意してください。

ITM 事象ポート

リダイレクトするポートと転送先を選択します。ITM 事象ポートは、アプリケーションからデバッガホストに、プログラムの実行を停止せずにデータを送信するときに使用されます。32 個のポートがあります。以下から選択します。

ポートの有効化 使用するポートを有効にします。有効にしたポートのみが、SWO 通信チャンネル経由でデバッガにデータを送信します。

転送先の [ターミナル I/O] ウィンドウ [ターミナル I/O] ウィンドウへのデータルーティングに使用するポートを指定します。

転送先ログファイル ログファイルへのデータルーティングに使用するポートを指定します。デフォルトのログファイル以外を使用する場合は、参照ボタンを使用して指定します。



アプリケーションの stdout と stderr は、セミホスティングではなく、SWO 経由で C-SPY の [ターミナル I/O] ウィンドウにルートすることができます。これには、[プロジェクト] > [オプション] > [一般オプション] > [ライブラリ構成] > [ライブラリ低レベルインタフェースの実装] > [stdout/stderr] > [SWO 経由] を選択します。こうすることで、セミホスティングを使用した場合に比べて、stdout/stderr のパフォーマンスが格段に向上します。

これは、[ポートの有効化] および [転送先ターミナル I/O] オプションでポート設定の選択を解除すると無効になります。

[トレース] ウィンドウ

[トレース] ウィンドウは、C-SPY ドライブメニューから使用できます。

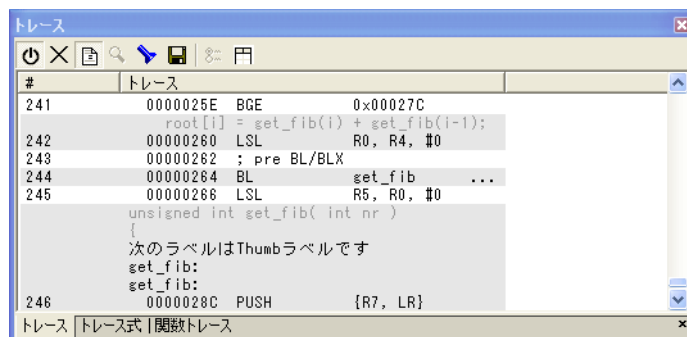


図 79: シミュレータの [トレース] ウィンドウ

注：C-SPY シミュレータには [ETM トレース]、[SWO トレース]、[トレース] の 3 つのウィンドウがあります。これらのウィンドウは少しずつ異なります。

このウィンドウには収集されたトレースデータが表示されます。その内容は、使用する C-SPY ドライバとデバッグプローブのトレースサポートによって異なります。

C-SPY シミュレータ このウィンドウには、収集された実行済みのマシン命令のシーケンスが表示されます。また、式のトレースデータも表示できます。

ETM トレース このウィンドウには、実行済みの命令シーケンス（オプションで組込みソース）が表示されます。これはアプリケーション実行時に継続して収集、すなわちフルトレースされたものです。データは ETM トレースバッファに収集されています。収集されたデータは、実行が停止した後に表示されます。

ETM トレースの要件については、180 ページの *ETM トレースを使用するための条件* を参照してください。

SWO トレース このウィンドウには、SWO チャンネルを介して送信されたすべてのイベントが表示されます。データは SWO 通信チャンネルを通してターゲットシステムからストリーム送信され、[トレース] ウィンドウで継続的にリアルタイムで更新されます。トレースプローブで SWO 通信チャンネルを使用する場合、データはトレースバッファで収集され、実行が停止した後に表示されます。

SWO トレースの要件については、180 ページの *SWO トレースを使用するための条件* を参照してください。

「トレース」 ツールバー

「トレース」 ウィンドウと 「関数トレース」 ウィンドウのツールバーには以下が含まれます。









有効 / 無効

このウィンドウにおけるトレースデータの収集および表示を有効および無効にします。[関数トレース] ウィンドウではこのボタンは使用できません。



トレースデータのクリア

トレースバッファをクリアします。[トレース] ウィンドウと [関数トレース] ウィンドウが両方ともクリアされます。

	ソースの切替え	[トレース] 列で、逆アセンブリだけを表示するか、逆アセンブリおよび対応するソースコードの両方を表示するか切り替えます。
	ブラウズ	[トレース] ウィンドウで選択された項目のブラウズモードのオンとオフを切り替えます (184 ページの <i>トレースデータの参照</i> を参照)。
	検索	検索を実行するダイアログボックスを表示します (219 ページの <i>[トレースを検索] ダイアログボックス</i> を参照)。
	保存	[ETM トレース] および [SWO トレース] の各ウィンドウでは、このボタンによって [トレースの保存] ダイアログボックスが表示されます (198 ページの <i>[トレースの保存] ダイアログボックス</i> を参照)。C-SPY シミュレータでは、このボタンは標準の [名前を付けて保存] ダイアログボックスを表示します。ここでは、収集されたトレースデータをタブで列ごとに区切られた形でテキストファイルに保存できます。
	設定の編集	<p>[ETM トレース] ウィンドウでは、このボタンによって [トレースの保存] ダイアログボックスが表示されます (186 ページの <i>[ETM トレース設定] ダイアログボックス</i> を参照)。</p> <p>[SWO トレース] ウィンドウでは、このボタンによって [SWO トレースウィンドウ設定] ダイアログボックスが表示されます (188 ページの <i>[SWO トレースウィンドウ設定] ダイアログボックス</i> を参照)。</p> <p>C-SPY シミュレータでは、このボタンは使用できません。</p>
	式の編集 (C-SPY シミュレータのみ)	[トレース式] ウィンドウを開きます (218 ページの <i>[トレース式] ウィンドウ</i> を参照)。

表示エリア (C-SPY シミュレータ内)

このエリアには C-SPY シミュレータの以下の列が表示されます。

#	トレースバッファの各行のシリアル番号。バッファ内の移動を簡略化します。
サイクル	この時点までのサイクル数。

トレース	収集された実行済みのマシン命令シーケンス。オプションで、対応するソースコードも表示できます。
式	表示するように定義した式はそれぞれ別の列に表示されます。式列の各エントリには、同じ行の命令が実行された後に値が表示されます。トレースデータを収集する式は、[トレース式] ウィンドウで指定します (218 ページの [トレース式] ウィンドウを参照)。

表示エリア (ETM トレース)

このエリアには、ETM トレースの以下の列が含まれます。

インデックス	それぞれのパケットに対応する番号。パケットの例は、命令や同期ポイント、例外マーカなどです。
フレーム 時間	<p>サイクルアキュレートモード (ARM7/9 が必要です) でトレースデータを収集する場合 ([ETM トレース設定] ダイアログボックスで [サイクルアキュレート・トレース] を有効化)、値は実行開始後に経過したサイクル数に一致します。この列は J-Link/J-Trace ドライバでのみ使用できます。</p> <p>サイクルアキュレート以外のモードでトレースデータを収集する場合、値はサイクルのおよその数となります。Cortex-M デバイスの場合、値は実際のサイクル数によって繰り返し調整されます。</p> <p>[ETM トレース設定] ダイアログボックスで [タイムスタンプを表示] オプションが選択されている場合、値としてサイクルではなく時間が表示されます。値を時間として表示するには、サイクルアキュレートモードでデータを収集する必要があります。187 ページの サイクルアキュレート・トレースおよび J-Link/J-Trace ドライバを参照してください。</p>
アドレス	実行した命令のアドレス
OP コード	実行した命令の動作コード
トレース	収集された実行済みのマシン命令シーケンス。オプションで、対応するソースコードも表示できます。
コメント	この列は J-Link/J-Trace ドライバでのみ使用できます。

注: RDI ドライバの場合、このウィンドウの表示は若干異なっています。

表示エリア (SWO トレース)

このエリアには、SWO トレースの以下の列が含まれます。

インデックス	トレースバッファの各行のインデックス番号。バッファ内の移動を簡略化します。
SWO パケット	取得した SWO パケットの内容。
サイクル	実行の開始からイベントまでのサイクルの概数。
イベント	取得した SWO パケットのイベントのタイプ。列にオーバーフローと表示される場合、多くの SWO 機能で同時に SWO チャンネルが使用されているため、データパケットは送信できませんでした。通信チャンネルの送信量を減らすには、SWO ボタン (IDE のメインウィンドウのツールバー上にあります) にマウスのポインタを合わせて、どの C-SPY 機能がトレースデータの生成を要求しているかについて詳細なツールチップ情報を入手します。一部の機能を無効にしてください。
値	イベントの値 (該当する場合)。
トレース	イベントが PC 値の場合、命令もこの列に表示されます。オプションで、対応するソースコードも表示できます。
コメント	追加情報。選択したトレースイベントカウンタの値、データログブレイクポイントに使用されたコンパレータ (ハードウェアブレイクポイント) の数が表示されます。



表示エリアに不明な文字が表示される場合、[SWO 設定] ダイアログボックスの [CPU クロック] に正しい値を指定したか確認してください。

【トレースの保存】ダイアログボックス

【トレースの保存】ダイアログボックスは、ドライバ固有のメニューと、【トレース】ウィンドウおよび【SWO トレース】ウィンドウから使用できます。

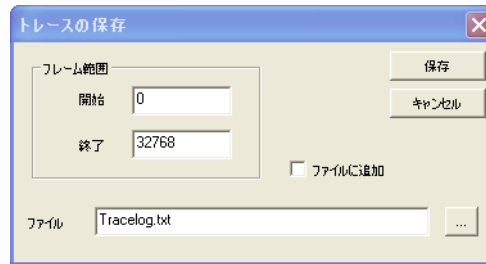


図 80: 【トレースの保存】ダイアログボックス

インデックス範囲

フレームの範囲をファイルに保存します。開始インデックスと終了インデックスを指定します（【トレース】ウィンドウのインデックス列の値）。

ファイルに追加

トレースデータを既存ファイルに追加します。

タブ区切りフォーマットを使用

列の内容を、スペースではなくタブ区切りで保存します。

ファイル

トレースデータのファイルを指定します。

「関数トレース」ウィンドウ

「関数トレース」ウィンドウは、デバッグセッション中に「C-SPY ドライバ」メニューから使用できます。

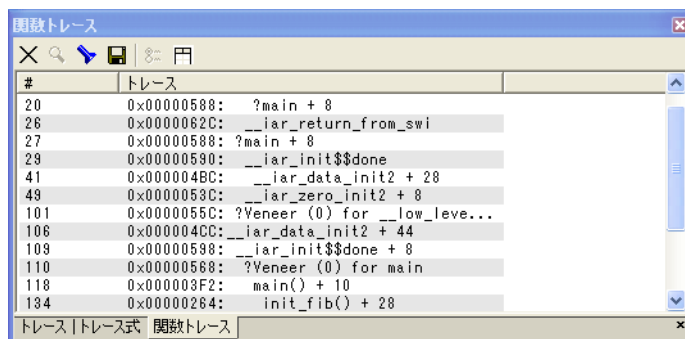


図 81: 「関数トレース」ウィンドウ

このウィンドウは以下で使用できます。

- C-SPY シミュレータ
- J-Trace ドライバ

このウィンドウには、「トレース」ウィンドウに表示されたトレースデータのサブセットが表示されます。「関数トレース」ウィンドウにはすべての行は表示されません。関数呼出しと関数からの復帰に対応するトレースデータだけが表示されます。

ツールバー

ツールバーについては、194 ページの「トレース」ツールバーを参照してください。

表示エリア

表示エリアの列については、以下を参照してください。

- 195 ページの 表示エリア (C-SPY シミュレータ内)
- 196 ページの 表示エリア (ETM トレース)

【タイムライン】 ウィンドウ

【タイムライン】 ウィンドウは、デバッグセッション中に *[C-SPY ドライバ]* メニューから使用できます。

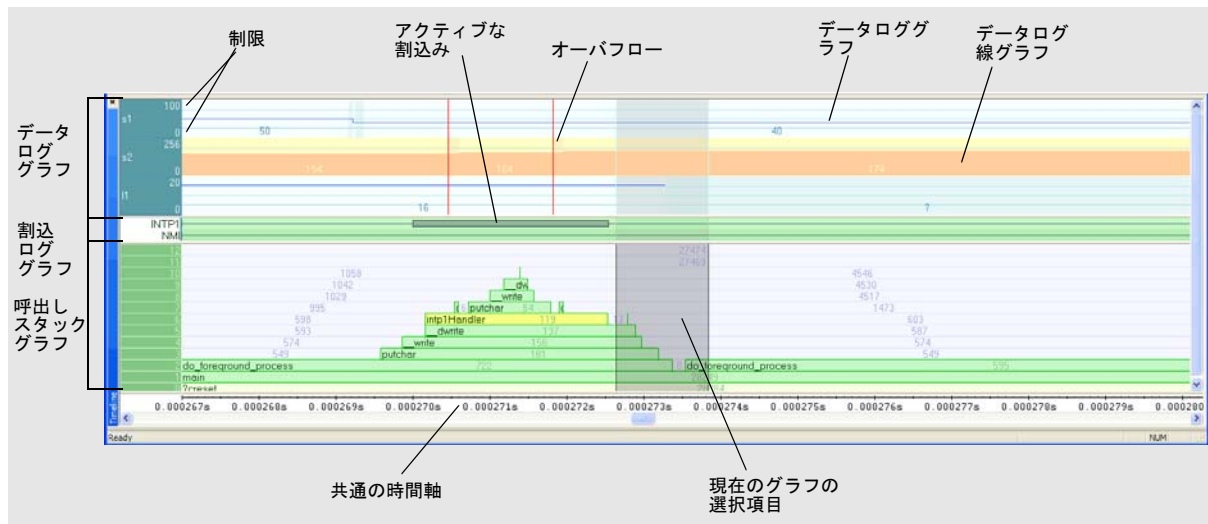


図 82: 【タイムライン】 ウィンドウ

このウィンドウは以下で使用できます。

- C-SPY シミュレータ
- J-Link/J-Trace ドライバ
- ST-LINK ドライバ

このウィンドウには (割り込みログ、データログ、Power ログ、呼出しスタックの) トレースデータが、共通の時間軸に沿ったグラフとして表示されます。

グラフを表示するには：

- 1 ドライバのメニュー > **[SWO 設定]** を選択して、**[SWO 設定]** ダイアログボックスを開きます。**[CPU クロック]** オプションが、アプリケーションの CPU クロックの値と同じに設定されているか確認してください。**SWO** クロックを設定して、デバッグブロープへ正しいデータ転送を行うには、こうする必要があります。

C-SPY シミュレータを使用する場合は、この手順は無視してかまいません。

- 2 *C-SPY* ドライバのメニューから **[タイムライン]** を選択して、**[タイムライン]** ウィンドウを開きます。

- 3 [タイムライン] ウィンドウのグラフエリア右クリックして、コンテキストメニューから **[有効化]** を選択し、特定のグラフを有効にします。
- 4 データロググラフの場合は、[タイムライン] ウィンドウでグラフィック表示を行う各変数にデータログブレイクポイントを設定する必要があります。
145 ページの [データログ] ブレイクポイントダイアログボックスを参照してください。
- 5 ツールバーで **[Go]** をクリックして、アプリケーションの実行を開始します。
グラフが表示されます。

グラフ内を移動するには、以下の選択肢のいずれかを使用します。

- 右クリックしてコンテキストメニューから **[ズームイン]** または **[ズームアウト]** を選択します。その他に、**[+]** や **[-]** のキーを使用することもできます。使用したコマンドに応じて、グラフがズームインまたはズームアウトします。
- グラフを右クリックして、コンテキストメニューから **[移動]** およびグラフ上で前後に移動するための適切なコマンドを選択します。または、矢印キーや **[Home]**、**[End]**、**[Ctrl]+[End]** などのショートカットキーをどれでも使用できます。
- 関心のあるサンプルの値をダブルクリックすると、対応するソースコードがエディタウィンドウと **[逆アセンブリ]** ウィンドウで強調表示されます。
- グラフをクリックし、ドラッグして間隔を選択します。**[Enter]** または右クリックして、コンテキストメニューから **[ズーム] > [選択範囲をズーム]** を選択します。選択内容にズームインします。



マウスポインタをグラフにあわせると、その場所の詳細なツールチップ情報が表示されます。

表示エリア

使用する C-SPY ドライバによっては、表示エリアに異なるグラフが読み込まれることがあります。

グラフ	C-SPY シミュレータ	J-Link ドライバ	J-Trace ドライバ	ST-LINK ドライバ
割込みロググラフ	X	X	X	X
データロググラフ	--	X	X ¹	X
呼出しスタックグラフ	X	--	X	--
Power ロググラフ	--	X	--	--

表 10: [タイムライン] ウィンドウでサポートされているグラフ

! ETM が無効の場合のみ使用可

特定のグラフが使用できるかどうかは、ハードウェアやデバッグプローブ、C-SPY ドライバの機能によって異なります。38 ページの *ドライバ間の差異*、179 ページの *トレースを使用するための条件*を参照。

ウィンドウの下部分に、秒を時間単位として使用する共通の時間軸があります。

割込みロググラフ

割込みロググラフには、SWO トレースまたは C-SPY シミュレータによって報告される割込みが表示されます。つまり、このグラフには、アプリケーションプログラム実行中の割込みイベントが以下のようにグラフィック表示されます。

- グラフ左端のラベルエリアには、割込み名が表示されます。
- グラフ自体には、アクティブな割込みが濃い緑の横棒として示されます。このグラフは、[割込みログ] ウィンドウの情報をグラフィック表示したものです (271 ページの [割込みログ] ウィンドウを参照)。

データロググラフ

データロググラフには、SWO トレースによって生成されるデータログが、データログブレイクポイントとして指定された最高 4 つの変数またはアドレス範囲について以下のように表示されます。

- 各グラフの左側には、データログブレイクポイントとして指定した変数名またはアドレスのラベルが付きます。
- グラフ自体には、変数の値が時間とともにどう変化するかを示されます。ラベルエリアには、変数に対して Y 軸の制限や範囲也表示されます。コンテキストメニューを使用して、これらの制限を変更できます。グラフは、細い線またはカラーの実線によるグラフとして表示することができます。このグラフは、[データログ] ウィンドウの情報をグラフィック表示したものです (115 ページの [データログ] ウィンドウを参照)。
- 赤色の垂直線はオーバーフローを示します。これは、通信チャンネルがすべてのデータログをターゲットシステムから送信できなかったことを示します。

呼出しスタックグラフ

呼出しスタックグラフには、ETM トレースによって収集された呼出しおよびリターンのシーケンスが表示されます。グラフの下部分には通常、main があり、その上には main という関数があります。横方向の棒は関数の呼出しを示し、4 つの異なる色を使用します。

- 緑は、デバッグ情報を持つ通常の C 関数です。
- ライトグリーンは、デバッガがアセンブララベルを通してのみ認識する関数です。

- 黄色または薄い黄色は、割込みハンドラで緑の場合と同じように区別されます。

数字は関数の呼出し時、または呼出し間のサイクル数を表します。

Power ロググラフ

Power ロググラフには、デバッグプローブや関連のハードウェアによって生成された電力測定サンプルが表示されます。

選択およびナビゲーション

選択するには、クリックしてドラッグします。選択内容はすべてのグラフについて縦方向に伸びますが、選択したグラフに対して濃い色で強調表示されます。左右の矢印キーを使用して、選択したグラフ内を前後に移動することができます。Home キーおよび End キーを使用して、先頭または最後の地点にそれぞれ移動します。選択内容を広げるには、ナビゲーションキーを Shift キーと組み合わせて使用します。

コンテキストメニュー

以下のコンテキストメニューがあります。

移動	▶
▼ オートスクロール	
ズーム	▶
割込み	
▼ 有効化	
ソースへ移動	
グラフを選択	▶
時間軸単位	▶
プロファイル選択	

図 83: 呼出しスタックグラフの [タイムライン] ウィンドウのコンテキストメニュー

注：コンテキストメニューには、すべてのグラフに共通なコマンドと、各グラフに固有のコマンドが含まれます。この図は呼出しスタックグラフのコンテキストメニューを表します。つまり、メニューの外観が他のグラフとは少し違います。

以下のコマンドがあります。

移動	すべてのグラフ	グラフ上を移動するコマンド。以下から選択します。 [次へ] は、グラフ内の次の適切な地点に選択内容を動かします。ショートカットキー：→ [前へ] は、グラフ内の次の適切な地点に選択内容を戻します。ショートカットキー：← [最初] は、グラフ内の最初のデータ項目に選択内容を動かします。ショートカットキー：Home [最後] は、グラフ内の最後のデータ項目に選択内容を動かします。ショートカットキー：End End は、表示されたすべてのグラフの最後のデータに選択内容を動かします。つまり、時間軸の最後です。ショートカットキー：Ctrl+End
オートスクロール	すべてのグラフ	スクロールのオンとオフを切り替えます。オンの場合、最近収集したデータが自動的に表示されます。
ズーム	すべてのグラフ	ウィンドウをズームするためのコマンド。つまり、タイムスケールを変更します。以下から選択します。 選択範囲をズームは、現在の選択内容をウィンドウに合わせます。ショートカットキー：リターン ズームインは、タイムスケールを拡大します。ショートカットキー：+。 ズームアウトは、タイムスケールを縮小します。ショートカットキー：-。 10ns や 100ns、1us などは、それぞれ間隔 10 ナノ秒、100 ナノ秒、1 マイクロ秒をウィンドウに合わせます。 1ms、10ms などは、間隔をそれぞれ 1 ミリ秒または 10 ミリ秒にしてウィンドウに合わせます。 10m、1h などは、間隔をそれぞれ 10 分または 1 時間にしてウィンドウに合わせます。

データログ	データログ グラフ	以下のデータログ固有のコマンドが使用可能なことを示す見出し。
Power ログ	Power ログ グラフ	以下の Power ログ固有のコマンドが使用可能なことを示す見出し。
呼出しスタック	呼出しスタック グラフ	以下の呼出しスタック固有のコマンドが使用可能なことを示す見出し。
割込	割込みログ グラフ	以下の割込みログ固有のコマンドが使用可能なことを示す見出し。
有効化	すべての グラフ	グラフの表示のオンとオフを切り替えます。グラフを無効にすると、そのグラフは [タイムライン] ウィンドウで OFF として示されます。グラフについて収集されたトレースデータがない場合、グラフではなくデータなしと表示されます。
変数	データログ グラフ	以下のデータログ固有のコマンドが適用される変数名。このメニューコマンドは、コンテキストに依存します。つまり、[タイムライン] ウィンドウで選択したデータロググラフが反映されます (最高 4 つまで)。
線グラフ	データログ グラフ	細い線ではなく、カラーの実線グラフとしてグラフを表示します。
表示範囲	データおよび Power ログ グラフ	ダイアログボックスが開きます (206 ページの [表示範囲] ダイアログボックスを参照)。
サイズ	データおよび Power ログ グラフ	グラフの縦のサイズを指定します。小、中、大から選択してください。
数値を表示	データおよび Power ログ グラフ	グラフに加えて、変数の数値を表示します。
ソースへ移動	共通	エディタウィンドウの対応するソースコードを表示します (該当する場合)。
グラフを選択	共通	[タイムライン] ウィンドウで表示するグラフを選択します。

時間軸単位	共通	時間軸で使用する単位として、秒とサイクルのどちらかを選択します。
プロファイル選択	共通	[関数プロファイラ] ウィンドウでプロファイリングの間隔を有効にします。このコマンドは、C-SPY ドライバが PC サンプルングをサポートしている場合にのみ使用できます。

「表示範囲」 ダイアログボックス

「表示範囲」 ダイアログボックスは、[タイムライン] ウィンドウの Power ロググラフまたはデータロググラフを右クリックして表示されるコンテキストメニューから使用できます。

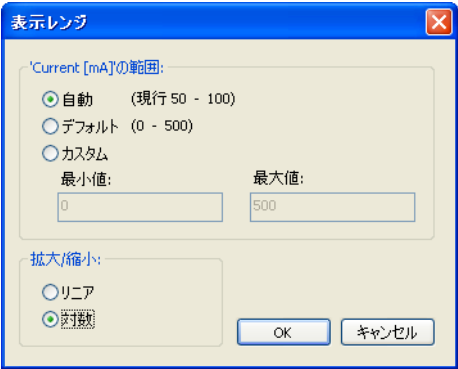


図 84: 「表示範囲」 ダイアログボックス

このダイアログボックスを使用して、値の範囲を指定します。つまり、グラフの Y 軸の範囲です。

xxxx の範囲

表示された値の表示範囲を選択します。

自動	最小値や最大値の継続的に管理しつつ、実際に収集された値の範囲に基づいた範囲を使用します。現在算出されている範囲があれば、それが括弧内に表示されます。範囲は適度に均等な限界値に丸められます。
----	--

- デフォルト** データロググラフの場合：変数の範囲の値に基づいて範囲を使用します。たとえば、符号なしの 16 ビットの整数の場合、0-65535 です。
- Power ロググラフの場合：測定用ハードウェアのプロパティに基づいた範囲を使用します。
- カスタム** テキストボックスを使用して、明示的な範囲を指定します。

スケール

Y 軸のスケールタイプを選択します。

- リニア
- 対数

〔トレース開始ブレイクポイント〕ダイアログボックス（シミュレータ）

〔トレース開始〕ダイアログボックスは、〔ブレイクポイント〕ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。

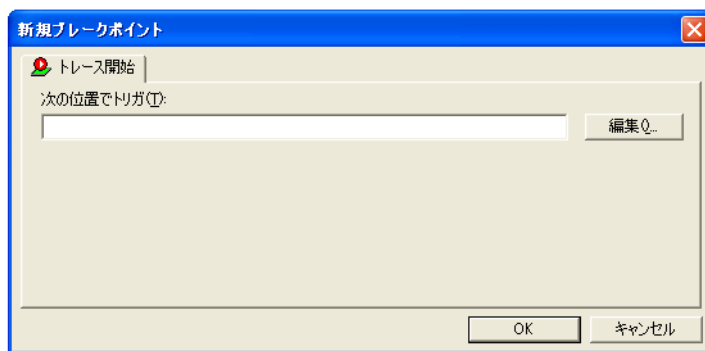


図 85: 〔トレース開始〕ブレイクポイントダイアログボックス（シミュレータ）

このダイアログボックスは、C-SPY シミュレータで使用できます。209 ページの 〔トレース開始〕ブレイクポイントダイアログボックスも参照してください。

トレース開始ブレークポイントを設定するには、以下の手順に従います。

- 1 エディタまたは [逆アセンブリ] ウィンドウで、右クリックしてコンテキストメニューから [トレース開始] を選択します。
または、[表示] > [ブレークポイント] を選択して、[ブレークポイント] ウィンドウを開きます。
- 2 [ブレークポイント] ウィンドウで、右クリックして [新規ブレークポイント] > [トレース開始] を選択します。
既存のブレークポイントを変更するには、[ブレークポイント] ウィンドウでブレークポイントを選択し、コンテキストメニューから [編集] を選択します。
- 3 [トリガ位置] テキストボックスで、式や絶対アドレス、ソース位置を指定します。[OK] をクリックします。
- 4 ブレークポイントがトリガされると、トレースデータの収集が始まります。

トリガ位置

[ブレーク位置] テキストボックスでブレークポイントの位置を指定します。または、[編集] 参照ボタンをクリックして [位置入力] ダイアログボックスを表示します (150 ページの [位置入力] ダイアログボックスを参照)。

[トレース停止ブレークポイント] ダイアログボックス (シミュレータ)

[トレース停止] ダイアログボックスは、[ブレークポイント] ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。

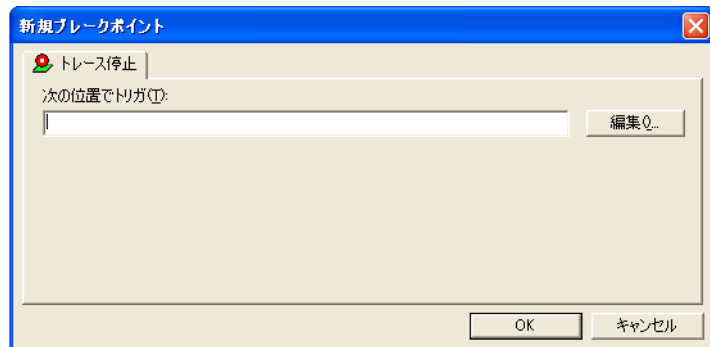


図 86: [トレース停止] ブレークポイントダイアログボックス (シミュレータ)

このダイアログボックスは、C-SPY シミュレータで使用できます。213 ページの [トレース停止] ブレークポイントダイアログボックスも参照してください。

レース停止ブレークポイントを設定するには：

- 1 エディタまたは [逆アセンブリ] ウィンドウで、右クリックしてコンテキストメニューから [トレース停止] を選択します。
または、[表示] > [ブレークポイント] を選択して、[ブレークポイント] ウィンドウを開きます。
- 2 [ブレークポイント] ウィンドウで、右クリックして [新規ブレークポイント] > [トレース停止] を選択します。
既存のブレークポイントを変更するには、[ブレークポイント] ウィンドウでブレークポイントを選択し、コンテキストメニューから [編集] を選択します。
- 3 [トリガ位置] テキストボックスで、式や絶対アドレス、ソース位置を指定します。[OK] をクリックします。
- 4 ブレークポイントがトリガされると、トレースデータの収集が終了します。

トリガ位置

[ブレーク位置] テキストボックスでブレークポイントの位置を指定します。または、[編集] ボタンをクリックして [位置入力] ダイアログボックスを表示します (150 ページの [位置入力] ダイアログボックスを参照)。

[トレース開始] ブレークポイントダイアログボックス

[トレース開始] ダイアログボックスは、[ブレークポイント] ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。または、[エディタ] ウィンドウまたは [逆アセンブリ] ウィンドウで右クリックして、[ブレークポイントの切替え (トレース開始)] を選択することもできます。

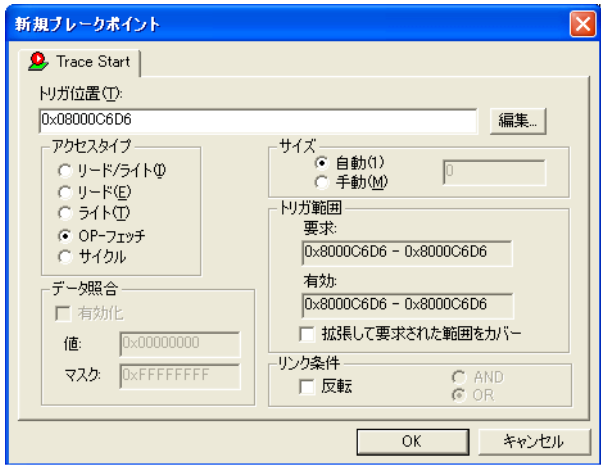


図 87: [トレース開始] ブレークポイントダイアログボックス (J-Link/J-Trace)

このダイアログボックスを使用して、トレースデータの収集を開始するタイミングを決定する条件を設定します。トレース条件がトリガされると、トレースデータの収集が始まります。

このダイアログボックスは、C-SPY J-Link/J-Trace ドライバで使用できます。

トリガ位置

トレースデータを収集するコードセクションの開始点を指定します。変数名やアドレス、サイクルカウンタの値を指定できます。

サイズ

アドレス範囲のサイズを制御します。このサイズに達すると、トレースデータ収集の開始がトリガされます。以下から選択します。

- 自動

サイズを自動的に設定します。これは、[トリガ位置] に変数が含まれる場合に便利です。
- 手動

ブレークポイント範囲のサイズを手動で指定します。

トリガ範囲

要求された範囲とトレースデータの収集でカバーする有効範囲が表示されます。推奨される範囲は、**【トリガ位置】**と**【サイズ】**オプションによって指定された領域とまったく同じか、その内側です。

拡張して要求された範囲をカバー データ構造体がカバーされるように範囲を拡張します。ハードウェアブレークポイント装置で提供可能な範囲のサイズと合わないデータ構造（3 バイトなど）の場合、範囲はデータ構造全体を対象としません。範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。

このオプションは、ARM7/9 デバイスでは有効になっていません。理由は、こうしたデバイスの範囲は常にデータ構造全体をカバーするためです。

アクセスタイプ

トレースデータの収集をトリガするメモリアクセスのタイプを指定します。以下から選択します。

リード/ライト	指定された位置から読み取り / 書き込みを行います。
リード	指定された位置から読み取ります。
ライト	指定された位置に書き込みます。
OP フェッチ	実行位置のアドレス
サイクル	実行開始点から数えた特定の時点でのカウンタサイクル数。このオプションは、Cortex-M デバイスでのみ使用できます。

データ照合

アクセスされるデータの照合を有効にします。[データ照合] オプションを、リード/ライト、リードまたはライトのデータアクセスタイプと組み合わせで使用します。このオプションは、変数が特定の値を持つときにトリガが必要な場合に便利です。

値	データ値を指定します。
マスク	値のどの部分（ワード、ハーフワード、バイト）を照合するか指定します。

[データ照合] オプションは、J-Link/J-Trace でのみ使用可能です（Cortex-M デバイス使用時のみ）。

注：Cortex-M デバイスについては、1つのブレイクポイントにのみ [データ照合] を設定できます。このようなブレイクポイントでは、2つのブレイクポイントリソースを使用します。

リンク条件

AND と **OR** を使用して、トレース条件の組合せ方法を指定します。リンク条件 **AND** を持つ条件を、リンク条件 **OR** を持つ条件と組み合わせる場合、**AND** が優先します。オプション [反転] はトレース条件を反転させ、各トレースフィルタ条件に対して個別に作用します。あるトレースの開始条件または停止条件が反転されると、他のすべてもそうなります。反転されたトレース開始条件または停止条件は、アプリケーションコードのこのセクションを除いて、トレースデータの収集がすべての場所で実行されることを意味します。

ARM7/9 デバイスの場合、トレースフィルタは **OR** アルゴリズムを使用して結合されます。トレースフィルタを反転するには、[反転] オプションを使用します。すべてのフィルタが対象になります。トレースフィルタは、**AND** アルゴリズムを使用して開始トリガおよび停止トリガと結合されます（存在する場合）。

〔トレース停止〕 ブレークポイントダイアログボックス

〔トレース停止〕ダイアログボックスは、〔ブレークポイント〕ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。または、〔エディタ〕ウィンドウまたは〔逆アセンブリ〕ウィンドウで右クリックして、〔ブレークポイントの切替え（トレース停止）〕を選択することもできます。

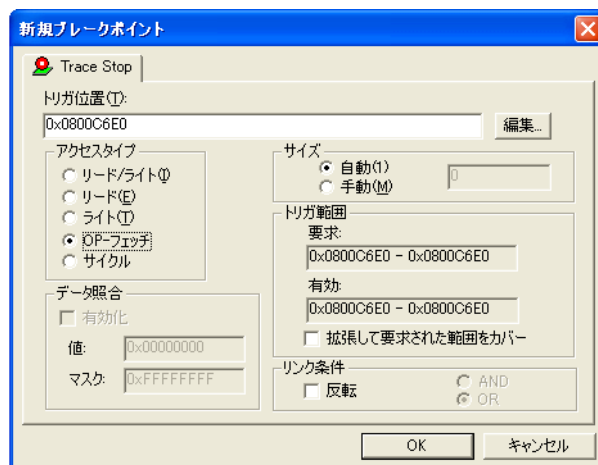


図 88: 〔トレース停止〕ブレークポイントダイアログボックス (J-Link/J-Trace)

トレース条件がトリガされると、何らかのさらなる指示についてトレースデータの収集が実行され、続いて収集が停止します。

このダイアログボックスは、C-SPY J-Link/J-Trace ドライバで使用できます。

トリガ位置

トレースデータを収集するコードセクションの停止点を指定します。変数名やアドレス、サイクルカウンタの値を指定できます。

サイズ

アドレス範囲のサイズを制御します。このサイズに達すると、トレースデータ収集の停止がトリガされます。以下から選択します。

- | | |
|-----------|---|
| 自動 | サイズを自動的に設定します。これは、〔トリガ位置〕に変数が含まれる場合に便利です。 |
| 手動 | ブレークポイント範囲のサイズを手動で指定します。 |

トリガ範囲

要求された範囲とトレースデータの収集でカバーする有効範囲が表示されます。推奨される範囲は、**[トリガ位置]** と **[サイズ]** オプションによって指定された領域とまったく同じか、その内側です。

拡張して要求された範囲をカバー

データ構造体がカバーされるように範囲を拡張します。ハードウェアブレークポイント装置で提供可能な範囲のサイズと合わないデータ構造（3 バイトなど）の場合、範囲はデータ構造全体を対象としません。範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。

このオプションは、ARM7/9 デバイスでは有効になっていません。理由は、こうしたデバイスの範囲は常にデータ構造全体をカバーするためです。

アクセスタイプ

トレースデータの収集をトリガするメモリアクセスのタイプを指定します。以下から選択します。

リード/ライト	指定された位置から読み取り / 書き込みを行います。
リード	指定された位置から読み取ります。
ライト	指定された位置に書き込みます。
OP フェッチ	実行位置のアドレス
サイクル	実行開始点から数えた特定の時点でのカウンタサイクル数。このオプションは、Cortex-M デバイスでのみ使用できます。

データ照合

アクセスされるデータの照合を有効にします。[データ照合] オプションを、リード/ライト、リードまたはライトのデータアクセスタイプと組み合わせて使用します。このオプションは、変数が特定の値を持つときにトリガが必要な場合に便利です。

値	データ値を指定します。
マスク	値のどの部分（ワード、ハーフワード、バイト）を照合するか指定します。

[データ照合] オプションは、J-Link/J-Trace でのみ使用可能です（Cortex-M デバイス使用時のみ）。

注：Cortex-M デバイスについては、1つのブレイクポイントにのみ [データ照合] を設定できます。このようなブレイクポイントでは、2つのブレイクポイントリソースを使用します。

リンク条件

AND と **OR** を使用して、トレース条件の組合せ方法を指定します。リンク条件 **AND** を持つ条件を、リンク条件 **OR** を持つ条件と組み合わせる場合、**AND** が優先します。オプション [反転] はトレース条件を反転させ、各トレースフィルタ条件に対して個別に作用します。あるトレースの開始条件または停止条件が反転されると、他のすべてもそうなります。反転されたトレース開始条件または停止条件は、アプリケーションコードのこのセクションを除いて、トレースデータの収集がすべての場所で実行されることを意味します。

ARM7/9 デバイスの場合、トレースフィルタは **OR** アルゴリズムを使用して結合されます。トレースフィルタを反転するには、[反転] オプションを使用します。すべてのフィルタが対象になります。トレースフィルタは、**AND** アルゴリズムを使用して開始トリガおよび停止トリガと結合されます（存在する場合）。

〔トレースフィルタ〕 ブレークポイントダイアログボックス

〔トレースフィルタ〕 ダイアログボックスは、〔ブレークポイント〕 ウィンドウを右クリックすると表示されるコンテキストメニューから使用できます。または、〔エディタ〕 ウィンドウまたは〔逆アセンブリ〕 ウィンドウで右クリックして、〔ブレークポイントの切替え（トレースフィルタ）〕 を選択することもできます。

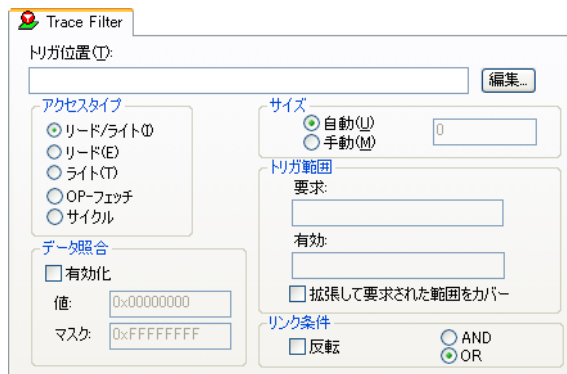


図 89: 〔トレースフィルタ〕 ブレークポイントダイアログボックス

このダイアログボックスは J-Trace ドライバで使用できます。

トレース条件がトリガされると、何らかのさらなる指示についてトレースデータの収集が実行され、続いて収集が停止します。

トリガ位置

トレースデータを収集するコードセクションを指定します。変数名やアドレス、サイクルカウンタの値を指定できます。

サイズ

フィルタトレースを有効にするアドレス範囲のサイズを制御します。以下から選択します。

自動 サイズを自動的に設定します。これは、〔トリガ位置〕に変数が含まれる場合に便利です。

手動 範囲のサイズを手動で指定します。

トリガ範囲

要求された範囲とフィルタされたトレースデータの収集でカバーする有効範囲が表示されます。推奨される範囲は、**[トリガ位置]** と **[サイズ]** オプションによって指定された領域とまったく同じか、その内側です。

拡張して要求された範囲をカバー

データ構造体がカバーされるように範囲を拡張します。ハードウェアブレイクポイント装置で提供可能な範囲のサイズと合わないデータ構造（3 バイトなど）の場合、範囲はデータ構造全体を対象としません。範囲がデータ構造のサイズを超えて拡張され、隣接するデータで誤ったトリガが発生することがある点に注意してください。

このオプションは、ARM7/9 デバイスでは有効になっていません。理由は、こうしたデバイスの範囲は常にデータ構造全体をカバーするためです。

アクセスタイプ

トレースデータの収集を有効にするメモリアクセスのタイプを指定します。以下から選択します。

リード/ライト	指定された位置から読み取り / 書き込みを行います。
リード	指定された位置から読み取ります。
ライト	指定された位置に書き込みます。
OP フェッチ	実行位置のアドレス。
サイクル	実行開始点から数えた特定の時点でのカウンタサイクル数。このオプションは、Cortex-M デバイスでのみ使用できます。

データ照合

アクセスされるデータの照合を有効にします。**[データ照合]** オプションを、リード/ライト、リードまたはライトのデータアクセスタイプと組み合わせて使用します。このオプションは、変数が特定の値を持つときにトリガが必要な場合に便利です。

値	データ値を指定します。
マスク	値のどの部分（ワード、ハーフワード、バイト）を照合するか指定します。

[データ照合] オプションは、J-Link/J-Trace でのみ使用可能です (Cortex-M デバイス使用時のみ)。

注：Cortex-M デバイスについては、1つのブレイクポイントにのみ [データ照合] を設定できます。このようなブレイクポイントでは、2つのブレイクポイントリソースを使用します。

リンク条件

AND と OR を使用して、トレース条件の組合せ方法を指定します。リンク条件 AND を持つ条件を、リンク条件 OR を持つ条件と組み合わせる場合、AND が優先します。オプション [反転] はトレース条件を反転させ、各トレースフィルタ条件に対して個別に作用します。あるトレースの開始条件または停止条件が反転されると、他のすべてもそうなります。反転されたトレース開始条件または停止条件は、アプリケーションコードのこのセクションを除いて、トレースデータの収集がすべての場所で実行されることを意味します。

ARM7/9 デバイスの場合、トレースフィルタは OR アルゴリズムを使用して結合されます。トレースフィルタを反転するには、[反転] オプションを使用します。すべてのフィルタが対象になります。トレースフィルタは、AND アルゴリズムを使用して開始トリガおよび停止トリガと結合されます (存在する場合)。

[トレース式] ウィンドウ

[トレース式] ウィンドウは、[トレース] ウィンドウツールバーから使用できます。



図 90: [トレース式] ウィンドウ

このダイアログボックスは、C-SPY シミュレータで使用できます。

このウィンドウを使用して、トレースデータを収集する特定の変数 (または式)などを指定します。

ツールバー

ツールバーのボタンによって、式の表示順序を変更します。

上向きの矢印 選択された行を上に移動します。

下向きの矢印 選択された行を下に移動します。

表示エリア

表示エリアを使用して、トレースデータを収集する式を指定します。

式 データを収集する元の任意の式を指定します。変数やレジスタなど、評価可能な式を指定できます。

フォーマット 各式で使用される表示フォーマットが表示されます。表示形式はコンテキストメニューから変更できます。

このエリアの各行は、[トレース] ウィンドウに追加列として表示されます。

「トレースを検索」ダイアログボックス

「トレースを検索」ダイアログボックスは、[トレース] ウィンドウで「検索」ボタンをクリックするか、[編集] > [検索と置換] > [検索] を選択すると使用できます。

[編集] > [検索と置換] > [検索] コマンドはコンテキスト依存型であることに、注意してください。このコマンドの操作時の現在のウィンドウが、[トレース] ウィンドウであれば「トレースを検索」ダイアログボックスが表示され、エディタウィンドウであれば「検索」ダイアログボックスがそれぞれ表示されます。

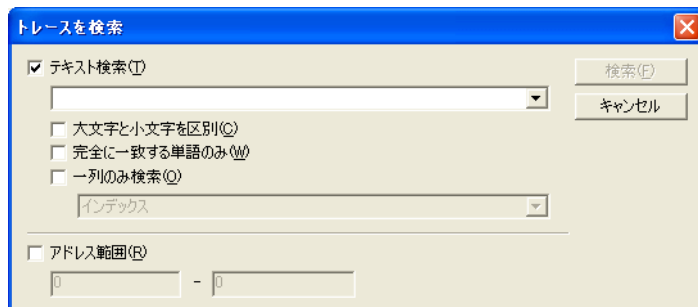


図 91: 「トレースを検索」ダイアログボックス

このダイアログボックスは以下で使用できます。

- C-SPY シミュレータ
- J-Link/J-Trace ドライバ
- ST-LINK ドライバ (SWO が有効な場合)

このダイアログボックスを使用して、トレースデータ内の高度な検索の検索基準を指定します。

検索結果は [トレースを検索] ウィンドウ ([表示] > [メッセージ] コマンドを選択) に表示されます (221 ページの [トレースを検索] ウィンドウを参照)。

関連項目 184 ページの *トレースデータの検索*。

テキスト検索

検索する文字列を指定します。検索基準を以下から選択します。

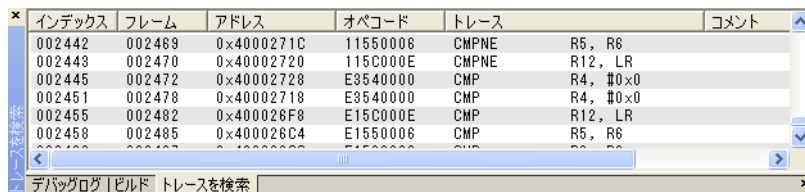
- | | |
|--------------------------|---|
| 大文字 / 小文字の
区別 | 指定されたテキストの大文字と小文字が完全に一致するものだけを検索します。このオプションを指定しない場合は、int を検索すると、INT、Int など検索されます。 |
| 完全に一致する
単語のみ | 単語として一致する箇所だけを検索します。このオプションを指定した場合、int を検索すると、print、sprintf などは検索されません。 |
| 一列のみ検索 | ドロップダウンリストから選択した列だけを検索します。 |

アドレス範囲

表示または検索するアドレス範囲を指定します。アドレス範囲内のトレースデータが表示されます。[テキスト検索] フィールドにテキスト文字列も指定すると、アドレス範囲内でテキスト文字列を検索します。

「トレースを検索」ウィンドウ

「トレースを検索」ウィンドウは、[表示] > [メッセージ] メニューから利用できます。または、[トレースを検索] ダイアログボックスを使用して検索を実行するか、またはエディタウィンドウのコンテキストメニューから「トレースを検索」コマンドを使用して検索を実行すると、自動的に表示されます。



インデックス	フレーム	アドレス	オペコード	トレース	コメント
002442	002469	0x4000271C	11550008	CMPNE	R5, R8
002443	002470	0x40002720	115C000E	CMPNE	R12, LR
002445	002472	0x40002728	E8540000	CMP	R4, #0x0
002451	002478	0x40002718	E8540000	CMP	R4, #0x0
002455	002482	0x400026F8	E16C000E	CMP	R12, LR
002458	002485	0x400026C4	E1550008	CMP	R5, R8

図 92: 「トレースを検索」ウィンドウ

このダイアログボックスは以下で使用できます。

- C-SPY シミュレータ
- J-Link/J-Trace ドライバ
- ST-LINK ドライバ (SWO が有効な場合)

このウィンドウには、トレースデータの検索結果が表示されます。「トレースを検索」ウィンドウで項目をダブルクリックすると、「トレース」ウィンドウに同じ項目が表示されます。

トレースデータを表示するには、「トレースを検索」ダイアログボックスで検索基準を指定する必要があります (219 ページの「トレースを検索」ダイアログボックスを参照)。

詳細については、184 ページの「トレースデータの検索」を参照してください。

表示エリア

「トレースを検索」ウィンドウは「トレース」ウィンドウと非常によく似ており、表示される列とデータは同じですが、表示される行は指定された検索基準に一致した行だけです。

プロファイラの使用

この章では、C-SPY® でのプロファイラの使用方法について説明します。具体的には以下の項目を解説します。

- プロファイラの概要
- プロファイラの使用手順
- プロファイラについてのリファレンス情報

プロファイラの概要

このセクションではプロファイラの概要について説明します。

以下のトピックを解説します。

- プロファイラの用途
- プロファイラの概要について
- プロファイラの使用に関する要件

プロファイラの用途

関数プロファイリングを行うと、実行中に最も多くの時間を費やす、ソースコードでの関数の検索が簡単になります。コードを最適化する際にはこれらの関数に注目します。簡単に関数を最適化するには、実行速度最適化を指定してコンパイルします。

命令プロファイリングは、特にアセンブラのソースコードを非常に詳細なレベルで微調整するときに役立ちます。命令プロファイリングは、C/C++ ソースコードのコンパイルで時間がかかった部分を把握するのに便利です。パフォーマンス向上のために、どのように書き直したらよいかのヒントになります。

プロファイラの概要について

関数プロファイリング情報が [関数プロファイラ] ウィンドウに表示されます。これはアプリケーションでの関数のタイミング情報です。プロファイリングは、ウィンドウのツールバーにあるボタンを使用して明示的に有効にする必要があります。有効にした後は、無効にするまではその状態に保持されます。

命令プロファイリング情報が [逆アセンブル] ウィンドウに表示されます。これは各命令の実行回数です。

ソースのプロファイリング

プロファイラは、さまざまなメカニズムやソースを使用してプロファイリング情報を収集できます。使用可能なトレースソースの特長によっては、1つまたは複数のソースをプロファイリングで使用できます。

- トレース（呼出し）
すべての関数の呼出しとリターンを判定するために、命令のトレース全体（ETM トレース）が解析されます。収集された命令シーケンスが不完全だったり不連続の場合（ETM トレースの使用時に起こることがあります）、プロファイリング情報は正確ではなくなります。
- トレース（フラット）/ サンプリング
命令フルトレース（ETM トレース）または各 PC サンプル（SWO トレースから）の各命令は、関数呼出しやリターンに関係なく、対応する関数またはコードフラグメントに割り当てられます。RTOS を使用していたり、完全なデバッグ情報を持たないコードをプロファイリングする場合など、アプリケーションが通常の呼出し/リターンシーケンスの動作を見せないときにこれは非常に便利です。
- ブレークポイント
プロファイラは、関数のエントリポイントごとにブレークポイントを設定します。実行中に、プロファイラは各ブレークポイントに当たるたびに関数呼出しおよびリターンについての情報を収集します。これは、ハードウェアで大量のブレークポイントがサポートされていることが前提で、実行のパフォーマンスに多大な影響を与えます。

Power サンプリング

一部のデバッグプローブでは、開発ボードまたはボード上のコンポーネントの電力消費のサンプリングがサポートされています。各サンプルは PC サンプルに関連付けられ、サンプル時に先立つわずかな間隔の電力消費（実際のところは電流）を表します。プロファイルが *Power* サンプリングを使用するように設定されている場合、[プロファイラ] ウィンドウに追加の列が表示されます。各 Power サンプルは、通常の PC サンプリングの場合と同じように関数またはコードフラグメントに関連付けられます。ただし、サンプルに対応するすべてのエネルギーが、その関数やコードフラグメントに関するものということではありません。Power サンプルと命令実行のタイムスケールは大きく異なります。電力測定を 1 回行う間に、通常 CPU は命令を数千回実行しています。Power サンプリングは、統計ツールです。

プロファイラの使用に関する要件

C-SPY シミュレータはプロファイラをサポートしており、使用にあたって特定の要件はありません。

ハードウェアデバッガシステムでプロファイラを使用するには、以下の設定のいずれかが必要です。

- プローブとターゲットシステム間の SWD/SWO インタフェースの J-Link、J-Trace、ST-LINK デバッグプローブ（Cortex-M デバイスベース）
- J-Trace デバッグプローブおよび ARM7/9 デバイス（ETM トレース）
- J-Link または J-Trace Ultra プローブ

次の表は、C-SPY ドライバのプロファイリングのサポート一覧です。

C-SPY ドライバ	トレース（呼出し）	トレース（フラット）	サンプリング	ブレイクポイント	電源
C-SPY シミュレータ	X	X	--	--	--
J-Link	--	--	X*	--	--
J-Link Ultra	--	--	X*	--	X
J-Trace	X	X	X*	--	--
RDI	--	--	--	--	--
Macraigor	--	--	--	--	--
GDB サーバ	--	--	--	--	--
ST-LINK	--	--	X	--	--
TI Stellaris FTDI	--	--	--	--	--
Angel	--	--	--	--	--
IAR ROM モニタ	--	--	--	--	--

表 11: C-SPY ドライバのプロファイリングのサポート

* Cortex-M デバイスの場合のみ。

プロファイラの使用手順

このセクションでは、プロファイラの使用方法についてステップごとに説明します。

具体的には、以下の項目について説明します。

- 関数レベルでプロファイラを使用するにあたって
- 命令レベルでプロファイラを使用するにあたって
- プロファイリング情報の間隔を選択する


関数レベルでプロファイラを使用するにあたって

関数プロファイリング情報を [関数プロファイラ] ウィンドウに表示するには：

- 1 以下のオプションを使用してアプリケーションをビルドします。

カテゴリ	設定
C/C++ コンパイラ	[出力] > [デバッグ情報の生成]
リンカ	[出力] > [出力ファイルにデバッグ情報を含める]

表 12: プロファイラを有効にするためのプロジェクトオプション

- 2 関数プロファイリングのプロファイラを設定するには、以下の手順を実行します。
 - ETM トレースを使用する場合、[トレース設定] ダイアログボックスで [サイクルアキュレート・トレース] オプションが選択されていることを確認します。
 - SWD/SWO インタフェースを使用する場合、設定は特に必要ありません。
- 3 アプリケーションをビルドして C-SPY を起動した後、[J-Link]> [関数プロファイラ] を選択して [関数プロファイラ] ウィンドウを開き、[有効化] ボタンをクリックして、プロファイラを有効にします。または、[関数プロファイラ] ウィンドウを右クリックして表示されるコンテキストメニューで [有効化] を選択します。
- 4 アプリケーションの実行を開始して、プロファイリング情報を収集します。
- 5 プロファイリング情報が [関数プロファイラ] ウィンドウに表示されます。ソートするには、対象の列見出しをクリックします。
- 6  新しいサンプリングを開始する場合は、[クリア] ボタンをクリックして (またはコンテキストメニューを使用して)、データをクリアします。

命令レベルでプロファイラを使用するにあたって

命令プロファイリング情報を [逆アセンブリ] ウィンドウに表示するには：

- 1 アプリケーションをビルドして C-SPY を起動したら、[表示] > [逆アセンブリ] を選択して [逆アセンブリ] ウィンドウを開き、プロファイラのウィンドウを右クリックすると表示されるコンテキストメニューで [有効化] を選択します。
- 2 プロファイリング情報を表示するために、コンテキストメニューで [表示] コマンドが選択されていることを確認します。

- 3 アプリケーションの実行を開始して、プロファイリング情報を収集します。
- 4 プログラムの終了に到達した、ブレークポイントがトリガされたなどの理由で実行が停止したときは、[逆アセンブリ] ウィンドウの左側の余白で命令レベルのプロファイリング情報を確認できます。

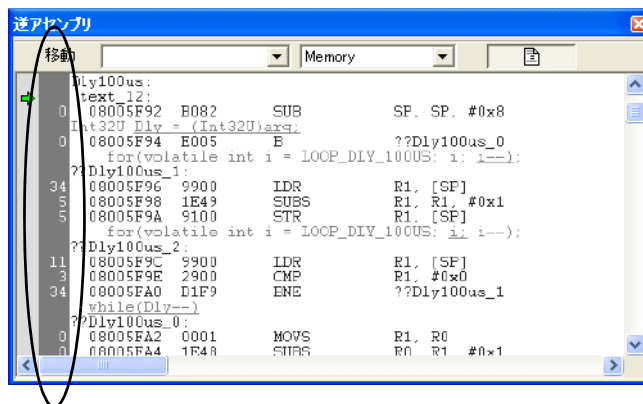


図 93: [逆アセンブリ] ウィンドウの命令カウント

命令ごとに、実行された回数が表示されます。

命令プロファイリングでは、関数プロファイラと同じソースを使用するよう試みます。関数プロファイラがオンでない場合、命令プロファイラは最初のトレースを使用してから、PC サンプリングをソースとして使用するよう試みます。プロファイラのウィンドウで使用できるコンテキストメニューから、使用するソースを変更できます。

プロファイリング情報の間隔を選択する

通常プロファイラは、受け取るすべての PC サンプルから情報を算出し、プロファイリング情報を明示的に消去するまで情報を蓄積します。ただし、プロファイラが PC サンプルを算出する間隔を選択できます。この機能は、J-Link プローブおよび J-Trace プローブ、ST-LINK プローブでサポートされています。

時間間隔を選択するには：

- 1 [J-Link] メニューで [関数プロファイラ] を選択します。
- 2 [関数プロファイラ] ウィンドウで右クリックして、コンテキストメニューから [ソース: サンプリング] を選択します。
- 3 アプリケーションを実行してサンプルを収集します。
- 4 [表示] > [タイムライン] を選択します。

5 [タイムライン] ウィンドウでクリックし、ドラッグして間隔を選択します。

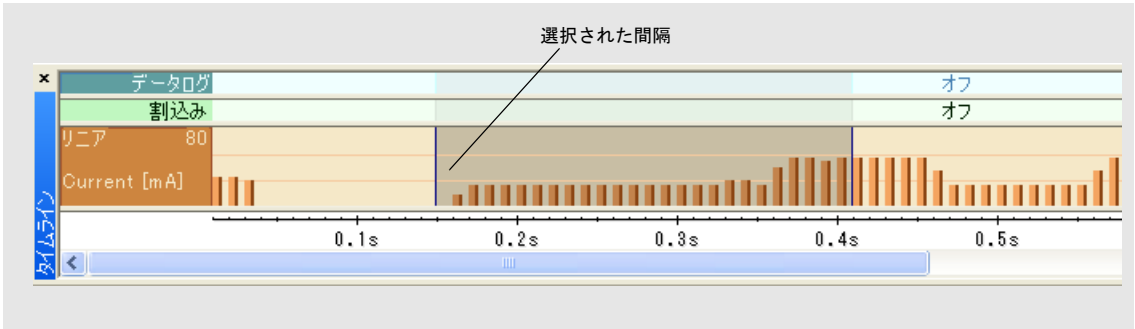


図 94: Power グラフで選択された間隔

6 選択された間隔で、右クリックしてコンテキストメニューから [プロファイラの選択] を選択します。

[関数プロファイラ] ウィンドウに選択した間隔のプロファイリング情報が表示されます。

The screenshot shows the '関数プロファイラ' (Function Profiler) window. It displays a table of profiling data for the selected interval. The table has columns for '関数' (Function), 'PCサンプル' (PC Samples), 'PCサンプル (%)' (PC Samples (%)), 'Power サンプル' (Power Samples), 'エネルギー (%)' (Energy (%)), and '平均' (Average). The 'main()' function is highlighted in blue.

関数	PCサンプル	PCサンプル (%)	Power サンプル	エネルギー (%)	平均
GetButtons()	791	33.10	9	30.82	19.1
Dly100us(void *)	463	19.37	7	15.38	12.0
GLCD_SPI_TranserByte(Int3...	353	14.77	4	8.32	12.0
memcpy	325	13.60	4	14.64	21.0
main()	288	12.05	6	20.07	19.1
GLCD_Backlight(Int8U)	108	4.52	2	6.77	19.1
GLCD_SendCmd(GLCD_Cm...	43	1.80	0	0.00	-
GLCD_SPI_SendBlock(plnt8...	19	0.79	2	4.00	11.0
GLCD_SetWindow(Int32U, Int...	0	0.00	0	0.00	-
GLCD_SetReset(Boolean)	0	0.00	0	0.00	-

図 95: 間隔モードの [関数プロファイラ] ウィンドウ

7 [フル/間隔プロファイリング] ボタンをクリックして、フルプロファイリング表示を切り替えます。

プロファイラについてのリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 229 ページの [関数プロファイラ] ウィンドウ
- 86 ページの [逆アセンブリ] ウィンドウ

関連項目：

- 186 ページの [ETM トレース設定] ダイアログボックス
- 188 ページの [SWO トレースウィンドウ設定] ダイアログボックス
- 190 ページの [SWO 設定] ダイアログボックス

[関数プロファイラ] ウィンドウ

[関数プロファイラ] ウィンドウは、[シミュレータ] メニュー、[J-Link] メニュー、または [ST-LINK] メニューから使用できます。



関数	呼出し	フラット時間	フラット時間(%)	累積時間	累積時間(%)
<その他>	0	98	13.86	0	0.00
do_foreground_process()	0	0	0.00	0	0.00
get_fib(int)	16	320	45.26	320	45.26
init_fib()	1	163	23.06	467	66.05
main()	1	7	0.99	7	0.99
next_counter()	0	0	0.00	0	0.00
put_fib(unsigned int)	0	0	0.00	0	0.00

図 96: [関数プロファイラ] ウィンドウ

このウィンドウは以下で使用できます。

- C-SPY シミュレータ
- J-Link/J-Trace ドライバ
- ST-LINK ドライバ

このウィンドウには関数プロファイラの情報が表示されます。

ツールバー

ツールバーの内容は以下のとおりです。



有効 / 無効

プロファイラを有効 / 無効にします。



クリア

すべてのプロファイリングデータをクリアします。



保存

標準の **[名前を付けて保存]** ダイアログボックスを開きます。ウィンドウの内容をタブ区切りでファイルに保存できます。展開されていない行のみがリストファイルに含まれます。



グラフィック表示

パーセント値を表す列の値をグラフィカルバーにオーバーレイします。

進行度バー

処理中のプロファイリングデータのバックログを表示します。受信データのレートが、データを処理するプロファイラのレートより高い場合は、バックログが蓄積されます。進行度バーは、プロファイラがデータ処理を実行中であることを示します。また、プロファイラがどのぐらいの速度で処理中か、おおよその速度も示します。プロファイラは特定のレートでデータを処理し、ターゲットシステムは違うレートでデータを提供するため、処理待ちのデータ量は増減する可能性があることに注意してください。進行度バーは、それに従って伸縮します。



間隔モード

選択した間隔のプロファイリングと完全なプロファイリングを切り替えます。このツールバーのボタンは、PC サンプリングがデバッグプローブでサポートされている場合にのみ使用できます。

使用する C-SPY ドライバでどのビューがサポートされているかについては、225 ページの *プロファイラの使用に関する要件* を参照してください。

**ステータス
フィールド**

選択した間隔の範囲が表示されます。すなわち、プロファイルされた選択範囲です。間隔プロファイリングモードが有効な場合、このフィールドは黄色です。このフィールドは、PC サンプリングがデバッグプローブ (SWO トレース) でサポートされている場合にのみ使用できます。

使用する C-SPY ドライバでどのビューがサポートされているかについては、225 ページの *プロファイラの使用に関する要件* を参照してください。

表示エリア

表示エリアの内容は、プロファイリング情報に使用されるソースによって決まります。

- ブレークポイントとトレース（呼出し）ソースの場合は、各行に、有効なデバッグ情報でコンパイルされた各関数が表示されます。複数のプロファイリング情報が収集された場合は、別の関数を呼び出した関数の行を展開できます。特定の関数の子要素には、その親関数によって呼び出されたすべての関数と、関連する統計がリストされます。
- サンプリングおよびトレース（フラット）ソースの場合は、各行にアプリケーションの C 関数がそれぞれ表示されます。ランタイムライブラリからのコードまたはデバッグ情報なしの別コードからのセクションが、対応するアセンブラベルによって示されたもののみ、表示されます。トレースデータからの実行済 PC アドレスは、個別のサンプルとして扱われ、[プロファイリング] ウィンドウで対応する行に関連付けられます。各行には、これらのサンプル数が含まれます。

使用する C-SPY ドライバでどのビューがサポートされているかについては、225 ページの *プロファイラの使用に関する要件*を参照してください。

表示エリアには以下の情報が表示されます。

関数	すべてのソース	プロファイルされた C 関数の名前。 サンプリングソースの場合は、ランタイムライブラリからのコードまたはデバッグ情報なしの別コードからのセクションが、対応するアセンブラベルによって示されたもののみ、表示されます。
呼出し	ブレークポイントおよびトレース（呼出し）	関数の呼出し回数。
フラット時間	ブレークポイントおよびトレース（呼出し）	関数内で消費された時間（サイクル数）。
フラット時間 (%)	ブレークポイントおよびトレース（呼出し）	合計時間の割合で表されたフラット時間。
累積時間	ブレークポイントおよびトレース（呼出し）	この関数およびこの関数によるすべての呼出しで消費された時間（サイクル数）。
累積時間 (%)	ブレークポイントおよびトレース（呼出し）	合計時間の割合で表された累計時間。

PC サンプル	トレース（フラット） およびサンプリング	関数に関連する PC サンプルの数。
PC サンプル (%)	トレース（フラット） およびサンプリング	関数に関連付けられた PC サンプルの数 (サンプル総数に対するパーセント値)。
Power サン プル	Power サンプリング	関数に関連する Power サンプルの数。
エネルギー (%)	Power サンプリング	関数に関連するすべての測定値の累 計。全測定値に対するパーセントで 表します。
平均電流 [mA]	Power サンプリング	関数に関連するすべてのサンプルの平 均測定値。
最小電流 [mA]	Power サンプリング	関数に関連するすべてのサンプルの最 小測定値。
最大電流 [mA]	Power サンプリング	関数に関連するすべてのサンプルの最 大測定値。

コンテキストメニュー

以下のコンテキストメニューがあります。

✓ 有効化
クリア
✓ ソース: サンプリング
✓ Powerサンプリング

図 97: [関数プロファイラ] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

有効化	プロファイラを有効にします。ウィンドウを閉じるときにも情報が記録されます。
クリア	すべてのプロファイリングデータをクリアします。
ソース*	プロファイリング情報に使用するソースを選択します。以下から選択します。 サンプリング — 命令プロファイリングの命令カウントは、各命令のサンプル数を示します。 トレース（呼出し） — 命令プロファイリングの命令カウントは、トレースデータ収集時の完了した数のみを示します。 トレース（フラット） — 命令プロファイリングの命令カウントは、トレースデータ収集時の完了した数のみを示します。
Power サンプリング	Power サンプリング情報の有効 / 無効を切り替えます。このコマンドは、J-Link および J-Trace Ultra プローブでサポートされています。

* 使用する C-SPY ドライバによって、使用可能なソースが決まります。

使用する C-SPY ドライバでどのビューがサポートされているかについては、225 ページの *プロファイラの使用に関する要件* を参照してください。

Power ドメインのデバッグ

この章では、Power デバッグのテクニックと、予期しない電力消費をもたらすソースコード構造を C-SPY® を使用して発見する方法を説明します。具体的には以下の項目を解説します。

- Power デバッグの概要
- 電力消費のソースコードの最適化
- Power デバッグの手順
- Power デバッグのリファレンス情報

Power デバッグの概要

このセクションでは、以下のトピックについて説明します。

- Power デバッグを使用する理由
- Power デバッグの概要
- Power デバッグの要件

POWER デバッグを使用する理由

バッテリー寿命の長さは、医療や家電、ホームオートメーションなど、ほとんどすべての市場区分において、多くの組込みシステムで非常に重要な要素です。これらのシステムの消費電力は、ハードウェアの設計だけでなく、ハードウェアの使用方法によっても異なります。システムソフトウェアは、使用方法を制御します。

Power デバッグが役に立つ例については、237 ページの *電力消費のソースコードの最適化* をご覧ください。

POWER デバッグの概要

Power デバッグは、消費電力（より正確に言うと、CPU と周辺ユニットによって消費される電力）をサンプリングし、それぞれのサンプルをアプリケーションの命令シーケンスと関連付けて、それからプログラム実行におけるソースコードやさまざまなイベントと関連付けます。

従来からソフトウェア設計の主なゴールは、使用するメモリをなるべく少なくすることです。しかし、アプリケーションの消費電力をソースコードに関連付けることで、ソフトウェアが消費電力にどのように影響するのか理解して、電力の消費を最小限にする方法を考えることができます。

消費電力の測定

消費電力はデバッグプローブによって測定します。J-Link/J-Trace Ultra デバッグプローブは、デバイスへの供給電力に直列の小さい抵抗（シャント抵抗）に対する電圧の低下を測定します。電圧の低下は差動増幅器で測定され、続いて AD コンバータによってサンプリングされます。

しきい値と、しきい値に達したときに実行される適切なアクションを指定できます。つまり、電力測定を有効または無効にしたり、アプリケーションの実行を停止して、予期しない電力値の原因を特定することができます。

C-SPY を使用した Power デバッグ

C-SPY は Power デバッグを設定するインタフェースとなるほか、電力の値を参照するウィンドウのセットを提供します。

- [Power 設定] ウィンドウでは、しきい値およびしきい値に達したときに実行されるアクションを指定できます。
- [Power ログ] ウィンドウには、記録された電力の値がすべて表示されます。このウィンドウは Power ロギングのピークを探すときに使用できます。値は実行されたコードに関連付けられているため、[Power ログ] ウィンドウの値をダブルクリックすれば、対応するコードを取得できます。精度はサンプルの周波数によって異なりますが、かなりの確率でピークの原因となったソースコードのシーケンスを見つけられます。
- [タイムライン] ウィンドウには、時系列で電力の値が表示されます。これは、ウィンドウに表示される他の情報と比較しながら消費電力を参照する便利な方法です。[タイムライン] ウィンドウは [Power ログ] ウィンドウと [ソースコード] ウィンドウ、[逆アセンブリ] ウィンドウに関連付けられており、タイムライン上の値に対応するソースコードがダブルクリックするだけで見つかります。
- [関数プロファイラ] ウィンドウは、関数プロファイリングと Power ロギングを組み合わせ、関数ごとの電力消費、つまり電力プロファイリングを表示します。関数別の値のリストのほか、最大値と最小値とともに平均値も得られます。こうすることで、電力消費を最適化する際に集中すべきアプリケーションの領域を発見します。

POWER デバッグの要件

C-SPY の機能を Power デバッグに使用するには、以下が必要です。

- SWO を持った Cortex-M3 デバイス
- J-Link デバッグプローブまたは J-Link Ultra デバッグプローブ。J-Link プローブの精度は非常に限られており、分解能も低い点に注意してください

電力消費のソースコードの最適化

ここでは、Power デバッグが役に立つ例をいくつか紹介して、低消費電力のために最適化できるソースコード構造を特定しやすくするのが狙いです。

デバイスのステータスの待機

不要な電力消費の原因となりうる一般的な構造は、たとえば周辺デバイスなどのステータス変更を待つためにポーリングを使用することです。次の例にある構造体は、ステータスの値が予想される状態になるまで中断なしに実行されます。

```
while (USBD_GetState() < USBD_STATE_CONFIGURED);
while ((BASE_PMC->PMC_SR & MC_MCKRDY) != PMC_MCKRDY);
```

電力消費を最小限にするには、デバイスのステータス変更のポーリングを記述し直して、ポーリングしていないときに CPU がスリープになれるように、割り込み、または可能であればタイマ割り込みを使用することです。

ソフトウェア遅延

ソフトウェア遅延は、たとえば次のように for または while ループとして実装できます。

```
i = 10000; /* ソフトウェア遅延 */
do i--;
while (i != 0);
```

このようなソフトウェア遅延は、時間を浪費する以外に目的のない命令の実行で CPU を稼動状態のままにします。時間の遅延は、ハードウェアタイマを使用して実装した方がずっと効率的です。タイマ割り込みはを設定した後は、CPU は割り込みによって稼動するまで低電力モードになります。

DMA とポーリングされた I/O の比較

これまで DMA は、転送速度の増加に使用されてきました。MCU の場合、柔軟性や速度を高めたり、消費電力を抑える DMA のテクニックはたくさんあります。時には、DMA 転送中に CPU をスリープモードにすることもできま

す。Power デバッグを使用すると、従来の CPU 主体のポールソリューションに対して、これらの DMA テクニックが消費電力に与える影響を直接デバッグで実験して確認できます。

低電力モードの診断

多くの組込みアプリケーションでは、ほとんどの時間を何かが起こるまで待機して過ごします。シリアルポートでのデータ受信や、I/O ピンの状態の変更を観察したり、時間の遅延が期限切れとなるまで待機するなどです。プロセッサが待機中にまだフルスピードで実行中であれば、ほぼ何も達成されていないにも関わらずバッテリーが消費されます。そのため、多くのアプリケーションでは、マイクロプロセッサは非常にわずかの時間だけアクティブになり、待機時間は低電力モードにすることで、バッテリー寿命を格段に長くすることができます。

タスク指向の設計を行って RTOS を使用するのが、懸命なアプローチです。タスク指向の設計では、タスクは最も優先度を低く定義できます。実行するタスクが他にないときにだけ実行されます。この待機タスクは、電力管理を導入する完璧な場所です。実際には、待機タスクがアクティブになるたびに、マイクロプロセッサが低電力モードに設定されます。多くのマイクロプロセッサおよびシリコンデバイスには、たくさんの低電力モードがあり、不要なときにマイクロプロセッサの異なる部分をオフにできます。たとえば、オシレータはオフにするか、低周波数に切り替えることができます。また、個々の周辺ユニットやタイマ、CPU は停止可能です。異なる低電力モードでは、オンのままになっている周辺ユニットに基づいて電力消費が異なります。Power デバッグツールは、異なる低電力モードで実験を行うときに非常に便利です。

C-SPY の関数プロファイラを使用して、異なる低電力モードが使用されたときにシステムを低電力モードにするタスクや関数の電力測定を比較できます。比較では平均値と合計消費電力のパーセント値がどちらも役に立ちます。

CPU 周波数

CMOS MCU の電力消費は、理論的には次の公式により算出されます。

$$P = f * V^2 * k$$

f はクロック周波数、 V は供給電圧、 k は定数です。

Power デバッグを使用すると、クロック周波数の係数として電力消費を検証できます。50 MHz でほとんどスリープモードの時間がないシステムは、100 MHz で実行した場合に 50% の時間がスリープモードになることが予想されます。C-SPY で収集された電力データを使用して予想される動作を検証し、クロック周波数にリニアでない依存性がある場合は、最も消費電力の少ない動作周波数を選択するようにしてください。

誤ってアンアテンドになっている周辺ユニットの検出

周辺ユニットは、頻繁に使用されていない場合でも大量の電力を消費することがあります。低電力を考えて設計する場合、使用していないときは周辺ユニットを無効にして、アンアテンドのままにしないことが重要です。ただし、さまざまな理由で周辺ユニットの電源供給をオンのままにすることがあります。慎重で正しい設計上の決定のこともあれば、不十分な設計か単なるミスの可能性もあります。前者の場合でなければ、予想を上回る電力がアプリケーションで消費されることになります。このことは、[タイムライン] ウィンドウの **Power** グラフで簡単に分かります。[タイムライン] ウィンドウで電力消費が予想外に高い部分をダブルクリックすると、対応するソースコードと逆アセンブリコードに移動します。ほとんどの場合、アクティブでないときに周辺ユニットを無効にするだけで十分です。たとえば、クロックをオフにすれば、たいていの場合は電力の消費が完全に停止します。

ただし、クロックのゲートだけでは不十分な場合もいくつかあります。コンバータやコンパレータなどアナログの周辺ユニットは、クロックがオフの場合でもかなりの電力を消費します。[タイムライン] ウィンドウでは、クロックをオフにするだけでは不十分で、周辺ユニットを完全にオフにする必要があることが示されます。

イベント駆動型システムでの周辺ユニット

実行中にあるタスクがアナログコンパレータを使用し、そのタスクがより優先度の高いタスクによって停止される場合のシステムを考えてください。理想的には、タスクが停止されたときにコンパレータがオフになり、タスクが再開したときに再びオンになるべきです。こうすれば、優先度の高いタスクの実行中に、消費される電力を最小限に抑えられます。

これはイベント駆動型を想定したシステムを電力消費の回路図で、 t_0 時点でシステムは非アクティブモードにあり、電流は I_0 です。

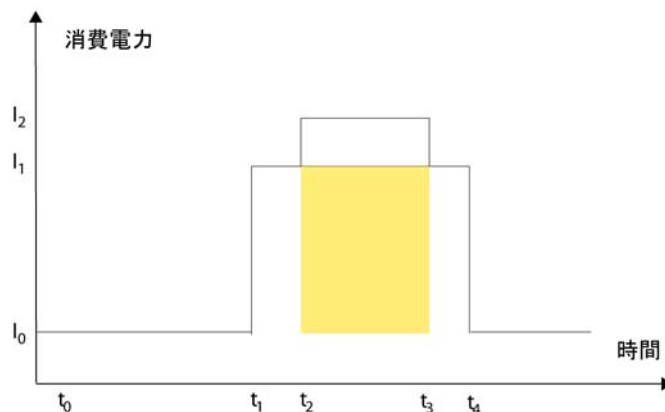


図98: イベント駆動型システムでの電力消費

t_1 の時点ではシステムがアクティブになり、電流は少なくとも1つの周辺デバイスがオンになっているアクティブ時のシステムの消費電力である I_1 に上がって、電流は I_1 に上がります。 t_2 では、優先度の高い割込みによって実行が停止します。すでにアクティブだった周辺デバイスは、優先度の高いタスクで使用されないに関わらず、オフになっていません。その代わりに、新しいタスクによってさらに周辺デバイスがアクティブになり、制御が優先度の低いタスクに戻る t_2 から t_3 の間に電流が I_2 に上がります。

システムの機能は申し分なく、速度とコードサイズの面では最適化が可能です。しかし、Power ドメインでもさらなる最適化が行えます。重なっているエリアは、 t_2 と t_3 の間で使用されない周辺デバイスをオフにしたり、2つのタスクの優先度が変わった場合に、節約できたエネルギーを表します。

[タイムライン] ウィンドウを使用すると、綿密な調査を行って、使用されていない周辺デバイスがアクティブになって、不要に長い時間にわたり電力を消費していたことを明らかにできます。当然ながら、例のような状況で、追加のクロックサイクルを使用して周辺デバイスをオンやオフにする価値があるかどうかを考えなければなりません。

衝突するハードウェア設定の検出

フローティング入力を回避するため、使用されていない MCU I/O ピンを接地するのが一般的な設計上の慣習です。誤ってソースコードで接地された I/O ピンのいずれかを論理的 1 出力として設定した場合、そのピンで高電流が失われる可能性があります。この予想外の高電流は、[タイムライン] ウィンドウの Power グラフから電流の値を読み取れば、簡単に観測できます。対応する間違った初期化コードも、アプリケーション起動時の Power グラフを見れば発見できます。

ある I/O ピンが入力として設計されて外部の回路によって駆動するにも関わらず、コードで誤って入力ピンを出力として設定した場合、同じような状況が発生します。

アナログ干渉

同じボード上でアナログとデジタルの回路を混在させる場合、ボードのレイアウトとルーティングがアナログのノイズレベルに影響することがあります。低レベルのアナログ信号の正確なサンプリングを確実に行うため、ノイズレベルを低く保つことが重要です。効率的に混在した信号設計を実現するには、ハードウェアを慎重に考慮する必要があります。また、ソフトウェア設計がアナログ測定の質に影響することもあります。アナログ信号のサンプリングと同時に I/O アクティビティを大量に実行すると、多くのデジタル線で状態が同時に切り替わり、AD コンバータにさらなるノイズが追加されることがあります。

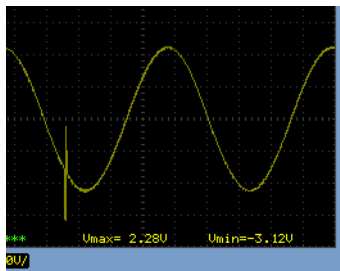


図 99: オシロスコープにより記録されたノイズの上昇

Power デバッグは、アナログ部品へのデジタルおよび電力供給線からの干渉を調べる上で役に立ちます。AD 変換の付近での電力の上昇はノイズの発生源かもしれません、調査する必要があります。[タイムライン] ウィンドウにあるすべてのデータは、実行されたコードに関連付けられています。疑問を感じる電力の値をダブルクリックするだけで、対応する C のソースコードが表示されます。

Power デバッグの手順

ここでは、Power デバッグに関連する機能の使用方法を手順ごとに説明します。

具体的には、以下の項目について説明します。

- 電力プロファイルの表示と結果の分析
- アプリケーション実行中の予想外の電力消費の検出

関連項目：

- 200 ページの [タイムライン] ウィンドウ
- 227 ページの プロファイリング情報の間隔を選択する

電力プロファイルの表示と結果の分析

電源プロファイルを表示するには：

- 1 ドライバのメニュー> [SWO 設定] を選択して、[SWO 設定] ダイアログボックスを開きます。[CPU クロック] オプションが、アプリケーションの CPU クロックの値と同じに設定されているか確認してください。SWO クロックを設定して、デバッグプローブへ正しいデータ転送を行うには、こうする必要があります。
- 2 ドライバのメニュー> [タイムライン] を選択して、[タイムライン] ウィンドウを表示します。
- 3 グラフエリアで右クリックして、コンテキストメニューから [有効化] を選択し、Power グラフを有効にします。
- 4 ドライバのメニュー> [Power ログ] を選択して、[Power ログ] ウィンドウを開きます。
- 5 オプションで、電力の値を特定の割込みや変数に関連付ける場合は、割込みまたはデータロググラフのエリアをそれぞれ右クリックして、コンテキストメニューから [有効化] を選択します。
変数の場合は、[タイムライン] ウィンドウでグラフィック表示を行う各変数にデータログブレークポイントを設定する必要もあります。145 ページの [データログ] ブレークポイントダイアログボックスを参照してください。
- 6 オプションで、アプリケーションの実行を開始する前に、グラフの Y 軸の表示範囲を設定できます。206 ページの [表示範囲] ダイアログボックスを参照してください。
- 7 ツールバーで [Go] をクリックして、アプリケーションの実行を開始します。
[Power ログ] ウィンドウに、すべての電力の値が表示されます。[タイムライン] ウィンドウでは電力の値がグラフィック表示され、これらのグラフを有効にした場合はデータおよび割込みのログも表示されます。グラフでのナ

ビゲートの方法については、200 ページの [タイムライン] ウィンドウを参照してください。

8 電力消費を分析するには：

- 関心のある電力の値をダブルクリックすると、対応するソースコードがエディタウィンドウと [逆アセンブリ] ウィンドウで強調表示されます。対応するログは、[Power ログ] ウィンドウで強調表示されます。これが役に立つ例については、237 ページの *電力消費のソースコードの最適化* をご覧ください。
- 使用していないときに無効にできる周辺ユニットを特定できます。これは、Power グラフと [タイムライン] ウィンドウの他のグラフを組み合わせることで分析すれば検出できます。239 ページの *誤ってアンアテンドになっている周辺ユニットの検出* も参照してください。
- 特定の割込みについては、割込みの終了後に電力消費が予想外に変化したかどうかを確認できます。たとえば、割込みによって電力消費の大きいユニットが有効になり、終了する前にオフにしない場合などです。
- 関数プロファイリングについては、227 ページの *プロファイリング情報の間隔を選択する* を参照してください。

アプリケーション実行中の予想外の電力消費の検出

予想外の電力消費を検出するには：

- 1 ドライバのメニュー > [SWO 設定] を選択して、[SWO 設定] ダイアログボックスを開きます。[CPU クロック] オプションが、アプリケーションの CPU クロックの値と同じに設定されているか確認してください。SWO クロックを設定して、デバッグプローブへ正しいデータ転送を行うには、こうする必要があります。
- 2 ドライバのメニュー > [Power 設定] を選択して、[Power 設定] ウィンドウを開きます。
- 3 [Power 設定] ウィンドウで、しきい値と適切なアクションを指定します。たとえば、[すべてをログしてしきい値以上で CPU を停止] などです。
- 4 ドライバのメニュー > [Power ログ] を選択して、[Power ログ] ウィンドウを開きます。電力の値を継続的にファイルに保存する場合は、コンテキストメニューから [ライブログファイルの選択] を選択します。この場合、[指定先へのライブログを有効にする] も選択する必要があります。
- 5 実行を開始します。

消費電力がしきい値を超えたとき、実行が停止して指定したアクションが行われます。

記録した電力の値をファイルに保存した場合、外部ツールでそのファイルを開いて、さらに分析することが可能です。

Power デバッグのリファレンス情報

ここでは、Power デバッグに関連するウィンドウやダイアログボックスのリファレンス情報を提供します。

- 244 ページの [Power ログ設定] ウィンドウ
- 246 ページの [Power ログ] ウィンドウ

関連項目：

- 193 ページの [トレース] ウィンドウ
- 200 ページの [タイムライン] ウィンドウ
- 206 ページの [表示範囲] ダイアログボックス
- 229 ページの [関数プロファイラ] ウィンドウ

[Power ログ設定] ウィンドウ

[Power ログ設定] ウィンドウは、デバッグセッション中に C-SPY ドライバのメニューから使用できます。

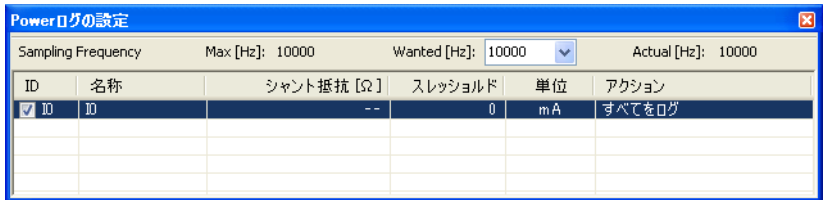


図 100: [Power 設定] ウィンドウ

このウィンドウは以下で使用できます。

- J-Link ドライバ

このウィンドウを使用して、電力の測定を設定します。

注：Power ログgingを有効にするには、[Power ログ] ウィンドウのコンテキストメニューまたは [タイムライン] ウィンドウの Power ロググラフのコンテキストメニューから **[有効化]** を選択します。

表示エリア

このエリアには以下の列が含まれます。

ID	ブローブの測定チャンネルを識別する一意の文字列。 このチェックボックスを使用して、チャンネルをアク ティブにします。このチェックボックスの選択を解除 すると、そのチャンネルのログは生成されません。
名称	ユーザ定義名を指定します。
シャント抵抗 [Ohm]	この列の内容は常に -- となります。
しきい値	選択した単位でしきい値を指定します。しきい値に 達したときに、指定したアクションが実行されます。
単位	電力（電流）の表示単位を選択します。 nA 、 uA 、 mA から選択してください。
アクション	測定チャンネルでどのアクションが選択されたかを 表示します。以下から選択します。 すべてをログ 、 しきい値以上をログ 、 しきい値未満をログ 、 すべて をログしてしきい値以上で CPU を停止 、 すべてをロ グしてしきい値未満で CPU を停止 。

コンテキストメニュー

以下のコンテキストメニューがあります。

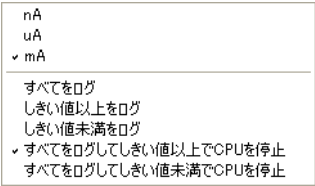


図 101: [Power 設定] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

nA 、 uA 、 mA	電力（電流）の表示単位を選択します。これら は、電力を測定するチャンネルで使用できます。
すべてをログ	すべての値を記録します。
しきい値以上をログ	しきい値を超えたすべての値を記録します。

しきい値未満をログ	しきい値未満の値をすべて記録します。
すべてをログしてしきい値以上で CPU を停止	すべての値を記録します。記録された値がしきい値を超える場合、実行が停止します。
すべてをログしてしきい値未満で CPU を停止	すべての値を記録します。記録された値がしきい値未満の場合、実行が停止します。

[Power ログ] ウィンドウ

[Power ログ] ウィンドウは、デバッグセッション中に C-SPY ドライバのメニューから使用できます。



時間	プログラムカウンタ	Current [mA]
315873.537us	0x1FFFC3BC	90
333485.262us	0x1FFFC3BC	88
350482.625us	0x1FFFC3BC	88
369937.438us	0x1FFFC37A	77
397993.325us	0x1FFFC3BC	88
397993.325us	0x1FFFC3BC	88
936902.950us	0x1FFFC380	88
992400.362us	0x1FFFC3A8	88
1s 15336.562us	0x1FFFC3A8	88

図102: [Power ログ] ウィンドウ

このウィンドウは以下で使用できます。

- J-Link ドライバ

このウィンドウには、収集された電力の値が表示されます。

時間 / サイクルとプログラムカウンタのみ灰色で表示された行は、[Power 設定] ウィンドウで実際のデータ収集時にアクティブで現在は無効になっているチャンネルについて記録された電力の値を示します。

注：記録される電力の値の数には制限があります。この制限を超過すると、バッファの最初のエントリが消去されます。

表示エリア

このエリアには以下の列が含まれます。

時間	<p>[SWO 設定] ダイアログボックスで指定したクロック周波数に基づく、アプリケーションのリセットからイベントまでの時間。</p> <p>ターゲットシステムが正確な時間を収集できなかった場合は、おおよその時刻が斜体で表示されます。</p> <p>この列は、コンテキストメニューから [時間表示] を選択した場合に有効になります。</p>
サイクル	<p>アプリケーションのリセットからイベントまでのサイクル数。この情報は、リセットでクリアされます。</p> <p>ターゲットシステムが正確な時間を収集できなかった場合は、おおよそのサイクルが斜体で表示されます。</p> <p>この列は、コンテキストメニューから [サイクル表示] を選択した場合に有効になります。</p>
プログラムカウンタ	<p>以下のいずれかが表示されます。</p> <p>PC の内容であるアドレス。つまり、電力の値が収集されたポイントに近い命令のアドレスです。</p> <p>---、ターゲットシステムがデバッガに情報を提供できなかったことを示します。</p> <p>赤色で Overflow と表示されている場合、通信チャンネルがすべてのデータをターゲットシステムから送信できなかったことを示します。</p> <p>待機、電力の値は待機モード中に記録されます。</p>
名称 [単位]	<p>[Power 設定] ウィンドウで指定した単位で表される電力の測定値。</p>

コンテキストメニュー

以下のコンテキストメニューがあります。

• 有効化 クリア ログファイルに保存...
ライブログファイルの選択... • 'PowerLogLive.log'へのライブログを有効にする 'PowerLogLive.log'をクリア
時間の表示 • サイクルの表示

図 103: [Power ログ] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

- 有効化

ロギングシステムを有効にします。つまり、電力の値は IDE で内部的に保存されます。値は [タイムライン] ウィンドウの [Power ログ] ウィンドウに表示されます (有効になっている場合)。ロギングシステムでは、ウィンドウを閉じるときにも情報が記録されます。
- クリア

IDE に内部的に保存された電力の値を消去します。デバッガをリセットしたり、[SWO 設定] ダイアログボックスで CPU クロックを変更した場合も、値は消去されます。
- ログファイルを保存

記録された電力の値の保存先ファイルを選択する、標準のファイル選択用ダイアログボックスを表示します。このコマンドにより、内部ログバッファの最新の内容が保存されます。

このファイルについては、249 ページの ログファイルのフォーマットを参照してください。
- ライブログファイルの選択

記録された電力の値の保存先ファイルを選択する、標準のファイル選択用ダイアログボックスを表示します。電力の値は実行時に連続してこのファイルに保存されます。ライブログファイルの内容は自動的にクリアされることはなく、記録された値は単にファイルの末尾に追加されます。

このファイルについては、249 ページの ログファイルのフォーマットを参照してください。

指定先へのライブログを有効にする	ライブロギングのオンとオフを切り替えます。ログは指定したファイルに保存されます。
ログファイルのクリア	ライブログファイルの内容を消去します。
時間表示	[Power ログ] ウィンドウに [時間] 列を表示します。この選択は、ログファイルにも反映されます。
サイクル表示	[Power ログ] ウィンドウに [サイクル] 列を表示します。この選択は、ログファイルにも反映されます。

ログファイルのフォーマット

ログファイルは、タブで区切られたフォーマットです。ログファイルのエントリは、タブおよびラインフィードで区切ります。記録された電力の値は、以下の列に表示されます。

時間 / サイクル	アプリケーションのリセットから電力の値が記録されるまでの時間。
概算値	この列の x は、電力の値が時間 / サイクルの概算値であることを示します。
プログラムカウンタ値	電力の値が記録されたポイントに近いプログラムカウンタの値。
名称 [単位]	[Power ログ] ウィンドウからの対応する値。名称と単位は、[Power 設定] ウィンドウの設定に従います。

コードカバレッジ

この章では、C-SPY® のコードカバレッジ機能について説明します。この機能は、コードのすべての部分が実行されたかどうかを検証する上で役立ちます。具体的には以下の項目を解説します。

- コードカバレッジの概要
- コードカバレッジについてのリファレンス情報

コードカバレッジの概要

このセクションでは、以下のトピックについて説明します。

- コードカバレッジを使用する理由
- コードカバレッジの概要
- コードカバレッジを使用するための要件

コードカバレッジを使用する理由

コードカバレッジ機能は、コードのあらゆる部分が実行されたことを確認するテスト手順を設計する場合に便利です。また、コードに到達不可能な部分が存在するかどうかを調べる場合にも使用できます。

コードカバレッジの概要

[コードカバレッジ] ウィンドウでは、現在のコードカバレッジ解析のステータスが表示されます。それぞれのプログラム、モジュール、関数について、コードカバレッジがオンになってからアプリケーションが停止するまでに実行されたコードの割合がパーセントで解析に表示されます。また、実行されていないすべての文の一覧も表示されます。オフになるまで解析は続行します。

コードカバレッジを使用するための要件

一部の C-SPY ドライバは、コードカバレッジをサポートしていません。使用するドライバについて詳しくは、38 ページの *C-SPY ドライバ間の差異* を参照してください。C-SPY シミュレータは、コードカバレッジをサポートしています。

コードカバレッジについてのリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 252 ページの [コードカバレッジ] ウィンドウ

関連項目 80 ページの ステップ実行。

[コードカバレッジ] ウィンドウ

[コードカバレッジ] ウィンドウは [表示] メニューから利用できます。

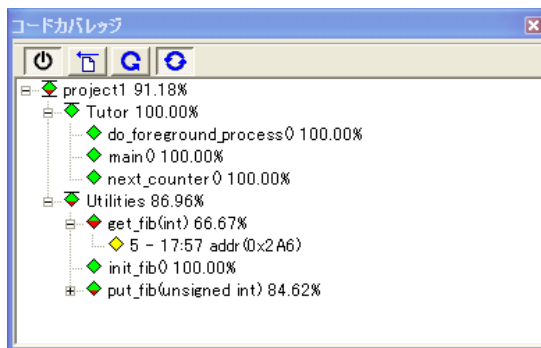


図104: [コードカバレッジ] ウィンドウ

このウィンドウには、現在のコードカバレッジ解析のステータスが表示されます。それぞれのプログラム、モジュール、関数について、コードカバレッジがオンになってからアプリケーションが停止するまでに実行されたコードの割合がパーセントで解析に表示されます。また、実行されていないすべての文の一覧も表示されます。解析は無効にするまで続行されます。



タイトルバーにアスタリスク (*) が表示されている場合は、C-SPY が実行を継続していること、および [コードカバレッジ] ウィンドウに表示されている情報が最新ではないため、それを最新の情報に更新する必要があることを示します。最新の情報に更新するには、[更新] コマンドを使用します。

コードカバレッジを使用するには、以下の手順に従います。

- 1 コードカバレッジ機能を使用するには、アプリケーションをビルドする際に以下のオプションを使用する必要があります。

カテゴリ	設定
C/C++ コンパイラ	[出力] > [デバッグ情報の生成]
リンカ	[出力] > [出力ファイルにデバッグ情報を含める]
デバッガ	[プラグイン] > [コードカバレッジ]

表 13: コードカバレッジを有効にするためのプロジェクトオプション

- 2 アプリケーションをビルドして C-SPY を起動した後、[表示] > [コードカバレッジ] を選択して [コードカバレッジ] ウィンドウを開きます。
- 3  [有効化] ボタンをクリックするか、コンテキストメニューから [有効化] を選択してコードカバレッジを有効にします。
- 4  実行を開始します。プログラムの終了に到達したり、ブレークポイントがトリガされたなどの理由で実行が停止したときは、[更新] ボタンをクリックして、コードカバレッジ情報を確認します。

表示エリア

コードカバレッジ情報には、ツリー構造でプログラム、モジュール、関数、文のレベルが表示されます。ウィンドウに表示されるのは、デバッグ情報付きでコンパイルされたソースコードだけです。したがって、起動コード、終了コード、ライブラリコードはウィンドウには表示されません。また、インライン化された関数内の文のカバレッジ情報は表示されません。インライン化された関数呼出しを含む文だけが実行済みとしてマークされます。プラス記号とマイナス記号をクリックすると、構造を展開したり折りたたんだりできます。

すべてのレベルの現在の状態は、以下のアイコンで示されます。

- 赤色のひし形 モジュールや関数の 0% が実行されたことを示します。
- 緑色のひし形 モジュールや関数の 100% が実行されたことを示します。
- 赤と緑のひし形 モジュールや関数の一部が実行されたことを示します。
- 黄色のひし形 文が 1 つ実行されていないことを示します。

プログラム、モジュール、関数の各行の末尾に表示されるパーセント値は、それまでにカバーされた文の量、すなわち実行済みの文の数を文の総数で割った値を表します。

実行されていない文（黄色のひし形）の場合、表示される情報はソースウィンドウの列番号の範囲と行番号、続いてステップポイントのアドレスです。

```
<column_start>-<column_end>:row address.
```

文は、その命令が 1 つでも実行されると、ステップポイントが実行されたとみなされます。文が実行されるとその文はウィンドウから削除され、それに対応してパーセント値が増加します。

[コードカバレッジ] ウィンドウで文か関数をダブルクリックすると、ソースウィンドウがアクティブウィンドウになり、ダブルクリックした文や関数がソースウィンドウでの現在の位置になります。プログラムレベルでモジュールをダブルクリックすると、ツリー構造を展開したり、折りたたんだりできます。

コンテキストメニュー

以下のコンテキストメニューがあります。

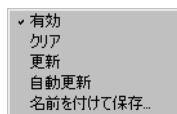








図 105: [コードカバレッジ] ウィンドウのコンテキストメニュー

以下のコマンドがあります。

	有効化	実行時のコードカバレッジの有効 / 無効を切り替えます。
	クリア	すべてのコードカバレッジ情報を消去します。すべてのステップポイントが未実行として表示されます。
	更新	コードカバレッジ情報を更新し、ウィンドウを再描画します。最後の更新以降に実行されたすべてのステップポイントは、ツリーから削除されます。
	自動更新	コードカバレッジ情報の自動再ロードの有効 / 無効を切り替えます。有効にした場合は、ブレークポイント、ステップポイント、プログラム終了で C-SPY が停止したときに、コードカバレッジ情報が自動的に再ロードされます。
	名前を付けて保存	現在のコードカバレッジ結果をテキストファイルに保存します。
	セッションの保存	コードカバレッジセッションのデータを *.dat ファイルに保存します。何らかの理由でデバッグセッションを中止しなければならず、後でセッションを続けたい場合にこれが役立ちます。このコマンドはツールバーから利用できます。
	セッションの復元	前回保存したコードカバレッジセッションのデータを復元します。何らかの理由でデバッグセッションを中止しなければならず、後でセッションを続けたい場合にこれが役立ちます。このコマンドはツールバーから利用できます。

割込み

ここでは、C-SPY® を使用した割込みサービスルーチンのロジックのテストや、ターゲットシステムでの割込み処理のデバッグの方法について説明します。割込みロギングを使用すると、割込みイベントに関する包括的な情報を得ることができます。具体的には、この章で以下について説明します。

- 割込みの概要
- 割込みの手順
- 割込みのリファレンス情報

割込みの概要

ここでは、割込みロギングおよび割込みのシミュレーションについて説明します。

ここでは以下のトピックについて説明します。

- 割込みロギングの概要
- 割込みシミュレーションシステムの概要について
- 割込み特性
- 割込みシミュレーションの状態
- 割込みシミュレーションの C-SPY システムマクロ
- ターゲットに合せた割込みシミュレーションシステムの調整

以下も参照してください。

- 292 ページの *C-SPY* システムマクロについてのリファレンス情報
- 119 ページの ブレークポイントの使用
- *ARM* 用 *IAR C/C++* 開発ガイド

割込みロギングの概要

割込みロギングを使用すると、割込みイベントに関する包括的な情報を得ることができます。この情報は、たとえば、高速化するためにどの割込みを微調整したらよいか調べるのに便利です。割込みの開始と終了を記録できます。また、トリガ済や期限切れなど、内部の割込みステータス情報も記録できま

す。ログは「割込みログ」ウィンドウに表示されます。概要は「割込みログサマリ」ウィンドウに表示されます。「タイムライン」ウィンドウの「割込みグラフ」には、アプリケーションプログラム実行中の割込みイベントがグラフィック表示されます。

割込みロギングの要件

割込みロギングを使用するには、以下が必要です。

- J-Link デバッグプローブまたは ST-LINK デバッグプローブ
- デバッグプローブとターゲットシステム間の SWD インタフェース
- 「割込みログ」ウィンドウからの割込みロギングを有効にする場合は、「割込みログ概要」または「タイムライン」ウィンドウ

割込みロギングは C-SPY シミュレータでもサポートされています。

割込みシミュレーションシステムの概要について

割込みをシミュレーションすることで、ハードウェアが入手可能になるよりかなり前に、割込みサービスルーチンのロジックをテストして、ターゲットシステムで割込み処理をデバッグできます。擬似割込みと C-SPY のマクロとブレイクポイントを連携させることによって、割込み駆動型の周辺デバイスのような複雑なシミュレーションを構築できます。

C-SPY シミュレータには、デバッグ中に割込みの実行をシミュレーションできる割込みシミュレーションシステムが用意されています。割込みシミュレーションシステムをハードウェア割込みシステムと同じように構成することができます。

割込みシステムの特長を以下に示します。

- ARM コアの割込みシミュレーションの提供
- 単発もしくはサイクルカウンタに基づく周期割込み
- さまざまなデバイス用の定義済割込み
- 保持時間、確率、タイミングのばらつきの設定
- タイミングの問題を特定するためのステータス情報
- ダイアログボックスまたは C-SPY システムマクロ、つまり対話型インタフェースと自動インタフェースを 1 つずつ使用した割込みの設定。また、割込みを即時強制することができます
- 定義された割込みごとにイベントを継続的に表示するログウィンドウ
- 現在の割込みアクティビティを示すステータスウィンドウ

【割込み設定】ダイアログボックスで定義した割込みはすべて、削除しない限りデバッグセッション終了後も保持されます。一方で強制割込みはサービスを受けるまで存在し、セッション終了後は保持されません。



割込みシミュレーションシステムはデフォルトでは有効になっていますが、必要がない場合は無効にすることでシミュレーションの実行速度を向上させることができます。無効にするには、【割込み設定】ダイアログボックスか、システムマクロを使用します。

割込み特性

擬似割込みは、ターゲットハードウェアの実際の割込みに似せるために、各割込みを微調整する属性のセットから構成されています。指定できる特性には、**初回割込み待機時間**、**繰返し間隔**、**保持時間**、**ばらつき**、**確率**があります。

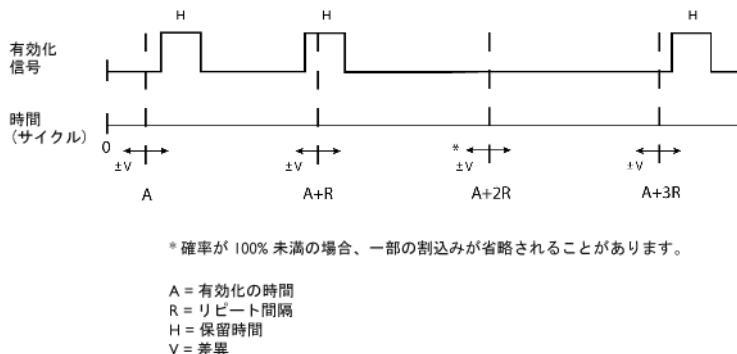


図 106: 擬似割込みの構成

割込みシミュレーションシステムは、サイクルカウンタを時計として使用し、シミュレータで割込みを発生させるタイミングを決定します。**初回割込み待機時間**は、サイクルカウンタ単位で指定します。**C-SPY**は、サイクルカウンタが指定された初回割込み待機時間を超えると、割込みを生成します。ただし、割込みが生成されるのは命令と命令の間だけです。すなわち、1つのアセンブラ命令の実行が完了するまでは、その命令に必要なサイクル数に関係なく、割込みの生成は待機させられます。

周期的に生成される割込みを定義するには、**繰返し間隔**を指定します。この値は、次の割込みを生成するまでの間隔を表すサイクル数を定義します。繰返し間隔の他に、その間隔が経過した後に実際に割込みを発生させる**確率**（パーセント値）と、繰返し間隔に対する**ばらつき**（パーセント値）の2つのオプションによって、実際の発生間隔を制御できます。この2つのオプションを使

用して、割込みシミュレーションをランダム化できます。その他に、割込みの*保持時間*を指定できます。この時間が経過しても処理されない割込みは削除されます。保持時間を*無限*に設定すると、割込みが確認され削除されるまで対応する保持ビットが設定されます。

割込みシミュレーションの状態

割込みシミュレーションシステムには、アプリケーションでのタイミング問題の場所を特定するために使用可能なステータス情報が含まれます。[割込みステータス] ウィンドウには、使用可能なステータス情報が表示されます。割込みに関して、下記の状態が表示されます。*待機*、*保留*、*実行中*、または*停止*です。

通常は、繰返し割込みの場合は 実行時間よりも長い繰返し間隔が指定されています。この場合、ステータス情報の経時変化は以下のようになります。

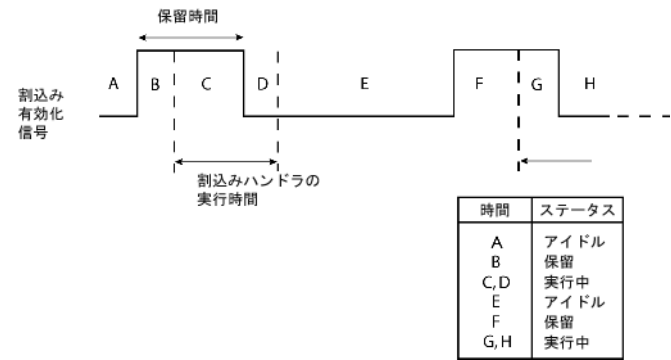


図 107: シミュレーション状態 — 例 1

注： 割込み有効信号（保持ビットともいう）は、割込みハンドラが割込みを認識するとすぐに、自動的に無効となります。

ただし、割込みの繰返し間隔が実行時間よりも短く、割込みがリエントラント（またはノンマスカブル）の場合、ステータス情報の経時変化は以下のようになります。

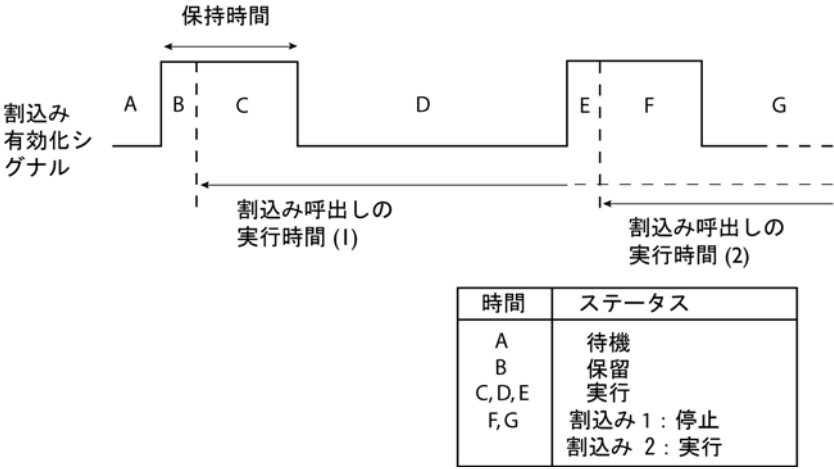


図 108: シミュレーション状態— 例 2

実行時間より繰返し間隔が長い場合は、割込みハンドラを記述し直して速くするか、割込みシミュレーションシステムの繰返し間隔を長く指定した方がよいことがあります。

割込みシミュレーションの C-SPY システムマクロ

マクロによる定義は、要求どおりのブレークポイント設定ができない場合に便利です。擬似割込みの定義を含むマクロ関数を記述することによって、C-SPY の起動時にそのマクロ関数を実行できます。他にも、マクロファイルを使用した場合に擬似割込みの定義がドキュメント化されたり、開発プロジェクトに複数のエンジニアが携わっている場合にグループ内でマクロファイルを共有できたりといった長所があります。

C-SPY シミュレータには、割込みに関連する以下の定義済みシステムマクロが用意されています。

```
__enableInterrupts
__disableInterrupts
__orderInterrupt
__cancelInterrupt
__cancelAllInterrupts
__popSimulatorInterruptExecutingStack
```

最初の 5 つのマクロのパラメータは、**【割込み】** ダイアログボックスの該当するエントリに対応します。

各マクロの詳細については、292 ページの *C-SPY システムマクロについてのリファレンス情報* を参照してください。

ターゲットに合せた割込みシミュレーションシステムの調整

割込みシミュレーションシステムは使いやすく設計されています。ただし、割込みシミュレーションシステムの力を最大限に利用するには、使用しているプロセッサに適合させる方法に習熟している必要があります。

割込みシミュレーションは、ハードウェアと同じように動作します。これは、割込みの実行はグローバルな割込みイネーブルビットのステータスに依存することを意味します。マスカブル割込みの実行も、個々の割込みイネーブルビットのステータスに依存します。

さまざまなデバイスに対してこれらのアクションを実行するには、使用できる割込みの詳細情報を割込みシステムに通知する必要があります。デフォルトの設定を除いて、この情報はデバイス記述ファイルにあります。デバイス記述ファイルが指定されなかった場合は、デフォルト設定が使用されます。

デバイス記述ファイルの詳細については、59 ページの *デバイス記述ファイルの選択* を参照してください。

割込みの手順

ここでは、割込みの手順について各段階ごとに説明します。

具体的には、以下の項目について説明します。

- シンプルな割込みシミュレーション
- マルチタスクシステムでの割込みシミュレーション
- C-SPY J-Link ドライバで割込みロギングを使用するにあたって

以下も参照してください。

- 設定ファイルを使用して C-SPY 起動時にシミュレーションされた割込みを定義する方法については、282 ページの *セットアップマクロとセットアップファイルによる登録と実行*
- インフォメーションセンタのチュートリアル「*割込みのシミュレーション*」

シンプルな割込みシミュレーション

次の例は、OKI ML674001 でのタイマ割込みをシミュレーションする方法を示します。ただし、他のタイプの割込みも同じ手順でシミュレーションできます。

割込みをシミュレーションしてデバッグするには、次の手順に従います。

- 1 このシンプルなアプリケーションには、システムタイマ割込みを処理する IRQ ハンドラルーチンが含まれます。tick 変数をインクリメントします。main 関数は必要に応じてステータスレジスタを設定します。アプリケーションは、100 回割込みが生成されると終了します。

```
/* 拡張キーワードの利用を有効にします。*/
#pragma language=extended

#include <intrinsics.h>
#include <oki/ioml674001.h>
#include <stdio.h>

unsigned int ticks = 0;

/* IRQ ハンドラ */
__irq __arm void IRQ_Handler( void )
{
    /* システムタイマ割込みのみを使用するため割込みソースをチェックする
       必要はありません。*/
    ticks += 1;
    TMOVFR_bit.OVF = 1; /* システムタイマのオーバフローフラグをクリアします*/
}

int main( void )
{
    __enable_interrupt();
    /* タイマ設定コード */
    ILC0_bit.ILR0 = 4; /* システムタイマ割込みの優先順位 */
    TMRLR_bit.TMRLR = 1E5; /* システムタイマの再ロード値 */
    TMEN_bit.TCEN = 1; /* システムタイマを有効にします */
    while (ticks < 100);
    printf("Done\n");
}
```

- 2 割込みサービルルーチンをアプリケーションソースコードに追加して、そのファイルをプロジェクトに追加します。
- 3 プロジェクトをビルドして、シミュレータを起動します。

- 4 [シミュレータ] > [割込み設定] を選択して、[割込み設定] ダイアログボックスを表示します。[割込みシミュレーションを有効にする] オプションを選択して、割込みシミュレーションを有効にします。[新規作成] をクリックして [割込みの編集] ダイアログボックスを開きます。Timer の例において、以下の設定を確認してください。

オプション	設定
割込み	IRQ
初回割込み	4000
繰返し間隔	2000
保持時間	10
確率 (%)	100
ばらつき (%)	0

表 14: タイマ割込み設定

[OK] をクリックします。

- 5 アプリケーションを実行します。アプリケーションソースコードで正常に割込みが有効になっている場合、C-SPY は以下を実行します。
- サイクルカウンタが 4000 を超えると割込みを生成
 - その後は約 2000 サイクルごとに周期割込みを生成
- 6 実行中の割込みを確認するには、[シミュレータ] > [割込みログ] を選択して、[割込みログ] ウィンドウを開きます。
- 7 [割込みログ] ウィンドウのコンテキストメニューから、[有効化] を選択してロギングを有効にします。プログラムの実行を再開する場合は、割込みの入口と出口のステータス情報が [割込みログ] ウィンドウに表示されます。

時間軸に沿って割込みをグラフィック表示する方法については、200 ページの [タイムライン] ウィンドウを参照してください。

マルチタスクシステムでの割込みシミュレーション

割込みハンドラから復帰する際に通常の命令以外の方法で割込みを使用する場合、たとえばタスク切替えが行われるオペレーティングシステムの場合、シミュレータは割込みの実行が完了したことを自動的に検出できません。割込みシミュレーションシステムは正常に動作しますが、[割込み設定] ダイアログボックスのステータス情報は予想したようには表示されない可能性があります。同時実行される割込みの数が多すぎると、ワーニングが出力される場合があります。

通常の割込み終了をシミュレーションするには、次の手順に従います。

- 1 割込み関数から復帰する命令にコードブレークポイントを設定します。
- 2 `__popSimulatorInterruptExecutingStack` マクロを条件としてブレークポイントに指定します。

ブレークポイントがトリガされると、マクロが実行され、それが完了するとアプリケーションが自動的に実行を継続します。

C-SPY J-LINK ドライバで割込みロギングを使用するにあたって

- 1 割込みロギングを設定するには、[J-Link]> [SWO 設定] を選択します。ダイアログボックスで、トレースデータの serial-wire output 通信チャンネルを設定します。特に [CPU クロック] オプションに注意してください。CPU クロックは、[プロジェクト] > [オプション] > [ST-Link] ページでも設定できます。
- 2 [J-Link]> [割込みログ] を選択して、[割込みログ] ウィンドウを開きます。または、以下のように選択することもできます。
 - [J-Link]> [割込みログ概要] を選択して、[割込みログ概要] ウィンドウを開く。
 - [J-Link]> [タイムラン] を選択して [タイムライン] ウィンドウを開き、割込みグラフを表示する。
- 3 [割込みログ] ウィンドウのコンテキストメニューから、[有効化] を選択してロギングを有効にします。

[SWO 設定] ダイアログボックスの [ログイベントの割込み] エリアで、割込みログが有効になっていることが確認できます。
- 4 アプリケーションプログラムの実行を開始して、ログ情報を収集します。
- 5 割込みログ情報を表示するには、[タイムライン] ウィンドウで割込みログ、割込みログ概要、または割込みグラフを参照します。
- 6 ログまたは概要をファイルに保存する場合は、対象のウィンドウのコンテキストメニューから [ログファイルを保存] を選択します。
- 7 割込みログを無効にするには、[割込みログ] ウィンドウのコンテキストメニューから [有効化] をオフにします。

割込みのリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 264 ページの [割込み設定] ダイアログボックス
- 266 ページの [割込みの編集] ダイアログボックス
- 268 ページの [強制割込み] ウィンドウ
- 269 ページの [割込みステータス] ウィンドウ
- 271 ページの [割込みログ] ウィンドウ
- 274 ページの [割込みログ概要] ウィンドウ

[割込み設定] ダイアログボックス

[割込み設定] ダイアログボックスは、[シミュレータ] > [割込み設定]。を選択することにより使用できます。

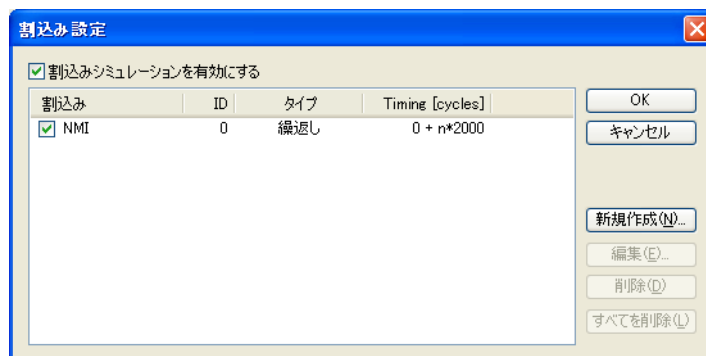


図 109: [割込み設定] ダイアログボックス

このダイアログボックスには、定義済みの割込みがすべて一覧表示されます。このダイアログボックスを使用して、割込みシミュレーションシステムおよび個々の割込みを有効/無効にします。

割込みシミュレーションの有効化

割込みシミュレーションを有効または無効にします。割込みシミュレーションが無効の場合、定義はそのまま残りますが、割込みは発生しません。インストール済みの割込みも、リストの割込み名の左にあるチェックボックスを使用して、個別に有効か無効にできます。

表示エリア

このエリアには以下の列が含まれます。

割込	すべての割込みが表示されます。チェックボックスを使用して、割込みを有効/無効にします。
ID	一意の割込み識別子。
タイプ	割込みタイプが表示されます。型は以下のいずれかです。 強制。単発の割込みで、[強制割込み] ウィンドウで定義します。 単一。単発の割込み。 繰返し。定期的に発生する割込みです。 割込みが C-SPY マクロから設定された場合、追加部分（マクロ）が追加されます。たとえば、Repeat (macro) というようになります。
タイミング	割込みのタイミング。単一と強制割込みの場合、割込み待機時間が表示されます。繰返し割込みの場合、情報は [割込み待機時間] + [n* 繰返し回数] の形式になります。たとえば、2000 + n*2345 というようになります。これは、この割込みが初めてトリガされるときが 2000 サイクルで、その後は 2345 サイクル間隔になることを示します。

ボタン

以下のボタンを選択できます。

新規	[割込みの編集] ダイアログボックスが開きます (266 ページの [割込みの編集] ダイアログボックスを参照)。
編集	[割込みの編集] ダイアログボックスが開きます (266 ページの [割込みの編集] ダイアログボックスを参照)。
削除	選択した割込みを削除します。
全て削除	すべての割込みを削除します。

〔割込みの編集〕 ダイアログボックス

〔割込みの編集〕 ダイアログボックスは、〔割込みの設定〕 ダイアログボックスから使用できます。



図 110: 〔割込みの編集〕 ダイアログボックス

このダイアログボックスを使用して、割込みパラメータを対話形式で微調整します。パラメータを追加して、すぐに要求どおり割込みが発生するかどうかをテストできます。

注：強制割込み以外の割込みのみ編集や削除ができます。

割込

編集する割込みを選択します。ドロップダウンリストに使用可能なすべての割込みが表示されます。ここで割込みを選択すると、自動的に〔説明〕ボックスが更新されます。**Cortex-M** デバイスの場合、リストには、選択したデバイス記述ファイルに記述されているエントリが表示されます。他のデバイスの場合、IRQ と FIQ の 2 つの割込みだけが使用できます。

説明

選択された割込みの内容。この内容は、選択したデバイス記述ファイルから取得され、優先順位、ベクタオフセット、イネーブルビット、および保持ビットを空白文字で区切って記述した文字列から構成されます。イネーブルビットと保持ビットはオプションです。何も使用しないか、イネーブルビットのみ、または両方を使用することができます。

Cortex-M デバイスの場合、記述は選択したデバイス記述ファイルから読み込まれ、これは編集可能です。イネーブルビットと保持ビットは `ddf` ファイルからは利用できません。必要な場合は手で編集する必要があります。優先順位はハードウェアの場合と同じで、数値が小さいほど優先順位は高くなり

ます。NMI と HardFault は特殊なため、記述を編集しないでください。
Cortex-M の割込みは、レジスタ PRIMASK、FAULTMASK、BASEPRI の影響も受けます。詳細は ARM のマニュアルを参照してください。

他のデバイスについては、IRQ および FIQ の記述文字列がハードコード化されており、編集はできません。これらの記述では、優先順位の番号が大きいほど優先順位が高くなります。

システムマクロ `__orderInterrupt` を使用して指定された割込みの場合、
[内容] ボックスは空欄です。

初回割込み

指定されたタイプの割込みを生成するまでの待機時間を表すサイクルカウンタ値を指定します。

繰返し間隔

割込みの間隔をサイクル数で指定します。

ばらつき (%)

タイミングのばらつきを繰返し間隔に対するパーセント値として選択します。このばらつきの範囲内に割込みが発生する可能性があります。たとえば、繰返し間隔が 100、ばらつきが 5% の場合、割込みは $T=95 \sim 105$ の間に発生する可能性があります、これによってタイミングのばらつきがシミュレーションされます。

保持時間

割込みの保持時間がサイクル単位で指定します。この時間が経過しても処理されない割込みは削除されます。[無限]を選択すると、対応する保持ビットは割込みが確認、削除されるまで設定されます。

確率 (%)

割込みが指定された期間内に実際に発生する確率をパーセント単位で選択します。

「強制割込み」ウィンドウ

「強制割込み」ウィンドウは「シミュレータ」メニューから利用できます。

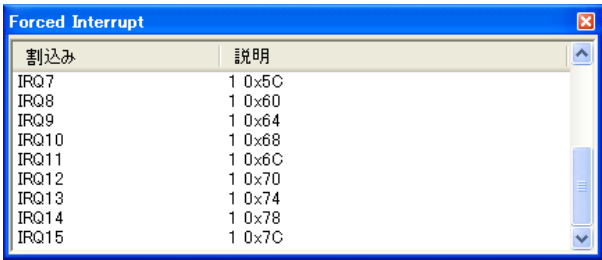


図 111: 「強制割込み」ウィンドウ

このウィンドウを使用して、割込みを即時に強制します。割込みロジックや割込みルーチンをチェックする場合に便利です。

強制割込みの保持時間は無制限です。割込みはサービスを受けるまで、またはデバッグセッションをリセットするまで存在します。

割込みを強制するには、次の手順に従います。

- 1 割込みシミュレーションシステムを有効にするには、264 ページの「割込み設定」ダイアログボックスを参照してください。
- 2 「強制割込み」ウィンドウで割込みをダブルクリックするか、コンテキストメニューの「強制」コマンドを使用して有効にします。

表示エリア

このエリアには使用可能なすべての割込みとその定義がリストされます。情報は、選択したデバイスの記述ファイルから読み込まれます。詳しい記述はこのファイルを参照してください。

コンテキストメニュー

以下のコンテキストメニューがあります。



図 112: 「強制割込み」ウィンドウのコンテキストメニュー

次のコマンドを使用できます。

強制 表示エリアで選択した割込みをトリガします。

「割込みステータス」ウィンドウ

「割込みステータス」ウィンドウは、「シミュレータ」メニューから使用できます。

Interrupt Status					
割込み	ID	タイプ	ステータス	Next Time	Timing [cycles]
NMI	0	強制	保留	--	--
IRQ0	1	強制	保留	--	--

図 113: 「割込みステータス」ウィンドウ

このウィンドウには、現在アクティブなすべての割込み、つまり実行中または実行を待つ割込みのステータスが表示されます。

表示エリア

このエリアには以下の列が含まれます。

割込	すべての割込みが表示されます。
ID	一意の割込み識別子。
タイプ	割込みのタイプ。タイプは以下のいずれかです。 強制。単発の割込みで、「強制割込み」ウィンドウで定義します。 単一。単発の割込み。 繰返し。定期的に発生する割込みです。 割込みが C-SPY マクロから設定された場合、追加部分 (マクロ) が追加されます。たとえば、Repeat (macro) というようになります。

ステータス	<p>割込みの状態</p> <p>待機。低い割込み有効信号です（無効）。</p> <p>保留。割込み有効信号はアクティブですが、割込みが割込みハンドラにまだ認識されていません。</p> <p>実行。割込みは現在サービス中です。すなわち、割込みハンドラ機能が実行中です。</p> <p>停止。優先度の高い割込みが実行中のため、割込みは現在停止中です。</p> <p>現在アクティブな割込みを削除した場合、「実行」と「停止」に（削除済）が追加されます。（削除済）は、割込みの実行が完了すると除去されます。</p>
次回	<p>次に待機中の割込みがトリガされる時。繰返し可能な割込みが実行を開始すると、その割込みのコピーが表示されて状態が「待機」になり「次回」が設定されます。次回が設定されていない割込み（保留や実行、停止）の場合、この列には -- と表示されます。</p>
タイミング	<p>割込みのタイミング。単一と強制割込みの場合、割込み待機時間が表示されます。繰返し割込みの場合、情報は [割込み待機時間] + [n* 繰返し回数] の形式になります。たとえば、2000 + n*2345 というようになります。これは、この割込みが初めてトリガされるときが 2000 サイクルで、その後は 2345 サイクル間隔になることを示します。</p>

「割り込みログ」ウィンドウ

「割り込みログ」ウィンドウは「シミュレータ」メニュー、「J-Link」メニュー、または「ST-LINK」メニューから使用できます。

別の割り込み中に発生した割り込み

サイクル	割り込み	ステータス	プログラムカウンタ	実行サイクル
2004	UART	トリガ済み	0x116	
2004	UART	入る	0x1FC	
2283	UART	抜ける	0x212	279
5001	NMI	トリガ済み	0x13A	
5001	NMI	入る	0x2FA	
5334	UART	トリガ済み	0x2FA	
5334	UART	入る	0x1FC	
5613	UART	抜ける	0x212	279
8666	UART	トリガ済み	0x2FA	
8666	UART	入る	0x1FC	
8945	UART	抜ける	0x212	279
12001	UART	トリガ済み	0x2FA	
12001	UART	入る	0x1FC	
12280	UART	抜ける	0x212	279
15333	UART	トリガ済み	0x2FA	
15333	UART	入る	0x1FC	
15612	UART	抜ける	0x212	279
18665	UART	トリガ済み	0x2FA	

白色の行は割り込みの開始を示します

灰色の行は割り込みの終了を示します

図114: 「割り込みログ」ウィンドウ

「割り込みログ」ウィンドウを使用するには、以下のいずれかが必要です。

- J-Link デバッグプローブまたは ST-LINK デバッグプローブと、デバッグプローブおよびターゲットシステム間の SWD インタフェース。
- C-SPY シミュレータ。

このウィンドウには割り込みの開始と終了が記録されます。C-SPY シミュレータには、内部の状態の変化も記録されます。

この情報は、ターゲットシステムの割り込み処理をデバッグする場合に役に立ちます。「割り込みログ」ウィンドウが開いている間は、実行時にステータスが常時更新されます。

注：保存されるログの数には制限があります。この制限を超過すると、バッファの最初のエントリが消去されます。

詳細については、263 ページの *C-SPY J-Link* ドライバで割込みロギングを使用するにあたってを参照してください。

アプリケーションの実行中に割込みイベントをグラフィック表示する方法については、200 ページの [タイムライン] ウィンドウを参照してください。

表示エリア : C-SPY J-Link ドライバと ST-LINK ドライバについて

このエリアには以下の列が含まれます。

時間	<p>[SWO 設定] ダイアログボックスで指定したクロック周波数に基づく、割込み開始の時間。</p> <p>ターゲットシステムが正確な時間を収集できなかった場合は、おおよその時間が斜体で表示されます。</p> <p>この列は、コンテキストメニューから [時間表示] を選択した場合に有効になります。</p>
サイクル	<p>実行の開始からイベントまでのサイクルの数。</p> <p>斜体表示のサイクルカウントは、おおよその値を示します。ターゲットシステムが正確な値を収集できなかった場合は、おおよその値が斜体で表示されます。</p> <p>この列は、コンテキストメニューから [サイクル表示] を選択した場合に有効になります。</p>
割込	<p>割込みが発生する割込みソース名。赤色で Overflow と表示されている場合、通信チャンネルがすべての割込みログをターゲットシステムから送信できなかったことを示します。</p>
ステータス	<p>割込みのイベントステータス :</p> <p>開始。 割込みが現在実行中です。</p> <p>終了。 割込みの実行が完了しました。</p>
プログラムカウンタ *	<p>割込みハンドラのアドレス。</p>
実行時間 / サイクル	<p>割込みの経過時間。開始と終了のタイムスタンプを使用して算出されます。特定の割込みで発生した他の割込みやサブルーチンの時間も含まれます。</p>

* アドレスをダブルクリックできます。ソースコードで使用可能な場合、エディタウィンドウに、たとえば、割込みハンドラなど、対応するソースコードが表示されます (ライブラリソースコードは除く)。

C-SPY シミュレータのトレース表示エリア

このエリアには以下の列が含まれます。

時間	内部で指定したクロック周波数に基づいた、割込み入口の時間。 この列は、コンテキストメニューから 【時間表示】 を選択した場合に有効になります。
サイクル	実行の開始からイベントまでのサイクルの数。 この列は、コンテキストメニューから 【サイクル表示】 を選択した場合に有効になります。
割込	デバイス記述ファイルで定義された割込み。
ステータス	割込みイベントステータスが表示されます： トリガ済み 。待機時間がすでに経過している割込み。 強制 。トリガ済みと同じですが、割込みは 【強制割込み】 ウィンドウから強制されています。 開始 。割込みが現在実行中です。 終了 。割込みが実行済みです。 期限切れ 。割込みが実行されることなく保持時間が経過しました。 却下 。割込みを受け入れるのに必要な割込みレジスタが設定されていないため、割込みが却下されました。
プログラムカウンタ	イベントが発生したときのプログラムカウンタの値。
実行時間 / サイクル	割込みの経過時間。開始と終了のタイムスタンプを使用して算出されます。特定の割込みで発生した他の割込みやサブルーチンの時間も含まれます。

【割込みログ】ウィンドウのコンテキストメニュー

このコンテキストメニューは、**【割込みログ】** ウィンドウと **【割込みログ概要】** ウィンドウから使用できます。

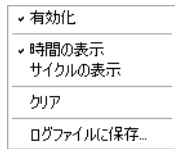


図 115: **【割込みログ】** ウィンドウのコンテキストメニュー

注：各コマンドはどのウィンドウにも表示されますが、特定のウィンドウでのみ機能します。

以下のコマンドがあります。

有効化	ロギングシステムを有効にします。ロギングシステムでは、ウィンドウを閉じるときにも情報が記録されます。
クリア	ログ情報を削除します。デバッガをリセットしたときにも同じことになります。
ログファイルを保存	ログ情報の保存先のファイルを指定する、標準のファイル選択用ダイアログボックスを表示します。ログファイルのエントリは、タブおよびLF（ラインフィード）で区切ります。[Approx] 列の x は、タイムスタンプが概算値であることを示します。
時間表示	[データログ] ウィンドウおよび [割込みログ] ウィンドウに [時間] 列を表示します。
サイクル表示	[データログ] ウィンドウおよび [割込みログ] ウィンドウに [サイクル] 列を表示します。

[割込みログ概要] ウィンドウ

[割込みログ概要] ウィンドウは [シミュレータ] メニュー、[J-Link] メニュー、または [ST-LINK] メニューから使用できます。

割込みログの一覧							
割込み	カウント	初回	合計時間	最速	最低	最小間隔	最大間隔
ADC	34	120.040us	469.200us	13.800us	13.600us	24.700us	60.020us
RTC	4	25.560us	65.280us	16.320us	16.320us	67.180us	1441.060us
近似時間がカウント: 0 オーバーフローカウント: 0							

図 116: [割込みログ概要] ウィンドウ

[割込みログ概要] ウィンドウを使用するには、以下のいずれかが必要です。

- J-Link デバッグプローブまたは ST-LINK デバッグプローブと、デバッグプローブおよびターゲットシステム間の SWD インタフェース。
- C-SPY シミュレータ。

このウィンドウには、記録された割込みの入口と出口の概要が表示されます。

詳細については、263 ページの *C-SPY J-Link* ドライバで割込みロギングを使用するにあたってを参照してください。

アプリケーションの実行中に割込みイベントをグラフィック表示する方法については、200 ページの [タイムライン] ウィンドウを参照してください。

C-SPY シミュレータのトレース表示エリア

このエリアの各行の以下の列には、ログ情報に基づいて特定の割込みに関する統計が表示されます。

割込 *	発生した割込みのタイプ。
カウント	割込みが発生した回数。
初回	割込みの初回の実行時間。
合計時間 **	割込みの累計時間。
最速 **	このタイプの割込み 1 回の最短実行時間。
最長 **	このタイプの割込み 1 回の最長実行時間。
最長間隔	このタイプの 2 つの割込みの最長間隔。

* 列の下には、現在の時間やサイクル（実行開始からのサイクル数や実行時間）が表示されます。オーバーフローカウントとおおよその時間は常にゼロです。

** [割込みログ] ウィンドウの [実行時間 / サイクル] と同じ方法で計算されます。

† この間隔は、2 つの連続した割込みの開始時間の間隔です。

コンテキストメニュー

273 ページの [割込みログ] ウィンドウのコンテキストメニューを参照してください。

C-SPY マクロの使用

C-SPY® には、包括的なマクロ言語が含まれています。これを使用して、デバッグ処理を自動化し、周辺デバイスをシミュレーションすることができます。

この章では、C-SPY マクロ言語、その機能、適用可能な目的、使用方法について説明します。具体的には以下の項目を解説します。

- C-SPY マクロの概要
- C-SPY マクロの使用手順
- マクロ言語についてのリファレンス情報
- 予約済みのセットアップマクロ関数名についてのリファレンス情報
- C-SPY システムマクロについてのリファレンス情報

C-SPY マクロの概要

このセクションでは、以下のトピックについて説明します。

- C-SPY マクロを使用する理由
- C-SPY マクロの使用の概要
- セットアップマクロ関数およびファイルの概要
- マクロ言語の概要

C-SPY マクロを使用する理由

C-SPY マクロは、単独で使用することもできますが、複雑なブレークポイントおよび割込みシミュレーションとともに使用することにより、さまざまなタスクを実行できます。マクロが役に立つ例をいくつか示します。

- トレース出力、変数値の出力、ブレークポイントの設定などによるデバッグセッションの自動化
- ハードウェアレジスタの初期化などのハードウェア設定
- 実行中のアプリケーションへのシミュレーションしたデータの入力

- 周辺デバイスのシミュレーション。「[割込み](#)」を参照してください。シミュレータドライバを使用している必要があります
- スタックの深さを計算する関数など、簡単なデバッグユーティリティ関数の開発については、`%arm%src%sim%` ディレクトリにある `stack.mac` の例を参照してください

C-SPY マクロの使用の概要

C-SPY マクロを使用するには、以下のことを行う必要があります。

- マクロ変数と関数を記述して、1 つまたは複数のマクロファイルに収集
- マクロを登録
- マクロを実行

マクロの登録と実行については、いくつかの方法から選択できます。どの方法を選択するかは、操作や自動化のレベル、どの段階でマクロを登録および実行するかによって異なります。

セットアップマクロ関数およびファイルの概要

予約済のセットアップマクロ関数名がいくつかあります。これらは以下のような特定のタイミングに呼び出されるマクロ関数を定義する際に使用できます。

- ターゲットシステムとの通信確立後、アプリケーションソフトウェアをダウンロードするまでの間
- アプリケーションソフトウェアのダウンロードが完了した直後
- リセットコマンドが発行されるたび
- デバッグセッションの終了直後

マクロ関数を呼び出すタイミングを定義するには、予約済の名前でマクロ関数を定義、登録する必要があります。たとえば、アプリケーションソフトウェアをロードする前に特定のメモリエリアをクリアする必要がある場合は、マクロセットアップ関数 `execUserPreload` が適しています。アプリケーションソフトウェアをロードする前に一部の CPU レジスタやメモリにマッピングされた周辺ユニットを初期化する必要がある場合にも、この関数は適しています。

これらの関数をセットアップマクロファイルに定義します。これは C-SPY の起動前にロードできます。この場合、マクロ関数は C-SPY を起動するたびに自動的に登録されます。これは、C-SPY の初期化を自動化する場合や、複数のセットアップマクロファイルを登録する必要がある場合にも使用できます。

各セットアップマクロ関数の詳細については、[291 ページの 予約済みのセットアップマクロ関数名についてのリファレンス情報](#)を参照してください。

メモリの再配置

ARM を基本とする多くのプロセッサに共通する機能は、メモリの再配置機能です。メモリコントローラでは、リセット後に、フラッシュのような不揮発性メモリにアドレスのゼロをマッピングするのが一般的です。メモリコントローラを構成することで、RAM をアドレスマップのゼロに配置し、不揮発性メモリをアドレスマップの上位に配置するように、システムメモリを再配置することができます。再配置することで、例外テーブルは RAM に置かれ、ターゲットハードウェアにコードをダウンロードしたときに簡単に修正できます。C-SPY でこれを処理するには、セットアップマクロの `execUserPreload()` 関数が適しています。例については、65 ページの *メモリの再配置* を参照してください。

マクロ言語の概要

マクロ言語の構文は C 言語に非常によく似ています。以下の共通点があります。

- C 言語の文に似たマクロ文があります。
- マクロ関数を、パラメータとリターン値の有無を指定して定義できます。
- C ライブラリ関数に似た定義済み組込みシステムマクロ。ファイルのオープンやクローズ、ブレークポイントの設定、割込みシミュレーションの定義など便利なタスクを実行できます。
- マクロ変数。グローバルかローカルのどちらかで、C-SPY 式で使用できます。
- マクロ文字列。定義済システムマクロを使用して操作することができます。

マクロ言語のコンポーネントの詳細については、286 ページの *マクロ言語* についての *リファレンス情報* を参照してください。

例

以下に示すマクロ関数の例では、マクロ言語のさまざまなコンポーネントが示されています。

```
__var oldVal;
CheckLatest(val)
{
    if (oldval != val)
    {
        __message "Message: Changed from ", oldval, " to ", val, "\n";
        oldval = val;
    }
}
```

注：マクロの予約語は、名前の衝突を避けるために、2 連のアンダースコアで始まります。

C-SPY マクロの使用手順

このセクションでは、C-SPY マクロの登録と実行方法についてステップごとに説明します。

具体的には、以下の項目について説明します。

- C-SPY マクロの登録 — 概要
- C-SPY マクロの実行 — 概要
- [マクロ設定] ダイアログボックスの使用
- セットアップマクロとセットアップファイルによる登録と実行
- [クイックウォッチ] によるマクロの実行
- ブレークポイントにマクロを接続して実行

その他の C-SPY マクロの使用例については、以下を参照してください。

- インフォメーションセンタのチュートリアル「*割込みのシミュレーション*」
- 64 ページの C-SPY の起動前にターゲットハードウェアを初期化する

C-SPY マクロの登録 — 概要

次に、定義したマクロ関数を使用することを C-SPY に通知する必要があるため、マクロファイルを登録する必要があります。マクロ関数の登録方法はいろいろあります。

- [マクロ設定] ダイアログボックスで対話的にマクロを登録できます (281 ページの [マクロ設定] ダイアログボックスの使用を参照)
- C-SPY の起動シーケンス中にマクロ関数を登録できます (282 ページの セットアップマクロとセットアップファイルによる登録と実行を参照)
- システムマクロ `__registerMacroFile` を使用すると、マクロ関数定義を含むファイルを登録できます。これは、実行時の条件に応じて、登録するマクロファイルを動的に選択できることを意味します。さらに、システムマクロを使用する場合は、同時に複数のファイルを登録できます。システムマクロの詳細については、315 ページの `__registerMacroFile` を参照してください

どの方法を選択するかは、操作や自動化のレベル、どの段階でマクロを登録するかによって異なります。

C-SPY マクロの実行 — 概要

マクロ関数の実行方法はいろいろあります。

- セットアップマクロファイルでセットアップマクロ関数を定義することにより、C-SPY の起動シーケンス中およびデバッグセッションの他の定義済みの段階でマクロ関数を実行できます (282 ページの セットアップマクロとセットアップファイルによる登録と実行を参照)。

- [クイックウォッチ] ウィンドウでは式を評価できるため、それによってマクロ関数を実行できます。例については、283 ページの [クイックウォッチ] によるマクロの実行を参照してください。
- マクロはブレークポイントに接続でき、ブレークポイントがトリガされると、マクロが実行されます。例については、284 ページのブレークポイントにマクロを接続して実行を参照してください。

どの方法を選択するかは、操作や自動化のレベル、どの段階でマクロを実行するかによって異なります。

[マクロ設定] ダイアログボックスの使用

[マクロ設定] ダイアログボックスは、[デバッグ] > [マクロ] を選択して利用します。



図 117: [マクロ設定] ダイアログボックス

このダイアログボックスを使用して、マクロファイルと関数の一覧表示、登録、編集を行います。このダイアログボックスは、マクロ関数を対話的に登録できるインタフェースを提供します。マクロ関数を開発するときに、そのロードとテストを繰り返すような場合に便利です。

ダイアログボックスを使用して登録したマクロ関数は、デバッグセッションを終了すると無効になり、次のデバッグセッションで自動的に登録されません。

マクロファイルを登録するには、次の手順に従います。

- 1 登録するマクロファイルをファイル選択リストで選択し、**[追加]** か **[すべて追加]** をクリックして **[選択したマクロファイル]** のリストに追加します。逆に、**[選択したマクロファイル]** リストからファイルを削除するには、**[削除]** か **[すべてを削除]** を使用します。
- 2 **[登録]** をクリックすると、以前に定義したマクロ関数や変数がすべて置換され、マクロ関数を登録できます。登録したマクロ関数は、**[登録マクロ]** のスクロールリストに表示されます。

注：システムマクロは常に登録されていて、リストから削除することはできません。

マクロ関数のリストを表示するには、次の手順に従います。

- 1 **[すべて]** を選択するとすべてのマクロ関数、**[ユーザ]** ではすべてのユーザ定義マクロ、**[システム]** はすべてのシステムマクロがそれぞれ表示されます。
- 2 **[登録マクロ]** にある **[マクロ名]** か **[ファイル]** をクリックすると、列の内容がマクロ名かファイル名を基準にしてソートされます。もう一度クリックすると、逆順でソートされます。

マクロファイルを修正するには、次の手順に従います。

ユーザ定義マクロ関数を **[マクロ名]** 列でダブルクリックすると、その関数が定義されているファイルが表示され、修正を行うことができます。

セットアップマクロとセットアップファイルによる登録と実行

C-SPY 起動シーケンスの途中でマクロファイルを登録すると便利なときがあります。そのためには、デバッグを起動する前にロードするマクロファイルを指定します。この場合、マクロ関数はデバッグを起動するたびに自動的に登録されます。

予約済のセットアップマクロ関数名を使用してマクロ関数を定義する場合、マクロ関数を実行するタイミングを正確に定義できます。

セットアップマクロ関数を定義して C-SPY 起動中にロードするには、次の手順に従います。

- 1 マクロ関数を定義するテキストファイルを作成します。

次に例を示します。

```
execUserSetup()
{
    ...
    __registerMacroFile("MyMacroUtils.mac");
    __registerMacroFile("MyDeviceSimulation.mac");
}
```

このマクロ関数は、追加のマクロファイル MyMacroUtils.mac および MyDeviceSimulation.mac を登録します。このマクロ関数は execUserSetup という関数名で定義されているため、アプリケーションのダウンロードが完了した直後に実行されます。

- 2 ファイル名拡張子を mac としてこのファイルを保存します。
- 3 C-SPY を起動する前に、[プロジェクト] > [オプション] > [デバッグ] > [設定] を選択します。[マクロファイルの使用] チェックボックスを選択して、作成したマクロファイルを選択します。

マクロが C-SPY の起動シーケンス中に登録されるようになります。

[クイックウォッチ] によるマクロの実行

[クイックウォッチ] ウィンドウでは、マクロ関数を実行するタイミングを動的に選択できます。

- 1 以下に示す、ウォッチドッグタイマ割込みイネーブルビットのステータスをチェックする単純なマクロ関数について考えます。

```
WDTstatus()
{
    if (#WD_SR & 0x01 != 0) /* WDOVF の状態を確認します */
        return "Watchdog triggered; /* 使用した C-SPY マクロ文字列 */
    else
        return "Watchdog not triggered"; /* 使用した C-SPY マクロ文字列 */
}
```

- 2 ファイル名の拡張子を mac としてこのマクロ関数を保存します。
- 3 マクロファイルを登録するには、[デバッグ] > [マクロ] を選択します。[マクロ設定] ダイアログボックスが表示されます。

- 4 マクロファイルを特定して、[追加] をクリックし、[登録] をクリックします。登録されているマクロのリストに、マクロ関数が表示されます。
- 5 [表示] > [クイックウォッチ] を選択して [クイックウォッチ] ウィンドウを開き、テキストフィールドにマクロ呼出し `WDTstatus()` と入力して [Enter] を押します。

または、マクロファイルエディタウィンドウで、マクロ関数名 `WDTstatus()` を選択します。右クリックして、表示されるコンテキストメニューで [クイックウォッチ] を選択します。



図 118: [クイックウォッチ] ウィンドウ

マクロは、[クイックウォッチ] ウィンドウに自動的に表示されます。

詳細については、112 ページの [クイックウォッチ] ウィンドウを参照してください。

ブレークポイントにマクロを接続して実行

マクロは、ブレークポイントに接続できます。これにより、ブレークポイントがトリガされると、マクロが実行されます。この方法では、特定の場所で実行を停止して、そこで特定のアクションを実行できるという長所があります。



たとえば、変数、シンボル、レジスタの値が変更された経緯などの情報を含むログレポートを簡単に作成できます。これを実現するには、疑わしい位置にブレークポイントを設定し、そのブレークポイントにログマクロを接続します。これにより、処理を実行した後で、レジスタの値が変更された経緯を調べることができます。

ログマクロを作成してブレークポイントに接続するには、次の手順に従います。

- 1 アプリケーションソースコードに以下の C 関数のスケルトンが定義されていると仮定します。

```
int fact(int x)
{
    ...
}
```

- 2 以下の例のような簡単なログマクロ関数を作成します。

```
logfact ()
{
  __message "fact (" ,x, ")";
}
```

__message 文で、メッセージが [ログ] ウィンドウにロギングされます。

マクロファイルのファイル名の拡張子を mac として、ログマクロ関数を保存します。

- 3 マクロを登録するには、[デバッグ] > [マクロ] を選択して [マクロ設定] ダイアログボックスを開き、マクロファイルをリストの [選択したマクロファイル] に追加します。[登録] をクリックすると、マクロ関数が [登録マクロ] のリストに表示されます。ダイアログボックスを閉じます。
- 4 コードブレークポイントを設定するには、アプリケーションのソースコードの関数 fact 内の最初の文で [ブレークポイントの切替え] ボタンをクリックします。[表示] > [ブレークポイント] を選択して、[ブレークポイント] ウィンドウを開きます。ブレークポイントのリストで自分のブレークポイントを選択し、コンテキストメニューで [編集] コマンドを選びます。
- 5 ログマクロ関数をブレークポイントに接続するには、マクロ関数名 logfact () を [アクション] フィールドに入力して [適用] をクリックします。ダイアログボックスを閉じます。
- 6 アプリケーションのソースコードを実行します。ブレークポイントがトリガされると、マクロ関数が実行されます。結果は [ログ] ウィンドウに表示されます。

[アクション] フィールドの式は、ブレークポイントによって実行が実際に停止する場合にのみ評価されます。値をログに記録して自動的に実行を継続する場合、以下のいずれかの方法を使用します。

- ログブレークポイントを使用 (141 ページの [ログ] ブレークポイントダイアログボックスを参照)
- [アクション] フィールドではなく [条件] フィールドを使用 例については、131 ページのタスクを処理して実行を継続するを参照してください

- 7 ログマクロ関数は簡単に拡張できます。たとえば、__fmessage 文を使用すると、ファイルにログ情報を出力できます。__fmessage 文については、289 ページのフォーマットした出力を参照してください。

マクロをブレークポイントに接続することによってシリアルポート入力バッファをシミュレーションする例については、インフォメーションセンタのチュートリアル、[割込みのシミュレーション] を参照してください。

マクロ言語についてのリファレンス情報

このセクションではマクロ言語のリファレンス情報を提供します。

- 286 ページの マクロ関数
- 286 ページの マクロ変数
- 287 ページの マクロ文字列
- 288 ページの マクロ文
- 289 ページの フォーマットした出力

マクロ関数

C-SPY のマクロ関数は、C-SPY 変数定義と、マクロが呼び出されたときに実行されるマクロ文で構成されます。マクロ関数には任意の個数のパラメータを引き渡すことができます。また、マクロ関数は終了時に値を返すことができます。

C-SPY マクロの形式は、以下のとおりです。

```
macroName (parameterList)
{
    macroBody
}
```

ここで、*parameterList* にはコンマ区切りのマクロパラメータリスト、*macroBody* には C-SPY 変数定義および C-SPY 文を記述します。

タイプチェックは、マクロ関数に引き渡される値とリターン値のいずれでも実行されません。

マクロ変数

マクロ変数は、アプリケーション外で定義して配置される変数です。C-SPY 式で利用できるほか、アプリケーションデータ（アプリケーションの変数値）を割り当てることができます。C-SPY 式の詳細については、98 ページの C-SPY 式を参照してください。

マクロ変数を定義する構文は、以下のとおりです。

```
__var nameList;
```

ここで、*nameList* にはコンマ区切りの C-SPY 変数名リストを指定します。

マクロ本体の外で定義したマクロ変数は、グローバルスコープになり、デバッグセッション全体に存在します。マクロ本体内部で定義されたマクロ変数は、その定義文の実行時に作成され、マクロから戻るときに破棄されます。

デフォルトでは、マクロ変数は符号付き整数として処理され、0 に初期化されます。式で C-SPY 変数に値を割り当てると、その式の型も変数に適用されます。次に例を示します。

式	意味
<code>myvar = 3.5;</code>	<code>myvar</code> の型は <code>float</code> 、値は 3.5 に設定されます。
<code>myvar = (int*)i;</code>	<code>myvar</code> は <code>int</code> 型ポインタになり、値は <code>i</code> と同一です。

表 15: C-SPY マクロ変数の例

C のシンボルと C-SPY マクロ変数の間で名前が重複する場合は、C-SPY マクロ変数の方が C の変数よりも優先されます。マクロ変数はデバッグホストに割り当てられるため、アプリケーションは影響されないことに注意してください。

マクロ文字列

C のデータ型に加えて、マクロ変数にマクロ文字列の値を保持できます。マクロ文字列は C 言語文字列と異なることに、注意してください。

C-SPY 式に `"Hello!"` などの文字列リテラルを書き込む場合、この値はマクロ文字列になります。`char*` はターゲットメモリの文字列を参照する必要がありますが、C-SPY ではターゲットメモリに実際に存在するいかなる文字列も使用できないので、これは C-形式文字ポインタ `char*` ではありません。

`__strFind` または `__subString` などの組み込みマクロ関数を使用して、マクロ文字列を操作できます。結果として新しいマクロ文字列が作成されます。`str + "tail"` のような + 演算子を使用して、マクロ文字列を連結できます。`str[3]` などのサブスクリプションを使用して、個々の文字も取得できます。`sizeof(str)` を使用して、文字列の長さを取得できます。マクロ文字列は NULL 終了ではないことに注意してください。

マクロ関数 `__toString` を使用して、アプリケーション内の NULL 終了 C 文字列 (`char*` または `char[]`) からマクロ文字列に変換します。たとえば、アプリケーションに以下の C 文字列の定義があると仮定します。

```
char const *cstr = "Hello";
```

次に、以下のマクロの例を検討します。

```
__var str;           /* マクロ変数 */
str = cstr           /* str は現在、char へのポインタです */
sizeof str          /* sizeof (char*) と同様で、通常は 2 か 4 です */
str = __toString(cstr,512) /* str は現在、マクロ文字列です */
sizeof str          /* 5、文字列の長さ */
str[1]              /* 101、'e' の ASCII コード */
str += " World!"    /* str はここで "Hello World!" になります */
```

289 ページの フォーマットした出力も参照してください。

マクロ文

マクロ文は、相当する C 文と同様に機能します。以下の C-SPY マクロ文を使用できます。

式

式 ;

C-SPY 式の詳細については、98 ページの *C-SPY* 式を参照してください。

条件文

```
if (式)
    文
```

```
if (式)
    文
else
    文
```

ループ文

```
for (init_expression; cond_expression; update_expression)
    文
```

```
while (式)
    文
```

```
do
    文
while (式) ;
```

return 文

```
return;
```

```
return 式;
```

リターン値が明示的に設定されていない場合は、デフォルトでは signed int 0 が返されます。

ブロック

マクロ文をブロックとしてグループ化できます。

```
{
    statement1
    statement2
    .
    .
    .
    statementN
}
```

フォーマットした出力

C-SPY では、フォーマットした出力をさまざまな方法で生成できます。

```
__message argList;           [デバッグログ] ウィンドウに出力します。
__fmessage file, argList;    指定ファイルに出力します。
__smessage argList;          フォーマットした出力を文字列に格納して戻
                              します。
```

ここで、*argList* は C-SPY の式か文字列をコンマで区切ったリストで、*file* は `__openFile` システムマクロの実行結果です（311 ページの `__openFile` を参照）。

〔デバッグログ〕ウィンドウにメッセージを出力するには、以下の手順に従います。

```
var1 = 42;
var2 = 37;
__message "This line prints the values ", var1, " and ", var2,
" in the Log window.";
```

この手順を実行すると、以下のメッセージが〔ログ〕ウィンドウに出力されます。

```
This line prints the values 42 and 37 in the Log window.
```

出力を指定のファイルに書き込む場合：

```
__fmessage myfile, "Result is ", res, "!\n";
```

文字列を生成する場合：

```
myMacroVar = __smessage 42, " is the answer.";
```

`myMacroVar` に、ここで文字列 `"42 is the answer."` が格納されます。

引数の表示フォーマットの指定

`argList` のスカラー引数（数値またはポインタ）のデフォルトの表示フォーマットは、後に：およびフォーマット指定子を記述することで変更することができます。使用可能な指定子は次のとおりです。

<code>%b</code>	2 進数のスカラー引数
<code>%o</code>	8 進数のスカラー引数
<code>%d</code>	10 進数のスカラー引数
<code>%x</code>	16 進数のスカラー引数
<code>%c</code>	文字のスカラー引数

これらの指定子は、[ウォッチ] ウィンドウや [ロケール] ウィンドウで使用可能なフォーマットと同一ですが、プレフィックスや文字列 / 文字の前後の引用符は出力されません。別の例を示します。

```
__message "The character '", cvar:%c, "' has the decimal value ",
cvar;
```

変数の値に応じて、以下のメッセージが出力されます。

```
The character 'A' has the decimal value 65
```

注：単一引用符で括った文字（文字定数）は整数定数として処理され、文字としてフォーマットされません。次に例を示します。

```
__message 'A', " is the numeric value of the character ", 'A':%c;
```

この場合、次のように出力されます。

```
65 is the numeric value of the character A
```

注：特定タイプ用デフォルトフォーマットは主に [ウォッチ] ウィンドウやその他の関連するウィンドウでできるように設計されています。たとえば、タイプ `char` は 'A' (0x41) に、文字（主に C 文字列）用ポインタは 0x8102 "Hello" にフォーマットされるように、文字列部分には文字列の始めの部分（現在、60 文字まで）が表示されます。

`char*` 型の値の出力時にフォーマット指定子 `%x` を使用して、ポインタ値を 16 進数表記で出力するか、またはシステムマクロ `__toString` を使用して全文字列値を取得します。

予約済みのセットアップマクロ関数名についてのリファレンス情報

セットアップマクロの定義に使用できる、予約済みのセットアップマクロ関数があります。これらの予約済みの名前を使用することにより、実行中に関数が定義された段階で実行されます。詳細については、278 ページの *セットアップマクロ関数およびファイルの概要* を参照してください。

予約済みのセットアップマクロ関数の名前を以下の表に示します。

マクロ	説明
execUserPreload	ターゲットシステムとの通信確立後、ターゲットアプリケーションのダウンロード前に呼び出します。 このマクロは、データを適切にロードするために必要なメモリアドレス（ロケーション）/レジスタを初期化する場合に実装します。
execUserFlashInit	フラッシュローダが RAM にダウンロードされる前に一度呼び出します。通常、このマクロは、フラッシュローダが必要なメモリマップを設定する場合に実装します。このマクロは、フラッシュのプログラミング時に一度だけ呼び出します。また、フラッシュローダ機能用にのみ使用します。
execUserSetup	ターゲットアプリケーションのダウンロード後に一度呼び出します。 このマクロは、メモリマップ、ブレイクポイント、割込み、レジスタマクロファイルなどの設定を行う場合に実装します。
execUserFlashReset	フラッシュローダが RAM にダウンロードされた後、フラッシュローダの実行前に一度呼び出します。このマクロは、フラッシュのプログラミング時に一度だけ呼び出します。また、フラッシュローダ機能用にのみ使用します。
execUserPreReset	リセットコマンドの実行直前に呼び出します。 このマクロを実装して、必要なデバイスの状態を設定します。
execUserReset	リセットコマンドの実行直後に呼び出します。 このマクロは、データの設定 / 復元を行う場合に実装します。
execUserExit	デバッグセッションの終了時に一度呼び出します。 このマクロは、状態データなどの保存を行う場合に実装します。
execUserFlashExit	デバッグセッションの終了時に一度呼び出します。 このマクロは、状態データなどの保存を行う場合に実装します。このマクロは、フラッシュローダ機能用に使用します。

表 16: C-SPY セットアップマクロ



システム起動時に実行されるマクロファイル（execUserSetup を使用）で割り込みやブレークポイントを定義する場合は、システム終了時にそれらが削除（execUserExit を使用）されていることを確認してください。サンプルは SetupSimple.mac にあります。インフォメーションセンタの「*割り込みのシミュレーション*」を参照してください。

シミュレータでは割り込み設定がセッション終了後も保持されるため、削除していない場合、execUserSetup の実行ごとに重複して作成されます。これが原因で、実行速度が大幅に低下します。

C-SPY システムマクロについてのリファレンス情報

ここでは、各 C-SPY システムマクロのリファレンスを収録しています。

以下の表に定義済みシステムマクロをまとめています。

マクロ	説明
__cancelAllInterrupts	設定されたすべての割り込みを取り消します。
__cancelInterrupt	割り込みを取り消します。
__clearBreak	ブレークポイントを削除します。
__closeFile	__openFile で開かれたファイルを閉じます。
__delay	実行を遅らせます。
__disableInterrupts	割り込み生成を無効にします。
__driverType	ドライバタイプを確認します。
__emulatorSpeed	エミュレータクロック周波数を設定します。
__emulatorStatusCheckOnRead	各リード処理後の CPSR レジスタの検証を有効 / 無効にします。
__enableInterrupts	割り込み生成を有効にします。
__evaluate	入力文字列を式として解釈して、評価します。
__gdbserver_exec_command	文字列やコマンドを GDB サーバに送信します。
__hwReset	ハードウェアリセットを実行し、ターゲット CPU を停止します。
__hwResetRunToBp	ハードウェアリセットを実行した後、指定されたアドレスまで実行します。
__hwResetWithStrategy	ハードウェアリセットを実行し、ターゲット CPU を遅れて停止します。
__isBatchMode	C-SPY がバッチモードで実行中かどうかをチェックします。

表 17: システムマクロのまとめ

マクロ	説明
__jlinkExecCommand	低レベルコマンドを J-Link/J-Trace ドライバに送信します。
__jtagCommand	低レベルコマンドを JTAG 命令レジスタに送信します。
__jtagCP15IsPresent	コプロセッサ CP15 が使用できるかどうかを確認します。
__jtagCP15ReadReg	コプロセッサ CP15 のレジスタ値を返します。
__jtagCP15WriteReg	コプロセッサ CP15 レジスタにライトします。
__jtagData	低レベルデータ値を JTAG データレジスタに送信します。
__jtagRawRead	JTAG インタフェースからリードデータを戻します。
__jtagRawSync	蓄積されたデータを JTAG インタフェースにライトします。
__jtagRawWrite	JTAG に転送されるデータを蓄積します。
__jtagResetTRST	TRST JTAG 信号経由で ARM TAP コントローラをリセットします。
__loadImage	イメージをロードします。
__memoryRestore	ファイルの内容を指定したメモリゾーンに復元します。
__memorySave	指定したメモリエリアの内容をファイルに保存します。
__openFile	ファイルを入出力処理用に開きます。
__orderInterrupt	割り込みを生成します。
__popSimulatorInterruptExecutingStack	割り込みシミュレーションシステムに、割り込みハンドラの実行が終了したことを通知します。
__readFile	指定したファイルからリードします。
__readFileByte	指定したファイルから 1 バイトリードします。
__readMemory8,	指定したメモリアドレス（ロケーション）から 1 バイトリードします。
__readMemoryByte	
__readMemory16	指定したメモリアドレス（ロケーション）から 2 バイトリードします。
__readMemory32	指定したメモリアドレス（ロケーション）から 4 バイトリードします。
__registerMacroFile	指定したファイルからマクロを登録します。

表 17: システムマクロのまとめ (続き)

マクロ	説明
__resetFile	__openFile で開かれたファイル内の位置を先頭に戻します。
__restoreSoftwareBreakpoints	システム起動中に破壊されたブレークポイントを復元します。
__setCodeBreak	コードブレークポイントを設定します。
__setDataBreak	データブレークポイントを設定します。
__setLogBreak	ログブレークポイントを設定します。
__setSimBreak	シミュレーションブレークポイントを設定します。
__setTraceStartBreak	トレース開始ブレークポイントを設定します。
__setTraceStopBreak	トレース停止ブレークポイントを設定します。
__sourcePosition	現在の実行位置がソース位置に対応する場合、ファイル名とソース位置を返します。
__strFind	指定した文字列で別の文字列を検索します。
__subString	文字列から部分文字列を抽出します。
__targetDebuggerVersion	ターゲットデバッガのバージョンを返します。
__toLower	パラメータ文字列のコピーを、すべての文字を小文字に変換して返します。
__toString	文字列を出力します。
__toUpper	パラメータ文字列のコピーを、すべての文字を大文字に変換して返します。
__unloadImage	デバッグイメージをアンロードします。
__writeFile	指定したファイルにライトします。
__writeFileByte	指定したファイルに 1 バイトライトします。
__writeMemory8,	指定したメモリアドレス（ロケーション）に 1 バイトライトします。
__writeMemoryByte	
__writeMemory16	指定したメモリアドレス（ロケーション）に 2 バイトワードをライトします。
__writeMemory32	指定したメモリアドレス（ロケーション）に 4 バイトワードをライトします。

表 17: システムマクロのまとめ（続き）

__cancelAllInterrupts

構文	<code>__cancelAllInterrupts()</code>
リターン値	<code>int 0</code>
説明	設定されたすべての割込みを取り消します。
適用範囲	このシステムマクロは、C-SPY シミュレータでのみ使用できます。

__cancelInterrupt

構文	<code>__cancelInterrupt(interrupt_id)</code>						
パラメータ	<table><tr><td><code>interrupt_id</code></td><td>対応する <code>__orderInterrupt</code> マクロ呼出し (unsigned long) で返される値</td></tr></table>	<code>interrupt_id</code>	対応する <code>__orderInterrupt</code> マクロ呼出し (unsigned long) で返される値				
<code>interrupt_id</code>	対応する <code>__orderInterrupt</code> マクロ呼出し (unsigned long) で返される値						
リターン値	<table><tr><th>結果</th><th>値</th></tr><tr><td>成功</td><td><code>int 0</code></td></tr><tr><td>失敗</td><td>ゼロ以外のエラー番号</td></tr></table>	結果	値	成功	<code>int 0</code>	失敗	ゼロ以外のエラー番号
結果	値						
成功	<code>int 0</code>						
失敗	ゼロ以外のエラー番号						
表 18: <code>__cancelInterrupt</code> のリターン値							
説明	指定した割込みを取り消します。						
適用範囲	このシステムマクロは、C-SPY シミュレータでのみ使用できます。						

__clearBreak

構文	<code>__clearBreak(break_id)</code>		
パラメータ	<table><tr><td><code>break_id</code></td><td>設定したブレークポイントマクロのいずれかが返した値</td></tr></table>	<code>break_id</code>	設定したブレークポイントマクロのいずれかが返した値
<code>break_id</code>	設定したブレークポイントマクロのいずれかが返した値		
リターン値	<code>int 0</code>		
説明	ユーザ定義ブレークポイントを削除します。		
関連項目	119 ページの <i>ブレークポイントの使用</i> 。		

__closeFile

構文	__closeFile(fileHandle)	
パラメータ	fileHandle	__openFile マクロでファイルハンドルとして使用するマクロ変数
リターン値	int 0	
説明	先に __openFile で開いたファイルを閉じます。	

__delay

構文	__delay(value)	
パラメータ	value	実行を遅らせる時間（ミリ秒）
リターン値	int 0	
説明	指定したミリ秒数だけ実行を遅らせます。	

__disableInterrupts

構文	__disableInterrupts()	
リターン値	結果	値
	成功	int 0
	失敗	ゼロ以外のエラー番号

表 19: __disableInterrupts のリターン値

説明	割込み生成を無効にします。
適用範囲	このシステムマクロは、C-SPY シミュレータでのみ使用できます。

__driverType

構文

`__driverType(driver_id)`

パラメータ

`driver_id` チェックするドライバに対応する文字列。以下のいずれかです。

"angel" は Angel ドライバに対応します。
"gdbserv" は GDB ドライバに対応します。
"generic" はサードパーティのドライバに対応します。
"jlink" は J-Link/J-Trace ドライバに対応します。
"jtag" は Macraigor ドライバに対応します。
"lmiftdi" は TI Stellaris FTDI ドライバに対応します。
"rdi" は RDI ドライバに対応します。
"rom" は IAR ROM モニタドライバに対応します。
"sim" はシミュレータドライバに対応します。
"stlink" は ST-LINK ドライバに対応します。

リターン値

結果	値
成功	1
失敗	0

表 20: __driverType のリターン値

説明

現在の C-SPY ドライバが、`driver_id` パラメータで指定したドライバタイプと同一かどうかを確認します。

例

`__driverType("sim")`

シミュレータが現在のドライバである場合は、1 が返されます。それ以外の場合は 0 が返されます。

__emulatorSpeed

構文

`__emulatorSpeed(speed)`

パラメータ

`speed` エミュレータ速度（単位は Hz）です。0（ゼロ）を使用すると、自動的に検出された速度にします。-1 を使用すると、適用する速度にします（アダプティブ速度をサポートしているエミュレータのみ）。

リターン値	結果	値
	成功	前の速度、あるいは未知の場合は 0（ゼロ）
	不成功、エミュレータが未サポートの速度	-1

表 21: `__emulatorSpeed` のリターン値

説明	エミュレータクロック周波数を設定します。JTAG インタフェースの場合、TCK 信号に見られる JTAG クロック周波数です。
適用範囲	このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。
例	<pre>__emulatorSpeed(0)</pre> <p>自動的に検出するエミュレータ速度を設定します。</p>

`__emulatorStatusCheckOnRead`

構文	<code>__emulatorStatusCheckOnRead(status)</code>		
パラメータ	<table><tr><td><i>status</i></td><td>0 は、チェックを有効にします（デフォルト）。 1 は、チェックを無効にします</td></tr></table>	<i>status</i>	0 は、チェックを有効にします（デフォルト）。 1 は、チェックを無効にします
<i>status</i>	0 は、チェックを有効にします（デフォルト）。 1 は、チェックを無効にします		
リターン値	int 0		
説明	<p>各リード処理後に行う CPSR（最新プロセッサステータスレジスタ）のドライバ検証を有効 / 無効にします。一般的にこのマクロは、Texas Instruments TMS470R1B1M など、いくつかの CPU で JTAG 接続を開始する場合に使用できます。</p> <p>注：この検証を有効にすると、たとえば無効な CPSR 値が戻された場合などに、CPU で問題の発生する原因となる場合があります。ただし、この検証が無効であると (<code>SetCheckModeAfterRead = 0</code>)、リード処理の成功を検証できず、またデータが中断する可能性を検出することができません。</p>		
適用範囲	このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。		
例	<pre>__emulatorStatusCheckOnRead(1)</pre> <p>メモリリード時のデータ中止のチェックを無効にします。</p>		

__enableInterrupts

構文 `__enableInterrupts()`

リターン値

結果	値
成功	int 0
失敗	ゼロ以外のエラー番号

表 22: __enableInterrupts のリターン値

説明 割り込み生成を有効にします。

適用範囲 このシステムマクロは、C-SPY シミュレータでのみ使用できます。

__evaluate

構文 `__evaluate(string, valuePtr)`

パラメータ

<i>string</i>	式文字列
<i>valuePtr</i>	結果を保持するマクロ変数ポインタ

リターン値

結果	値
成功	int 0
失敗	int 1

表 23: __evaluate リターン値

説明 このマクロは入力文字列を式として解釈して、評価します。結果は *valuePtr* で参照される変数に格納されます。

例 以下の例では、変数 *i* が定義され、値 5 に設定されていると仮定しています。
`__evaluate("i + 3", &myVar)`
マクロ変数 *myVar* に値 8 が割り当てられます。

__gdbserver_exec_command

構文	__gdbserver_exec_command("string")	
パラメータ	"string"	GDB サーバに送信する文字列またはコマンドです。 詳細についてはドキュメントを参照してください
説明	このオプションは、文字列やコマンドを GDB サーバに送信するときに使用します。	
適用範囲	このシステムマクロは、GDB サーバインタフェースに使用できます。	

__hwReset

構文

__hwReset(halt_delay)

パラメータ

halt_delay

リセットパルスの終わりと CPU の停止との間の遅延（マイクロ秒単位）です。0（ゼロ）を設定すると、リセット後すぐに CPU を停止します

リターン値

結果	値
成功。実際の遅延値がエミュレータで実現	>=0
成功。この遅延機能はエミュレータが未サポート	-1
不成功。ハードウェアリセットはエミュレータが未サポート	-2

表 24: __hwReset のリターン値

説明

ハードウェアリセットを実行し、ターゲット CPU を停止します。

適用範囲

このシステムマクロは、すべての JTAG インタフェースに使用できます。

例

__hwReset(0)

CPU をリセットし、ただちに停止します。

__hwResetRunToBp

構文

```
__hwResetRunToBp(strategy, breakpoint_address, timeout)
```

パラメータ

<i>strategy</i>	サポートされているリセット方式について詳しくは、『 <i>ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド</i> 』を参照してください。
<i>breakpoint_address</i>	実行を停止するブレークポイントのアドレスです。整数値を指定します（シンボルは使用できません）。
<i>timeout</i>	ブレークポイントのタイムアウト（ミリ秒）。指定した時間内にブレークポイントに達しない場合、コアが停止します。

リターン値

値	結果
>=0	成功。ブレークポイントにヒットするまでのおよその実行時間 (ms) です。
-2	不成功。ハードウェアリセットはエミュレータでサポートされていません。
-3	不成功。このリセット方式はエミュレータでサポートされていません。

表 25: __hwResetRunToBp のリターン値

説明

ハードウェアリセット、指定アドレスでのブレークポイントの設定、ブレークポイントまでの実行、ブレークポイントの削除を行います。ブレークポイントアドレスは、ダウンロードされたイメージが RAM にコピーされた後の開始アドレスです。

このマクロは、アプリケーションイメージをフラッシュから RAM にコピーするブートローダの実行を目的としています。イメージがフラッシュにダウンロードされた後、イメージが検証される前に実行されます。このマクロは、execUserFlashExit または execUserPreload で実行できます。

適用範囲

このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。

例

```
__hwResetRunToBp(0, 0x4000000, 10000)
```

リセット方法 0 を使用して CPU をリセットし、アドレス 0x4000000 まで実行します。10 秒以内にブレークポイントに達しない場合には、指定されたタイムアウト時間に従って停止を実行します。

__hwResetWithStrategy

構文	__hwResetWithStrategy(halt_delay, strategy)	
パラメータ	halt_delay	リセットパルスの終わりと CPU の停止との間の遅延（マイクロ秒）です。0（ゼロ）を設定すると、リセット後すぐに CPU を停止します（strategy が 0 に設定されている場合にのみ）。
	strategy	サポートされているリセット方式について詳しくは、『ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド』を参照してください。

リターン値

結果	値
成功。実際の遅延（ミリ秒）。エミュレータで実現	>=0
成功。この遅延機能はエミュレータが未サポート	-1
不成功。ハードウェアリセットはエミュレータが未サポート	-2
不成功。このリセット方式はエミュレータが未サポート	-3

表 26: __hwResetWithStrategy のリターン値

説明	ハードウェアリセットを実行し、ターゲット CPU を遅れて停止します。
適用範囲	このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。
例	<pre>__hwResetWithStrategy(0,1)</pre> <p>CPU をリセットし、メモリアドレスがゼロのブレイクポイントを使用して停止します。</p>

__isBatchMode

構文	__isBatchMode()	
リターン値	結果	値
	True	int 1
	False	int 0

表 27: __isBatchMode のリターン値

説明	このマクロは、デバッガがバッチモードで実行中の場合に True（真）を、それ以外の場合は False（偽）を返します。
----	---

__jlinkExecCommand

構文	<code>__jlinkExecCommand(cmdstr)</code>	
パラメータ	<code>cmdstr</code>	J-Link/J-Trace コマンド文字列
リターン値	<code>int 0</code>	
説明	低レベルコマンドを J-Link/J-Trace ドライバに送信するには、『 <i>ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド</i> 』を参照してください。	
適用範囲	このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。	

__jtagCommand

構文	<code>__jtagCommand(ir)</code>	
パラメータ	ここで <i>ir</i> は以下のいずれかです。	
	2	SCAN_N
	4	RESTART
	12	INTEST
	14	IDCODE
	15	BYPASS
リターン値	<code>int 0</code>	
説明	低レベルコマンドを JTAG 命令レジスタ IR に送信します。	
適用範囲	このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。	
例	<pre>__jtagCommand(14); Id = __jtagData(0,32);</pre> <p>ARM ターゲットデバイスの JTAG ID を戻します。</p>	

__jtagCP15IsPresent

構文	<code>__jtagCP15IsPresent()</code>
リターン値	1 (CP15 が使用できる場合)。それ以外の場合は 0。
説明	コプロセッサ CP15 が使用できるかどうかを確認します。
適用範囲	このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。

__jtagCP15ReadReg

構文	<code>__jtagCP15ReadReg(CRn, CRm, op1, op2)</code>
パラメータ	MRC 命令のパラメータ (レジスタおよびオペランド)。詳細については、『 <i>ARM Architecture Reference Manual</i> 』を参照してください。op1 は常に 0 である必要がある点に注意してください。
リターン値	レジスタ値です。
説明	CP15 レジスタの値を呼び込み、その値を戻します。
適用範囲	このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。

__jtagCP15WriteReg

構文	<code>__jtagCP15WriteReg(CRn, CRm, op1, op2, value)</code>
パラメータ	MRC 命令のパラメータ (レジスタおよびオペランド)。詳細については、『 <i>ARM Architecture Reference Manual</i> 』を参照してください。op1 は、常に 0。value が書き込まれる値である点に注意してください。
説明	CP15 レジスタに値を書込みます。
適用範囲	このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。

__jtagData

構文

```
__jtagData(dr, bits)
```

パラメータ

<i>dr</i>	32 ビットデータレジスタ値
<i>bits</i>	<i>dr</i> の有効ビット数です。マクロパラメータおよびリターン値の両方に対応します。最下位ビットから始まります (1...32)

リターン値

処理結果を戻します。結果のビット数は *bits* パラメータで指定されます。

説明

低レベルデータ値を JTAG データレジスタ DR に送信します。DR からシフトされたビットが戻されます。

適用範囲

このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。

例

```
__jtagCommand(14);
Id = __jtagData(0,32);
```

ARM ターゲットデバイスの JTAG ID を戻します。

__jtagRawRead

構文

```
__jtagRawRead(bitpos, numbits)
```

パラメータ

<i>bitpos</i>	戻された JTAG ビットの開始ビット位置で、そこからデータを戻します
<i>numbits</i>	読み込みビット数です。最大値は 32 です

説明

JTAG TDO から読み込まれたデータを戻します。最下位ビットだけはデータを含みます。最下位ビットリードは最下位ビットから行われます。任意の数だけこの関数を呼び出すと、操作で戻すすべてのビットを取得することができます。この関数は、蓄積された書込みビットの同期を間接的に取ることも行います。

適用範囲

このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。

例

以下は、TMS および TDI ピンにある JTAG ヘデータを書き込む方法と、TDO からデータを読み込む擬似コードを示します。

```
__var Id;
__var BitPos;
/*****
 *
 * ReadId()
 */
ReadId() {
__message "Reading JTAG Id\n";
__jtagRawWrite(0, 0x1f, 6); /*RESET ステート経由で IDLE へ移動 */
__jtagRawWrite(0, 0x1, 3); /*DR スキャンチェーンを入力 */
BitPos = __jtagRawWrite(0, 0x80000000, 32); /*32 ビットを
DR にシフト。Read 操作の BitPos を記憶 */
__jtagRawWrite(0, 0x1, 2); /*IDLE へ移動 */
Id = __jtagRawRead(BitPos, 32); /*Id を読み込み */
__message "JTAG Id: ", Id:%x, "\n";
}
```

__jtagRawSync

構文

```
__jtagRawSync()
```

リターン値

```
int 0
```

説明

任意のデータを JTAG インタフェースに送信します。__jtagRawWrite を使用して蓄積されたすべてのビットは、JTAG スキャンチェーンに書き込まれます。このデータは TCK と同期を取って送信され、通常は TCK の立上がりエッジ上のデバイスによってサンプリングされます。

適用範囲

このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。

例

以下は、TMS および TDI ピンにある JTAG ヘデータを書き込む方法と、TDO からデータを読み込む擬似コードを示します。

```
int i;
U32 tdo;
for (i = 0; i < numBits; i++) {
    TDI = tdi & 1; /*TDI ピンを設定 */
    TMS = tms & 1; /*TMS ピンを設定 */
    TCK = 0;
    TCK = 1;
```

```

tdo <= 1;
if (TDO) {
    tdo |= 1;
}
tdi >= 1;
tms >= 1;
}

```

__jtagRawWrite

構文

```
__jtagRawWrite(tdi, tms, numbits)
```

パラメータ

<i>tdi</i>	TDI ピンに出力されるデータです。このデータは、最初に最下位ビットから送信されます。
<i>tms</i>	TMS ピンに出力されるデータです。このデータは、最初に最下位ビットから送信されます。
<i>numbits</i>	転送ビット数です。各ビットは、JTAG TCK ラインの立ち下がりエッジ、および立ち上がりエッジとなります。最大値は 64 です。

リターン値

蓄積されたパケットのデータのビット位置を戻します。通常、この値は JTAG からデータを読み込むときに使用されます。

説明

JTAG に転送されたビットを蓄積します。32 ビットで不十分な場合、この関数を何回も呼び出すことができます。両方のデータ出力ライン（TMS および TDI）は、別々に制御できます。

適用範囲

このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。

例

```

/*TMS の 1 ビットを 5 つ送信し、TAP-RESET 状態に移動する */
__jtagRawWrite(0x1F, 0, 5); /* バッファにビットを格納 */
__jtagRawSync(); /* 転送バッファ、書込み tms、tdi 読み込み tdo */
ARM ターゲットデバイスの JTAG ID を戻します。

```

__jtagResetTRST

構文

__jtagResetTRST()

リターン値

結果	値
成功	int 0
失敗	ゼロ以外のエラー番号

表 28: __jtagResetTRST のリターン値

説明

TRST JTAG 信号経由で ARM TAP コントローラをリセットします。

適用範囲

このシステムマクロは、J-Link/J-Trace インタフェースに使用できます。

__loadImage

構文

__loadImage(path, offset, debugInfoOnly)

パラメータ

path	ダウンロードするイメージのパスを識別する文字列。パスは絶対パスか、引数変数を使用する必要があります。引数変数について詳しくは、『ARM 用 IDE プロジェクト管理およびビルドガイド』を参照してください。
offset	ダウンロードされたイメージの目的地のアドレスのオフセットを識別する整数。
debugInfoOnly	ゼロ以外の値の場合、ターゲットシステムにコードやデータはダウンロードされません。すなわち、C-SPY はデバッグファイルからデバッグ情報を読み込むのみとなります。0 の場合、ダウンロードされます。

リターン値

値	結果
ゼロ以外整数値	固有のモジュール識別子
int 0	ロードに失敗しました

表 29: __loadImage のリターン値

説明

イメージ（デバッグファイル）をロードします。

例 1

システムが ROM ライブラリとアプリケーションで構成されているとします。アプリケーションはアクティブプロジェクトですが、ライブラリに対応するデバッグファイルがあります。この場合には、C-SPY マクロファイルの `execUserSetup` マクロに以下のマクロ呼出しを追加し、これを自分のプロジェクトに関連付けます。

```
__loadImage(ROMfile, 0x8000, 1);
```

このマクロ呼出しは、ROM ライブラリ `ROMfile` のデバッグ情報をロードし、その内容はダウンロードしません（内容はすでに ROM にあると推定されるため）。すると、ライブラリと一緒にアプリケーションのデバッグが可能になります。

例 2

システムが ROM ライブラリとアプリケーションで構成されており、主にライブラリに心配があるとします。ライブラリは、デバッグセッションの前にフラッシュメモリにプログラミングする必要があります。したがって、ライブラリの開発中は、ライブラリプロジェクトが IDE でアクティブプロジェクトでなければなりません。この場合には、C-SPY マクロファイルの `execUserSetup` マクロに以下のマクロ呼出しを追加し、これを自分のプロジェクトに関連付けます。

```
__loadImage(ApplicationFile, 0x8000, 0);
```

このマクロ呼出しは、アプリケーションのデバッグ情報をロードし、その内容をダウンロードします（通常は RAM に）。すると、アプリケーションと一緒にライブラリのデバッグが可能になります。

関連項目

369 ページの *イメージ*、62 ページの *複数イメージのロード*。

__memoryRestore

構文

```
__memoryRestore(zone, filename)
```

パラメータ

ゾーン	メモリゾーン名（文字列）。指定可能なゾーンのリストについては、「156 ページの <i>C-SPY</i> メモリゾーン」を参照してください。
filename	読み込むファイルを指定する文字列。filename にはパスを含める必要があります。パスは絶対パスか、引数変数を使用しなければなりません。引数変数について詳しくは、『 <i>ARM 用 IDE プロジェクト管理およびビルドガイド</i> 』を参照してください。

リターン値

```
int 0
```

説明	ファイルの内容を読み込んで、指定したメモリゾーンに保存します。
例	<code>__memoryRestore("Memory", "c:¥¥temp¥¥saved_memory.hex");</code>
関連項目	164 ページの [メモリ復元] ダイアログボックス。

__memorySave

構文	<code>__memorySave(start, stop, format, filename)</code>								
パラメータ	<table><tr><td><i>start</i></td><td>保存するメモリエリアの最初の位置を指定する文字列。</td></tr><tr><td><i>stop</i></td><td>保存するメモリエリアの最後の位置を指定する文字列。</td></tr><tr><td><i>format</i></td><td>保存したメモリに使用されるフォーマットを指定する文字列。 以下から選択します。 Intel-extended motorola motorola-s19 motorola-s28 motorola-s37</td></tr><tr><td><i>filename</i></td><td>書き込むファイルを指定する文字列。filename にはパスを含める必要があります。パスは絶対パスか、引数変数を使用しなければなりません。引数変数について詳しくは、『<i>ARM 用 IDE プロジェクト管理およびビルドガイド</i>』を参照してください。</td></tr></table>	<i>start</i>	保存するメモリエリアの最初の位置を指定する文字列。	<i>stop</i>	保存するメモリエリアの最後の位置を指定する文字列。	<i>format</i>	保存したメモリに使用されるフォーマットを指定する文字列。 以下から選択します。 Intel-extended motorola motorola-s19 motorola-s28 motorola-s37	<i>filename</i>	書き込むファイルを指定する文字列。 filename にはパスを含める必要があります。パスは絶対パスか、引数変数を使用しなければなりません。引数変数について詳しくは、『 <i>ARM 用 IDE プロジェクト管理およびビルドガイド</i> 』を参照してください。
<i>start</i>	保存するメモリエリアの最初の位置を指定する文字列。								
<i>stop</i>	保存するメモリエリアの最後の位置を指定する文字列。								
<i>format</i>	保存したメモリに使用されるフォーマットを指定する文字列。 以下から選択します。 Intel-extended motorola motorola-s19 motorola-s28 motorola-s37								
<i>filename</i>	書き込むファイルを指定する文字列。 filename にはパスを含める必要があります。パスは絶対パスか、引数変数を使用しなければなりません。引数変数について詳しくは、『 <i>ARM 用 IDE プロジェクト管理およびビルドガイド</i> 』を参照してください。								
リターン値	int 0								
説明	指定したメモリエリアの内容をファイルに保存します。								
例	<code>__memorySave("Memory:0x00", "Memory:0xFF", "intel-extended", "c:¥¥temp¥¥saved_memory.hex");</code>								
関連項目	163 ページの [メモリ保存] ダイアログボックス。								

__openFile

構文

```
__openFile(filename, access)
```

パラメータ

filename 開くファイル。 *filename* にはパスを含める必要があります。パスは絶対パスか、引数変数を使用しなければなりません。引数変数について詳しくは、『*ARM 用 IDE プロジェクト管理 およびビルドガイド*』を参照してください。

access アクセスタイプ（文字列）。

以下は必須ですが、混在できません（互いに排他的）。

"a" 付加。開かれたファイルの最後に新しいデータが付加されます。

"r" リード

"w" ライト

以下はオプションです。これらは混在できません（互いに排他的）。

"b" バイナリ。ファイルをバイナリモードで開きます。

"t" ASCII テキスト。ファイルをテキストモードで開きます。

以下のアクセスタイプはオプションです。

"+" "r"、"w"、"a; r+" または "w+ と併用する場合は *read* および *write*"、"a+" の場合は *read* と *append* です。

リターン値

結果	値
成功	ファイルハンドル
失敗	無効なファイルハンドル（評価結果は False）

表 30: __openFile のリターン値

説明

ファイルを入出力処理用に開きます。このマクロのデフォルトの基準ディレクトリは、開かれているプロジェクトファイル (*.ewp) のある場所になります。__openFile の引数では、このディレクトリとの相対位置を指定できます。また、\$PROJ_DIR\$ や \$TOOLKIT_DIR\$ などの引数変数をパスとして指定することもできます。

例	<pre>__var myFileHandle; /* ファイルハンドルを保管する */ /* マクロ変数 */ myFileHandle = __openFile("\$PROJ_DIR\$¥¥Debug¥¥Exe¥¥test.tst", "r"); if (myFileHandle) { /* 正常に開きます */ }</pre>
関連項目	引数変数について詳しくは、『 <i>ARM 用 IDE プロジェクト管理およびビルドガイド</i> 』を参照してください。

__orderInterrupt

構文	<pre>__orderInterrupt(<i>specification</i>, <i>first_activation</i>, <i>repeat_interval</i>, <i>variance</i>, <i>infinite_hold_time</i>, <i>hold_time</i>, <i>probability</i>)</pre>														
パラメータ	<table><tr><td><i>specification</i></td><td>割込み（文字列）。<i>specification</i> には、デバイス記述ファイル (ddf) で使用されている仕様全体か、名前のみを指定できます。名前のみを指定した場合は、割込みシステムはデバイス記述ファイルから詳細を自動的に取得します</td></tr><tr><td><i>first_activation</i></td><td>サイクル単位で指定した最初の実行時間（整数）</td></tr><tr><td><i>repeat_interval</i></td><td>サイクル単位で指定した周期（整数）</td></tr><tr><td><i>variance</i></td><td>パーセントで指定したタイミング変動範囲（0 ～ 100 の 整数）</td></tr><tr><td><i>infinite_hold_time</i></td><td>無制限の場合は 1、それ以外の場合は 0</td></tr><tr><td><i>hold_time</i></td><td>ホールド時間（整数）</td></tr><tr><td><i>probability</i></td><td>パーセントで指定した確立（0 ～ 100 の 整数）</td></tr></table>	<i>specification</i>	割込み（文字列）。 <i>specification</i> には、デバイス記述ファイル (ddf) で使用されている仕様全体か、名前のみを指定できます。名前のみを指定した場合は、割込みシステムはデバイス記述ファイルから詳細を自動的に取得します	<i>first_activation</i>	サイクル単位で指定した最初の実行時間（整数）	<i>repeat_interval</i>	サイクル単位で指定した周期（整数）	<i>variance</i>	パーセントで指定したタイミング変動範囲（0 ～ 100 の 整数）	<i>infinite_hold_time</i>	無制限の場合は 1、それ以外の場合は 0	<i>hold_time</i>	ホールド時間（整数）	<i>probability</i>	パーセントで指定した確立（0 ～ 100 の 整数）
<i>specification</i>	割込み（文字列）。 <i>specification</i> には、デバイス記述ファイル (ddf) で使用されている仕様全体か、名前のみを指定できます。名前のみを指定した場合は、割込みシステムはデバイス記述ファイルから詳細を自動的に取得します														
<i>first_activation</i>	サイクル単位で指定した最初の実行時間（整数）														
<i>repeat_interval</i>	サイクル単位で指定した周期（整数）														
<i>variance</i>	パーセントで指定したタイミング変動範囲（0 ～ 100 の 整数）														
<i>infinite_hold_time</i>	無制限の場合は 1、それ以外の場合は 0														
<i>hold_time</i>	ホールド時間（整数）														
<i>probability</i>	パーセントで指定した確立（0 ～ 100 の 整数）														
リターン値	このマクロは、割込み識別子 (unsigned long) を返します。 <i>specification</i> の構文に誤りがある場合は、-1 を返します。														
説明	割込みを生成します。														

- 適用範囲

このシステムマクロは、C-SPY シミュレータでのみ使用できます。
- 例

以下の例では、4000 サイクルの後に初めて実行された保持時間無制限の周期割込みを生成します。

```
__orderInterrupt( "IRQ", 4000, 2000, 0, 1, 0, 100 );
```

__popSimulatorInterruptExecutingStack

- 構文

__popSimulatorInterruptExecutingStack(void)
- リターン値

このマクロにはリターン値はありません。
- 説明

割込みシミュレーションシステムに、割込みハンドラ実行から戻る際に使用される通常の命令と同様に、割込みハンドラの実行が終了したことを通知します。

割込みハンドラから復帰する際に通常の命令を使用しない場合（タスク切替え機能のあるオペレーティングシステムの場合など）に使用します。この場合、割込みシミュレーションシステムは割込みハンドラの実行が終了したことを自動的には検出できません。
- 適用範囲

このシステムマクロは、C-SPY シミュレータでのみ使用できます。
- 関連項目

262 ページの マルチタスクシステムでの割込みシミュレーション。

__readFile

- 構文

__readFile(fileHandle, valuePtr)
- パラメータ

fileHandle	__openFile マクロでファイルハンドルとして使用するマクロ変数
valuePtr	変数へのポインタ
- リターン値

結果	値
成功	0
失敗	ゼロ以外のエラー番号

表 31: __readFile リターン値

説明	指定したファイルから 16 進数を順にリードし、unsigned long に変換して、value パラメータに代入します。この値は、マクロ変数へのポインタになります。
例	<pre>__var number; if (__readFile(myFileHandle, &number) == 0) { // 数字を処理します }</pre>

__readFileByte

構文	<code>__readFileByte(fileHandle)</code>		
パラメータ	<table><tr><td><i>fileHandle</i></td><td><code>__openFile</code> マクロでファイルハンドルとして使用するマクロ変数</td></tr></table>	<i>fileHandle</i>	<code>__openFile</code> マクロでファイルハンドルとして使用するマクロ変数
<i>fileHandle</i>	<code>__openFile</code> マクロでファイルハンドルとして使用するマクロ変数		
リターン値	エラー、ファイル終端時に -1 それ以外は、0 ～ 255。		
説明	<code>file</code> で指定したファイルから 1 バイトリードします。		
例	<pre>__var byte; while ((byte = __readFileByte(myFileHandle)) != -1) { /* バイトを処理します */ }</pre>		

__readMemory8, __readMemoryByte

構文	<code>__readMemory8(address, zone)</code> <code>__readMemoryByte(address, zone)</code>				
パラメータ	<table><tr><td><i>address</i></td><td>メモリアドレス（整数）</td></tr><tr><td><i>ゾーン</i></td><td>メモリゾーン名（文字列）。使用可能なゾーンの詳細は、156 ページの <i>C-SPY</i> メモリゾーンを参照してください</td></tr></table>	<i>address</i>	メモリアドレス（整数）	<i>ゾーン</i>	メモリゾーン名（文字列）。使用可能なゾーンの詳細は、156 ページの <i>C-SPY</i> メモリゾーンを参照してください
<i>address</i>	メモリアドレス（整数）				
<i>ゾーン</i>	メモリゾーン名（文字列）。使用可能なゾーンの詳細は、156 ページの <i>C-SPY</i> メモリゾーンを参照してください				
リターン値	このマクロは、メモリから取得した値を返します。				
説明	指定したメモリアドレス（ロケーション）から 1 バイトリードします。				
例	<code>__readMemory8(0x0108, "Memory");</code>				

__readMemory16

構文	<code>__readMemory16(address, zone)</code>	
パラメータ	<i>address</i>	メモリアドレス (整数)
	<i>ゾーン</i>	メモリゾーン名 (文字列)。使用可能なゾーンの詳細は、 <i>156 ページの C-SPY メモリゾーンを参照してください</i>
リターン値	このマクロは、メモリから取得した値を返します。	
説明	指定したメモリアドレス (ロケーション) から 2 バイトワードをリードします。	
例	<code>__readMemory16(0x0108, "Memory");</code>	

__readMemory32

構文	<code>__readMemory32(address, zone)</code>	
パラメータ	<i>address</i>	メモリアドレス (整数)
	<i>ゾーン</i>	メモリゾーン名 (文字列)。使用可能なゾーンの詳細は、 <i>156 ページの C-SPY メモリゾーンを参照してください</i>
リターン値	このマクロは、メモリから取得した値を返します。	
説明	指定したメモリアドレス (ロケーション) から 4 バイトワードをリードします。	
例	<code>__readMemory32(0x0108, "Memory");</code>	

__registerMacroFile

構文	<code>__registerMacroFile(filename)</code>	
パラメータ	<i>filename</i>	登録するマクロが記述されたファイル (文字列) <i>filename</i> にはパスを含める必要があります。パスは絶対パスか、引数変数を使用しなければなりません。引数変数について詳しくは、『 <i>ARM 用 IDE プロジェクト管理およびビルドガイド</i> 』を参照してください。

リターン値	int 0
説明	セットアップマクロファイルからマクロを登録します。この関数を使用して、複数のマクロファイルを C-SPY 起動時に登録できます。
例	<code>__registerMacroFile("c:¥¥testdir¥¥macro.mac");</code>
関連項目	282 ページの セットアップマクロとセットアップファイルによる登録と実行。

__resetFile

構文	<code>__resetFile(fileHandle)</code>
パラメータ	<div><div><code>fileHandle</code></div><div><code>__openFile</code> マクロでファイルハンドルとして使用するマクロ変数</div></div>
リターン値	int 0
説明	先に <code>__openFile</code> で開いたファイルを先頭に戻します。

__restoreSoftwareBreakpoints

構文	<code>__restoreSoftwareBreakpoints()</code>
リターン値	int 0
説明	<p>システム起動中に破壊されたブレークポイントを自動的に復元します。</p> <p>起動中に RAM にコピーしてから RAM で実行しているアプリケーションの場合に有効です。たとえば、リンカ構成ファイルのコードに <code>initialize by copy</code> ディレクティブを使用する場合、あるいはアプリケーションに <code>__ramfunc</code> 宣言関数がある場合に有効となることがあります。この場合、C-SPY デバッガが起動すると、すべてのブレークポイントは RAM のコピー中にオーバーライドされます。</p> <p>C-SPY はこのマクロを使用して、破棄されたブレークポイントを復元します。</p>
適用範囲	このシステムマクロは、J-Link/J-Trace と TI Stellaris FTDI インタフェース、Macraigor インタフェースに使用できます。

__setCodeBreak

構文	__setCodeBreak(location, count, condition, cond_type, action)	
パラメータ		
	location	位置を記述する文字列。以下から選択します。 C-SPY 式。値は、たとえばmain 関数など、有効なアドレスに評価されます。C-SPY 式の詳細については、98 ページのC-SPY 式を参照してください。 フォーム [ゾーン:hexaddress] 上、または単にhexaddress たとえばMemory:42 など) 上の絶対アドレス。ゾーンはC-SPY のメモリゾーンを参照し、アドレスがどのメモリに属するかを指定します。 C ソースコードのソースの場所で、構文は {filename}.row.col を使用します。たとえば {D:¥¥src¥¥prog.c}.22.3 は、ソースファイル prog.c の 22 行目の 3 文字目の位置にブレークポイントを設定します。[ソース位置] のタイプは、通常はコードのブレークポイントだけに使用します。
	count	実行を停止するまでのブレークポイント条件発生回数 (整数)
	condition	ブレークポイント条件 (文字列)
	cond_type	条件の種類。「CHANGED」または「TRUE」 (文字列)
	action	ブレークポイント検出時に評価される式 (通常はマクロ呼出し)

リターン値

結果	値
成功	ブレークポイントを一意に特定する符号なし整数。ブレークポイントを削除する際には、この値を使用する必要があります
失敗	0

表 32: __setCodeBreak のリターン値

説明	コードブレークポイント (プロセッサが指定位置で命令をフェッチする直前にトリガされるブレークポイント) を設定します
例	<pre>__setCodeBreak (" {D:¥¥src¥¥prog.c}.12.9", 3, "d>16", "TRUE", "ActionCode()");</pre>

以下の例は、ソース中の `main` というラベルにコードブレークポイントを設定します。

```
__setCodeBreak("main", 0, "1", "TRUE", "");
```

関連項目 *119 ページの ブレークポイントの使用。*

__setDataBreak

構文	<pre>__setDataBreak(location, count, condition, cond_type, access, action)</pre>	
パラメータ	<i>location</i>	<p>位置を記述する文字列。以下から選択します。</p> <p>C-SPY 式。値は、変数名など有効なアドレスに評価されます。たとえば、<code>my_var</code> は変数 <code>my_var</code> の位置を、<code>arr[3]</code> は配列 <code>arr</code> の 3 番目のエレメントの位置をそれぞれ参照します。いくつかの関数で同じ名前が宣言された静的変数については、特定の変数を参照するために構文 <code>my_func::my_static_variable</code> を使用してください。C-SPY 式の詳細については、98 ページの <i>C-SPY 式</i> を参照してください。</p> <p>フォーム <code>[ゾーン:hexaddress]</code> 上、または単に <code>hexaddress</code> (たとえば <code>Memory:42</code> など) 上の絶対アドレス。ゾーンは C-SPY のメモリゾーンを参照し、アドレスがどのメモリに属するかを指定します。</p> <p>C ソースコードのソースの場所で、構文は <code>{filename}.row.col</code> を使用します。たとえば <code>{D:\¥¥src¥¥prog.c}.22.3</code> は、ソースファイル <code>prog.c</code> の 22 行目の 3 文字目の位置にブレークポイントを設定します。<code>[ソース位置]</code> のタイプは、通常はコードのブレークポイントだけに使用します。</p>
	<i>count</i>	実行を停止するまでのブレークポイント条件発生回数 (整数)
	<i>condition</i>	ブレークポイント条件 (文字列)
	<i>cond_type</i>	条件の種類。「CHANGED」または「TRUE」(文字列)
	<i>access</i>	メモリアクセスタイプ: リードの場合は "R"、ライトの場合は "W"、リード/ライトの場合は "RW"
	<i>action</i>	ブレークポイント検出時に評価される式 (通常はマクロ呼出し)

リターン値

結果	値
成功	ブレイクポイントを一意に特定する符号なし整数。ブレイクポイントを削除する際には、この値を使用する必要があります
失敗	0

表 33: __setDataBreak のリターン値

説明

データブレイクポイント（プロセッサが指定位置でデータのリード/ライトを実行した直後にトリガされるブレイクポイント）を設定します。

適用範囲

このシステムマクロは、C-SPY シミュレータでのみ使用できます。

例

```
__var brk;
brk = __setDataBreak("Memory:0x4710", 3, "d>6", "TRUE",
                    "W", "ActionData()");
...
__clearBreak(brk);
```

関連項目

119 ページの *ブレイクポイントの使用*。

__setLogBreak

構文

```
__setLogBreak(location, message, msg_type, condition,
              cond_type)
```

パラメータ

location 位置を記述する文字列。以下から選択します。

C-SPY 式。値は、たとえば main 関数など、有効なアドレスに評価されます。C-SPY 式の詳細については、98 ページの *C-SPY 式* を参照してください。

フォーム *[ゾーン:hexaddress]* 上、または単に *hexaddress* (たとえば *Memory:42* など) 上の絶対アドレス。ゾーンは C-SPY のメモリゾーンを参照し、アドレスがどのメモリに属するかを指定します。

C ソースコードのソースの場所で、構文は *{filename}.row.col* を使用します。たとえば *{D:¥¥src¥¥prog.c}.22.3* は、ソースファイル *prog.c* の 22 行目の 3 文字目の位置にブレイクポイントを設定します。*[ソース位置]* のタイプは、通常はコードのブレイクポイントだけに使用します。

<i>message</i>	メッセージテキスト
<i>msg_type</i>	以下のメッセージタイプのどちらかを選択します。 TEXT：メッセージがそのまま書き込まれます。 ARGS：メッセージは、 C-SPY の式または文字列をコンマで区切ったリストとして認識されます。
<i>condition</i>	ブレークポイント条件（文字列）
<i>cond_type</i>	条件の種類。「CHANGED」または「TRUE」（文字列）

リターン値

結果	値
成功	ブレークポイントを一意に特定する符号なし整数。ブレークポイントを削除する際には、同じ値を使用する必要があります
失敗	0

表 34: `__setLogBreak` のリターン値

説明

ログブレークポイントを設定します。つまり、命令が指定の場所からフェッチされたときにトリガされるブレークポイントです。特定のマシン命令にブレークポイントを設定した場合は、命令の実行前にブレークポイントがトリガされ、実行が一時停止し、指定したメッセージが [C-SPY デバッグログ] ウィンドウに出力されます。

例

```
__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);
}
```

関連項目

289 ページの *フォーマットした出力*、119 ページの *ブレークポイントの使用*。

__setSimBreak

構文	__setSimBreak(location, access, action)	
パラメータ	location	位置を記述する文字列。以下から選択します。 C-SPY 式。値は、変数名など有効なアドレスに評価されます。たとえば、my_var は変数 my_var の位置を、arr[3] は配列 arr の 3 番目のエレメントの位置をそれぞれ参照します。いくつかの関数で同じ名前で宣言された静的変数については、特定の変数を参照するために構文 my_func::my_static_variable を使用してください。 C-SPY 式の詳細については、98 ページの C-SPY 式を参照してください。 フォーム [ゾーン:hexaddress] 上、または単に hexaddress たとえば Memory:42 など) 上の絶対アドレス。ゾーンは C-SPY のメモリゾーンを参照し、アドレスがどのメモリに属するかを指定します。 C ソースコードのソースの場所で、構文は {filename}.row.col を使用します。たとえば {D:¥¥src¥¥prog.c}.22.3 は、ソースファイル prog.c の 22 行目の 3 文字目の位置にブレークポイントを設定します。[ソース位置] のタイプは、通常はコードのブレークポイントだけに使用します。
	access	メモリアクセスタイプ: リードの場合は "R"、ライトの場合は "W"
	action	ブレークポイント検出時に評価される式 (通常はマクロ関数呼出し)

リターン値

結果	値
成功	ブレークポイントを一意に特定する符号なし整数。ブレークポイントを消去する際には、この値を使用する必要があります
失敗	0

表 35: __setSimBreak のリターン値

説明

このシステムマクロを使用して、一時的にのみ命令の実行を停止するイミディエイトブレークポイントを設定します。このブレークポイントを使用すると、プロセッサがある位置からデータを読み込む直前かある位置にデータを書き込んだ直後に、C-SPY マクロ関数を呼び出すことができます。アクションが終了すると、命令の実行が再開されます。

イミディエイトブレイクポイントは、メモリにマッピングされたさまざまな種類のデバイス（シリアルポートやタイマなど）をシミュレーションする場合に便利です。デバイスがメモリマッピングされた位置をプロセッサが読み込むと、**C-SPY** マクロ関数が実行されて適切なデータを供給します。逆に、デバイスがメモリマッピングされた位置にプロセッサが書き込むと、**C-SPY** マクロ関数が実行されて、書き込まれた値に応じた適切な動作を実行します。

適用範囲

このシステムマクロは、**C-SPY** シミュレータでのみ使用できます。

__setTraceStartBreak

構文

__setTraceStartBreak(location)

パラメータ

location

位置を記述する文字列。以下から選択します。

C-SPY 式。値は、たとえば main 関数など、有効なアドレスに評価されます。C-SPY 式の詳細については、98 ページの C-SPY 式を参照してください。
フォーム [ゾーン:hexaddress] 上、または単に hexaddress たとえば Memory:42 など）上の絶対アドレス。ゾーンは C-SPY のメモリゾーンを参照し、アドレスがどのメモリに属するかを指定します。
C ソースコードのソースの場所で、構文は {filename}.row.col を使用します。たとえば {D:¥¥src¥¥prog.c}.22.3 は、ソースファイル prog.c の 22 行目の 3 文字目の位置にブレイクポイントを設定します。[ソース位置] のタイプは、通常はコードのブレイクポイントだけに使用します。

リターン値

結果	値
成功	ブレイクポイントを一意に特定する符号なし整数。ブレイクポイントを削除する際には、同じ値を使用する必要があります
失敗	0

表 36: __setTraceStartBreak のリターン値

説明

指定の位置にブレイクポイントを設定します。そのブレイクポイントがトリガされると、トレースシステムが起動します。

適用範囲

このシステムマクロは、**C-SPY** シミュレータでのみ使用できます。

例

```

__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);
}

```

関連項目

119 ページの ブレークポイントの使用。

__setTraceStopBreak

構文

```
__setTraceStopBreak(location)
```

パラメータ

location

位置を記述する文字列。以下から選択します。

C-SPY 式。値は、たとえば main 関数など、有効なアドレスに評価されます。C-SPY 式の詳細については、98 ページの *C-SPY* 式を参照してください。

フォーム [ゾーン:hexaddress] 上、または単に hexaddress (たとえば Memory:42 など) 上の絶対アドレス。ゾーンは C-SPY のメモリゾーンを参照し、アドレスがどのメモリに属するかを指定します。

C ソースコードのソースの場所で、構文は {filename}.row.col を使用します。たとえば {D:¥¥src¥¥prog.c}.22.3 は、ソースファイル prog.c の 22 行目の 3 文字目の位置にブレークポイントを設定します。[ソース位置] のタイプは、通常はコードのブレークポイントだけに使用します。

リターン値

結果	値
成功	ブレークポイントを一意に特定する符号なし整数。ブレークポイントを削除する際には、同じ値を使用する必要があります
失敗	int 0

表 37: `__setTraceStopBreak` のリターン値

説明	指定の位置にブレークポイントを設定します。そのブレークポイントがトリガされると、トレースシステムが停止します。
適用範囲	このシステムマクロは、C-SPY シミュレータでのみ使用できます。
例	322 ページの <code>__setTraceStartBreak</code> を参照。
関連項目	119 ページの <code>ブレークポイントの使用</code> 。

`__sourcePosition`

構文

`__sourcePosition(linePtr, colPtr)`

パラメータ

<code>linePtr</code>	行番号を格納する変数へのポインタ
<code>colPtr</code>	列番号を格納する変数へのポインタ

リターン値

結果	値
成功	ファイル名文字列
失敗	空 ("") 文字列

表 38: `__sourcePosition` リターン値

説明	現在の実行位置がソース位置に対応する場合、このマクロはファイル名を文字列として返します。パラメータで参照する変数の値を、ソース位置の行番号と列番号に設定します。
----	--

`__strFind`

構文

`__strFind(macroString, pattern, position)`

パラメータ

<code>macroString</code>	検索先のマクロ文字列
--------------------------	------------

	<i>pattern</i>	検索対象の文字列パターン
	<i>position</i>	検索の開始位置。最初の位置は 0 です
リターン値	パターンが見つかった位置。文字列が見つからなかった場合は -1。	
説明	このマクロは、指定した文字列で別の文字列を検索します。	
例	<pre>__strFind("Compiler", "pile", 0) = 3 __strFind("Compiler", "foo", 0) = -1</pre>	
関連項目	287 ページの マクロ文字列。	

__subString

構文	<code>__subString(<i>macroString</i>, <i>position</i>, <i>length</i>)</code>	
パラメータ	<i>macroString</i>	部分文字列の抽出元のマクロ文字列
	<i>position</i>	部分文字列の開始位置。最初の位置は 0 です
	<i>length</i>	部分文字列の長さ
リターン値	指定したマクロ文字列から抽出した部分文字列。	
説明	このマクロは、文字列から部分文字列を抽出します。	
例	<pre>__subString("Compiler", 0, 2) 生成されたマクロ文字列に co が含まれます。 __subString("Compiler", 3, 4) 生成されたマクロ文字列に pile が含まれます。</pre>	
関連項目	287 ページの マクロ文字列。	

__targetDebuggerVersion

構文	<code>__targetDebuggerVersion</code>
リターン値	C-SPY デバッガのプロセッサモジュールのバージョン番号を表す文字列。

説明	このマクロは、C-SPY デバッガのプロセッサモジュールのバージョン番号を返します。		
例	<pre>__var toolVer; toolVer = __targetDebuggerVersion(); __message "The target debugger version is, ", toolVer;</pre>		
__toLower			
構文	<code>__toLower(<i>macroString</i>)</code>		
パラメータ	<i>macroString</i>	任意のマクロ文字列	
リターン値	変換後のマクロ文字列。		
説明	このマクロは、パラメータ文字列のコピーを、すべての文字を小文字に変換して返します。		
例	<pre>__toLower("IAR") 生成されたマクロ文字列に <code>iar</code> が含まれます。 __toLower("Mix42") 生成されたマクロ文字列 <code>mix42</code> が含まれます。</pre>		
関連項目	287 ページの <i>マクロ文字列</i> 。		
__toString			
構文	<code>__toString(<i>C_string</i>, <i>maxlength</i>)</code>		
パラメータ	<i>C_string</i>	NULL 終端 C 文字列	
	<i>maxlength</i>	返されるマクロ文字列の最大長	
リターン値	マクロ文字列。		
説明	このマクロを使用して、C 文字列（ <code>char*</code> または <code>char[]</code> ）をマクロ文字列に変換します。		

説明	すでにダウンロード済みのイメージからデバッグ情報をアンロードします。
関連項目	62 ページの 複数イメージのロード、369 ページの イメージ。

__writeFile

構文	<code>__writeFile(fileHandle, value)</code>				
パラメータ	<table><tr><td><code>fileHandle</code></td><td>__openFile マクロでファイルハンドルとして使用するマクロ変数</td></tr><tr><td><code>value</code></td><td>整数</td></tr></table>	<code>fileHandle</code>	__openFile マクロでファイルハンドルとして使用するマクロ変数	<code>value</code>	整数
<code>fileHandle</code>	__openFile マクロでファイルハンドルとして使用するマクロ変数				
<code>value</code>	整数				
リターン値	<code>int 0</code>				
説明	整数値を 16 進数フォーマット（後の空白文字を含む）でファイル <code>file</code> に出 力します。 注： __fmessage 文でも同様の操作を実行できます。__writeFile マクロは、 __readFile との対称性の観点から提供されています。				

__writeFileByte

構文	<code>__writeFileByte(fileHandle, value)</code>				
パラメータ	<table><tr><td><code>fileHandle</code></td><td>__openFile マクロでファイルハンドルとして使用するマクロ変数</td></tr><tr><td><code>value</code></td><td>0 ～ 255 の範囲の整数</td></tr></table>	<code>fileHandle</code>	__openFile マクロでファイルハンドルとして使用するマクロ変数	<code>value</code>	0 ～ 255 の範囲の整数
<code>fileHandle</code>	__openFile マクロでファイルハンドルとして使用するマクロ変数				
<code>value</code>	0 ～ 255 の範囲の整数				
リターン値	<code>int 0</code>				
説明	<code>fileHandle</code> ファイルに 1 バイトライトします。				

__writeMemory8, __writeMemoryByte

構文	<pre>__writeMemory8(value, address, zone) __writeMemoryByte(value, address, zone)</pre>	
パラメータ	<div> <div>value</div> <div>書き込む値 (整数)</div> </div> <div> <div>address</div> <div>メモリアドレス (整数)</div> </div> <div> <div>ゾーン</div> <div>メモリゾーン名 (文字列)。使用可能なゾーンの詳細は、 156 ページの <i>C-SPY</i> メモリゾーンを参照してください</div> </div>	
リターン値	int 0	
説明	指定したメモリアドレス (ロケーション) に 1 バイトライトします。	
例	<pre>__writeMemory8(0x2F, 0x8020, "Memory");</pre>	

__writeMemory16

構文	<pre>__writeMemory16(value, address, zone)</pre>	
パラメータ	<div> <div>value</div> <div>書き込む値 (整数)</div> </div> <div> <div>address</div> <div>メモリアドレス (整数)</div> </div> <div> <div>ゾーン</div> <div>メモリゾーン名 (文字列)。使用可能なゾーンの詳細は、 156 ページの <i>C-SPY</i> メモリゾーンを参照してください</div> </div>	
リターン値	int 0	
説明	指定したメモリアドレス (ロケーション) に 2 バイトライトします。	
例	<pre>__writeMemory16(0x2FFF, 0x8020, "Memory");</pre>	

__writeMemory32

構文	<code>__writeMemory32(value, address, zone)</code>	
パラメータ	<i>value</i>	書き込む値 (整数)
	<i>address</i>	メモリアドレス (整数)
	<i>ゾーン</i>	メモリゾーン名 (文字列)。使用可能なゾーンの詳細は、 <i>156 ページの C-SPY</i> メモリゾーンを参照してください。
リターン値	<code>int 0</code>	
説明	指定したメモリアドレス (ロケーション) に 4 バイトライトします。	
例	<code>__writeMemory32(0x5555FFFF, 0x8020, "Memory");</code>	

C-SPY コマンドラインユーティリティ — cspybat

この章では、C-SPY コマンドラインユーティリティ (cspybat.exe) を使用してバッチモードで C-SPY® を使用方法について説明します。具体的には、下記の項目をカバーします。

- C-SPY をバッチモードで使用
- C-SPY コマンドラインオプションの概要
- C-SPY コマンドラインオプションについてのリファレンス情報

C-SPY をバッチモードで使用

コマンドラインユーティリティの cspybat を使用すると、C-SPY をバッチモードで実行できます。このユーティリティは、common¥bin ディレクトリにインストールされています。

呼出し構文

cspybat の呼出し構文は以下のとおりです。

```
cspybat processor_DLL driver_DLL debug_file [cspybat_options]
        --backend driver_options
```

注：ファイル名 (DLL ファイルも含む) が要求される場合には、ファイル名のフルパスを指定することが推奨されます。

パラメータ

パラメータを以下に示します。

パラメータ	説明
processor_DLL	プロセッサ固有の DLL ファイルです。arm¥bin にあります。
driver_DLL	C-SPY ドライバ DLL ファイルです。arm¥bin にあります。
debug_file	デバッグ対象のオブジェクトファイルです (ファイル名拡張子 out)。

表 40: cspybat のパラメータ

パラメータ	説明
<code>cspybat_options</code>	<code>cspybat</code> に渡すコマンドラインオプションです。これらのオプションは省略可能です。それぞれのオプションについては、338 ページの <i>C-SPY コマンドラインオプションについてのリファレンス情報</i> を参照してください。
<code>--backend</code>	C-SPY ドライバに送信するパラメータの開始を示します。後に続くすべてのオプションがドライバに渡されます。このオプションは必須です。
<code>driver_options</code>	C-SPY ドライバに渡すコマンドラインオプションです。これらのオプションには、必須のものと省略可能なものがあります。それぞれのオプションについては、338 ページの <i>C-SPY コマンドラインオプションについてのリファレンス情報</i> を参照してください。

表 40: `cspybat` のパラメータ (続き)

例

以下の例では、シミュレータドライバを使用して `cspybat` を起動します。

```
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
```

ここで、`EW_DIR` は IAR Embedded Workbench のインストール先ディレクトリのフルパス、

`PROJ_DIR` はプロジェクトディレクトリのパスです。

出力

`cspybat` の実行時、以下のタイプの出力を生成できます。

- `cspybat` 自身からのターミナル出力
このターミナル出力はすべて `stderr` に転送されます。コマンドラインから引数なしで `cspybat` を実行する場合、`cspybat` のバージョン番号と利用可能なすべてのオプション（簡単な説明を含む）が `stdout` に転送され、画面に表示されます。
- デバッグ対象アプリケーションからのターミナル出力
こうしたすべてのターミナル出力先は、`stdout` になります。ただし、`--plugin` オプションを使用していることが条件です。357 ページの `--plugin` を参照してください。

- エラーリターンコード

cspybat は、バッチファイル内で評価可能なステータス情報をホストオペレーティングシステムに返します。*成功*の場合は値 `int 0` が返され、*失敗*の場合は値 `int 1` が返されます。

自動生成されたバッチファイルの使用

IDE で C-SPY を使用する場合、C-SPY が初期化されるたびに `projectname.cspy.bat` というバッチファイルが C-SPY から生成されます。このファイルは、`$PROJ_DIR$¥settings` ディレクトリに保存されています。このバッチファイルには、IDE での設定と同じ設定が記述されており、最小限の変更を加えることにより、このファイルをコマンドラインから使用して cspybat を起動することができます。このファイルには、必要な変更に関する情報も記述されています。

C-SPY コマンドラインオプションの概要

CSPYBAT の一般オプション

<code>--backend</code>	C-SPY ドライバに送信するパラメータの開始を示します（必須）。
<code>--code_coverage_file</code>	コードカバレッジ情報の生成を可能にして、それを指定ファイルに記録します。
<code>--cycles</code>	実行するサイクルの最大回数を指定します。
<code>--download_only</code>	コードイメージを後でデバッグセッションを開始せずにダウンロードします。
<code>--flash_loader</code>	フラッシュローダ仕様 XML ファイルを指定します。
<code>--macro</code>	使用するマクロファイルを指定します。
<code>--plugin</code>	使用するマクロファイルを指定します。
<code>--silent</code>	サインオンメッセージを省略します。
<code>--timeout</code>	最長実行時間を設定します。

すべての C-SPY ドライバで使用可能なオプション

--BE8	ビッグエンディアンフォーマット BE8 を使用します。リファレンス情報については、『 <i>ARM 用 IAR C/C++ 開発ガイド</i> 』を参照してください。
--BE32	ビッグエンディアンフォーマット BE32 を使用します。リファレンス情報については、『 <i>ARM 用 IAR C/C++ 開発ガイド</i> 』を参照してください。
--cpu	派生プロセッサを指定します。リファレンス情報については、『 <i>ARM 用 IAR C/C++ 開発ガイド</i> 』を参照してください。
--device	デバイスの名前を指定します。
--drv_attach_to_program	実行中のアプリケーションの現在の位置にデバッガを接続します。リファレンス情報については、367 ページの <i>プログラムにアタッチする</i> を参照してください。
--drv_catch_exceptions	特定の例外の場合にアプリケーションを停止します。
--drv_communication	使用する通信リンクを指定します。
--drv_communication_log	ログファイルを作成します。
--drv_default_breakpoint	ブレークポイントの設定時に使用するブレークポイントリソースの種類を設定します。
--drv_reset_to_cpu_start	デバッガの起動時やリセット時の PC の設定を省略します。
--drv_restore_breakpoints	システム起動中に破壊されたブレークポイントを自動的に復元します。
--drv_suppress_download	実行可能イメージのダウンロードを抑制します。リファレンス情報については、367 ページの <i>ダウンロードしない</i> を参照してください。
--drv_swo_clock_setup	CPU クロックと必要な SWO 速度を指定します。

<code>--drv_vector_table_base</code>	Cortex-M リセットベクタの位置およびスタックポインタの初期値を指定します。
<code>--drv_verify_download</code>	ターゲットプログラムを検証します。リファレンス情報については、367 ページのベリファイするを参照してください。
	Angel、GDB Server、IAR ROM-monitor、J-Link/J-Trace、TI Stellaris FTDI、Macraigor、RDI、ST-LINK で使用できます。
<code>--endian</code>	生成されるコードおよびデータのバイトオーダを指定します。リファレンス情報については、『 <i>ARM 用 IAR C/C++ 開発ガイド</i> 』を参照してください。
<code>--fpu</code>	浮動小数点ユニット型を選択します。リファレンス情報については、『 <i>ARM 用 IAR C/C++ 開発ガイド</i> 』を参照してください。
<code>-p</code>	使用するデバイス記述ファイルを指定します。
<code>--proc_stack_stack</code>	予約されているスタックに関する情報を C-SPY プラグインモジュールに提供します。
<code>--semihosting</code>	セミホスト I/O を有効にします。

シミュレータドライバで使用可能なオプション

<code>--disable_interrupts</code>	割込みシミュレーションを無効化します。
<code>--mapu</code>	メモリアクセスチェックをアクティブにします。

C-SPY ANGEL デバッグモニタドライバで使用可能なオプション

- rdi_heartbeat C-SPY でターゲットシステムを定期的にポーリングします。リファレンス情報については、371 ページのハートビート送信を参照してください。
- rdi_step_max_one 命令を 1 つ実行します。

C-SPY GDB サーバドライバで使用可能なオプション

- gdbserv_exec_command コマンド文字列を GDB サーバに送信します。

C-SPY IAR ROM-MONITOR ドライバで使用可能なオプション

C-SPY IAR ROM-monitor ドライバに固有のオプションは他にありません。

C-SPY J-LINK/J-TRACE ドライバで使用可能なオプション

- jlink_device_select JTAG スキャンチェーン内の特定デバイスを選択します。
- jlink_exec_command ターゲット接続が確立された後に __jlinkExecCommand マクロを呼び出します。
- jlink_initial_speed 最初の JTAG 通信速度 (kHz) を指定します。
- jlink_interface J-Link デバッグプローブとターゲットシステム間の通信を指定します。
- jlink_ir_length デバッグ対象の ARM デバイスの前の IR ビット数を設定します。
- jlink_reset_strategy デバッガの起動時に使用するリセット方法を選択します。
- jlink_script_file ハードウェアを設定するスクリプトファイルを指定します。
- jlink_speed JTAG 通信速度 (kHz) を指定します。

C-SPY TI STELLARIS FTDI ドライバで使用可能なオプション

- lmiftdi_speed JTAG 通信速度 (kHz) を指定します。

C-SPY MACRAIGOR ドライバで使用可能なオプション

<code>--mac_handler_address</code>	Intel XScale デバイスで使用するデバッグハンドラの位置を指定します。
<code>--mac_interface</code>	Macraigor デバッグプローブとターゲットシステム間の通信を指定します。
<code>--mac_jtag_device</code>	ハードウェアデバイスに対応するデバイスを選択します。
<code>--mac_multiple_targets</code>	JTAG スキャンチェーンに複数のデバイスがある場合、接続するデバイスを指定します。
<code>--mac_reset_pulls_reset</code>	C-SPY で最初のハードウェアリセットを生成します。
<code>--mac_set_temp_reg_buffer</code>	ドライバにコプロセッサのアクセス用の物理 RAM アドレスを提供します。
<code>--mac_speed</code>	JTAG プローブと ARM JTAG ICE ポートの間の JTAG 速度を設定します。
<code>--mac_xscale_ir7</code>	XScale ir7 アーキテクチャの使用を指定します。

C-SPY RDI ドライバで使用可能なオプション

<code>--rdi_allow_hardware_reset</code>	ハードウェアリセットを実行します。
<code>--rdi_driver_dll</code>	RDI ドライバ DLL ファイルのパスを指定します。
<code>--rdi_step_max_one</code>	命令を 1 つ実行します。

C-SPY ST-LINK ドライバで使用可能なオプション

<code>--stlink_interface</code>	ST-LINK デバッグプローブとターゲットシステム間の通信を指定します。
<code>--stlink_reset_strategy</code>	使用するリセット方法を指定します。

C-SPY サードパーティ製ドライバで使用可能なオプション

使用するサードパーティ製ドライバに固有なオプションについては、そのドライバのドキュメントを参照してください。

C-SPY コマンドラインオプションについてのリファレンス情報

ここでは、`cspybat` の各オプションおよび C-SPY ドライバで使用可能な各オプションに関する詳細なリファレンス情報を提供します。

--backend

構文	<code>--backend {driver options}</code>
パラメータ	<div> <div><code>driver options</code></div> <div>使用している C-SPY ドライバで使用可能なすべてのオプション。</div> </div>
適用範囲	<code>cspybat</code> への送信（必須）。
説明	このオプションは、各種オプションを C-SPY ドライバに送信するときに使用します。 <code>--backend</code> の後に続くすべてのオプションは、C-SPY ドライバに送信され、 <code>cspybat</code> 自身では処理されません。

--code_coverage_file

構文	<code>--code_coverage_file file</code>
パラメータ	<div> <div><code>file</code></div> <div>コードカバレッジ情報の対象ファイル名。</div> </div>
適用範囲	<code>cspybat</code> への送信。
説明	<p>このオプションを使用して、コードカバレッジ情報の生成を有効にします。コードカバレッジ情報は実行が完了した後に生成され、指定したファイルに保存されます。</p> <p>このオプションでは、使用する C-SPY ドライバがコードカバレッジをサポートしている必要があります。このオプションをコードカバレッジをサポートしていない C-SPY ドライバで使用すると、エラーメッセージが <code>stderr</code> に出力されます。</p>
関連項目	251 ページの コードカバレッジ。


--cycles

構文	<code>--cycles cycles</code>	
パラメータ	<code>cycles</code>	実行するサイクル数。
適用範囲	cspybat への送信。	
説明	このオプションは、実行するサイクルの最大回数を指定するときに使用します。ターゲットプログラムが指定サイクル数より長く実行された場合、ターゲットプログラムは異常終了されます。このオプションを使用するには、使用している C-SPY ドライバがサイクルカウンタをサポートし、実行中にサイクルカウンタのサンプリングが可能であることが必要です。	


--device

構文	<code>--device=device_name</code>	
パラメータ	<code>device_name</code>	ADuC7030、AT91SAM7S256、LPC2378、STR912FM44、TMS470R1B1M などのデバイス名。
適用範囲	すべての C-SPY ドライバ。	
説明	このオプションを使用して、デバイス名を指定します。 オプションを設定するには、以下のように選択します。 [プロジェクト] > [オプション] > [一般オプション] > [ターゲット] > [デバイス]	

--disable_interrupts

構文	<code>--disable_interrupts</code>	
適用範囲	C-SPY シミュレータドライバ。	
説明	このオプションは、命令シミュレーションを無効にするときに使用します。  このオプションを設定するには、[シミュレータ] > [割込み設定] を選択して、[割込みシミュレーションを有効にする] オプションの選択を解除します。	

--download_only

構文	<code>--download_only</code>
適用範囲	cspybat への送信。
説明	このオプションを使用して、後でデバッグセッションを開始せずにコードイメージをダウンロードします。
	 オプションを設定するには、以下のように選択します。 [プロジェクト] > [ダウンロード]

--drv_catch_exceptions

構文	<code>--drv_catch_exceptions=value</code>				
パラメータ	<table><tr><td><code>value</code> (ARM9 および Cortex-R4 の場合)</td><td>0-0x1FF の範囲の値です。各ビットは、以下の ように、キャッチする例外を指定します。 ビット 0 = リセット ビット 1 = 未定義の命令 ビット 2 = SWI ビット 3 = 未使用 ビット 4 = データポート ビット 5 = プリフェッチポート ビット 6 = IRQ ビット 7 = FIQ ビット 8 = その他のエラー</td></tr><tr><td><code>value</code> (Cortex-M の場合)</td><td>0-0x7FF の範囲の値です。各ビットは、以下のよ うに、キャッチする例外を指定します。 Bit 0 = CORERESSET - リセットベクタ Bit 4 = MMERR - メモリ管理の障害 Bit 5 = NOCPERR - コプロセッサのアクセスエラー Bit 6 = CHKERR - チェッキングエラー Bit 7 = STATERR - ステートエラー Bit 8 = BUSERR - バスエラー Bit 9 = INTERR - 割込みサービスエラー Bit 10 = HARDERR - ハード障害</td></tr></table>	<code>value</code> (ARM9 および Cortex-R4 の場合)	0-0x1FF の範囲の値です。各ビットは、以下の ように、キャッチする例外を指定します。 ビット 0 = リセット ビット 1 = 未定義の命令 ビット 2 = SWI ビット 3 = 未使用 ビット 4 = データポート ビット 5 = プリフェッチポート ビット 6 = IRQ ビット 7 = FIQ ビット 8 = その他のエラー	<code>value</code> (Cortex-M の場合)	0-0x7FF の範囲の値です。各ビットは、以下のよ うに、キャッチする例外を指定します。 Bit 0 = CORERESSET - リセットベクタ Bit 4 = MMERR - メモリ管理の障害 Bit 5 = NOCPERR - コプロセッサのアクセスエラー Bit 6 = CHKERR - チェッキングエラー Bit 7 = STATERR - ステートエラー Bit 8 = BUSERR - バスエラー Bit 9 = INTERR - 割込みサービスエラー Bit 10 = HARDERR - ハード障害
<code>value</code> (ARM9 および Cortex-R4 の場合)	0-0x1FF の範囲の値です。各ビットは、以下の ように、キャッチする例外を指定します。 ビット 0 = リセット ビット 1 = 未定義の命令 ビット 2 = SWI ビット 3 = 未使用 ビット 4 = データポート ビット 5 = プリフェッチポート ビット 6 = IRQ ビット 7 = FIQ ビット 8 = その他のエラー				
<code>value</code> (Cortex-M の場合)	0-0x7FF の範囲の値です。各ビットは、以下のよ うに、キャッチする例外を指定します。 Bit 0 = CORERESSET - リセットベクタ Bit 4 = MMERR - メモリ管理の障害 Bit 5 = NOCPERR - コプロセッサのアクセスエラー Bit 6 = CHKERR - チェッキングエラー Bit 7 = STATERR - ステートエラー Bit 8 = BUSERR - バスエラー Bit 9 = INTERR - 割込みサービスエラー Bit 10 = HARDERR - ハード障害				

適用範囲	C-SPY Angel デバッグモニタドライバ。 C-SPY J-Link/J-Trace ドライバ。 C-SPY RDI ドライバ。
説明	このオプションは、特定の例外発生時にアプリケーションを停止するときに使用します。
関連項目	125 ページの <i>例外ベクタのブレークポイント</i> 。
	 C-SPY Angel デバッグモニタドライバの場合 [プロジェクト] > [オプション] > [デバッグ] > [追加オプション] C-SPY J-Link/J-Trace ドライバの場合 [プロジェクト] > [オプション] > [デバッグ] > [J-Link/J-Trace] > [例外の取得] C-SPY RDI ドライバの場合 [プロジェクト] > [オプション] > [デバッグ] > [RDI] > [例外の取得]

--drv_communication

構文	--drv_communication= <i>connection</i>
パラメータ	<p>C-SPY Angel デバッグモニタドライバの場合、<i>connection</i> は以下のいずれかです。</p> <p>イーサネット経由 UDP:<i>ip_address</i> UDP:<i>ip_address,port</i> UDP:<i>hostname</i> UDP:<i>hostname,port</i></p> <p>シリアルポート経由 <i>port:baud,parity,stop_bit,handshake</i> <i>port</i> = COM1-COM256 (デフォルト COM1) <i>baud</i> = 9600、19200、38400、57600、または 115200 (デフォルト 9600 <i>baud</i>) <i>parity</i> = N (パリティなし) <i>stop_bit</i> = 1 (ストップビットは 1 ビット) <i>handshake</i> = NONE または RTSCTS (ハンドシェイクなしの場合はデフォルト NONE) 例: COM1:9600,N,8,1,NONE</p>

C-SPY GDB サーバドライバの場合、*connection* は以下のいずれかです。

イーサネット経由 *TCPIP:ip_address*
 TCPIP:ip_address,port
 TCPIP:hostname
 TCPIP:hostname,port

ポートを指定しない場合、デフォルトでポート 3333 が使用されます。

C-SPY IAR ROM-monitor ドライバの場合、*connection* は以下のいずれかです。

シリアルポート経由 *port:baud,parity,stop_bit,handshake*
 port = COM1-COM256 (デフォルト COM1)
 baud = 9600、19200、38400、57600、または 115200
 (デフォルト 9600 *baud*)
 parity = N (パリティなし)
 stop_bit = 1 (ストップビットは 1 ビット)
 handshake = NONE または RTSCTS (ハンドシェイクなしの場合はデフォルト NONE)
 例: COM1:9600,N,8,1,NONE

C-SPY J-Link/J-Trace ドライバの場合、*connection* は以下のいずれかです。

USB 経由で J-Link USB0-USB3
 に直接

USB0 を使用し、USB 接続上に複数の J-Link デバッグブローブがある場合は、デバッグセッションを開始するとダイアログボックスが表示されます。このダイアログボックスを使用して、接続する J-Link デバッグブローブを選択します。

USB:#*number* は、USB 接続上のシリアル番号が *number* の J-Link に接続します。

LAN 上の J-Link を
経由

TCPIP:
コロン符号の後にアドレスやホスト名、シリアル番号
が何もない場合は、J-Link ドライバはローカルネッ
トワーク上のすべての J-Link デバッグプローブを
検索して、それらをダイアログボックスに表示し
ます。ここで接続するプローブを選択できます
(自動検出)。
TCPIP:*ip_address*
TCPIP:*ip_address,port*
TCPIP:*hostname*
TCPIP:*hostname,port*
TCPIP:*#number* は、ローカルネットワーク上のシリア
ル番号が *number* の J-Link に接続します。
ポートを指定しない場合、デフォルトでポート 19020
が使用されます。

C-SPY Macraigor ドライバの場合、*connection* は以下のいずれかです。

mpDemon の場合 *port:baud*
 port = COM1-COM4
 baud = 9600、19200、38400、57600、または 115200
 (デフォルト 9600 baud)

mpDemon の場合 TCPIP:*ip_address*
 TCPIP:*ip_address,port*
 TCPIP:*hostname*
 TCPIP:*hostname,port*
 ポートを指定しない場合、デフォルトでポート 19020
 が使用されます。

USB 経由で usbDemon USB ports = USB0-USB3
および usb2Demon

適用範囲

C-SPY Angel デバッグモニタドライバ。
C-SPY GDB サーバドライバ。
C-SPY IAR ROM-monitor ドライバ。
C-SPY J-Link/J-Trace ドライバ。
C-SPY Macraigor ドライバ。

説明

このオプションは、通信リンクを選択するときに使用します。


[プロジェクト] > [オプション] > [デバッガ] >[Angel]> [通信]
[プロジェクト] > [オプション] > [デバッガ] > [GDB サーバ] > [TCP/IP
アドレスまたはホスト名 [,port]]
[プロジェクト] > [オプション] > [デバッガ] > [IAR ROM モニタ] >
[通信]
[プロジェクト] > [オプション] > [デバッガ] >[J-Link/J-Trace]> [接続] >
[通信]
C-SPY Macraigor ドライバの関連オプションを設定するには、以下のよう
に選択します。
[プロジェクト] > [オプション] > [デバッガ] >[Macraigor]

--drv_communication_log


構文	--drv_communication_log= <i>filename</i>		
パラメータ	<table><tr><td><i>filename</i></td><td>ログファイルの名前。</td></tr></table>	<i>filename</i>	ログファイルの名前。
<i>filename</i>	ログファイルの名前。		
適用範囲	すべての C-SPY ドライバ。		
説明	このオプションを選択すると、C-SPY とターゲットシステムとの間の通信が ファイルにロギングされます。結果を分析するには、通信プロトコルに 関する詳しい知識が必要です。 [プロジェクト] > [オプション] > [デバッガ] > [ドライバ] > [通信ログ]		

--drv_default_breakpoint

構文	--drv_default_breakpoint={0 1 2}						
パラメータ	<table><tr><td>0</td><td>自動 (デフォルト)</td></tr><tr><td>1</td><td>ハードウェア</td></tr><tr><td>2</td><td>ソフトウェア</td></tr></table>	0	自動 (デフォルト)	1	ハードウェア	2	ソフトウェア
0	自動 (デフォルト)						
1	ハードウェア						
2	ソフトウェア						
適用範囲	C-SPY GDB サーバドライバ。 C-SPY J-Link/J-Trace ドライバ。						


	C-SPY Macraigor ドライバ。
説明	このオプションでは、ブレークポイントの設定時に使用するブレークポイントリソースの種類を選択します。
関連項目	147 ページの デフォルトのブレークポイントタイプ。
	 [プロジェクト] > [オプション] > [デバッグ] > [ドライバ] > [ブレークポイント] > [デフォルトのブレークポイントタイプ]

--drv_reset_to_cpu_start


構文	<code>--drv_reset_to_cpu_start</code>
適用範囲	C-SPY Angel デバッグモニタドライバ C-SPY GDB サーバドライバ C-SPY J-Link/J-Trace ドライバ C-SPY TI Stellaris FTDI ドライバ C-SPY Macraigor ドライバ C-SPY RDI ドライバ C-SPY ST-LINK ドライバ
説明	このオプションは、デバッグの起動時やリセット時に PC の設定を省略するとき 사용합니다。その代わり、PC では、CPU による元の値のセット（アプリケーションのエントリポイントのアドレス）が使用されます。  このオプションを設定するには、 [プロジェクト] > [オプション] > [デバッグ] > [追加オプション] を選択します。

--drv_restore_breakpoints

構文	<code>--drv_restore_breakpoints=location</code>
パラメータ	<code>location</code> アドレスまたは関数名ラベル。
適用範囲	C-SPY GDB サーバドライバ C-SPY J-Link/J-Trace ドライバ

	C-SPY Macraigor ドライバ
説明	このオプションを使用すると、システム起動中にオーバーライドされたすべてのソフトウェアブレークポイントを自動的に復元します。
関連項目	148 ページの ソフトウェアブレークポイント復元位置。
	 [プロジェクト] > [オプション] > [デバッグ] > [ドライバ] > [ブレークポイント] > [ソフトウェアブレークポイント復元位置]

--drv_swo_clock_setup

構文	<code>--drv_swo_clock_setup=frequency,autodetect,wanted</code>	
パラメータ	<i>frequency</i>	内部プロセッサクロック HCLK の正確なクロック周波数 (Hz)。この値は、SWO の通信速度の設定およびタイムスタンプの計算に使用します。
	<i>autodetect</i>	0: パラメータ <i>wanted</i> を使用して、希望する周波数を指定します。 1: J-Link デバッグプローブで使用できる最大可能周波数を自動的に使用します。
	<i>wanted</i>	<i>autodetect</i> が 0 の場合に使用する周波数 (Hz)。 <i>wanted</i> は、送信中にデータパケットが失われる場合に使用します。
適用範囲	J-Link ドライバおよび ST-LINK ドライバ。	
説明	このオプションを使用して、CPU クロックを設定します。このオプションを使用しない場合、CPU クロック周波数はデフォルトで 72 MHz に設定されます。	
		[J-Link]> [SWO 設定] > [CPU クロック] [J-Link]> [SWO 設定] > [SWO クロック] > [自動検出] [J-Link]> [SWO 設定] > [SWO クロック] > [希望する値]

--drv_vector_table_base

構文	<code>--drv_vector_table_base=expression</code>
----	---

パラメータ

expression ラベルまたはアドレス

適用範囲

C-SPY GDB サーバドライバ
 C-SPY J-Link/J-Trace ドライバ
 C-SPY TI Stellaris FTDI ドライバ
 C-SPY Macraigor ドライバ
 C-SPY RDI ドライバ
 C-SPY ST-LINK ドライバ
 C-SPY シミュレータドライバ

説明

このオプションは、Cortex-M においてリセットベクタの位置およびスタックポインタの初期値を指定するときに使用します。このオプションは、アプリケーション内でデフォルトの `__vector_table` ラベル（システム起動コードで定義）をオーバーライドする場合や、アプリケーションにこのラベルが欠落している場合に便利です。後者は、別のベンダ製ツールによってビルドされたコードをデバッグする場合に起こり得ます。



このオプションを設定するには、[プロジェクト] > [オプション] > [デバッグ] > [追加オプション] を選択します。

--flash_loader

構文

`--flash_loader filename`

パラメータ

filename ファイル名の拡張子が board のフラッシュローダの仕様 XML ファイル。

適用範囲

cspybat への送信。

説明

このオプションは、フラッシュのロードに関するすべての関連情報を記述したフラッシュローダ仕様 xml ファイルを指定するときに使用します。この引数は複数指定できます。その場合、それぞれの引数が指定の順序で処理され、複数のフラッシュプログラミングパスが得られます。

関連項目

IAR Embedded Workbench flash loader User Guide.

--gdbserv_exec_command

構文	<code>--gdbserv_exec_command="string"</code>	
パラメータ	<code>"string"</code>	GDB サーバに送信する文字列またはコマンドです。 詳細についてはドキュメントを参照してください。
適用範囲	C-SPY GDB サーバドライバ。	
説明	このオプションは、文字列やコマンドを GDB サーバに送信するときに使用します。	



[プロジェクト] > [オプション] > [デバッグ] > [追加オプション]

--jlink_device_select

構文	<code>--jlink_device_select=tap_number</code>	
パラメータ	<code>tap_number</code>	接続先のデバイスの TAP 位置。
適用範囲	C-SPY J-Link/J-Trace ドライバ。	
説明	JTAG スキャンチェーンに複数のデバイスがある場合に、このオプションを使用してデバイスを選択します。	
関連項目	380 ページの JTAG スキャンチェーン。	



[プロジェクト] > [オプション] > [デバッグ] > [J-Link/J-Trace] > [接続] > [JTAG スキャンチェーン] > [TAP 番号]

--jlink_exec_command

構文	<code>--jlink_exec_command=cmdstr1; cmdstr2; cmdstr3 ...</code>	
パラメータ	<code>cmdstrn</code>	J-Link/J-Trace コマンド文字列。
適用範囲	C-SPY J-Link/J-Trace ドライバ	

説明 このオプションは、ターゲットの接続が完了した後に、デバッガで 1 つまたは複数のコマンド文字列を指定して `__jlinkExecCommand` マクロを呼び出すときに使用します。

関連項目 303 ページの `__jlinkExecCommand`。



[プロジェクト] > [オプション] > [デバッガ] > [追加オプション]

--jlink_initial_speed

構文 `--jlink_initial_speed=speed`

パラメータ

speed 最初の通信速度 (kHz)。速度を指定しない場合、最初の速度として 32kHz が使用されます。

適用範囲 C-SPY J-Link/J-Trace ドライバ。

説明 このオプションは、最初の JTAG 通信速度 (kHz) を指定するときに使用します。

関連項目 377 ページの *JTAG/SWD 速度*。



[プロジェクト] > [オプション] > [デバッガ] > [J-Link/J-Trace] > [設定] > [JTAG の速度] > [固定]

--jlink_interface

構文 `--jlink_interface={JTAG|SWD}`


パラメータ

JTAG ターゲットシステムとの通信に JTAG を使用します (デフォルト)。

SWD ターゲットシステムとの通信に SWD 接続を使用します (Cortex-M のみ)。JTAG 通信より少ないピンを使用します。

適用範囲 C-SPY J-Link/J-Trace ドライバ。

説明 このオプションは、J-Link デバッグプロブとターゲットシステム間の通信チャンネルを指定するときに使用します。

関連項目	380 ページの <i>インタフェース</i> 。
	[プロジェクト] > [オプション] > [デバッグ] > [J-Link/J-Trace] > [接続] > [インタフェース]

--jlink_ir_length


構文	<code>--jlink_ir_length=length</code>	
パラメータ	<i>length</i>	デバッグ対象 ARM デバイスの前の IR ビット数です。 ARM デバイスと他のデバイスが混在する JTAG スキャンチェーン用。
適用範囲	C-SPY J-Link/J-Trace ドライバ。	
説明	このオプションは、デバッグ対象の ARM デバイスの前の IR ビット数を設定します。	
関連項目	380 ページの <i>JTAG スキャンチェーン</i> 。	



[プロジェクト] > [オプション] > [デバッグ] > [J-Link/J-Trace] > [接続] > [JTAG スキャンチェーン] > [先行ビット]

--jlink_reset_strategy

構文	<code>--jlink_reset_strategy=delay, strategy</code>	
パラメータ	<i>delay</i>	Cortex-M および ARM 7/9/11 で方式が 1-9 の場合、 <i>delay</i> は 0（無視）に設定してください。ARM 7/9/11 で方式が 0 の場合、 <i>delay</i> の値は 0-10000 のいずれかに する必要があります。
	<i>strategy</i>	サポートされているリセット方式について詳しくは、 『ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド』を参照してください。
適用範囲	C-SPY J-Link/J-Trace ドライバ。	


説明	このオプションは、デバッガの起動時に使用するリセット方法を選択するとき に使用します。
関連項目	374 ページの <i>リセット</i> 。
	 [プロジェクト] > [オプション] > [デバッガ] > [J-Link/J-Trace] > [設定] > [リセット]

--jlink_script_file


構文	--jlink_script_file= <i>filename</i>
パラメータ	<div><div><i>filename</i></div><div>J-Link スクリプトファイル名。</div></div>
適用範囲	C-SPY J-Link/J-Trace ドライバ。
説明	<p>このオプションは、使用する J-Link スクリプトファイルを指定するときに使 用します。</p> <p>J-Link には、ハードウェアの設定に使用できるスクリプト言語が用意されて います。特定のターゲットについては、既成のスクリプトファイルが IAR Embedded Workbench によって自動的に指定されます。コマンドラインモード では、このオプションを使用してスクリプトファイルを手動で指定する必要 があります。</p> <p>既成以外のスクリプトファイルを使用する場合、[プロジェクト] > [オプ ション] > [デバッガ] > [追加オプション] ページで、このオプションを C-SPY に引き渡すことができます。</p>
関連項目	スクリプト言語について詳しくは、『 <i>J-Link/J-Trace ARM ユーザガイド</i> 』 (JLinkARM.pdf、ドキュメント番号 UM08001) のセクション 5.10。

--jlink_speed


構文	--jlink_speed={ <i>fixed</i> auto adaptive}
パラメータ	<div><div><div><i>fixed</i></div><div>1-12000</div></div><div><div>自動</div><div>信頼性の高い動作をするための最大可能周波数（デ フォルト）。</div></div><div><div>adaptive</div><div>RTCK JTAG 信号が使用可能な ARM デバイス用。</div></div></div>

適用範囲	C-SPY J-Link/J-Trace ドライバ。
説明	このオプションは、JTAG 通信速度 (kHz) を指定するときに使用します。
関連項目	377 ページの <i>JTAG/SWD 速度</i> 。
	 [プロジェクト] > [オプション] > [デバッガ] >[J-Link/J-Trace]> [設定] > [JTAG 速度]


--lmiftdi_speed

構文	<code>--lmiftdi_speed=frequency</code>
パラメータ	<i>frequency</i> 周波数 (kHz)。
適用範囲	C-SPY TI Stellaris FTDI ドライバ。
説明	このオプションは、JTAG 通信速度 (kHz) を指定するときに使用します。
関連項目	381 ページの <i>JTAG/SWD 速度</i> 。
	 [プロジェクト] > [オプション] > [デバッガ] >[TI Stellaris FTDI]> [設定] > [JTAG 速度]


--mac_handler_address

構文	<code>--mac_handler_address=address</code>
パラメータ	<i>address</i> デバッガハンドラのメモリエリアの開始アドレス。
適用範囲	C-SPY Macraigor ドライバ。
説明	このオプションでは、Intel XScale デバイスが使用するデバッグハンドラの場所（メモリアドレス）を指定します。
関連項目	384 ページの <i>デバッグハンドラアドレス</i> 。
	 [プロジェクト] > [オプション] > [デバッガ] >[Macraigor]> [デバッグハンドラアドレス]

--mac_interface


構文	<code>--mac_interface={JTAG SWO}</code>	
パラメータ	JTAG	ターゲットシステムとの通信に JTAG を使用します (デフォルト)。
	SWD	ターゲットシステムとの通信に SWD 接続を使用します (Cortex-M のみ)。JTAG 通信より少ないピンを使用します。
適用範囲	C-SPY Macraigor ドライバ。	
説明	このオプションは、Macraigor デバッグプローブとターゲットシステム間の通信チャンネルを指定するときに使用します。	
		[プロジェクト] > [オプション] > [デバッグ] > [Macraigor] > [インタフェース]

--mac_jtag_device


構文	<code>--mac_jtag_device=device</code>	
パラメータ	device	使用するハードウェアインタフェースに対応するデバイス。Macraigor mpDemon、usbdemon、usb2demon から選択します。
適用範囲	C-SPY Macraigor ドライバ。	
説明	このオプションは、使用するハードウェアインタフェースに対応するデバイスを選択するときに使用します。	
関連項目	382 ページの <i>OCD</i> インタフェースデバイス。	
		[プロジェクト] > [オプション] > [デバッグ] > [Macraigor] > [OCD インタフェースデバイス]

--mac_multiple_targets

構文	<code>--mac_multiple_targets=<tap-no>@dev0,dev1,dev2,dev3,...</code>
----	--

パラメータ	<code>tap-no</code>	接続先のデバイスの TAP 番号です。最初のデバイスに接続する場合は 0、2 番目のデバイスに接続する場合は 1 などとします。
	<code>dev0-devn</code>	Macraigor JTAG プローブ上で最も近い TDO ピン。
適用範囲	C-SPY Macraigor ドライバ。	
説明	JTAG スキャンチェーンに複数のデバイスがある場合、それぞれのデバイスを定義する必要があります。このオプションは、接続先のデバイスを指定するときに使用します。	
例	<code>--mac_multiple_targets=0@ARM7TDMI,ARM7TDMI</code>	
関連項目	384 ページの <i>JTAG スキャンチェーン (マルチターゲット)</i> 。	
		[プロジェクト] > [オプション] > [デバッグ] > [Macraigor] > [JTAG スキャンチェーン (マルチターゲット)]


--mac_reset_pulls_reset

構文	<code>--mac_reset_pulls_reset=time</code>	
パラメータ	<code>time</code>	0-2000。リセット後の遅延（ミリ秒）です。
適用範囲	C-SPY Macraigor ドライバ。	
説明	このオプションは、デバッグの起動時に最初のハードウェアリセットを C-SPY で実行するときに使用します。また、リセット後の遅延を指定します。	
関連項目	383 ページの <i>ハードウェアリセット</i> 。	
		[プロジェクト] > [オプション] > [デバッグ] > [Macraigor] > [ハードウェアリセット]

--macro

構文	<code>--macro filename</code>
パラメータ	<i>filename</i> 使用する C-SPY マクロファイル（ファイル名拡張子 <code>mac</code> ）。
適用範囲	<code>cspybat</code> への送信。
説明	このオプションは、ターゲットアプリケーションを実行する前にロードする C-SPY マクロファイルを指定するときに使用します。このオプションは、1 つのコマンドラインで複数個使用できます。
関連項目	278 ページの <i>C-SPY マクロの使用の概要</i> 。

--mac_set_temp_reg_buffer

構文	<code>--mac_set_temp_reg_buffer=address</code>
パラメータ	<i>address</i> RAM エリアの開始アドレス。
適用範囲	C-SPY Macraigor ドライバ。
説明	このオプションは、MMU の制御および CP15 コプロセッサ経由のキャッシュに使用する RAM エリアの開始アドレスを指定するときに使用します。
	 このオプションを設定するには、[プロジェクト] > [オプション] > [デバッグ] > [追加オプション] を選択します。

--mac_speed

構文	<code>--mac_speed={factor}</code>
パラメータ	<i>factor</i> スキャンクロックの生成時に JTAG プロブクロックを分周する係数です。この値は 1-8 の範囲でなければなりません。1 が最速となります。
適用範囲	C-SPY Macraigor ドライバ。

説明 このオプションは、JTAG プローブと ARM JTAG ICE ポートの間の JTAG 速度を設定するときに使用します。

関連項目 383 ページの *JTAG 速度*。



[プロジェクト] > [オプション] > [デバッグ] > [Macraigor] > [JTAG 速度]

--mac_xscale_ir7

構文 --mac_xscale_ir7

適用範囲 C-SPY Macraigor ドライバ。

説明 このオプションは、XScale ir5 コアに代えて XScale ir7 コアを使用することを指定するときに使用します。XScale ir7 コアを使用する場合、このオプションは必須です。

以下の XScale コアが IAR C-SPY Macraigor ドライバでサポートされます。

Intel XScale Core 1 (5 ビット命令レジスタ — ir5)

Intel XScale Core 2 (7 ビット命令レジスタ — ir7)



このオプションを設定するには、[プロジェクト] > [オプション] > [デバッグ] > [追加オプション] を選択します。

--mapu

構文 --mapu

適用範囲 C-SPY シミュレータドライバ。

説明 このオプションは、デバッグファイル内の section 情報をメモリアクセスチェックに使用する場合に指定します。すると、実行中に未指定のメモリエリアへのアクセスがシミュレータでチェックされます。こうしたアクセスが見つかった場合、C 関数の呼出しスタックとメッセージが、stderr に出力されて、実行が停止します。

関連項目 158 ページの *メモリアクセスチェック*。



オプションを設定するには、以下のように選択します。

[シミュレータ] > [メモリアクセス設定]

-p

構文	<code>-p filename</code>
パラメータ	<i>filename</i> 使用するデバイス記述ファイル。
適用範囲	すべての C-SPY ドライバ。
説明	このオプションは、使用するデバイス記述ファイルを指定するときに使用します。
関連項目	59 ページの <i>デバイス記述ファイルの選択</i> 。

--plugin

構文	<code>--plugin filename</code>
パラメータ	<i>filename</i> 使用するプラグインファイル（ファイル名拡張子 <code>dll</code> ）。
適用範囲	cspybat への送信。
説明	<p>特定の C/C++ 標準ライブラリ関数（<code>printf</code> など）を、実際のハードウェアデバイスの代わりに、C-SPY（[C-SPY ターミナル I/O] ウィンドウなど）でサポートできます。このようなサポートを cspybat で有効にするには、<code>arm¥bin</code> ディレクトリにある <code>armbat.dll</code> という専用のプラグインモジュールを使用する必要があります。</p> <p>このオプションは、このプラグインをデバッグセッション中にインクルードするときに使用します。このオプションは、1 つのコマンドラインで複数個使用できます。</p> <p>注：このオプションは、別のプラグインモジュールのインクルードにも使用できますが、その場合の条件として、特にモジュールが cspybat で動作可能であることが必要となります。これは、<code>common¥plugin</code> ディレクトリにある C-SPY プラグインモジュールは、通常、cspybat で使用できないことを意味します。</p>

--proc_stack_stack

構文	<code>--proc_stack_stack=startaddress,endaddress</code>
	Cortex-M の場合、 <i>stack</i> は、 <i>main</i> または <i>proc</i> のいずれかです。

他の ARM コアの場合、*stack* は、*usr*、*svc*、*irq*、*fiq*、*und*、*abt* のいずれかです。

パラメータ

startaddress スタックの開始アドレス。値または式として指定します。
endaddress スタックの終了アドレス。値または式として指定します。

適用範囲

すべての C-SPY ドライバ このコマンドラインオプションは、C-SPY を IDE から使用する場合にのみ有効です。*cspybat* を使用したバッチモードでは使用できません。

説明

このオプションは、予約されているスタックに関する情報を C-SPY スタックプラグインモジュールに提供するときに使用します。デフォルトでは、C-SPY はこの情報をシステム起動コードから受け取りますが、何らかの理由でデフォルト値をオーバーライドする場合に、このオプションを活用できます。

例

`--proc_stack_irq=0x8000,0x80FF`



このオプションを設定するには、[プロジェクト] > [オプション] > [デバッグ] > [追加オプション] を選択します。

--rdi_allow_hardware_reset

構文

`--rdi_allow_hardware_reset`

適用範囲

C-SPY RDI ドライバ。

説明

このオプションは、ターゲットのハードウェアリセットをエミュレータで実行可能にするときに使用します。エミュレータでサポートされていることが必要です。

関連項目

385 ページの *ハードウェアリセットを許可*。



[プロジェクト] > [オプション] > [デバッグ] > [RDI] > [ハードウェアリセットを許可する]

--rdi_driver_dll

構文

`--rdi_driver_dll filename`

パラメータ

`filename` RDI ドライバ DLL ファイルのパス。

適用範囲

C-SPY RDI ドライバ。

説明

このオプションは、JTAG ポッドに付属の RDI ドライバ DLL ファイルのファイルパスを指定するときに使用します。

関連項目

385 ページの *Manufacturer RDI driver*。

[プロジェクト] > [オプション] > [デバッグ] > [RDI] > [メーカー RDI ドライバ]

--rdi_step_max_one

構文

`--rdi_step_max_one`

適用範囲

C-SPY Angel デバッグモニタドライバ。

C-SPY RDI ドライバ。

説明

このオプションは、命令を 1 つだけ実行するときに使用します。デバッグは、ステップ動作中、割込みをオフにします。また、必要に応じて、命令を実行せずに命令のシミュレーションを行います。



このオプションを設定するには、[プロジェクト] > [オプション] > [デバッグ] > [追加オプション] を選択します。

--semihosting

構文

`--semihosting={none|iar_breakpoint}`

パラメータ

パラメータはありません 標準セミホストを使用してください。

`none` セミホスト I/O は使用されません。

`iar_breakpoint` IAR 独自の派生セミホストが使用されます。

適用範囲

すべての C-SPY ドライバ。

説明	<p>このオプションは、セミホスト I/O を有効にし、使用するセミホストインタフェースの種類を選択するときに使用します。このオプションを使用しない場合、デフォルトでセミホストが有効化され、C-SPY は正しいセミホストモードを自動的に選択するように動作します。すなわち、アプリケーションがセミホストとリンクされている場合には、通常、このオプションを使用する必要はありません。</p> <p>セミホストが動作するためには、アプリケーションがセミホストライブラリとリンクされていることが必要です。</p>
関連項目	セミホストとのリンクに関する詳細については、『 <i>ARM 用 IAR C/C++ 開発ガイド</i> 』を参照してください。



[プロジェクト] > [オプション] > [一般オプション] > [ライブラリ構成]

--silent

構文	--silent
適用範囲	cspybat への送信。
説明	このオプションは、サインオンメッセージを省略するときに使用します。


--stlink_interface

構文	--stlink_interface={JTAG SWD}				
パラメータ	<table><tr><td>JTAG</td><td>ターゲットシステムとの通信に JTAG を使用します (デフォルト)。</td></tr><tr><td>SWD</td><td>ターゲットシステムとの通信に SWD を使用します。</td></tr></table>	JTAG	ターゲットシステムとの通信に JTAG を使用します (デフォルト)。	SWD	ターゲットシステムとの通信に SWD を使用します。
JTAG	ターゲットシステムとの通信に JTAG を使用します (デフォルト)。				
SWD	ターゲットシステムとの通信に SWD を使用します。				
適用範囲	C-SPY ST-LINK ドライバ。				
説明	このオプションは、 ST-LINK デバッグプローブとターゲットシステム間の通信チャンネルを指定するときに使用します。				
関連項目	387 ページの <i>インタフェース</i> 。				




[プロジェクト] > [オプション] > [デバッグ] >[ST-LINK]>[ST-LINK]>[インタフェース]


--stlink_reset_strategy

構文	<code>--stlink_reset_strategy=delay, strategy</code>	
パラメータ	<code>delay</code>	遅延時間（ミリ秒）。 <code>delay</code> は無視されるため、0にしてください。
	<code>strategy</code>	リセット方式。 0: (通常) 標準のリセットの手順を実行します。 1: (リセットピン) リセットピンを使用してハードウェアリセットを実行します。ST-LINK バージョン 2 でのみ使用できます。 2: (リセット中に接続) ST-LINK は、リセットをアクティブにしたままでターゲットに接続します（リセットは「低」になり、ターゲットに接続中はそのままになります）。ST-LINK バージョン 2 でのみ使用できます。
適用範囲	C-SPY ST-LINK ドライバ	
説明	このオプションは、デバッガの起動時に使用するリセット方法を選択するときに使用します。	
関連項目	374 ページの <i>リセット</i> 。	
		[プロジェクト] > [オプション] > [デバッガ] > [ST-LINK] > [設定] > [リセット]

--timeout

構文	<code>--timeout milliseconds</code>	
パラメータ	<code>milliseconds</code>	実行が停止するまでの時間（ミリ秒）。
適用範囲	cspybat への送信。	
説明	このオプションを使用して、最長実行時間を制限します。	
		このオプションは、IDE では使用できません。

--verify_download

構文	--verify_download
適用範囲	すべての C-SPY ドライバ。
説明	<p>このオプションは、ダウンロードしたコードイメージがターゲットメモリからリードバックでき、その内容が正しいことを確認する場合に使用します。</p> <div> [プロジェクト] > [オプション] > [デバッガ] > [ドライバ] > [ダウンロードを中止する]</div>

デバuggaオプション

この章では、IAR Embedded Workbench® IDE の C-SPY® オプションについて説明します。具体的には以下の項目を解説します。

- デバuggaオプションの設定
- デバuggaオプションについてのリファレンス情報
- C-SPY ドライバオプションのリファレンス情報

デバuggaオプションの設定

C-SPY デバuggaを起動する前に、C-SPY の一般オプションと the ターゲットシステムで必要なオプション (C-SPY ドライバ固有のオプション) をどちらも設定する必要があります。このセクションでは、**[デバugga]** カテゴリのオプションについて説明します。

IDE のデバuggaオプションを設定するには、以下の手順に従います。

- 1 **[プロジェクト] > [オプション]** を選択して、**[オプション]** ダイアログボックスを表示します。
- 2 **[カテゴリ]** リストで **[デバugga]** を選択します。
一般オプションのリファレンス情報は、以下を参照してください。
 - 365 ページの *設定*
 - 367 ページの *ダウンロード*
 - 368 ページの *追加オプション*
 - 369 ページの *イメージ*
 - 370 ページの *プラグイン*
- 3 **[設定]** ページで、**[ドライバ]** ドロップダウンリストから適切な C-SPY ドライバを選択します。

- 4 ドライバ固有オプションを設定するには、[カテゴリ] リストから該当するドライバを選択します。使用する C-SPY ドライバごとに、表示されるオプションページが異なります。

C-SPY ドライバ	使用可能なオプションのページ
C-SPYAngel デバッグモニタドライバ	371 ページの Angel
C-SPY GDB サーバドライバ	372 ページの GDB サーバ 147 ページの [ブレークポイント] オプション
C-SPY IAR ROM モニタドライバ	373 ページの IAR ROM モニタ
C-SPY J-Link/J-Trace ドライバ	374 ページの J-Link/J-Trace の設定オプション 379 ページの J-Link/J-Trace 接続オプション 147 ページの [ブレークポイント] オプション
C-SPY TI Stellaris FTDI ドライバ	381 ページの TI Stellaris FTDI の設定オプション
C-SPY Macraigor ドライバ	382 ページの Macraigor 147 ページの [ブレークポイント] オプション
RDI ドライバ	385 ページの RDI
ST-LINK ドライバ	386 ページの ST-LINK
サードパーティ製ドライバ	388 ページの サードパーティ製ドライバのオプション

表 41: 使用する C-SPY ドライバに固有のオプション

- 5 すべての設定をデフォルトの出荷時設定に戻すには、[工場出荷時設定] ボタンをクリックします。
- 6 必要なオプションをすべて設定し終わったら、[オプション] ダイアログボックスの [OK] をクリックします。

デバッグオプションについてのリファレンス情報

このセクションでは、C-SPY デバッグオプションのリファレンス情報を提供します。

設定

〔設定〕 オプションでは、使用する C-SPY ドライバ、セットアップマクロファイル、デバイス記述ファイル、を選択し、実行先のデフォルトのソースコード位置を指定します。

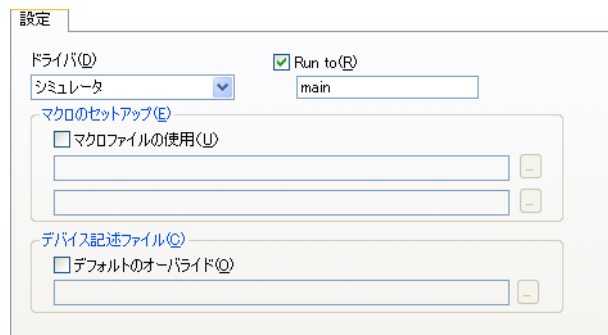


図 119: 〔デバッグ〕 設定オプション

ドライバ

ターゲットシステムの C-SPY ドライバを選択します。

シミュレータ

Angel

GDB サーバ

IAR ROM モニタ

J-Link/J-Trace

TI Stellaris FTDI

Macraigor

RDI

ST-LINK

Run to

リセット後、デバッグを起動したときに、**C-SPY** をどこまで実行するかを指定します。デフォルトでは、**C-SPY** は `main` 関数まで実行します。

デフォルトの位置をオーバーライドするには、**C-SPY** の実行先となる別の位置名を指定してください。アセンブララベルかそれに相当するもの（関数名など）を指定できます。

オプションを選択していない場合は、リセットごとにプログラムカウンタに通常のハードウェアリセットアドレスが格納されます。

セットアップマクロ

C-SPY 起動シーケンスのセットアップマクロファイルの内容を登録します。**[マクロファイルの使用]** を選択して、セットアップファイルのパスと名前を指定します。たとえば、`SetupSimple.mac` とします。拡張子を指定していない場合は、`mac` が使用されます。参照ボタンを使用して選択することもできます。

最大 2 つの異なるマクロファイルを指定できます。

デバイス記述ファイル

デフォルトのデバイス記述ファイル（**IAR** 固有の `ddf` ファイルまたは **CMSIS** システムビュー記述ファイル）が、プロジェクトの設定に基づいて自動的に選択されます。デフォルトのファイルをオーバーライドするには、**[デフォルトのオーバーライド]** を選択し、他のファイルを指定します。参照ボタンを使用して選択することもできます。

デバイス記述ファイルの詳細については、『63 ページの *デバイス記述ファイルの修正*』を参照してください。

各 **ARM** デバイスの **IAR** 固有のデバイス記述ファイルは、`arm%config` ディレクトリにあり、ファイル名の拡張子は `ddf` です。

ダウンロード

デフォルトでは、デバッグセッションが起動したときに、C-SPY によってアプリケーションが RAM またはフラッシュにダウンロードされます。[ダウンロード] オプションを使用すると、ダウンロードの動作を変更できます。

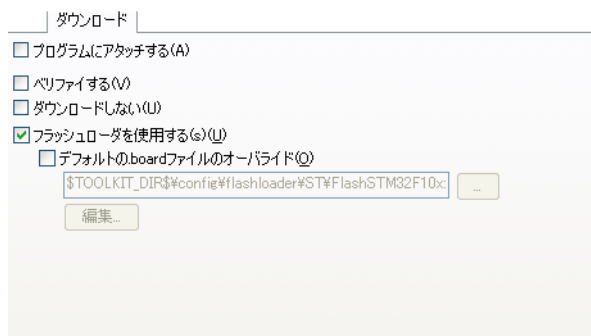


図 120: C-SPY ダウンロードオプション

プログラムにアタッチする

ターゲットシステムのリセットや中止 (J-Link のみ) をしないで、カレント位置で実行中のアプリケーションにデバッグを接続させます。このオプションの使用時に予期しない動作を回避するには、[デバッグ] > [設定] オプションの [指定位置まで実行] の選択を解除してください。

ベリファイする

ダウンロードしたコードイメージがターゲットメモリからリードバックでき、その内容が正しいことを確認します。

ダウンロードしない

現在のフラッシュの内容を保持しながら、コードのダウンロードを無効にします。このコマンドは、ターゲットメモリにすでに格納されているアプリケーションをデバッグする場合に便利です。

このオプションと [ベリファイする] オプションを組み合わせると、デバッグは不揮発性メモリからコードイメージをリードバックして、デバッグしたアプリケーションと同一かどうかをベリファイします。

注：ターゲットメモリに存在するイメージが、デバッグするための C-SPY の使用方法と常に関連することが重要です。たとえばこれは、まずデバッグ情報のない出力形式 (Intel-hex など) を使用するアプリケーションをリンクする場合、C-SPY から切り離してアプリケーションをロードします。ダウンロードせずにデバッグ目的のみで C-SPY を使用する場合、[セミホスティング] や

【IAR ブレークポイント】のオプション（【一般オプション】>【ライブラリ設定】ページから表示）のいずれかを使用してデバッグされたアプリケーションをビルドすることはできません。余分なコードが追加されて、2つの異なるコードイメージが生成されます。

フラッシュローダを使用する(s)

このオプションを使用して、フラッシュメモリへアプリケーションをダウンロードするときに1つまたは複数のフラッシュローダを使用します。フラッシュローダが、選択したチップで使用可能であれば、デフォルトで使用されます。【編集】ボタンを押し、【フラッシュローダの概要】ダイアログボックスを表示します。【編集】ボタンを押して、【フラッシュローダの概要】ダイアログボックスを表示します。

フラッシュローダの詳細は、403 ページの *フラッシュローダの使用*。

デフォルトの .board ファイルのオーバライド

デフォルトのフラッシュローダの選択は、【一般オプション】>【ターゲット】ページで選択したデバイスに基づいて行われます。デフォルトフラッシュローダをオーバライドするには、【デフォルトの .board ファイルのオーバライド】を選択し、使用するフラッシュロードのパスを指定します。参照ボタンを使用して選択することもできます。【編集】をクリックして、【フラッシュローダの概要】ダイアログボックスを表示します。詳細については、405 ページの *【フラッシュローダの概要】ダイアログボックスを参照してください*。

追加オプション

【追加オプション】ページは、C-SPY へのコマンドラインインタフェースを提供します。

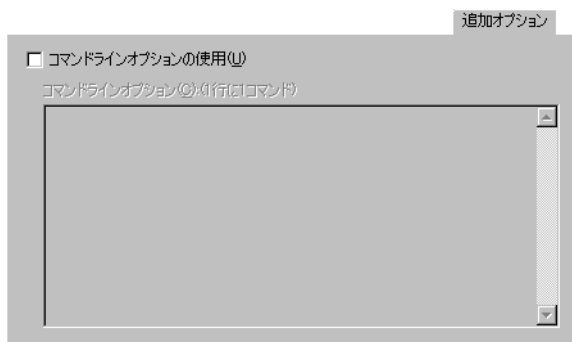


図 121: デバッグのその他のオプション

コマンドラインオプションの使用

C-SPY に引き渡される追加のコマンドライン引数を指定します（GUI でサポートされていません）。

イメージ

[イメージ] オプションは、ダウンロードする追加のデバッグファイルの使用を制御します。

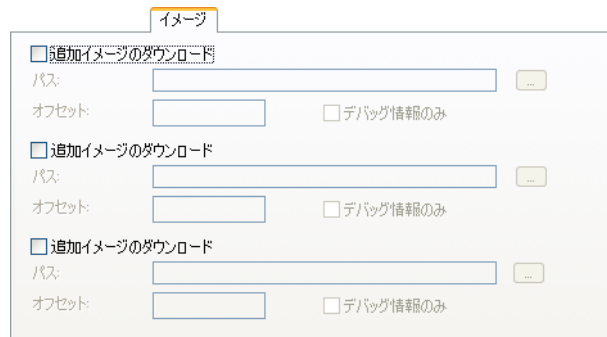


図 122: デバッガのイメージオプション

追加イメージのダウンロード

ダウンロードする追加のデバッグファイルの使用を制御します。

- | | |
|-----------------|---|
| パス | ダウンロードするデバッグファイルを指定します。参照ボタンを使用して選択することもできます。 |
| オフセット | ダウンロードしたデバッグファイルの目的地のアドレスを確定する整数を指定します。 |
| デバッグ情報のみ | 完全なデバッグファイルではなく、デバッグ情報のみをデバッガでダウンロードします。 |

4 つ以上のイメージをダウンロードするには、関連の C-SPY マクロを使用します（308 ページの `__loadImage` を参照）。

詳細については、62 ページの *複数イメージのロード* を参照してください。

プラグイン

[プラグイン] オプションでは、デバッグセッション中にロードして使用可能にする C-SPY プラグインモジュールを選択します。



図 123: デバッガのプラグインオプション

ロードするプラグインの選択

デバッグセッション中にロードして使用可能にするプラグインモジュールを選択します。このリストには、製品のインストール時に同梱されたプラグインモジュールが含まれます。

説明

プラグインモジュールについて説明しています。

場所

プラグインモジュールの位置を知らせます。
一般プラグインモジュールは、common¥plugins ディレクトリに格納されます。ターゲット固有のプラグインモジュールは、arm¥plugins ディレクトリに格納されます。

作成者

プラグインモジュールの提供元を示します。これは IAR システムズやサードパーティベンダなどです。

バージョン

バージョン番号を示します。

C-SPY ドライバオプションのリファレンス情報

このセクションでは、C-SPY ドライバオプションのリファレンス情報を提供します。

Angel

Angel オプションは、C-SPY Angel デバッグモニタドライバを制御します。

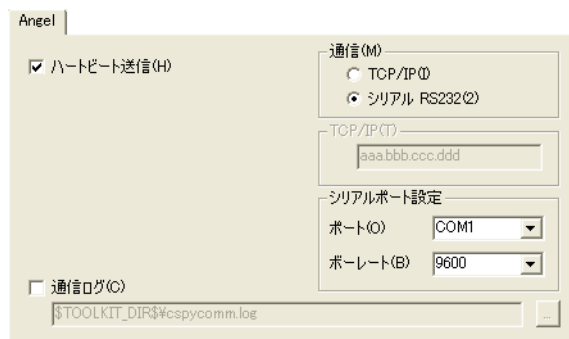


図 124: C-SPY Angel オプション

ハートビート送信

アプリケーションの実行中に C-SPY でターゲットシステムに定期的にポーリングを行います。ポーリングをすると、デバッガではターゲットアプリケーションが継続して実行しているか、または異常終了したかについて検出することができます。ハートビートを有効にすると、実行するプログラムから余分な CPU サイクルをある程度使用することになります。

通信

Angel の通信リンクを選択します。RS232 シリアルポート接続とイーサネット接続経由の TCP/IP がサポートされています。

TCP/IP

ターゲットデバイスの IP アドレスをテキストボックスに指定します。

シリアルポート設定

シリアルポートを設定します。以下を指定できます。

- ポート

Angel の通信リンクとして使用するホストコンピュータ上のポートを選択します。
- ボーレート

通信速度を設定します。

Angel のシリアル初期速度は、常に 9600 baud です。最初のハンドシェイクの後に、リンク速度は指定した速度に変更されます。通信に関する問題は、非常に高速で発生する場合があります。Angel ベースのいくつかの評価ボードは 38,400 baud を超えては動作しません。

通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、Angel モニタプロトコルに対する十分な知識が必要です。

GDB サーバ

[GDB サーバ] オプションは、STR9-comStick 評価ボード用の C-SPY GDB サーバを制御します。

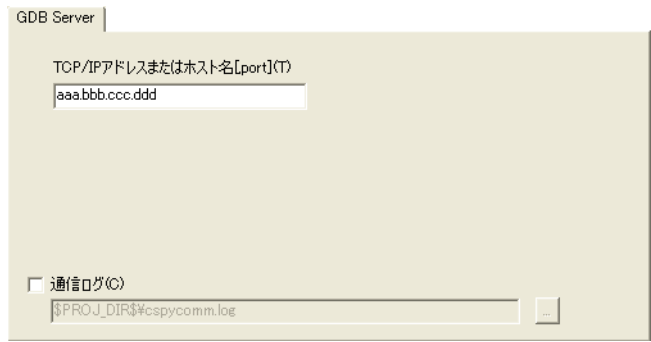


図 125: [GDB サーバ] のオプション

TCP/IP アドレスまたはホスト名

GDB サーバの IP アドレスおよびポート番号を指定します。デフォルトでは、ポート番号 3333 が使用されます。TCP/IP 接続は、リモートコンピュータで動作する J-Link サーバに接続するために使用します。

通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、JTAG インタフェースに対する十分な知識が必要です。

IAR ROM モニタ

IAR ROM モニタ オプションは、C-SPY IAR ROM モニタインタフェースを制御します。



図 126: IAR ROM モニタオプション

シリアルポート設定

シリアルポートを設定します。以下を指定できます。

- | | |
|--------------|---|
| ポート | ROM の通信リンクとして使用するホストコンピュータ上のポートを選択します。 |
| ボーレート | 通信速度を設定します。シリアルポートの通信リンク速度は、ターゲットボードで選択された速度と一致する必要があります。 |

通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、ROM モニタプロトコルに対する十分な知識が必要です。

J-Link/J-Trace の設定オプション

[設定] オプションでは、J-Link/J-Trace プローブを指定します。



図 127: J-Link/J-Trace 設定オプション

リセット

デバッガの起動時に使用するリセット方法を選択します。Cortex-M の場合、他のデバイスとは異なる方式を使用します。実際のリセット方法の種別番号は、使用可能な選択肢ごとに指定します。以下から選択します。

- ノーマル (0、デフォルト)

最初に、[コアとペリフェラル] 方式によるリセットを試行します。これが失敗した場合、[コアのみ] 方式が使用されます。ターゲットのリセットには、この方式を使用することをお勧めします。
- コア (1)

コアは VECTRESET ビットでリセットされます。ペリフェラルユニットは影響を受けません。
- コアとペリフェラル (8)

コアとペリフェラルをリセットします。
- リセットピン (2)

J-Link は RESET ピンを low に設定して、コアとペリフェラルユニットをリセットします。通常、ターゲットデバイスの CPU RESET ピンも low になり、その結果、CPU とペリフェラルユニット両方がリセットされます。
- リセット中に接続 (3)

J-Link は、リセットをアクティブにしたままでターゲットに接続します (リセットは「低」になり、ターゲットに接続中はそのままになります)。これは、STM32 デバイスの推奨リセット方式です。この方式は、STM32 デバイスでのみ使用できます。

- ブートロード後に停止** (4 または 7) NXP Cortex-M0 デバイス。これは通常のセット方式と同じですが、ブートローダの実行完了後にターゲットが停止します。これは、LPC11xx および LPC13xx デバイスの推奨リセット方式です。
- Analog Devices Cortex-M3 デバイス (7)。AIRCRR の SYSRESETREQ ビットを設定することにより、コアとペリフェラルユニットをリセットします。コアは ADI カーネルを実行できますが（これによりデバッグインタフェースが有効になります）、リセット後にユーザアプリケーションが実行されないよう徹底するために、カーネルが実行された後、最初の命令の前にコアが停止します。
- ブートロード前に停止** (5) これは通常のセット方式と同じですが、ブートローダの実行開始前にターゲットが停止します。ブートローダのデバッグが必要な場合を除いて、この方式は通常は使用しません。この方式は、LPC11xx および LPC13xx デバイスでのみ使用できます。
- ノーマル、ウォッチドッグの無効化** (6 または 9) まずノーマルのリセットを実行してコアとペリフェラルユニットをリセットし、リセット直後に CPU を停止します。CPU の停止後はウォッチドッグは無効になります。これは、ウォッチドッグはデフォルトでリセット後に実行されるためです。ターゲットアプリケーションがウォッチドッグにフィードしない場合、永久にリセットされるため J-Link からデバイスへの接続が解除されます。この方式は、Freescall Kinetis デバイス (6) および NXP LPC 1200 デバイス (9) で利用できます。
- これらの方式はすべて、JTAG および SWD インタフェースどちらにも使用できます。すべての方式は CPU をリセット後に停止します。
- その他のコアの場合は、以下の方式から選択します。
- ハードウェア、停止までの遅延時間を指定** (ms) (0) ハードウェアリセットからプロセッサの停止までの遅延時間を指定します。これは、C-SPY がアクセスを開始したときに、チップが完全な動作状態であることを確認するために使用されます。デフォルトでは遅延はゼロに設定され、できるだけ速くプロセッサを停止します。
- これはハードウェアリセットです。

ハードウェア、ブレークポイントを使用して停止 (1)	リセット後、J-Link はブレークポイントを使用して CPU の停止を継続的に試行します。通常、CPU は、リセット後間もなく停止されます。ほとんどのシステムでは、いくつかの命令を実行してから CPU を停止できます。 これはハードウェアリセットです。
ハードウェア、0 で停止 (4)	ブレークポイントをアドレスゼロに設定することでプロセッサを停止します。なお、この機能はすべての ARM マイクロコントローラでサポートされているわけではありません。 これはハードウェアリセットです。
ハードウェア、DBGRRQ を使用して停止 (5)	リセット後、J-Link は DBGRRQ を使用して CPU の停止を継続的に試行します。通常、CPU は、リセット後間もなく停止されます。ほとんどのシステムでは、いくつかの命令を実行してから CPU を停止できます。 これはハードウェアリセットです。
ソフトウェア (-)	PC をプログラムのエントリアドレスに設定します。 これはソフトウェアリセットです。
ソフトウェア、Analog デバイス (2)	Analog Devices ADuC7xxx ファミリ専用のリセットシーケンスを使用します。この方式は、[一般オプション] > [ターゲット] ページで、[デバイス] ドロップダウンリストからこの種類のデバイスを選択した場合にのみ使用できます。 これはソフトウェアリセットです。
ハードウェア、NXP LPC (9)	この方式は、[一般オプション] > [ターゲット] ページで、[デバイス] ドロップダウンリストからこの種類のデバイスを選択した場合にのみ使用できます。 これは NXP LPC デバイス専用のハードウェアリセットです。
ハードウェア、Atmel AT91SAM7 (8)	この方式は、[一般オプション] > [ターゲット] ページで、[デバイス] ドロップダウンリストからこの種類のデバイスを選択した場合にのみ使用できます。 これは Atmel AT91SAM7 ファミリ専用のハードウェアリセットです。

さまざまなリセット方式に関する詳細については、arm\doc ディレクトリの『*IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores*』（ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド）を参照してください。

ターゲットのソフトウェアリセットを使用しても、ターゲットシステムの設定値を変更することはありません。プログラムカウンタとモードレジスタ CPSR をリセット状態にするだけです。一般的に、C-SPY リセットはソフトウェアリセットだけです。[ハードウェアリセット] オプションを使用する場合、C-SPY では、デバッグの起動時に最初のハードウェアリセットを生成します。これはダウンロードの前に一度実行されます。[フラッシュロードを使用する] オプションが選択されている場合は、フラッシュダウンロード後にもう一度行われます。図 11 「フラッシュのコードをデバッグする場合のデバッグの起動」、図 12 「RAM のコードをデバッグする場合のデバッグの起動」を参照してください。



ハードウェアリセットは、アプリケーションの低レベル設定が完全でないと、問題が発生する可能性があります。低レベル設定でメモリ構成とクロックを設定しないと、ハードウェアリセット後のアプリケーションは動作しません。C-SPY でこれを処理するには、セットアップマクロの `execUserReset()` 関数が適しています。同様な例（`execUserPreload()` を使用）については、65 ページの *メモリの再配置* を参照してください。

JTAG/SWD 速度

JTAG 通信速度 (kHz) を指定します。以下から選択します。

自動

信頼性の高い動作をするための最も高い周波数を自動的に使用します。初期速度には、最大可能周波数が見つかるまで固定周波数が使用されます。一般的に、デフォルトの初期周波数 (32kHz) を使用できますが、初期リセット後にできるだけ短時間で CPU の停止が必要な場合、初期周波数を上げてください。

速い初期速度が必要な場合に設定します。リセット後にフラッシュまたは RAM から CPU で望ましくない命令（電源停止の命令など）が実行された場合などです。このような場合、初期速度が速いとデバッグではリセット後に短時間で確実に CPU を停止できます。

初期値は 1 ～ 12,000kHz の範囲である必要があります。

固定	JTAG 通信速度 (kHz) を指定します。値は 1 ～ 12,000kHz の範囲である必要があります。 JTAG 通信に関する問題や、ターゲットメモリへの書き込みに関する問題がある場合（プログラムのダウンロード中など）、速度をより低い周波数に設定すると、これらの問題が回避できる可能性があります。
クロックに同期 (A)	RTCK JTAG 信号が使用可能な ARM デバイスでのみ機能します。クロックに同期した速度について詳しくは、arm¥doc ディレクトリの『 <i>IAR J-Link and IAR J-Trace User Guide for JTAG Emulators for ARM Cores</i> 』（ARM コア向け JTAG エミュレータ IAR J-Link および IAR J-Trace ユーザガイド）を参照してください。

クロック設定

CPU クロックを指定します。以下から選択します。

CPU クロック	内部プロセッサクロック HCLK の正確なクロック周波数を指定します (MHz)。10 進数で指定できます。この値は、SWO の通信速度の設定およびタイムスタンプの計算に使用します。
SWO クロック	SWO 通信チャンネルのクロック周波数を指定します (KHz)。
自動	J-Link デバッグプローブで使用できる最大可能周波数を自動的に使用します。[自動] が選択されていない場合、希望する SWO クロックの値をテキストボックスに入力できます。10 進数で指定できます。このオプションは、送信中にデータパケットが失われる場合に使用します。

[クロック設定] オプションをオーバーライドするには、[SWO 設定] ダイアログボックスの [プロジェクトのデフォルトのオーバーライド] オプションを使用します（192 ページのプロジェクトデフォルトのオーバーライドを参照）。

J-Link/J-Trace 接続オプション

[接続] オプションでは、J-Link/J-Trace プローブとの接続を指定します。

図 128: J-Link/J-Trace 接続オプション

通信

C-SPY と J-Link デバッグプローブ間の通信チャンネルを選択します。以下から選択します。

USB

USB 接続を選択します。ドロップダウンリストでシリアル番号が選択されている場合、指定したシリアル番号の J-Link デバッグプローブが選択されます。

TCP/IP

J-Link サーバの IP アドレスを指定します。TCP/IP 接続は、リモートコンピュータで動作する J-Link サーバに接続するために使用します。

IP アドレス : LAN に接続された J-Link プローブの IP アドレスを指定します。

自動検出 : J-Link プローブを探してネットワークを自動的にスキャンします。このダウンロードを使用して、検出された J-Link プローブから選択します。

シリアル番号 : 指定したシリアル番号を持つネットワーク上の J-Link プローブに接続します。

インタフェース

J-Link デバッグプローブとターゲットシステム間の通信インタフェースを選択します。以下から選択します。

JTAG (デフォルト) JTAG インタフェースを使用します。

SWD JTAG よりも少数のピンを使用します。serial-wire output (SWO) 通信チャンネルを使用する場合は、SWD を選択します。[一般オプション] > [ライブラリ構成] ページで SWO 経由の stdout/stderr を選択すると、SWD が自動的に選択されることに注意してください。SWO 設定の詳細については、188 ページの [SWO トレースウィンドウ設定] ダイアログボックスを参照してください。

JTAG スキャンチェーン

JTAG スキャンチェーンを指定します。以下から選択します。

JTAG スキャンチェーン JTAG スキャンチェーンに複数のデバイスがあること
ン (マルチターゲット) を指定します。

TAP 番号 接続先のデバイスの TAP (Test Access Port) 位置を指定
します。TAP 番号はゼロから始まります。

**スキャンチェーンに非
ARM デバイスが含ま
れています** FPGA など、ARM デバイスと他のデバイスを混在さ
せる JTAG スキャンチェーンを有効にします。

先行ビット デバッグ対象の ARM デバイスの前の IR ビット数を
指定します。

通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、JTAG インタフェースに対する十分な知識が必要です。

TI Stellaris FTDI の設定オプション

[設定] オプションでは、TI Stellaris FTDI インタフェースを指定します。



図 129: TI Stellaris FTDI 設定オプション

インタフェース

J-Link デバッグプローブとターゲットシステム間の通信インタフェースを選択します。以下から選択します。

JTAG (デフォルト) JTAG インタフェースを使用します。

SWD JTAG よりも少数のピンを使用します。serial-wire output (SWO) 通信チャンネルを使用する場合は、SWD を選択します。[一般オプション] > [ライブラリ構成] ページで SWO 経由の stdout/stderr を選択すると、SWD が自動的に選択されることに注意してください。SWO 設定の詳細については、188 ページの [SWO トレースウィンドウ設定] ダイアログボックスを参照してください。

JTAG/SWD 速度

JTAG 通信速度 (kHz) を指定します。

通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。結果を分析するには、通信プロトコルに関する詳しい知識が必要です。

Macraigor

Macraigor オプションでは、Macraigor インタフェースを指定します。

Macraigor

OCDインタフェースデバイス(O)
usb2Demon

☐ ハードウェアリセット(H)
リセット後のディレイ

☐ JTAGスキャンチェーン(マルチターゲット)(J)
0@ARM7TDMI

デバッグハンドアドレス
0x00800000

☐ 通信ログ(C)
\$TOOLKIT_DIR\$%cspycomm.log

インタフェース(E)
☒ JTAG(J)
☐ SWD(S)

JTAG速度(S)
2

TOP/IP(T)
aaa.bbb.ccc.ddd

ポート(O)
USB0

ボーレート(B)
115200

図 130: Macraigor オプション

OCD インタフェースデバイス

使用しているハードウェアインタフェースに対応するデバイスを選択します。サポートされている Macraigor JTAG プロブは、Macraigor **mpDemon** です。

インタフェース

J-Link デバッグプロブとターゲットシステム間の通信インタフェースを選択します。以下から選択します。

JTAG (デフォルト) JTAG インタフェースを使用します。

SWD

JTAG よりも少数のピンを使用します。serial-wire output (SWO) 通信チャンネルを使用する場合は、**SWD** を選択します。[一般オプション] > [ライブラリ構成] ページで SWO 経由の stdout/stderr を選択すると、SWD が自動的に選択されることに注意してください。SWO 設定の詳細については、188 ページの [SWO トレースウィンドウ設定] ダイアログボックスを参照してください。

JTAG 速度

JTAG プローブと ARM JTAG ICE ポート間の速度を指定します。この値は 1 ～ 8 の範囲にあり、スキヤククロックの生成時に JTAG プローブクロックが分割された係数を設定する必要があります。



mpDemon インタフェースには、低速である大きな値 (2 や 3 など) の設定が必要となる場合があります。

TCP/IP

イーサネット / LAN ポートに接続された JTAG プローブの IP アドレスを指定します。

ポート

通信リンクとして使用するホストコンピュータ上のシリアルポートまたはパラレルポートを選択します。JTAG プローブが接続されるホストのポートを選択します。

パラレルポートでは、コンピュータが 1 つのパラレルポートを搭載している場合、通常は LPT1 を使用してください。なお、ラップトップコンピュータにはその 1 つのパラレルポートを LPT2 または LPT3 にマッピングしている場合があります。できれば、EPP モードが高速であるため、パラレルポートをこのモードに設定してください。双方向で互換性のあるモードでは、動作はしますが低速となります。

ボーレート

シリアル通信速度を選択します。

ハードウェアリセット

デバッグの起動時に最初のハードウェアリセットを生成します。これはダウンロードの前に一度実行されます。**[フラッシュロードを使用する]** オプションが選択されている場合は、フラッシュダウンロード後にもう一度行われます。図 11 「フラッシュのコードをデバッグする場合のデバッグの起動」および図 12 「RAM のコードをデバッグする場合のデバッグの起動」を参照してください。

ターゲットのソフトウェアリセットを使用しても、ターゲットシステムの設定値を変更することはありません。プログラムカウンタをリセット状態にするだけです。一般的に、C-SPY リセットはソフトウェアリセットだけです。



ハードウェアリセットは、アプリケーションの低レベル設定が完全でないと、問題が発生する可能性があります。低レベル設定でメモリ構成とクロックを設定しないと、ハードウェアリセット後のアプリケーションは動作しません。C-SPY でこれを処理するには、セットアップマクロの `execUserReset ()` 関

数が適しています。同様な例（`execUserPreload()`）については、*65 ページのメモリの再配置*を参照してください。

JTAG スキャンチェーン（マルチターゲット）

JTAG スキャンチェーン上に複数のデバイスがある場合に、各デバイスを定義します。また、どのデバイスに接続するか指定する必要があります。構文は以下のとおりです。

```
<0>@dev0, dev1, dev2, dev3, ...
```

ここで、0 は接続先デバイスの TAP 番号で、`dev0` は Macraigor JTAG プローブで最も近い TDO ピンです。

デバッグハンドラアドレス

Intel XScale デバイスで使用されるデバッグハンドラの位置（メモリアドレス）を指定します。メモリ空間を保存するには、キャッシュ RAM の一部がマッピングできるアドレスを指定してください。その位置には物理メモリは含まれていません。できれば、下位 16MB メモリの未使用領域を見つけて、そこにハンドラのアドレスを置きます。

通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、JTAG インタフェースに対する十分な知識が必要です。

RDI

[RDI] オプションを使用すると、ARM Ltd. RDI 1.5.1 仕様に準拠する JTAG インタフェースを使用できます。こうしたインタフェースの一例が、ARM RealView Multi-ICE JTAG インタフェースです。



図 131: RDI オプション

Manufacturer RDI driver

JTAG ボードを提供する RDI ドライバ DLL ファイルのファイルパスを指定します。

ハードウェアリセットを許可

エミュレータでターゲットのハードウェアリセットを実行可能にします。



ターゲットのソフトウェアリセットを使用しても、ターゲットシステムの設定値を変更することはありません。プログラムカウンタをリセット状態にするだけです。

アプリケーションの低レベル設定が完全である場合にのみ、ハードウェアリセットを行うことができます。低レベル設定でメモリ構成とクロックを設定しないと、ハードウェアリセット後のアプリケーションは動作しません。

C-SPY でこれを処理するには、セットアップマクロの `execUserReset()` 関数が適しています。同様な例 (`execUserPreload()` を使用) については、65 ページの **メモリの再配置** を参照してください。

注： このオプションを使用するには、使用する RDI ドライバでハードウェアリセットがサポートされている必要があります。

例外をキャッチ

例外がブレークポイントとして処理されるようにします。実行中のプログラムが定義したように例外処理を行うのではなく、デバグが停止します。
取得できる ARM コアの例外は以下のとおりです。

例外	説明
リセット	リセット
未定義	未定義の命令
SWI	ソフトウェア割り込み
データ	データの異常終了（データアクセスのメモリ障害）
プリフェッチ	プリフェッチの異常終了（命令フェッチのメモリ障害）
IRQ	通常の割り込み
FIQ	高速割り込み

表 42: 例外の取得

RDI とのコミュニケーションのログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。
ロギングされたデータを解析するには、RDI インタフェースに対する十分な知識が必要です。

ST-LINK

[ST-LINK] ページには、ST-LINK プローブのオプションが含まれます。



図 132: ST-LINK（設定オプション）

リセット

デバッグの起動時に使用するリセット方法を選択します。実際のリセット方法の種別番号は、使用可能な選択肢ごとに指定します。以下から選択します。

- ノーマル (0) 標準のリセット処理を実行します。
- リセットピン (1) リセットピンを使用して、ハードウェアリセットを実行します。ST-LINK バージョン 2 でのみ使用できます。
- リセット中の接続 (2) ST-LINK は、リセットピンをアクティブにしたままターゲットに接続します（リセットピンは「低」になり、ターゲットに接続中はそのままになります）。ST-LINK バージョン 2 でのみ使用できます。

インタフェース

ST-LINK デバッグプローブとターゲットシステム間の通信インタフェースを選択します。以下から選択します。

- JTAG（デフォルト） JTAG インタフェースを使用します。
- SWD JTAG よりも少数のピンを使用します。

クロック設定

CPU クロックを指定します。以下から選択します。

- CPU クロック 内部プロセッサクロック HCLK の正確なクロック周波数を指定します (MHz)。10 進数で指定できます。この値は、SWO の通信速度の設定およびタイムスタンプの計算に使用します。
- SWO クロック SWO 通信チャンネルのクロック周波数を指定します (KHz)。
- 自動 J-Link デバッグプローブで利用できる最大可能周波数を自動的に使用します。[自動] が選択されていない場合、希望する SWO クロックの値をテキストボックスに入力できます。10 進数で指定できます。このオプションは、送信中にデータパケットが失われる場合に使用します。

[クロック設定] オプションをオーバーライドするには、[SWO 設定] ダイアログボックスの [プロジェクトのデフォルトのオーバーライド] オプションを使用します (192 ページのプロジェクトデフォルトのオーバーライドを参照)。

サードパーティ製ドライバのオプション

[サードパーティ製ドライバ] オプションは、サードパーティベンダが提供するドライバプラグインをロードする場合に使用されます。これらのドライバは C-SPY デバッガドライバ仕様に準拠している必要があります。



図 133: C-SPY サードパーティ製ドライバオプション

IAR デバッガのドライバプラグイン

サードパーティ製ドライバプラグインの DLL ファイルのファイルパスを指定します。参照ボタンを使用して選択することもできます。

ダウンロードを中止する

現在のフラッシュの内容を保持しながら、コードのダウンロードを無効にします。このコマンドは、しばらく C-SPY を終了する必要があり、コードをダウンロードしないでデバッグセッションを続行する場合に便利です。ただし、起動時に C-SPY が実行する暗黙的 RESET は無効にはなりません。

このオプションを [すべてベリファイ] と組み合わせた場合、デバッガはフラッシュメモリからアプリケーションをリードバックして、現在デバッグ中のアプリケーションと同一かどうかを検証します。

このオプションは、サードパーティ製ドライバでサポートされている場合に使用できます。

すべてベリファイ

ターゲットシステム上のメモリがライト可能であり、一貫した方法でマッピングされているか検証します。ダウンロード中に問題があれば、ワーニングメッセージが表示されます。各バイトがロード後にチェックされます。これは時間がかかりますが、メモリを完全にチェックします。このオプションは、サードパーティ製ドライバでサポートされている場合に使用できます。

通信ログ

C-SPY とターゲットシステムとの間の通信がファイルにロギングされます。ロギングされたデータを解析するには、インタフェースに対する十分な知識が必要です。このオプションは、サードパーティ製ドライバでサポートされている場合に使用できます。

C-SPY ドライバについての追加情報

この章では、C-SPY® ドライバの追加メニューや機能について説明します。具体的には以下の項目を解説します。

- C-SPY シミュレータ
- C-SPY GDB サーバドライバ
- C-SPY J-Link/J-Trace ドライバ
- C-SPY TI Stellaris FTDI ドライバ
- C-SPY Macraigor ドライバ
- C-SPY RDI ドライバ
- C-SPY ST-LINK ドライバ

C-SPY シミュレータ

このセクションでは、C-SPY シミュレータに関する追加リファレンス情報を提供します。これらは本書のここだけに記載されている情報です。

具体的には以下の項目を解説します。

- 392 ページの シミュレータのメニュー

シミュレータのメニュー

シミュレータドライバを使用する場合、[シミュレータ] メニューがメニューバーに追加されます。

✓ 割込み設定(U)... 強制割込み(F) 割込みログ(L) 割込みログ概要 メモリアクセスの設定(M)...
トレース(T) 関数トレース(W) 関数プロファイラ(Q)
タイムライン(N)
ブレークポイントの使用(B)

図 134: [シミュレータ] メニュー

メニューから実行できるコマンドは、以下のとおりです。

割込み設定	C-SPY の割込みシミュレーションを設定するためのダイアログボックスを開きます (「264 ページの [割込み設定] ダイアログボックス」を参照)。
強制割込み	割込みをすぐにトリガできるウィンドウを開きます (268 ページの [強制割込み] ウィンドウを参照)。
割込みログ	定義されているすべての割込みのステータスを表示するウィンドウを開きます (271 ページの [割込みログ] ウィンドウを参照)。
割込みログ概要	割込みの入口と出口の記録の概要を表示するウィンドウが開きます (274 ページの [割込みログ概要] ウィンドウを参照)。
メモリアクセスの設定	さまざまなアクセスタイプでメモリエリアを指定して、メモリアクセスチェックをシミュレーションするダイアログボックスを開きます (173 ページの [メモリアクセス設定] ダイアログボックスを参照)。
トレース	収集されたトレースデータを表示するウィンドウが開きます (193 ページの [トレース] ウィンドウを参照)。
関数トレース	関数の呼出しや復帰を示すトレースデータを表示するウィンドウが開きます (199 ページの [関数トレース] ウィンドウを参照)。

関数プロファイラ	関数のタイミング情報を示すウィンドウを表示します (229 ページの [関数プロファイラ] ウィンドウを参照)。
タイムライン	割込みログと呼出しスタックのトレースデータを示すウィンドウを開きます (200 ページの [タイムライン] ウィンドウを参照)。
ブレークポイントの使用	アクティブなすべてのブレークポイントのリストを表示するダイアログボックスを開きます (135 ページの [ブレークポイントの使用] ダイアログボックスを参照)。

C-SPY GDB サーバドライバ

このセクションでは、C-SPY GDB サーバドライバに関する追加リファレンス情報を提供します。これらは本書のここだけに記載されている情報です。

具体的には以下の項目を解説します。

- 393 ページの [GDB サーバ] メニュー

[GDB サーバ] メニュー

C-SPYGDB サーバドライバを使用する場合、[GDB サーバ] メニューがメニューバーに追加されます。

ブレークポイントの使用(B) ...

図 135: [GDB サーバ] メニュー

このコマンドはメニューから実行できます。

ブレークポイントの使用	アクティブなすべてのブレークポイントのリストを表示するダイアログボックスを開きます (135 ページの [ブレークポイントの使用] ダイアログボックスを参照)。
-------------	--

C-SPY J-Link/J-Trace ドライバ

このセクションでは、C-SPY J-Link/J-Trace ドライバに関する追加リファレンス情報を提供します。これらは本書のここだけに記載されている情報です。

具体的には以下の項目を解説します。

- 394 ページの J-Link メニュー

J-Link メニュー

C-SPYJ-Link ドライバを使用する場合、[J-Link] メニューがメニューバーに追加されます。

ウォッチポイント(W)... ベクタキャッチ(V)... ステップ実行時の割込みの無効化(A)
ETMトレース設定(B)... ETMトレースの保存(S)... ETMトレース(T) 関数トレース(F)
SWO 設定(O)... SWO トレースウィンドウ設定(G)... SWOTレースの保存(E)... SWOTレース(C) 割込みログ(N) 割込みログの一覧(U) データログ(D) データログの一覧(M) Powerログの設定 パワログ(P)
タイムラインΦ
関数プロファイル(L)
ブレークポイントの使用(B)

図 136: J-Link メニュー

メニューから実行できるコマンドは、以下のとおりです。

- ウォッチポイント

ウォッチポイントを設定するダイアログボックスが表示されます (136 ページの [コード] ブレークポイントダイアログボックスを参照)。
- ベクタキャッチ

割込みベクタテーブルのベクタに直接ブレークポイントを設定するダイアログボックスが表示されます (125 ページの 例外ベクタのブレークポイントを参照)。なお、このコマンドはすべての ARM コアに使用できるわけではありません。

ステップ実行中の 割込みを無効化	ステップ実行済みの命令のみが実行されるように徹底します。割込みは実行されません。このコマンドは、フルスピードで実行中でない場合に使用可能です。また、一部の割込みはデバッグプロセスを妨害します。
ETM トレース 設定 ²	ETM トレースデータの生成と収集を設定するダイアログボックスを表示します (186 ページの [ETM トレース設定] ダイアログボックスを参照)。
ETM トレース の保存 ²	収集されたトレースデータをファイルに保存するダイアログボックスを表示します (198 ページの [トレースの保存] ダイアログボックスを参照)。
ETM トレース ²	[ETM トレース] ウィンドウを開いて、収集したトレースデータを表示します (193 ページの [トレース] ウィンドウを参照)。
関数トレース ²	関数の呼出しや復帰を示すトレースデータを表示するウィンドウが開きます (199 ページの [関数トレース] ウィンドウを参照)。
SWO 設定 ¹	ダイアログボックスが開きます (190 ページの [SWO 設定] ダイアログボックスを参照)。
SWO トレース ウィンドウの 設定 ¹	ダイアログボックスが開きます (188 ページの [SWO トレースウィンドウ設定] ダイアログボックスを参照)。
SWO トレース の保存 ¹	収集されたトレースデータをファイルに保存するダイアログボックスを表示します (198 ページの [トレースの保存] ダイアログボックスを参照)。
SWO トレース ¹	[SWO トレース] ウィンドウを開いて、収集したトレースデータを表示します (193 ページの [トレース] ウィンドウを参照)。
割込みログ ¹	ウィンドウを開きます (271 ページの [割込みログ] ウィンドウを参照)。
割込みログ一覧 ¹	ウィンドウを開きます (274 ページの [割込みログ概要] ウィンドウを参照)。
データログ ¹	ウィンドウを開きます (115 ページの [データログ] ウィンドウを参照)。
データログ一覧 ¹	ウィンドウを開きます (117 ページの [データログ概要] ウィンドウを参照)。

Power ログの設定	ウィンドウを開きます (244 ページの <i>[Power ログ設定]</i> ウィンドウを参照)。
Power ログ	ウィンドウを開きます (246 ページの <i>[Power ログ]</i> ウィンドウを参照)。
タイムライン³	ウィンドウを開きます (200 ページの <i>[タイムライン]</i> ウィンドウを参照)。
関数プロファイラ	関数のタイミング情報を示すウィンドウを表示します (229 ページの <i>[関数プロファイラ]</i> ウィンドウを参照)。
ブレークポイントの使用	アクティブなすべてのブレークポイントのリストを表示するダイアログボックスを開きます (135 ページの <i>[ブレークポイントの使用]</i> ダイアログボックスを参照)。

1 SWD/SWO インタフェースの使用時のみ
2 ETB を持つ ETM か J-Link の使用時のみ
3 ETM または SWD/SWO の使用時のみ

ライブウォッチおよび DCC の使用

以下の場合にライブウォッチを使用します。

Cortex-M の場合

メモリへのアクセスやブレークポイントは、実行中常に設定可能です。DCC (デバッグ通信チャンネル) ユニットは使用できません。

ARMxxx-S デバイスの場合

ハードウェアブレークポイントは、実行中常に設定可能です。

ARM7/ARM9 デバイスの場合 (ARMxxx-S を含む)

アプリケーションからメモリへのアクセスが発生します。DCC ユニット経由でデバッガと通信する小さなプログラム (DCC ハンドラ) をアプリケーションに追加すると、実行中にメモリが読み取り/書き込み可能になります。ソフトウェアブレークポイントも、DCC ハンドラによって設定できます。

arm¥src¥debugger¥dcc にあるファイル JLINKDCC_Process.c と JLINKDCC_HandleDataAbort.s をプロジェクトに追加して、JLINKDCC_Process 関数をミリ秒ごとなど定期的に呼び出します。

cstartup ファイルのローカルコピーで、データ異常終了によって JLINKDCC_HandleDataAbort ハンドラが呼び出されるように割込みベクタテーブルを変更します。

ターミナル I/O および DCC の使用

以下の場合にターミナル I/O を使用します。

Cortex-M の場合

193 ページの *ITM 事象ポート* を参照してください。

ARM7/ARM9 デバイスの場合 (ARMxxx-S を含む)

ファイル `arm\src\debugger\dcc\DCC_Write.c` をプロジェクトに追加することによって、DCC をターミナル I/O 出力に使用できます。DCC_write.c は、ライブラリ関数 `write` をオーバーライドします。printf などの関数を使用して、テキストをリアルタイムで C-SPY の [ターミナル I/O] ウィンドウに出力できます。

この場合、セミホストを無効にして、ブレークポイントを他の用途に解放できます。セミホストを無効にするには、[一般オプション] > [ライブラリ構成] > [ライブラリ低レベルインタフェースの実装] > [なし] を選択します。

C-SPY TI Stellaris FTDI ドライバ

このセクションでは、C-SPY TI Stellaris FTDI ドライバに関する追加リファレンス情報を提供します。これらは本書のここだけに記載されている情報です。

具体的には以下の項目を解説します。

- 397 ページの *TI Stellaris FTDI メニュー*

TI Stellaris FTDI メニュー

C-SPY TI Stellaris FTDI ドライバを使用する際は、[TI Stellaris FTDI] メニューがメニューバーに追加されます。

ブレークポイントの使用(B) ...

図 137: [TI Stellaris FTDI] メニュー

このコマンドはメニューから実行できます。

ブレークポイントの使用

アクティブなすべてのブレークポイントのリストを表示するダイアログボックスを開きます (135 ページの [ブレークポイントの使用] ダイアログボックスを参照)。

C-SPY Macraigor ドライバ

このセクションでは、C-SPY Macraigor ドライバに関する追加リファレンス情報を提供します。これらは本書のここだけに記載されている情報です。

具体的には以下の項目を解説します。

- 398 ページの *Macraigor* の [JTAG] メニュー

Macraigor の [JTAG] メニュー

C-SPY Macraigor ドライバを使用する際は、[JTAG] メニューがメニューバーに追加されます。

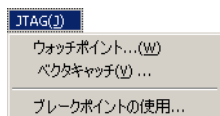


図 138: Macraigor の [JTAG] メニュー

メニューから実行できるコマンドは、以下のとおりです。

ウォッチポイント	ウォッチポイントを設定するダイアログボックスを開きます (136 ページの [コード] ブレイクポイントダイアログボックスを参照)。
ベクタキャッチ	割込みベクタテーブルのベクタに直接ブレイクポイントを設定するダイアログボックスを開きます (125 ページの 例外ベクタのブレイクポイントを参照)。なお、このコマンドはすべての ARM コアに使用できるわけではありません。
ブレイクポイントの使用	アクティブなすべてのブレイクポイントのリストを表示するダイアログボックスを開きます (135 ページの [ブレイクポイントの使用] ダイアログボックスを参照)。

C-SPY RDI ドライバ

このセクションでは、C-SPY RDI ドライバに関する追加リファレンス情報を提供します。これらは本書のここだけに記載されている情報です。

具体的には以下の項目を解説します。

- 399 ページの *RDI* メニュー

RDI メニュー

C-SPY RDI ドライバを使用する際は、**[RDI]** メニューがメニューバーに追加されます。

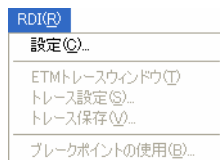


図 139: RDI メニュー

メニューから実行できるコマンドは、以下のとおりです。

設定	RDI ドライバベンダから提供されたダイアログボックスを開きます。このダイアログボックスの詳細については、ドライバの資料を参照してください。
トレース設定	ETM トレースを設定するダイアログボックスを表示します（186 ページの <i>[ETM トレース設定]</i> ダイアログボックスを表示）。
トレースの保存	収集されたトレースデータをファイルに保存するダイアログボックスを表示します（198 ページの <i>[トレースの保存]</i> ダイアログボックスを参照）。
ブレイクポイントの使用	アクティブなすべてのブレイクポイントのリストを表示するダイアログボックスを開きます（135 ページの <i>[ブレイクポイントの使用]</i> ダイアログボックスを参照）。

注： 設定ダイアログボックスのデフォルト設定値を取得するには、プロジェクトに特別な設定が必要ない場合であっても、ダイアログボックスを開く / 閉じるためだけにいくつかの RDI ドライバが必要です。

C-SPY ST-LINK ドライバ

このセクションでは、C-SPY ST-LINK ドライバに関する追加リファレンス情報を提供します。これらは本書のここだけに記載されている情報です。

具体的には以下の項目を解説します。

- 400 ページの *ST-LINK* メニュー

ST-LINK メニュー

C-SPY ST-LINK ドライバを使用する際は、[ST-LINK] メニューがメニューバーに追加されます。

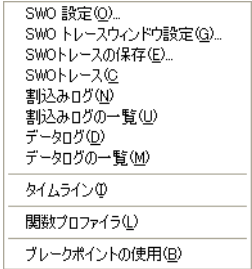


図 140: [ST-LINK] メニュー

メニューから実行できるコマンドは、以下のとおりです。

- SWO 設定¹

ダイアログボックスが開きます (190 ページの [SWO 設定] ダイアログボックスを参照)。
- SWO トレースウィンドウの設定¹

ダイアログボックスが開きます (188 ページの [SWO トレースウィンドウ設定] ダイアログボックスを参照)。
- SWO トレースの保存¹

収集されたトレースデータをファイルに保存するダイアログボックスを表示します (198 ページの [トレースの保存] ダイアログボックスを参照)。
- SWO トレース¹

[SWO トレース] ウィンドウを開いて、収集したトレースデータを表示します (193 ページの [トレース] ウィンドウを参照)。
- 割込みログ¹

ウィンドウを開きます (271 ページの [割込みログ] ウィンドウを参照)。
- 割込みログ概要¹

ウィンドウを開きます (274 ページの [割込みログ概要] ウィンドウを参照)。
- データログ¹

ウィンドウを開きます (115 ページの [データログ] ウィンドウを参照)。
- データログ概要¹

ウィンドウを開きます (117 ページの [データログ概要] ウィンドウを参照)。
- タイムライン²

ウィンドウを開きます (200 ページの [タイムライン] ウィンドウを参照)。

- 関数プロファイラ** 関数のタイミング情報を示すウィンドウを表示します
(229 ページの [関数プロファイラ] ウィンドウを参照)。
- ブレークポイントの使用** アクティブなすべてのブレークポイントのリストを表示
するダイアログボックスを開きます (135 ページの [ブ
レークポイントの使用] ダイアログボックスを参照)。

1 SWD/SWO インタフェースの使用時のみ。

2 ETM または SWD/SWO の使用時のみ。

フラッシュローダの使用

この章では、フラッシュローダの機能と利用方法について説明します。具体的には以下の項目を解説します。

- フラッシュローダの概要
- フラッシュローダについてのリファレンス情報

フラッシュローダの概要

このセクションではフラッシュローダの概要を説明します。

このセクションでは、以下のトピックについて説明します。

- フラッシュローダの概要について
- フラッシュローダの設定
- フラッシュローディング機構

フラッシュローダの概要について

フラッシュローダは、ターゲットにダウンロードされるエージェントです。デバッガからアプリケーションをフェッチして、フラッシュメモリにプログラムします。フラッシュローダでは、ファイル I/O 機構を使用してホストからアプリケーションプログラムを読み込みます。1 つまたは複数のフラッシュローダを選択できます。各フラッシュローダでは、アプリケーションの選択した部分をロードします。すなわち、異なるフラッシュローダを使用して、アプリケーションのさまざまな部分をロードすることができます。

さまざまなマイクロコントローラに対するフラッシュローダセットが、ARM 用 IAR Embedded Workbench に用意されています。これらのローダに加えて、多くのフラッシュローダがチップメーカーおよびサードパーティベンダから提供されています。独自のフラッシュローダを実装できるように、フラッシュローダの API、ドキュメント、およびいくつかの実装例が使用できます。

フラッシュローダの設定

アプリケーションのダウンロードにフラッシュローダを使用するには、以下の手順に従います。

- 1 [プロジェクト] > [オプション] を選択します。
- 2 [デバッガ] カテゴリを選択して、[ターゲット] タブをクリックします。

- 3 **【フラッシュローダを使用する】** オプションを選択します。指定したデバイスに設定されたデフォルトのフラッシュローダが使用されます。設定は、事前に定義された board ファイルで指定します。
- 4 デフォルトのフラッシュローダをオーバーライドしたり、自分のボードに合わせてデフォルトのフラッシュローダの動作を変更するには、**【デフォルトの .board ファイルのオーバーライド】** オプションに続いて **【編集】** を選択し、**【フラッシュローダの構成】** ダイアログボックスを開きます。 .board ファイルのコピーがプロジェクトディレクトリに作成され、それによって .board ファイルのパスが更新されます。
- 5 **【フラッシュローダの概要】** ダイアログボックスでは、現在設定されているフラッシュローダをすべて一覧表示します。405 ページの **【フラッシュローダの概要】** ダイアログボックスを参照してください。フラッシュローダを選択できます。あるいは、**【フラッシュローダの構成】** ダイアログボックスを開くことができます。

【フラッシュローダの構成】 ダイアログボックスでは、ダウンロードを設定できます。さまざまなフラッシュローダオプションの詳細については、407 ページの **【フラッシュローダの構成】** ダイアログボックスを参照してください。

フラッシュローディング機構

【フラッシュローダを使用する】 オプションが選択され、1 つまたは複数のフラッシュローダが設定されると、デバッグセッションの開始時に以下の手順が実行されます。

- 1 C-SPY では、フラッシュローダをターゲット RAM にダウンロードします。
- 2 C-SPY では、フラッシュローダの実行を開始します。
- 3 フラッシュローダは、アプリケーションコードをフラッシュメモリにプログラムします。
- 4 フラッシュローダが終了します。
- 5 C-SPY では、コンテキストをユーザアプリケーションに切り替えます。

アプリケーションの各メモリ範囲に対して、手順 2 から 4 が実行されます。

選択したフラッシュローダごとに、手順 1 から 4 まで実行します。

フラッシュローダについてのリファレンス情報

このセクションでは、以下のウィンドウおよびダイアログボックスのリファレンス情報を提供します。

- 405 ページの [フラッシュローダの概要] ダイアログボックス
- 407 ページの [フラッシュローダの構成] ダイアログボックス

[フラッシュローダの概要] ダイアログボックス

[フラッシュローダの概要] ダイアログボックスは、[デバッグ] > [ダウンロード] ページから使用できます。

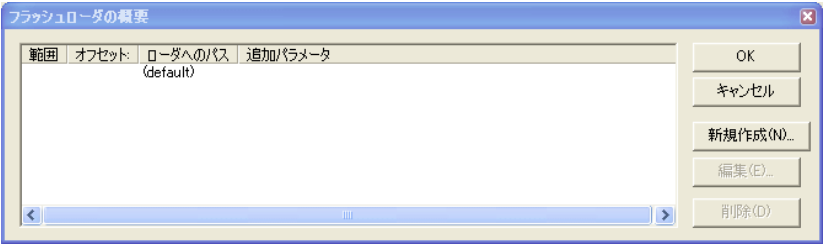


図 141: [フラッシュローダの概要] ダイアログボックス

このダイアログボックスには、定義済みのフラッシュローダがすべて一覧表示されます。[一般オプション] > [ターゲット] ページでフラッシュローダのあるデバイスを選択した場合、デフォルトでは、このフラッシュローダは [フラッシュローダの概要] ダイアログボックスに一覧表示されます。

表示エリア

表示エリアの各列には、メモリの特定の部分をフラッシュするフラッシュローダの設定が表示されます。

範囲	選択したフラッシュローダによってプログラミングされる、アプリケーションの部分
オフセット / アドレス	アプリケーションがフラッシュされる、メモリの開始位置 アドレスの先頭に a が付いている場合は絶対アドレスです。 それ以外の場合は、メモリの開始部分への相対オフセットです
ローダへのパス	使用されるフラッシュローダ *.flash ファイルへのパス (古いスタイルのフラッシュローダの場合は *.out)

追加パラメータ フラッシュローダに渡される追加パラメータの一覧

列のヘッダをクリックして、範囲、オフセット / アドレスなどの順にリストをソートします。

機能ボタン

以下の機能ボタンを使用できます。

OK	選択したフラッシュローダを使用して、アプリケーションをメモリにダウンロードします。
キャンセル	標準の「キャンセル」。
新規	使用するフラッシュローダを指定できるダイアログボックスが表示されます (407 ページの [フラッシュローダの構成] ダイアログボックスを参照)。
編集	選択したフラッシュローダの設定を変更できるダイアログボックスが表示されます (407 ページの [フラッシュローダの構成] ダイアログボックスを参照)。
削除	選択されたフラッシュローダ設定を削除します。

「フラッシュローダの構成」ダイアログボックス

「フラッシュローダの構成」ダイアログボックスは、「フラッシュローダの概要」ダイアログボックスから使用できます。

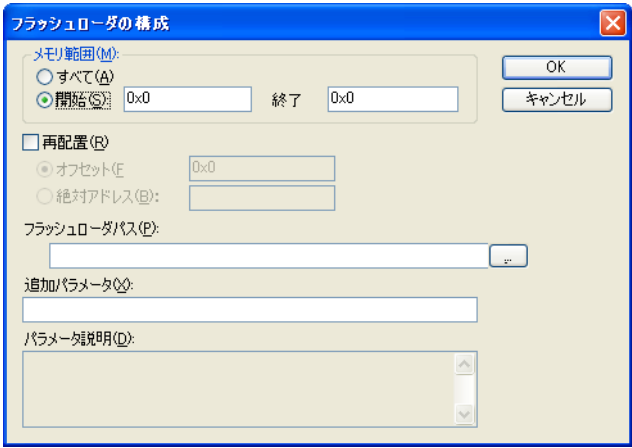


図 142: 「フラッシュローダの構成」ダイアログボックス

「フラッシュローダの構成」ダイアログボックスを使用して、使用するボードに合ったダウンロードを設定します。デフォルトの board ファイルのコピーが、プロジェクトディレクトリに作成されます。

メモリ範囲

フラッシュメモリにダウンロードするアプリケーションの部分を指定します。以下から選択します。

- | | |
|---------|--|
| すべて | すべてのアプリケーションが、このフラッシュローダを使用してダウンロードされます。 |
| 開始 / 終了 | アプリケーションをダウンロードするメモリエリアの開始と終了を指定します。 |

再配置

デフォルトのフラッシュベースアドレスをオーバーライドします。つまりメモリ内のアプリケーション位置を再配置します。アプリケーションを、リンクされていた位置とは別の所にフラッシュできます。以下から選択します。

- | | |
|-------|---|
| オフセット | 相対オフセットの数値。このオフセットは、アプリケーションファイルのアドレスに追加されます。 |
|-------|---|

絶対アドレス アプリケーションがフラッシュされる、絶対ベースアドレスの数値。アプリケーションの最も小さいアドレスが、このアドレスに配置されます。絶対アドレスを指定する場合、アプリケーションに 1 つだけしかフラッシュロードを使用できない点に注意してください。

使用できる数値の形式は以下のとおりです。

- 123456 10 進数
- 0x123456 16 進数
- 0123456 8 進数

最初のバイト（最も低いアドレス）をフラッシュに書き込むために使用されるデフォルトベースアドレスは、アプリケーション用のリンカ構成ファイルで指定します。しかし、フラッシュベースアドレスをオーバーライドして、アドレス空間の別の場所で起動することが必要な場合もあります。たとえば、フラッシュメモリの位置を再配置するデバイスに必要となることがあります。

フラッシュローダパス

このテキストボックスを使用して、ボード設定で使用するフラッシュローダファイル (*.flash) のパスを指定します。

追加パラメータ

フラッシュローダによっては、特別なオプションセットを独自に定義します。このテキストボックスを使用して、フラッシュローダを制御するオプションを指定します。使用可能なフラッシュロードについては、**[パラメータ説明]** フィールドを参照してください。

パラメータ説明

[パラメータ説明] フィールドには、**[追加パラメータ]** テキストボックスで指定された追加パラメータの説明が表示されます。

A

Angel (デバッグオプション)	365
Angel インタフェース、指定	52
Angel デバッグモニタ (C-SPY ドライバ)	52
ARM モードでの逆アセンブル ([逆アセンブリ] メニュー)	73

B

--backend (C-SPY コマンドラインオプション)	338
--BE32 (C-SPY コマンドラインオプション)	334
--BE8 (C-SPY コマンドラインオプション)	334

C

__cancelAllInterrupts (C-SPY システムマクロ)	295
__cancelInterrupt (C-SPY システムマクロ)	295
__clearBreak (C-SPY システムマクロ)	295
__closeFile (C-SPY システムマクロ)	296
--code_coverage_file (C-SPY コマンドラ インオプション)	338
CPI (生成の設定)	189
--cpu (C-SPY コマンドラインオプション)	334
CPU クロック (SWO 設定オプション)	192
CPU クロック (SWO 設定)	378, 387
cspybat	331
--cycles (C-SPY コマンドラインオプション)	339
C 関数情報、C-SPY	84
C 規格、C-SPY の sizeof 演算子	100
C 言語のシンボル、C-SPY 式で使用	98
C 言語の変数、C-SPY 式で使用	98
C-SPY	
デバッグシステム、概要	35
デバッグの起動	60
ドライバ間の差異	38
バッチモード、使用	331
プラグインモジュール、ロード	60
環境の概要	32

設定	58-59
C-SPYLink	37
C-SPY オプション	363
イメージ	369
プラグイン	370
設定	365
追加オプション	368
C-SPY ドライバ	
Angel デバッグモニタ	52
GDB サーバ	45
J-Link	39
Macraigor	44
P&E Microcomputer システム (C-SPY ドライバ)	38
RDI	42
ROM モニタ	54
ST-LINK	47
TI Stellaris FTDI	50
概要	37
指定	365
C-SPY マクロ	
C-SPY 式	288
システムマクロ、まとめ	292
セットアップマクロファイル	278
実行	282
セットアップマクロ関数	278
概要	291
ダイアログボックス、使用	281
ブロック	289
マクロ文	288
ループ文	288
関数	99, 286
使用	277
実行	280
セットアップマクロとセットアップファ イルの使用	282
ブレークポイントに接続	284
[クイックウォッチ] を使用	283
条件文	288
変数	99, 286

例.....	279
execUserPreload、使用.....	65
WDT のステータスのチェック.....	283
ダウンロード前のメモリの再配置.....	65
レジスタのステータスのチェック.....	283
ログマクロの作成.....	284
C-SPY マクロ " __message" style	
style (ログブレイクポイントのオプション).....	142
C-SPY 式.....	98
C-SPY マクロ.....	288
ツールチップウォッチ、使用.....	97
評価.....	112
[ウォッチ] ウィンドウ、使用.....	97
C++ 例外	
ステップ実行.....	80
デバッグ.....	72
C++ 例外 > スロー時にブレーク ([デバッグ]	
メニュー).....	72
C++ 例外 > 例外が取得されなかったときにブレーク	
([デバッグ] メニュー).....	72
C++ 用語.....	29

D

DCC (デバッグ通信チャンネル).....	396
ddf (ファイル名の拡張子)、ファイルの選択.....	60
__delay (C-SPY システムマクロ).....	296
--device (C-SPY コマンドラインオプション).....	339
__disableInterrupts (C-SPY システムマクロ).....	296
--disable_interrupts (C-SPY コマンドラ	
インオプション).....	339
DLIB、ドキュメント.....	27
do (マクロ文).....	288
--download_only (C-SPY コマンドラ	
インオプション).....	340
__driverType (C-SPY システムマクロ).....	297
--drv_attach_to_program (C-SPY コマンドラ	
インオプション).....	334
--drv_catch_exceptions (C-SPY コマンドラ	
インオプション).....	340

--drv_communication (C-SPY コマンドラ	
インオプション).....	341
--drv_communication_log (C-SPY コマンドラ	
インオプション).....	344
--drv_default_breakpoint (C-SPY コマンドラ	
インオプション).....	344
--drv_reset_to_cpu_start (C-SPY コマンドラ	
インオプション).....	345
--drv_restore_breakpoints (C-SPY コマンドラ	
インオプション).....	345
--drv_suppress_download (C-SPY コマンドラ	
インオプション).....	334
--drv_swo_clock_setup (C-SPY コマンドラ	
インオプション).....	346
--drv_vector_table_base (C-SPY コマンドラ	
インオプション).....	346
--drv_verify_download (C-SPY コマンドラ	
インオプション).....	335

E

Embedded C++ Technical Committee.....	28
EmbeddedICE マクロセル.....	122
__emulatorSpeed (C-SPY システムマクロ).....	297
__emulatorStatusCheckOnRead	
(C-SPY システムマクロ).....	298
__enableInterrupts (C-SPY システムマクロ).....	299
--endian (C-SPY コマンドラインオプション).....	335
ETM トレース ([J-Link] メニュー).....	395
ETM トレースの保存 ([J-Link] メニュー).....	395
ETM トレース設定 ([J-Link] メニュー).....	395
__evaluate (C-SPY システムマクロ).....	299
EXC (生成の設定).....	189
execUserExit (C-SPY セットアップマクロ).....	291
execUserFlashExit (C-SPY セットアップマクロ).....	291
execUserFlashInit (C-SPY セットアップマクロ).....	291
execUserFlashReset (C-SPY セットアップマクロ).....	291
execUserPreload (C-SPY セットアップマクロ).....	291
execUserPreReset (C-SPY セットアップマクロ).....	291
execUserReset (C-SPY セットアップマクロ).....	291

execUserSetup (C-SPY セットアップマクロ) 291

F

FIFO フルで停止 (ETM トレース
設定オプション) 187
--flash_loader (C-SPY コマンドラ
インオプション) 347
FOLD (生成の設定) 189
for (マクロ文) 288
--fpu (C-SPY コマンドラインオプション) 335

G

GDB サーバ (デバッガオプション) 365
__gdbserver_exec_command
(C-SPY システムマクロ) 300
--gdbserv_exec_command
(C-SPY コマンドラインオプション) 348
GDB サーバ (C-SPY ドライバ) 45
メニュー 393
Go ([デバッグ] メニュー) 71, 83

H

__hwReset (C-SPY システムマクロ) 300
__hwResetRunToBp (C-SPY システムマクロ) 301
__hwResetWithStrategy (C-SPY システムマクロ) 302

I

IAR ROM モニタ (デバッガオプション) 365
IAR デバッガのドライバプラグイン
(デバッガのオプション) 388
if else (マクロ文) 288
if (マクロ文) 288
Intel-extended、C-SPY 出力フォーマット 37
__isBatchMode (C-SPY システムマクロ) 302
ITM 事象ポート (SWO 設定オプション) 193

J

__jlinkExecCommand (C-SPY システムマクロ) 303
--jlink_device_select (C-SPY コマンドラ
インオプション) 348
--jlink_exec_commmmand (C-SPY コマンドラ
インオプション) 348
--jlink_initial_speed (C-SPY コマンドラ
インオプション) 349
--jlink_interface (C-SPY コマンドラ
インオプション) 349
--jlink_ir_length (C-SPY コマンドラ
インオプション) 350
--jlink_reset_strategy (C-SPY コマンドラ
インオプション) 350
--jlink_script_file (C-SPY コマンドラ
インオプション) 351
--jlink_speed (C-SPY コマンドラインオプション) .. 351
JTAG スキャンチェーン
(J-Link/J-Trace オプション) 380
JTAG スキャンチェーン (マルチターゲット)
(Macraigor オプション) 384
JTAG スキャンチェーン (マルチターゲット)
(JTAG スキャンチェーンの設定) 380
JTAG (インタフェース設定) 380-382, 387
__jtagCommand (C-SPY システムマクロ) 303
__jtagCP15IsPresent (C-SPY システムマクロ) 304
__jtagCP15ReadReg (C-SPY システムマクロ) 304
__jtagCP15WriteReg (C-SPY システムマクロ) 304
__jtagData (C-SPY システムマクロ) 305
__jtagRawRead (C-SPY システムマクロ) 305
__jtagRawSync (C-SPY システムマクロ) 306
__jtagRawWrite (C-SPY システムマクロ) 307
__jtagResetTRST (C-SPY システムマクロ) 308
JTAG インタフェース、J-Link 374, 379
JTAG ウォッチポイント、概要 122
JTAG 速度 (Macraigor オプション) 383
JTAG/SWD 速度 (J-Link/J-Trace オプション) 377
JTAG/SWD 速度 (TI Stellaris FTDI オプション) 381
J-Link JTAG インタフェース 374, 379

J-Link (C-SPY ドライバ).....	39
メニュー.....	394
J-Link 通信上の問題.....	378
J-Link/J-Trace (デバッグオプション).....	365

L

lightbulb アイコン、本ガイドの.....	29
--lmiftdi_speed (C-SPY コマンドラ インオプション).....	352
__loadImage (C-SPY システムマクロ).....	308
LSU (生成の設定).....	189

M

mac (ファイル名の拡張子)、マクロファイルの 使用.....	59
Macraigor (C-SPY ドライバ).....	44
メニュー.....	398
Macraigor (デバッグオプション).....	365
--macro (C-SPY コマンドラインオプション).....	355
--mac_handler_address (C-SPY コマンドラ インオプション).....	352
--mac_interface (C-SPY コマンドラ インオプション).....	353
--mac_jtag_device (C-SPY コマンドラ インオプション).....	353
--mac_multiple_targets (C-SPY コマンドラ インオプション).....	353
--mac_reset_pulls_reset (C-SPY コマンドラ インオプション).....	354
--mac_set_temp_reg_buffer (C-SPY コマンドラインオプション).....	355
--mac_speed (C-SPY コマンドラ インオプション).....	355
--mac_xscale_ir7 (C-SPY コマンドラ インオプション).....	356
main 関数、C-SPY 起動時に実行.....	58, 366
Manufacturer RDI driver (RDI オプション).....	385
--mapu (C-SPY コマンドラインオプション).....	356

__memoryRestore (C-SPY システムマクロ).....	309
__memorySave (C-SPY システムマクロ).....	310
MISRA-C、ドキュメント.....	28
Motorola、C-SPY 出力フォーマット.....	37
Multi-ICE インタフェース RealView Multi-ICE インタ フェースを参照	

O

OCD インタフェースデバイス (Macraigor オプション).....	382
OP フェッチ (アクセスタイプの設定).....	139
__openFile (C-SPY システムマクロ).....	311
__orderInterrupt (C-SPY システムマクロ).....	312

P

-p (C-SPY コマンドラインオプション).....	357
PC + データ値 + ベースアドレス (データログイベントの設定).....	191
PC サンプリング (SWO 設定オプション).....	191
PC サンプル (強制設定).....	189
PC のみ (データログイベントの設定).....	191
PC へ移動 ([逆アセンブリ] ウィンドウの コンテキストメニュー).....	88
--plugin (C-SPY コマンドラインオプション).....	357
__popSimulatorInterruptExecutingStack (C-SPY システムマクロ).....	313
Power サンプリング.....	224
Power ログ ([J-Link] メニュー).....	396
Power ログ ([タイムライン] ウィンドウの コンテキストメニュー).....	205
Power ログ設定 ([J-Link] メニュー).....	396
--proc_stack_xxx (C-SPY コマンドラ インオプション).....	357

R

RDI とのコミュニケーションのログ (RDI オプション).....	386
--	-----

RDI (C-SPY ドライバ)	42
メニュー	399
RDI (デバuggアオプション)	365
--rdi_allow_hardware_reset (C-SPY コマンドラインオプション)	358
--rdi_driver_dll (C-SPY コマンドラ インオプション)	358
--rdi_heartbeat (C-SPY コマンドラ インオプション)	336
--rdi_step_max_one (C-SPY コマンドラ インオプション)	359
__readFile (C-SPY システムマクロ)	313
__readFileByte (C-SPY システムマクロ)	314
__readMemoryByte (C-SPY システムマクロ)	314
__readMemory8 (C-SPY システムマクロ)	314
__readMemory16 (C-SPY システムマクロ)	315
__readMemory32 (C-SPY システムマクロ)	315
RealView Multi-ICE インタフェース	385
__registerMacroFile (C-SPY システムマクロ)	315
__resetFile (C-SPY システムマクロ)	316
__restoreSoftwareBreakpoints (C-SPY システムマクロ)	316
return (マクロ文)	288
ROM モニタ (C-SPY ドライバ)	54
ROM モニタプロトコル、Angel	52
ROM モニタ、定義	36
RTOS 認識デバugg	34
RTOS 認識 (C-SPY プラグインモジュール)	34
R/W (アクセスタイプの設定)	139

S

--semihosting (C-SPY コマンドラ インオプション)	359
__setCodeBreak (C-SPY システムマクロ)	317
__setDataBreak (C-SPY システムマクロ)	318
__setLogBreak (C-SPY システムマクロ)	319
__setSimBreak (C-SPY システムマクロ)	321
__setTraceStartBreak (C-SPY システムマクロ)	322
__setTraceStopBreak (C-SPY システムマクロ)	323

SFR	
アセンブラシンボルとして使用	99
[レジスタ] ウィンドウ内	172
--silent (C-SPY コマンドラインオプション)	360
sizeof	100
SLEEP (生成の設定)	189
__sourcePosition (C-SPY システムマクロ)	324
stack.mac	278
stdin と stdout、C-SPY ウィンドウにリダイレクト	92
--stlink_interface (C-SPY コマンドラ インオプション)	360
--stlink_reset_strategy (C-SPY コマンドラ インオプション)	361
__strFind (C-SPY システムマクロ)	324
ST-LINK (C-SPY ドライバ)	47
メニュー	400
ST-LINK (デバuggアオプション)	365
__subString (C-SPY システムマクロ)	325
SWD (インタフェース設定)	380–382, 387
SWD インタフェース、 [トレース] ウィンドウの情報	180
SWO クロック (SWO 設定オプション)	192
SWO トレース ([J-Link] メニュー)	395
SWO トレース ([ST-LINK] メニュー)	400
SWO トレースウィンドウの設定 ([J-Link] メニュー)	395
SWO トレースウィンドウの設定 ([ST-LINK] メニュー)	400
SWO トレースにおけるタイムスタンプ	189
SWO トレースの保存 ([J-Link] メニュー)	395
SWO トレースの保存 ([ST-LINK] メニュー)	400
SWO 設定 ([J-Link] メニュー)	395
SWO 設定 ([ST-LINK] メニュー)	400
SWO 通信チャンネル トレースのタイムスタンプ	189
有効	380–382
SWO (J-Link/J-Trace オプション)	378

T

TAP 番号 (JTAG スキャンチェーンの設定)	380
__targetDebuggerVersion (C-SPY システムマクロ) ..	325
TCP/IP Macraigor オプション	383
TCP/IP (Angel オプション)	371
TCP/IP (通信設定)	379
TCP/IP アドレスまたはホスト名 (C-SPY サーバオプション)	372
Thumb モードでの逆アセンブル ([逆アセンブリ] メニュー)	73
TI Stellaris FTDI (C-SPY ドライバ)	50
メニュー	397
TI Stellaris FTDI (デバッグオプション)	365
--timeout (C-SPY コマンドラインオプション)	361
__toLower (C-SPY システムマクロ)	326
__toString (C-SPY システムマクロ)	326
__toUpper (C-SPY システムマクロ)	327

U

__unloadImage (C-SPY システムマクロ)	327
USB (通信設定)	379

V

--verify_download (C-SPY コマンドラ インオプション)	362
visualSTATE、C-SPY プラグインモジュール	37

W

Web サイト、推奨	28
while (マクロ文)	288
__writeFile (C-SPY システムマクロ)	328
__writeFileByte (C-SPY システムマクロ)	328
__writeMemoryByte (C-SPY システムマクロ)	329
__writeMemory8 (C-SPY システムマクロ)	329

__writeMemory16 (C-SPY システムマクロ)	329
__writeMemory32 (C-SPY システムマクロ)	330

あ

アクション (イミディエイトブレークポイントの オプション)	149
アクション (コードブレークポイントの オプション)	137
アクセスタイプ (イミディエイトブレークポ イントのオプション)	149
アクセスタイプ (データブレークポイントの オプション)	139, 144, 146
アクセスタイプ (トレースフィルタオ プション)	217
アクセスタイプ (メモリアクセスの 編集オプション)	175
アクセスタイプ ([トレース開始] オプション) ..	211
アクセスタイプ ([トレース停止] オプション) ..	214
アセンブラシンボル、C-SPY 式で使用	99
アセンブラのソースコード、微調整	223
アセンブララベル、表示	101-102
アセンブラ変数、表示	101-102
アドレス (JTAG ウォッチポイントの オプション)	139
アドレスバスパターン (アドレス設定)	139
アドレス範囲 (トレース検索オプション)	220
アプリケーション、IDE の外部でビルド	61

い

イミディエイトブレークポイント、概要	121
イメージ、複数のロード	369
インストール先ディレクトリ	29
インタフェース (J-Link/J-Trace オプション)	380
インタフェース (Macraigor オプション)	382
インタフェース (ST-LINK オプション)	387
インタフェース (TI Stellaris FTDI オプション) ..	381
インデックス範囲 (トレース保存オプション)	198

う

ウィンドウの内容のコピー ([逆アセンブリ] ウィンドウのコンテキストメニュー).....	90
ウィンドウ、C-SPY 専用	74
ウォッチポイント (J-Link メニュー).....	394
ウォッチポイント (Macraigor の [JTAG] メニュー)	398

お

オプション	
IDE	363
コマンドライン	338, 368
オプション ([スタック] ウィンドウの コンテキストメニュー).....	170
オフセットを表示 ([スタック] ウィンドウの コンテキストメニュー).....	170
オートスクロール ([タイムライン] ウィンドウの コンテキストメニュー).....	204

か

ガイドラインの確認	25
カーソルまで実行 ([逆アセンブリ] ウィンドウの コンテキストメニュー).....	88
カーソルまで実行 ([呼出しスタック] ウィンドウのコンテキストメニュー).....	91
カーソルまで実行、実行コマンド.....	83
カーソル位置まで実行 ([デバッグ] メニュー)	72
カーソル、C-SPY の [逆アセンブリ] ウィンドウ...	87

く

グラフを選択 ([タイムライン] ウィンドウの コンテキストメニュー).....	205
クリア ([Power ログ] ウィンドウの コンテキストメニュー).....	248
クリア ([データログ] ウィンドウの コンテキストメニュー).....	274

クロックに同期 (A) (JTAG/SWD 速度設定).....	378
クロック設定 (ST-LINK オプション)	387

こ

コア (リセット設定).....	374
コアとペリフェラル (リセット設定)	374
このガイドで使用されている規則.....	29
コピー ([デバッグログ] ウィンドウの コンテキストメニュー).....	94
コマンドプロンプトアイコン、本ガイド.....	29
コマンドラインオプション.....	338
表記規則.....	29
コマンドラインオプションの使用 (デバッグのオプション).....	369
コンテキストメニュー、ウィンドウ.....	102
コンピュータスタイル、表記規則.....	29
コードカバレッジ ([逆アセンブリ] ウィンドウのコンテキストメニュー).....	88
コードブレイクポイント、概要.....	120
コード、実行のカバレッジ.....	252

さ

サイクルアキュレート・トレース (ETM トレース設定オプション)	187
サイクル表示 ([Power ログ] ウィンドウの コンテキストメニュー).....	249
サイクル表示 ([データログ] ウィンドウの コンテキストメニュー).....	274
サイズ (コードブレイクポイントのオプション) ..	137
サイズ (データブレイクポイントのオプション) ..	146
サイズ (トレースフィルタオプション)	216
サイズ (トレース開始オプション)	210
サイズ (トレース停止オプション)	213
サイズ ([タイムライン] ウィンドウの コンテキストメニュー).....	205
サンプリング、ソースのプロファイリング	224, 231
サードパーティ製ドライバ (デバッグオプション)	388

し

シミュレータ (デバッグオプション)	365
シミュレータドライバ、選択.....	39
シミュレータ、概要	39
シリアルポート (Angel オプション).....	372
シリアルポート設定 (IAR ROM モニタオプション).....	373
シンボルを 1 つ選択してください (シンボルの曖昧さの解決オプション).....	114
シンボル、C-SPY 式で使用	98

す

スキャンチェーンに非 ARM デバイスが 含まれています (JTAG スキャンチェーンの設定) ..	380
スケール (表示範囲オプション)	207
スタックの使用、計算	158
ステップアウト ([デバッグ] メニュー)	71
ステップアウト、説明	82
ステップイン ([デバッグ] メニュー)	71
ステップイン、説明	81
ステップオーバー ([デバッグ] メニュー)	71
ステップオーバー、説明	81
ステップポイント、定義.....	80
ステップ実行中に割込みを無効化 ([J-Link] メニュー).....	395
すべてのイメージの表示 ([イメージ] ウィンドウのコンテキストメニュー).....	76
すべてベリファイ (デバッグオプション)	389
すべてをクリア ([デバッグログ] ウィンドウの コンテキストメニュー).....	94
すべて選択 ([デバッグログ] ウィンドウの コンテキストメニュー).....	94
すべて無効 ([ブレークポイント] ウィンドウの コンテキストメニュー).....	134
すべて有効 ([ブレークポイント] ウィンドウの コンテキストメニュー).....	134

ズーム ([タイムライン] ウィンドウの コンテキストメニュー).....	204
--	-----

せ

セットアップマクロ (デバッグオプション)	366
セットアップマクロ関数.....	278
予約済みの名前.....	291
セミホスティング, SWI (オプション)、使用	92

そ

ソフトウェア (デフォルトのブレークポ イントタイプ設定)	148
ソフトウェア (リセット設定)	376
ソフトウェアブレークポイント復元位置 (ブレークポイントのオプション).....	148
ソフトウェア遅延、電力消費.....	237
ソフトウェア、Analog デバイス (リセット設定) ..	376
ソースのプロファイリング サンプリング.....	224, 231
トレース (フラット).....	224, 231
トレース (呼出し)	224, 231
ブレークポイント.....	224, 231
ソースの切替え (トレースツールバー)	194
ソースへ移動 ([タイムライン] ウィンドウの コンテキストメニュー).....	205
ソースへ移動 ([ブレークポイント] ウィンドウの コンテキストメニュー).....	134
ソースへ移動 ([呼出しスタック] ウィンドウの コンテキストメニュー).....	91
ゾーン (フィルオプション)	165
ゾーン (メモリアクセスの編集オプション)	175
ゾーン (メモリ復元オプション)	164
ゾーン (メモリ保存オプション)	163
ゾーン ([メモリ] ウィンドウのコンテキストメ ニュー)	161
ゾーン、C-SPY.....	156

た

タイム割込み、例	261
タイムスタンプ (SWO 設定オプション)	192
タイムスタンプ (強制設定)	189
タイムスタンプを表示 (ETM トレース 設定オプション)	187
タイムライン ([J-Link] メニュー)	396
タイムライン ([ST-LINK] メニュー)	400
タイムライン ([シミュレータ] メニュー)	393
ダウンロードを中止する (デバッグオプション)	367, 388
タブ区切りフォーマットを使用 (トレース保存オプション)	198
ターゲットシステム、定義	36
ターミナル IO ログファイル (ターミナル IO ログファイルのオプション)	93

ち

チェーン (ブレイク条件の設定)	141
------------------	-----

つ

ツールアイコン、本ガイド	29
--------------	----

て

テキスト検索 (トレース検索オプション)	220
デバイス記述ファイル	60
割込みの指定	312
修正	63
定義	63
デバイス記述ファイル (デバッグオプション)	366
デバイス待機、電力消費時	237
デバッグシステムの概要	35
デバッグドライバ	
Angel デバッグモニタ	52
GDB サーバ	45

J-Link	39
Macraigor	44
P&E Microcomputer システム	38
RDI	42
ROM モニタ	54
ST-LINK	47
TI Stellaris FTDI	50
デバッグの概念	35
デバッグの停止 ([デバッグ] メニュー)	71
デバッグハンドラアドレス (Macraigor オプション)	384
デバッグメニュー (C-SPY メインウィンドウ)	71
デバッグ、RTOS 認識	34
デフォルトのブレイクポイントタイプ (ブレイクポイントオプション)	147
デフォルトの .board ファイルのオーバーライド (デバッグオプション)	368
データ (JTAG ウォッチポイントのオプション)	139
データカバレッジ ([メモリ] ウィンドウの コンテキストメニュー)	162
データカバレッジ、[メモリ] ウィンドウ内	160
データバスパターン (データ設定)	140
データブレイクポイントの設定 ([メモリ] ウィンドウのコンテキストメニュー)	162
データブレイクポイント、概要	121
データログ ([J-Link] メニュー)	395
データログ ([ST-LINK] メニュー)	400
データログ ([タイムライン] ウィンドウの コンテキストメニュー)	205
データログイベント (SWO 設定オプション)	191
データログブレイクポイント、概要	122
データログ概要 ([J-Link] メニュー)	395
データログ概要 ([ST-LINK] メニュー)	400
データ照合 (データブレイクポイントの オプション)	144
データ照合 (トレースフィルタオプション)	217
データ照合 (トレース開始オプション)	212
データ照合 (トレース停止オプション)	215
データ値 + 正確なアドレス (データログイベントの設定)	191

と

ドキュメント	
ガイドの概要	27
本ガイド	25
本ガイドの概要	26
ドライバ (デバッガオプション)	365
トリガ ([強制割込み] ウィンドウの コンテキストメニュー)	268
トリガ位置 (データログブレイクポイントの オプション)	145
トリガ位置 (トレースフィルタオプション)	216
トリガ位置 ([トレース開始] オプション)	210
トリガ位置 ([トレース停止] オプション)	213
トリガ範囲 (データブレイクポイントの オプション)	144, 146
トリガ範囲 (トレースフィルタオプション)	217
トリガ範囲 ([トレース開始] オプション)	211
トリガ範囲 ([トレース停止] オプション)	214
トレース (フラット)、ソースの プロファイリング	224, 231
トレース (呼出し)、ソースの プロファイリング	224, 231
トレース ([シミュレータ] メニュー)	392
トレースデータのクリア (トレースツールバー)	194
トレースの設定ダイアログボックス	186
トレースの保存 ([RDI] メニュー)	399
トレースバッファサイズ (トレース設定オプション)	187
トレースポートの幅 (トレース設定オプション)	186
トレースポートモード (トレース設定オプション)	186
トレース開始および停止ブレイクポイント、概要	121
トレース設定 ([RDI] メニュー)	399
トレース、[タイムライン] ウィンドウ	200

の

ノーマル (リセット設定)	374, 387
ノーマル、ウォッチドッグの無効化 (リセット設定)	375

は

バイト (データの設定)	139
バックトレース情報	
コンパイラが生成	84
[呼出しスタック] ウィンドウでの表示	90
バッチモード、C-SPY を使用	331
ばらつき (割込みプロパティ)、定義	257
ばらつき % (割込み編集オプション)	267
パラメータ	
フラッシュローダに渡されるリスト	406
表記規則	29
不正な値のトレース	84
バージョン番号、本ガイド	2
ハードウェア (デフォルトのブレイクポ イントタイプ設定)	148
ハードウェアリセット (Macraigor オプション)	383
ハードウェアリセットを許可 (RDI オプション)	385
ハードウェア設定、電力消費	241
ハードウェア、Atmel AT91SAM7 (リセット設定)	376
ハードウェア、DBGCRQ を使用して停止 (リセット設定)	376
ハードウェア、NXP LPC (リセット設定)	376
ハードウェア、ブレイクポイントを使用して停止 (1) (リセット設定)	376
ハードウェア、停止までの遅延時間を指定 (ms) (リセット設定)	375
ハードウェア、0 で停止 (リセット設定)	376
ハートビート送信 (Angel オプション)	371
ハーフワード (データの設定)	139

ひ

ビッグエンディアン ([メモリ] ウィンドウの コンテキストメニュー)	161
--	-----

ふ

ファイル (トレース保存オプション)	198
ファイルタイプ	
macro	59
デバイス記述、IDE で指定	60
ファイルタイプ、マクロ	366
ファイルに追加 (トレース保存オプション)	198
ファイルフォーマット (メモリ保存オプション) ..	163
ファイル名 (メモリ復元オプション)	164
ファイル名 (メモリ保存オプション)	163
ファイル名の拡張子	
ddf、デバイス記述ファイルの選択	60
mac、マクロファイルの使用	59
ブラウズ (トレースツールバー)	195
プラグインモジュール (C-SPY)	37
ロード	60
プラグイン (C-SPY オプション)	370
フラッシュメモリ、ライブラリモジュールの	
ロード	309
フラッシュローダ	
パスの指定	408
使用	403
制御するパラメータ	408
フラッシュローダを使用する	
(デバッグオプション)	368
ブレーク ([デバッグ] メニュー)	71
ブレークポイント	
C-SPY マクロに接続	284
IDE のアイコン	123
コード、例	317
すべてのリスト表示	135
ソースのプロファイリング	224, 231
データ	143, 145
例	319
トグル	127
トレース開始、例	323
トレース停止、例	324
ヒント	130

ログ、例	320
使用の理由	119
種類	120
設定	
システムマクロの使用	129
ダイアログボックスを使用	127
[メモリ] ウィンドウで	128
設定されていない場合のステップ実行	59
設定元	124
説明	120
[スタック] ウィンドウで使用するブレークポ	
イントの無効化	125
[メモリ] ウィンドウで	128
ブレークポイントオプション (C-SPY オプション) ..	147
ブレークポイントの使用 (Macraigor の [JTAG]	
メニュー)	398
ブレークポイントの使用 ([J-Link] メニュー)	396
ブレークポイントの使用 ([ST-LINK] メニュー) ..	401
ブレークポイントの使用 ([RDI] メニュー)	399
ブレークポイントの使用 ([シミュレータ]	
メニュー)	393
ブレークポイントの種類	
(コードブレークポイントのオプション)	137
ブレークポイントの切り替え (コード)	
([逆アセンブリ] ウィンドウの	
コンテキストメニュー)	89
ブレークポイントの切り替え (コード)	
([呼出しスタック] ウィンドウの	
コンテキストメニュー)	91
ブレークポイントの切り替え (トレース開始)	
([逆アセンブリ] ウィンドウの	
コンテキストメニュー)	89
ブレークポイントの切り替え (トレース停止)	
([逆アセンブリ] ウィンドウの	
コンテキストメニュー)	89
ブレークポイントの切り替え (ログ)	
([逆アセンブリ] ウィンドウの	
コンテキストメニュー)	89
ブレークポイントの切り替え (ログ)	
([呼出しスタック] ウィンドウの	
コンテキストメニュー)	91

ブレイクポイント条件、例.....	130-131
ブレイク位置（イミディエイトブレイクポイントのオプション）.....	149
ブレイク位置（コードブレイクポイントのオプション）.....	136
ブレイク位置（データブレイクポイントのオプション）.....	143
ブレイク位置（ログブレイクポイントのオプション）.....	142
ブレイク条件（JTAG ウォッチポイントのオプション）.....	141
プログラミング経験.....	25
プログラムの実行、C-SPY.....	79
プロジェクトデフォルトのオーバーライド（SWO 設定オプション）.....	192
プロジェクトのデバッグ	
外部でビルドされたアプリケーション.....	61
複数イメージのロード.....	62
プロジェクト、外部でビルドされたアプリケーションのデバッグ.....	61
ブロック、C-SPY マクロ.....	289
プロファイリング	
関数レベル.....	226
命令レベル.....	226
プロファイリング情報、関数と命令.....	223
プロファイル選択（[タイムライン] ウィンドウのコンテキストメニュー）.....	206
ブロードキャスト・オールブランチ（ETM トレース設定オプション）.....	187
ブートローダ後に停止（リセット設定）.....	375
ブートローダ前に停止（リセット設定）.....	375



ベクタキャッチ（Macraigor の [JTAG] メニュー）.....	398
ベリファイする（デバッグオプション）.....	367

ほ

ポート（Macraigor オプション）.....	383
ポート（シリアルポートの設定オプション）.....	372-373
ポートの有効化（ITM 事象ポートの設定）.....	193
ボーレート（Macraigor オプション）.....	383
ボーレート（シリアルポートの設定オプション）.....	372-373

ま

マクロ	
使用.....	277
実行.....	280
マクロ（[デバッグ] メニュー）.....	73
マクロファイル、指定.....	59, 366
マクロ文.....	288
マスク（アドレス設定）.....	139
マスク（データ照合の設定）.....	144, 212, 215, 217
マスク（データ設定）.....	140

め

メッセージ（ログブレイクポイントのオプション）.....	142
メニューバー、C-SPY 専用.....	70
メモリアクセスチェック.....	158
メモリアクセスチェック（メモリアクセスの設定オプション）.....	174
メモリアクセス設定（[シミュレータ] メニュー）.....	392
メモリアクセス、不正な.....	158
メモリゾーン.....	156
メモリフィル（[メモリ] ウィンドウのコンテキストメニュー）.....	162
メモリマップ.....	173
メモリ範囲（メモリアクセスの編集オプション）.....	175
メモリ復元（[メモリ] ウィンドウのコンテキストメニュー）.....	162

メモリ保存 ([メモリ] ウィンドウの コンテキストメニュー).....	162
メモリ>復元 ([デバッグ] メニュー)	72
メモリ>保存 ([デバッグ] メニュー)	72

も

モード (JTAG ウォッチポイントのオプション) ...	140
-------------------------------	-----

ゆ

ユーザ (モードの設定)	140
ユーザアプリケーション、定義.....	36

ら

ライト (アクセスタイプの設定)	139, 146
ラベル (アセンブラ)、表示.....	101-102

り

リセット (J-Link/J-Trace オプション)	374, 387
リセット ([デバッグ] メニュー)	71
リセットピン (リセット設定)	374, 387
リセット中の設定 (リセット設定)	374, 387
リトルエンディアン ([メモリ] ウィンドウの コンテキストメニュー).....	161
リンク条件 (トレースフィルタオプション)	218
リンク条件 ([トレース開始] オプション)	212
リンク条件 ([トレース停止] オプション)	215
リード (アクセスタイプの設定)	139, 146

る

ループ文、C-SPY マクロ	288
----------------------	-----

れ

レジスタグループ	156
定義済み、有効化	171
レジスタ、[レジスタ] ウィンドウに表示	171
レート (PC サンプリング設定)	191

ろ

ログの有効化 (ログファイルのオプション)	95
ログファイルを保存 ([Power ログ] ウィンドウの コンテキストメニュー).....	248
ログファイルを保存 ([データログ] ウィンドウの コンテキストメニュー).....	274
ログブレイクポイント、概要.....	121
ログ>ターミナル I/O ログファイルの設定 ([デバッグ] メニュー).....	73
ログ>ログファイルの設定 ([デバッグ] メニュー)	73
ロードするプラグインの選択 (デバッグの オプション)	370

わ

ワード (データの設定)	139
--------------------	-----

記号

__cancelAllInterrupts (C-SPY システムマクロ)	295
__cancelInterrupt (C-SPY システムマクロ)	295
__clearBreak (C-SPY システムマクロ)	295
__closeFile (C-SPY システムマクロ)	296
__delay (C-SPY システムマクロ)	296
__disableInterrupts (C-SPY システムマクロ)	296
__driverType (C-SPY システムマクロ)	297
__emulatorSpeed (C-SPY システムマクロ)	297
__emulatorStatusCheckOnRead (C-SPY システムマクロ)	298
__enableInterrupts (C-SPY システムマクロ)	299

__evaluate (C-SPY システムマクロ).....	299	__setTraceStartBreak (C-SPY システムマクロ).....	322
__fmessage (C-SPY マクロ文).....	289	__setTraceStopBreak (C-SPY システムマクロ).....	323
__gdbserver_exec_command (C-SPY システムマクロ).....	300	__smessage (C-SPY マクロ文).....	289
__hwReset (C-SPY システムマクロ).....	300	__sourcePosition (C-SPY システムマクロ).....	324
__hwResetRunToBp (C-SPY システムマクロ).....	301	__strFind (C-SPY システムマクロ).....	324
__hwResetWithStrategy (C-SPY システムマクロ)...	302	__subString (C-SPY システムマクロ).....	325
__isBatchMode (C-SPY システムマクロ).....	302	__targetDebuggerVersion (C-SPY システムマクロ) ..	325
__jlinkExecCommand (C-SPY システムマクロ)....	303	__toLower (C-SPY システムマクロ).....	326
__jtagCommand (C-SPY システムマクロ).....	303	__toString (C-SPY システムマクロ).....	326
__jtagCP15IsPresent (C-SPY システムマクロ)....	304	__toUpper (C-SPY システムマクロ).....	327
__jtagCP15ReadReg (C-SPY システムマクロ).....	304	__unloadImage (C-SPY システムマクロ).....	327
__jtagCP15WriteReg (C-SPY システムマクロ)....	304	__writeFile (C-SPY システムマクロ).....	328
__jtagData (C-SPY システムマクロ).....	305	__writeFileByte (C-SPY システムマクロ).....	328
__jtagRawRead (C-SPY システムマクロ).....	305	__writeMemoryByte (C-SPY システムマクロ).....	329
__jtagRawSync (C-SPY システムマクロ).....	306	__writeMemory8 (C-SPY システムマクロ).....	329
__jtagRawWrite (C-SPY システムマクロ).....	307	__writeMemory16 (C-SPY システムマクロ).....	329
__jtagResetTRST (C-SPY システムマクロ).....	308	__writeMemory32 (C-SPY システムマクロ).....	330
__loadImage (C-SPY システムマクロ).....	308	-p (C-SPY コマンドラインオプション).....	357
__memoryRestore (C-SPY システムマクロ).....	309	--backend (C-SPY コマンドラインオプション)....	338
__memorySave (C-SPY システムマクロ).....	310	--BE32 (C-SPY コマンドラインオプション).....	334
__message (C-SPY マクロ文).....	289	--BE8 (C-SPY コマンドラインオプション).....	334
__openFile (C-SPY システムマクロ).....	311	--code_coverage_file (C-SPY コマンドラ インオプション).....	338
__orderInterrupt (C-SPY システムマクロ).....	312	--cpu (C-SPY コマンドラインオプション).....	334
__popSimulatorInterruptExecutingStack (C-SPY システムマクロ).....	313	--cycles (C-SPY コマンドラインオプション).....	339
__readFile (C-SPY システムマクロ).....	313	--device (C-SPY コマンドラインオプション).....	339
__readFileByte (C-SPY システムマクロ).....	314	--disable_interrupts (C-SPY コマンドラ インオプション).....	339
__readMemoryByte (C-SPY システムマクロ).....	314	--download_only (C-SPY コマンドラ インオプション).....	340
__readMemory8 (C-SPY システムマクロ).....	314	--drv_attach_to_program (C-SPY コマンドラインオプション).....	334
__readMemory16 (C-SPY システムマクロ).....	315	--drv_catch_exceptions (C-SPY コマンドラ インオプション).....	340
__readMemory32 (C-SPY システムマクロ).....	315	--drv_communication (C-SPY コマンドラ インオプション).....	341
__registerMacroFile (C-SPY システムマクロ)....	315	--drv_communication_log (C-SPY コマンドラインオプション).....	344
__resetFile (C-SPY システムマクロ).....	316	--drv_default_breakpoint (C-SPY コマンドラ インオプション).....	344
__restoreSoftwareBreakpoints (C-SPY システムマクロ).....	316		
__setCodeBreak (C-SPY システムマクロ).....	317		
__setDataBreak (C-SPY システムマクロ).....	318		
__setLogBreak (C-SPY システムマクロ).....	319		
__setSimBreak (C-SPY システムマクロ).....	321		

--drv_reset_to_cpu_start (C-SPY コマンドラ インオプション)	345	--mac_jtag_device (C-SPY コマンドラ インオプション)	353
--drv_restore_breakpoints (C-SPY コマンドラインオプション)	345	--mac_multiple_targets (C-SPY コマンドラ インオプション)	353
--drv_suppress_download (C-SPY コマンドラインオプション)	334	--mac_reset_pulls_reset (C-SPY コマンドラ インオプション)	354
--drv_swo_clock_setup (C-SPY コマンドラインオプション)	346	--mac_set_temp_reg_buffer (C-SPY コマンドラインオプション)	355
--drv_vector_table_base (C-SPY コマンドラ インオプション)	346	--mac_speed (C-SPY コマンドラ インオプション)	355
--drv_verify_download (C-SPY コマンドラ インオプション)	335	--mac_xscale_ir7 (C-SPY コマンドラ インオプション)	356
--endian (C-SPY コマンドラインオプション)	335	--mapu (C-SPY コマンドラインオプション)	356
--flash_loader (C-SPY コマンドラインオプション)	347	--plugin (C-SPY コマンドラインオプション)	357
--fpu (C-SPY コマンドラインオプション)	335	--proc_stack_xxx (C-SPY コマンドラ インオプション)	357
--gdbserv_exec_command (C-SPY コマンドラインオプション)	348	--rdi_allow_hardware_reset (C-SPY コマンドラインオプション)	358
--jlink_device_select (C-SPY コマンドラ インオプション)	348	--rdi_driver_dll (C-SPY コマンドラ インオプション)	358
--jlink_exec_commmmand (C-SPY コマンドラ インオプション)	348	--rdi_heartbeat (C-SPY コマンドラ インオプション)	336
--jlink_initial_speed (C-SPY コマンドラ インオプション)	349	--rdi_step_max_one (C-SPY コマンドラ インオプション)	359
--jlink_interface (C-SPY コマンドラ インオプション)	349	--semihosting (C-SPY コマンドラインオプション)	359
--jlink_ir_length (C-SPY コマンドラ インオプション)	350	--silent (C-SPY コマンドラインオプション)	360
--jlink_reset_strategy (C-SPY コマンドラ インオプション)	350	--stlink_interface (C-SPY コマンドラ インオプション)	360
--jlink_script_file (C-SPY コマンドラ インオプション)	351	--stlink_reset_strategy (C-SPY コマンドラ インオプション)	361
--jlink_speed (C-SPY コマンドラ インオプション)	351	--timeout (C-SPY コマンドラインオプション)	361
--lmiftdi_speed (C-SPY コマンドラ インオプション)	352	--verify_download (C-SPY コマンドラ インオプション)	362
--macro (C-SPY コマンドラ インオプション)	355	[JTAG ウォッチポイント] ダイアログボックス	138
--mac_handler_address (C-SPY コマンドラ インオプション)	352	[Power ログ設定] ウィンドウ	244
--mac_interface (C-SPY コマンドラ インオプション)	353	[Power ログ] ウィンドウ	246
		[SWO トレース設定] ダイアログボックス	188
		[SWO 設定] ダイアログボックス	190
		[イメージ] ウィンドウ	75
		[ウォッチ] ウィンドウ	106
		使用	97

[ウォッチ] ウィンドウに追加 ([シンボルメモリ] ウィンドウのコンテキストメニュー).....	168
[クイックウォッチ] ウィンドウ.....	112
C-SPY マクロの実行	283
[コードカバレッジ] ウィンドウ.....	252
[シミュレータ] メニュー.....	392
[シンボルメモリ] ウィンドウ.....	166
[シンボル] ウィンドウ.....	113
[スタック] ウィンドウ.....	168
[ソースの曖昧さの解決] ダイアログボックス.....	152
[タイムライン] ウィンドウ.....	200
[ターミナル I/O ログファイル] ダ イアログボックス	93
[ターミナル I/O] ウィンドウ	84, 92
[デバッグログ] ウィンドウ.....	94
[デバッグログ] ウィンドウのコンテ キストメニュー	94
[データログ概要] ウィンドウ.....	117
[データログ] ウィンドウ.....	115
[トレースの保存] ダイアログボックス.....	198
[トレースフィルタ] ブレークポイントダ イアログボックス (J-Link)	216
[トレースを検索] ダイアログボックス	219
[トレースを検索] ウィンドウ.....	221
[トレース開始] ブレークポイントダ イアログボックス	207, 209
[トレース式] ウィンドウ.....	218
[トレース停止] ブレークポイントダ イアログボックス	208
[トレース] ウィンドウ.....	193
[フィル] ダイアログボックス.....	164
[フラッシュローダの概要] ダイアログボックス	405
[ブレークポイントの使用] ダイアログボックス	135
[ブレークポイント] ウィンドウ.....	133
[ブレークポイント] ダイアログボックス イミディエイト.....	149
コード.....	136
データ.....	143
データログ.....	145
トレースフィルタ (J-Link).....	216

トレース開始.....	207, 209
トレース停止.....	208
ログ.....	141
[ベクタキャッチ] ダイアログボックス.....	150
[マクロ設定] ダイアログボックス.....	281
[メモリアクセスの編集] ダイアログボックス.....	175
[メモリアクセス設定] ダイアログボックス.....	173
[メモリ復元] ダイアログボックス.....	164
[メモリ保存] ダイアログボックス.....	163
[メモリ] ウィンドウ	159
[ライブウォッチ] ウィンドウ.....	108
[レジスタ] ウィンドウ.....	171
[ログファイル] ダイアログボックス.....	95
[ローカル] ウィンドウ.....	105
[位置入力] ダイアログボックス.....	150
[割込みステータス] ウィンドウ.....	269
[割込みの編集] ダイアログボックス.....	266
[割込みログ概要] ウィンドウ.....	274
[割込みログ] ウィンドウ.....	271
[割込み設定] ダイアログボックス.....	264
[関数トレース] ウィンドウ	199
[関数プロファイラ] ウィンドウ.....	229
[強制割込み] ウィンドウ.....	268
[呼出しスタック] ウィンドウ.....	90
バックトレース情報	84
[自動ステップの設定] ダイアログボックス.....	96
[自動] ウィンドウ	105
[静的変数の選択] ダイアログボックス.....	111
[静的] ウィンドウ	109
[入力モード] ダイアログボックス.....	93
[表示範囲] ダイアログボックス.....	206

数字

1x ユニット ([シンボルメモリ] ウィンドウの コンテキストメニュー).....	167
1x ユニット ([スタック] ウィンドウの コンテキストメニュー).....	170
1x ユニット ([メモリ] ウィンドウの コンテキストメニュー).....	161

2x ユニット ([シンボルメモリ] ウィンドウの コンテキストメニュー).....	167
2x ユニット ([スタック] ウィンドウの コンテキストメニュー).....	170
2x ユニット ([メモリ] ウィンドウの コンテキストメニュー).....	161
4x ユニット ([シンボルメモリ] ウィンドウの コンテキストメニュー).....	168
4x ユニット ([スタック] ウィンドウの コンテキストメニュー).....	170
4x ユニット ([メモリ] ウィンドウの コンテキストメニュー).....	161

