



GMD (GOWIN MCU Designer)

ユーザーガイド

SUG549-2.0J, 2025-06-13

著作権について(2025)

著作権に関する全ての権利は、**Guangdong Gowin Semiconductor Corporation** に留保されています。

GOWIN高云、、Gowin、Arora、LittleBee、及び GOWINSEMI は、当社により、中国、米国特許商標庁、及びその他の国において登録されています。商標又はサービスマークとして特定されたその他全ての文字やロゴは、それぞれの権利者に帰属しています。何れの団体及び個人も、当社の書面による許可を得ず、本文書の内容の一部もしくは全部を、いかなる視聴覚的、電子的、機械的、複写、録音等の手段によりもしくは形式により、伝搬又は複製をしてはなりません。

免責事項

当社は、GOWINSEMI Terms and Conditions of Sale (GOWINSEMI取引条件)に規定されている内容を除き、(明示的か又は黙示的に拘わらず)いかなる保証もせず、また、知的財産権や材料の使用によりあなたのハードウェア、ソフトウェア、データ、又は財産が被った損害についても責任を負いません。当社は、事前の通知なく、いつでも本文書の内容を変更することができます。本文書を参照する何れの団体及び個人も、最新の文書やエラッタ(不具合情報)については、当社に問い合わせる必要があります。

バージョン履歴

日付	バージョン	説明
2019/07/29	1.0J	初版。
2019/09/12	1.1J	● GNU ARM および RISC-V MCU コンパイルツールチェーンを統合。 ● Eclipse プラグインを更新。 ● GNU ARM MCU コンパイルツールチェーンを更新。 ● インターフェースを更新・最適化。
2019/11/12	1.2J	● GNU RISC-V MCU コンパイルツールチェーンを更新。 ● IDE に Java Development Kit を統合、IDE のインストールを簡素化。 ● IDE に GOWIN Programmer を統合。 ● IDE インターフェースをカスタマイズ、インターフェースオプションを簡素化、IDE の使用を簡素化。 ● IDE のインストールと使用に関する既知の問題を解決。
2019/12/10	1.3J	● License の管理を追加。 ● IDE のインターフェースを最適化。
2020/06/01	1.4J	● Gowin_EMPU(GW1NS-4C)および Gowin_EMPU_M3 のプログラミング、コンパイル、ダウンロード、およびオンラインデバッグをサポート。 ● Gowin_PicoRV32 ソフトウェアのオンラインデバッグをサポート。 ● デバッグソフトウェア OpenOCD を統合。 ● Olimex 社のデバッグドライバーを統合。 ● Gowin_EMPU(GW1NS-4C)のダウンロードをサポートするように Programmer を更新。 ● Gowin_EMPU(GW1NS-2C)、Gowin_EMPU_M1、Gowin_EMPU(GW1NS-4C)、Gowin_EMPU_M3、および Gowin_PicoRV32 のソフトウェアプログラミングのリファレンス・デザインを更新。 ● GMD および Gowin MCU の関連マニュアルをオンライン表示できる Help システムを追加。
2022/10/28	1.4.1J	Gowin_EMPU(GW1NS-2C)を削除。
2023/04/20	1.5J	● B バージョンの GW5AT-138 および GW5AST-138 デバイスのサポートを追加。 ● ヘルプドキュメントを更新。 ● Programmer を更新。 ● インストールパスの構成とツールチェーンの構成を追加。 ● MCU ソフトウェア開発ライブラリ、アプリケーション例、お

日付	バージョン	説明
		よびユーザーマニュアルを追加。 ● RTOS のアプリケーション例を追加。
2025/06/13	2.0J	● Eclipse ベースバージョンを 2024-12 にアップグレード。 Eclipse CDT 2024-12 ベースバージョンに基づいてこのバージョンを開発。 ● Arm GCC/GDB Toolchain のベースバージョンを 13.3.1-1.1 にアップグレード。 ● RISC-V GCC/GDB Toolchain のベースバージョンを 14.2.0-3 にアップグレード。 ● OpenOCD のベースバージョンを 0.12.0-4 にアップグレード。 ● OpenOCD が GWU2X、GWUSB、WinUSB JLink などのデバッグをサポート。 ● OpenOCD が Gowin_EMPU_M1、Gowin_EMPU(GW1NS-4C)、Gowin_PicoRV32 などの MCU のデバッグをサポート。 ● Windows build tools のベースバージョンを 4.4.1-2 にアップグレード。 ● Arm QEMU エミュレータ(ベースバージョン 8.2.6-1)を統合。 ● RISC-V QEMU エミュレータ(ベースバージョン 8.2.6-1)を統合。 ● Arm QEMU が Gowin_EMPU(GW1NS-4C) MCU のエミュレーションをサポート。 ● RISC-V QEMU が Gowin_PicoRV32 MCU のエミュレーションをサポート。 ● Profiling 機能をサポートする Gprof ツールを統合。 ● Code Coverage 機能をサポートする Gcov ツールを統合。 ● Parse HexDump Data 機能をサポート。Gprof および Gcov と併用して実行データを収集および解析。 ● Programmer のベースバージョンを 1.9.11.02 にアップグレード。 ● WinUSB ドライバーに基づくダウンロードとデバッグをサポートする GoBridge Driver ツールを統合。

目次

目次	i
図一覧	v
表一覧	xi
1 本マニュアルについて	1
1.1 マニュアルの内容	1
1.2 関連ドキュメント	1
1.3 用語、略語	2
1.4 テクニカル・サポートとフィードバック	3
2 概要	4
2.1 関連ドキュメント	5
2.2 MCU のサポート	5
3 サポートされる OS	7
3.1 サポートされる OS	7
3.2 動作環境	7
4 ダウンロードとインストール	8
4.1 ソフトウェアのダウンロード	8
4.2 IDE のインストール	8
4.3 ドライバーのインストール	9
4.3.1 GWU2X	10
4.3.2 GWUSB V3.3	11
4.3.3 GWUSB V4.1	13
4.3.4 Olimex	13
4.3.5 J-Link WinUSB	14
4.3.6 Segger J-Link	14

5 GUI	15
5.1 タイトル・バー	16
5.2 メニュー・バー	16
5.2.1 File メニュー	16
5.2.2 Edit メニュー	21
5.2.3 Source メニュー	23
5.2.4 Refactor メニュー	24
5.2.5 Navigate メニュー	24
5.2.6 Search メニュー	27
5.2.7 Project メニュー	27
5.2.8 Run メニュー	28
5.2.9 Gowin メニュー	31
5.2.10 Window メニュー	33
5.2.11 Help メニュー	35
5.3 ツールバー	36
5.4 プロジェクト・エクスプローラー	38
5.5 コード・エディタ	39
5.6 コンソール	39
5.6.1 コンソール・ビュー	39
5.6.2 問題ビュー	40
5.6.3 ターミナル・ビュー	40
6 プロジェクトの新規作成	43
6.1 プロジェクトの新規作成	43
6.1.1 プロジェクト(Arm MCU)の新規作成	44
6.1.2 プロジェクト(RISC-V MCU)の新規作成	46
6.2 既存のプロジェクトのインポート	49
7 旧バージョンの GMD で作成されたプロジェクトのインポート	52
7.1 インポートした Arm MCU プロジェクトの変更	52
7.2 インポートした RISC-V MCU プロジェクトの変更	54
8 プロジェクトのコンパイル	58

8.1 コンパイルツール	58
8.2 プロジェクトの設定.....	60
8.2.1 Arm MCU プロジェクトの設定.....	62
8.2.2 RISC-V MCU プロジェクトの設定	81
8.3 プロジェクトのコンパイル.....	102
8.3.1 コンパイル方法.....	102
8.3.2 プロジェクトのコンパイル.....	104
9 プロジェクトのダウンロード	106
9.1 ダウンロードツール.....	106
9.2 ダウンロード方法	107
9.3 プロジェクトのダウンロード	107
10 プロジェクトのデバッグ	113
10.1 デバッグツール.....	113
10.2 注意事項.....	113
10.3 デバッグモード.....	114
10.4 デバッグ方法	114
10.5 デバッグの設定.....	115
10.5.1 Arm MCU のデバッグの設定	116
10.5.2 RISC-MCU のデバッグの設定	124
10.6 プロジェクトの実行.....	127
10.6.1 デバッグ・ビューの起動	127
10.6.2 デバッグ機能のオプション.....	128
10.7 デバッグ・ビュー	129
10.7.1 メモリ・ビュー.....	130
10.7.2 メモリ・ブラウザ・ビュー.....	137
10.7.3 レジスタ・ビュー	140
10.7.4 変数ビュー	141
10.7.5 式ビュー.....	141
10.7.6 ブレークポイント・ビュー.....	142
10.7.7 逆アセンブリ・ビュー	142

10.7.8 コンソール・ビュー	143
11 QEMU によるエミュレーション・デバッグ	144
11.1 QEMU ツール	144
11.2 QEMU の使用方法	144
11.2.1 Arm MCU QEMU の設定	145
11.2.2 RISC-V MCU QEMU	149
11.3 プロジェクトのデバッグ	153
12 Code Coverage 機能	154
12.1 Code Coverage 機能について	154
12.2 Code Coverage 機能の使用	155
12.2.1 プロジェクトの作成	155
12.2.2 関連オプションの設定	155
12.2.3 プロジェクトのコンパイル	159
12.2.4 プロジェクトの実行および Code Coverage 機能の使用	159
13 Profiling 機能	167
13.1 Profiling 機能について	167
13.2 Profiling 機能の使用	167
13.2.1 プロジェクトの新規作成	168
13.2.2 関連オプションの設定	168
13.2.3 プロジェクトの実行および Profiling 機能の使用	170

図一覧

図 4-1 Gowin > GoBridgeDriver を選択	10
図 4-2 GoBridgeDriver ツール	10
図 4-3 Options > List All Devices を選択	11
図 4-4 GWU2X ドライバーのインストール	11
図 4-5 GWUSB V3.3 ドライバーのインストール	12
図 4-6 GWUSB V4.1 ドライバーのインストール	13
図 4-7 Olimex ドライバーのインストール	14
図 4-8 J-Link WinUSB ドライバーのインストール	14
図 5-1 ソフトウェアの GUI	15
図 5-2 New C/C++ Project	18
図 5-3 C Project	19
図 5-4 Import	20
図 5-5 Import Projects	21
図 5-6 Debug Configurations	30
図 5-7 Programmer	32
図 5-8 GoBridgeDriver	32
図 5-9 About GMD	36
図 5-10 プロジェクト・エクスプローラー	38
図 5-11 コード・エディタ	39
図 5-12 コンソール・ビュー	40
図 5-13 問題ビュー	40
図 5-14 ターミナルを起動	41
図 5-15 シリアルターミナル	42
図 5-16 シリアルターミナルの出力	42
図 6-1 プロジェクトの新規作成	43

図 6-2 プロジェクト・タイプとツールチェーンの選択.....	44
図 6-3 プラットフォームと構成の選択.....	45
図 6-4 ツールチェーンおよびそのパスの選択.....	46
図 6-5 プロジェクト・タイプとツールチェーンの選択.....	47
図 6-6 プラットフォームと構成の選択.....	48
図 6-7 ツールチェーンおよびそのパスの選択.....	49
図 6-8 インポート方法の選択.....	50
図 6-9 インポートするプロジェクトの選択.....	51
図 7-1 Arm プロジェクトをインポートした後のコンパイル・エラー.....	53
図 7-2 ツールチェーンの設定.....	54
図 7-3 RISC-V プロジェクトをインポートした後のコンパイル・エラー.....	55
図 7-4 RISC-V ツールチェーンの設定.....	56
図 7-5 コンパイル・オプションの設定.....	57
図 8-1 Arm 用ツールチェーン.....	59
図 8-2 RISC-V 用ツールチェーン.....	60
図 8-3 Arm MCU プロジェクトの設定.....	61
図 8-4 RISC-V MCU プロジェクトの設定.....	62
図 8-5 Target Processor.....	63
図 8-6 Optimization.....	64
図 8-7 Warnings.....	66
図 8-8 Debugging.....	67
図 8-9 GNU ARM Cross Assembler > Preprocessor.....	68
図 8-10 GNU ARM Cross Assembler > Includes.....	69
図 8-11 GNU ARM Cross Assembler > Warnings.....	70
図 8-12 GNU ARM Cross Assembler > Miscellaneous.....	71
図 8-13 GNU ARM Cross C Compiler > Preprocessor.....	72
図 8-14 GNU ARM Cross C Compiler > Includes.....	73
図 8-15 GNU ARM Cross C Compiler > Optimization.....	74
図 8-16 GNU ARM Cross C Compiler > Warnings.....	75
図 8-17 GNU ARM Cross C Compiler > Miscellaneous.....	76

☒ 8-18 GNU ARM Cross C Linker > General.....	77
☒ 8-19 GNU ARM Cross C Linker > Libraries.....	78
☒ 8-20 GNU ARM Cross C Linker > Miscellaneous.....	79
☒ 8-21 GNU ARM Cross Create Flash Image	80
☒ 8-22 GNU ARM Cross Print Size.....	81
☒ 8-23 Target Processor	82
☒ 8-24 Optimization	83
☒ 8-25 Warnings	85
☒ 8-26 Debugging	86
☒ 8-27 GNU RISC-V Cross Assembler > Preprocessor	87
☒ 8-28 GNU RISC-V Cross Assembler > Includes	88
☒ 8-29 GNU RISC-V Cross Assembler > Warnings	89
☒ 8-30 GNU RISC-V Cross Assembler > Miscellaneous.....	90
☒ 8-31 GNU RISC-V Cross C Compiler > Preprocessor	91
☒ 8-32 GNU RISC-V Cross C Compiler > Includes	92
☒ 8-33 GNU RISC-V Cross C Compiler > Optimization	93
☒ 8-34 GNU RISC-V Cross C Compiler > Warnings	94
☒ 8-35 GNU RISC-V Cross C Compiler > Miscellaneous	95
☒ 8-36 GNU RISC-V Cross C Linker > General	96
☒ 8-37 GNU RISC-V Cross C Linker > Libraries	97
☒ 8-38 GNU RISC-V Cross C Linker > Miscellaneous	98
☒ 8-39 GNU RISC-V Cross Create Flash Image.....	99
☒ 8-40 GNU RISC-V Cross Create Listing	100
☒ 8-41 GNU RISC-V Cross Print Size	101
☒ 8-42 Build	103
☒ 8-43 Arm MCU プロジェクトのコンパイル例.....	104
☒ 8-44 RISC-V MCU プロジェクトのコンパイル例.....	105
☒ 9-1 ダウンロードツール.....	106
☒ 9-2 Programmer	107
☒ 9-3 ダウンロードの設定	108

図 9-4 ダウンロードの設定	109
図 9-5 ダウンロードの設定	110
図 9-6 ダウンロードの設定	111
図 9-7 ダウンロードの設定	112
図 10-1 Debug.....	115
図 10-2 デバッグ・ビューの作成.....	116
図 10-3 Main ビュー	117
図 10-4 Debugger ビュー	118
図 10-5 Startup ビュー	119
図 10-6 Main ビュー	120
図 10-7 Debugger ビュー	121
図 10-8 Startup ビュー	123
図 10-9 Main ビュー	124
図 10-10 Debugger ビュー	125
図 10-11 Startup ビュー	126
図 10-12 デバッグ・ビューの起動.....	128
図 10-13 Window > Show View > Other.....	129
図 10-14 Debug View	130
図 10-15 メモリ・ビュー	130
図 10-16 監視したいスレッドまたはスタックを選択.....	131
図 10-17 Add Memory Monitor	131
図 10-18 Monitor Memory	132
図 10-19 従来形式のメモリ・レンダラー	132
図 10-20 浮動小数点形式のメモリ・レンダラーを作成	133
図 10-21 浮動小数点形式のメモリ・レンダラー	133
図 10-22 バイトオーダーの設定	134
図 10-23 格納内容の表示	134
図 10-24 レンダリング効果の設定.....	135
図 10-25 メモリ・レンダリング効果の設定	136
図 10-26 Import Memory	136

図 10-27 Export Memory	137
図 10-28 メモリ・ブラウザ・ビュー	137
図 10-29 従来形式のメモリ・レンダラー	138
図 10-30 浮動小数点形式のメモリ・レンダラーの作成	138
図 10-31 浮動小数点形式のメモリ・レンダラー	139
図 10-32 バイトオーダーの設定	139
図 10-33 Find/Replace	140
図 10-34 レジスタ・ビュー	140
図 10-35 変数ビュー	141
図 10-36 式ビュー	141
図 10-37 ブレークポイント・ビュー	142
図 10-38 逆アセンブリ・ビュー	142
図 10-39 コンソール・ビュー	143
図 11-1 GDB QEMU arm Debugging	145
図 11-2 Main ビュー	146
図 11-3 Debugger ビュー	147
図 11-4 Startup ビュー	148
図 11-5 GDB QEMU riscv32 Debugging	149
図 11-6 Main ビュー	150
図 11-7 Debugger ビュー	151
図 11-8 Startup ビュー	152
図 11-9 QEMU によるエミュレーション・デバッグの開始	153
図 12-1 すべてのソースコード・ファイルの指定	156
図 12-2 特定のソースコード・ファイルの指定	157
図 12-3 Linux Tools Path の設定	158
図 12-4 Linux Tools Path の設定	158
図 12-5 プロジェクトのコンパイル	159
図 12-6 Profiling 情報の表示	160
図 12-7 Select All	161
図 12-8 Parse HexDump Data	162

図 12-9 Profiling 情報の解析と対応ファイルの生成	162
図 12-10 生成される対応ファイルの確認	163
図 12-11 gcda ファイルを開く	163
図 12-12 Gcov	164
図 12-13 Create chart	165
図 12-14 棒グラフ	166
図 12-15 円グラフ	166
図 13-1 コンパイル・オプションの設定	168
図 13-2 Linux Tools Path の設定	169
図 13-3 Linux Tools Path の設定	170
図 13-4 Profiling 情報の表示	171
図 13-5 Select All	172
図 13-6 Parse HexDump Data	172
図 13-7 Profiling 情報の解析と対応ファイルの生成	173
図 13-8 生成される gmon.out ファイルの確認	173
図 13-9 gmon.out ファイルを開く	174
図 13-10 Gprof	174
図 13-11 Profiling 情報の表示	175
図 13-12 Create chart	176
図 13-13 棒グラフ	177
図 13-14 円グラフ	177

表一覧

表 1-1 用語、略語	2
表 2-1 MCU のサポート	5
表 4-1 GMD ソフトウェアの主要ファイルの説明	9
表 5-1 File メニュー	16
表 5-2 プロジェクト・タイプ	19
表 5-3 ツールチェーン	19
表 5-4 Edit メニュー	21
表 5-5 Source メニュー	23
表 5-6 Refactor メニュー	24
表 5-7 Navigate メニュー	24
表 5-8 Search メニュー	27
表 5-9 Project メニュー	27
表 5-10 Run メニュー	28
表 5-11 デバッグ方法	31
表 5-12 Gowin メニュー	31
表 5-13 Window メニュー	33
表 5-14 Help メニュー	35
表 5-15 ツールバーのオプション	36
表 8-1 Target Processor	63
表 8-2 Optimization	65
表 8-3 Debugging	67
表 8-4 GNU ARM Cross Assembler > Preprocessor	68
表 8-5 GNU ARM Cross C Compiler > Includes	73
表 8-6 GNU ARM Cross C Compiler > Optimization	74

表 8-7 GNU ARM Cross C Linker > General	77
表 8-8 GNU ARM Cross Create Flash Image.....	80
表 8-9 GNU ARM Cross Print Size.....	81
表 8-10 Target Processor.....	82
表 8-11 Optimization.....	84
表 8-12 Debugging	86
表 8-13 GNU RISC-V Cross Assembler > Miscellaneous	90
表 8-14 GNU RISC-V Cross C Compiler > Includes.....	92
表 8-15 GNU RISC-V Cross C Compiler > Optimization	93
表 8-16 GNU RISC-V Cross C Compiler > Miscellaneous	95
表 8-17 GNU RISC-V Cross C Linker > General.....	96
表 8-18 GNU RISC-V Cross C Linker > Miscellaneous	98
表 8-19 GNU RISC-V Cross Create Flash Image	99
表 8-20 GNU RISC-V Cross Create Listing	100
表 8-21 GNU RISC-V Cross Print Size.....	102
表 9-1 ダウンロードの設定	108
表 9-2 ダウンロードの設定	109
表 9-3 ダウンロードの設定	110
表 9-4 ダウンロードの設定	111
表 9-5 ダウンロードの設定	112
表 10-1 デバッグモード	114
表 10-2 Debugger ビュー	118
表 10-3 OpenOCD 構成ファイル	118
表 10-4 Startup ビュー	119
表 10-5 Debugger ビュー.....	121
表 10-6 Arm MCU デバイスのモデル	122
表 10-7 Startup ビュー	123
表 10-8 Debugger ビュー.....	125
表 10-9 OpenOCD 構成ファイル	126
表 10-10 Startup ビュー	126

表 10-11 デバッグ機能のオプション	128
表 11-1 サポートされる周辺機器	144
表 11-2 Debugger ビュー	147
表 11-3 Arm MCU QEMU	148
表 11-4 Startup ビュー	148
表 11-5 Debugger ビュー	151
表 11-6 RISC-V MCU QEMU	152
表 11-7 Startup ビュー	152
表 12-1 Gcov の機能	164
表 13-1 Gprof の機能	175

1 本マニュアルについて

1.1 マニュアルの内容

このマニュアルでは、プラットフォームのサポート、ダウンロードとインストール、ユーザー画面、プロジェクトの作成、旧バージョンの GMD で作成されたプロジェクトのインポート、プロジェクトのコンパイル、プロジェクトのダウンロード、プロジェクトのデバッグと実行、QEMU エミュレーション、Code Coverage 機能、Profiling 機能など、GMD ソフトウェアの使用方法について説明しています。本マニュアルに記載のソフトウェア GUI のスクリーンショットとサポートされる製品の一覧は、GMD 2025.01 の場合のものです。ソフトウェアのアップデートにより、一部の内容が変更される場合があります。

1.2 関連ドキュメント

GOWIN セミコンダクターのホームページ www.gowinsemi.com/ja から、以下の関連ドキュメントがダウンロード、参考できます：

- Gowin_EMPU(GW1NS-4C) IDE ユーザーガイド([IPUG928](#))
- Gowin_EMPU(GW5AS-25)クイックスタート ユーザーガイド([MUG1186](#))
- Gowin_EMPU_M1 IDE ソフトウェア ユーザーガイド([IPUG536](#))
- Gowin PicoRV32 IDE ユーザーガイド([IPUG910](#))
- Gowin_EMPU_M3 IDE ソフトウェア ユーザーガイド([IPUG919](#))

1.3 用語、略語

本マニュアルで使用される用語、略語、及びその意味を表 1-1 に示します。

表 1-1 用語、略語

用語、略語	正式名称	意味
GMD	GOWIN MCU Designer	GOWIN MCU ソフトウェア開発環境
ARM	Advanced RISC Machine	高度な RISC マシン
RISC-V	Reduced Instruction Set Computer Five	5 世代目の縮小命令セットコンピュータ
IDE	Integrated Development Environment	統合開発環境
MCU	Microcontroller Unit	マイクロコントローラ・ユニット
FPGA	Field Programmable Gate Array	フィールド・プログラマブル・ゲート・アレイ
SoC	System on Chip	システム・オン・チップ
GNU	GNU's Not Unix	自由なオペレーティングシステム
GCC	GNU Compiler Collection	GNU コンパイラコレクション
GDB	GNU Debug	GNU デバッガ
OpenOCD	Open On-Chip Debugger	オープンソースのオンチップデバッガ
QEMU	Quick Emulator	オープンソースのエミュレータ
CDT	C/C++ Development Tooling	C/C++開発ツール
EPL	Eclipse Public License	Eclipse パブリック・ライセンス
ABI	Application Binary Interface	アプリケーション・バイナリ・インターフェース
UART	Universal Asynchronous Receiver/Transmitter	汎用非同期送受信機
GPIO	General Purpose Input/Output	汎用入出力

1.4 テクニカル・サポートとフィードバック

GOWIN セミコンダクターは、包括的な技術サポートをご提供しています。使用に関するご質問、ご意見については、直接弊社までお問い合わせください。

ホームページ : www.gowinsemi.com/ja

E-mail : support@gowinsemi.com

2 概要

GMD ソフトウェアは、GOWIN セミコンダクターがオープンソースの GCC コンパイラツールチェーンとオープンソースの Eclipse フレームワーク、および自社の FPGA + MCU SoC アーキテクチャに基づいて独自に開発した、一般的な C/C++ 組み込みソフトウェア開発をサポートする新世代の MCU ソフトウェア開発環境です。これを使用することにより、開発者は、MCU ソフトウェア開発プロセスでコードのコンパイル、リンク、実行ファイルの生成、ダウンロード、デバッグを簡単に実装できます。

GMD ソフトウェアは、OpenOCD や QEMU などのデバッガとエミュレータを統合しています。

GMD ソフトウェアは、Code Coverage および Profiling 機能をサポートする Gcov および Gprof ツールを統合しています。Gcov ツールは、C/C++ のコード・カバレッジ(Code Coverage)をテストするためのツールです。GCC と併用することによって、C/C++ プログラムのステートメント、関数、および分岐のカバレッジ・テストを実現できます。Gprof ツールは、開発者が C/C++ プログラムの動作を理解するのに役立つパフォーマンス分析ツールです。Gprof を使用することにより、開発者はプログラム内の各関数の呼び出し情報、呼び出し回数、実行時間などを取得できるため、プログラムの最適化と実行効率の向上が容易になります。

GMD ソフトウェアは、Gowin 社の Arm および RISC-V アーキテクチャ MCU 製品 (Gowin_EMPU (GW1NS-4C)、Gowin_EMPU_M1、Gowin_EMPU (GW5AS-25)、Gowin_EMPU_M3、Gowin_PicoRV32 など) をサポートしています。

GMD ソフトウェアは、迅速なコード編集、コンパイル結果の表示、オンラインデバッグ、エミュレーションが可能なグラフィカルインターフェース(GUI)を提供します。また、GMD ソフトウェアには、ダウンロード用の Gowin Programmer が統合されています。

2.1 関連ドキュメント

Eclipse プラットフォームはオープンソースであり、EPL パブリック・ライセンス（無料、オープンソース）とグローバルな公開権を提供します。Eclipse 自体は単なるフレームワーク・プラットフォームであり、Eclipse プラットフォームのランタイムカーネル以外のすべての機能はさまざまなプラグインに配置されています。開発者は、さまざまな Eclipse プラグインを通じてプラットフォームの機能を拡張することも、必要なプラグインを自分で開発することもできます。また、プラグインを別のプラグインに挿入することで、最大限の統合を実現できます。

Eclipse はコミュニティで広く使用されているため、Eclipse IDE には一般的な使用方法が既にいくつか組み込まれています。特定のハードウェアに依存しない限り、通常は Eclipse の公式ドキュメントやその他のチュートリアルを参照できます。以下は、よく使われているおすすめの学習サイトです。

- <https://help.eclipse.org/latest/index.jsp>
- <https://eclipse-embed-cdt.github.io/>
- <https://mcuoneclipse.com/>

2.2 MCU のサポート

GMD ソフトウェアでサポートされている Gowin MCU、アーキテクチャ、および FPGA デバイスを表 2-1 に示します。

表 2-1 MCU のサポート

MCU	アーキテクチャ	FPGA デバイス
Gowin_EMPU(GW1NS-4C)	Arm Cortex-M3	GW1NS/R/ER-4C
Gowin_EMPU_M1	Arm Cortex-M1	GW1N/R シリーズ GW2A/N/R/NR シリーズ GW3A/R シリーズ GW5A/NT/NRT/R/RT/S/ST/T シリーズ
Gowin_EMPU(GW5AS-25)	Arm Cortex-M4	GW5AS-25
Gowin_EMPU_M3	Arm Cortex-M3	GW2A/N/R/NR シリーズ
Gowin_PicoRV32	RISC-V	GW1N/R シリーズ GW2A/N/R/NR シリーズ

MCU	アーキテクチャ	FPGA デバイス
		GW3A/R シリーズ GW5A/NT/NRT/R/RT/S/ST/T シリ ーズ

3 サポートされる OS

3.1 サポートされる OS

- Windows 10/11 (64-bit)
- Ubuntu 20.04/22.04/24.04 LTS

3.2 動作環境

- 4GB の RAM
- 6GB のストレージ空き容量

4 ダウンロードとインストール

4.1 ソフトウェアのダウンロード

Windows OS 用と Linux OS 用の 2 種類の GMD ソフトウェア圧縮パッケージが提供されています。

- GMD_20yy.xx_win.zip
- GMD_20yy.xx_linux.tar.gz

注記：

- ソフトウェアをダウンロードする前に、Gowin のホームページで登録・ログインしてください。
- ソフトウェアのインストール・パッケージ名「20yy.xx」の「y」はリリース年、「x」はリリースバージョンを表します。

ユーザーは、ローカルシステム環境に応じて、Gowin のホームページから Windows または Linux 用の対応する GMD ソフトウェア圧縮パッケージをダウンロードできます。

このソフトウェアは、すでに Windows 10/11 64-bit、Ubuntu 20.04 LTS で検証およびテストされており、上記のバージョンのオペレーティングシステムのご使用をお勧めします。

4.2 IDE のインストール

GMD ソフトウェアの圧縮パッケージをダウンロード後、IDE のインストールや環境変数の設定をすることなく、ローカルで解凍するだけで直接使用できます。解凍パスには漢字を含めることはできませんのでご注意ください。

GMD ソフトウェアはライセンスを必要とせず、直接開いて使用できます。

例えば、Windows 版 GMD_20yy.xx_win.zip には、解凍後に複数のフォルダとファイルが含まれています(表 4-1)。

表 4-1 GMD ソフトウェアの主要ファイルの説明

GMD ソフトウェアの ファイル	説明
configuration	GMD ソフトウェアの動作中に生成されるローカルの一時構成ファイル
dropins	ユーザーが手動で追加およびインストールしたプラグイン・ファイル
features	ソフトウェア機能に関連するモジュール
plugins	インストールされたすべてのプラグイン・ファイル
tool	Programmer や GoBridgeDriver などのツール
toolchain	GCC、OpenOCD、QEMU などのツールチェーン
GMD.exe	GMD ソフトウェアの実行ファイル（ダブルクリックで GMD ソフトウェアを実行）
GMD.ini	GMD ソフトウェアの初期化ファイル
ver.20yy.xx.txt	バージョン説明ファイル

4.3 ドライバーのインストール

GMD ソフトウェアは、OpenOCD による Arm および RISC-V MCU のデバッグ、および Segger J-Link による Arm MCU のデバッグをサポートしています。OpenOCD によるデバッグは、GWU2X、GWUSB、Olimex、J-Link など、Gowin またはサードパーティが開発した一般的なデバッガーをサポートしています。Windows OS では、WinUSB ドライバーが必要です。Segger J-Link によるデバッグの場合は、Segger J-Link ドライバーが必要です。

Windows OS では、GMD ソフトウェアは、GoBridgeDriver ツールを使用して関連する WinUSB ドライバーをインストールすることをサポートしています。図 4-1 と図 4-2 に示すように、GMD ソフトウェアのメニューバー > Gowin > GoBridgeDriver を選択し、GoBridgeDriver ツールを開きます（PC 管理者権限が必要です）。

図 4-1 Gowin > GoBridgeDriver を選択

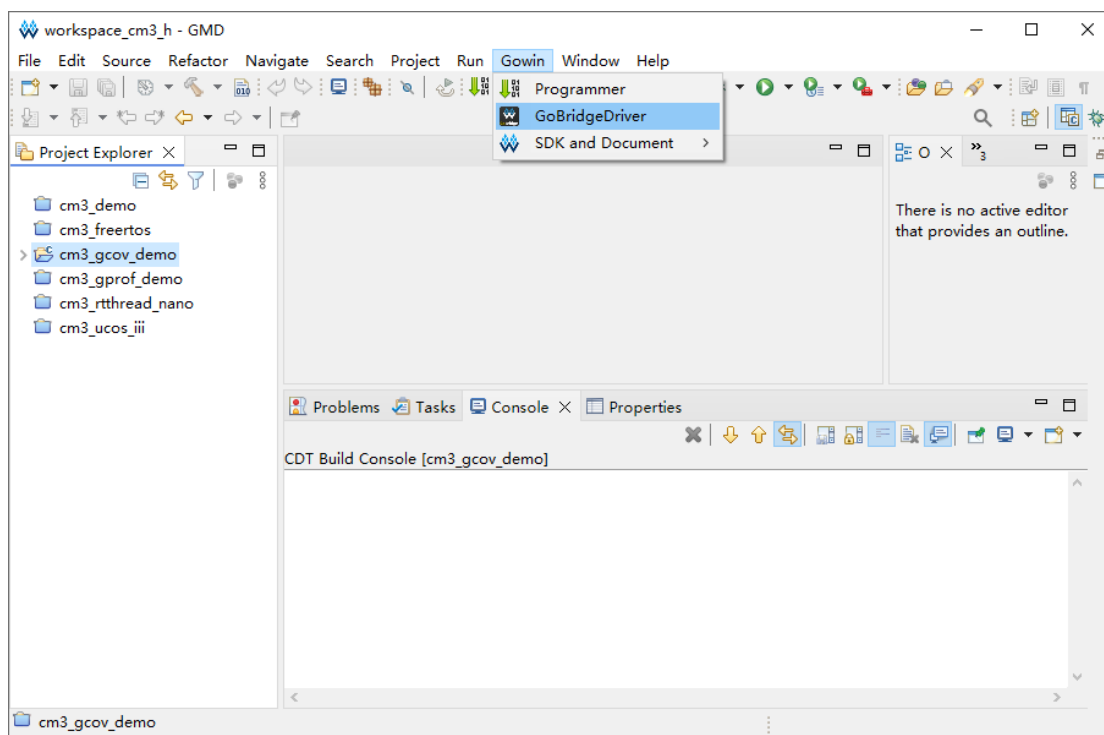
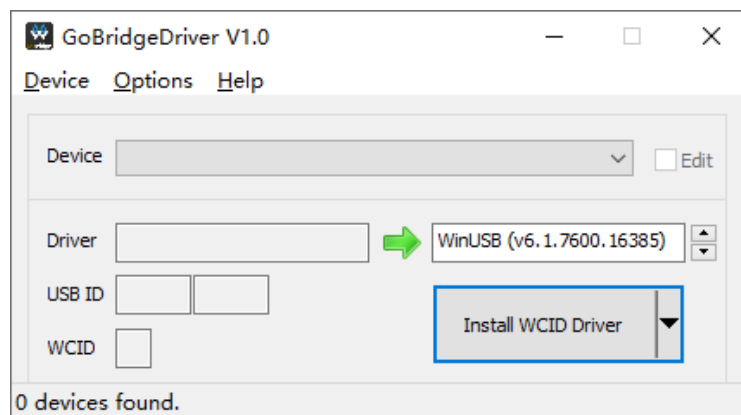


図 4-2 GoBridgeDriver ツール



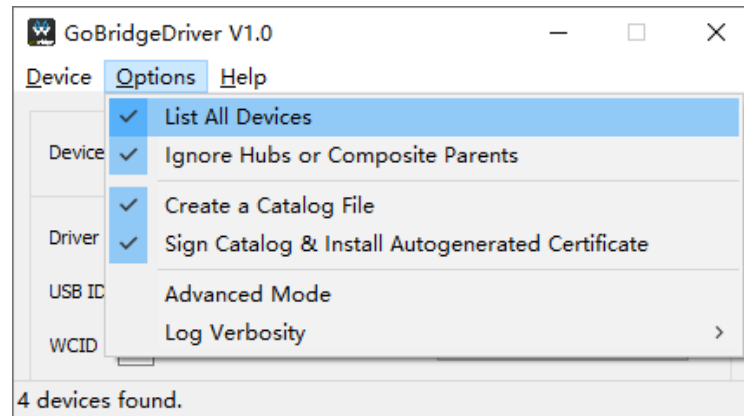
上記デバッガーのドライバーのインストール方法は以下に示すとおりです。

4.3.1 GWU2X

Gowin GWU2X ダウンローダー/デバッガーは、Gowin が独自に開発した USB to JTAG チップを内蔵しています。そのインストール方法は以下に示すとおりです。

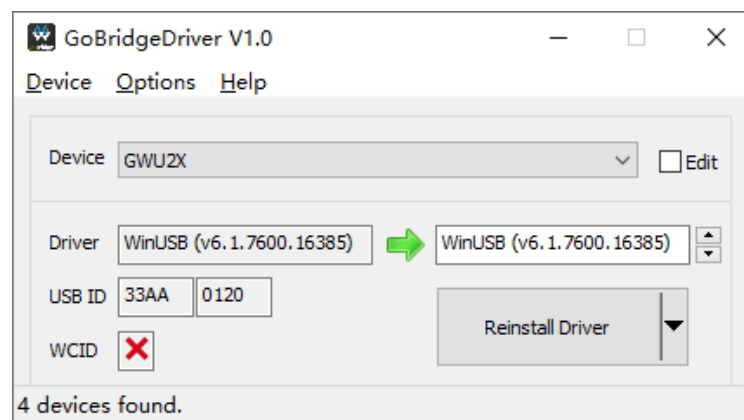
1. GWU2X デバイスを PC に挿入します。
2. メニューバー > Options > List All Devices を選択します。こうすると、PC に挿入されているすべての USB デバイスの一覧が表示されます(図 4-3)。

図 4-3 Options > List All Devices を選択



3. デバイス「GWU2X」を選択し、ドライバー「WinUSB」を選択し、「Install Driver」をクリックします(図 4-4)。

図 4-4 GWU2X ドライバーのインストール



注記：

- ドライバーが現在インストールされていない場合、このオプションは「Install Driver」と表示されます。
- ドライバーがすでにインストールされている場合、このオプションは「Reinstall Driver」と表示されます。
- 他のドライバーがインストールされている場合、「Replace Driver」と表示されます。

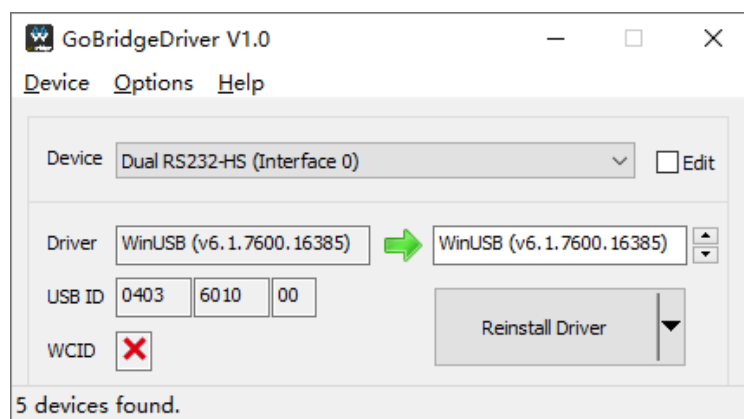
4.3.2 GWUSB V3.3

Gowin GWUSB V3.3 ダウンローダー/デバッガーは、FTDI 232H デュ

アル・チャネル・チップを内蔵しています。そのインストール方法は以下に示すとおりです。

1. GWUSB V3.3 デバイスを PC に挿入します。
2. メニューバー > Options > List All Devices を選択します。こうすると、PC に挿入されているすべての USB デバイスの一覧が表示されます。
3. それぞれデバイス「Dual RS232-HS (Interface 0)」と「Dual RS232-HS (Interface 1)」を選択し、ドライバー「WinUSB」を選択し、「Install Driver」をクリックします(図 4-5)。

図 4-5 GWUSB V3.3 ドライバーのインストール

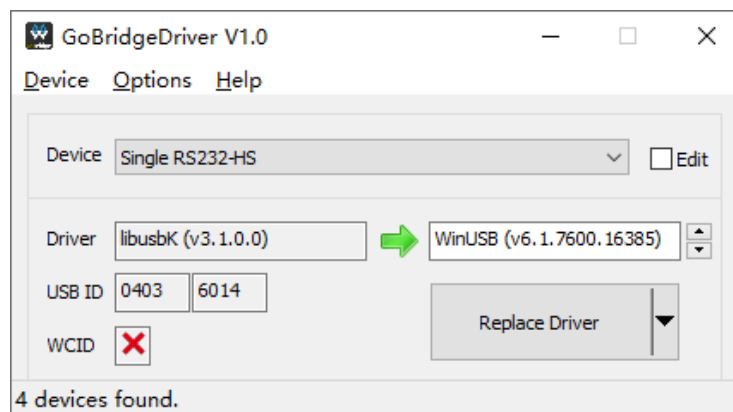


4.3.3 GWUSB V4.1

Gowin GWUSB V4.1 ダウンローダー/デバッガーは、FTDI 232H シングル・チャンネル・チップを内蔵しています。そのインストール方法は以下に示すとおりです。

1. GWUSB V4.1 デバイスを PC に挿入します。
2. メニューバー > Options > List All Devices を選択します。こうすると、PC に挿入されているすべての USB デバイスの一覧が表示されます。
3. デバイス「Single RS232-HS」を選択し、ドライバー「WinUSB」を選択し、「Install Driver」をクリックします(図 4-6)。

図 4-6 GWUSB V4.1 ドライバーのインストール

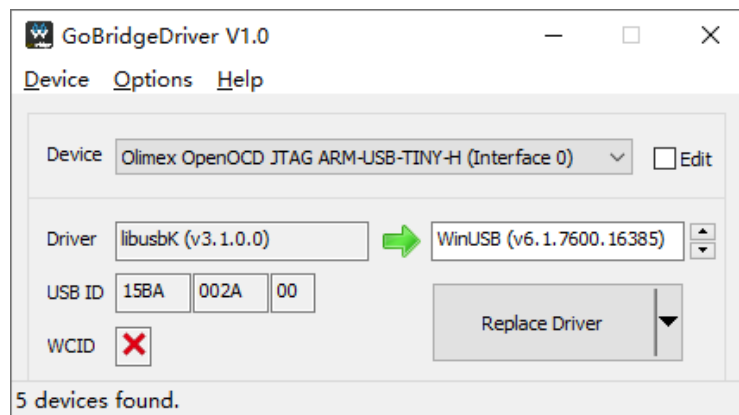


4.3.4 Olimex

ARM-USB-TINY-H 版 Olimex デバッガーは、FTDI FT2232H チップを内蔵しています。そのインストール方法は以下に示すとおりです。

1. Olimex デバイスを PC に挿入します。
2. メニューバー > Options > List All Devices を選択します。こうすると、PC に挿入されているすべての USB デバイスの一覧が表示されます。
3. それぞれデバイス「Olimex OpenOCD JTAG ARM-USB-TINY-H (Interface 0)」と「Olimex OpenOCD JTAG ARM-USB-TINY-H (Interface 1)」を選択し、ドライバー「WinUSB」を選択し、「Install Driver」をクリックします(図 4-7)。

図 4-7 Olimex ドライバーのインストール

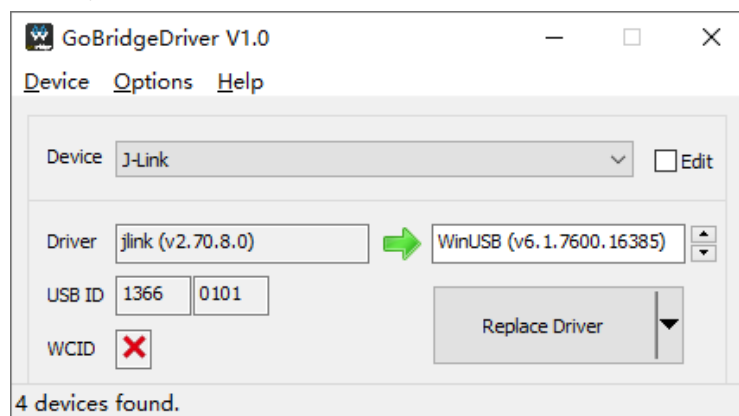


4.3.5 J-Link WinUSB

J-Link デバッガーのインストール方法は以下に示すとおりです。

1. J-Link デバイスを PC に挿入します。
2. メニューバー > Options > List All Devices を選択します。こうすると、PC に挿入されているすべての USB デバイスの一覧が表示されます。
3. デバイス「J-Link」を選択し、ドライバー「WinUSB」を選択し、「Install Driver」をクリックします(図 4-8)。

図 4-8 J-Link WinUSB ドライバーのインストール



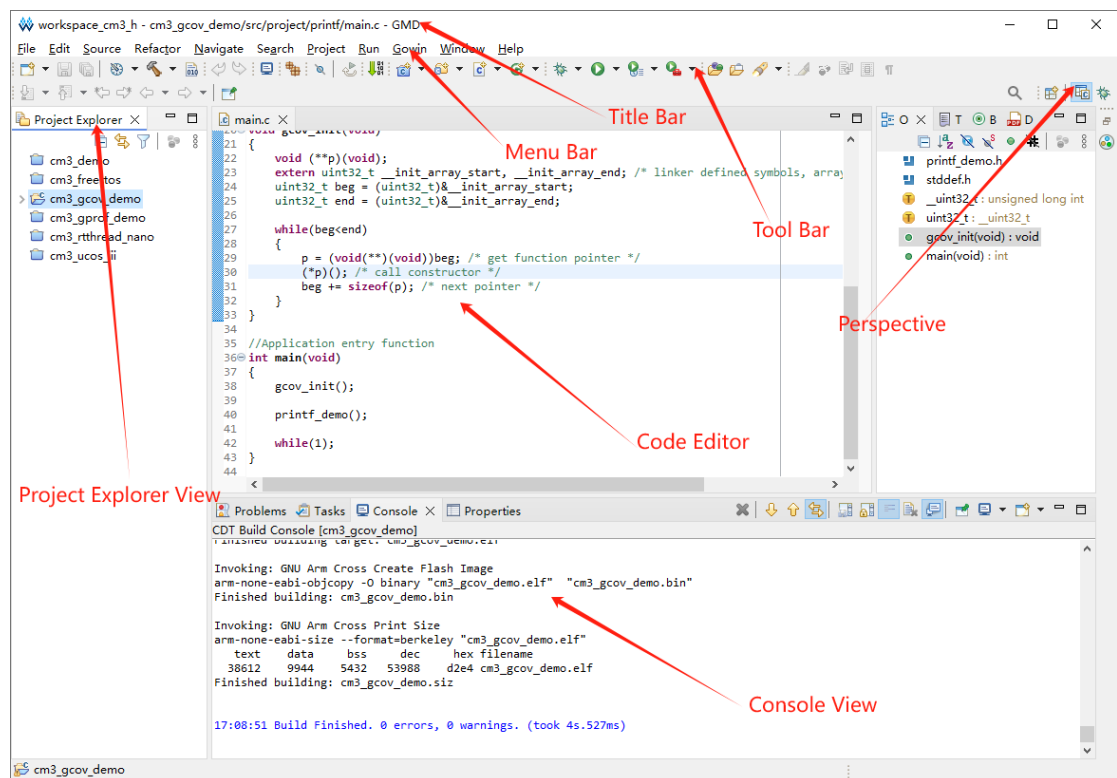
4.3.6 Segger J-Link

Segger J-Link のドライバーのインストールについては、<https://www.segger.com/downloads/jlink/>を参照してください。

5_{GUI}

「GMD¥GMD.exe」をダブルクリックして GMD ソフトウェアを開きます。図 5-1 に示すように、GMD ソフトウェアの GUI には、主にタイトル・バー(Title Bar)、メニュー・バー(Menu Bar)、ツール・バー(Tool Bar)、プロジェクト・エクスプローラー(Project Explorer)、コード・エディタ(Code Editor)、コンソール(Console View)、パースペクティブ (Perspective)などがあります。

図 5-1 ソフトウェアの GUI



ソフトウェアの各部分の説明は下記に示すとおりです。

5.1 タイトル・バー

主に、現在のソフトウェアのプロジェクトのワーク・スペース、名前、現在開いているファイルが表示されます。

5.2 メニュー・バー

File、Edit、Source、Refactor、Navigate、Search、Project、Run、Gowin、Window、および Help などのメニューがあります。

よく使用されるメニュー項目を以下に説明します。

5.2.1 File メニュー

File メニューを表 5-1 に示します。

表 5-1 File メニュー

メニュー	サブメニュー	機能の説明
New	Makefile Project with Existing Code	makefile を含む C/C++ プロジェクトを新規作成
	C/C++ Project	C/C++プロジェクトを新規作成
	Project...	C、C++、C/C++プロジェクトを新規作成
	Convert to a C/C++ Project (Adds C/C++ Nature)	C/C++特徴を非 C/C++プロジェクトに追加。 C/C++ツールチェーンを有効にする
	Source Folder	ソースコードフォルダを新規作成
	Folder	フォルダを新規作成
	Source File	C/C++ソースコードファイルを新規作成
	Header File	C/C++ヘッダーファイルを新規作成
	File from Template	C/C++テンプレートファイルを新規作成
	Class	C++クラスを新規作成
	Task	開発中の ToDo 項目や問

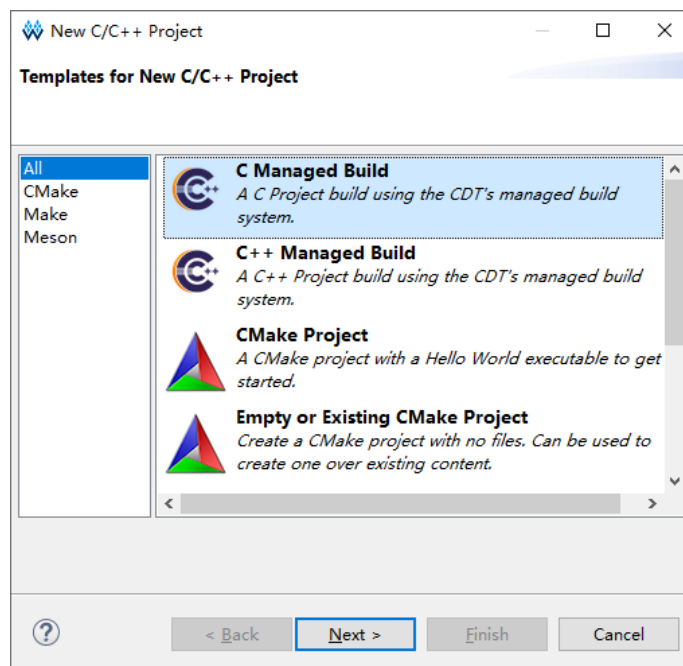
メニュー	サブメニュー	機能の説明
		題を追跡・管理
	Example...	XML ファイルの編集と検証を提供
	Other...	すべてのメニュー項目を表示する選択ウィザードを開く
Open File...	-	既存のファイルを開く
Open Projects from File System...	-	ファイル・システムから既存のプロジェクトをインポートして開く
Recent Files		最近開いたファイル
Close Editor	-	現在のエディタを閉じる
Close All Editors	-	すべてのエディタを閉じる
Save	-	すべてのエディタを閉じる
Save As...	-	別名で現在のエディタの内容を保存
Save All	-	すべてのエディタの内容と変更を保存
Revert	-	現在のエディタの内容を保存済みアーカイブの内容に復元
Move...	-	リソースを移動
Rename...	-	リソースの名前を変更
Refresh	-	プロジェクトのリソースを再編成してロード
Convert Line Delimiters To	Windows	Windows システム形式
	Unix	Unix システム形式
Print...	-	現在のエディタの内容を印刷
Import...	-	プロジェクトをインポート
Export...	-	プロジェクトをエクスポート

メニュー	サブメニュー	機能の説明
Properties	-	プロジェクトの属性を構成
Switch Workspace		別のワーク・スペースに切り替えてワーク・スペースを再起動
Restart	-	GMD ソフトウェアを再起動
Exit	-	GMD ソフトウェアを終了

C/C++ Project

「New > C/C++ Project」ダイアログ・ボックスは図 5-2 に示すとおりです。

図 5-2 New C/C++ Project



よく使われるオプションは「C Managed Build」や「C++ Managed Build」などです。たとえば、「C Managed Build」は図 5-3 に示すとおりです。そのプロジェクト・タイプは表 5-2、利用可能なツールチェーンは表 5-3 に示すとおりです。

図 5-3 C Project

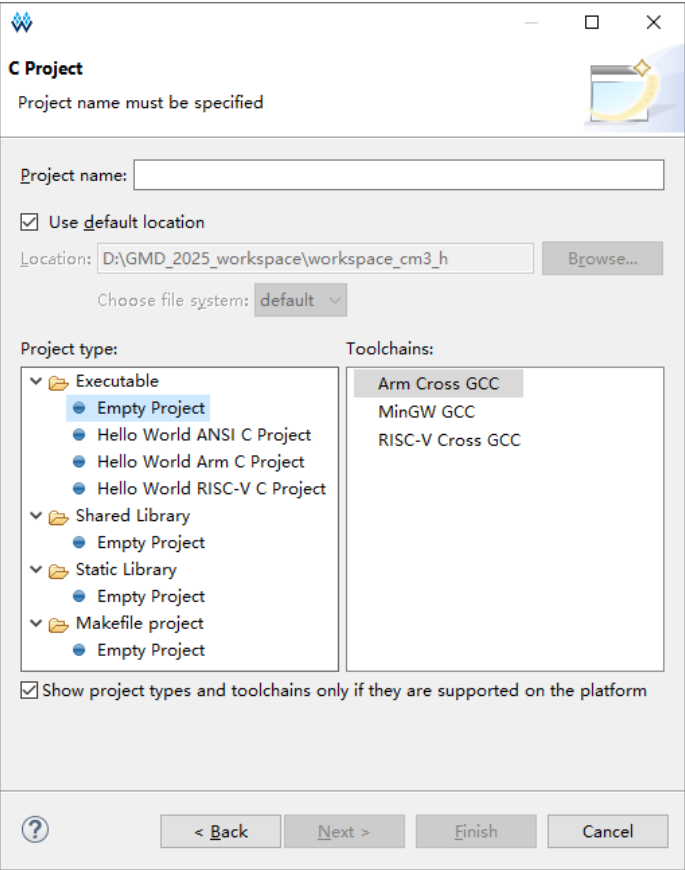


表 5-2 プロジェクト・タイプ

プロジェクト・タイプ	説明
Executable	実行可能プロジェクトを作成
Shared Library	共有ライブラリ・プロジェクトを作成
Static Library	静的ライブラリ・プロジェクトを作成
Makefile project	Makefile ベースのプロジェクトを作成

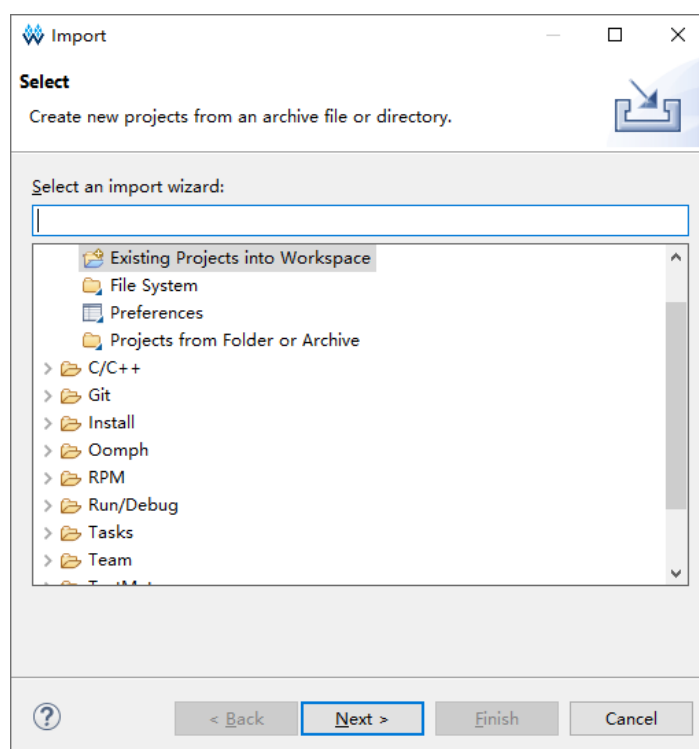
表 5-3 ツールチェーン

ツールチェーン	説明
Arm Cross GCC	Arm GCC クロスコンパイル・ツールチェーン
RISC-V Cross GCC	RISC-V GCC クロスコンパイル・ツールチェーン

Import

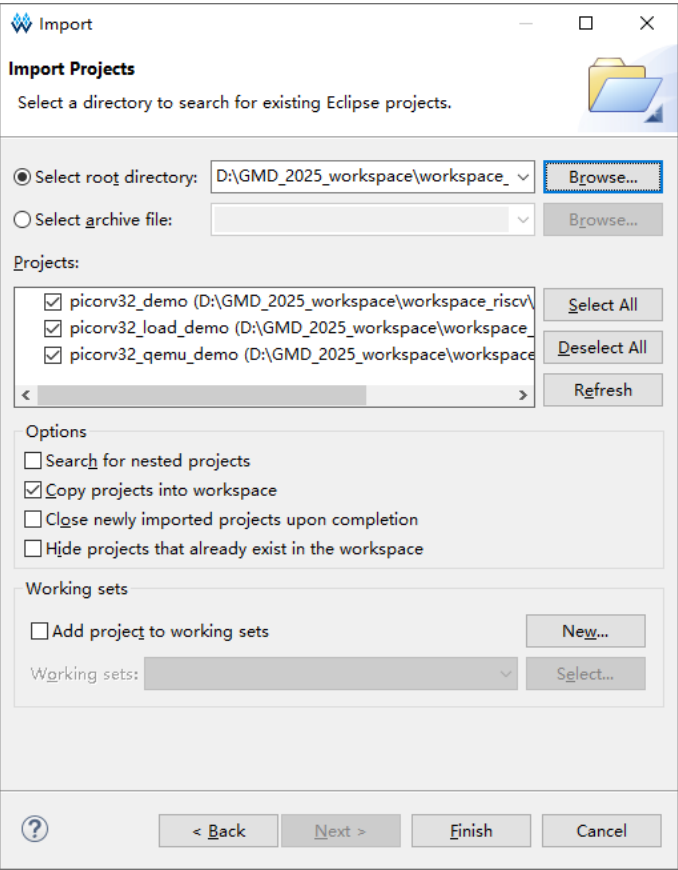
「New > Import…」ダイアログ・ボックスは図 5-4 に示すとおりです。

図 5-4 Import



よく使われるオプション「Existing Projects into Workspace」は図 5-5 に示すとおりです。

図 5-5 Import Projects



5.2.2 Edit メニュー

Edit メニューを表 5-4 に示します。

表 5-4 Edit メニュー

メニュー	サブメニュー	機能の説明
Undo	-	操作を取り消す
Redo	-	取り消した操作をやり直す
Cut	-	切り取り
Copy	-	コピー
Paste	-	貼り付け
Delete	-	選択したテキストまたは要素を削除
Select All	-	すべてのエディタの内容を選択
Expand Selection to	Enclosing Element	現在のコード選択範囲を、選択範囲を含む最も外側の要素まで拡張
	Next Element	現在の選択範囲を次のコード要素まで拡張
	Previous Element	選択範囲を前のコード要素まで拡張

メニュー	サブメニュー	機能の説明
	Restore Last Selection	選択範囲を前の状態に戻す
Toggle Block Selection	-	ブロック選択モードを切り替える
To multi-selection	-	複数の場所または複数のコード・セグメントを同時に選択
Find/Replace...	-	検索/置換
Find Word	-	現在選択されたテキストを検索
Find Next	-	現在選択されたテキストの次の結果を検索
Find Previous	-	現在選択されたテキストの前の結果を検索
Incremental Find Next	-	増分検索モードを開始、次の結果を増分検索
Incremental Find Previous	-	増分検索モードを開始、前の結果を増分検索
Bookmark...	-	現在選択されているテキストまたは要素にブックマークを追加
Task...	-	現在選択されているテキストまたは要素にユーザー定義の操作を追加
Smart Insert Mode	-	カーソルのテキスト編集領域挿入モードをインテリジェントに切り替える
Insert ChangeLog entry	-	コードに変更ログ・エントリを挿入
Show Tooltip Description	-	現在のカーソル位置の値をツールチップとして表示
Word Completion	-	入力中の文字に基づいて単語入力を自動補完
Quick Fix	-	コンパイラや IDE が提供するエラーや警告に基づいて、修正候補を自動的に生成したり、コードの修正を実行
Content Assist	-	インテリジェントなプロンプトと自動補完
Parameter Hints	-	メソッド呼び出しの際にメソッドの

メニュー	サブメニュー	機能の説明
		パラメータ・リストを自動的に表示
Set Encoding...	-	現在のファイルの文字エンコーディングを設定

5.2.3 Source メニュー

Source メニューを表 5-5 に示します。

表 5-5 Source メニュー

メニュー	機能の説明
Toggle Comment	選択した行において 1 行コメントを追加または削除
Add Block Comment	選択したコードブロックをブロックコメント形式 (<code>/* ... */</code>) でコメントアウト
Remove Block Comment	選択したブロックコメントを削除
Shift Right	選択したコード行を右にインデント
Shift Left	選択したコード行を左にインデント
Align Const	選択した領域内の定数 (通常は静的定数) を揃える
Correct Indentation	選択したコードのインデントを修正
Format	ファイル全体または選択したコードを自動的にフォーマット
Copy Qualified Name	現在のクラス、メソッド、変数、またはその他の要素の完全修飾名をコピー
Add Include	ヘッダーファイルの <code>#include</code> ディレクティブを追加
Organize Includes	ソースファイル内の <code>#include</code> 文を整理。未使用の <code>include</code> 項目を削除し、アルファベット順に並べ替える
Sort Lines	選択したコード行を並べ替える
Override Methods	現在のクラスのために親クラスまたはインターフェースから継承された 1 つ以上のメソッドを書き換える
Implement Method...	現在のクラスにおいて、インターフェースまたは抽象クラスの未実装メソッドを実装
Generate Getters and Setters...	クラスのフィールドに基づいて、対応する <code>getter</code> メソッドと <code>setter</code> メソッドを自動的に生成
Surround With	選択したコードを特定の構造(<code>try-catch</code> 、 <code>if</code> 、 <code>for</code> など)でラップ

5.2.4 Refactor メニュー

Refactor メニューを表 5-6 に示します。

表 5-6 Refactor メニュー

メニュー	機能の説明
Rename…	選択した要素の名前を変更し、要素へのすべての参照を更新
Extract Local Variable	新しい変数を作成し、現在選択されている式に割り当て、選択された項目を新しい変数への参照に置換
Extract Constant…	選択した式から static final フィールドを作成し、フィールド参照を置き換え、同じ式が表示される他の場所で書き換える
Extract Function…	新しい関数を作成し、現在選択されているステートメントに割り当て、選択された項目を新しい関数への参照に置換
Toggle Function	選択された関数定義をヘッダーファイル(クラス定義の内部または外部)から実装ファイルに移動するか、または戻る
Hide Method…	選択された方法を非表示
Apply Script…	保存されたリファクタリングリストを適用
Create Script…	完成したリファクタリングリストを後で使用するためにエクスポート
History…	リファクタリング履歴を表示

5.2.5 Navigate メニュー

Navigate メニューを表 5-7 に示します。

表 5-7 Navigate メニュー

メニュー	サブメニュー	機能の説明
Go Into	-	現在のビューを、選択したリソースのルートディレクトリにフォーカス
Go To	Back	前に表示していたレベルに戻る
	Forward	前回の「Back」操作前のレベルに戻る
	Up One Level	1 つ上のレベルに移動
	Resource…	特定のリソースに移動

メニュー	サブメニュー	機能の説明
	Next Member	現在のクラスまたはインターフェースの次のメンバーにジャンプ
	Previous Member	現在のクラスまたはインターフェースの前のメンバーにジャンプ
	Go to Matching Bracket	現在の括弧、中括弧、角括弧、または丸括弧に一致する別の括弧位置にジャンプ
	Next Bookmark	次のブックマークにジャンプ
Open Declaration	-	現在カーソルがある要素の宣言位置にジャンプ
Open Type Hierarchy	-	現在のタイプ（クラスまたはインターフェース）の継承階層を表示
Open Call Hierarchy	-	現在のメソッドの呼び出し階層を表示
Open Include Browser	-	現在のファイルに関連するインクルード関係を表示
Toggle Source/Header	-	ソースファイル（.c または .cpp ファイル）と対応するヘッダーファイル（.h ファイル）を切り替える
Open Element...	-	現在カーソルがある要素の詳細情報を開く
Open Type in Hierarchy...	-	タイプ階層ビューで現在のタイプを開く
Open Element in Call Hierarchy...	-	呼び出し階層ビューで現在の要素（メソッド、クラスなど）を表示
Open Resource...	-	すべてのリソースを表示。ワークスペース内の任意のリソースを選択して開くことが可能
Open Discovered Type...	-	検出されたタイプを表示して開く
Open Task...	-	現在のワークスペースのタスクのリストを開く
Activate Task...	-	現在選択されているタスクをアクティブにし、ハイライト表示
Deactivate Task	-	現在選択されているタスクを非アクティブにし、アクティブな状態から

メニュー	サブメニュー	機能の説明
		削除
Open Setup	User	現在のユーザーに関連する設定を表示または修正
	Installation	現在のインストールの構成と設定を表示
	Workspace	ワークスペースに関連する設定を表示または修正
	Self Products	現在のインスタンスに関連する製品の設定を表示
	Parent Models	現在のワークスペースまたはプロジェクトの親モデルに関連する設定を表示および管理
Open Setup Log	-	プロジェクトまたはワークスペースに関連する設定ログを表示
Show In	Include Browser	「Include Browser」ビューを表示
	C/C++ Projects	「C/C++ Projects」ビューを表示
	Project Explorer	「Project Explorer」ビューを表示
	Problem Details	「Problem Details」ビューを表示
	Terminal	「Terminal」ビューを表示
	Outline	「Outline」ビューを表示
	System Explorer	「System Explorer」ビューを表示
	Properties	現在選択されているファイル、プロジェクト、またはリソースの属性を表示
Quick Outline	-	現在のエディタの構造アウトラインを表示
Next	-	次のプロジェクトに移動
Previous	-	前のプロジェクトに移動
Previous Edit Location	-	カーソルを前回の編集位置に移動
Next Edit Location	-	カーソルを次の編集位置に移動
Go to Line...	-	指定した行番号の行にジャンプ
Back	-	エディタで前回表示したリソースに戻る
Forward	-	エディタで前回表示したリソースに

メニュー	サブメニュー	機能の説明
		進む

5.2.6 Search メニュー

Search メニューを表 5-8 に示します。

表 5-8 Search メニュー

メニュー	サブメニュー	機能の説明
Search...	-	検索
File...	-	ファイル内の特定のテキストまたはパターンを検索
C/C++...	-	C または C++コード内を検索
Remote...	-	リモートシステム内を検索
Text	Workspace	ワークスペース全体を検索
	Project	現在のプロジェクト内を検索
	File	単一ファイル内を検索
	Working Set...	現在選択されているワーキングセット内を検索
Quick Search	-	クイック検索

5.2.7 Project メニュー

Project メニューを表 5-9 に示します。

表 5-9 Project メニュー

メニュー	サブメニュー	機能の説明
Open Project	-	現在のプロジェクトを開く
Close Project	-	現在のプロジェクトを閉じる
Build All	-	すべてのプロジェクトをコンパイル
Build Configurations	Set Active	Release または Debug をアクティブに設定
	Manage...	管理
	Build by Working Set	コンパイルのワーキングセットを設定
	Set Active by Working Set	ワーキングセットをアクティブに設定

メニュー	サブメニュー	機能の説明
	Manage Working Sets...	ワーキングセットを管理
Build Project	-	現在選択したプロジェクトに増分コンパイルを実行
Build Working Set	Select Working Set	ワーキングセット内のすべてのプロジェクトに増分コンパイルを実行
Clean...	-	コンパイル結果を削除
Build Automatically	-	ワークスペース内のすべてのプロジェクトを自動的にコンパイル
Build Targets	Create...	新規作成
	Build...	コンパイル
	Rebuild Last Target	最近の目標を再コンパイル
C/C++ Index	Rebuild	C/C++プロジェクト全体のインデックスを再構築
	Freshen All Files	プロジェクト内のすべてのファイルをリフレッシュ
	Update with Modified Files	変更されたファイルのみのインデックスを更新
	Re-resolve Unresolved Includes	未解析のヘッダファイルのインクルード(#include)を再解析
	Search for Unresolved Includes	プロジェクト内のすべての未解析のヘッダーファイルを検索
	Create Parser Log File	パーサーのログファイルを生成
Properties	-	属性を構成

5.2.8 Run メニュー

Run メニューを表 5-10 に示します。

表 5-10 Run メニュー

メニュー	サブメニュー	機能の説明
Run Last Launched	-	前回起動したプログラムを実行
Debug Last Launched	-	前回起動したプログラムをデバッグ
Profile Last	-	前回起動したプログラムをプロファ

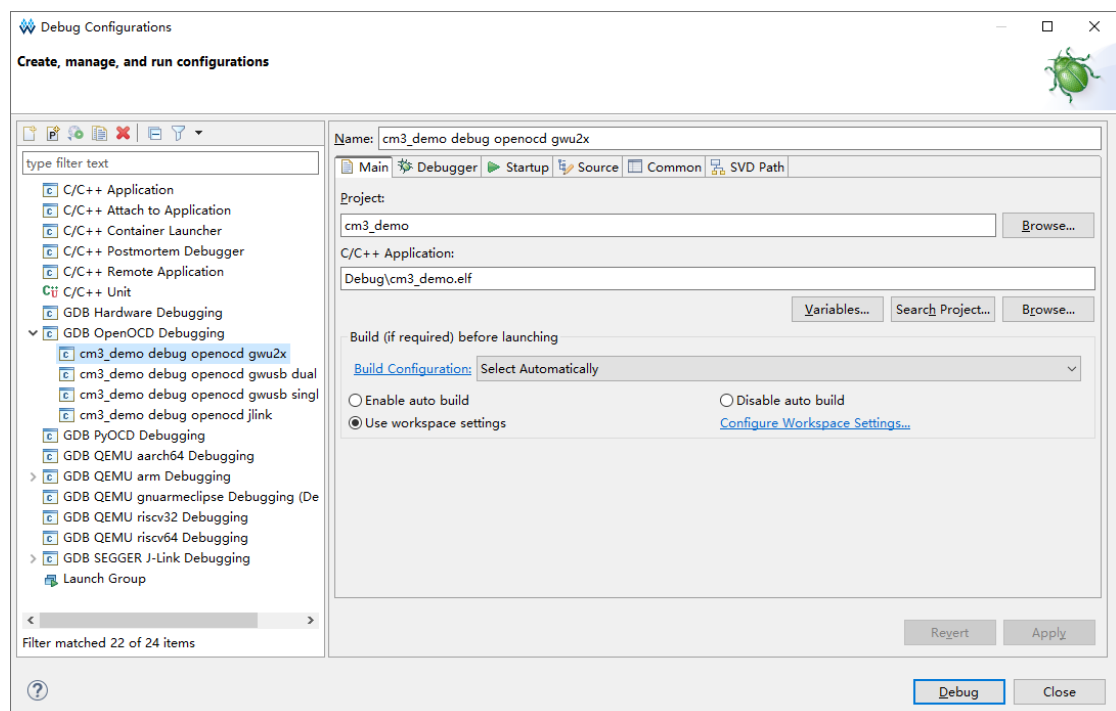
メニュー	サブメニュー	機能の説明
Launched		イリング
Profile History	-	プロファイリング履歴における最後のプロファイリング構成を使用
Profile As	Local C/C++ Application	ローカル C/C++ アプリケーション方式でプログラムを分析。ローカルにインストールされたコンパイラとデバッガを使用してホスト環境で直接プロファイリング。
Profile Configurations...	-	プログラム・プロファイリングのオプションを構成・管理
Run History	-	実行履歴における最後の実行構成を使用
Run As	C/C++ Container Application	C/C++ アプリケーションをコンテナ方式で実行し、実行ファイルを Docker コンテナにマウントして、コンテナ内の分離された環境で実行
	Local C/C++ Application	ローカル C/C++ アプリケーション方式でプログラムを実行。ローカルにインストールされたコンパイラとデバッガを使用してホスト環境で直接実行
Run Configurations ...	-	プログラム実行のオプションを構成・管理
Debug History	-	デバッグ履歴における最後のデバッグ構成を使用
Debug As	C/C++ Container Application	C/C++ アプリケーションをコンテナ方式でデバッグし、実行ファイルを Docker コンテナにマウントして、コンテナ内の分離された環境でデバッグ
	Local C/C++ Application	ローカル C/C++ アプリケーション方式でプログラムをデバッグ。ローカルにインストールされたコンパイラとデバッガを使用してホスト環境で直接デバッグ
Debug Configuration...	-	プログラム・デバッグのオプションを構成・管理
Breakpoint Types	-	ブレークポイントのタイプ(C/C++)

メニュー	サブメニュー	機能の説明
		Breakpoint または C/C++ Dynamic Printf)を構成
Toggle Breakpoint	-	現在の行にブレークポイントを追加または削除
Toggle Line Breakpoint	-	コード行に行ブレークポイントを設定
Toggle Watchpoint	-	現在のフィールドに対してウォッチポイントを追加または削除
Toggle Method Breakpoint	-	メソッドの入口を指定するか、終了位置にブレークポイントを設定
Skip All Breakpoints	-	すべてのブレークポイントを無効にする
Remove All Breakpoint	-	ワークスペース内のすべてのブレークポイントを完全に削除
External Tools	-	外部ツールの実行と構成を許可

Debug Configurations

図 5-6 に示すように、「Run > Debug Configurations…」オプションは、デバッグを構成するために使用されます。

図 5-6 Debug Configurations



よく使われるデバッグ方法を表 5-11 に示します。

表 5-11 デバッグ方法

デバッグ方法	説明
GDB OpenOCD Debugging	OpenOCD を使用して Arm および RISC-V MCU をデバッグ
GDB QEMU arm Debugging	QEMU を使用して Arm MCU をエミュレート・デバッグ
GDB QEMU riscv32 Debugging	QEMU を使用して 32 ビットの RISC-V MCU をエミュレート・デバッグ
GDB QEMU riscv64 Debugging	QEMU を使用して 64 ビットの RISC-V MCU をエミュレート・デバッグ
GEB SEGGER J-Link Debugging	Segger の公式ドライバーと J-Link を使用して Arm MCU をデバッグ

Run Configurations

「Run > Run Configurations…」オプションは「Debug Configurations…」オプションと同じなので、ここでは説明を省略します。

5.2.9 Gowin メニュー

Gowin メニューには、Programmer、GoBridgeDriver など、Gowin が提供するツールがいくつか含まれています(表 5-12)。

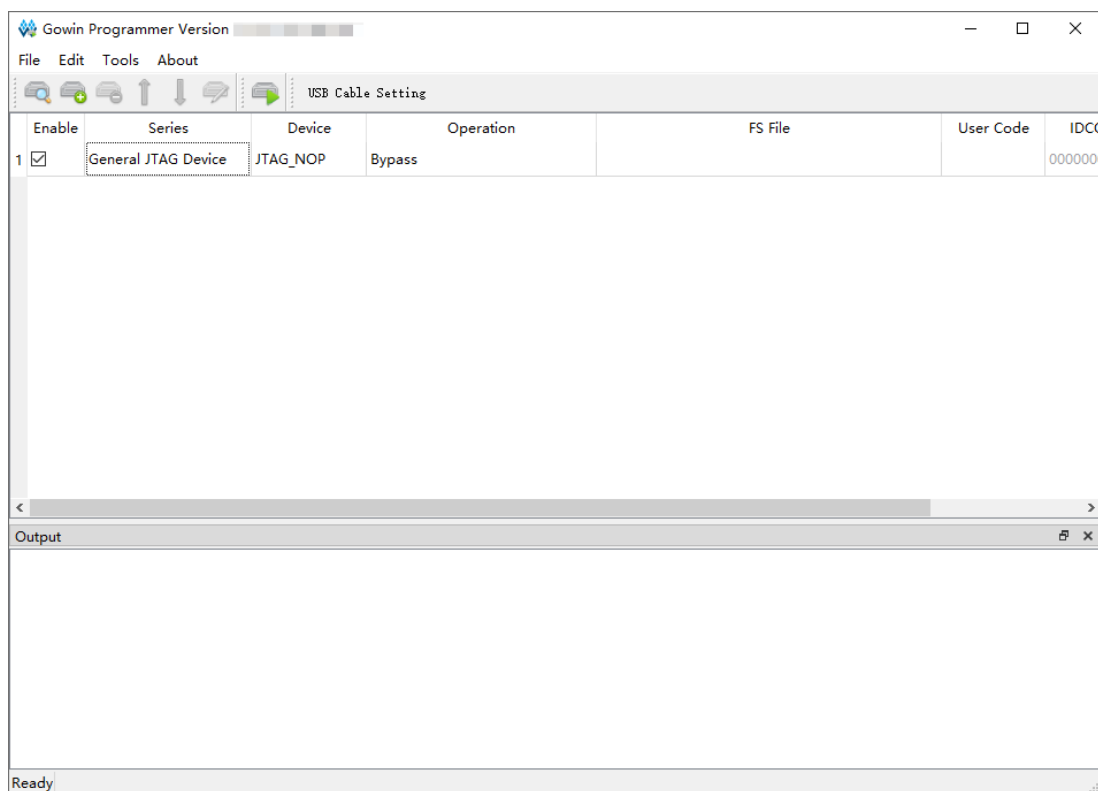
表 5-12 Gowin メニュー

メニュー	機能の説明
Programmer	ダウンロード・ソフトウェアである Programmer を起動
GoBridgeDriver	ドライバー・ソフトウェアである GoBridgeDriver を起動
SDK and Document	Gowin MCU 関連の SDK とユーザーガイドの公式リンク

Programmer

「Gowin > Programmer」オプションをクリックしてダウンロード・ソフトウェアである **Programmer** を開きます(図 5-7)。

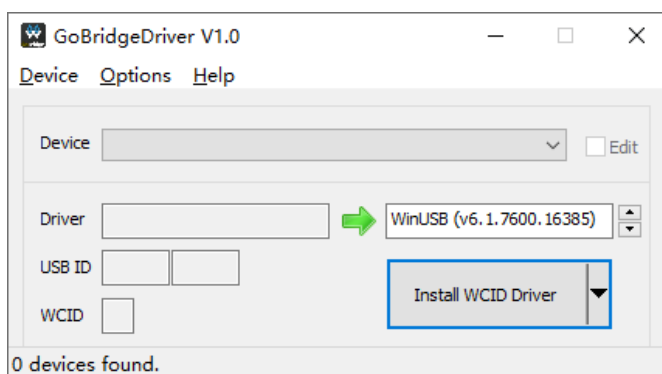
図 5-7 Programmer



GoBridgeDriver

「Gowin > GoBridgeDriver オプション」をクリックしてドライバー・ソフトウェアである **GoBridgeDriver** を開きます(図 5-8)。

図 5-8 GoBridgeDriver



5.2.10 Window メニュー

Window メニューを表 5-13 に示します。

表 5-13 Window メニュー

メニュー	サブメニュー	機能の説明
New Window	-	新しいワークスペース・ウィンドウを開く
Editor	Toggle Split Editor (Horizontal)	現在のウィンドウに新しいエディタ・インスタンスを水平方向に開く
	Toggle Split Editor (Vertical)	現在のウィンドウに新しいエディタ・インスタンスを垂直方向に開く
	Clone	現在のウィンドウにエディタ・インスタンスを複製
Appearance	Hide Toolbar	ツールバーを非表示にする
	Hide Status Bar	ステータスバーを非表示にする
	Toggle Full Screen	全画面モードを切り替える
	Maximize Active View or Editor	アクティブなビューまたはエディタを最大化
	Minimize Active View or Editor	アクティブなビューまたはエディタを最小化
Show View	Build Targets	Build Targets ビューを表示
	C/C++ Projects	C/C++ Projects ビューを表示
	Console	Console ビューを表示
	Documents	Documents ビューを表示
	Include Browser	Include Browser ビューを表示
	Navigator	Navigator ビューを表示
	Outline	Outline ビューを表示
	Problem Details	Problem Details ビューを表示
	Problems	Problems ビューを表示
	Project Explorer	Project Explorer ビューを表示
	Properties	Properties ビューを表示
	Search	Search ビューを表示
	Task List	Task List ビューを表示

メニュー	サブメニュー	機能の説明
	Tasks	Tasks ビューを表示
	Terminal	Terminal ビューを表示
	Other...	その他
Perspective	Open Perspective	新しいパースペクティブを開く
	Customize Perspective ...	現在のパースペクティブをカスタマイズ
	Save Perspective As...	現在のパースペクティブのレイアウトと設定を新しいパースペクティブとして保存
	Reset Perspective...	現在のパースペクティブをデフォルトのレイアウトに復元
	Close Perspective	現在のパースペクティブを閉じる
	Close All Perspectives	開いているすべてのパースペクティブを閉じる
Navigation	Show System Menu	現在アクティブなビューまたはエディタのシステムメニューを表示
	Show View Menu	現在アクティブなビューのメニューを表示
	Active Editor	エディタをアクティブ化し、フォーカスを現在アクティブなエディタに切り替える
	Next Editor	次の開いているエディタに切り替える
	Previous Editor	前の開いているエディタに切り替える
	Switch to Editor...	開いているすべてのエディタを一覧表示。ユーザーがエディタを切り替えることが可能
	Next View	次の開いているビューに切り替える
	Previous View	前の開いているビューに切り替える
	Next Perspective	次の開いているパースペクティブに切り替える
	Previous Perspective	前の開いているパースペクティブに切り替える

メニュー	サブメニュー	機能の説明
Preferences	-	プリファレンス

5.2.11 Help メニュー

Help メニューを表 5-14 に示します。

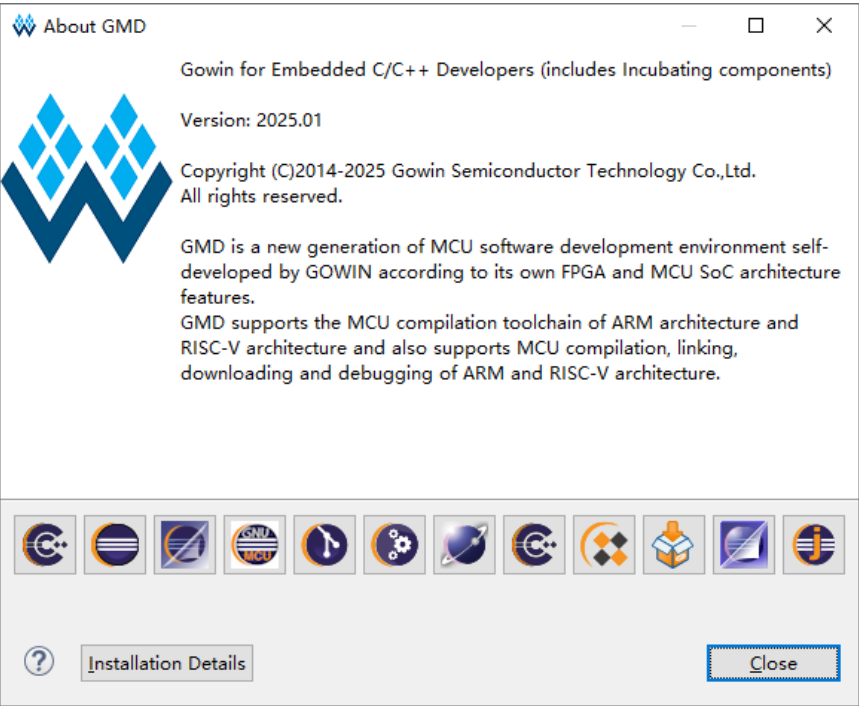
表 5-14 Help メニュー

メニュー	機能の説明
Welcome	Eclipse の基本機能のあるウェルカムページを開く
Help Contents	Eclipse のヘルプドキュメントを開く
Search	ヘルプドキュメントでキーワードを検索
Show Context Help	コンテキストヘルプを表示
Show Active Keybindings...	現在利用可能なショートカットキーの一覧を表示
Tips and Tricks...	Eclipse の使用に関するヒントとコツを提供
Report Bug and Enhancement ...	質問や意見を提出
Cheat Sheets...	インタラクティブな操作ガイドを提供
Eclipse User Storage	機密情報を暗号化して安全に保管するために使用
Performance Setup Tasks...	IDE のパフォーマンスを最適化するツール
Check for Updates	Eclipse またはインストール済みプラグインのアップデートをチェック
Install New Software...	新しいプラグインや拡張機能をインストール
Eclipse Marketplace...	Eclipse プラグインマーケットを開く
About GMD	GMD ソフトウェアの現在のバージョンなどの情報を表示

About GMD

「Help > About GMD」オプションを図 5-9 に示します。

図 5-9 About GMD



5.3 ツールバー

主に、ファイルまたはプロジェクトの作成(New)、ファイルの保存(Save)、すべてのファイルの保存(Save All)、取り消し(Undo)、やり直し(Redo)、構成の管理(Manage configurations)、コンパイル(Build、Build All)、ターミナル(Open a Terminal)、ダウンロード(Programmer)、デバッグ(Debug)、実行(Run)、プロファイリング(Profile)、検索(Search)など、よく使用される機能へのクイックアクセスを提供します(表 5-15)。

表 5-15 ツールバーのオプション

オプション	アイコン	機能の説明
New		ファイル/プロジェクトを新規作成
Save		現在のファイルを保存
Save All		開いているすべてのファイルを保存
Manage configurations		構成を管理 (Debug と Release を含む)
Build		現在のプロジェクトをコンパイル

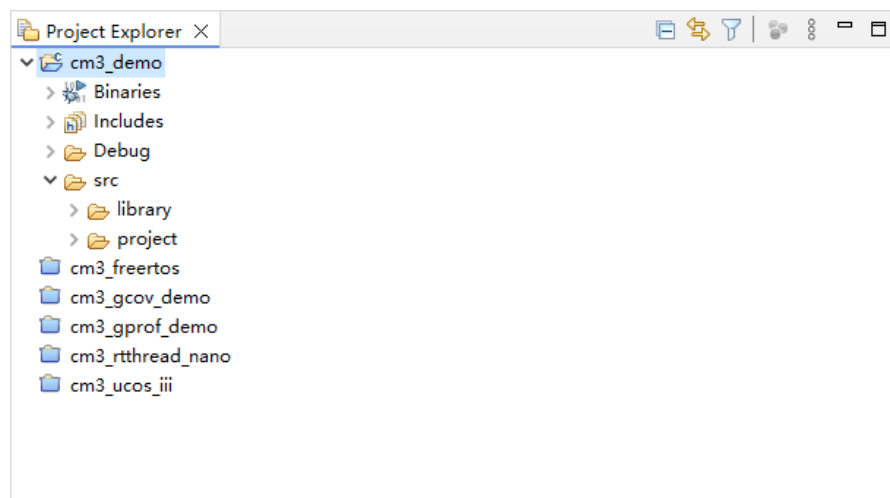
オプション	アイコン	機能の説明
Build All		ワークスペースで開いているすべてのプロジェクトをコンパイル
Undo		最近の編集を取り消す
Redo		最近の編集をやり直す
Open a Terminal		シリアルターミナルなどのターミナルを開く
Skip All Breakpoints		すべてのブレークポイントをスキップ
Restart		デバッグ・セッション全体を終了せずに、現在のメソッドの実行を再開
Debug		アプリケーションのデバッグ・セッションを開始
Run		アプリケーションの実行セッションを開始
Profile		アプリケーションのプロファイリング・セッションを開始
Programmer		ダウンロード・ソフトウェアである Programmer を起動
Open Element		現在カーソルがある要素の詳細情報を開く
Open Task		現在のワークスペースのタスクのリストを開く
Search		検索
Toggle Mark Occurrences		現在のファイル内で選択した要素（変数、メソッド、型など）へのすべての参照を強調表示
Toggle Work Wrap		エディタで自動行折り返しを有効または無効にする
Toggle Block Selection Mode		テキストをブロック方式で選択して編集
Show Whitespace Characters		エディタで空白文字を表示
Next Annotation		ソースコード内の次の注釈マーカーに移動

オプション	アイコン	機能の説明
Previous Annotation		ソースコード内の前の注釈マーカーに移動
Last Edit Location		前の編集位置にジャンプ
Next Edit Location		次の編集位置にジャンプ
Back		前に表示したリソースまたはビューに戻る
Forward		前回の「Back」操作前のリソースまたはビューに戻る
Pin Editor		エディタを自動的に閉じる
Open Perspective		新しいパースペクティブを開く
C/C++		C/C++パースペクティブに切り替える
Debug		Debug パースペクティブに切り替える

5.4 プロジェクト・エクスプローラー

ここでプロジェクトおよびその関連ファイルを管理することができます(図 5-10)。

図 5-10 プロジェクト・エクスプローラー



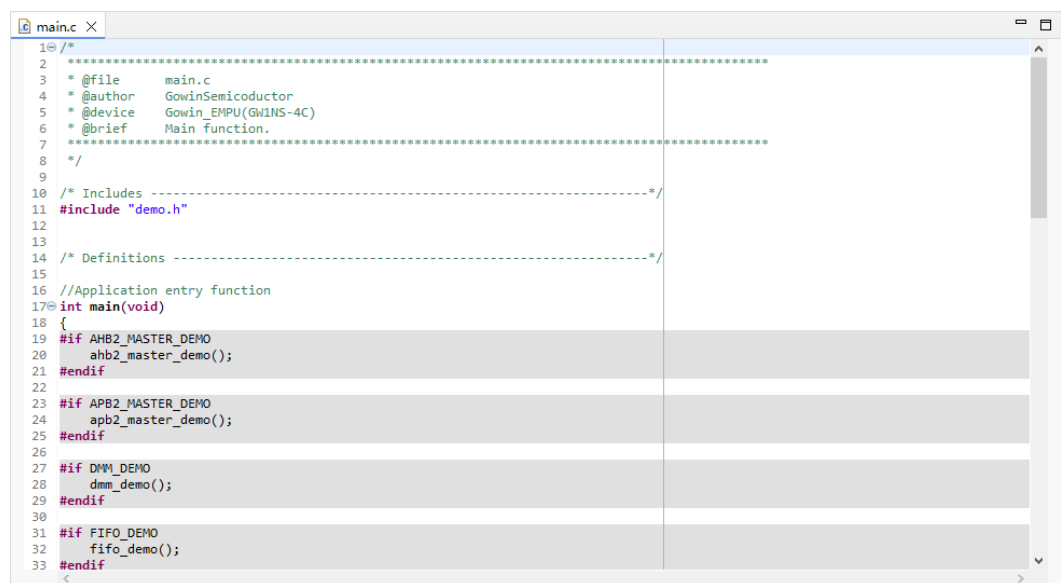
5.5 コード・エディタ

ここでファイルを編集することができます(図 5-11)。

「File > New > Source File」または「File > New > Header File」オプション：新しいソースファイル(.c/cpp)またはヘッダーファイル(.h)を作成します。

「File > Close Editor」オプション：編集領域に現在表示されているファイルを閉じます。「File > Close All Editors」オプション：編集領域で開いているすべてのファイルを閉じます。

図 5-11 コード・エディタ



5.6 コンソール

ソフトウェアの動作中に、さまざまな処理情報を表示します。コンソール(Console)、問題(Problems)、タスク(Tasks)、ターミナル(Terminal)など、さまざまな種類の出力情報のビューがあります。

5.6.1 コンソール・ビュー

コンソール・ビューには、例えばコンパイル情報が表示されます(図 5-12)。

図 5-12 コンソール・ビュー

```

CDT Build Console [cm3_demo]
./src/library/middlewares/dmm/malloc.o ./src/library/middlewares/delay/delay.o
./src/library/libraries/drivers/src/gw1ns4c_gpio.o ./src/library/libraries/drivers/src/gw1ns4c_i2c.o
./src/library/libraries/drivers/src/gw1ns4c_misc.o ./src/library/libraries/drivers/src/gw1ns4c_rtc.o
./src/library/libraries/drivers/src/gw1ns4c_spi.o ./src/library/libraries/drivers/src/gw1ns4c_syscon.o
./src/library/libraries/drivers/src/gw1ns4c_timer.o ./src/library/libraries/drivers/src/gw1ns4c_uart.o
./src/library/libraries/drivers/src/gw1ns4c_wdog.o
./src/library/libraries/cmsis/cm3/device_support/startup/gmd/startup_gw1ns4c.o
./src/library/libraries/cmsis/cm3/device_support/system_gw1ns4c.o
./src/library/libraries/cmsis/cm3/core_support/gmd/core_cm3.o
Finished building target: cm3_demo.elf

Invoking: GNU ARM Cross Create Flash Image
arm-none-eabi-objcopy -O binary "cm3_demo.elf" "cm3_demo.bin"
Finished building: cm3_demo.bin

Invoking: GNU ARM Cross Print Size
arm-none-eabi-size --format=berkeley "cm3_demo.elf"
text      data      bss      dec      hex filename
29988     1764       392    32144    7d90 cm3_demo.elf
Finished building: cm3_demo.siz

10:44:04 Build Finished. 0 errors, 0 warnings. (took 15s.231ms)

```

5.6.2 問題ビュー

問題ビューには、例えばコンパイル中に報告された警告とエラーが表示されます(図 5-13)。

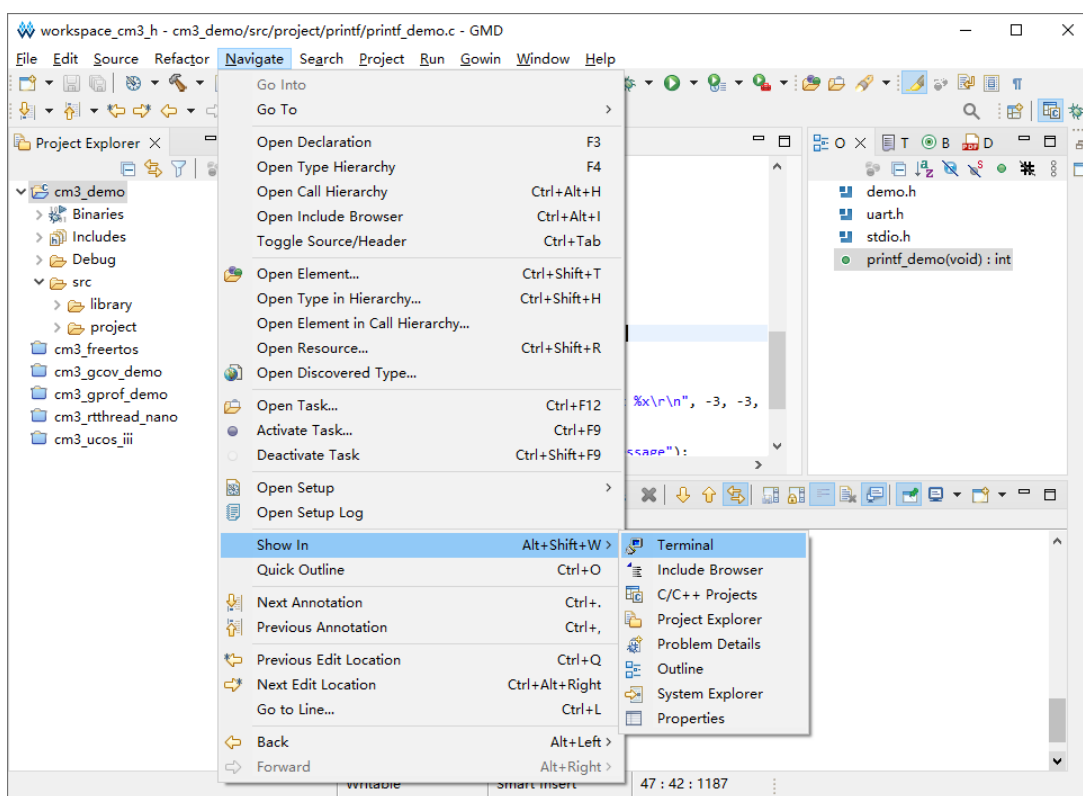
図 5-13 問題ビュー

Description	Resource	Path	Location	Type
3 errors, 0 warnings, 1 other				
✖ Errors (3 items)				
✖ 'mi' undeclared (first use in this function);	printf_demo.c	/cm3_demo/src/p...	line 43	C/C++ Prob...
✖ expected '=', ',', ';', 'asm' or '__attribute__'	printf_demo.c	/cm3_demo/src/p...	line 43	C/C++ Prob...
✖ make: *** [src/project/printf/subdir.mk:20: cm3_demo				C/C++ Prob...
ℹ Infos (1 item)				
ℹ each undeclared identifier is reported onl	printf_demo.c	/cm3_demo/src/p...	line 43	C/C++ Prob...

5.6.3 ターミナル・ビュー

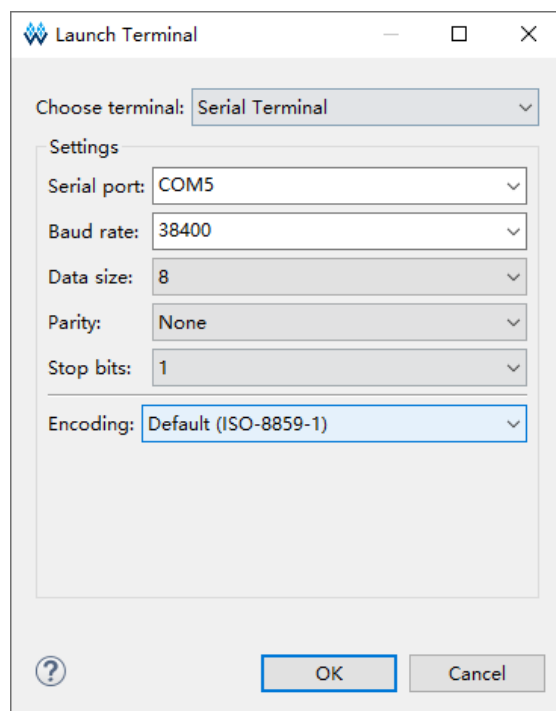
「Navigate > Show In > Terminal」 オプション：ターミナルを起動します(図 5-14)。

図 5-14 ターミナルを起動



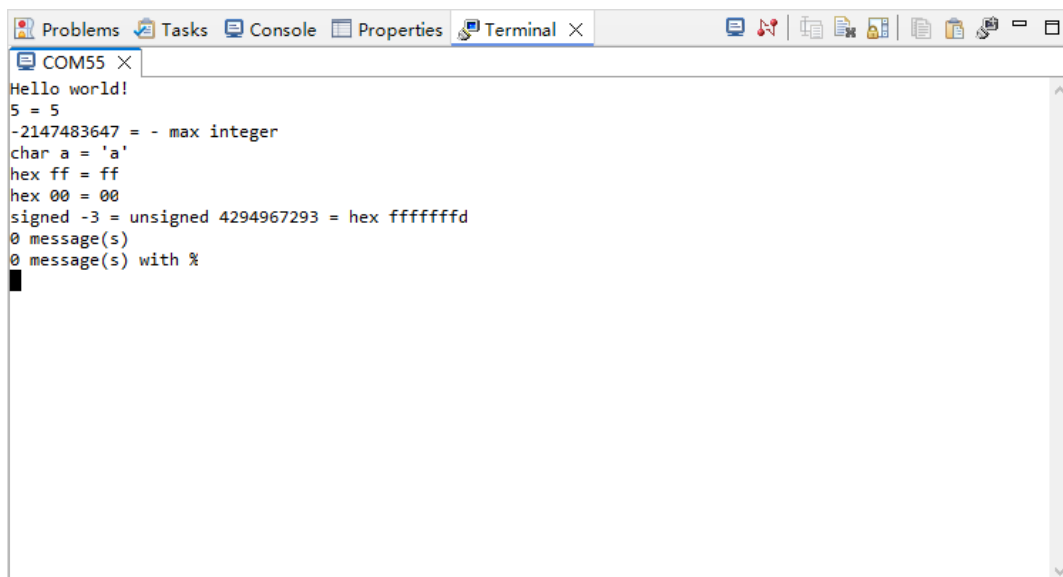
Local Terminal、SSH Terminal、Serial Terminal、Telnet Terminal など、さまざまな種類のターミナルを設定できます。例えば、よく使用されるシリアルターミナルを図 5-15 に示します。

図 5-15 シリアルターミナル



シリアルターミナルの情報出力の例を図 5-16 に示します。

図 5-16 シリアルターミナルの出力

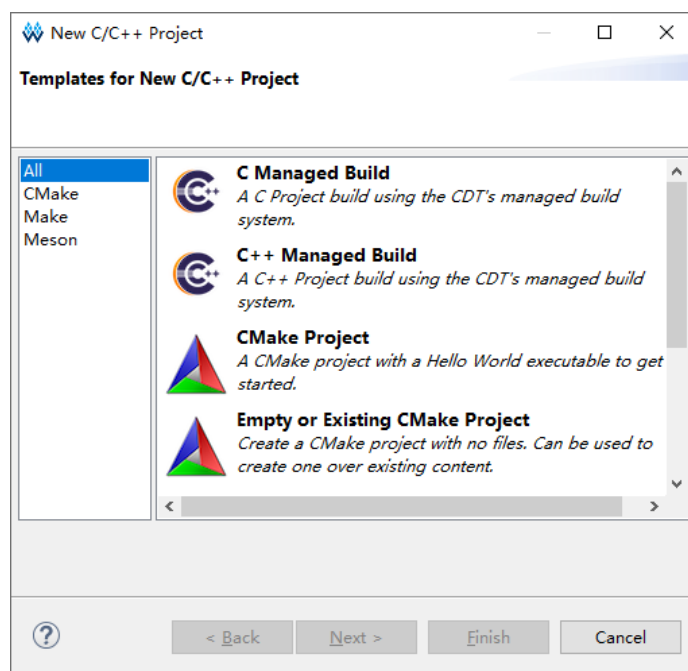


6 プロジェクトの新規作成

6.1 プロジェクトの新規作成

メニューバーの「File > New > C/C++ Project」オプションまたはツールバーの「New > C/C++ Project」()オプションをクリックしてC/C++プロジェクトを作成します(図 6-1)。

図 6-1 プロジェクトの新規作成



よく使われるプロジェクト・タイプは、Cプロジェクト作成用の「C Managed Build」とC++プロジェクト作成用の「C++ Managed Build」です。

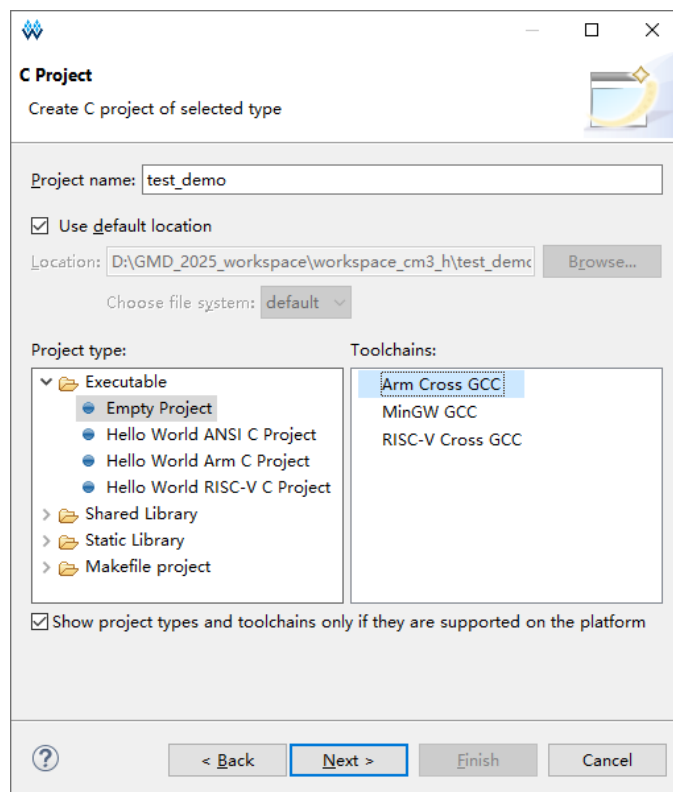
例えば、Arm MCU と RISC-V MCU の場合の C プロジェクト新規作成は次のとおりです。

6.1.1 プロジェクト(Arm MCU)の新規作成

1. プロジェクト・タイプとツールチェーンを選択

プロジェクトの名前と場所を入力し、プロジェクトのタイプとツールチェーンを選択します(図 6-2)。

図 6-2 プロジェクト・タイプとツールチェーンの選択



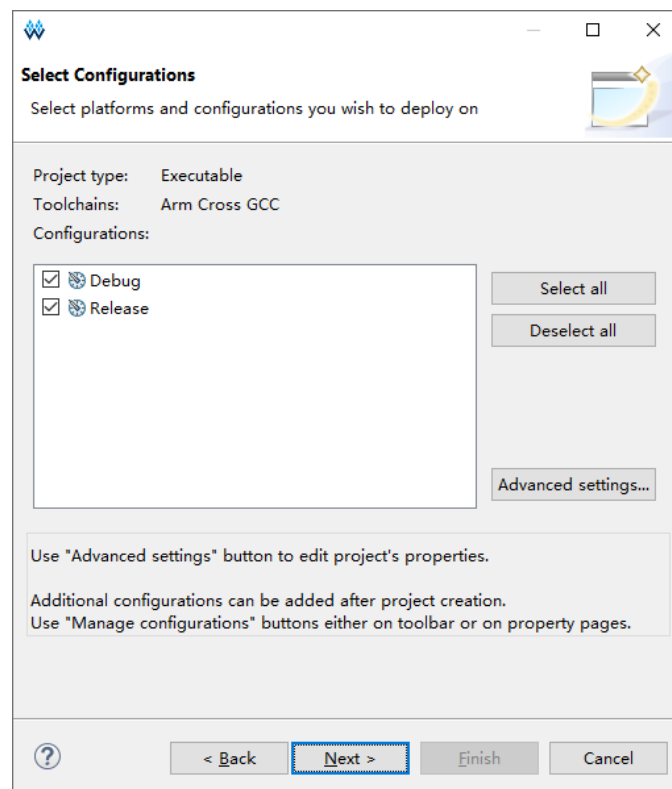
例 :

- Project name : test_demo
- Location : Use default location。つまり、現在のワークスペースを使用します
- Project type : Empty Project
- Toolchains : Arm Cross GCC

2. プラットフォームと構成を選択

プラットフォームと構成(Debug と Release を含む)を選択します(図 6-3)。

図 6-3 プラットフォームと構成の選択



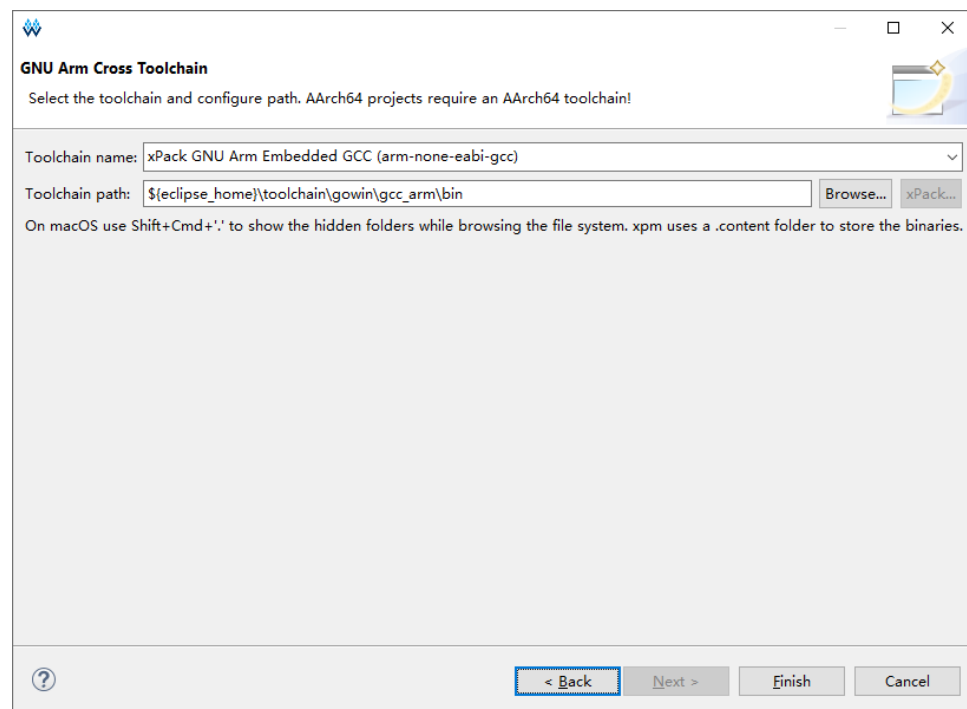
例：

Configurations において Debug と Release をチェックします。

3. ソフトウェア・ツールチェーンおよびそのパスを選択

特定の **Arm GCC** クロスコンパイル・ツールチェーンとそのパスを選択します(図 6-4)。**GMD** ソフトウェアはツールチェーンを統合し、パスを構成しています。したがって、通常はデフォルトの選択で問題ありません。

図 6-4 ツールチェーンおよびそのパスの選択



例：

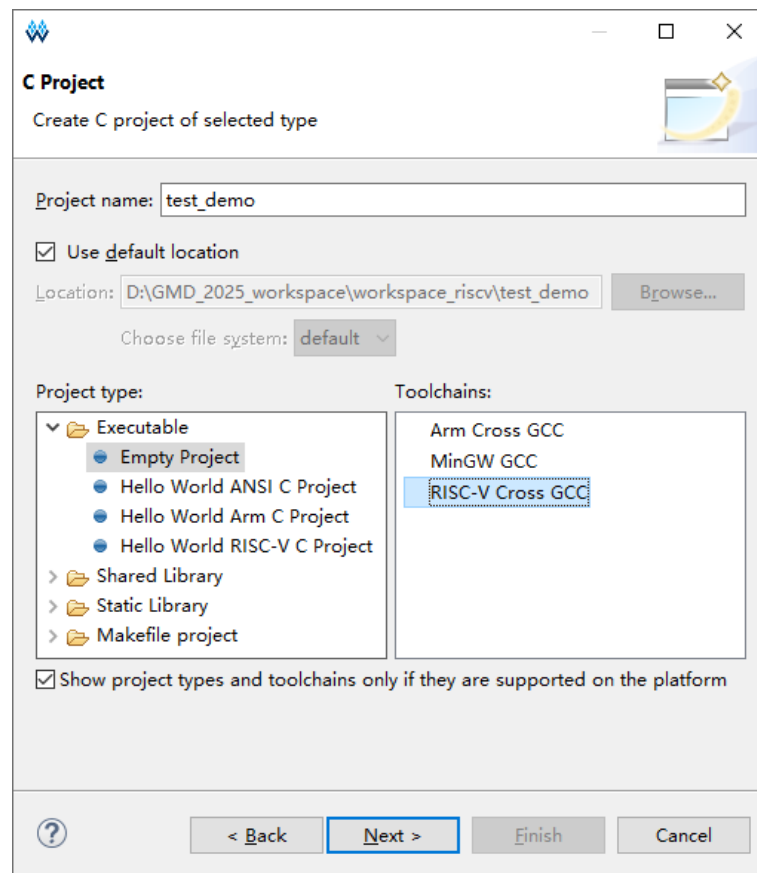
- Toolchain name : xPack GNU Arm Embedded GCC (arm-none-eabi-gcc) ;
- Toolchain path : \${eclipse_home}¥toolchain¥gowin¥gcc_arm¥bin。

6.1.2 プロジェクト(RISC-V MCU)の新規作成

1. プロジェクト・タイプとツールチェーンを選択

プロジェクトの名前と場所を入力し、プロジェクトのタイプとツールチェーンを選択します(図 6-5)。

図 6-5 プロジェクト・タイプとツールチェーンの選択

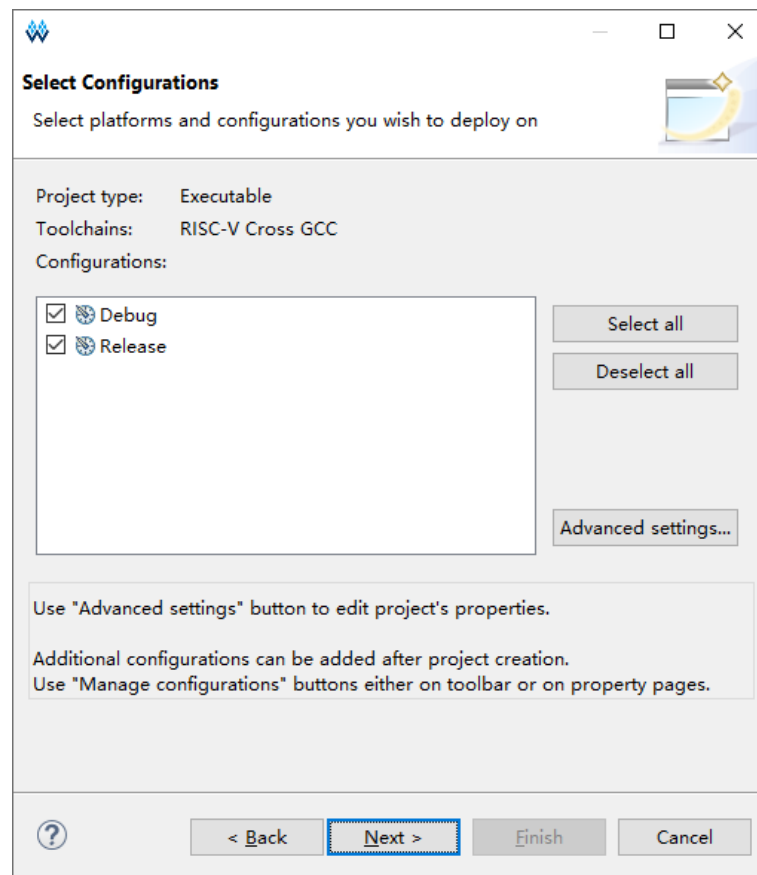


例：

- Project name : test_demo
 - Location : Use default location。つまり、現在のワークスペースを使用します
 - Project type : Empty Project
 - Toolchains : RISC-V Cross GCC
2. プラットフォームと構成を選択

プラットフォームと構成(Debug と Release を含む)を選択します(図 6-6)。

図 6-6 プラットフォームと構成の選択



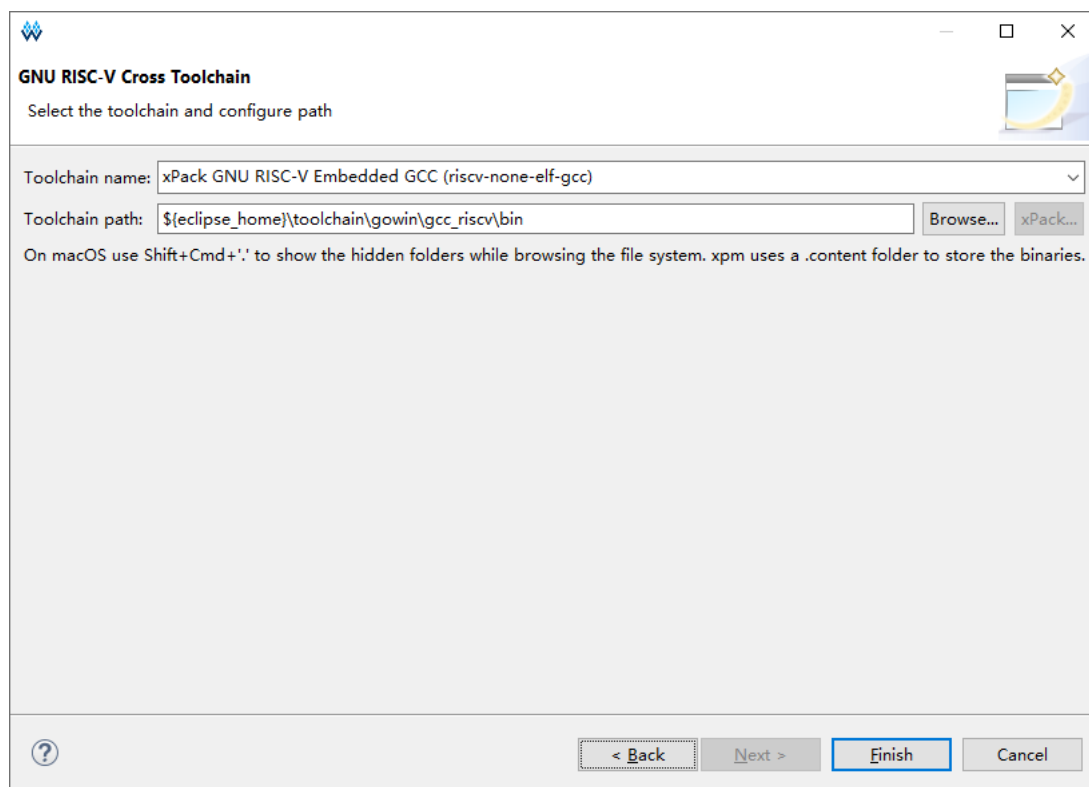
例：

Configurations において **Debug** と **Release** をチェックします。

3. ソフトウェア・ツールチェーンおよびそのパスを選択

特定の **RISC-V GCC** クロスコンパイル・ツールチェーンとそのパスを選択します(図 6-7)。**GMD** ソフトウェアはツールチェーンを統合し、パスを構成しています。したがって、通常はデフォルトの選択で問題ありません。

図 6-7 ツールチェーンおよびそのパスの選択



例：

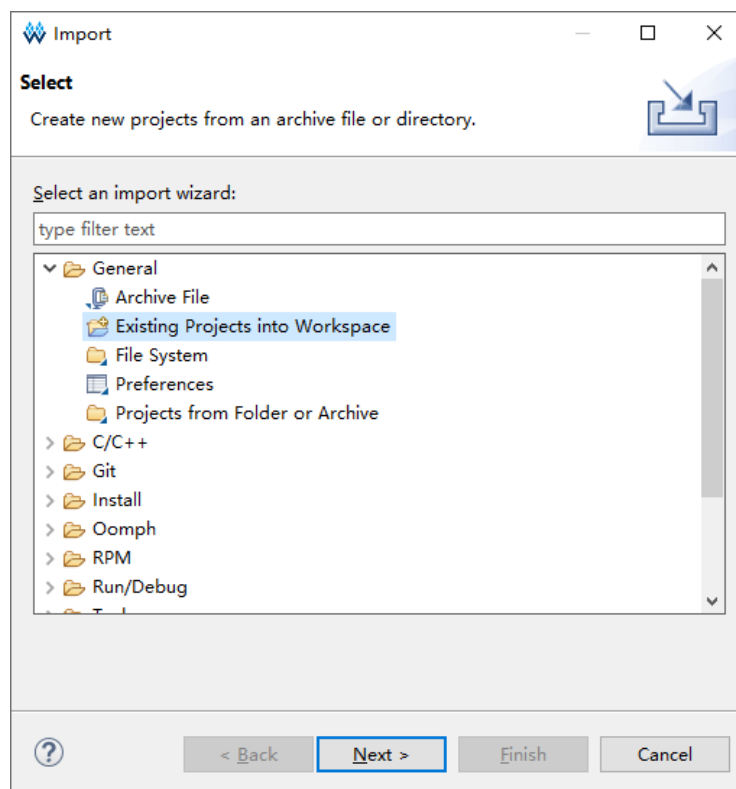
- Toolchain name : xPack GNU RISC-V Embedded GCC (riscv-none-elf-gcc) ;
- Toolchain path : \${eclipse_home}¥toolchain¥gowin¥gcc_riscv¥bin。

6.2 既存のプロジェクトのインポート

Gowin は、すべての Gowin MCU 製品向けのリファレンス・デザインを提供しており、ユーザーはこれを直接インポートできます。既存プロジェクトのインポート手順は次のとおりです。

1. 「File > Import…」をクリックして、既存プロジェクトのインポートを開始します。
2. 「Existing Projects into Workspace」を選択します(図 6-8)。

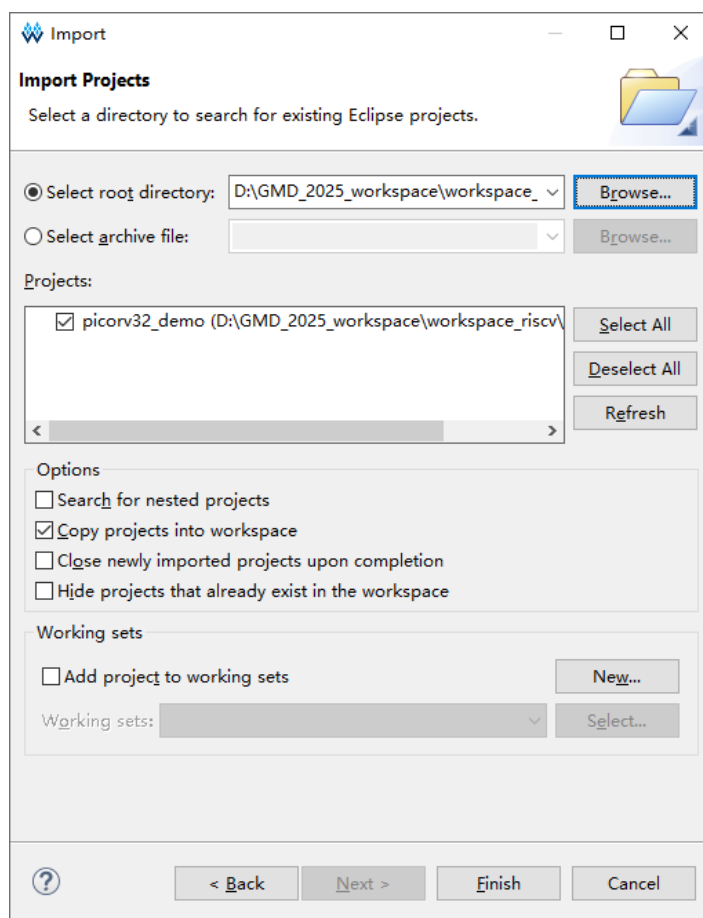
図 6-8 インポート方法の選択



3. インポートする既存プロジェクトを選択し、オプションを設定

「Browse」をクリックして、インポートするプロジェクトパスを選択します(図 6-9)。「Copy projects into workspace」オプションがチェックされている場合、インポートしたプロジェクトは現在のワークスペースにコピーされます。

図 6-9 インポートするプロジェクトの選択



7 旧バージョンの GMD で作成されたプロジェクトのインポート

GMD 2025.01 以降のバージョンでは、ツールチェーンが大幅に更新されているため、新しい GMD ソフトウェアで V1.2 以前の旧バージョンの GMD で作成されたプロジェクトを使用する場合、または旧バージョンの Gowin MCU SDK を使用する場合は、この章の内容を参照して修正する必要があります。

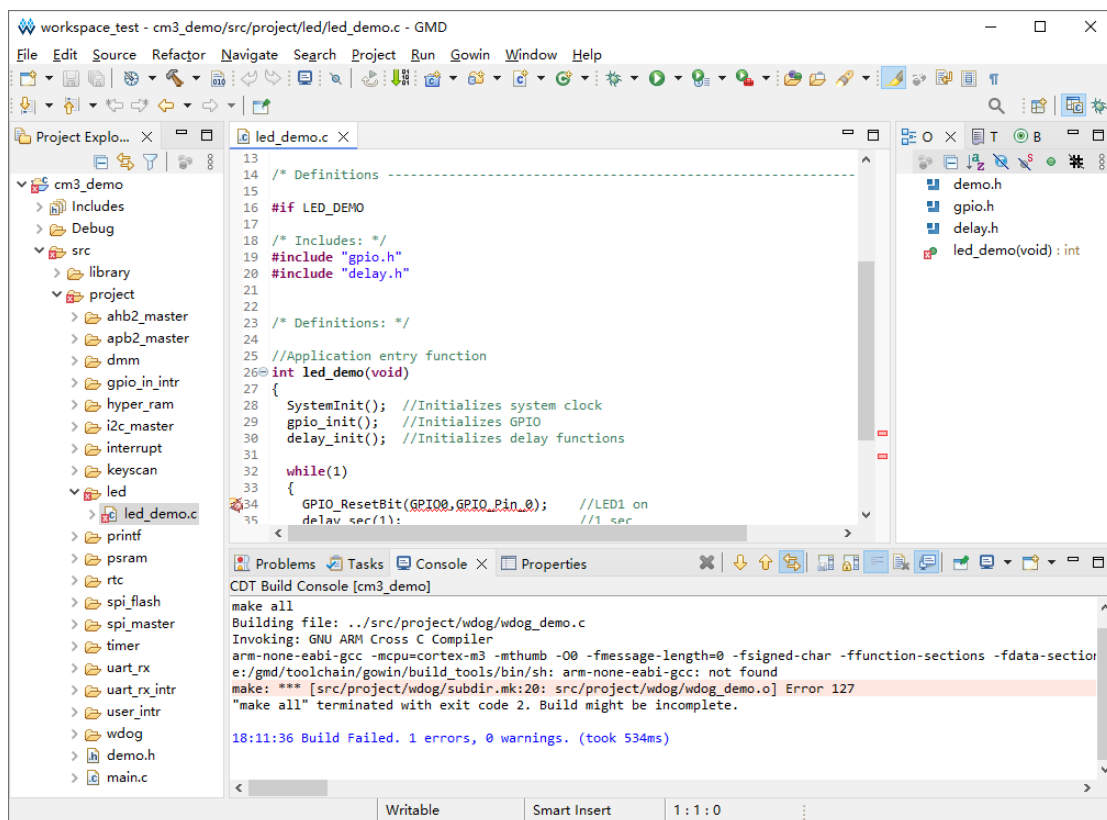
GMD 2025.01 以降のバージョンのソフトウェアに旧バージョンの GMD のプロジェクトをインポートする場合、または旧バージョンの MCU SDK（旧 SDK とは、GMD 2025.01 より前にリリースされた SDK を指します）で作成されたプロジェクトを使用する場合、プロジェクト構成で旧バージョンの GCC が使用されているため、対応するツールチェーンが見つからないとき、コンパイル・エラーなどの問題が発生します。

以下の方法を参照しながら、インポートした Arm MCU プロジェクトと RISC-V MCU プロジェクトのツールチェーンを再構成する必要があります。

7.1 インポートした Arm MCU プロジェクトの変更

1. 「6.2 既存のプロジェクトのインポート」を参照して旧バージョンの GMD で作成された Arm MCU プロジェクトをインポートします。
2. プロジェクトをコンパイルする際、対応する旧バージョンのツールチェーンが見つからないため、コンパイル・エラーが表示されます(図 7-1)。

図 7-1 Arm プロジェクトをインポートした後のコンパイル・エラー

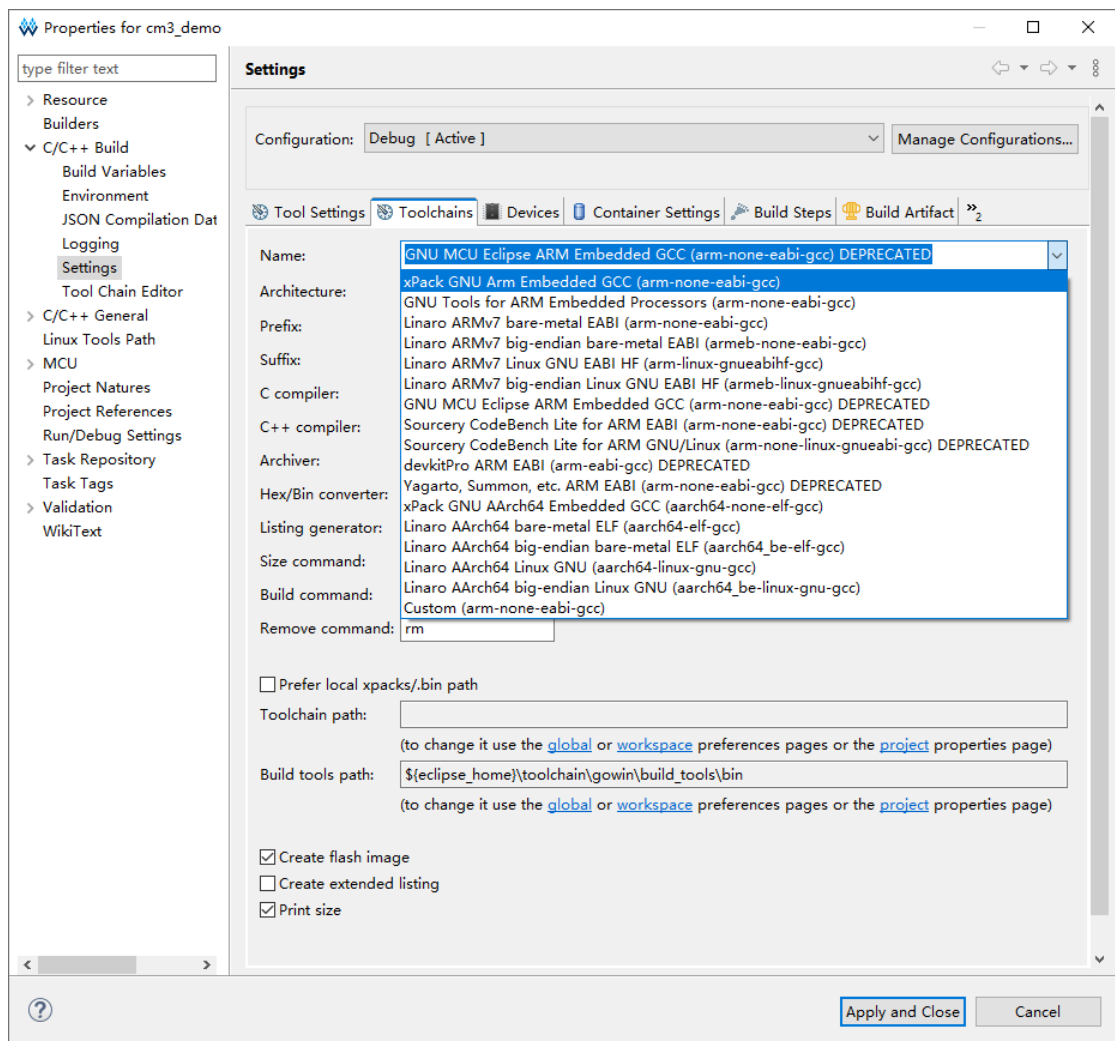


3. ツールチェーンを変更し、利用可能な Arm ツールチェーンを選択します。

現在のプロジェクトを選択し、右クリックして「**Properties > C/C++ Build > Settings > Toolchains**」を選択し、ツールチェーンを変更します。

図 7-2 に示すように、「Name」ドロップダウンリストから **xPack GNU Arm Embedded GCC (arm-none-eabi-gcc)** ツールチェーンを選択します。

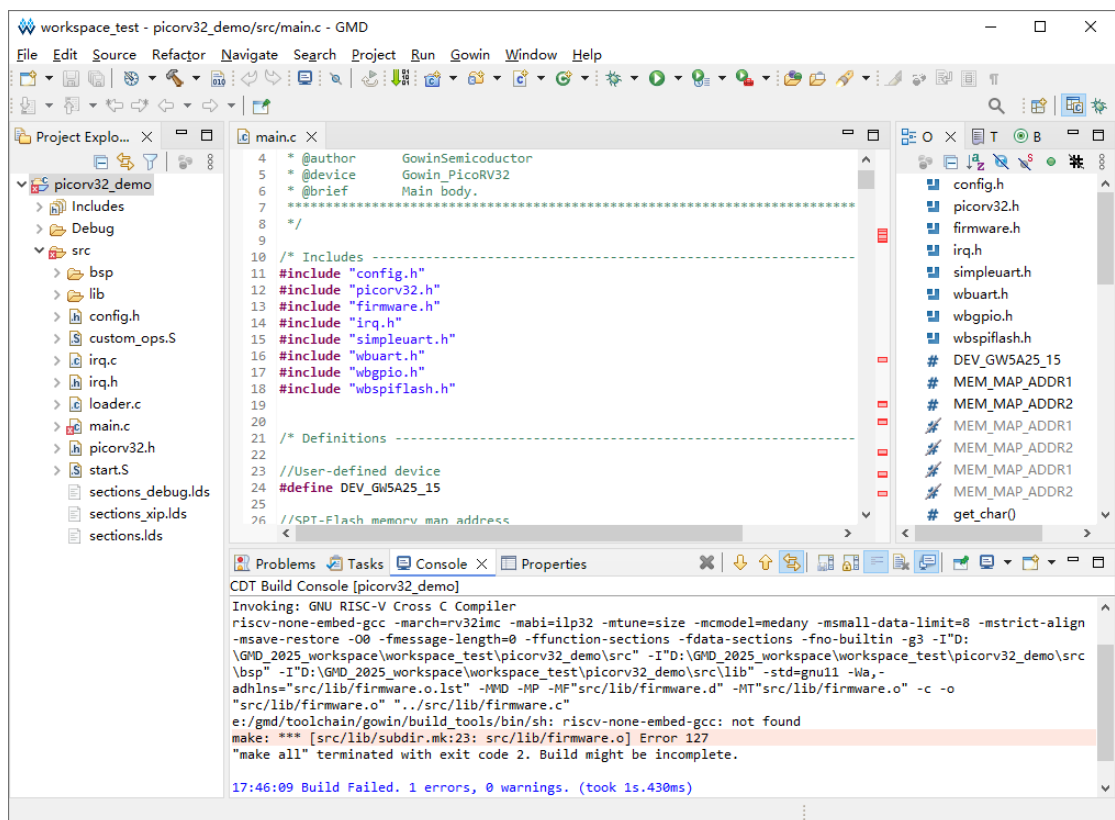
図 7-2 ツールチェーンの設定



7.2 インポートした RISC-V MCU プロジェクトの変更

1. 「6.2 既存のプロジェクトのインポート」を参照して旧バージョンの GMD で作成された RISC-V MCU プロジェクトをインポートします。
2. プロジェクトをコンパイルする際、対応する旧バージョンのツールチェーンが見つからないため、コンパイル・エラーが表示されます(図 7-3)。

図 7-3 RISC-V プロジェクトをインポートした後のコンパイル・エラー

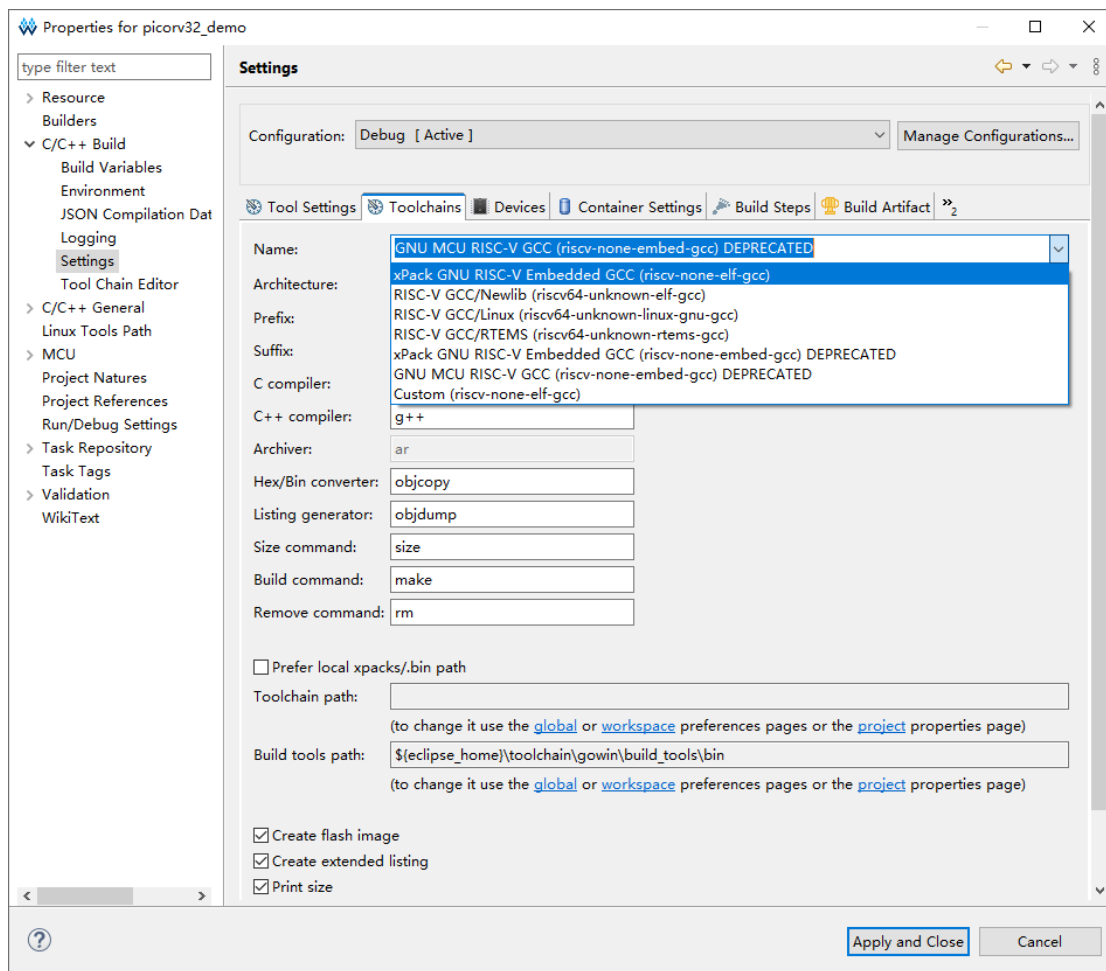


3. ツールチェーンを変更し、利用可能な RISC-V ツールチェーンを選択します。

現在のプロジェクトを選択し、右クリックして「Properties > C/C++ Build > Settings > Toolchains」を選択し、ツールチェーンを変更します。

図 7-4 に示すように、「Name」ドロップダウンリストから xPack GNU RISC-V Embedded GCC (riscv-none-elf-gcc) ツールチェーンを選択します。

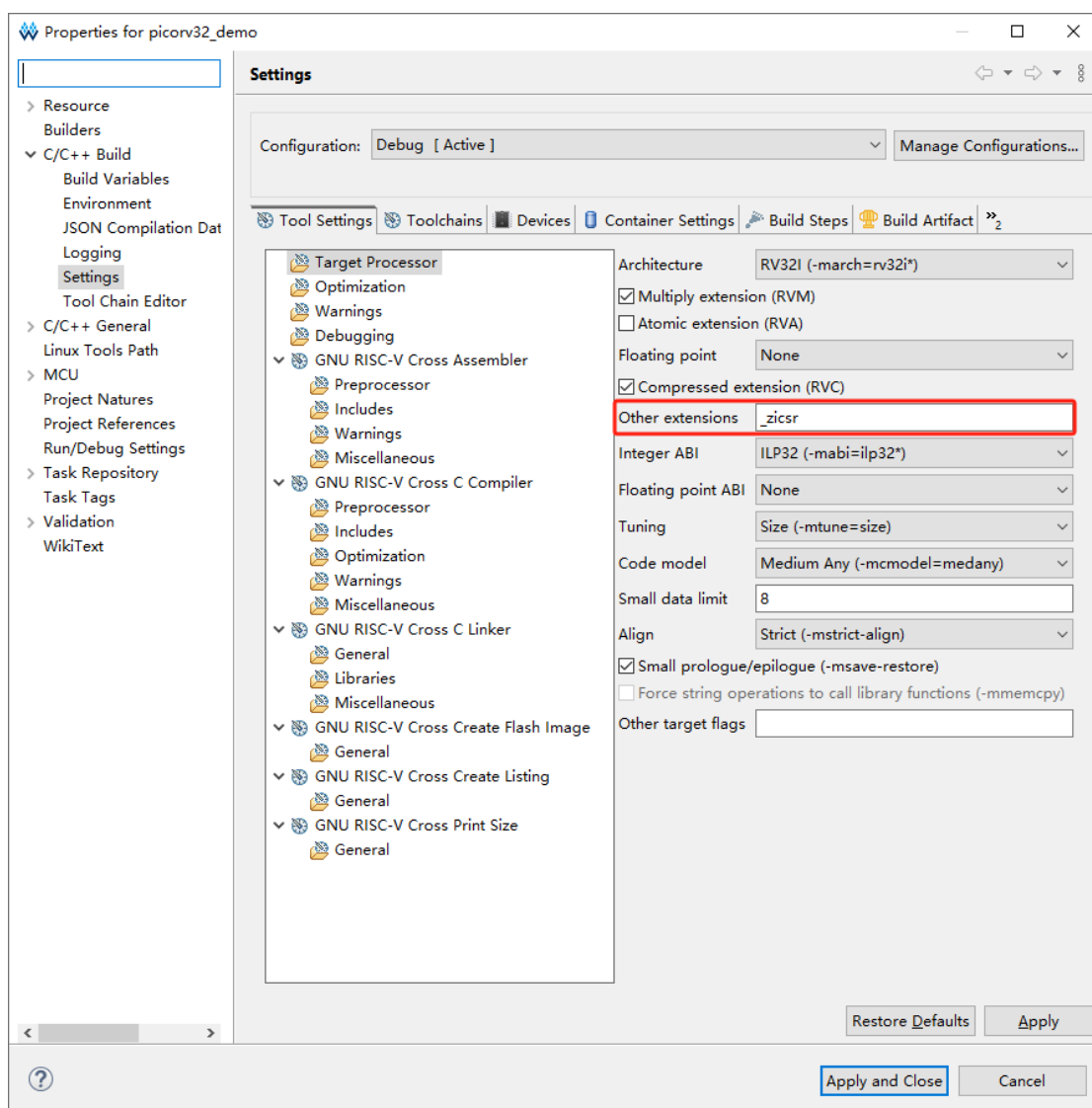
図 7-4 RISC-V ツールチェーンの設定



4. コンパイル・オプションを変更し、追加の拡張命令「`_zicsr`」を入力します。

現在のプロジェクトを選択し、右クリックして「**Properties > C/C++ Build > Settings > Tool Settings > Target Processor**」を選択し、「**Other extensions**」フィールドに「`_zicsr`」を入力します(図 7-5)。

図 7-5 コンパイル・オプションの設定



8 プロジェクトのコンパイル

GMD ソフトウェアは複数のツールチェーンを統合しており、Arm MCU および RISC-V MCU プロジェクトのコンパイルをサポートします。

8.1 コンパイルツール

GMD ソフトウェアは、Gowin GNU ツールチェーンを統合しています。これは、Gowin の MCU 製品特徴向けに GNU ベーシックバージョンをベースに再開発されたものです。GNU ツールチェーンは、GNU プロジェクトが提供するソフトウェア開発ツールチェーンの完全なセットです。GNU ツールチェーンには、コンパイラ、デバッガ、リンカ、ライブラリファイルなどソフトウェア開発と構築に必要なツールが含まれています。GNU ツールチェーンは、オープンソース、クロスプラットフォーム、高度なカスタマイズ性、および強力な機能により、世界中の開発者コミュニティで広く使用されている開発ツールセットとなっています。

GMD ソフトウェアでプロジェクトを作成する際、Arm MCU プロジェクトを作成する場合は、ツールチェーン「xPack GNU Arm Embedded GCC (arm-none-eabi-gcc)」を選択できます(図 8-1)。RISC-V MCU プロジェクトを作成する場合は、ツールチェーン「xPack GNU RISC-V Embedded GCC (riscv-none-elf-gcc)」を選択できます(図 8-2)。

図 8-1 Arm 用ツールチェーン

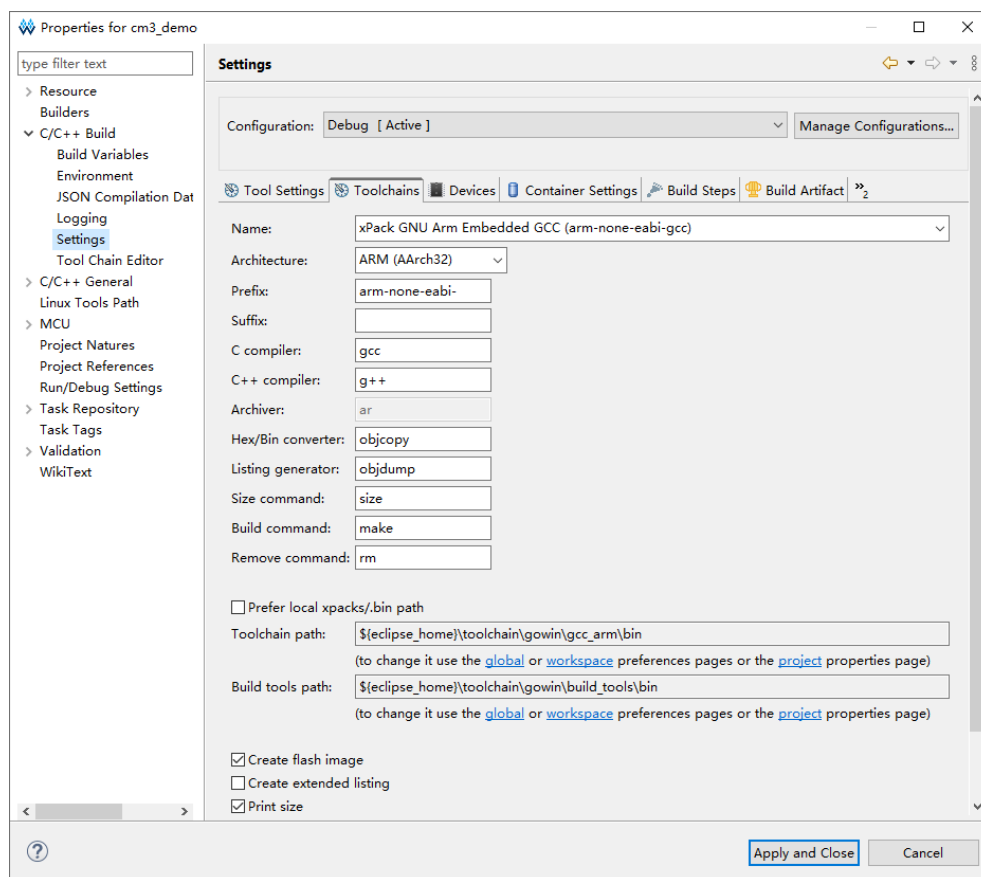
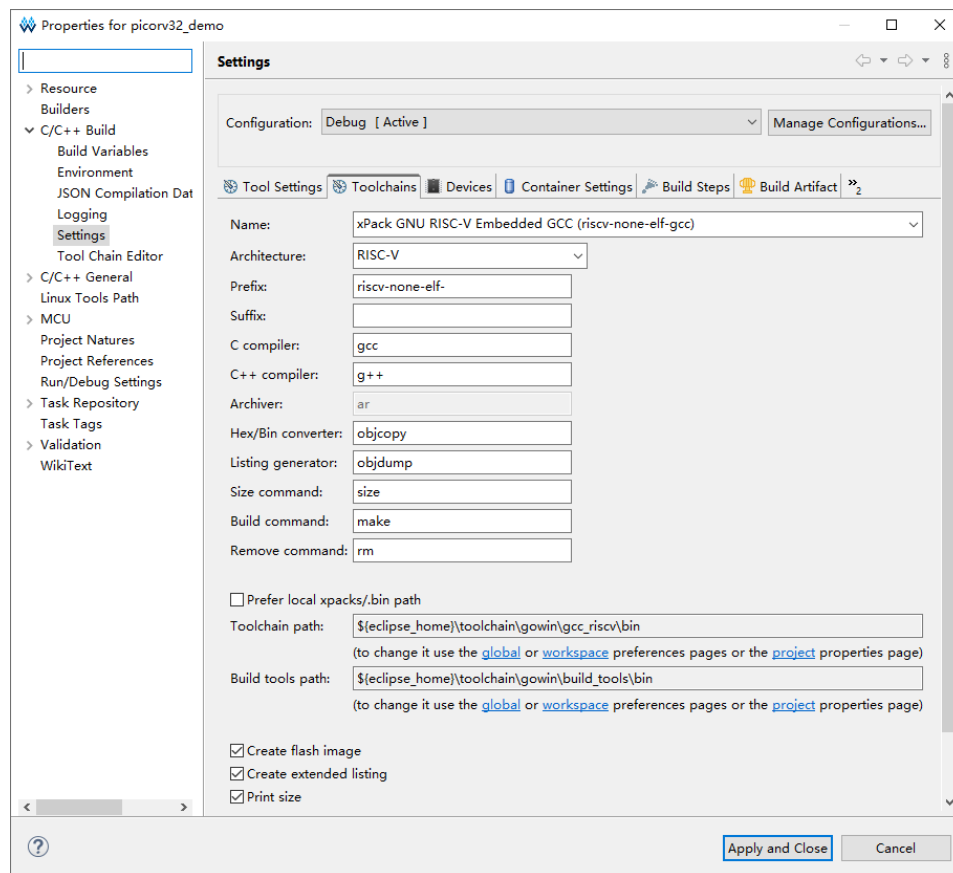


図 8-2 RISC-V 用ツールチェーン



8.2 プロジェクトの設定

プロジェクトを作成後、コンパイルする前に、ユーザーのアプリケーション要件に応じて、プロジェクトのさまざまなコンパイルおよびリンクのオプションを設定する必要があります。

現在のプロジェクトを右クリックして「**Properties > C/C++ Build > Settings > Tool Settings**」を選択してプロジェクトを設定します。例えば、Arm MCU プロジェクトの設定は図 8-3 に、RISC-V MCU プロジェクトの設定は図 8-4 に示されています。

図 8-3 Arm MCU プロジェクトの設定

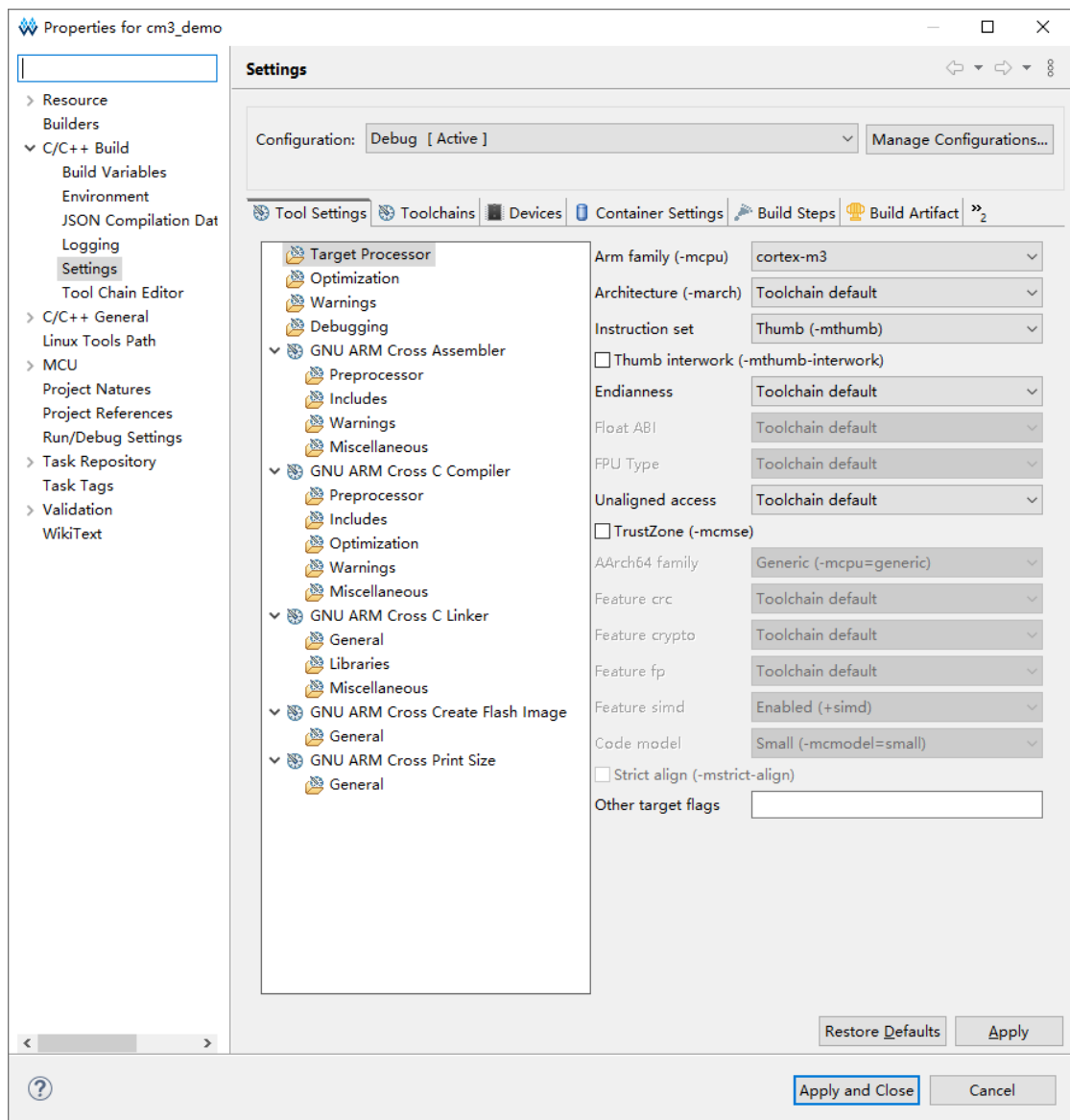
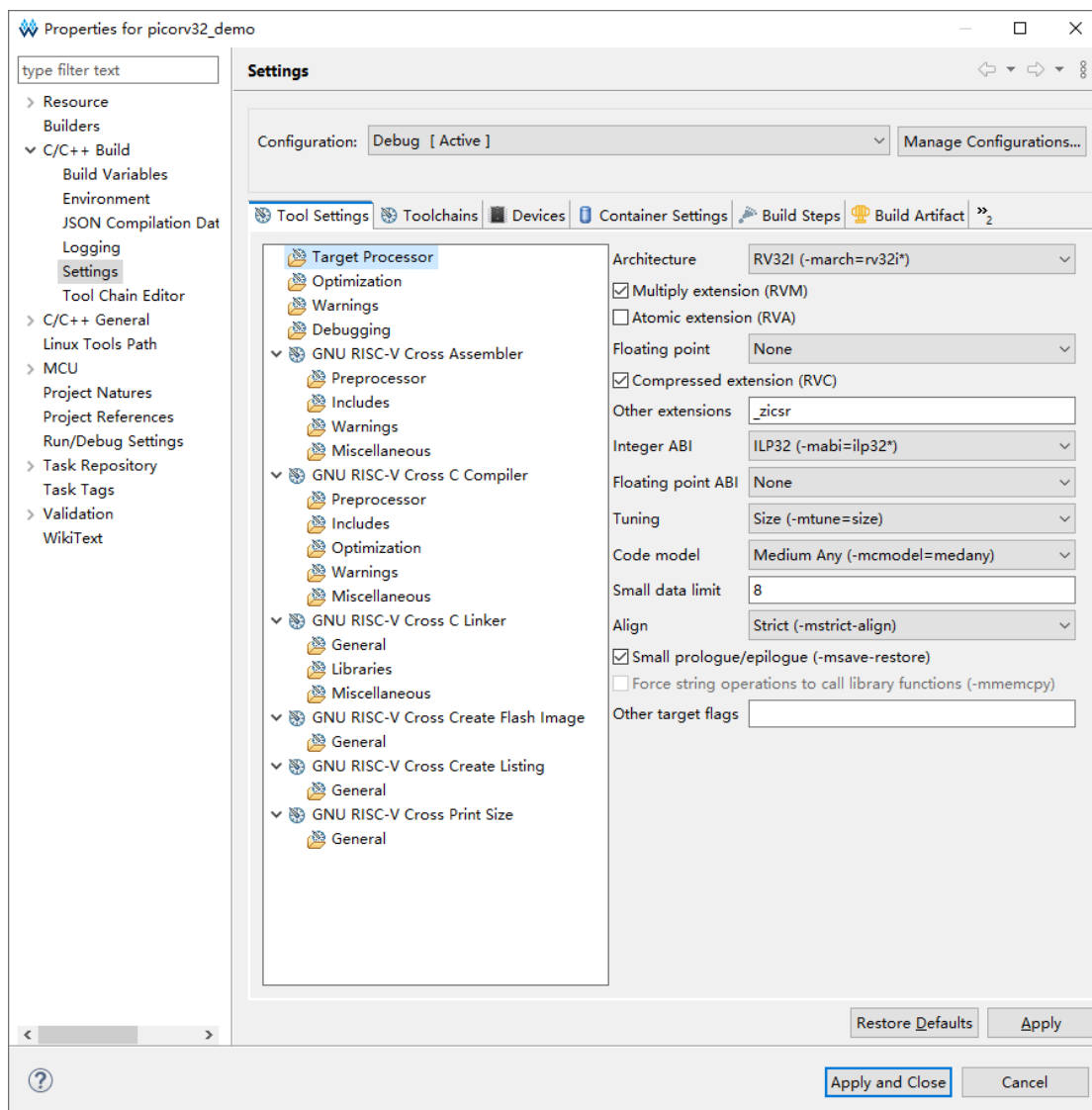


図 8-4 RISC-V MCU プロジェクトの設定



Arm MCU プロジェクトと RISC-V MCU プロジェクトの設定は以下に示すとおりです。

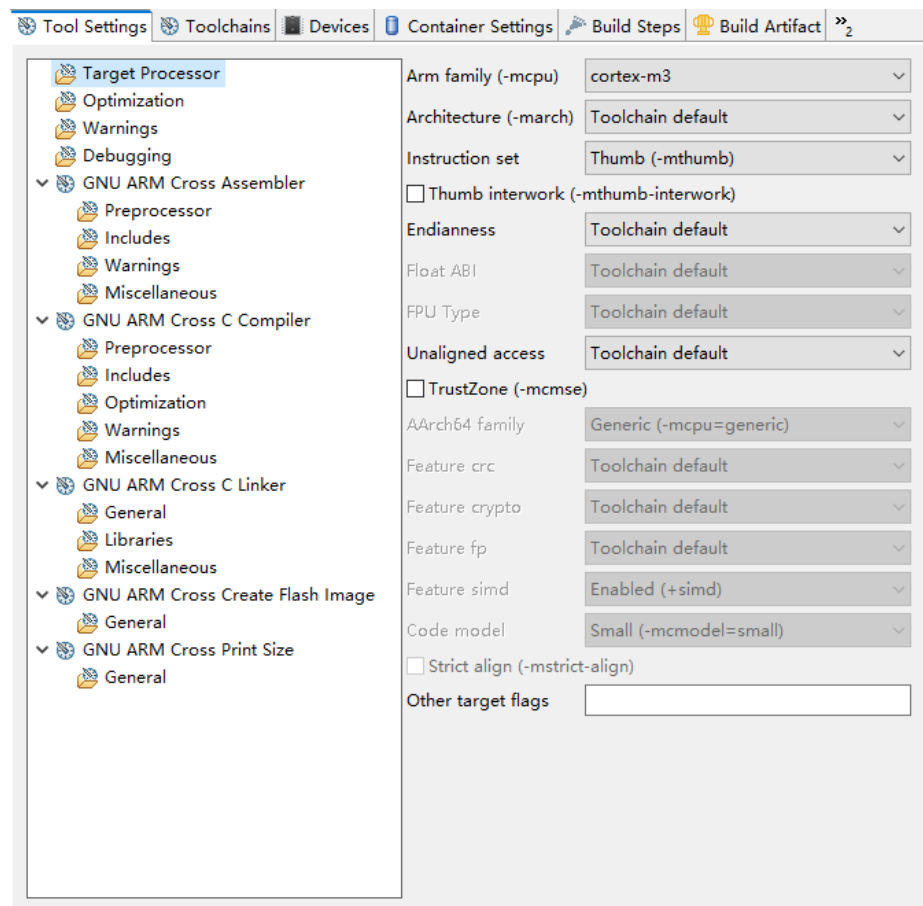
8.2.1 Arm MCU プロジェクトの設定

Gowin_EMPU(GW1NS-4C)を例に説明します。

Target Processor

Arm ターゲットプロセッサを設定します(図 8-5)。

図 8-5 Target Processor



オプションの設定を表 8-1 に示します。

表 8-1 Target Processor

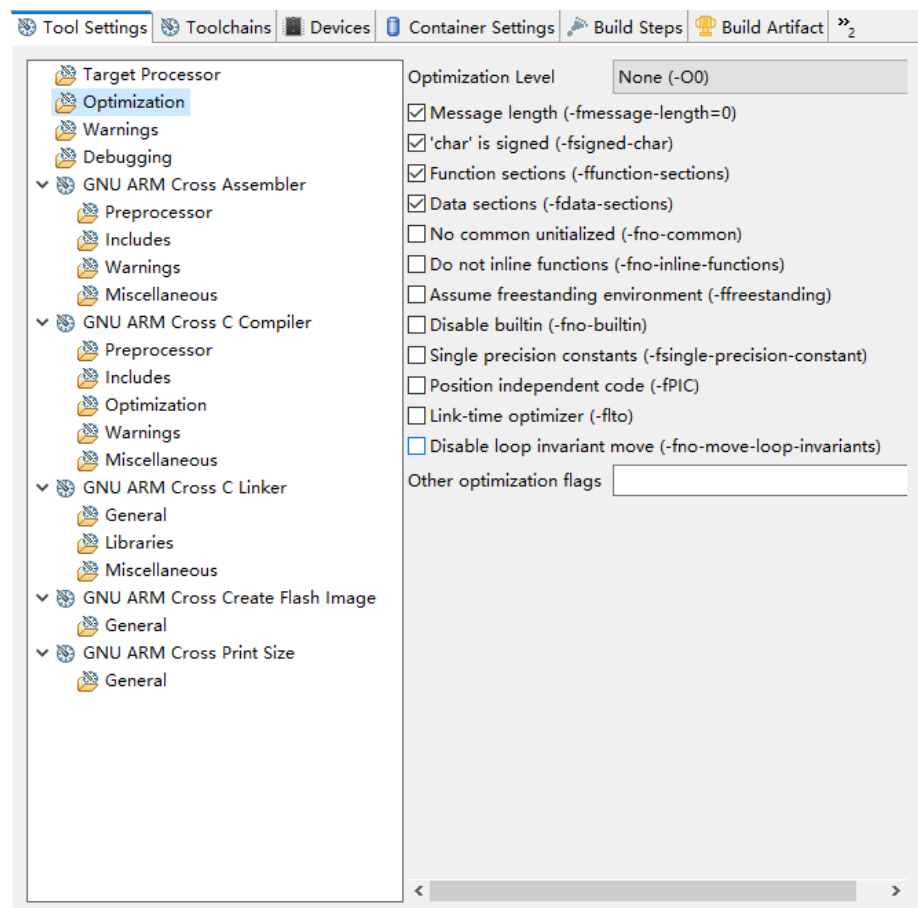
オプション	設定	説明
Arm family (-mcpu)	cortex-m3	Arm コアのモデル。Cortex-M1/M3/M4 をサポート
Architecture (-march)	Toolchain default	Arm アーキテクチャ
Instruction set	Thumb (-mthumb)	Arm 命令セット
Thumb interwork (-mthumb)	-	Thumb インターワーク。同一実行ファイルにおいて 32 ビットの ARM 命令と 16 ビットの Thumb 命令セットを混在可能。
Endianness	Toolchain default	リトルエンディアンまたはビッグエンディアンなどのバイトオーダー
Float ABI	-	浮動小数点 ABI のサポート
FPU Type	-	ハードウェア浮動小数点のサポート

オプション	設定	説明
Unaligned access	Toolchain default	非アラインアクセス
TrustZone (-mcmse)	-	コンパイラは TrustZone セキュリティ機能をサポートするコードを生成します
Other target flags	-	ターゲットプロセッサの追加コンパイル・オプション

Optimization

コンパイル速度、コードのパフォーマンス、およびコードのサイズのバランスをとるために、-O0、-O1、-O2、-O3、-Os、-Ofast、-Og、-Ozなどの最適化レベルを設定します(図 8-6)。

図 8-6 Optimization



オプションの設定を表 8-2 に示します。

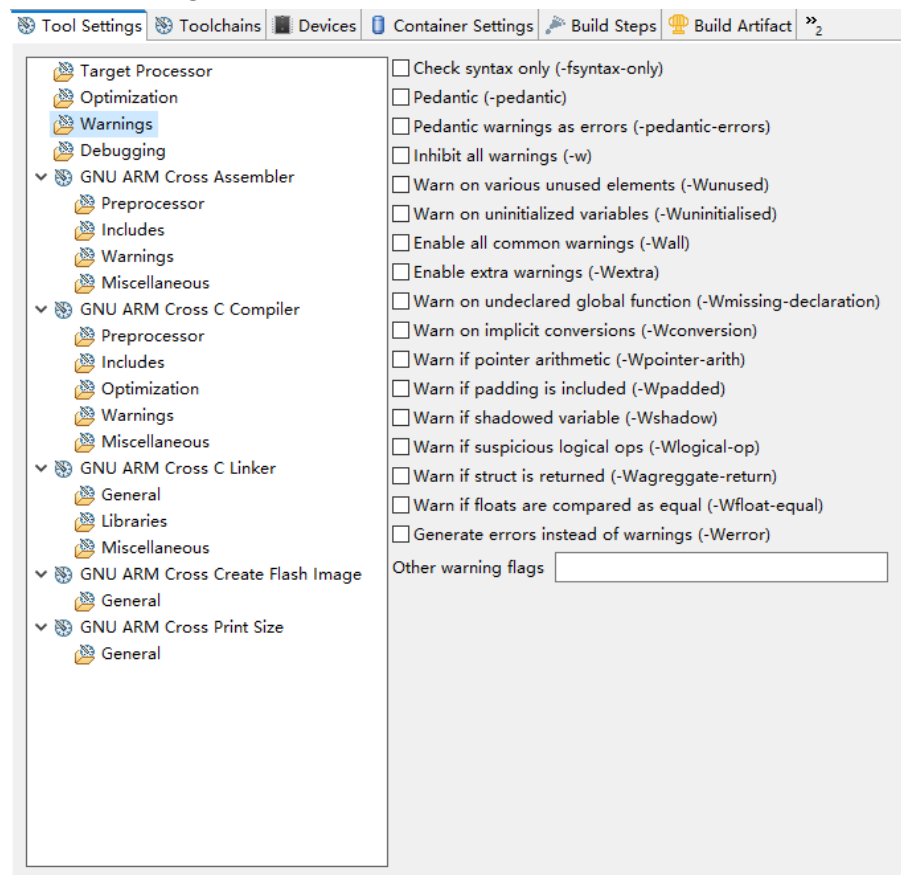
表 8-2 Optimization

オプション	設定
Optimization Level	None (-O0)
Message length (-fmessage-length = 0)	√
‘char’ is signed (-fsigned-char)	√
Function sections (-ffunction-sections)	√
Data sections (-fdata-sections)	√
No common uninitialized (-fno-common)	-
Do not inline functions (-fno-inline-functions)	-
Assume freestanding environment (-ffreestanding)	-
Disable builtin (-fno-builtin)	-
Single precision constants (-fsingle-precision-constant)	-
Position independent code (-fPIC)	-
Link-time optimizer (-flto)	-
Disable loop invariant move (-fno-move-loop-invariants)	-
Other optimization flags	-

Warnings

警告情報スイッチを設定します。これによりユーザーは潜在的な問題を早期に検出し、必要に応じて警告をエラーとして扱うことができます (図 8-7)。

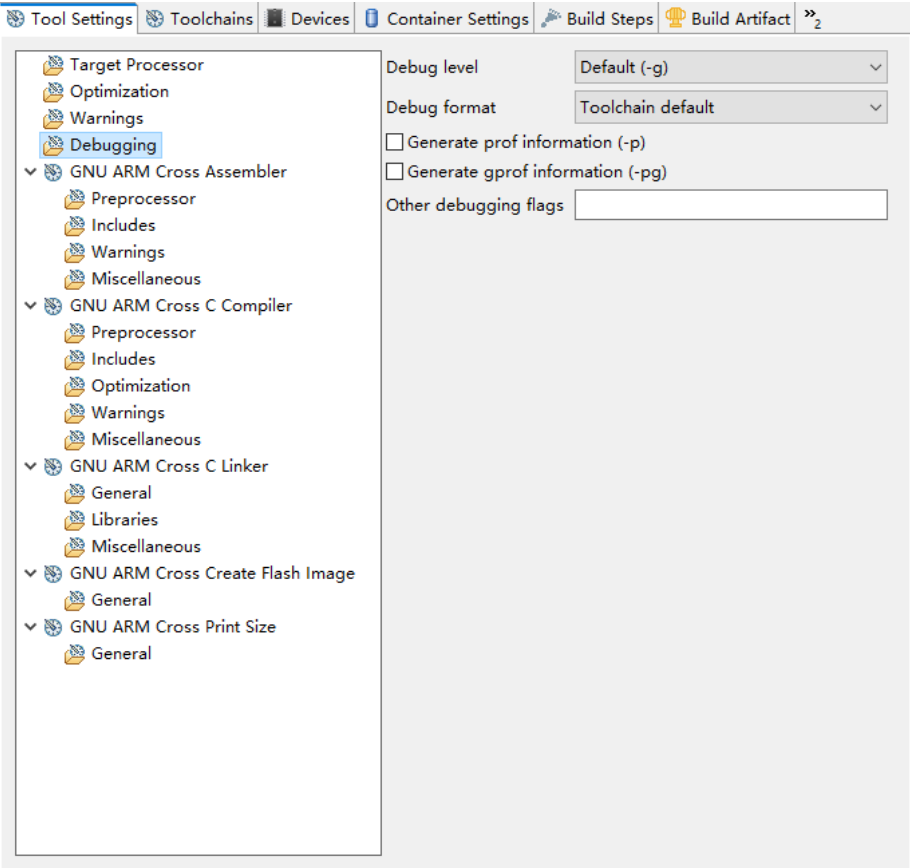
図 8-7 Warnings



Debugging

-g、-g1、-g3 などのデバッグ・レベルを設定します(図 8-8)。

図 8-8 Debugging



オプションの設定を表 8-3 に示します。

表 8-3 Debugging

オプション	設定
Debug level	Default (-g)
Debug format	Toolchain default
Generate prof information (-p)	-
Generate gprof information (-pg)	-
Other debugging flags	-

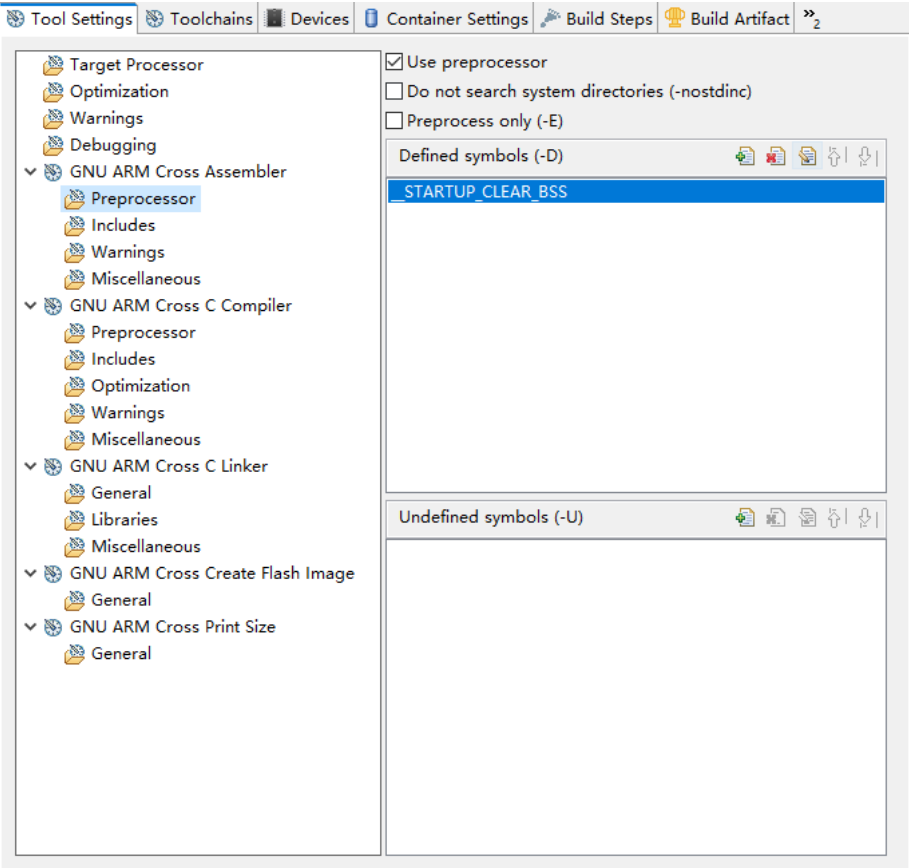
GNU ARM Cross Assembler

Preprocessor、Includes、Warnings、Miscellaneous を含む Arm MCU アセンブリのオプションを設定します。

1. Preprocessor

マクロ定義を設定します。プロジェクト関連のプリコンパイル済みマクロを追加するために使用されます(図 8-9)。

図 8-9 GNU ARM Cross Assembler > Preprocessor



オプションの設定を表 8-4 に示します。

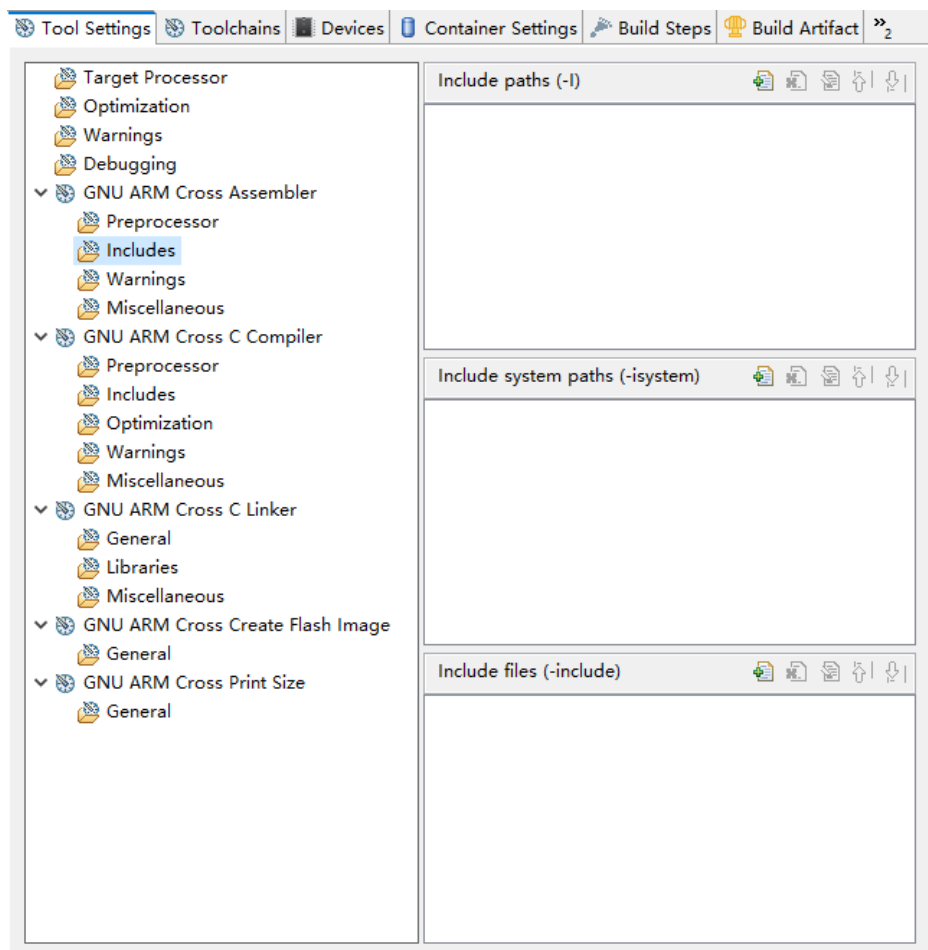
表 8-4 GNU ARM Cross Assembler > Preprocessor

オプション	設定
Use preprocessor	√
Do not search system directions (-nostdinc)	-
Preprocess only (-E)	-
Defined symbols (-D)	__STARTUP_CLEAR_BSS
Undefined symbols (-U)	-

2. Includes

ヘッダーファイルの検索パスを指定します。絶対パス、相対パス、または Eclipse 変数（\${workspace_loc}など）を入力できます(図 8-10)。

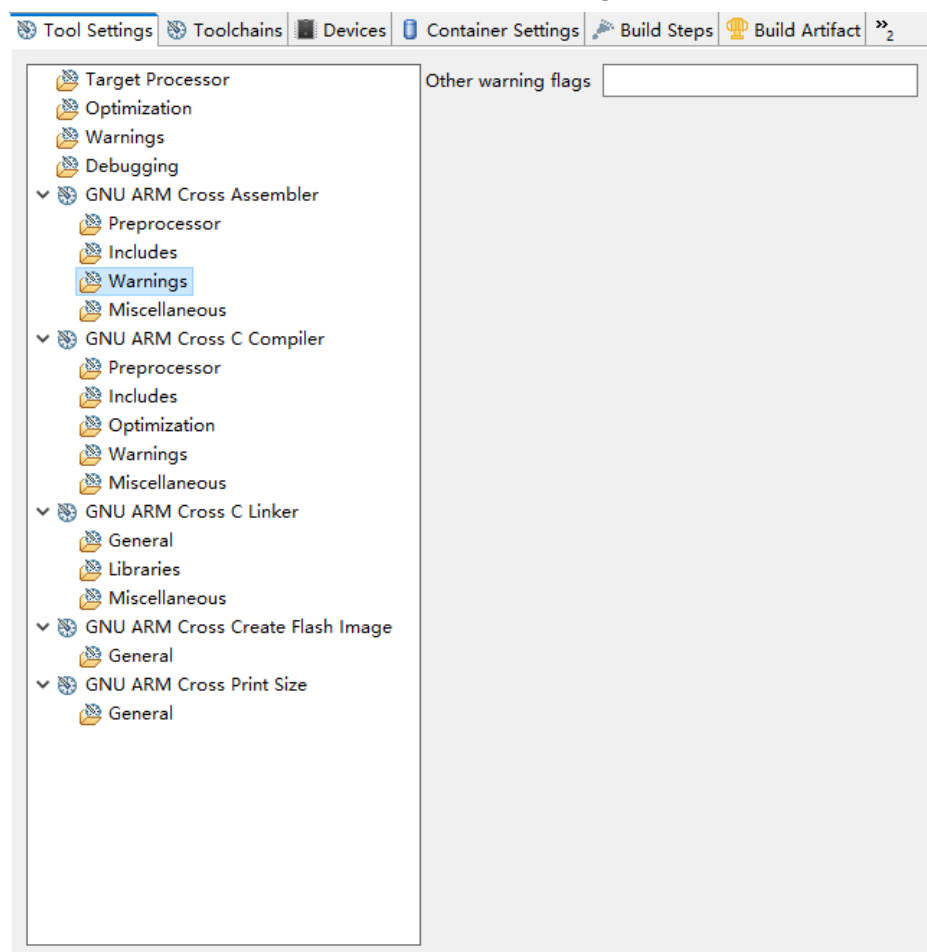
図 8-10 GNU ARM Cross Assembler > Includes



3. Warnings

必要に応じて追加の警告情報を入力します(図 8-11)。

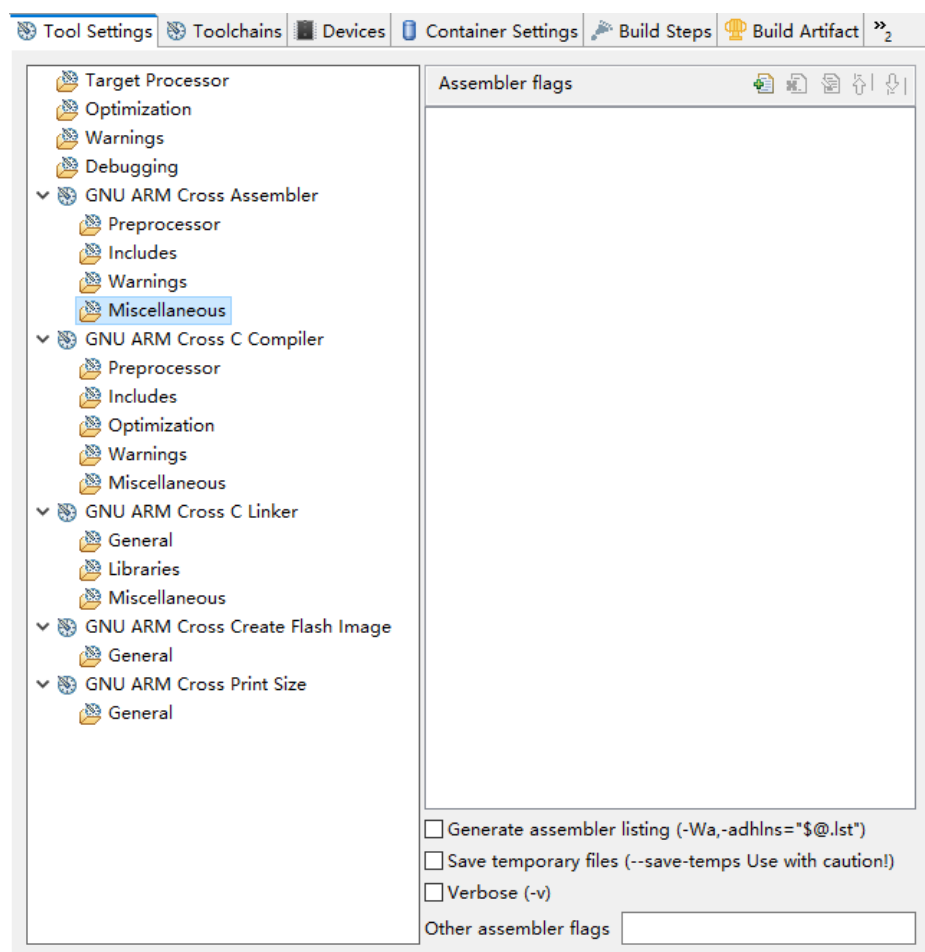
図 8-11 GNU ARM Cross Assembler > Warnings



4. Miscellaneous

必要に応じて追加のコンパイラ情報を入力します(図 8-12)。

図 8-12 GNU ARM Cross Assembler > Miscellaneous



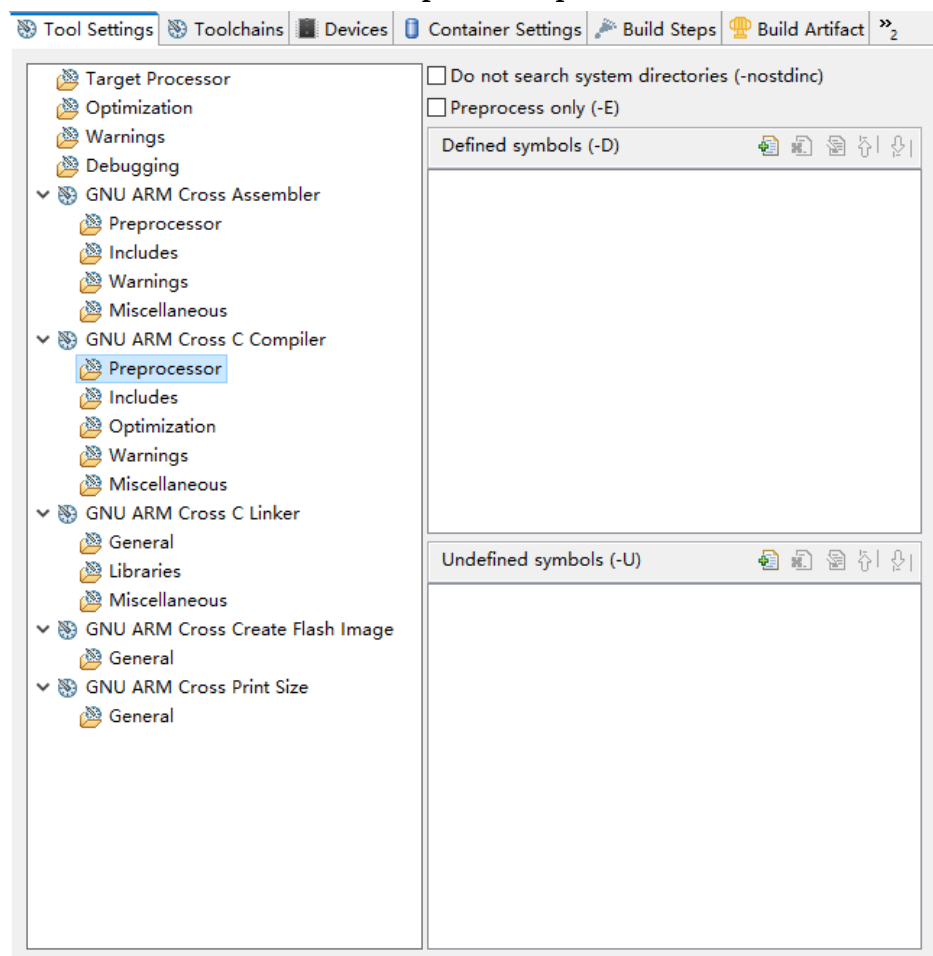
GNU ARM Cross C Compiler

Preprocessor、Includes、Optimization、Warnings、Miscellaneous を含む Arm MCU C コンパイル・オプションを設定します。

1. Preprocessor

マクロ定義を設定します。プロジェクト関連のプリコンパイル済みマクロを追加するために使用されます(図 8-13)。

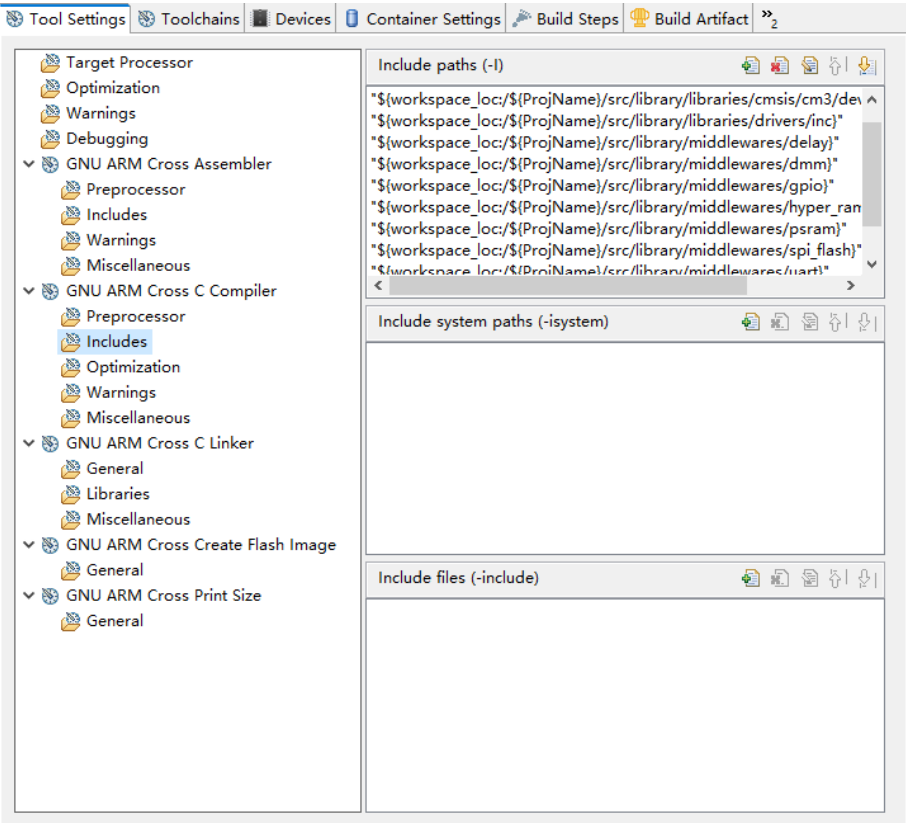
図 8-13 GNU ARM Cross C Compiler > Preprocessor



2. Includes

ヘッダーファイルの検索パスを指定します。絶対パス、相対パス、または Eclipse 変数（`${workspace_loc}`など）を入力できます(図 8-14)。

図 8-14 GNU ARM Cross C Compiler > Includes



オプションの設定を表 8-5 に示します。

表 8-5 GNU ARM Cross C Compiler > Includes

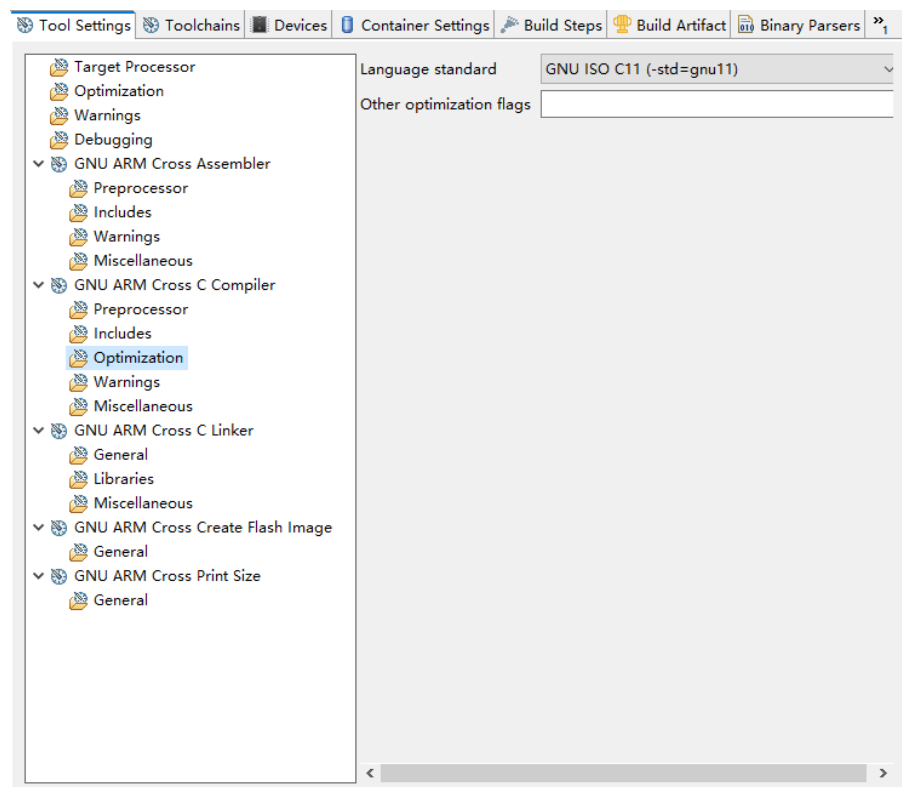
オプション	設定
Include paths (-I)	“\${workspace_loc}/\${ProjName}/src/library/libraries/cmsi s/cm3/core_support/gmd}” “\${workspace_loc}/\${ProjName}/src/library/libraries/cmsi s/cm3/device_support}” “\${workspace_loc}/\${ProjName}/src/library/libraries/drive r/inc}” “\${workspace_loc}/\${ProjName}/src/library/middlewares/ delay}” “\${workspace_loc}/\${ProjName}/src/library/middlewares/ dmm}” “\${workspace_loc}/\${ProjName}/src/library/middlewares/ gpio}” “\${workspace_loc}/\${ProjName}/src/library/middlewares/ hyper_ram}” “\${workspace_loc}/\${ProjName}/src/library/middlewares/ psram}”

オプション	設定
	“\${workspace_loc:/\${ProjName}}/src/library/middlewares/ spi_flash” “\${workspace_loc:/\${ProjName}}/src/library/middlewares/ uart” “\${workspace_loc:/\${ProjName}}/src/project”
Include system paths (-isystem)	-
Include files (-include)	-

3. Optimization

C 言語の標準と拡張機能を設定します。対応する言語機能と追加の最適化オプションを必要に応じて有効にします(図 8-15)。

図 8-15 GNU ARM Cross C Compiler > Optimization



オプションの設定を表 8-6 に示します。

表 8-6 GNU ARM Cross C Compiler > Optimization

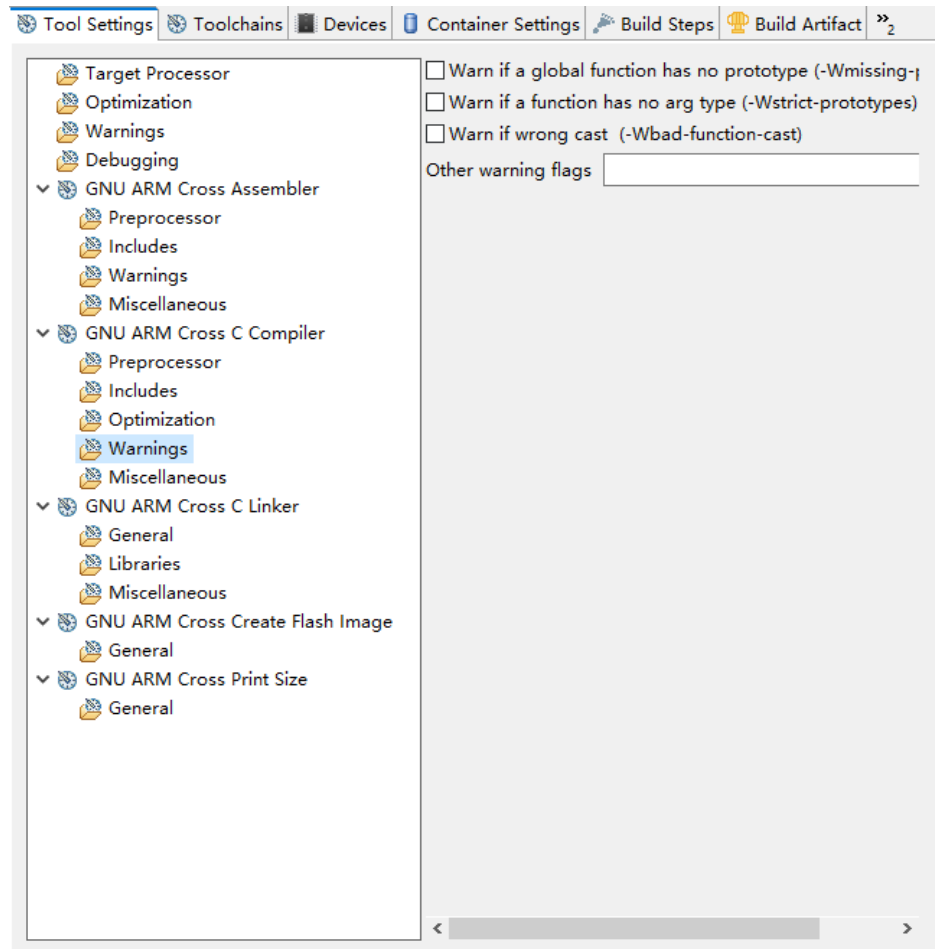
オプション	設定
Language standard	GNU ISO C11 (-std=gnu11)

オプション	設定
Other optimization flags	-

4. Warnings

必要に応じて追加の警告情報を入力します(図 8-16)。

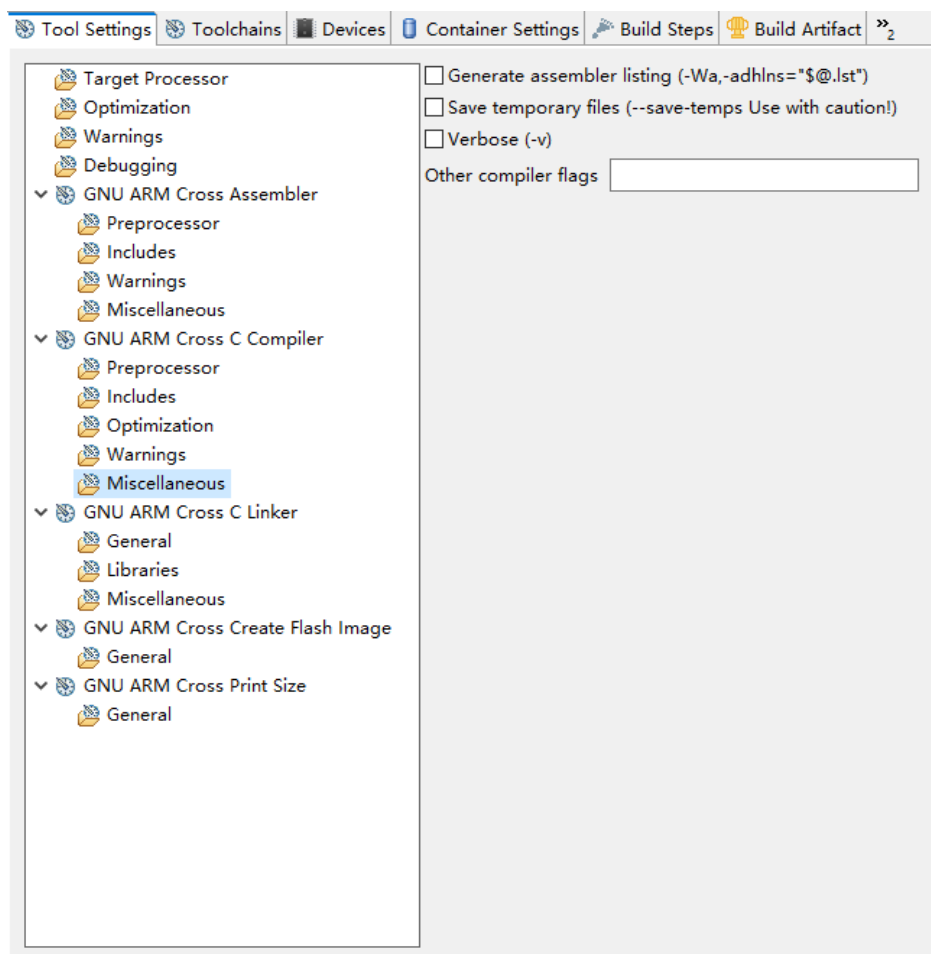
図 8-16 GNU ARM Cross C Compiler > Warnings



5. Miscellaneous

必要に応じて追加のコンパイラ情報を入力します(図 8-17)。

図 8-17 GNU ARM Cross C Compiler > Miscellaneous



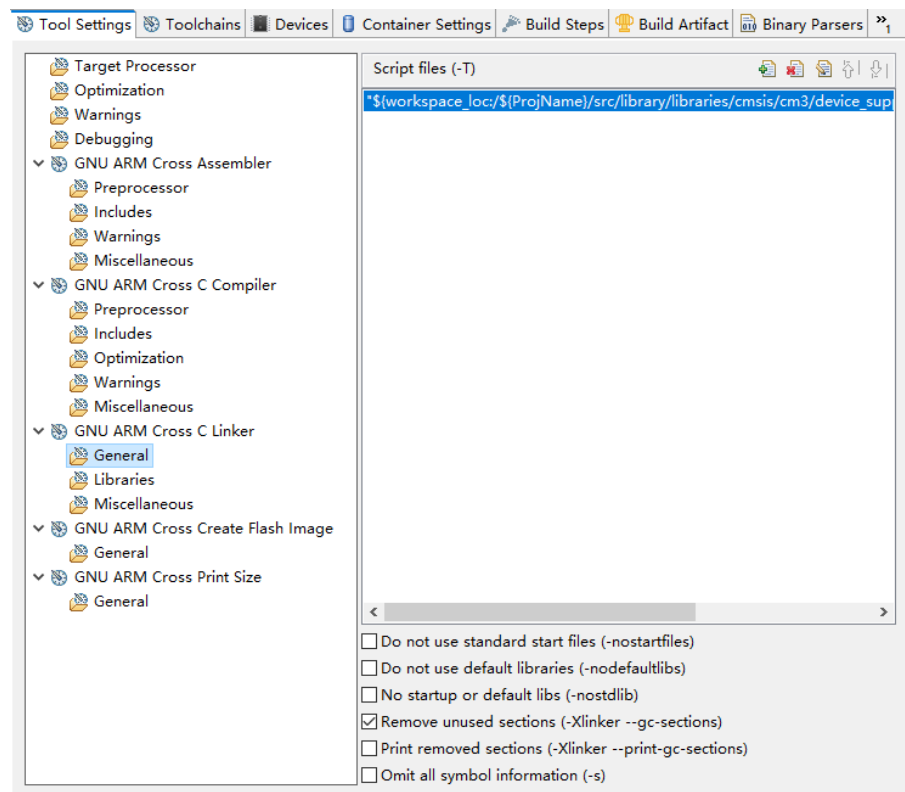
GNU ARM Cross C Linker

General、Libraries、Miscellaneous を含む Arm MCU C リンク・オプションを設定します。

1. General

Script files (-T)は、リンカー・スクリプト・ファイルを指定するために使用されます(図 8-18)。

図 8-18 GNU ARM Cross C Linker > General



オプションの設定を表 8-7 に示します。

表 8-7 GNU ARM Cross C Linker > General

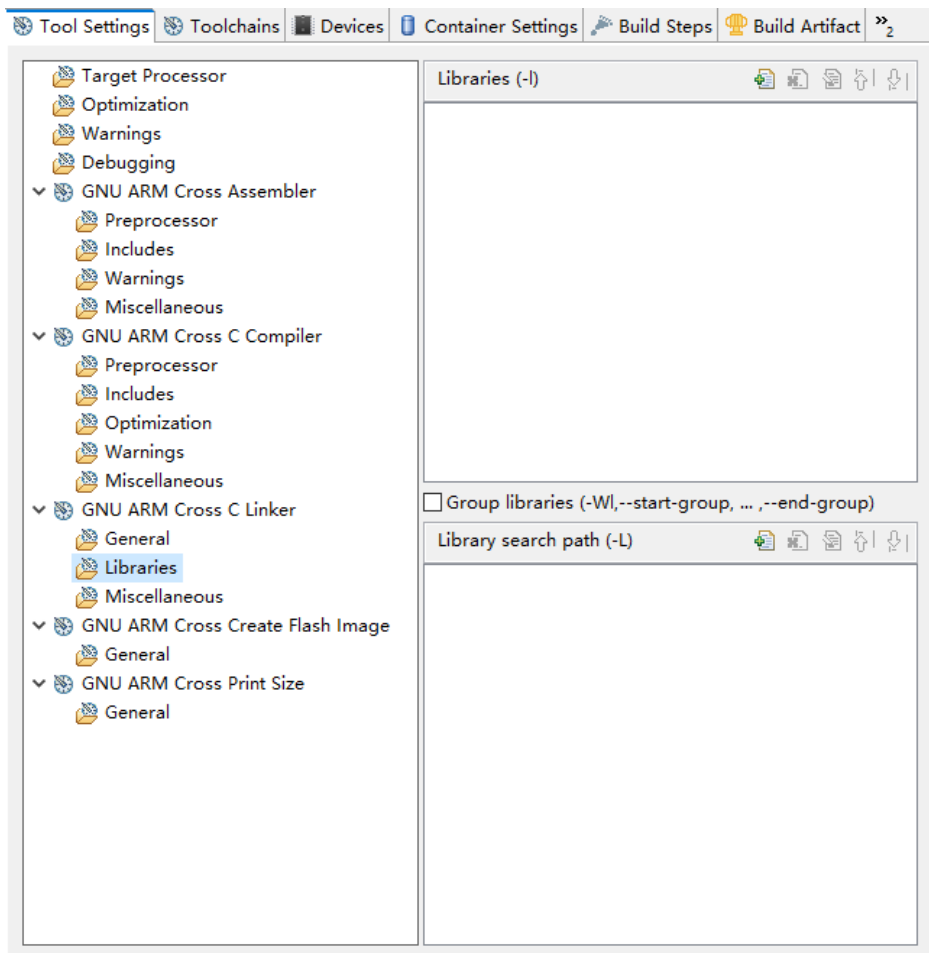
オプション	設定
Script files (-T)	“\${workspace_loc}/\${ProjName}/src/library/libraries/cmsis/cm3/device_support/startup/gmd/linker/gw1ns4c_flash.ld}”
Do not use standard start files (-nostartfiles)	-
Do not use default libraries (-nodefaultlibs)	-
No startup or default libs (-nostdlib)	-
Remove unused sections (-Xlinker --gc-sections)	√
Print removed sections (-Xlinker --print-gc-sections)	-
Omit all symbol information (-s)	-

2. Libraries

ライブラリのコンパイル・オプションを設定します(図 8-19)。

Libraries (-l)テキストボックスに必要な応じて、リンクするライブラリを入力します。**Library search path (-L)**は、静的ライブラリ(.a)と共有ライブラリ(.so)の検索パスを設定するために使用されます。

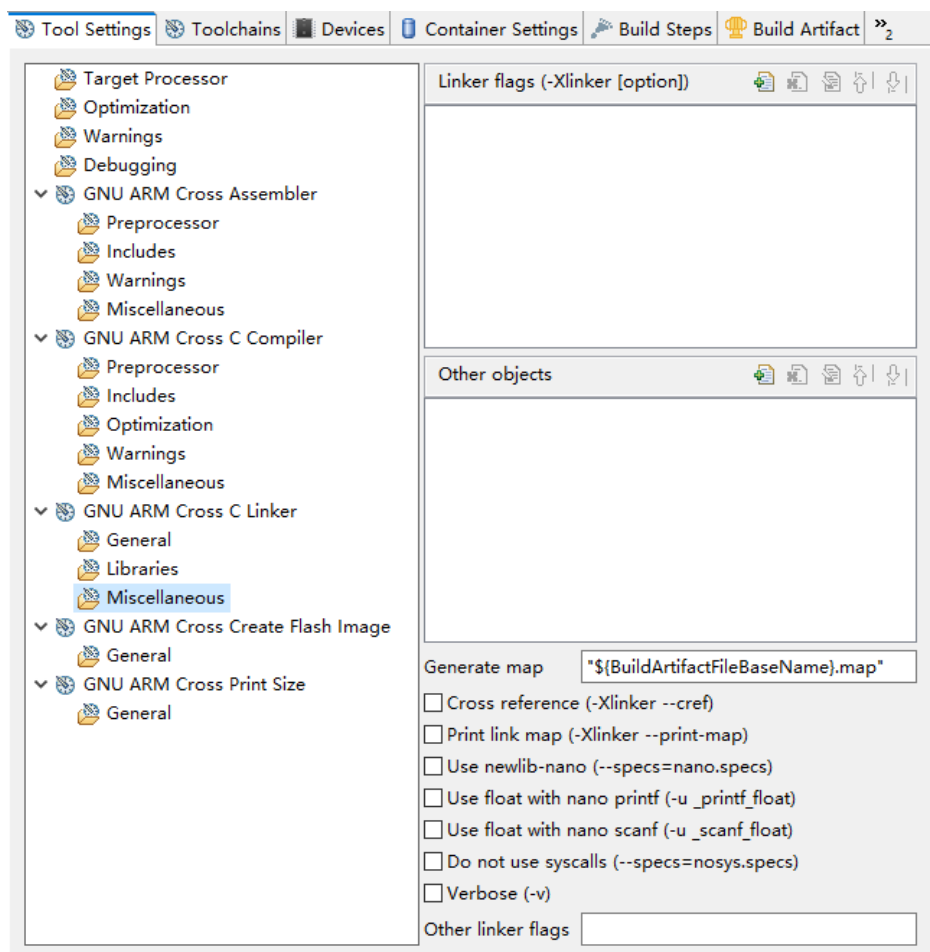
図 8-19 GNU ARM Cross C Linker > Libraries



3. Miscellaneous

必要な応じて追加のリンク・オプションを設定・入力します(図 8-20)。

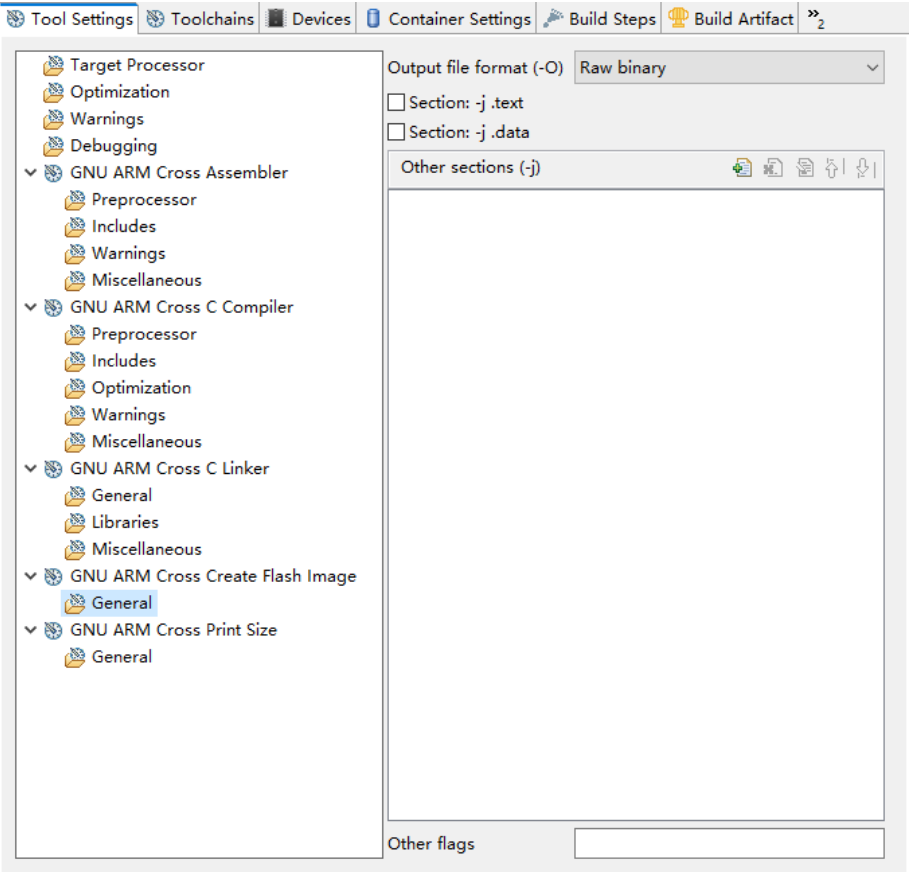
図 8-20 GNU ARM Cross C Linker > Miscellaneous



GNU ARM Cross Create Flash Image

Raw binary、Intel HEX など、Arm MCU 実行ファイルの出力形式を設定します(図 8-21)。

図 8-21 GNU ARM Cross Create Flash Image



オプションの設定を表 8-8 に示します。

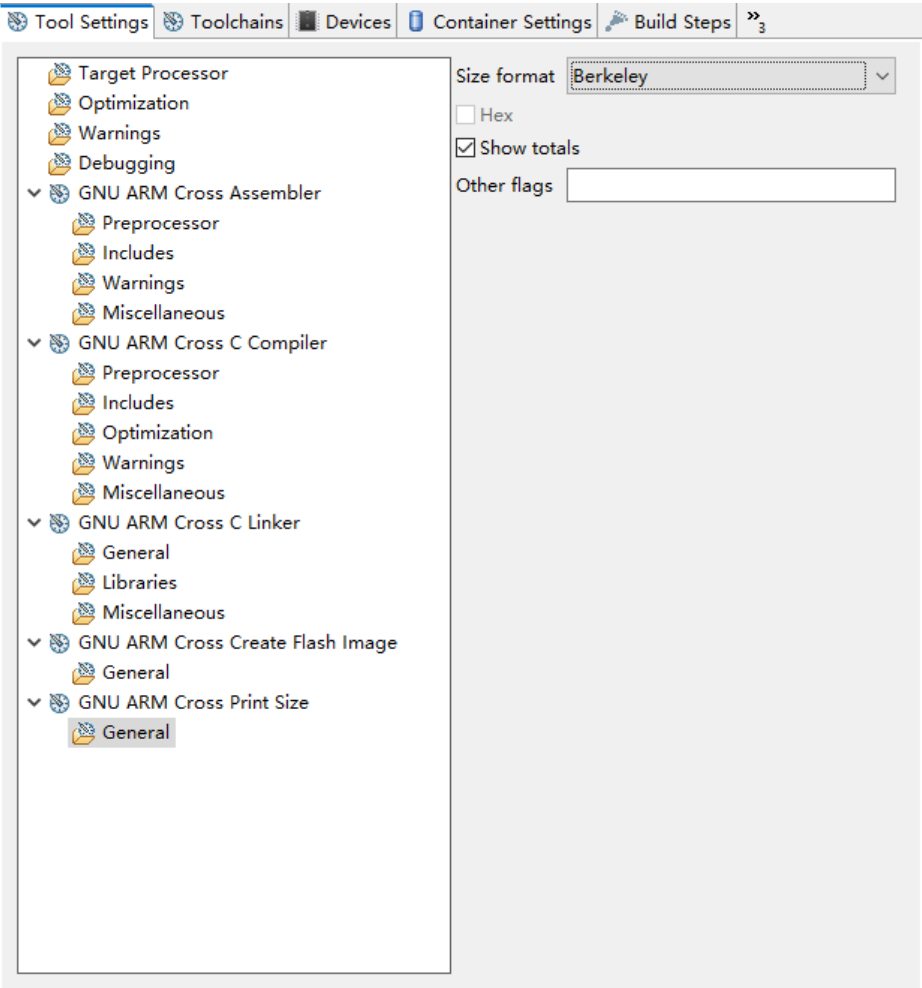
表 8-8 GNU ARM Cross Create Flash Image

オプション	設定
Output file format (-O)	Raw binary
Section: -j .text	-
Section: -j .data	-
Other sections (-j)	-
Other flags	-

GNU ARM Cross Print Size

Arm MCU 実行ファイル（Berkeley 形式および SysV 形式を含む）内の各セグメント（text、data、bss）と合計サイズの統計情報を出力するために使用されます。形式の違いは、実際のセグメントサイズの計算には影響しません（図 8-22）。

図 8-22 GNU ARM Cross Print Size



オプションの設定を表 8-9 に示します。

表 8-9 GNU ARM Cross Print Size

オプション	設定
Size format	Berkeley
Show totals	√
Other flags	-

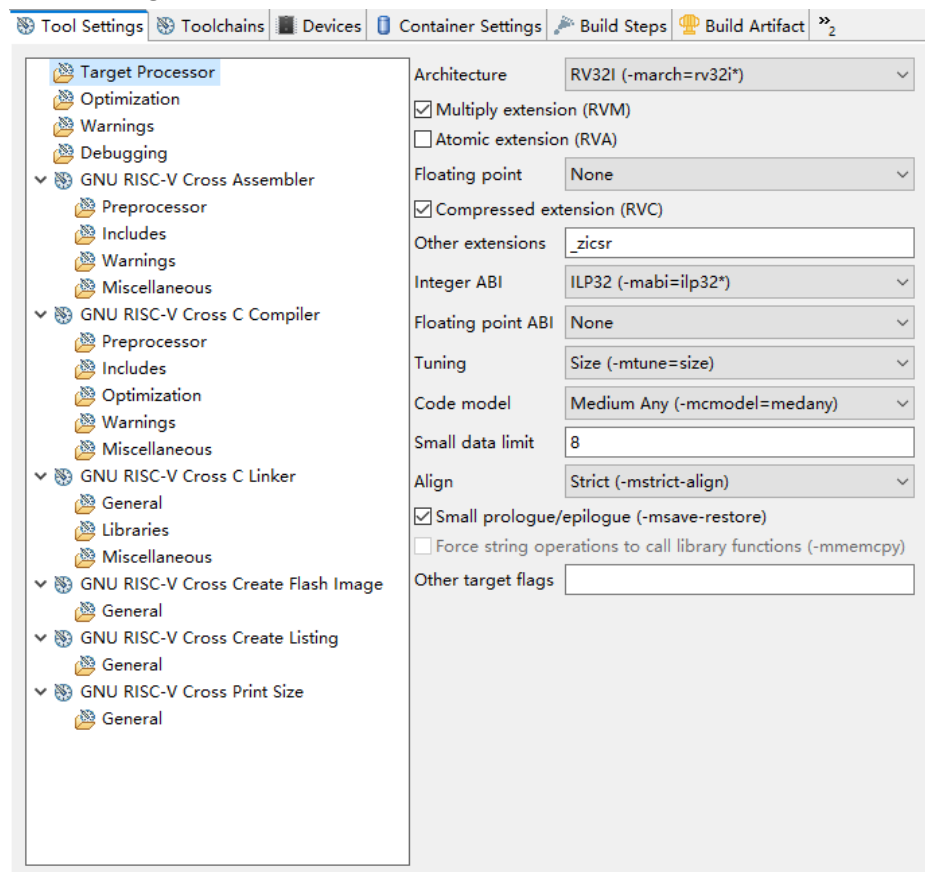
8.2.2 RISC-V MCU プロジェクトの設定

Gowin_PicoRV32 を例に説明します。

Target Processor

RISC-V ターゲットプロセッサを設定します(図 8-23)。

図 8-23 Target Processor



オプションの設定を表 8-10 に示します。

表 8-10 Target Processor

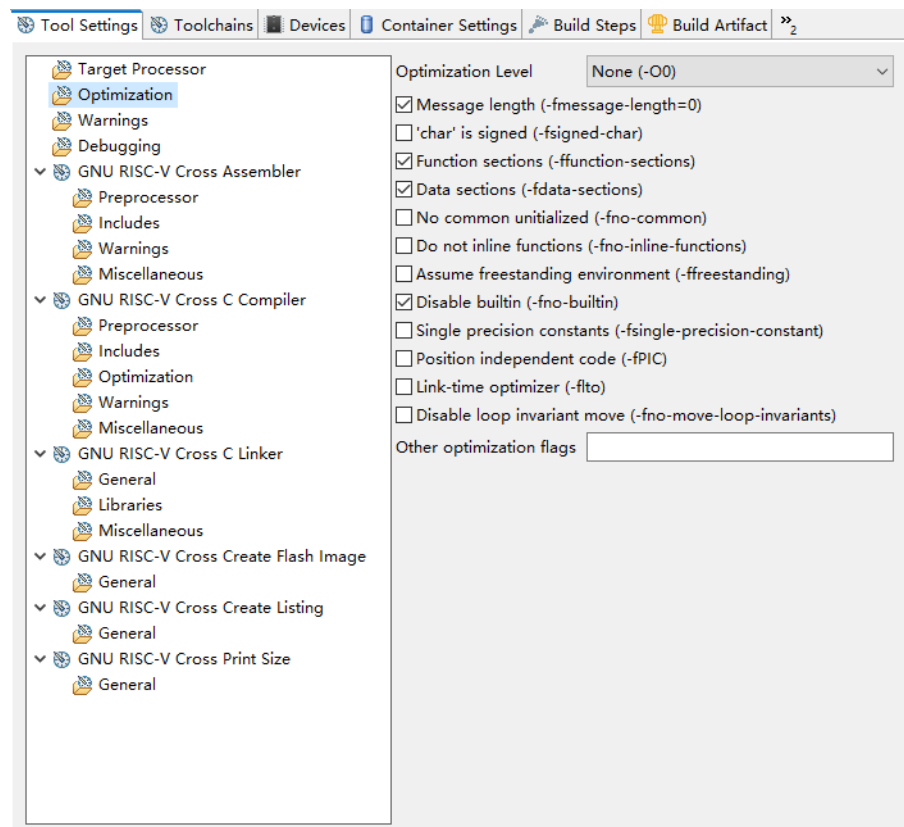
オプション	設定	説明
Architecture	RV32I (-march = rv32i)	RISC-V アーキテクチャ
Multiply extension (RVM)	√	乗算および除算命令
Atomic extension (RVA)	-	アトミック命令
Floating point	None	浮動小数点命令
Compressed extension (RVC)	√	圧縮命令
Other extensions	_zicsr	追加の拡張命令
Integer ABI	ILP32 (-mabi = ilp32*)	整数演算の ABI
Floating point ABI	None	浮動小数点演算の ABI
Tuning	Size (-mtune = size)	特定のプロセッサ・マイクロ・アーキテクチャ向けにコード生

オプション	設定	説明
		成を最適化
Code model	Medium Any (-mcmmodel = medany)	コード・モデル
Small data limit	8	小データ領域のサイズ制限
Align	Strict (-mstrict-align)	アライメント方式
Small prologue/epilogue (-msave-restore)	√	コンパイラはレジスタの保存と復元に外部関数を使用することを選択可能
Other target flags	-	ターゲットプロセッサの追加コンパイル・オプション

Optimization

コンパイル速度、コードのパフォーマンス、およびコードのサイズのバランスをとるために、-O0、-O1、-O2、-O3、-Os、-Ofast、-Og、-Ozなどの最適化レベルを設定します(図 8-24)。

図 8-24 Optimization



オプションの設定を表 8-11 に示します。

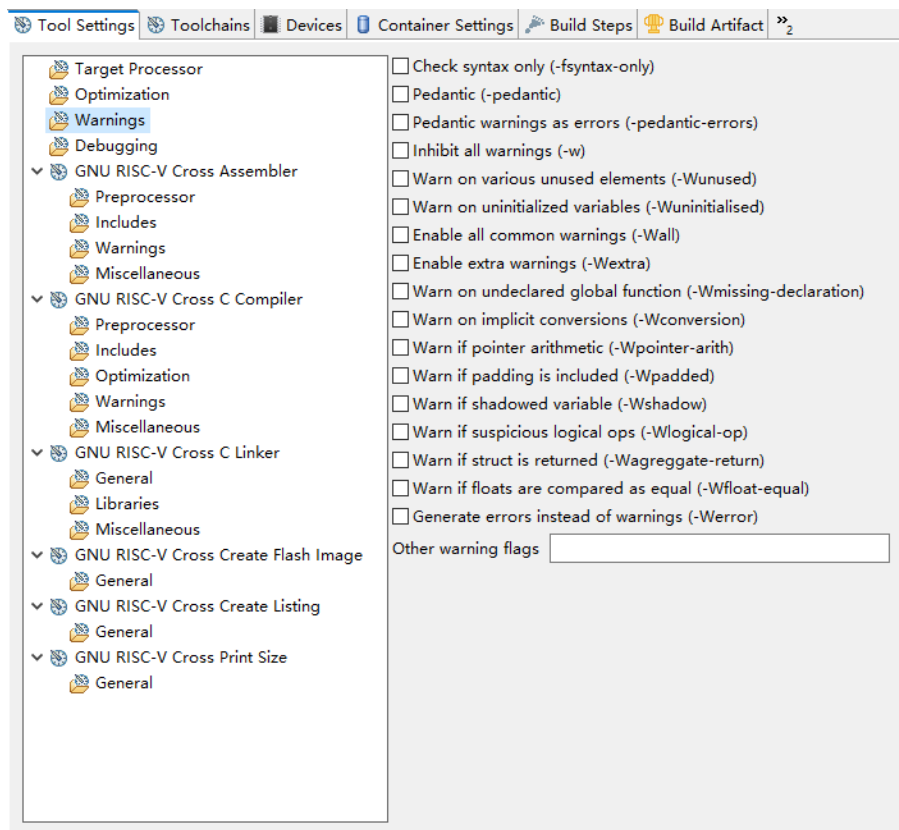
表 8-11 Optimization

オプション	設定
Optimization Level	None (-O0)
Message length (-fmessage-length = 0)	√
‘char’ is signed (-fsigned-char)	-
Function sections (-ffunction-sections)	√
Data sections (-fdata-sections)	√
No common uninitialized (-fno-common)	-
Do not inline functions (-fno-inline-functions)	-
Assume freestanding environment (-ffreestanding)	-
Disable builtin (-fno-builtin)	√
Single precision constants (-fsingle-precision-constant)	-
Position independent code (-fPIC)	-
Link-time optimizer (-flto)	-
Disable loop invariant move (-fno-move-loop-invariants)	-
Other optimization flags	-

Warnings

警告情報スイッチを設定します。これによりユーザーは潜在的な問題を早期に検出し、必要に応じて警告をエラーとして扱うことができます (図 8-25)。

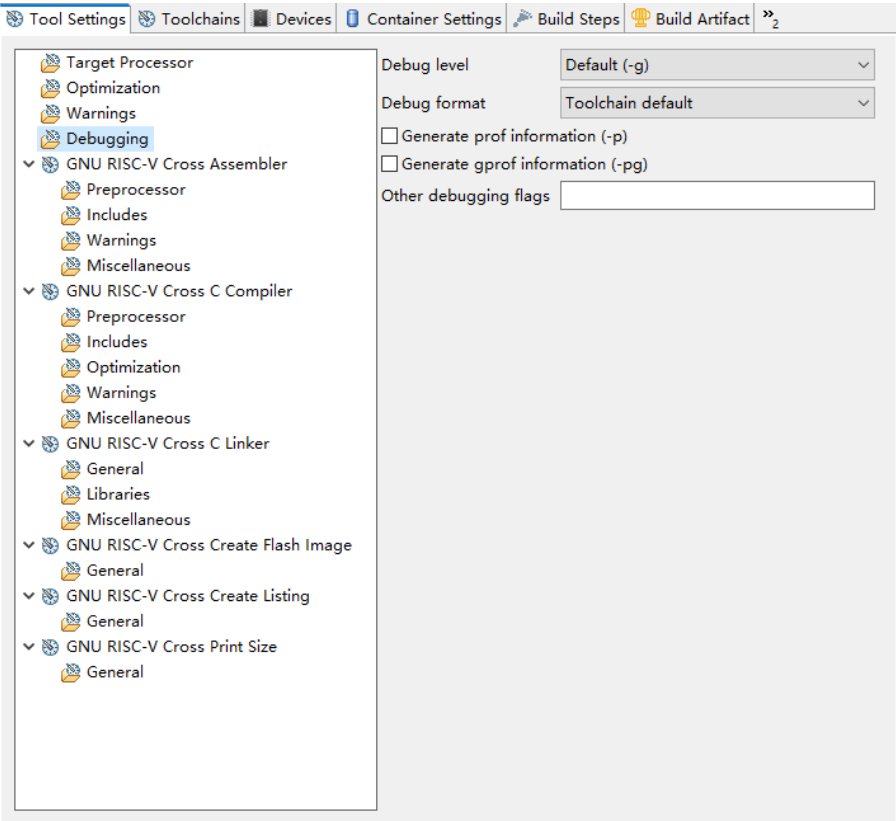
図 8-25 Warnings



Debugging

-g、-g1、-g3 などのデバッグ・レベルを設定します(図 8-26)。

図 8-26 Debugging



オプションの設定を表 8-12 に示します。

表 8-12 Debugging

オプション	設定
Debug level	Default (-g)
Debug format	Toolchain default
Generate prof information (-p)	-
Generate gprof information (-pg)	-
Other debugging flags	-

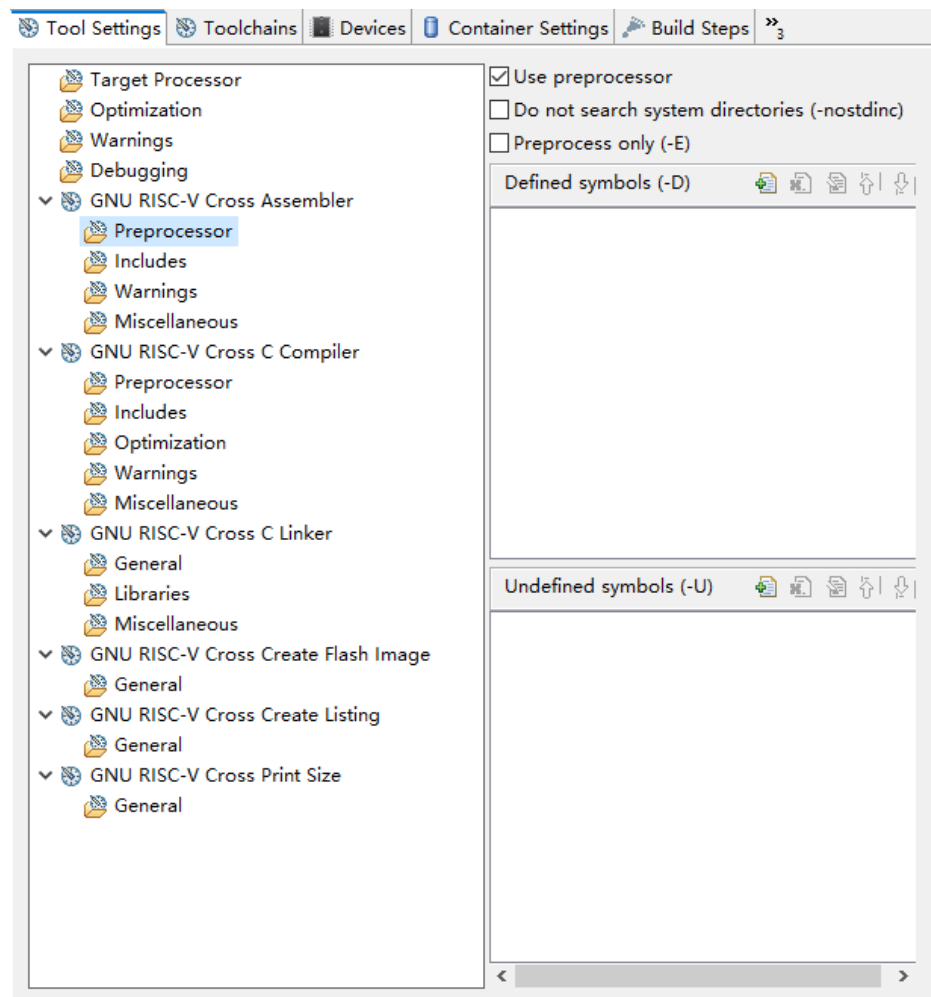
GNU RISC-V Cross Assembler

Preprocessor、Includes、Warnings、Miscellaneous を含む RISC-V MCU アセンブリのオプションを設定します。

1. Preprocessor

マクロ定義を設定します。プロジェクト関連のプリコンパイル済みマクロを追加するために使用されます(図 8-27)。

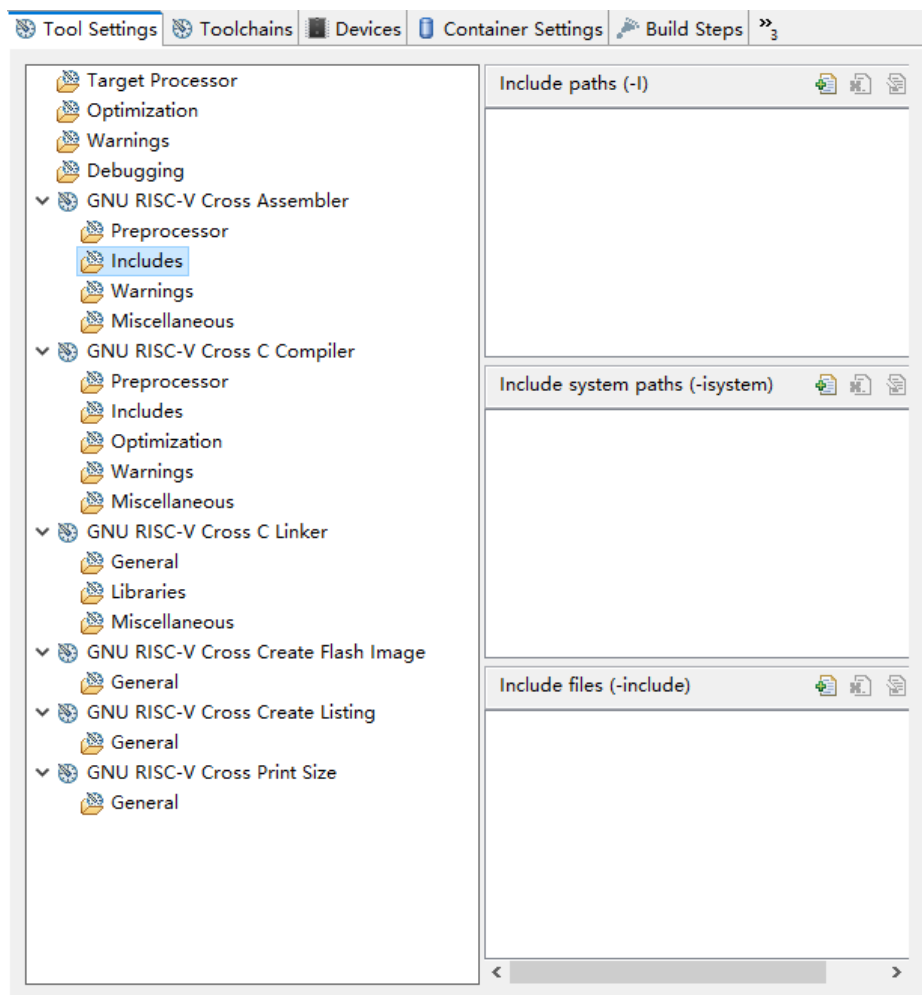
図 8-27 GNU RISC-V Cross Assembler > Preprocessor



2. Includes

ヘッダーファイルの検索パスを指定します。絶対パス、相対パス、または Eclipse 変数（`${workspace_loc}`など）を入力できます(図 8-28)。

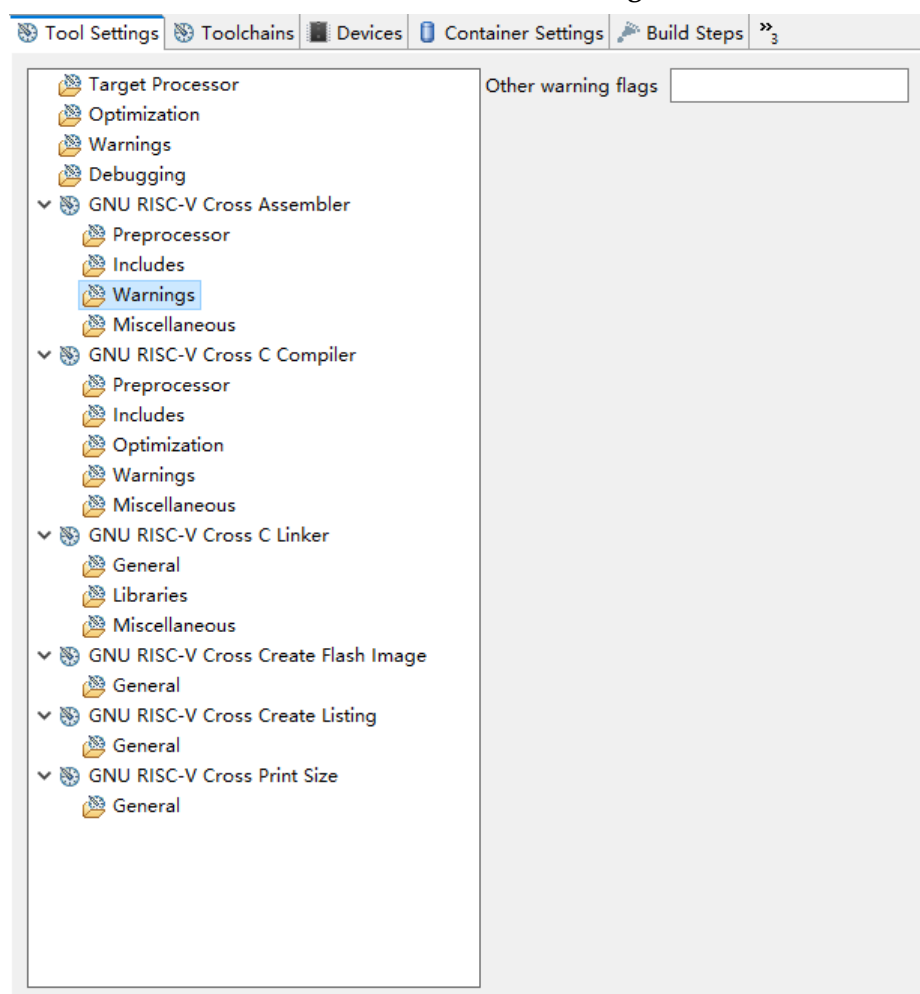
図 8-28 GNU RISC-V Cross Assembler > Includes



3. Warnings

必要に応じて追加の警告情報を入力します(図 8-29)。

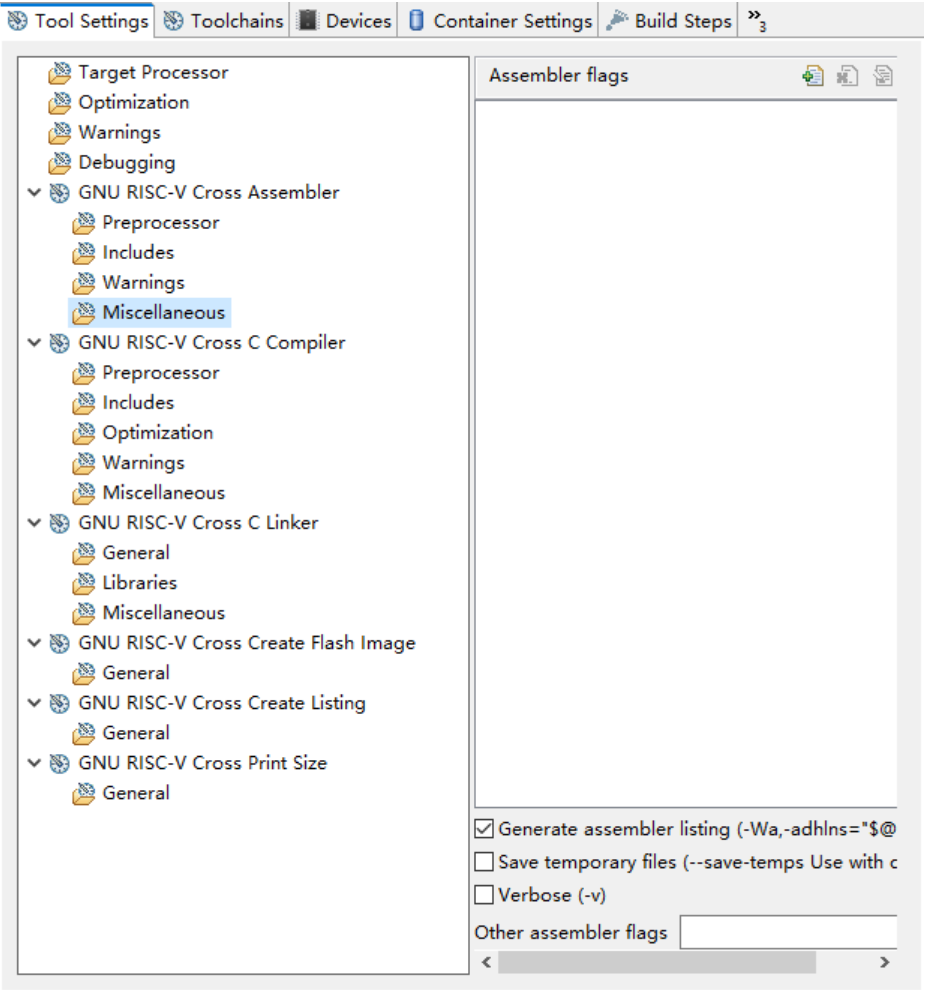
図 8-29 GNU RISC-V Cross Assembler > Warnings



4. Miscellaneous

必要に応じて追加のコンパイラ情報を入力します(図 8-30)。

図 8-30 GNU RISC-V Cross Assembler > Miscellaneous



オプションの設定を表 8-13 に示します。

表 8-13 GNU RISC-V Cross Assembler > Miscellaneous

オプション	設定
Assembler flags	-
Generate assembler listing (-Wa,-adhlns=" \$@.lst")	√
Save temporary files (--save-temps Use with caution!)	-
Verbose (-v)	-
Other assembler flags	-

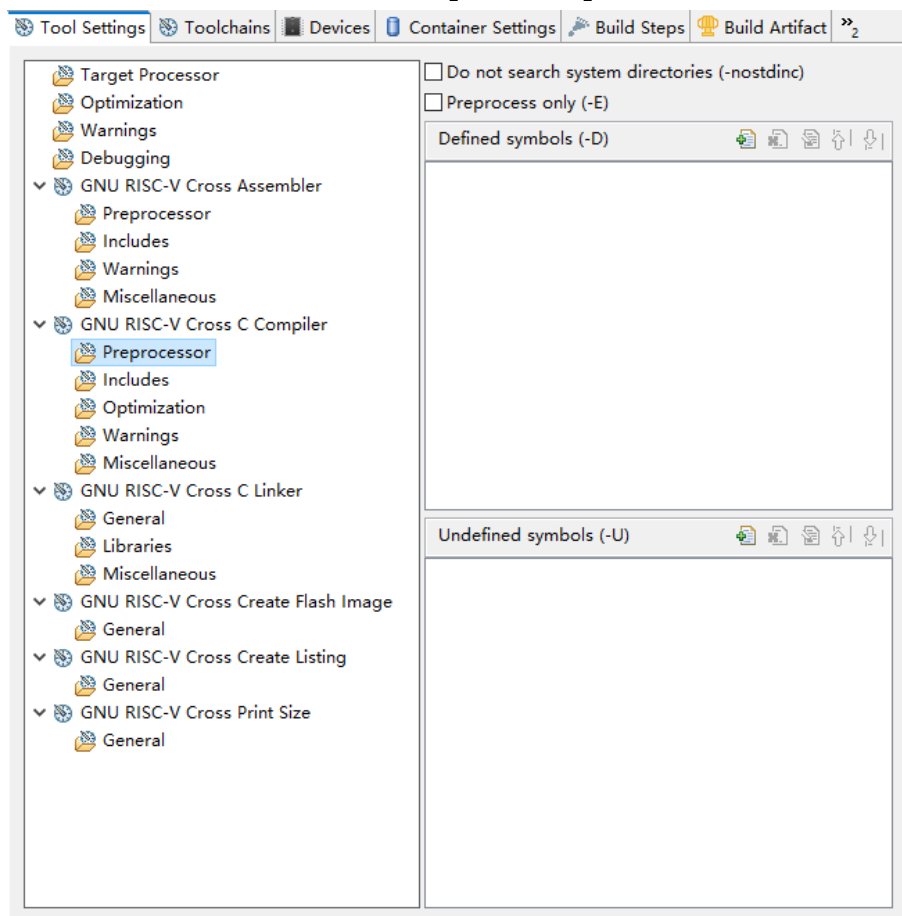
GNU RISC-V Cross C Compiler

Preprocessor、Includes、Optimization、Warnings、Miscellaneous を含む RISC-V MCU C コンパイル・オプションを設定します。

1. Preprocessor

マクロ定義を設定します。プロジェクト関連のプリコンパイル済みマクロを追加するために使用されます(図 8-31)。

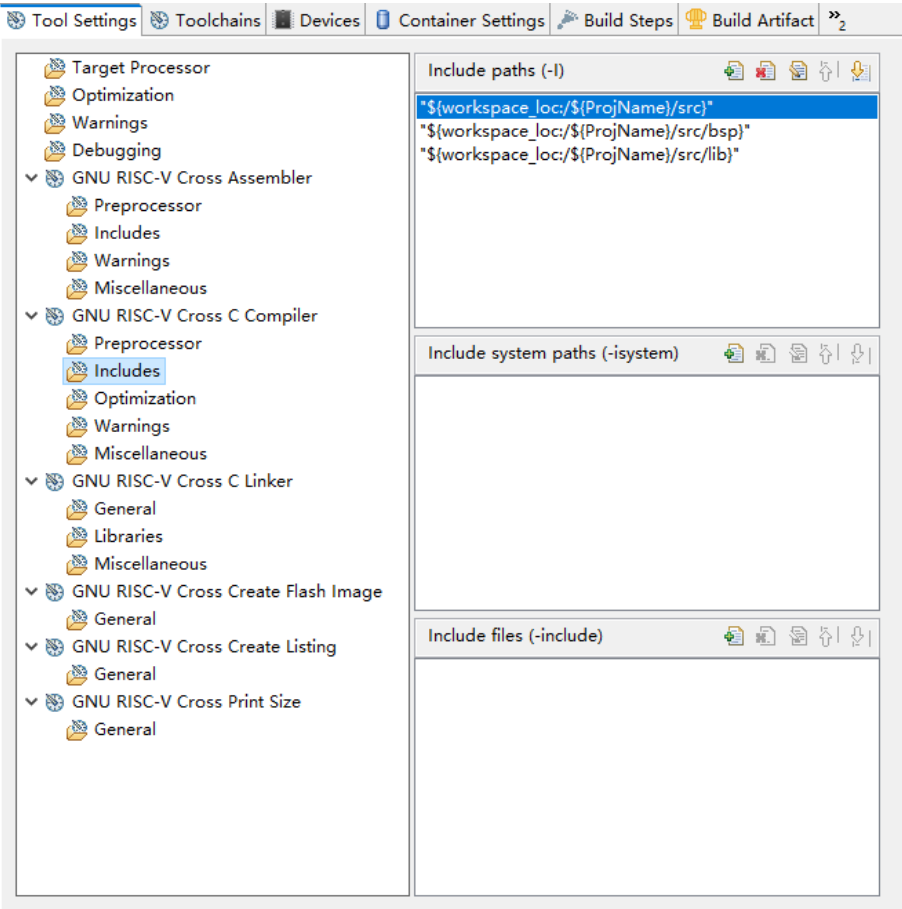
図 8-31 GNU RISC-V Cross C Compiler > Preprocessor



2. Includes

ヘッダーファイルの検索パスを指定します。絶対パス、相対パス、または Eclipse 変数（`${workspace_loc}`など）を入力できます(図 8-32)。

図 8-32 GNU RISC-V Cross C Compiler > Includes



オプションの設定を表 8-14 に示します。

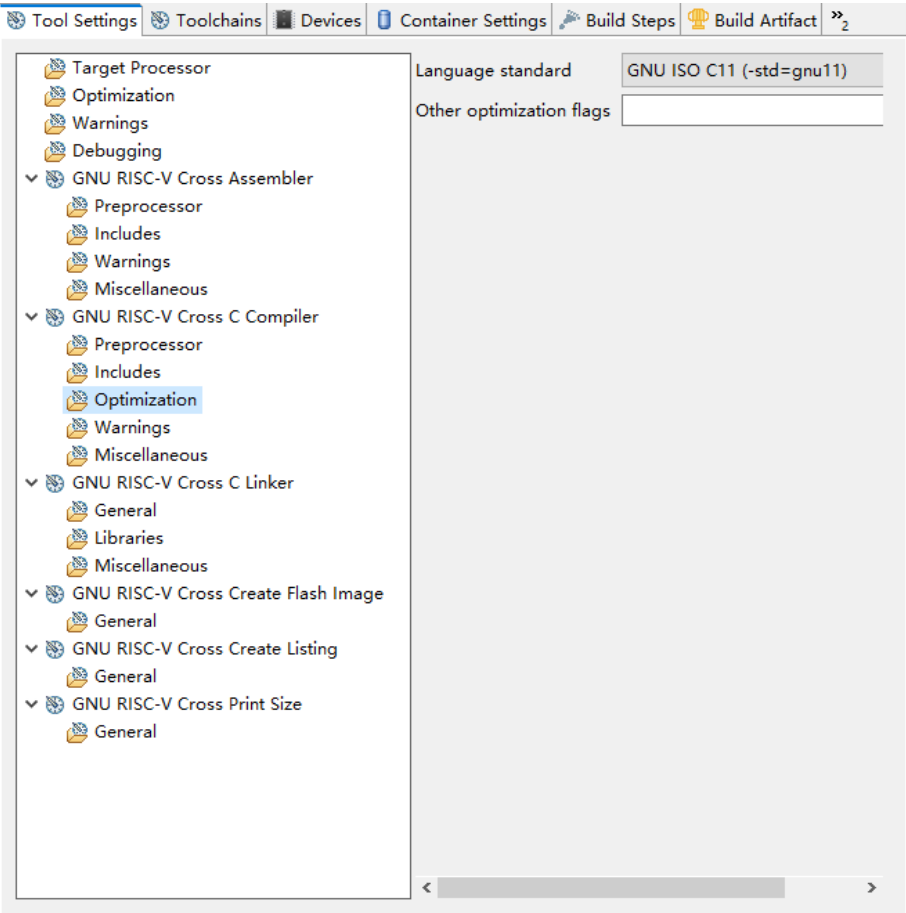
表 8-14 GNU RISC-V Cross C Compiler > Includes

オプション	設定
Include paths (-I)	“\$(workspace_loc:\${ProjName}/src)” “\$(workspace_loc:\${ProjName}/src/bsp)” “\$(workspace_loc:\${ProjName}/src/lib)”
Include system paths (-isystem)	-
Include files (-include)	-

3. Optimization

C 言語の標準と拡張機能を設定します。対応する言語機能と追加の最適化オプションを必要に応じて有効にします(図 8-33)。

図 8-33 GNU RISC-V Cross C Compiler > Optimization



オプションの設定を表 8-15 に示します。

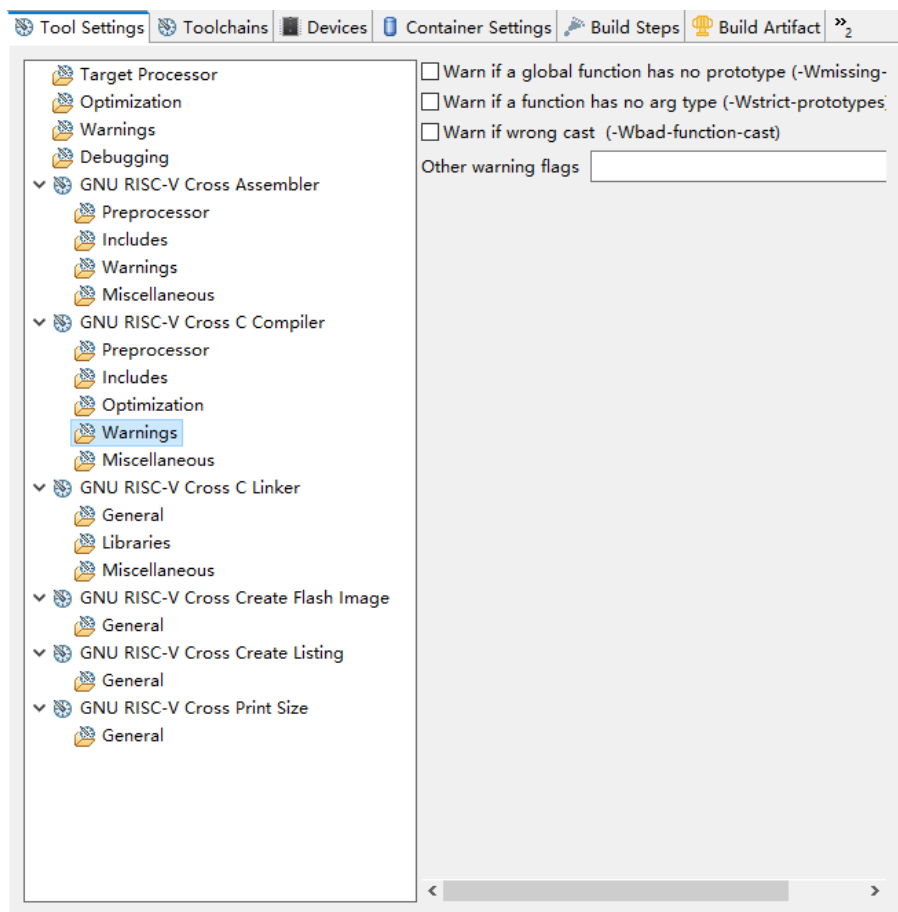
表 8-15 GNU RISC-V Cross C Compiler > Optimization

オプション	設定
Language standard	GNU ISO C11 (-std=gnu11)
Other optimization flags	-

4. Warnings

必要に応じて追加の警告情報を入力します(図 8-34)。

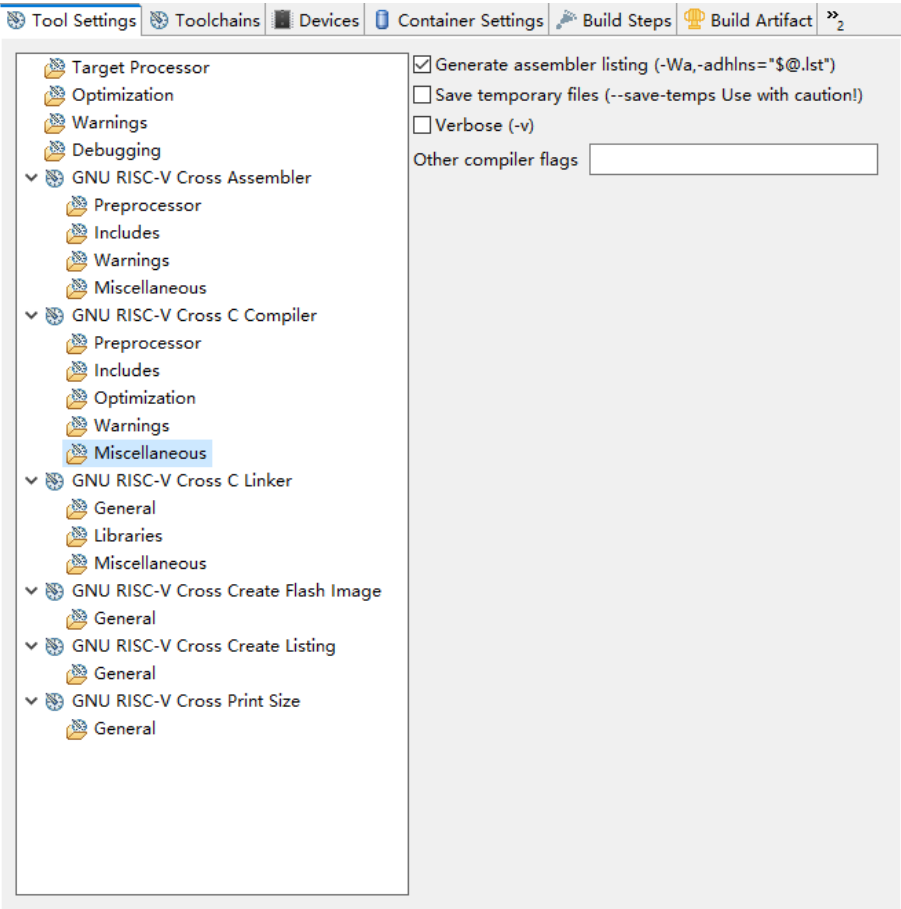
図 8-34 GNU RISC-V Cross C Compiler > Warnings



5. Miscellaneous

必要に応じて追加のコンパイラ情報を入力します(図 8-35)。

図 8-35 GNU RISC-V Cross C Compiler > Miscellaneous



オプションの設定を表 8-16 に示します。

表 8-16 GNU RISC-V Cross C Compiler > Miscellaneous

オプション	設定
Generate assembler listing (-Wa,-adhlns="\$@.lst")	√
Save temporary files (--save-temps Use with caution)	-
Verbose (-v)	-
Other compiler flags	-

GNU RISC-V Cross C Linker

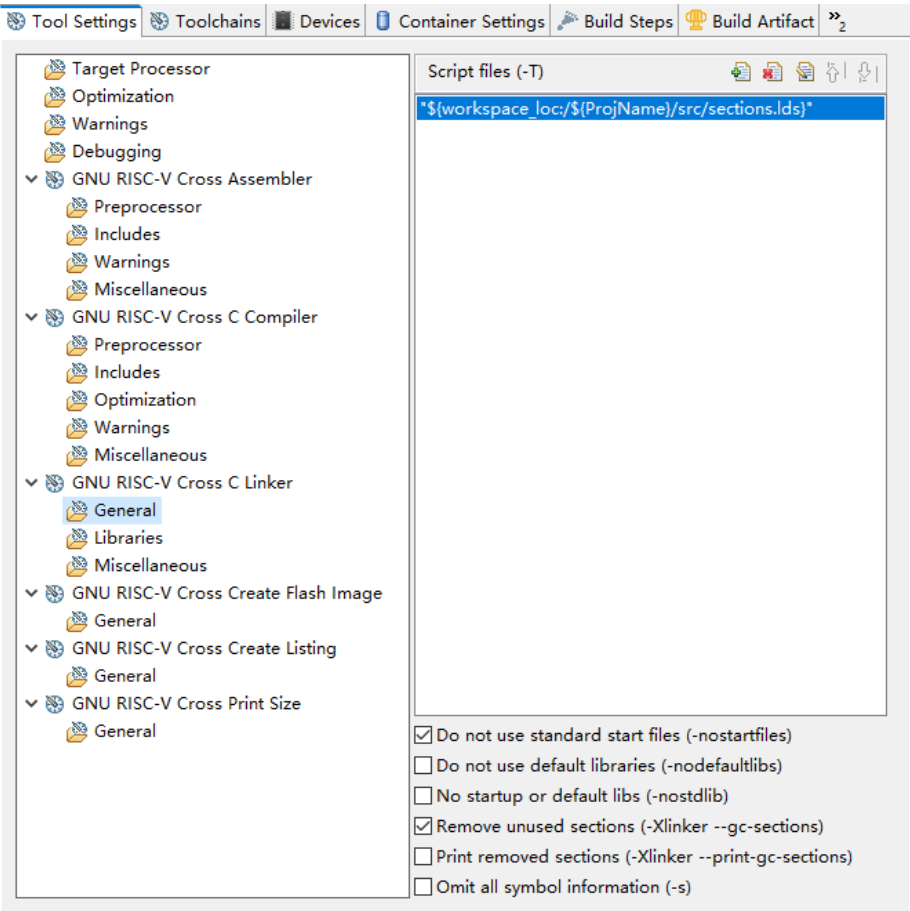
General、Libraries、Miscellaneous を含む RISC-V MCU C リンク・オプションを設定します。

1. General

Script files (-T)は、リンカー・スクリプト・ファイルを指定するため

に使用されます(図 8-36)。

図 8-36 GNU RISC-V Cross C Linker > General



オプションの設定を表 8-17 に示します。

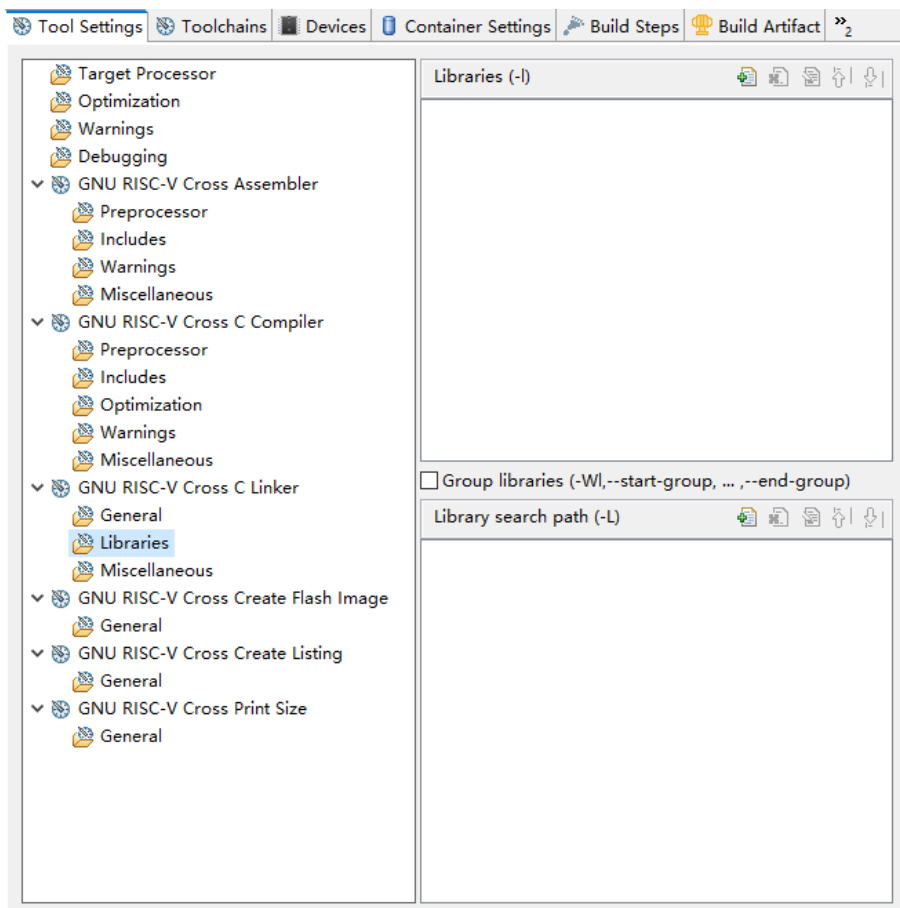
表 8-17 GNU RISC-V Cross C Linker > General

オプション	設定
Script files (-T)	“\${workspace_loc}/\${ProjName}/src/sections.lds”
Do not use standard start files (-nostartfiles)	√
Do not use default libraries (-nodefaultlibs)	-
No startup or default libs (-nostdlib)	-
Remove unused sections (-Xlinker --gc-sections)	√
Print removed sections (-Xlinker --print-gc-sections)	-
Omit all symbol information (-s)	-

2. Libraries

ライブラリのコンパイル・オプションを設定します(図 8-37)。
Libraries (-l)テキストボックスに必要な応じて、リンクするライブラリを入力します。**Library search path (-L)**は、静的ライブラリ(.a)と共有ライブラリ(.so)の検索パスを設定するために使用されます。

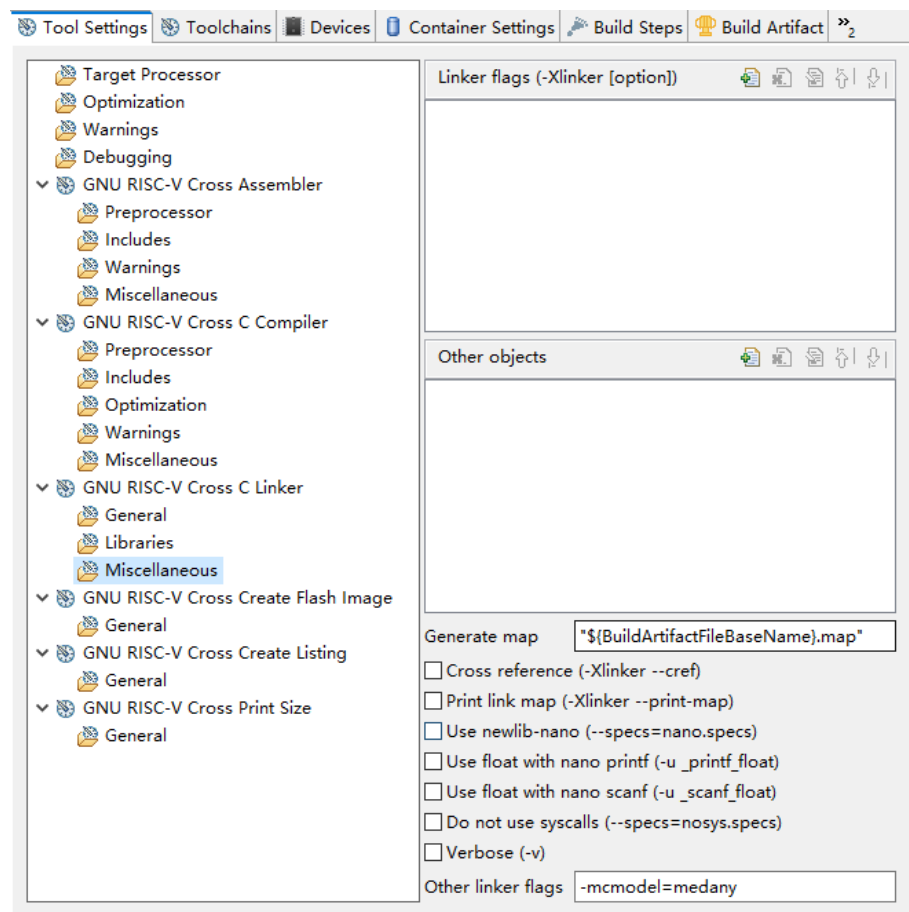
図 8-37 GNU RISC-V Cross C Linker > Libraries



3. Miscellaneous

必要な応じて追加のリンク・オプションを設定・入力します(図 8-38)。

図 8-38 GNU RISC-V Cross C Linker > Miscellaneous



オプションの設定を表 8-18 に示します。

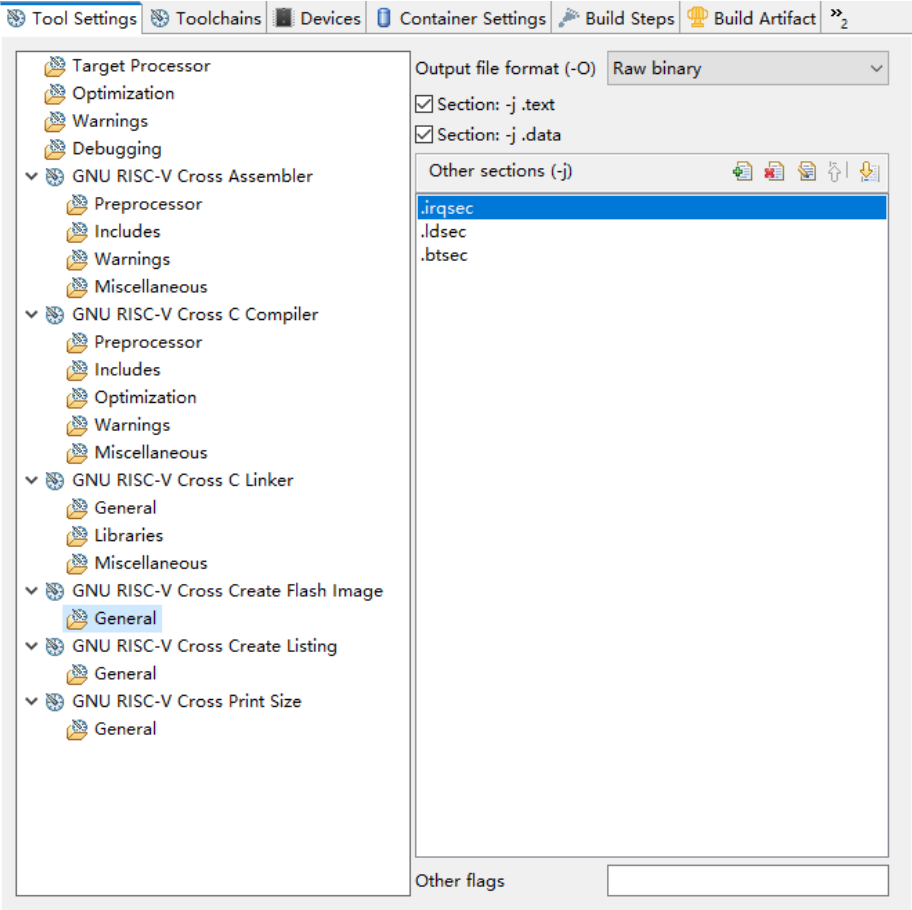
表 8-18 GNU RISC-V Cross C Linker > Miscellaneous

オプション	設定
Linker flags (-Xlinker [option])	-
Other objects	-
Cross reference (-Xlinker --cref)	-
Print link map (-Xlinker --print-map)	-
Use newlib-nano (--specs=nano.specs)	必要に応じて選択
Use float with nano printf (-u _printf_float)	必要に応じて選択
Use float with nano scanf (-u _scanf_float)	必要に応じて選択
Do not use syscalls (--specs = nosys.specs)	-
Verbose (-v)	-
Other linker flags	-mcmodel=medany

GNU RISC-V Cross Create Flash Image

Raw binary、Intel HEX など、RISC-V MCU 実行ファイルの出力形式を設定します(図 8-39)。

図 8-39 GNU RISC-V Cross Create Flash Image



オプションの設定を表 8-19 に示します。

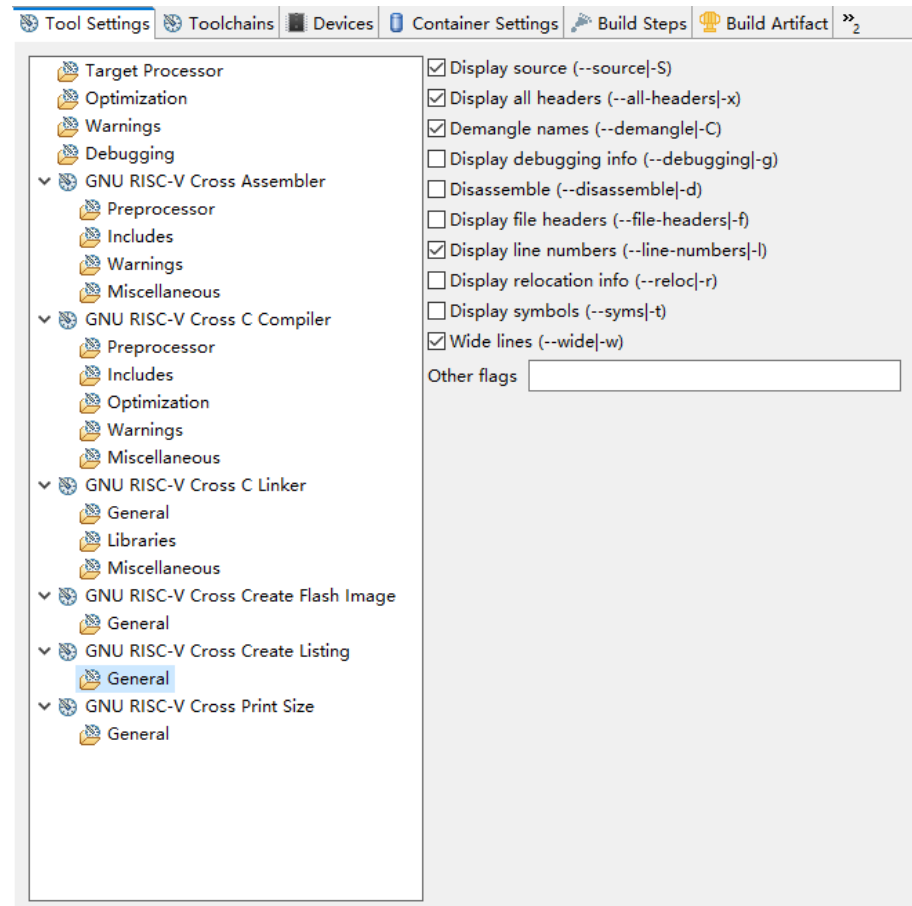
表 8-19 GNU RISC-V Cross Create Flash Image

オプション	設定
Output file format (-O)	Raw binary
Section: -j .text	√
Section: -j .data	√
Other sections (-j)	.irqsec .ldsec .btsec
Other flags	-

GNU RISC-V Cross Create Listing

アセンブリ・コード、シンボル情報、デバッグ情報を含むリスト・ファイル（通常は`.lst` ファイル）を生成するために使用されます。これらのリスト・ファイルは、デバッグ、パフォーマンス分析、コード・レビューに非常に役立ちます(図 8-40)。

図 8-40 GNU RISC-V Cross Create Listing



オプションの設定を表 8-20 に示します。

表 8-20 GNU RISC-V Cross Create Listing

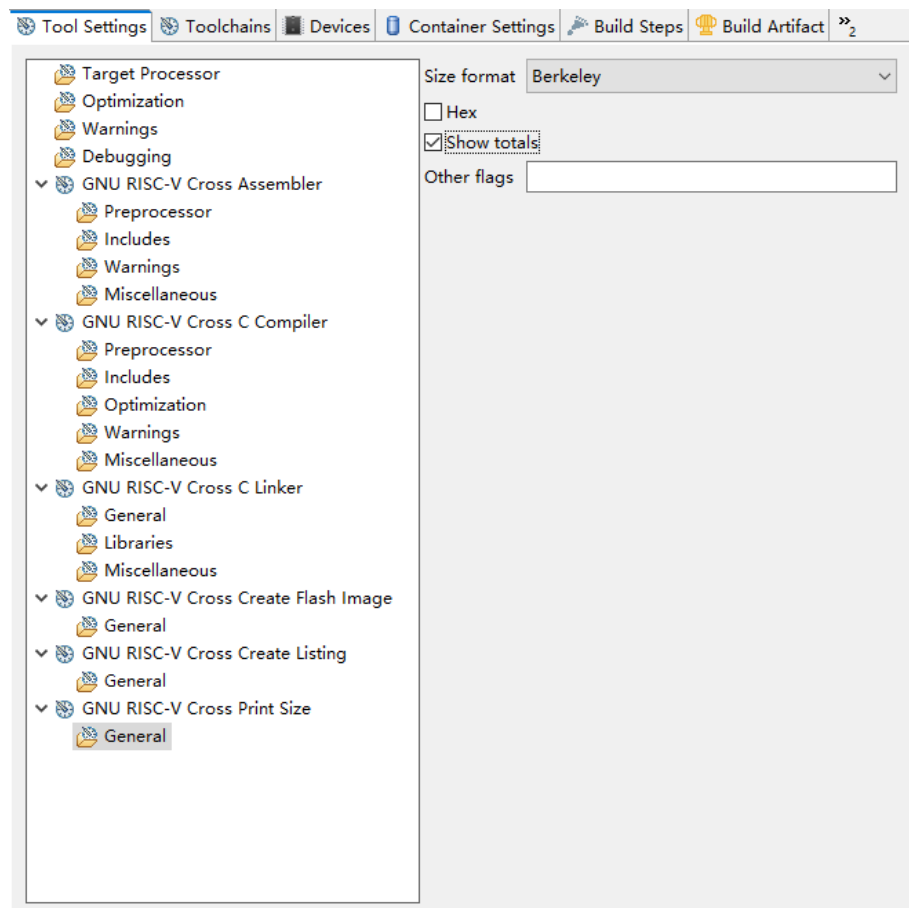
オプション	設定
Display source (--source -S)	√
Display all headers (--all-headers -x)	√
Demangle names (--demangle -C)	√
Display debugging info (--debugging -g)	-
Disassemble (--disassemble -d)	-
Display file headers (--file-headers -f)	-
Display line numbers (--line-numbers -l)	√

オプション	設定
Display relocation info (--reloc -r)	-
Display symbols (--syms -t)	-
Wide lines (--wide -w)	√
Other flags	-

GNU RISC-V Cross Print Size

RISC-V MCU 実行ファイル（Berkeley 形式および SysV 形式を含む）内の各セグメント（text、data、bss）と合計サイズの統計情報を出力するために使用されます。形式の違いは、実際のセグメントサイズの計算には影響しません（図 8-41）。

図 8-41 GNU RISC-V Cross Print Size



オプションの設定を表 8-21 に示します。

表 8-21 GNU RISC-V Cross Print Size

オプション	設定
Size format	Berkeley
Hex	-
Show totals	√
Other flags	-

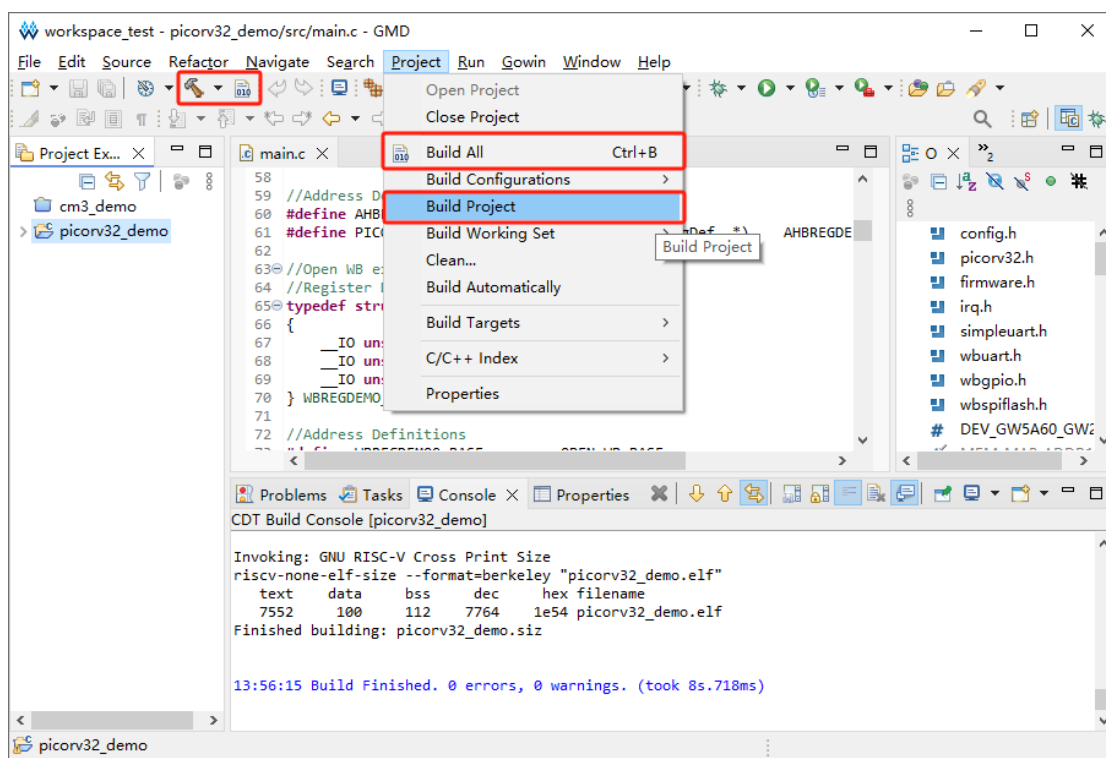
8.3 プロジェクトのコンパイル

8.3.1 コンパイル方法

Arm MCU および RISC-V MCU には、図 8-42 に示すようにいくつかのコンパイル方法があります。

- 現在のプロジェクトを右クリックして「Build Project」を選択します。
- ツールバー上の「Build」()を選択します。
- ツールバー上の「Build All」()を選択します。
- メニューバー > 「Project > Build All」を選択します。
- メニューバー > 「Project > Build Project」を選択します。

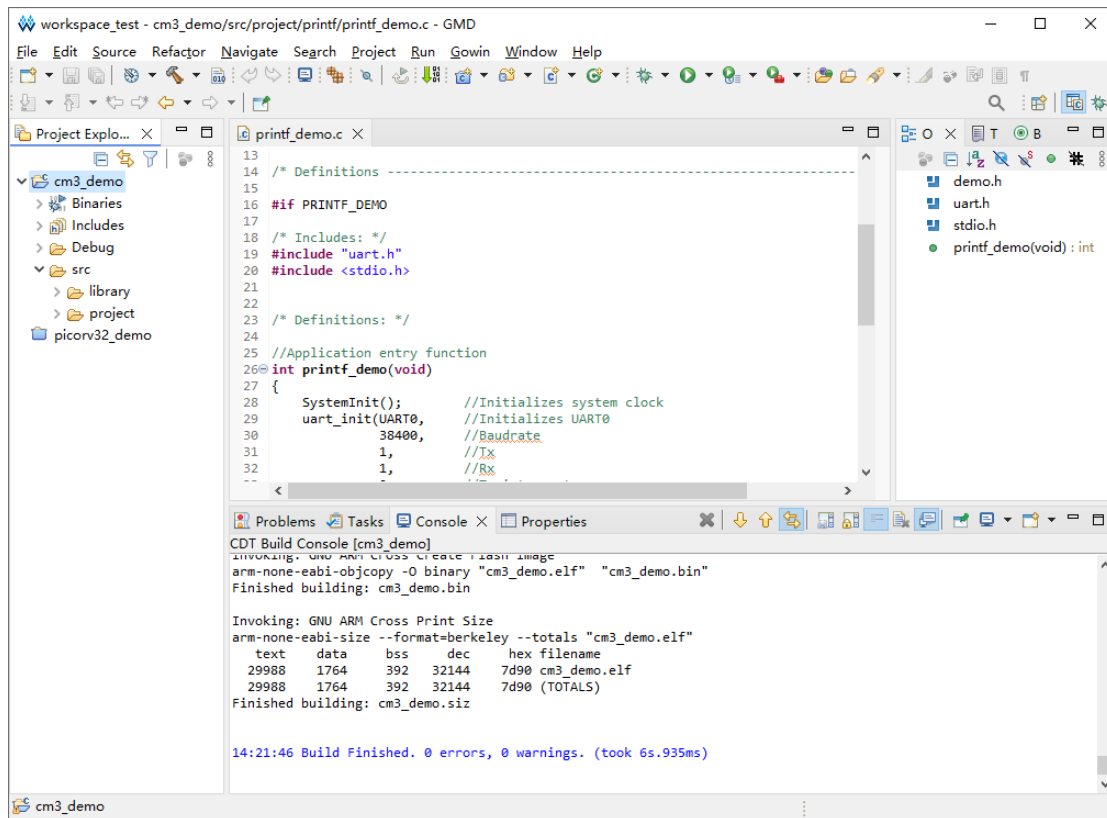
図 8-42 Build



8.3.2 プロジェクトのコンパイル

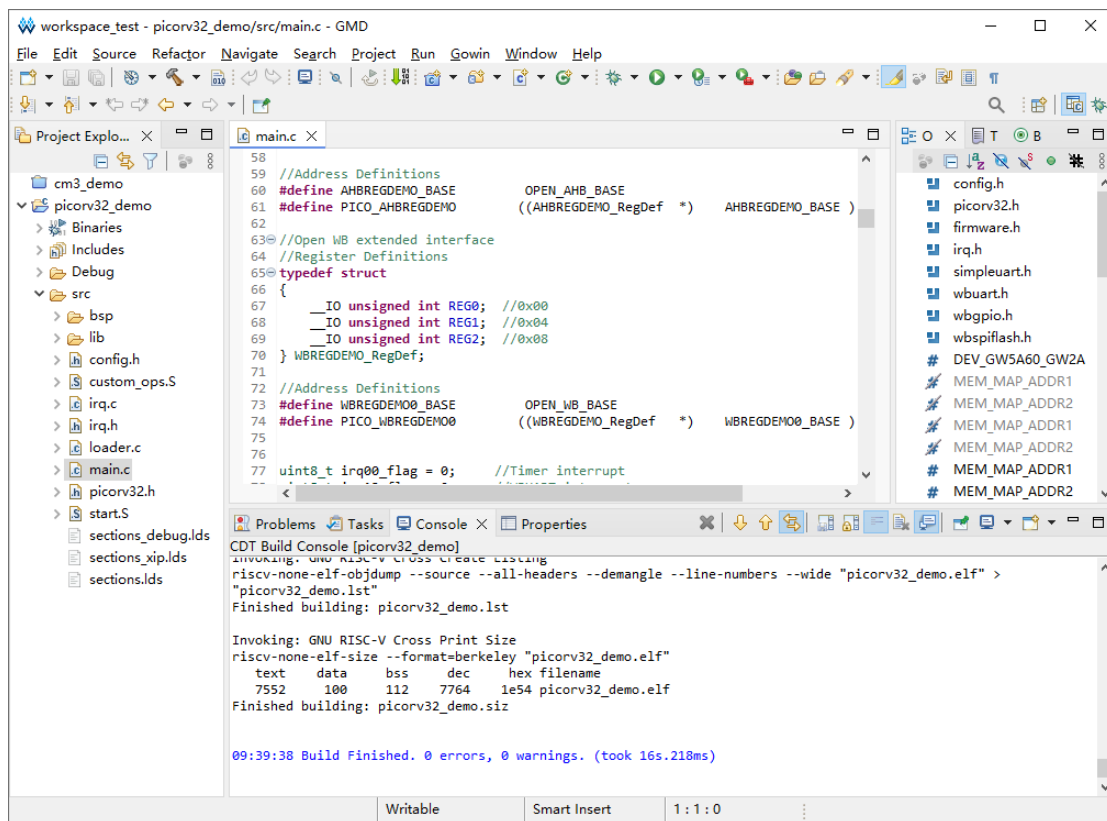
Arm MCU プロジェクトのコンパイル例を図 8-43 に示します。

図 8-43 Arm MCU プロジェクトのコンパイル例



RISC-V MCU プロジェクトのコンパイル例を図 8-44 に示します。

図 8-44 RISC-V MCU プロジェクトのコンパイル例

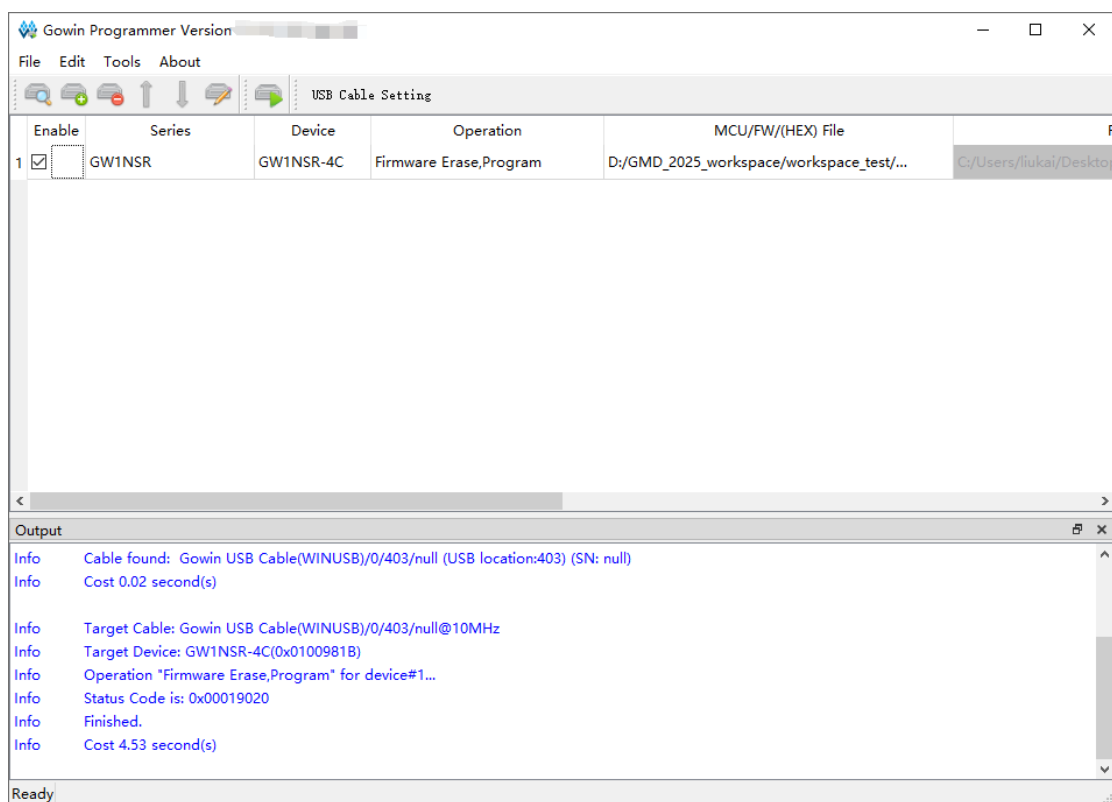


9 プロジェクトのダウンロード

9.1 ダウンロードツール

GMD ソフトウェアには、Arm MCU および RISC-V MCU の実行ファイルをダウンロードするために使用されるダウンロードツール Programmer が統合されています(図 9-1)。

図 9-1 ダウンロードツール

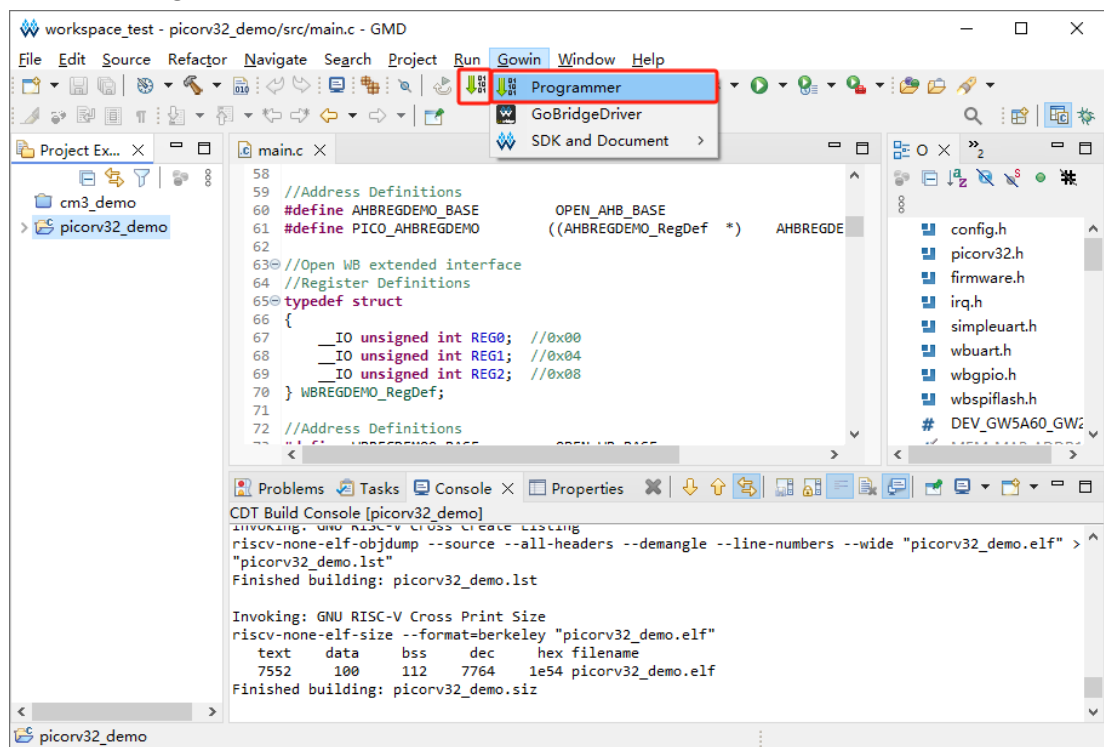


9.2 ダウンロード方法

Arm MCU および RISC-V MCU には、図 9-2 に示すようにいくつかのダウンロード方法があります。

- ツールバー上の「Programmer」(📉) を選択します
- メニューバー > 「Gowin > Programmer」を選択します

図 9-2 Programmer



9.3 プロジェクトのダウンロード

Arm MCU および RISC-V MCU プロジェクトの Programmer 設定は、使用する FPGA デバイスに直接関連しています。以下に、いくつかのダウンロード設定の例を示します。

1. GW1NS/R-4C

例えば、Gowin_EMPU(GW1NS-4C)の場合、図 9-3 と表 9-1 に示すとおりです。

図 9-3 ダウンロードの設定

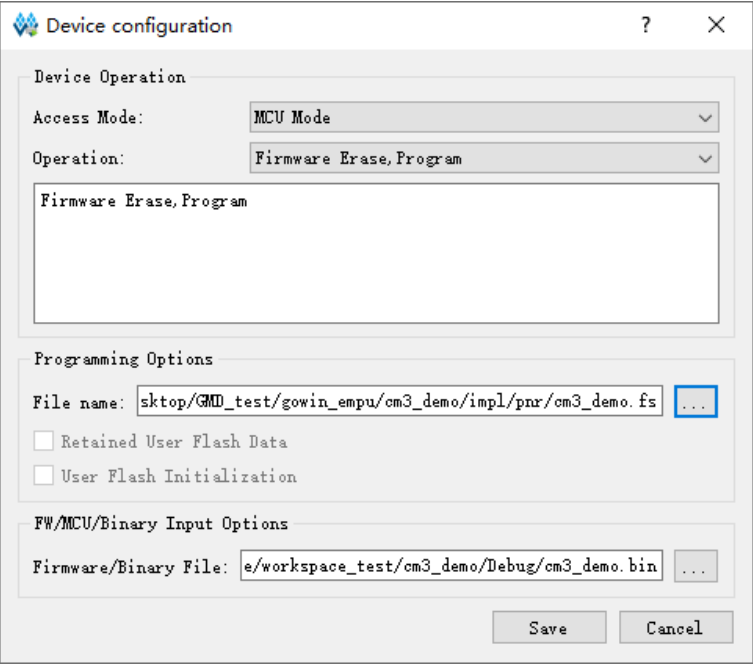


表 9-1 ダウンロードの設定

オプション	設定
Access Mode	MCU Mode
Operation	Firmware Erase, Program Firmware Erase, Program, Verify
File name	.fs ファイル
Firmware/Binary File	.bin ファイル

2. GW1INSER-4C

例えば、Secure FPGA Gowin_EMPU(GW1NS-4C)の場合、図 9-4 と表 9-2 に示すとおりです。

図 9-4 ダウンロードの設定

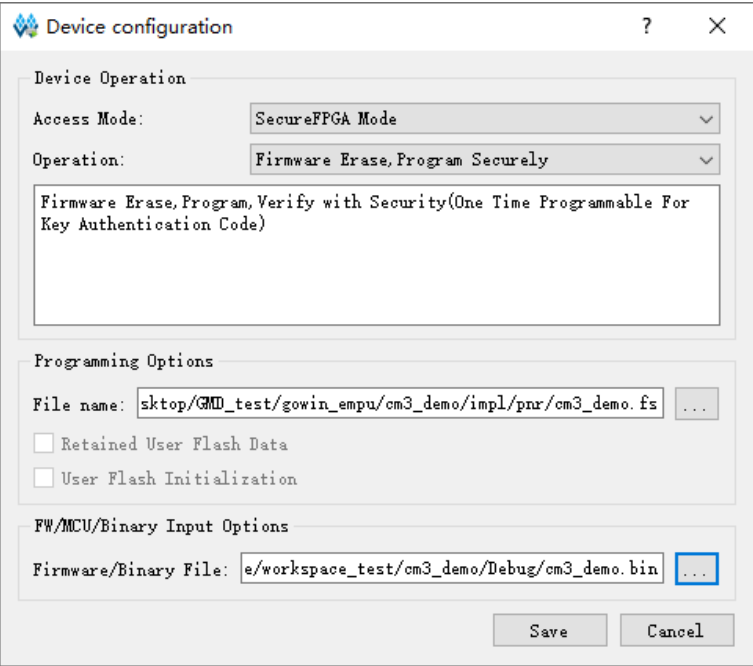


表 9-2 ダウンロードの設定

オプション	設定
Access Mode	SecureFPGA Mode
Operation	Firmware Erase, Program Securely
File name	.fs ファイル
Firmware/Binary File	.bin ファイル

3. GW1N/R-9 (C)

例えば、Gowin_EMPU_M1 と Gowin_PicoRV32 の場合、図 9-5 と表 9-3 に示すとおりです。

図 9-5 ダウンロードの設定

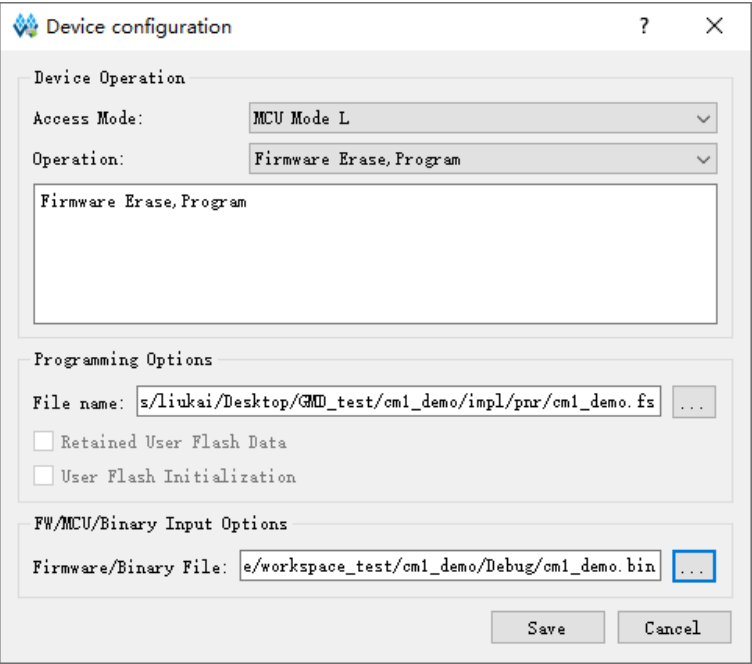


表 9-3 ダウンロードの設定

オプション	設定
Access Mode	MCU Mode L
Operation	Firmware Erase, Program Firmware Erase, Program, Verify
File name	.fs ファイル
Firmware/Binary File	.bin ファイル

4. GW2A/N/R/NR

例えば、Gowin_EMPU_M1 と Gowin_PicoRV32 の場合、図 9-6 と表 9-4 に示すとおりです。

図 9-6 ダウンロードの設定

The screenshot shows a 'Device configuration' window with the following settings:

- Device Operation:**
 - Access Mode: External Flash Mode
 - Operation: exFlash C Bin Erase, Program thru GAO-Bridge
 - Note: When programming exFlash using GAO Bridge in DUAL BOOT mode, please ensure that the "Start Address" is set to 0x000000; otherwise, the FPGA will fail to load data from exflash.
- External Flash Options:**
 - Device: Generic Flash
 - Start Address: 0x400000
- FW/MCU/Binary Input Options:**
 - Firmware/Binary File: e:/workspace_test/cml_demo/Debug/cml_demo.bin

Buttons at the bottom: Save, Cancel

表 9-4 ダウンロードの設定

オプション	設定
Access Mode	External Flash Mode
Operation	exFlash C Bin Erase, Program thru GAO-Bridge exFlash C Bin Erase, Program, Verify thru GAO-Bridge
Device	Generic Flash
Start Address	GW2A/N/NR/R-18 (C/X): 0x400000 GW2A/N-55 (C): 0x400000
Firmware/Binary File	.bin ファイル

5. GW5A/NT/NRT/R/RT/S/ST/T

例えば、Gowin_EMPU_M1 と Gowin_PicoRV32 の場合、図 9-7 と表 9-5 に示すとおりです。

図 9-7 ダウンロードの設定

Device configuration

Device Operation

Access Mode: External Flash Mode 5A

Operation: exFlash C Bin Erase, Program 5A

exFlash C Bin Erase, Program 5A

External Flash Options

Device: Generic Flash

Start Address: 0x600000

FW/MCU/Binary Input Options

Firmware/Binary File: e:/workspace_test/cm1_demo/Debug/cm1_demo.bin

Save

Cancel

表 9-5 ダウンロードの設定

オプション	設定
Access Mode	External Flash Mode 5A
Operation	exFlash C Bin Erase, Program 5A exFlash C Bin Erase, Program, Verify 5A
Device	Generic Flash
Start Address	GW5ANRT/NT/RT/T-15 (A/B): 0x100000 GW5A/R/S-25 (A/B): 0x100000 GW5A/T-60B: 0x400000 GW5AT-75 (B/C): 0x600000 GW5A/S/ST/T-138 (B/C): 0x600000
Firmware/Binary File	.bin ファイル

10 プロジェクトのデバッグ

10.1 デバッグツール

GMD ソフトウェアは、Gowin OpenOCD デバッグツールを統合しています。これは、Gowin の MCU 製品特徴向けに OpenOCD ベーシックバージョンをベースに再開発されたものです。OpenOCD は、組み込みデバイスのデバッグ、システム・プログラミング、バウンダリ・スキャン機能を提供するために設計されたオープンソースのオンチップ・デバッガです。OpenOCD の機能を使用するには、デバッグ対象の電気信号を提供する小型ハードウェア・ユニットであるデバッグ・エミュレータが必要です。

GMD ソフトウェアは、J-Link によるデバッグもサポートしています。Segger の公式ウェブサイトアクセスし、お使いのオペレーティングシステムに応じて Segger J-Link ドライバをダウンロードしてインストールしてください。

GMD ソフトウェアは、Gowin QEMU ツールを統合しています。これは、Gowin の MCU 製品特徴向けに QEMU ベーシックバージョンをベースに再開発されたものです。QEMU は、さまざまなハードウェアおよび周辺機器モデルを提供するホスト型仮想マシンであり、実際の物理プラットフォームを必要とせずに、シミュレーション機能をオンラインで直接エミュレートできます。

10.2 注意事項

Gowin MCU 製品はすべて、デバッグ中の GMD ソフトウェアによる自動ダウンロードをサポートしていません。したがって、毎回のデバッグの前に、ダウンロード・ツール「Programmer」を使用してデバッグしたい実行ファイルダウンロードしてからデバッグを開始してください。これにより、毎回のデバッグ対象が現在の設計であることが保証されます。

10.3 デバッグモード

GMD ソフトウェアは、表 10-1 に示すように、さまざまな Gowin MCU 製品、デバッグモード、およびデバッガーをサポートしています。

表 10-1 デバッグモード

MCU 製品	デバッグモード	デバッガー
Gowin_EMPU(GW1NS-4C)	GDB OpenOCD Debugging	GWU2X
		GWUSB (dual FTDI)
		GWUSB (single FTDI)
		J-Link (WinUSB driver)
	GDB SEGGER J-Link Debugging	J-Link (Segger J-Link driver)
Gowin_EMPU_M1	GDB OpenOCD Debugging	GWU2X
		GWUSB (dual FTDI)
		GWUSB (single FTDI)
		J-Link (WinUSB driver)
	GDB SEGGER J-Link Debugging	J-Link (Segger J-Link driver)
Gowin_EMPU(GW5AS-25)	GDB SEGGER J-Link Debugging	J-Link (Segger J-Link driver)
Gowin_EMPU_M3	GDB SEGGER J-Link Debugging	J-Link (Segger J-Link driver)
Gowin_PicoRV32	GDB OpenOCD Debugging	GWU2X
		GWUSB (dual FTDI)
		GWUSB (single FTDI)
		J-Link (WinUSB driver)
		Olimex
	GDB QEMU riscv32 Debugging	-

10.4 デバッグ方法

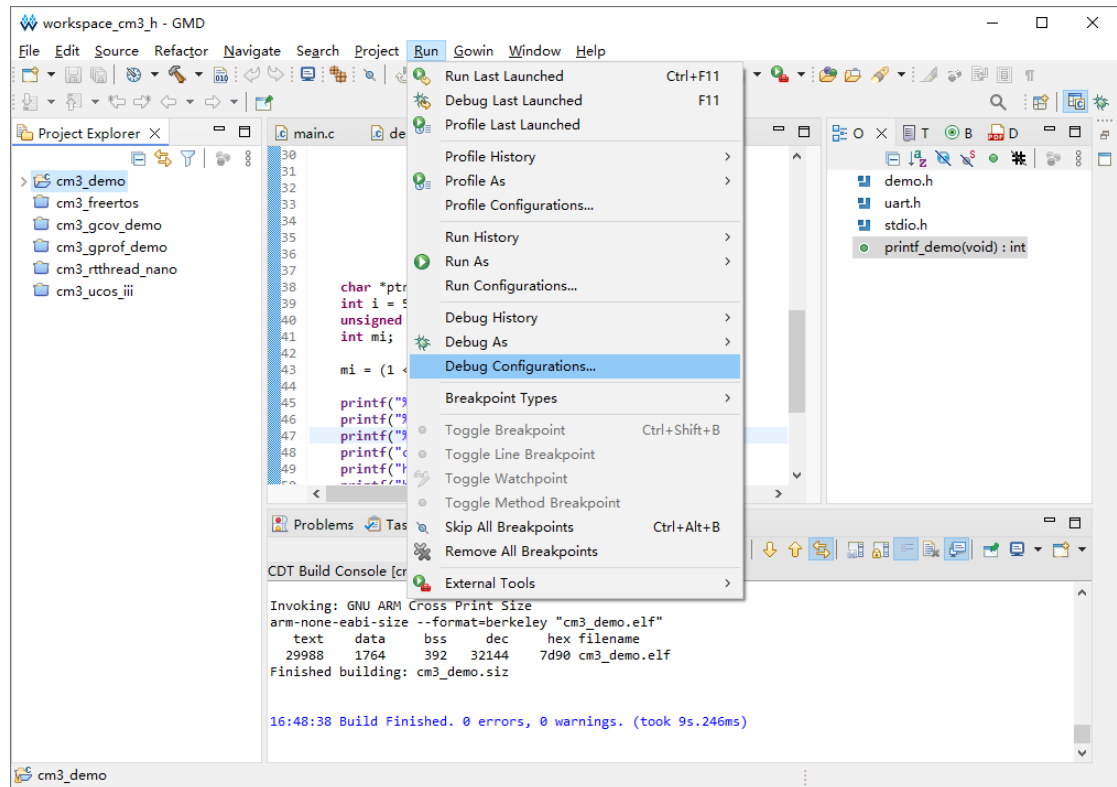
Arm MCU および RISC-V MCU には、図 10-1 に示すようにいくつかのデバッグ方法があります。

- 現在のプロジェクトを右クリックして「Debug As > Debug

Configurations…を選択します

- ツールバー > 「Debug > Debug Configurations…」 () を選択します
- メニューバー > 「Run > Debug Configurations…」 を選択します

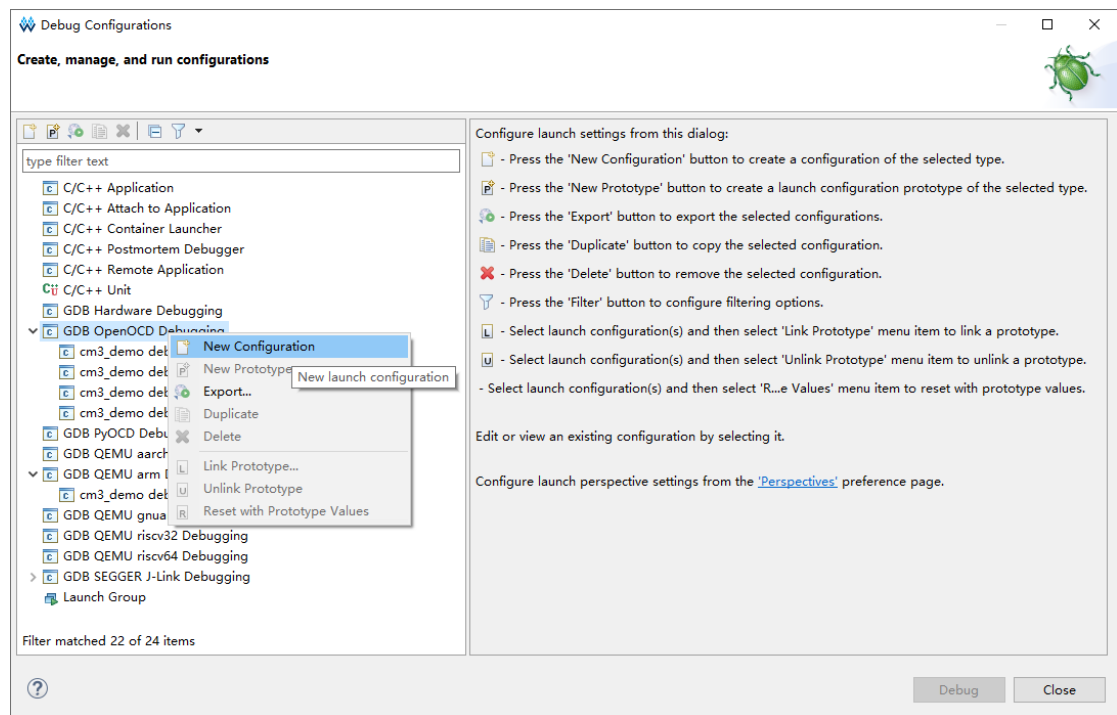
図 10-1 Debug



10.5 デバッグの設定

「Debug Configurations」画面でデバッグモードのデバッグ・ビューを作成し、デバッグ・オプションを設定します(図 10-2)。

図 10-2 デバッグ・ビューの作成



Arm MCU と RISC-V MCU のデバッグの設定は以下に示すとおりです。

10.5.1 Arm MCU のデバッグの設定

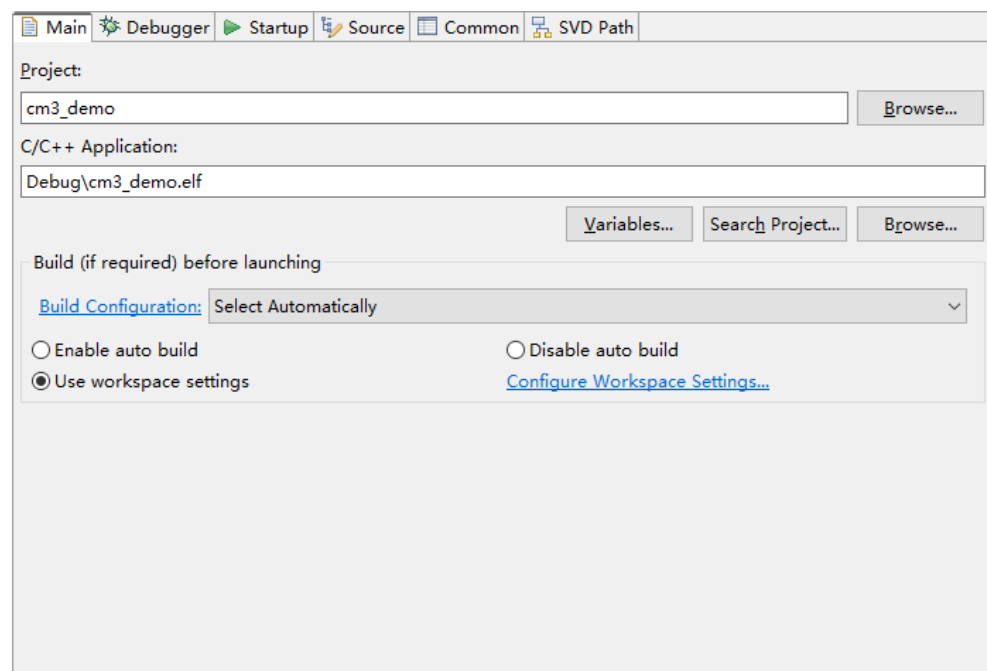
例えば、Gowin_EMPU(GW1NS-4C)は GDB OpenOCD Debugging、GDB SEGGER J-Link Debugging、および GDB QEMU Arm Debugging デバッグモードをサポートしています(表 10-1)。

GDB OpenOCD Debugging

1. Main ビュー

Main ビューでのデバッグ設定を図 10-3 に示します。

図 10-3 Main ビュー



2. Debugger ビュー

Debugger ビューでのデバッグ設定を図 10-4 と表 10-2 に示します。

図 10-4 Debugger ビュー

Debugger Setup dialog box showing configuration for OpenOCD and GDB.

OpenOCD Setup

- ☒ Start OpenOCD locally
- Executable path: `${openocd_path}/${openocd_executable}` (Browse... Variables...)
- Actual executable: `E:\GMD\toolchain\gowin\openocd\bin\openocd.exe`
(to change it use the [global](#) or [workspace](#) preferences pages or the [project](#) properties page)
- GDB port: `3333`
- Telnet port: `4444`
- Tcl port: `6666`
- Config options: `-f board/gowin_u2x_m3.cfg`
- ☒ Allocate console for OpenOCD ☐ Allocate console for the telnet connection

GDB Client Setup

- ☒ Start GDB session
- Executable name: `${cross_prefix}gdb${cross_suffix}` (Browse... Variables...)
- Actual executable: `arm-none-eabi-gdb`
- Other options:
- Commands: `set mem inaccessible-by-default off`

Remote Target

- Host name or IP address: `localhost`
- Port number: `3333`
- ☐ Force thread list update on suspend

表 10-2 Debugger ビュー

オプション	設定
Start OpenOCD locally	√
GDB port	3333
Config options	-f board/x.cfg Arm MCU 製品の x.cfg 対応関係は表 10-3 に示すとおりです。
Start GDB session	√
Commands	set mem inaccessible-by-default off

表 10-3 OpenOCD 構成ファイル

Arm MCU 製品	デバッガー	構成ファイル
Gowin_EMPU(GW1NS-4C)	GWU2X	gowin_u2x_m3.cfg
	GWUSB (dual FTDI)	gowin_ftdi_dual_m3.cfg
	GWUSB (single FTDI)	gowin_ftdi_single_m3.cfg

Arm MCU 製品	デバッガー	構成ファイル
	J-Link (WinUSB driver)	gowin_jlink_m3.cfg
Gowin_EMPU_M1	GWU2X	gowin_u2x_m1.cfg
	GWUSB (dual FTDI)	gowin_ftdi_dual_m1.cfg
	GWUSB (single FTDI)	gowin_ftdi_single_m1.cfg
	J-Link (WinUSB driver)	gowin_jlink_m1.cfg

3. Startup ビュー

Startup ビューでのデバッグ設定を図 10-5 と表 10-4 に示します。

図 10-5 Startup ビュー

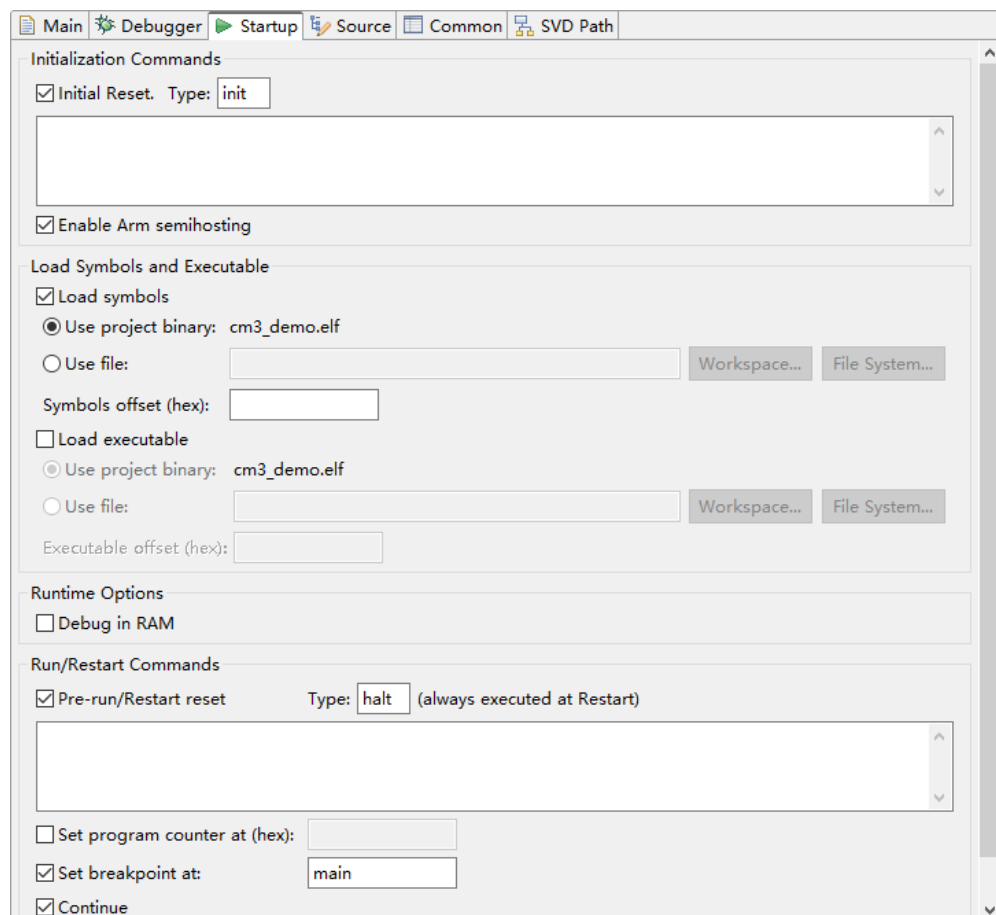


表 10-4 Startup ビュー

オプション	設定
Initial Reset.	√
Type	init
Enable Arm semihosting	√

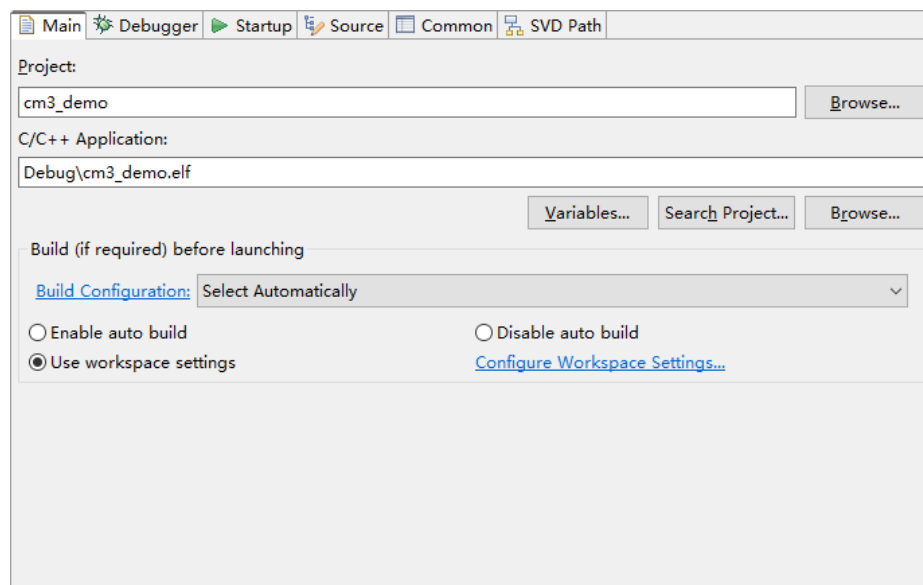
オプション	設定
Load symbols	√
Load executable	×
Debug in RAM	×
Pre-run/Restart reset	√
Type	halt
Set program counter at (hex)	-
Set breakpoint at	main
Continue	√

GDB SEGGER J-Link Debugging

1. Main ビュー

Main ビューでのデバッグ設定を図 10-6 に示します。

図 10-6 Main ビュー



2. Debugger ビュー

Debugger ビューでのデバッグ設定を図 10-7 と表 10-5 に示します。

図 10-7 Debugger ビュー

J-Link GDB Server Setup

☒ Start the J-Link GDB server locally ☐ Connect to running target

Executable path:

Actual executable: C:/Program Files (x86)/SEGGER/JLink/JLinkGDBServerCL.exe
(to change it use the [global](#) or [workspace](#) preferences pages or the [project](#) properties page)

Device name: [Supported device names](#)

Endianness: ☒ Little ☐ Big

Connection: ☒ USB ☐ IP (USB serial or IP name/address)

Interface: ☐ SWD ☒ JTAG

Initial speed: ☐ Auto ☐ Adaptive ☒ Fixed kHz

GDB port:

SWO port: ☒ Verify downloads ☒ Initialize registers on start

Telnet port: ☒ Local host only ☐ Silent

Log file:

Other options:

☒ Allocate console for the GDB server ☒ Allocate console for semihosting and SWO

GDB Client Setup

Executable name:

Actual executable: arm-none-eabi-gdb

Other options:

Commands:

Remote Target

Host name or IP address:

Port number:

☐ Force thread list update on suspend

表 10-5 Debugger ビュー

オプション	設定
Start the J-Link GDB server locally	√
Connect to running target	×
Device name	例えば、Cortex-M3。 Arm MCU 製品に対応するモデルは表 10-6 に示すとおりです。
Endianness	Little
Connection	USB
Interface	JTAG SWD

オプション	設定
Initial speed	Fixed
GDB port	2331
Verify downloads	√
Initialize registers on start	√
Local host only	√
Silent	-
Other options	-singlerun -strict -timeout 0 -nogui
Allocate console for semihosting and SWO	√
Commands	set mem inaccessible-by-default off

表 10-6 Arm MCU デバイスのモデル

Arm MCU 製品	デバイスのモデル
Gowin_EMPU(GW1NS-4C)	Cortex-M3
Gowin_EMPU_M1	Cortex-M1
Gowin_EMPU(GW5AS-25)	Cortex-M4
Gowin_EMPU_M3	Cortex-M3

3. Startup ビュー

Startup ビューでのデバッグ設定を図 10-8 と表 10-7 に示します。

図 10-8 Startup ビュー

Main Debugger Startup Source Common SVD Path

Initialization Commands
☒ Initial Reset and Halt Type: Low speed: kHz
 JTAG/SWD Speed: ☒ Auto ☐ Adaptive ☐ Fixed kHz
☒ Enable flash breakpoints
☒ Enable semihosting Console routed to: ☒ Telnet ☐ GDB client
☒ Enable SWO CPU freq: Hz. SWO freq: Hz. Port mask:

Load Symbols and Executable
☒ Load symbols
☒ Use project binary: cm3_demo.elf
☐ Use file: Workspace... File System...
 Symbols offset (hex):
☐ Load executable
☒ Use project binary: cm3_demo.elf
☐ Use file: Workspace... File System...
 Executable offset (hex):

Runtime Options
☐ RAM application (reload after each reset/restart)

Run/Restart Commands
☒ Pre-run/Restart reset Type: (always executed at Restart)
☐ Set program counter at (hex):
☒ Set breakpoint at:
☒ Continue

[Restore defaults](#)

表 10-7 Startup ビュー

オプション	設定
Initial Reset and Halt	√
JTAG/SWD Speed	Auto
Enable flash breakpoints	√
Enable semihosting Console routed to	Telnet
Load symbols	√
Load executable	×
RAM application (reload after each reset/restart)	×

オプション	設定
Pre-run/Restart reset	√
Set program counter at (hex)	-
Set breakpoint at	main
Continue	√

GDB QEMU arm Debugging

Arm MCU 製品の QEMU デバッグについては、11 QEMU によるエミュレーション・デバッグを参照してください。

10.5.2 RISC-MCU のデバッグの設定

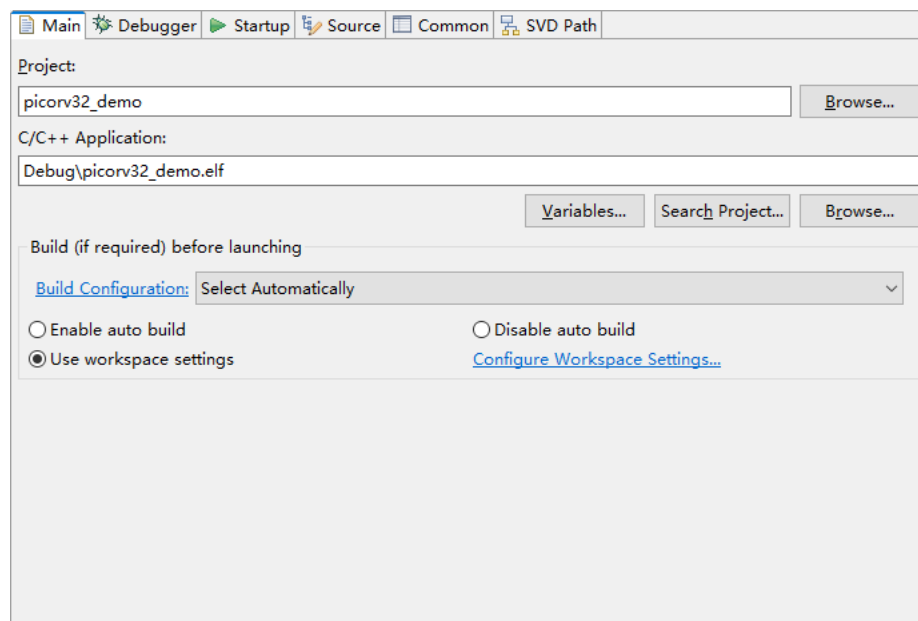
例えば、Gowin_PicoRV32 は GDB OpenOCD Debugging、GDB QEMU riscv32 Debugging デバッグモードをサポートしています(表 10-1)。

GDB OpenOCD Debugging

1. Main ビュー

Main ビューでのデバッグ設定を図 10-9 に示します。

図 10-9 Main ビュー



2. Debugger ビュー

Debugger ビューでのデバッグ設定を図 10-10 と表 10-8 に示します。

図 10-10 Debugger ビュー

Debugger Setup dialog showing configuration for OpenOCD and GDB.

OpenOCD Setup

- ☒ Start OpenOCD locally
- Executable path: `${openocd_path}/${openocd_executable}` (Browse... Variables...)
- Actual executable: `E:\GMD\toolchain\gowin\openocd\bin\openocd.exe`
(to change it use the [global](#) or [workspace](#) preferences pages or the [project](#) properties page)
- GDB port: `3333`
- Telnet port: `4444`
- Tcl port: `6666`
- Config options: `-f board/gowin_u2x_pico.cfg`
- ☒ Allocate console for OpenOCD ☐ Allocate console for the telnet connection

GDB Client Setup

- ☒ Start GDB session
- Executable name: `${cross_prefix}gdb${cross_suffix}` (Browse... Variables...)
- Actual executable: `riscv-none-elf-gdb`
- Other options: (empty)
- Commands: `set mem inaccessible-by-default off`
`set arch riscv:rv32`
`set remotetimeout 250`

Remote Target

- Host name or IP address: `localhost`
- Port number: `3333`
- ☐ Force thread list update on suspend

表 10-8 Debugger ビュー

オプション	設定
Start OpenOCD locally	√
GDB port	3333
Config options	-f board/x.cfg RISC-V MCU 製品の x.cfg 対応関係は表 10-9 に示すとおりです。
Start GDB session	√
Commands	set mem inaccessible-by-default off set arch riscv:rv32 set remotetimeout 250

表 10-9 OpenOCD 構成ファイル

RISC-V MCU 製品	デバッガー	構成ファイル
Gowin_PicoRV32	GWU2X	gowin_u2x_pico.cfg
	GWUSB (dual FTDI)	gowin_ftdi_dual_pico.cfg
	GWUSB (single FTDI)	gowin_ftdi_single_pico.cfg
	J-Link (WinUSB driver)	gowin_jlink_pico.cfg
	Olimex	gowin_olimex_pico.cfg

3. Startup ビュー

Startup ビューでのデバッグ設定を図 10-11 と表 10-10 に示します。

図 10-11 Startup ビュー

The screenshot shows the 'Startup' configuration window in OpenOCD. The 'Startup' tab is selected. The 'Initialization Commands' section has 'Initial Reset' checked with a type of 'init'. The 'Load Symbols and Executable' section has both 'Load symbols' and 'Load executable' checked, with 'Use project binary' selected for both, pointing to 'picorv32_demo.elf'. The 'Runtime Options' section has 'Debug in RAM' checked. The 'Run/Restart Commands' section has 'Pre-run/Restart reset' checked with a type of 'halt', 'Set breakpoint at' checked and set to 'main', and 'Continue' checked.

表 10-10 Startup ビュー

オプション	設定
Initial Reset.	√

オプション	設定
Type	init
Enable Arm semihosting	×
Load symbols	√
Load executable	×
Debug in RAM	√
Pre-run/Restart reset	√
Type	halt
Set program counter at (hex)	-
Set breakpoint at	main
Continue	√

GDB QEMU riscv32 Debugging

RISC-V MCU 製品の QEMU デバッグについては、11 QEMU によるエミュレーション・デバッグを参照してください。

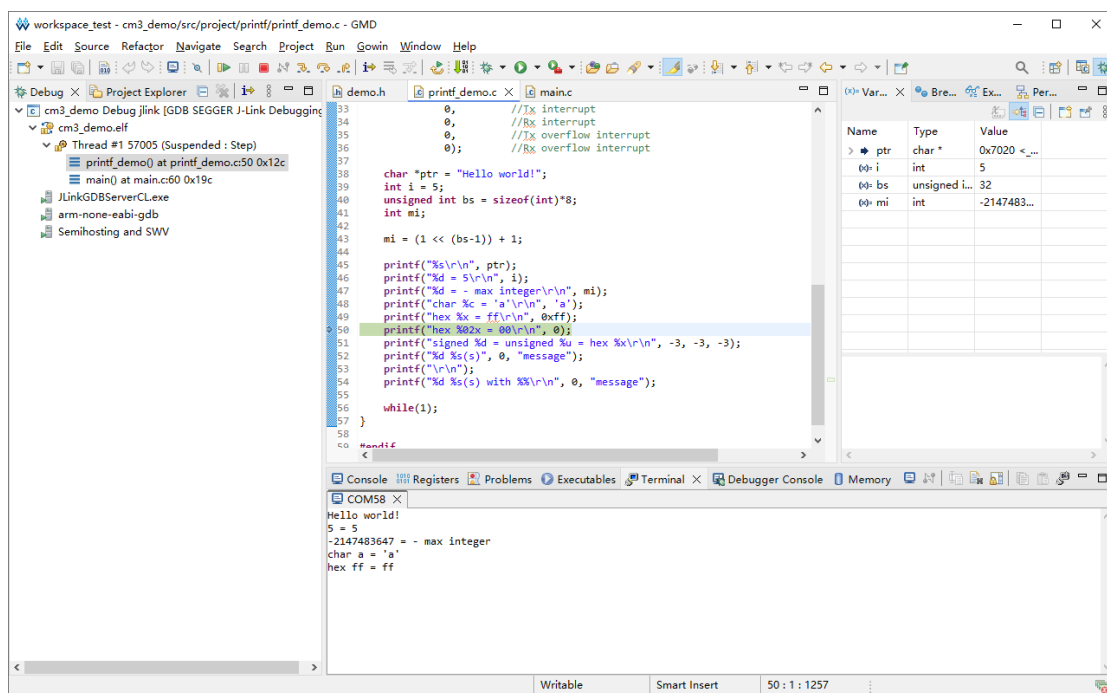
10.6 プロジェクトの実行

例えば、Gowin_EMPU(GW1NS-4C)プロジェクトを実行する場合、Segger J-Link エミュレーターを使用できます。

10.6.1 デバッグ・ビューの起動

デバッグの設定後、「Debug」をクリックしてデバッグ・ビューを起動し、プロジェクトを実行します(図 10-12)。

図 10-12 デバッグ・ビューの起動



10.6.2 デバッグ機能のオプション

デバッグ機能のオプションは表 10-11 に示すとおりです。

表 10-11 デバッグ機能のオプション

オプション	機能の説明
Resume ()	一時停止中のデバッグ・スレッドの実行を継続します
Suspend ()	デバッグ・スレッドを一時停止します
Terminate ()	デバッグ・スレッドを停止します
Step Into ()	デバッグ・スレッドの現在のステートメントにステップインし、サブ関数呼び出しを実行します
Step Over ()	デバッグ・スレッドの現在のステートメントにステップインし、サブ関数呼び出しをスキップします
Step Return ()	デバッグ・スレッドの現在のステートメントにステップインし、呼び出し元に戻るまで実行します
Restart ()	現在のデバッグ・スレッドを停止し、新しいデバッグ・スレッドを再開しま

オプション	機能の説明
	す
Instruction Stepping Mode ()	逆アセンブリ・ビューに入り、アセンブリ命令にステップインします

10.7 デバッグ・ビュー

デバッグ・スレッドを開始後、GMD ソフトウェアは対応するデバッグ・ビューを自動的に呼び出します。指定のデバッグ・ビューを呼び出すには、図 10-13 に示すように、メニューバー > 「Window > Show View > Other…」を選択します。「Show View > Debug」ドロップダウンリストから、表示するデバッグ・ビューを選択します (図 10-14)。

図 10-13 Window > Show View > Other…

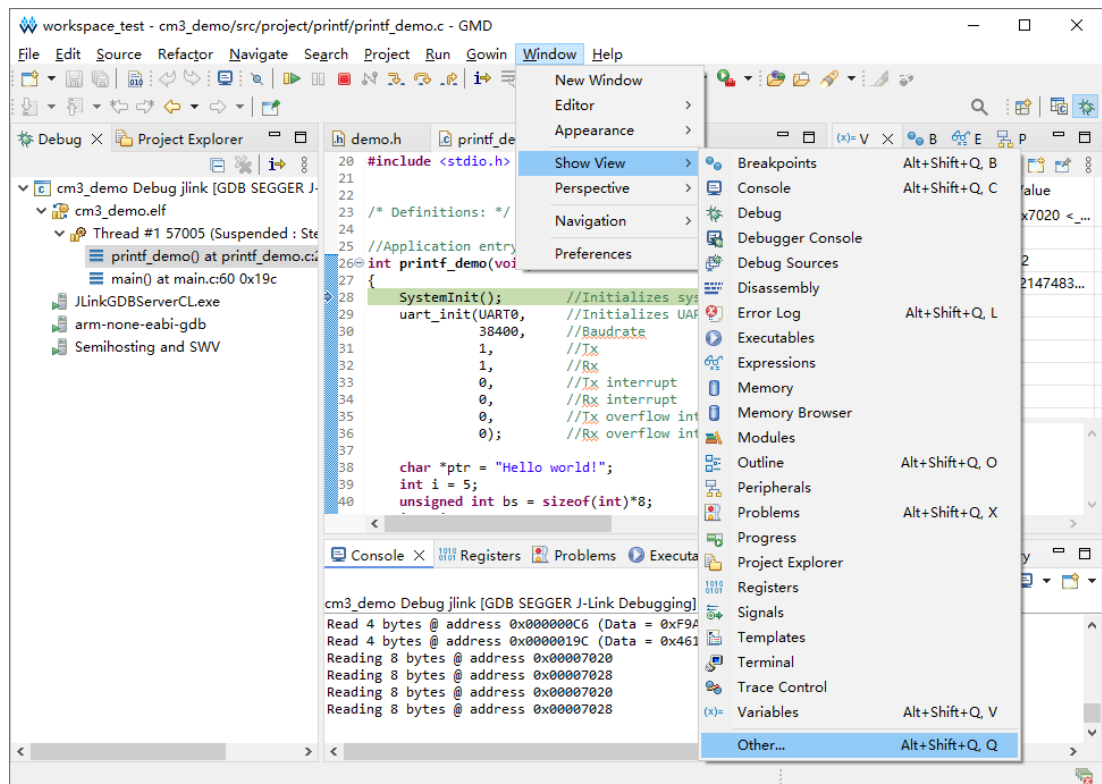
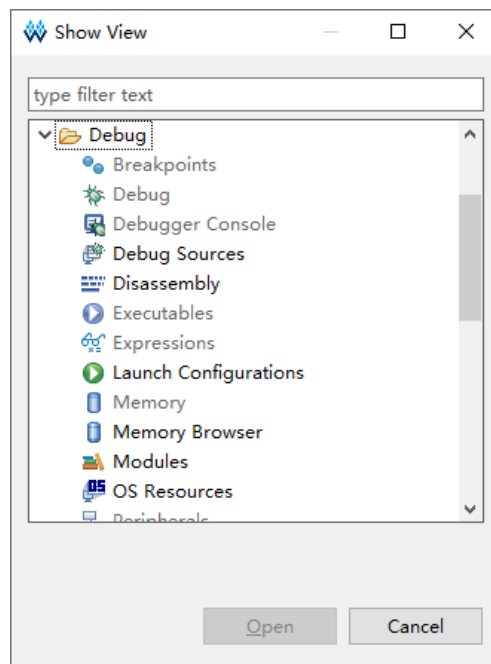


図 10-14 Debug View

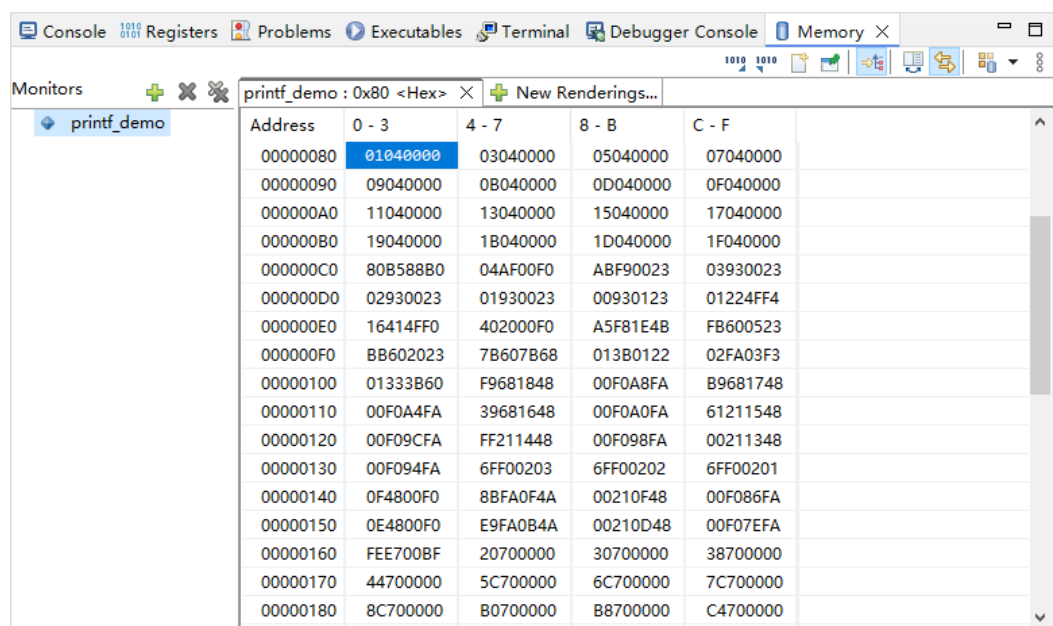


以下に、よく使用されるデバッグ・ビューをいくつか解説します。

10.7.1 メモリ・ビュー

メモリ・ビューは、メモリ・モニターのリスト形式でプログラムのメモリを検査および変更します(図 10-15)。各メモリ・モニターは、ベース・アドレスまたは式により指定されるメモリ領域に関連付けられており、各メモリ・モニターのデータは複数の定義済み形式で表示できます。

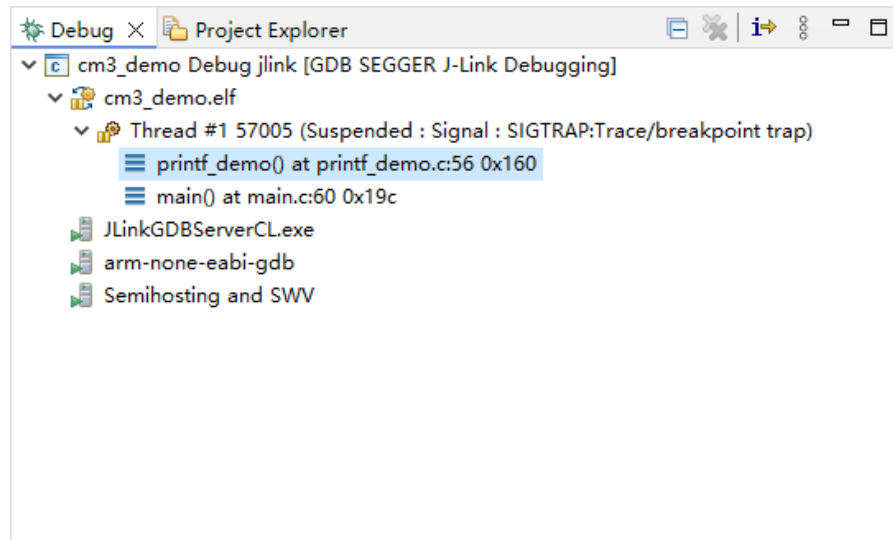
図 10-15 メモリ・ビュー



メモリ・モニターの作成

1. プログラムを一時停止した後、デバッグ・ビューから監視したいスレッドまたはスタック構造を選択します(図 10-16)。

図 10-16 監視したいスレッドまたはスタックを選択



2. メモリ・ビューで、「Monitors > Add Memory Monitor」 (+)を選択します (図 10-17)。「Monitor Memory」ウィンドウで、目的のメモリ・アドレスまたは式を入力します(図 10-18)。

図 10-17 Add Memory Monitor

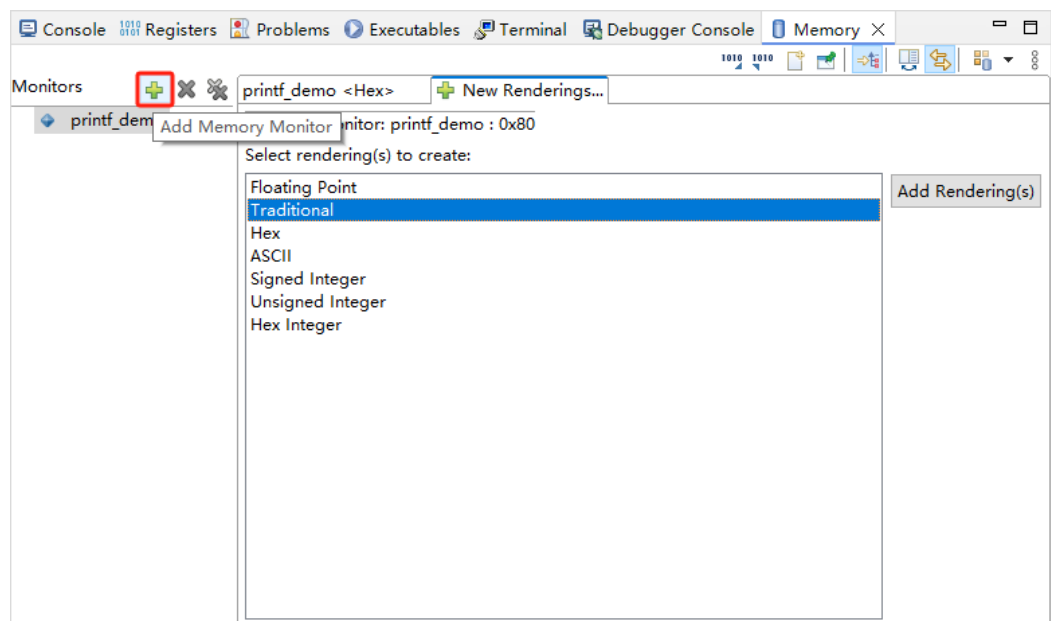
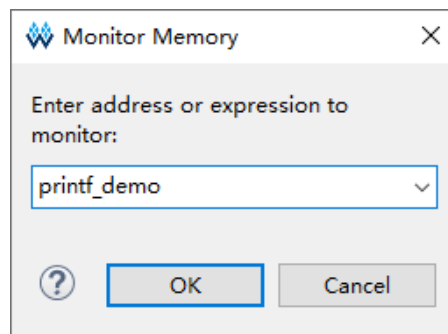
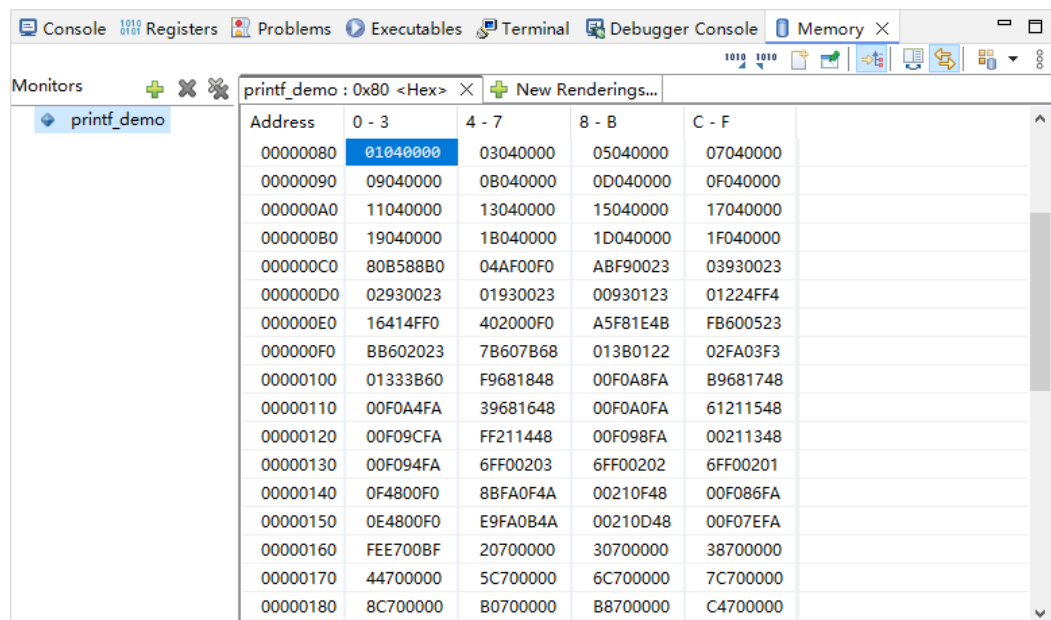


図 10-18 Monitor Memory



3. メモリ・ビューでは、デフォルトで 1 つの従来形式のメモリ・レンダラーが表示されます (図 10-19)。

図 10-19 従来形式のメモリ・レンダラー



また、「New Renderings… > Floating Point > Add Rendering(s)」を選択して(図 10-20)、浮動小数点形式のメモリ・レンダラーを作成することもできます (図 10-21)。

図 10-20 浮動小数点形式のメモリ・レンダラーを作成

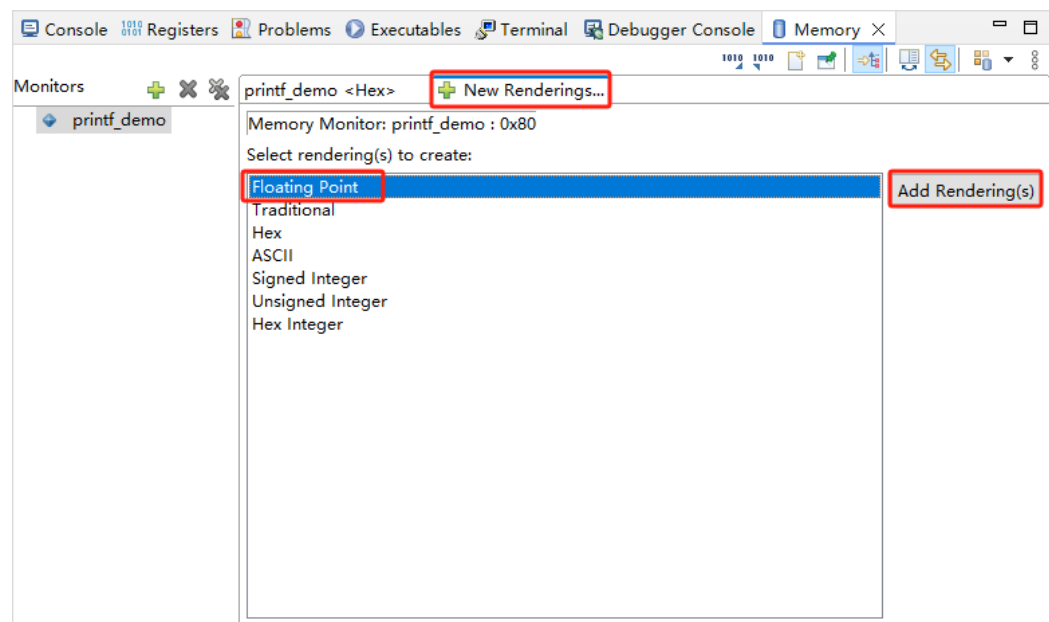
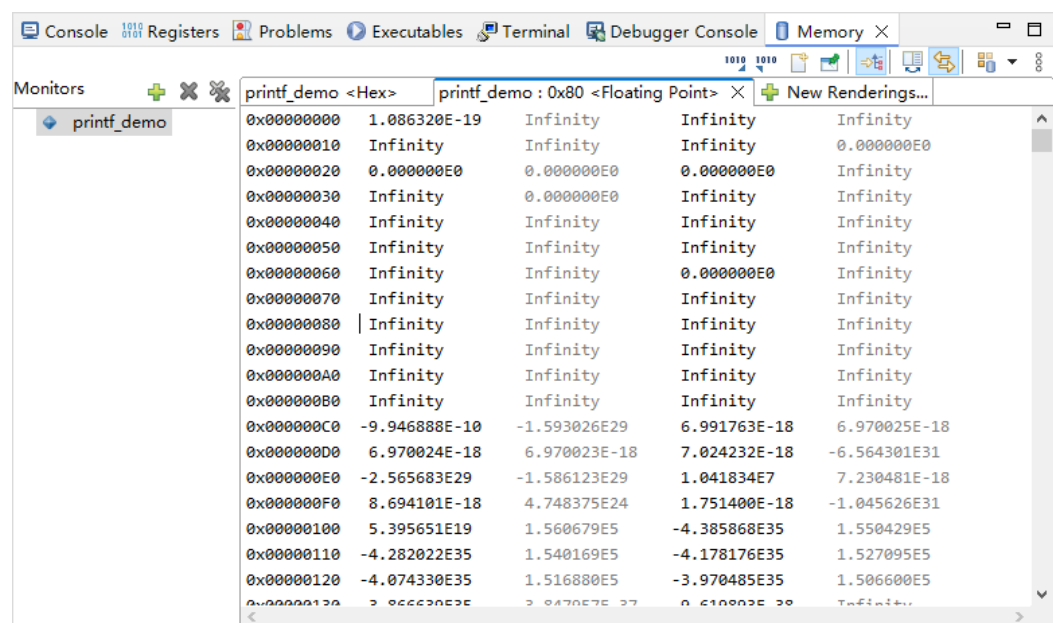
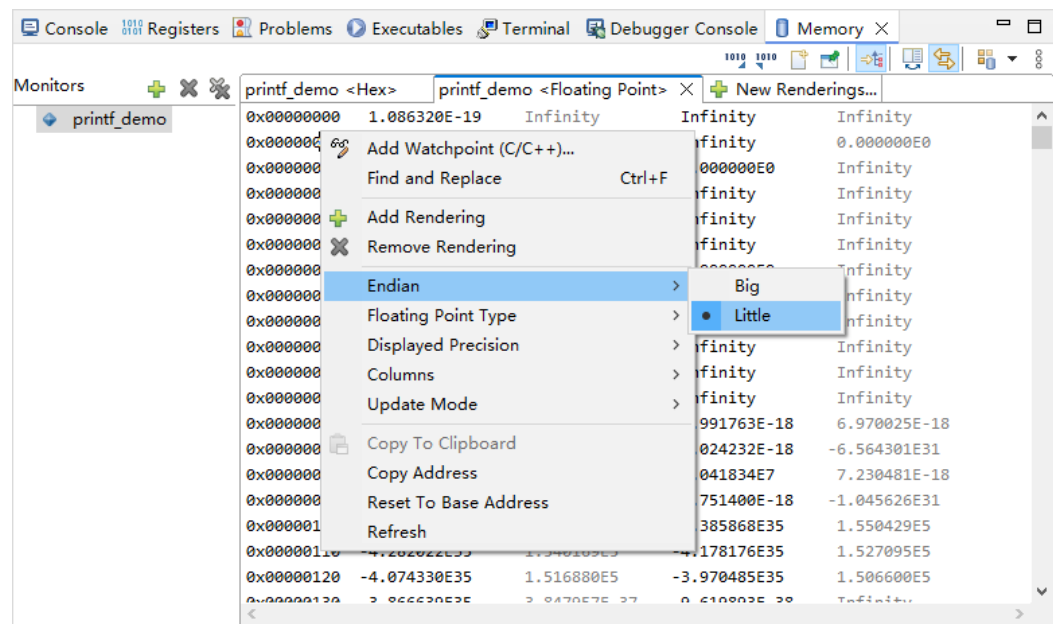


図 10-21 浮動小数点形式のメモリ・レンダラー



- メモリ・レンダラー内の任意の場所を右クリックし、ポップアップするドロップダウンメニューから「**Endian**」を選択すると、データ表示のバイトオーダーまたはアドレス単位サイズを変更できます (図 10-22)。

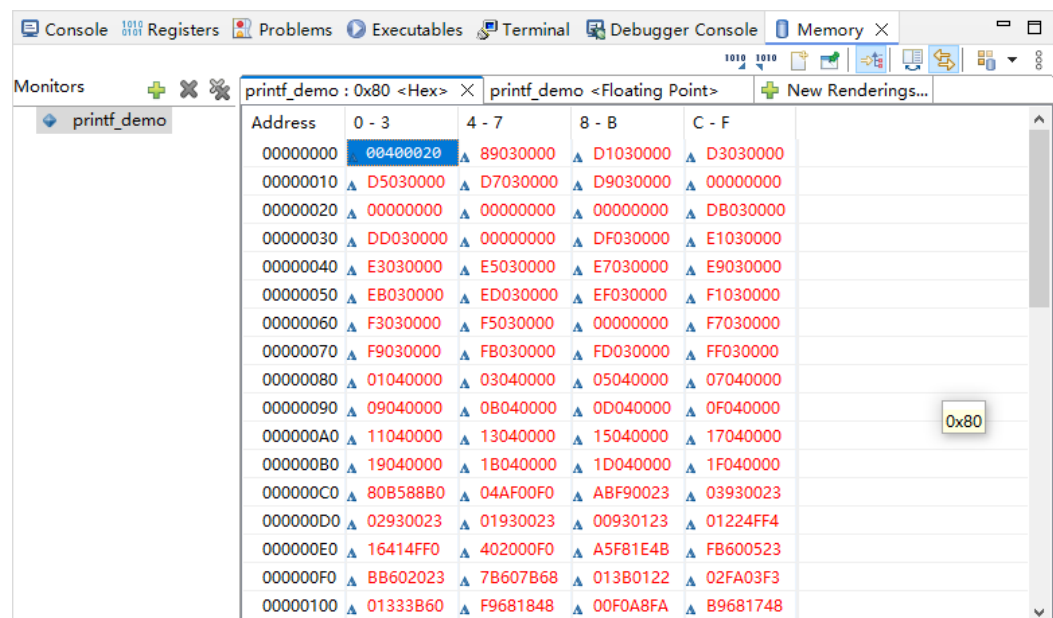
図 10-22 バイトオーダーの設定



各セル内のメモリ値を変更することもできますが、不適切な値を入力するとプログラムがクラッシュします。

5. プログラム実行中は、格納されている内容の偏差が赤で表示されます (図 10-23)。

図 10-23 格納内容の表示



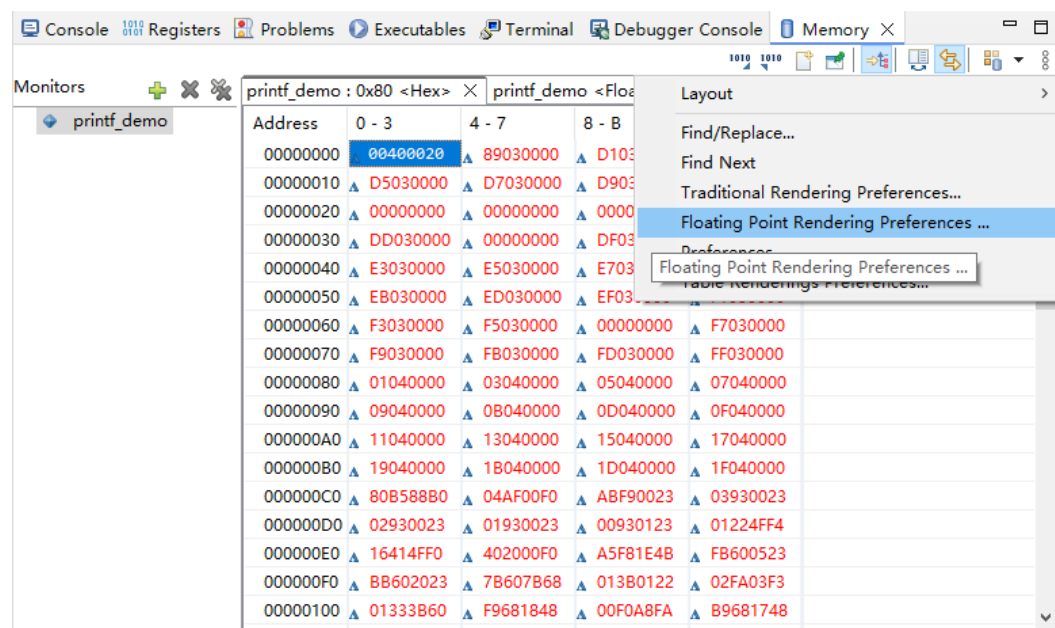
レンダリング効果の設定

メモリ・レンダラーはデフォルトで黒のテキストで表示されます。レ

レンダリングされたデータとアドレスを異なる色で区別するには、以下の手順を参照してください。

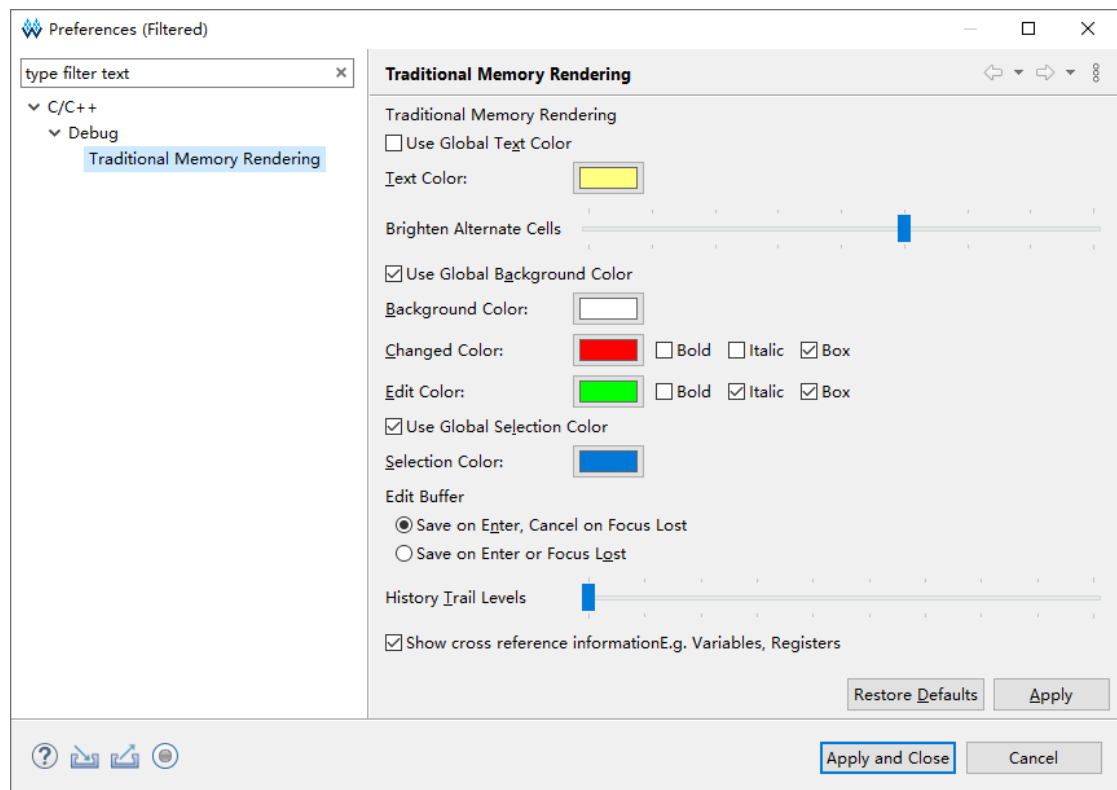
1. メモリ・ビューで「View Menu」()を選択し、ポップアップするドロップダウンメニューから「Traditional Rendering Preferences…」または「Floating Point Rendering Preferences…」を選択すると、従来の形式または浮動小数点形式のメモリレンダラーのレンダリング効果設定画面が表示されます(図 10-24)。

図 10-24 レンダリング効果の設定



2. 例えば、「Preferences > C/C++ > Debug > Traditional Memory Rendering > Use Global Text Color」のチェックを外すと、テキスト・カラーをカスタマイズできるようになります(図 10-25)。

図 10-25 メモリ・レンダリング効果の設定

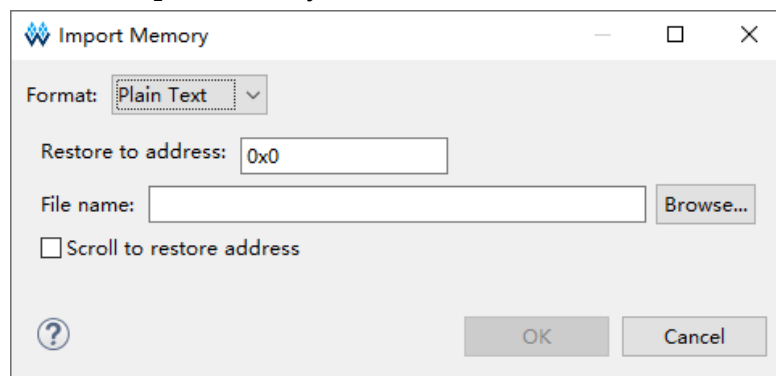


ビューのツールバー

メモリ・ビューのツールバーには、以下の機能が含まれています。

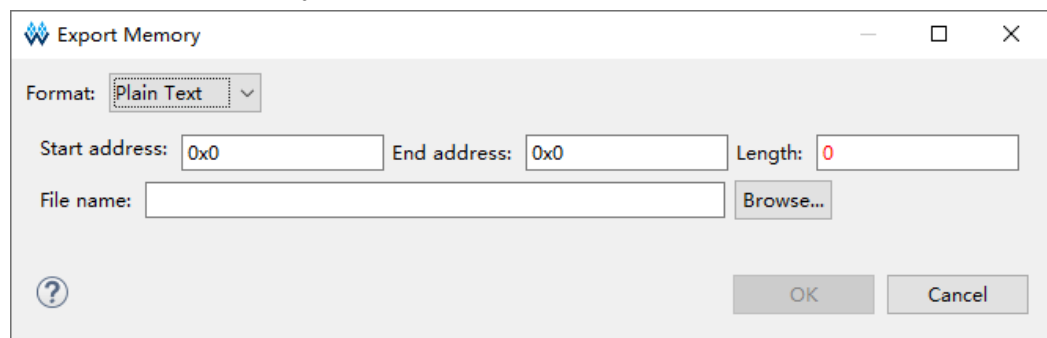
- Import () : 指定されたメモリ値のセットをインポートします(図 10-26)。

図 10-26 Import Memory



- Export() : メモリ・レンダラーによって表示されたメモリ値をファイルにエクスポートします(図 10-27)。

図 10-27 Export Memory



GMD ソフトウェアは、メモリ・データのインポート/エクスポートに、Plain Text、RAW Binary、SRecord の 3 つの形式をサポートしています。Plain Text と RAW Binary は、一般的によく使用されます。Plain Text には 16 進数が含まれ、1 バイトに 2 つの 16 進文字が格納されます。RAW Binary にはバイナリ値が含まれます。

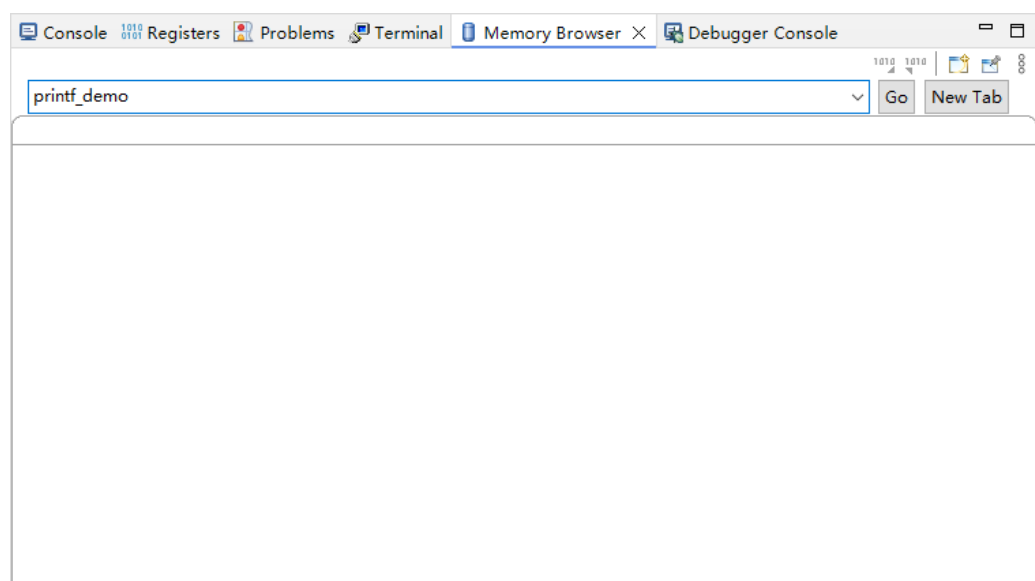
10.7.2 メモリ・ブラウザ・ビュー

メモリ・ブラウザ(Memory Brower)ビューは、プロセス・メモリの検査と変更を行うためのメモリ・ビューの代替ビューです。

メモリ・モニターの作成

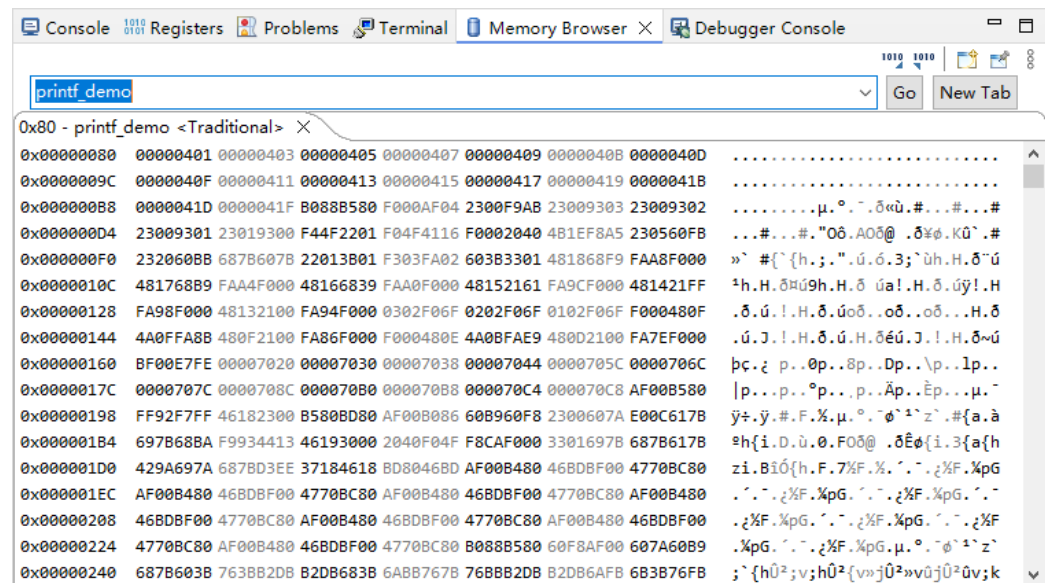
1. プログラムを一時停止後、メモリ・ブラウザ・ビューのテキストボックスに目的のメモリ・アドレスまたは式を入力し、「Go」をクリックしてメモリ・モニターを作成します(図 10-28)。

図 10-28 メモリ・ブラウザ・ビュー



2. メモリ・ブラウザ・ビューでは、デフォルトで1つの従来形式のメモリ・レンダラーが表示されます (図 10-29)。

図 10-29 従来形式のメモリ・レンダラー



浮動小数点形式のメモリ・レンダラーを作成することもできます。

「View Menu」() を選択し、ポップアップするドロップダウン・メニューから「Default Rendering > Floating Point > New Tab」をクリックして浮動小数点形式のメモリ・レンダラーを作成します (図 10-30 と図 10-31)。

図 10-30 浮動小数点形式のメモリ・レンダラーの作成

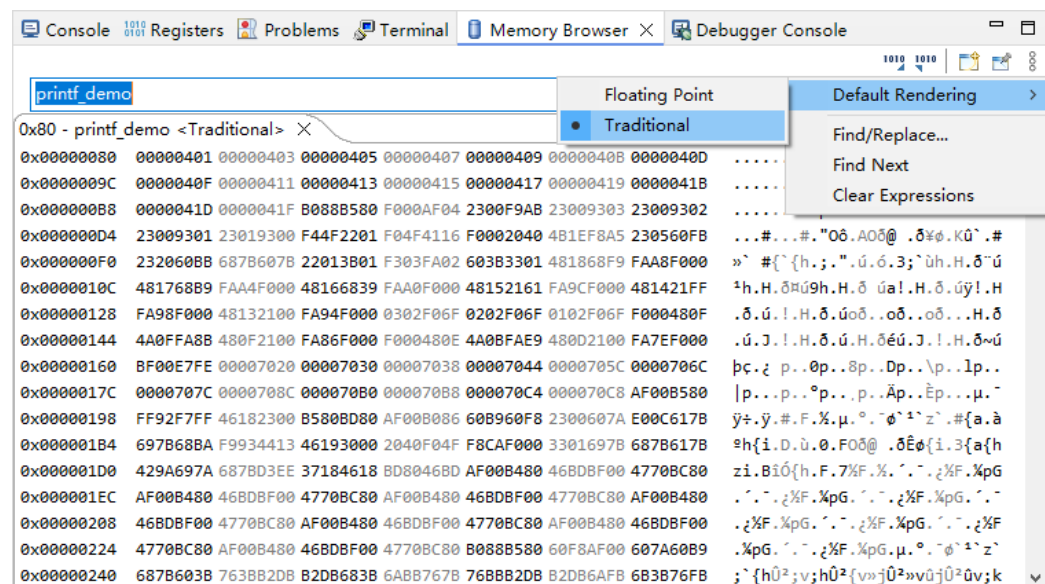
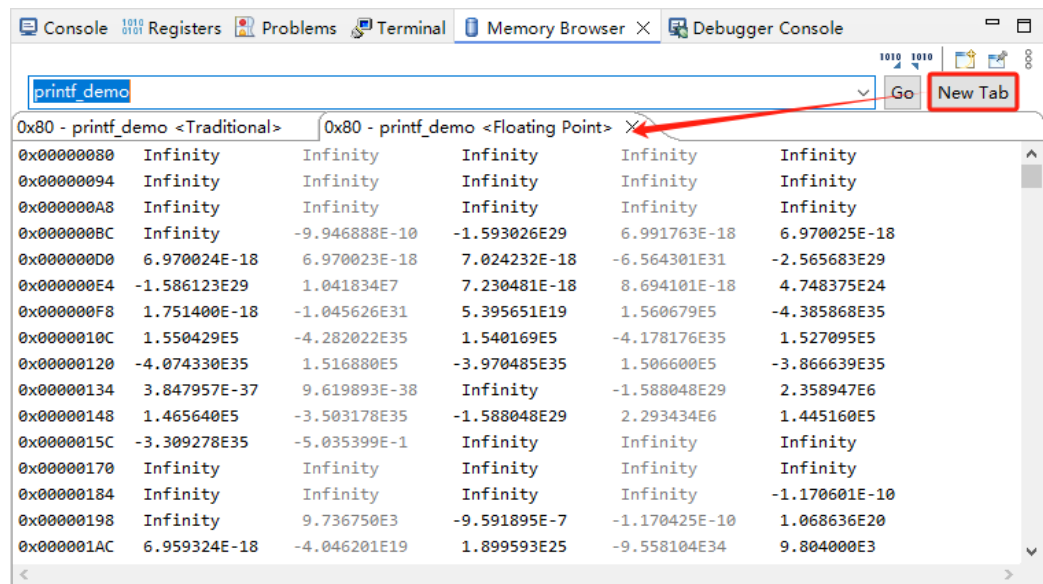
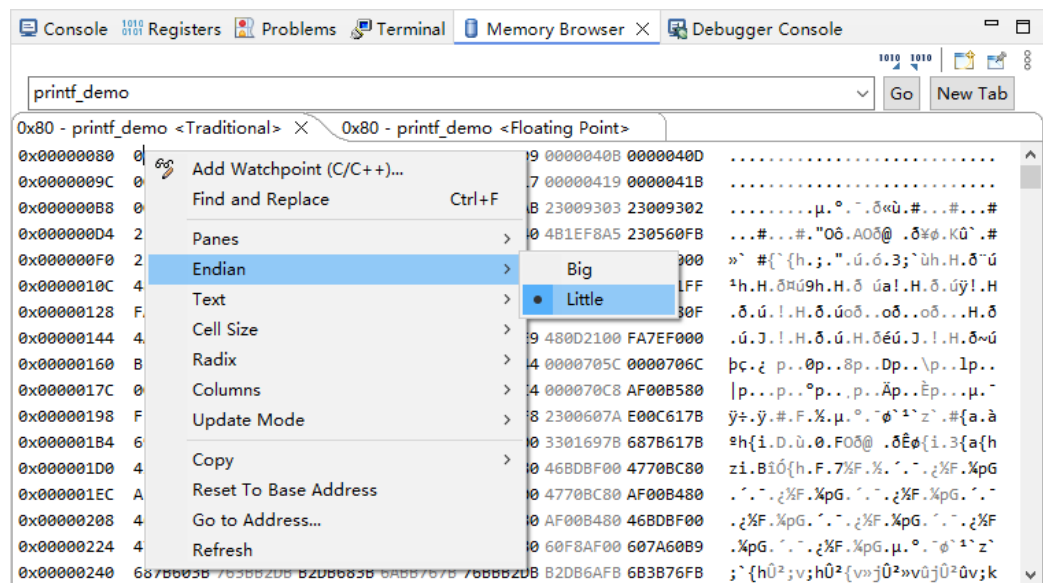


図 10-31 浮動小数点形式のメモリ・レンダラー



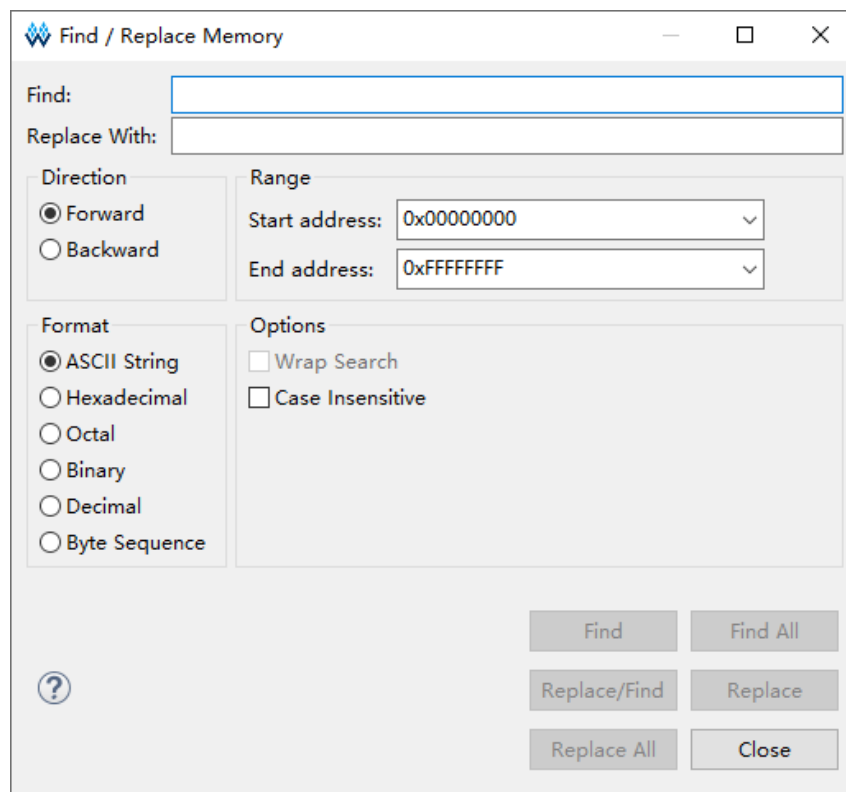
3. メモリ・レンダラー内の任意の場所を右クリックし、ポップアップするドロップダウンメニューから「**Endian**」を選択すると、データ表示のバイトオーダーまたはアドレス単位サイズを変更できます (図 10-32)。

図 10-32 バイトオーダーの設定



4. 「**View Menu**」 () のドロップダウン・メニューから「**Find/Replace...**」を選択すると、選択したメモリ範囲内の指定データを検索・置換できます (図 10-33)。

図 10-33 Find/Replace



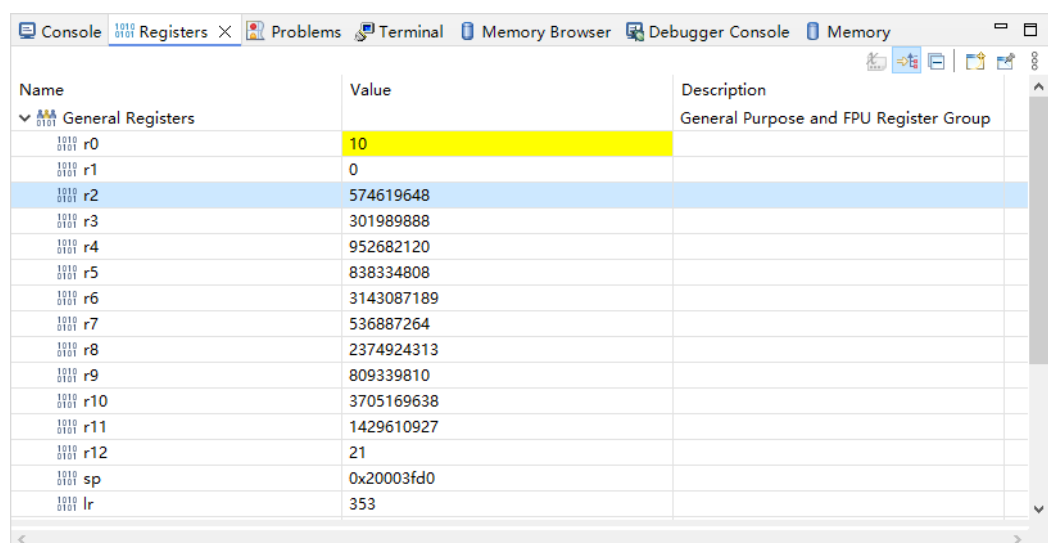
The dialog box titled "Find / Replace Memory" contains the following fields and controls:

- Find:** A text input field.
- Replace With:** A text input field.
- Direction:** Radio buttons for **Forward** (selected) and **Backward**.
- Range:** Two dropdown menus for **Start address:** (0x00000000) and **End address:** (0xFFFFFFFF).
- Format:** Radio buttons for **ASCII String** (selected), **Hexadecimal**, **Octal**, **Binary**, **Decimal**, and **Byte Sequence**.
- Options:** Checkboxes for **Wrap Search** and **Case Insensitive**.
- Buttons:** **Find**, **Find All**, **Replace/Find**, **Replace**, **Replace All**, and **Close**.
- Help:** A question mark icon in the bottom left corner.

10.7.3 レジスタ・ビュー

レジスタ(Registers)ビューには、レジスタのビット・フィールド・レベルで CPU 情報が表示されます。図 10-34 に示すように、レジスタまたはビット・フィールドの値はセル内で直接変更できます。

図 10-34 レジスタ・ビュー



The screenshot shows the "Registers" view in a debugger. It displays a table of CPU registers with their names, values, and descriptions. The "r2" register is highlighted in blue.

Name	Value	Description
General Registers		General Purpose and FPU Register Group
r0	10	
r1	0	
r2	574619648	
r3	301989888	
r4	952682120	
r5	838334808	
r6	3143087189	
r7	536887264	
r8	2374924313	
r9	809339810	
r10	3705169638	
r11	1429610927	
r12	21	
sp	0x20003fd0	
lr	353	

10.7.4 変数ビュー

変数ビューには、現在のスコープ内のすべての変数の名前と値が表示されます(図 10-35)。プログラムが一時停止している際、変数値の変化を表示および監視できます。

図 10-35 変数ビュー

[illegible]

10.7.5 式ビュー

式(Expressions)ビューでは、カスタム式を追加して値を監視できます(図 10-36)。これは、複雑な式の値を計算したり、デバッグ中にコード・スニペットをテストしたりするのに非常に便利です。

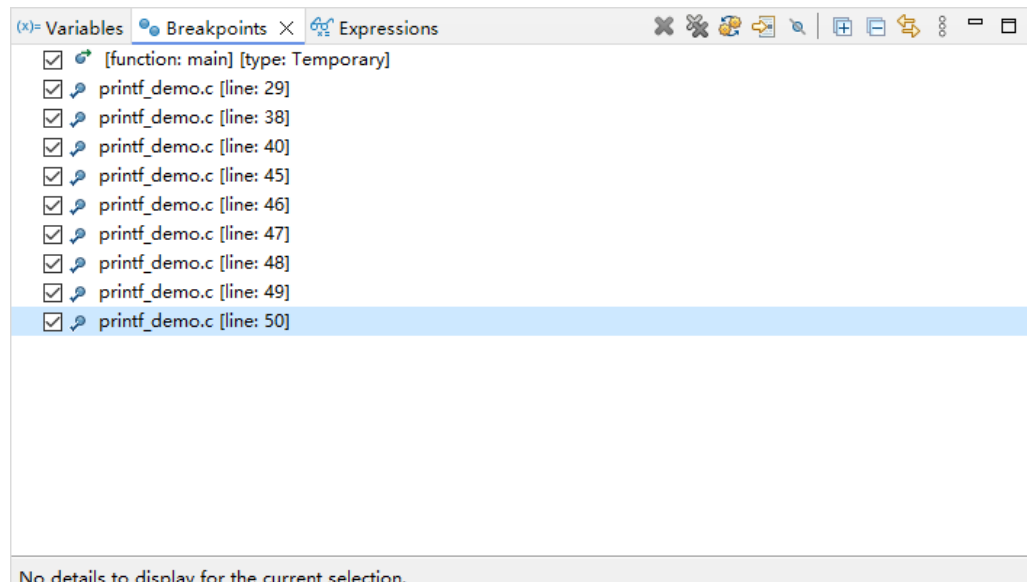
図 10-36 式ビュー

[illegible]

10.7.6 ブレークポイント・ビュー

ブレークポイント(Breakpoints)ビューには、プロジェクトに設定されているすべてのブレークポイントが一覧表示されます。ユーザーはここでブレークポイントの有効化、無効化、追加、削除、条件の設定など、ブレークポイントを管理できます(図 10-37)。

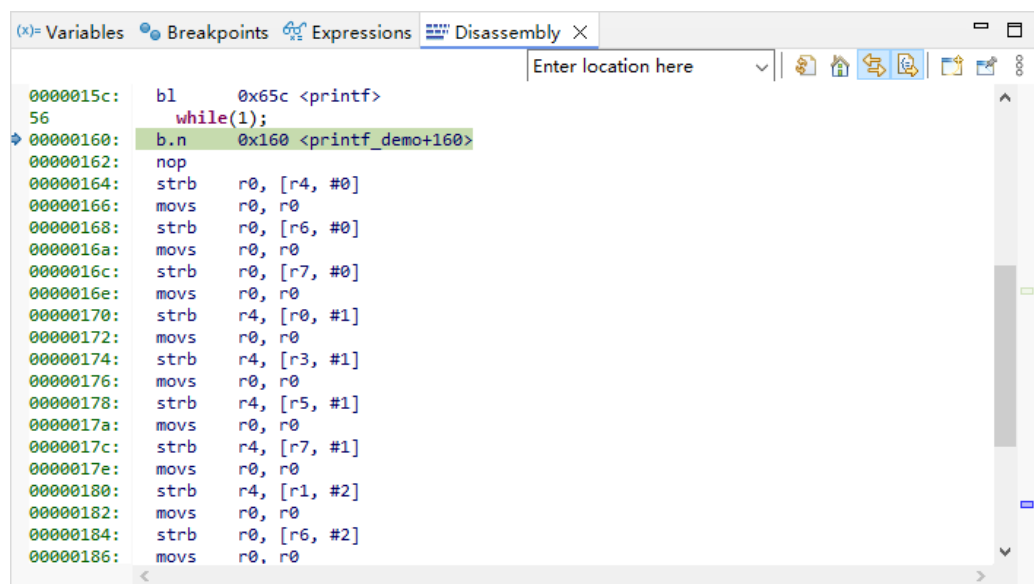
図 10-37 ブレークポイント・ビュー



10.7.7 逆アセンブリ・ビュー

逆アセンブリ(Disassembly)ビューには、高水準言語に対応するアセンブリコードが表示されます。これは、デバッグ中に命令の問題を特定するために使用されます (図 10-38)。

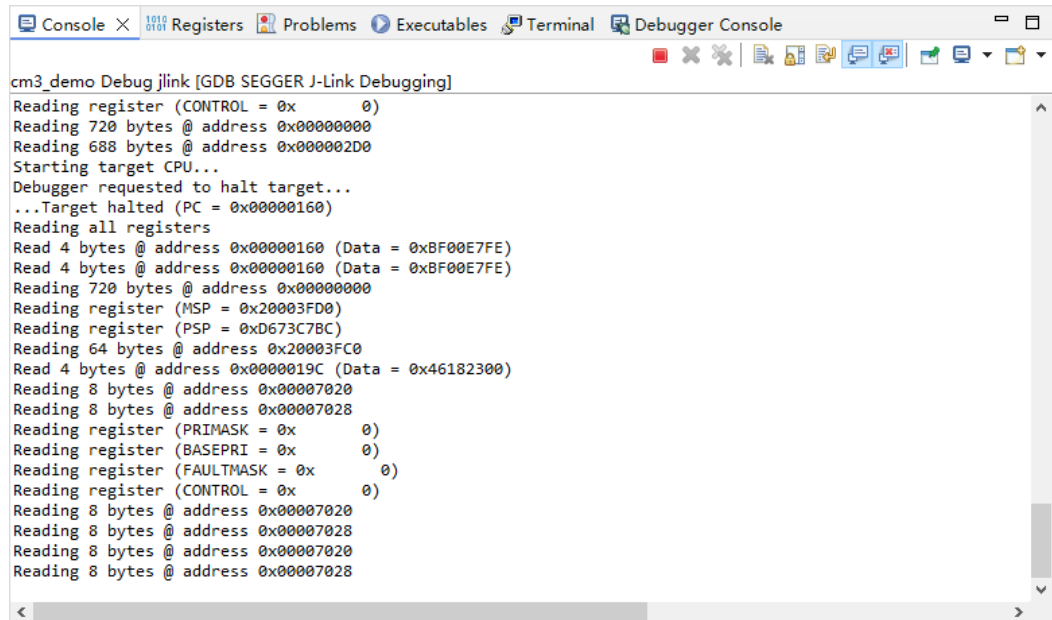
図 10-38 逆アセンブリ・ビュー



10.7.8 コンソール・ビュー

コンソール(Console)ビューには、図 10-39 に示すように、デバッグ・プロセスの出力などのプロセス出力が表示されます。

図 10-39 コンソール・ビュー



11 QEMU によるエミュレーション・デバッグ

11.1 QEMU ツール

GMD ソフトウェアは、Gowin QEMU ツールチェーンを統合しています。これは、Gowin の MCU 製品特徴向けに QEMU ベーシックバージョンをベースに再開発されたものです。QEMU は、さまざまなハードウェアおよび周辺機器モデルを提供するホスト型仮想マシンであり、実際の物理プラットフォームを必要とせずに、シミュレーション機能をオンラインで直接エミュレートできます。

GMD 2025.01 バージョンの QEMU ツールは、Gowin_EMPU (GW1NS-4C) と Gowin_PicoRV32 をすでにサポートしています。それぞれの周辺機器のサポートは表 11-1 に示すとおりです。

表 11-1 サポートされる周辺機器

MCU 製品	サポートされる周辺機器	機能の説明
Gowin_EMPU(GW1NS-4C)	UART	入出力、割り込み処理
	GPIO	入出力
	SysTick	カウント、割り込み処理
	Timer	カウント、割り込み処理
Gowin_PicoRV32	UART	入出力
	GPIO	入出力

11.2 QEMU の使用方法

QEMU 設定画面に入る方法は以下に示すとおりです。

- 現在のプロジェクトを右クリックして「Debug As > Debug Configurations…」を選択します

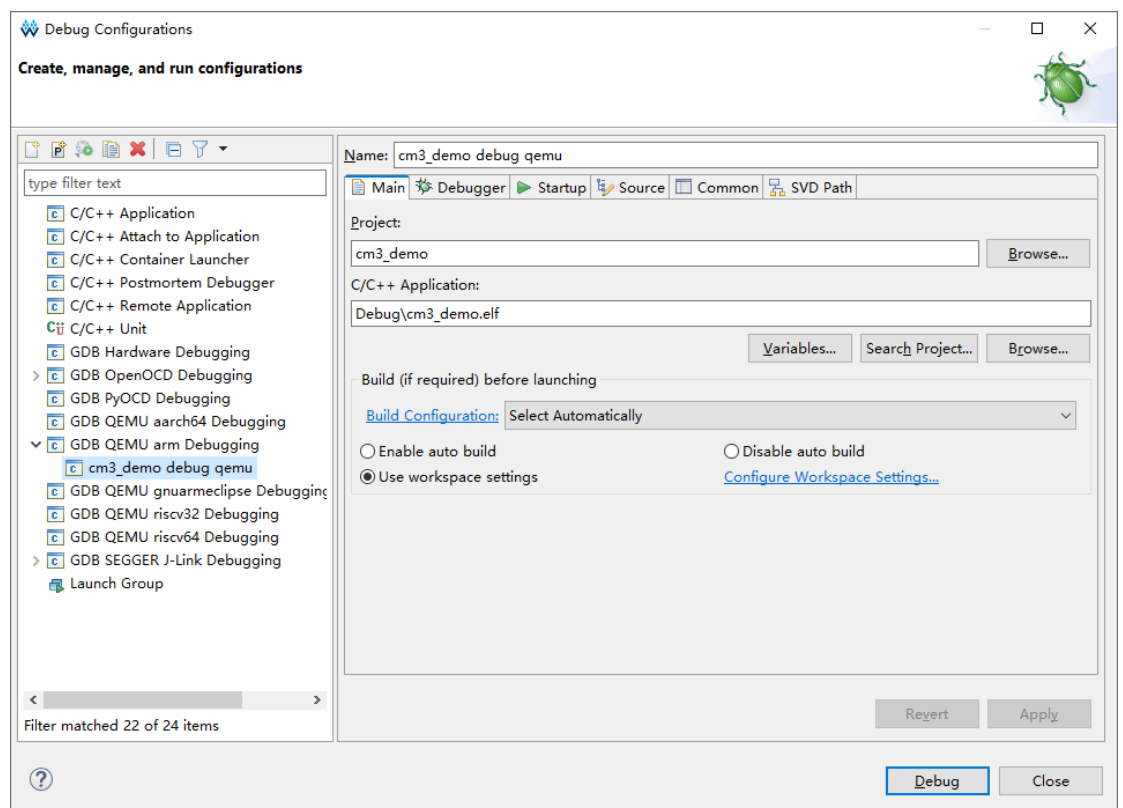
- ツールバー > 「Debug > Debug Configurations…」 (🐞) を選択します
- メニューバー > 「Run > Debug Configurations…」 を選択します

Arm MCU と RISC-V MCU は異なる QEMU ツールを使用します。その設定方法は以下に示すとおりです。

11.2.1 Arm MCU QEMU の設定

例えば、Gowin_EMPU (GW1NS-4C)の場合は、GDB QEMU arm Debugging を選択します(図 11-1)。

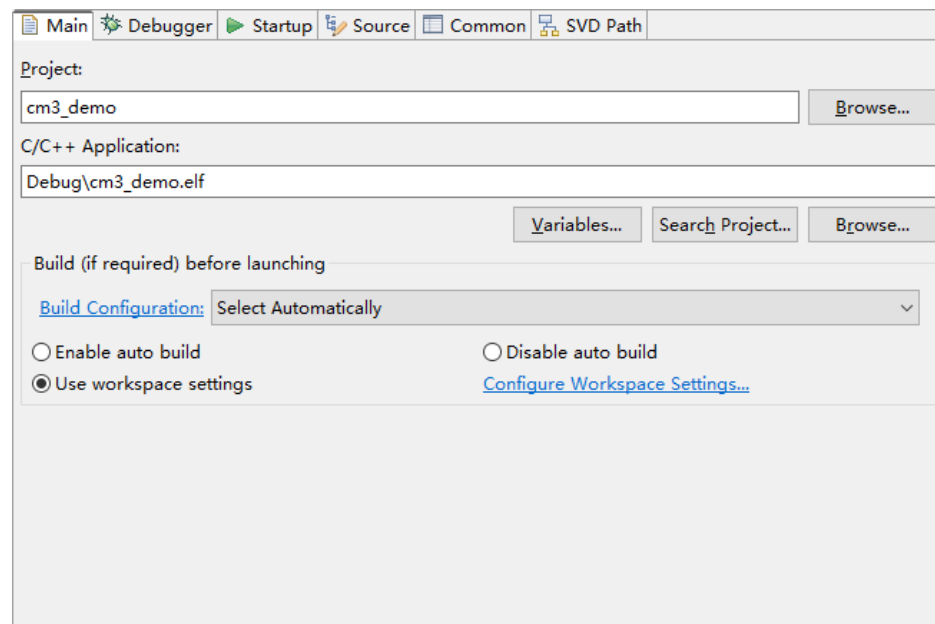
図 11-1 GDB QEMU arm Debugging



1. Main ビュー

Main ビューでの QEMU 設定を図 11-2 に示します。

図 11-2 Main ビュー



2. Debugger ビュー

Debugger ビューでの QEMU 設定を図 11-3 と表 11-2 に示します。

図 11-3 Debugger ビュー

The screenshot shows the 'Debugger' tab in an IDE. It contains three main sections: 'QEMU Setup', 'GDB Client Setup', and 'Remote Target'.

QEMU Setup:

- ☒ Start QEMU locally
- Executable path: `${qemu_arm_path}/${qemu_arm_executable}` (with 'Browse...' and 'Variables...' buttons)
- ☐ Prefer the xpacks/.bin path
- Actual executable: `E:\GMD\toolchain\gowin\qemu_arm\bin\qemu-system-arm.exe` (with a note: '(to change it use the [global](#) or [workspace](#) preferences pages or the [project](#) properties page)')
- QEMU machine name: `gowin_cm3`
- QEMU CPU name: `cortex-m3`
- GDB port: `1234`
- More options: `-serial stdio -nodefaults -S`
- ☒ Enable Arm semihosting
- ☒ Do not open graphic windows
- Command line: `cm3_demo`
- ☒ Allocate console for QEMU
- Delay (seconds): `0`

GDB Client Setup:

- Executable name: `${cross_prefix}gdb${cross_suffix}` (with 'Browse...' and 'Variables...' buttons)
- Actual executable: `arm-none-eabi-gdb`
- Other options: (empty field)
- Commands: `set mem inaccessible-by-default off`

Remote Target:

- Host name or IP address: `localhost`
- Port number: `1234`
- ☐ Force thread list update on suspend

表 11-2 Debugger ビュー

オプション	設定
Start QEMU locally	√
QEMU machine name	表 11-3 に示す通りです。
QEMU CPU name	表 11-3 に示す通りです。
GDB port	1234
More options	-serial stdio -nodefaults -S
Enable Arm semihosting	√
Do not open graphic windows	√
Allocate console for QEMU	√
Delay (second)	0
Commands	set mem inaccessible-by-default off

表 11-3 Arm MCU QEMU

Arm MCU	Machine name	CPU name
Gowin_EMPU(GW1NS-4C)	gowin_cm3	cortex-m3

3. Startup ビュー

Startup ビューでの QEMU 設定を図 11-4 と表 11-4 に示します。

図 11-4 Startup ビュー

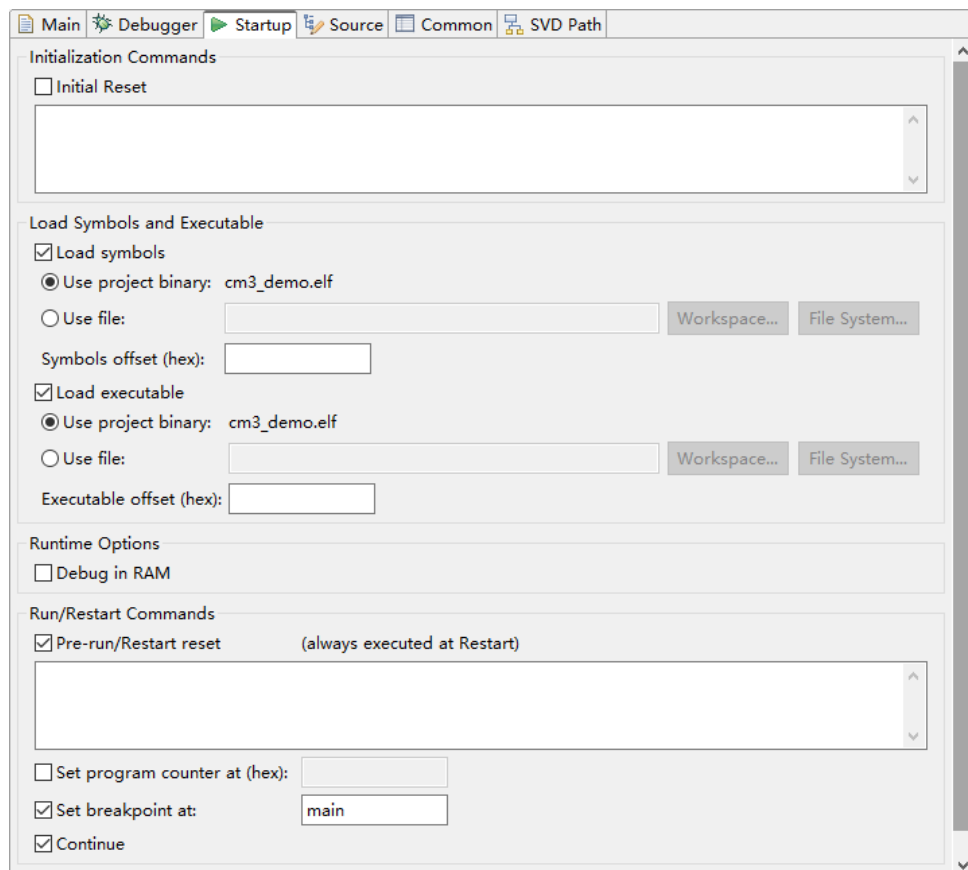


表 11-4 Startup ビュー

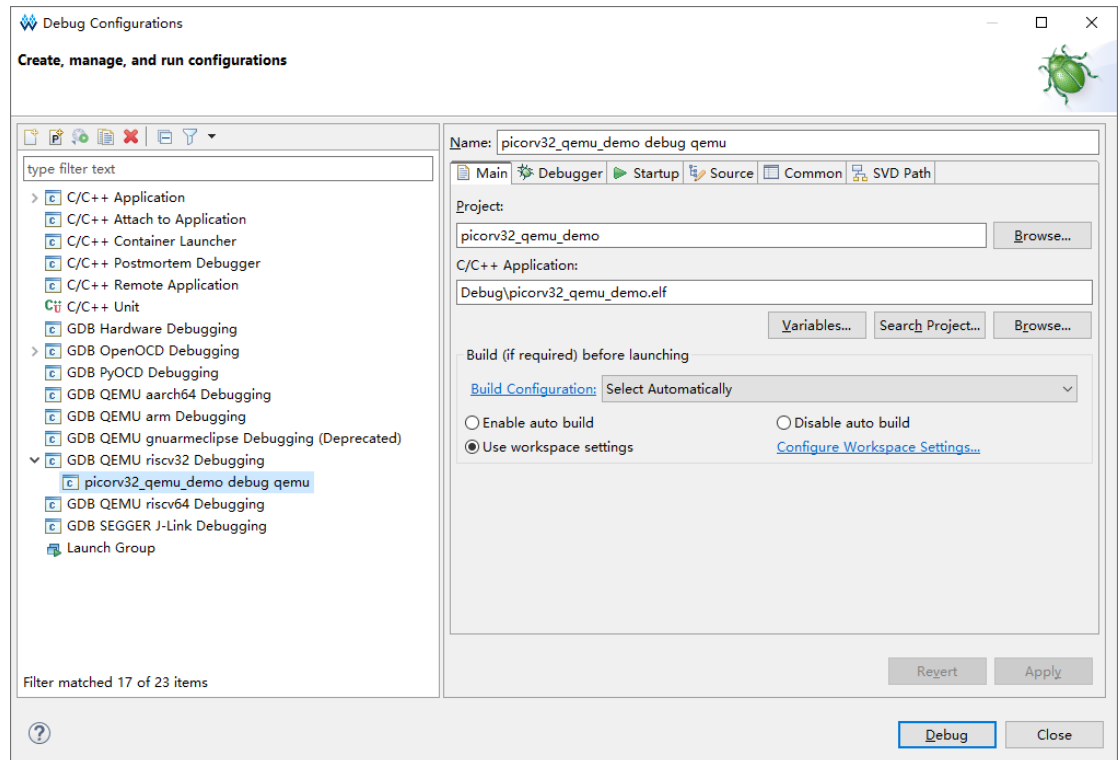
オプション	設定
Initial Reset.	×
Load symbols	√
Load executable	√
Debug in RAM	×
Pre-run/Restart reset	√
Set program counter at (hex)	-
Set breakpoint at	main

オプション	設定
Continue	√

11.2.2 RISC-V MCU QEMU

例えば、Gowin_PicoRV32 の場合は、GDB QEMU riscv32 Debugging を選択します(図 11-5)。

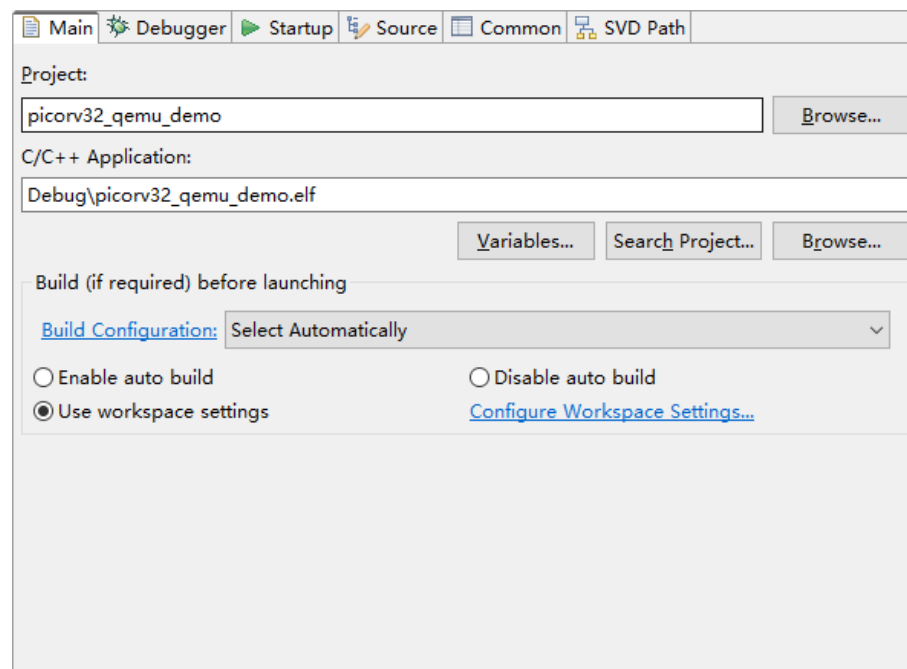
図 11-5 GDB QEMU riscv32 Debugging



1. Main ビュー

Main ビューでの QEMU 設定を図 11-6 に示します。

図 11-6 Main ビュー



2. Debugger ビュー

Debugger ビューでの QEMU 設定を図 11-7 と表 11-5 に示します。

図 11-7 Debugger ビュー

The screenshot shows the 'Debugger' tab in an IDE. It contains two main sections: 'QEMU Setup' and 'GDB Client Setup'.

QEMU Setup:

- ☒ Start QEMU locally
- Executable path: `${qemu_riscv32_path}/${qemu_riscv32_executable}` (with a 'Browse...' button)
- ☐ Prefer the xpacks/.bin path
- Actual executable: `E:\GMD\toolchain\gowin\qemu_riscv\bin\qemu-system-riscv32.exe` (with a note to change it via global/workspace preferences or project properties)
- QEMU machine name: `gowin_pico`
- QEMU CPU name: `gowin-picorv32`
- GDB port: `1234`
- More options: `-serial stdio -nodefaults -S`
- ☒ Enable Arm semihosting
- ☒ Do not open graphic windows
- Command line: `picorv32_qemu_demo`
- ☒ Allocate console for QEMU
- Delay (seconds): `0`

GDB Client Setup:

- Executable name: `${cross_prefix}gdb${cross_suffix}` (with a 'Browse...' button)
- Actual executable: `riscv-none-elf-gdb`
- Other options: (empty field)
- Commands: `set mem inaccessible-by-default off`

Remote Target:

- ☐ Force thread list update on suspend
- Host name or IP address: `localhost`
- Port number: `1234`

表 11-5 Debugger ビュー

オプション	設定
Start QEMU locally	√
QEMU machine name	表 11-6 に示す通りです。
QEMU CPU name	表 11-6 に示す通りです。
GDB port	1234
More options	-serial stdio -nodefaults -S
Enable Arm semihosting	√
Do not open graphic windows	√
Allocate console for QEMU	√
Delay (second)	0

オプション	設定
Commands	set mem inaccessible-by-default off

表 11-6 RISC-V MCU QEMU

RISC-V MCU	Machine name	CPU name
Gowin_PicoRV32	gowin_pico	gowin-picorv32

3. Startup ビュー

Startup ビューでの QEMU 設定を図 11-8 と表 11-7 に示します。

図 11-8 Startup ビュー

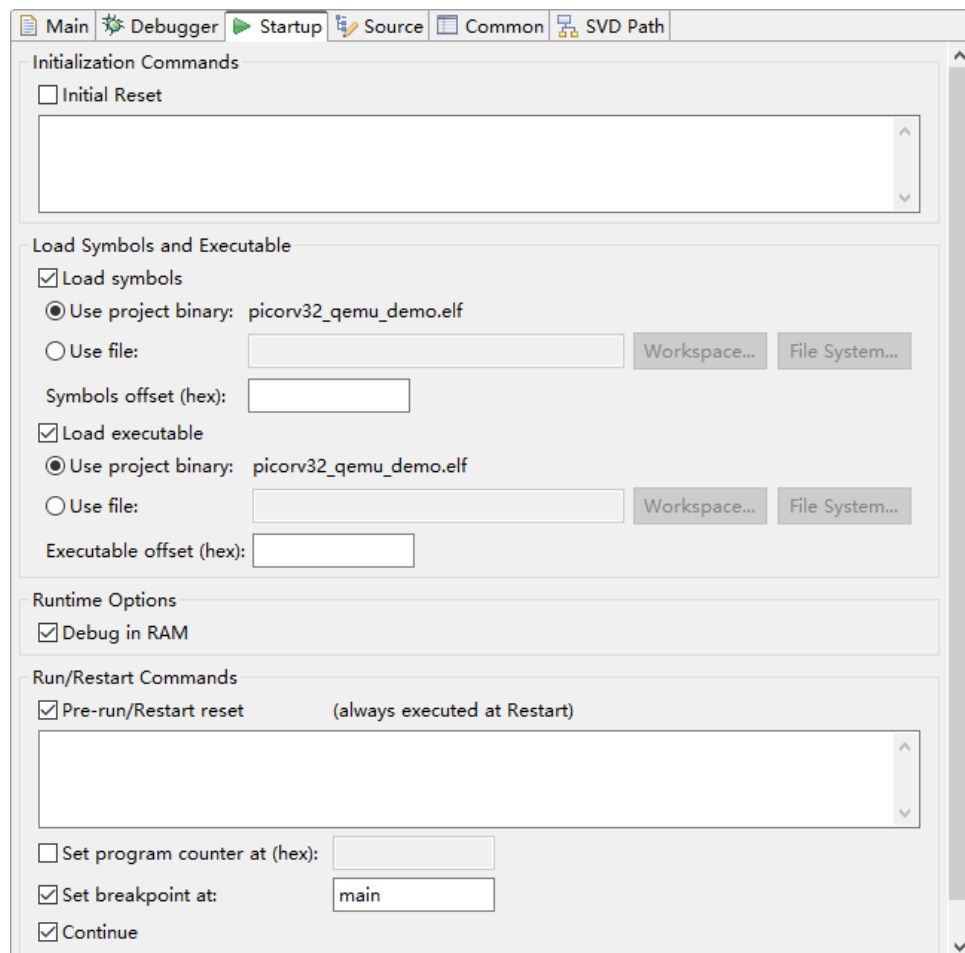


表 11-7 Startup ビュー

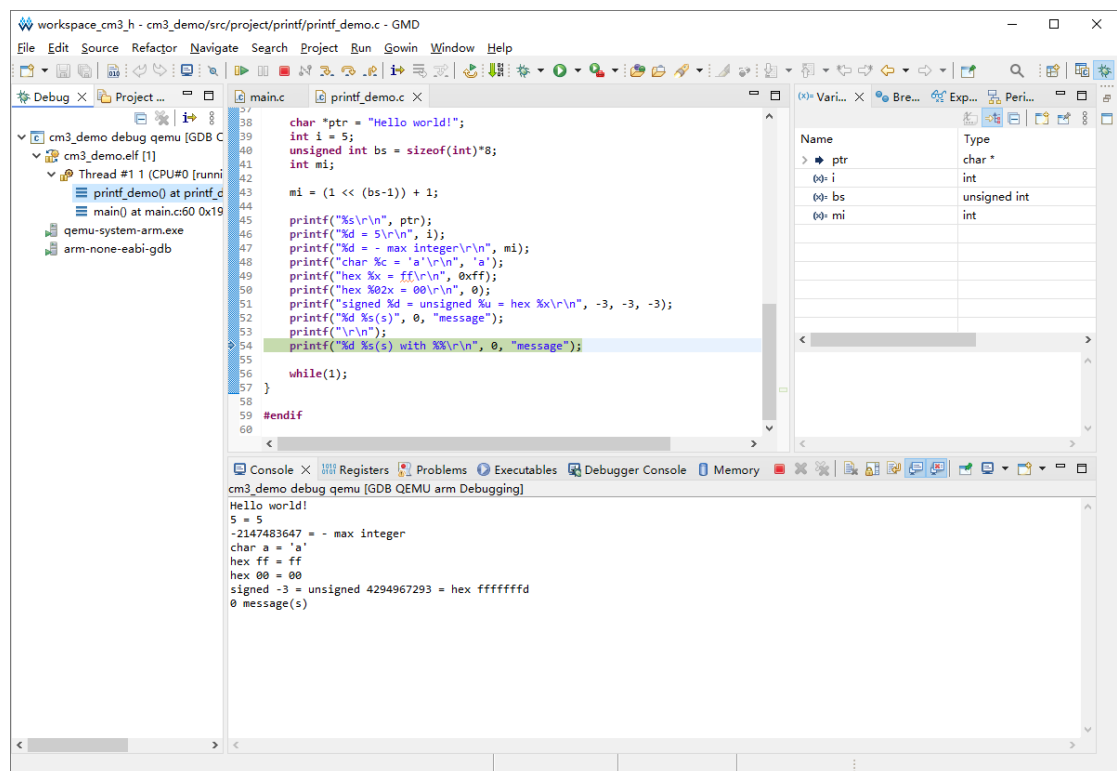
オプション	設定
Initial Reset	×
Load symbols	√

オプション	設定
Load executable	√
Debug in RAM	√
Pre-run/Restart reset	√
Set program counter at (hex)	-
Set breakpoint at	main
Continue	√

11.3 プロジェクトのデバッグ

例えば、Gowin_EMPU(GW1NS-4C)の場合、QEMU の設定後、「Debug」をクリックしてデバッグ・ビューを起動し、プロジェクトを実行します(図 11-9)。

図 11-9 QEMU によるエミュレーション・デバッグの開始



12 Code Coverage 機能

12.1 Code Coverage 機能について

GMD ソフトウェアは **Code Coverage** 機能をサポートしています。**Code Coverage** 機能は、GCC コンパイラが提供する **Gcov** ツールを使用して、指定されたソースコード・ファイルのコード・カバレッジをチェックすることです。これにより、開発者はテスト・ケースが重複していないか、テスト対象コードのすべての分岐とパスがカバーされているかを確認できます。

Gcov ツールは、C/C++のコード・カバレッジをテストするためのツールです。GCC と併用することによって、C/C++ファイルのステートメント、関数、および分岐のカバレッジ・テストを実現できます。

GMD ソフトウェアでは、指定されたソースコード・ファイルを特定の **-coverage** コンパイル・オプションでコンパイルすることで、実際の開発ボード上で実行し、**semihost** 機能を使用して必要なカバレッジ・ファイル (**gcda/gcno** ファイル) を収集するか、**QEMU** 上で直接エミュレートして実行することができます。**Gcov** ツールと併用することにより、グラフィカルな方法で表示できます。

- **.gcno** ファイルは、GCC コンパイラの **-ftest-coverage** オプションを使用してソースコードをコンパイルする際に生成されるもので、基本ブロックの構造を再構築するための情報や、各ブロックに対応するソースコードの行番号に関する情報が含まれています。
- **.gcda** ファイルは、GCC コンパイラの **-fprofile-arcs** オプションを使用してコンパイルされたターゲット・ファイルを実行する際に生成されるものです。このオプションを使用してコンパイルされた各ターゲット・ファイルは、弧の遷移回数、値の分布数、およびいくつかの要約情報を含む個別の **.gcda** ファイルを生成します。

通常、**-coverage** オプションを直接使用して、コンパイラに上記のファイルを生成するよう指示できます。また、**.gcda** ファイルは実行時に生成されることに注意してください。

Code Coverage 機能を使用する場合は、カバレッジ情報が可能な限り正確になるように、**-O0** 最適化オプションを使用してコンパイルすることをお勧めします。

12.2 Code Coverage 機能の使用

例えば、QEMU プラットフォーム上で Gowin_EMPU (GW1NS-4C) をエミュレート方式で実行することができます。

12.2.1 プロジェクトの作成

Code Coverage 機能用のソフトウェア・テスト・プロジェクトを作成します。従来のソフトウェア・プロジェクトと比較して、このプロジェクトにはいくつかの特別なコードセグメントが必要です。

- **gcov.c** ファイルを追加します。
- **gcov_init** 関数を追加して呼び出し、**Gcov** 機能を初期化します。
- **gcov_collect** 関数を呼び出して、関連する **Profiling** 情報を収集します。
- Code Coverage 機能のために **startup.S** アセンブリ・ファイルと **flash.ld** リンク・ファイルを変更します。

具体的な実装については、Gowin_EMPU (GW1NS-4C) の関連リファレンス・デザイン (**cm3_gcov_demo** など) を参照してください。

12.2.2 関連オプションの設定

コンパイル・オプションの設定

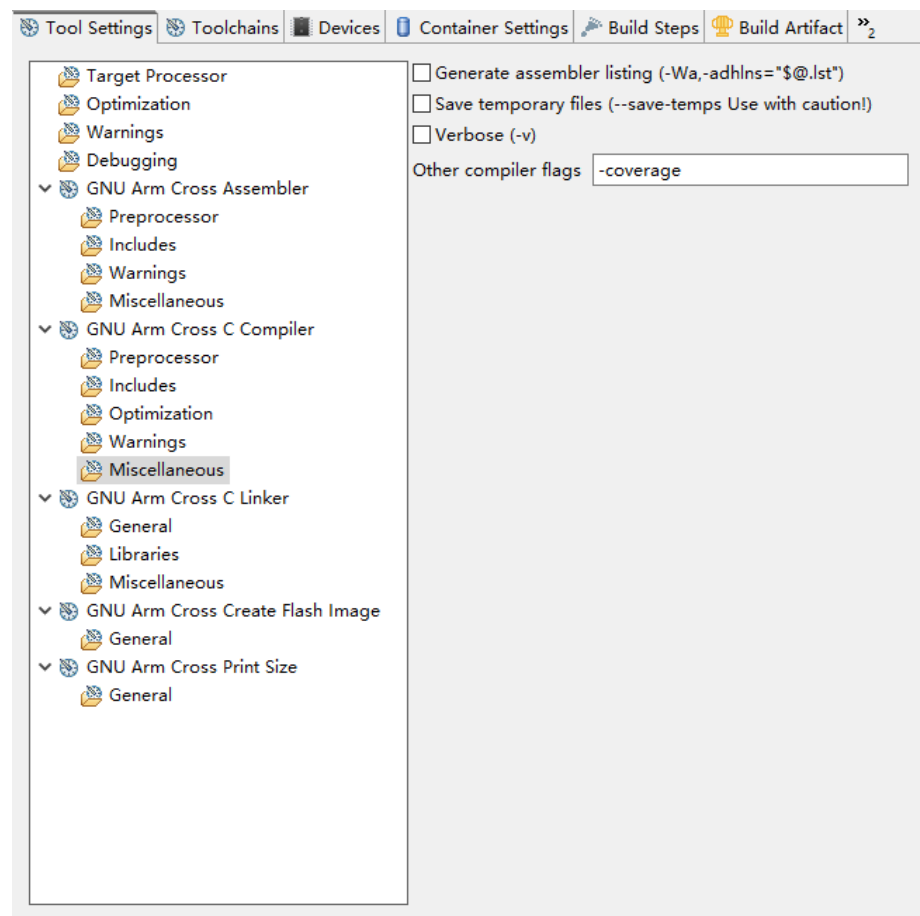
プロジェクトを作成後、コードを解析するファイルまたはフォルダに追加のコンパイル・オプション **-coverage** を設定し、プロジェクトをコンパイルします。

- プロジェクト内のすべてのソースコード・ファイルを指定することができます。
- 特定のソースコード・ファイルを指定することもできます。

1. すべてのソースコード・ファイルを指定

現在のプロジェクトを右クリックして「**Properties > C/C++ Build > Settings > Tool Settings > GNU Arm Cross C Compiler > Miscellaneous > Other compiler flags**」を選択し、「**-coverage**」を追加します(図 12-1)。

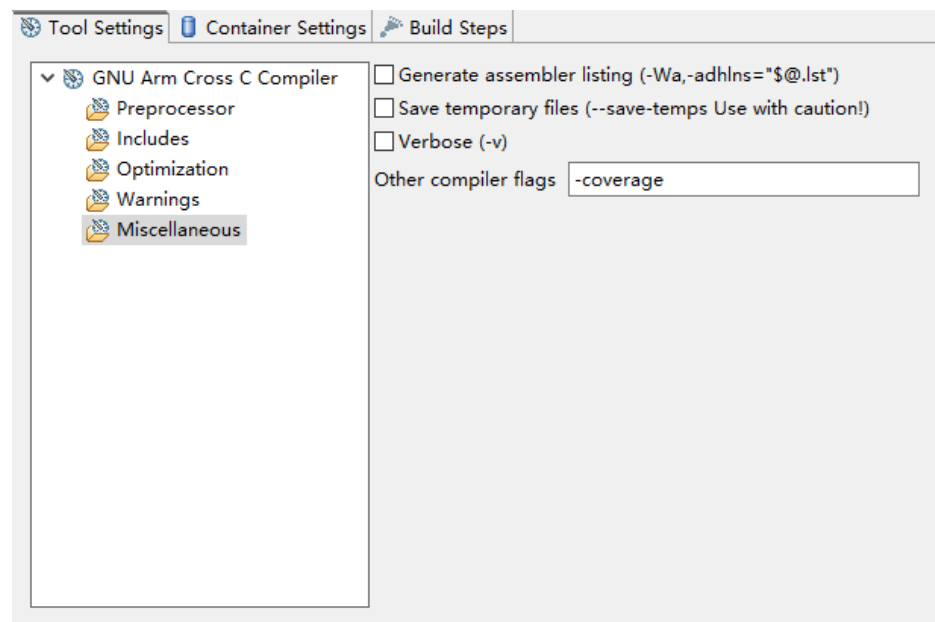
図 12-1 すべてのソースコード・ファイルの指定



2. 特定のソースコード・ファイルを指定

特定のファイルを右クリックして「**Properties > C/C++ Build > Settings > GNU Arm Cross C Compiler > Other compiler flags**」を選択し、「**-coverage**」を追加します(図 12-2)。

図 12-2 特定のソースコード・ファイルの指定



Linux Tools Path の設定

Code Coverage 機能を使用する際は、Parse HexDump Data 機能を呼び出す必要があります。ユーザー環境が異なるため、まず「Linux Tools Path」でコンパイル・ツールチェーンを指定することをお勧めします。

Windows の場合は、

「`${eclipse_home}\toolchain\gowin\gcc_arm\bin`」や「`arm-none-eabi-`」などのツールチェーン・パスとプレフィックス情報を手動で指定します(図 12-3)。

Linux の場合は、システムのデフォルトの環境パス（「Use the System Environment PATH」）を選択します(図 12-4)。

現在のプロジェクトを右クリックして「Properties > Linux Tools Path」を選択し、コンパイルツール・チェーンとそのパスを指定します。

図 12-3 Linux Tools Path の設定

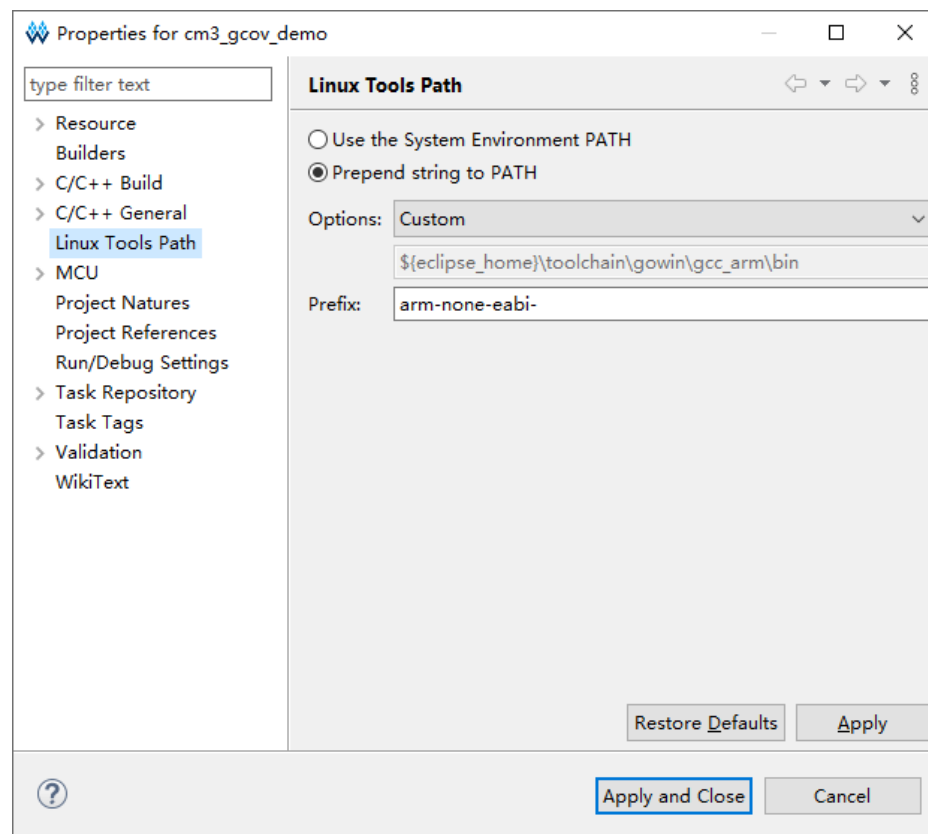
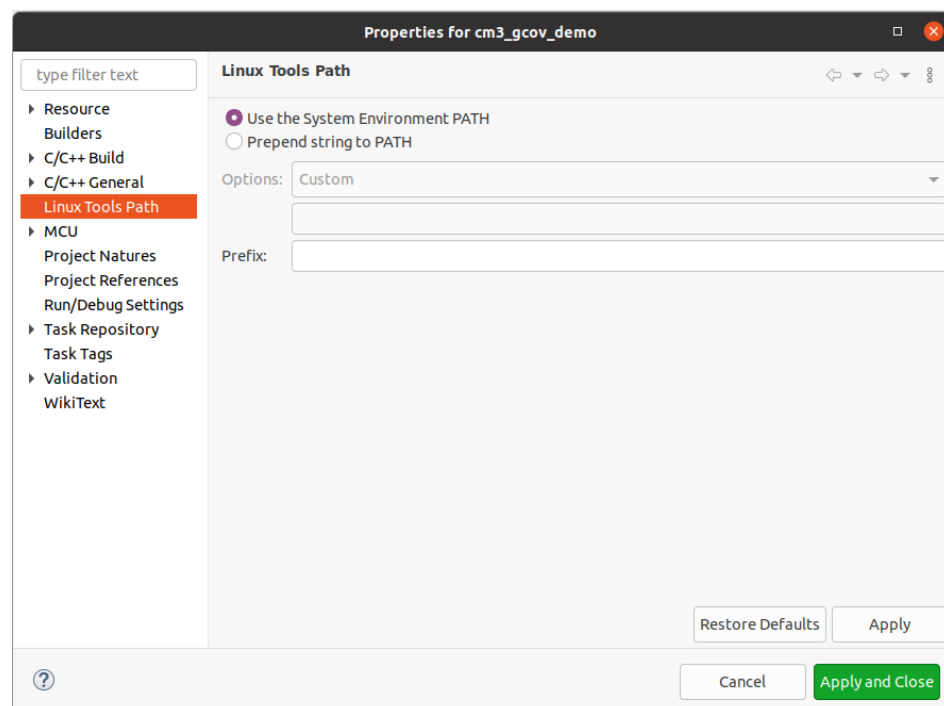


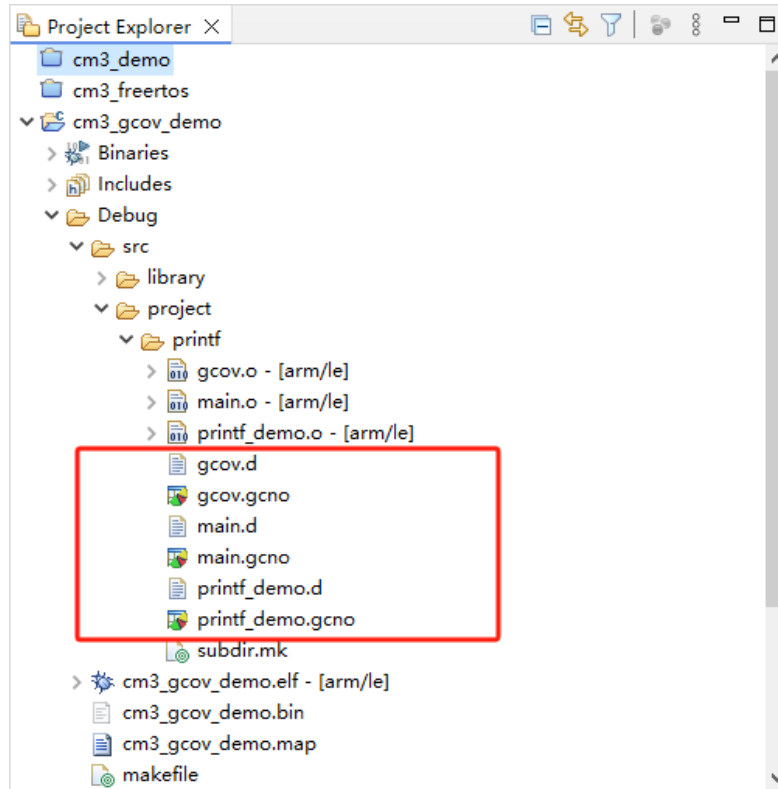
図 12-4 Linux Tools Path の設定



12.2.3 プロジェクトのコンパイル

コンパイルされたプロジェクトの **Debug** フォルダには、複数の **.gcno** ファイルが生成されています(図 12-5)。

図 12-5 プロジェクトのコンパイル



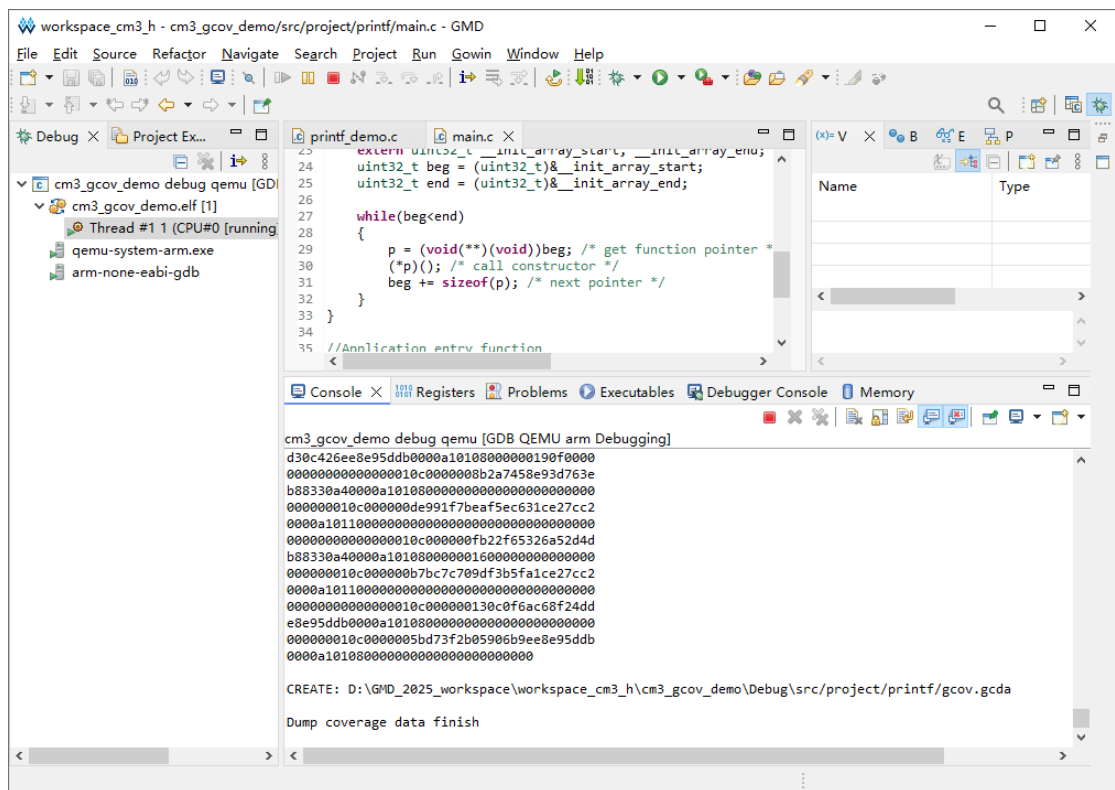
12.2.4 プロジェクトの実行および Code Coverage 機能の使用

プロジェクトをコンパイル後、プロジェクトを実行またはデバッグできます。**QEMU** でエミュレート方式で実行するか、実際の開発ボードでデバッグすることができます。例えば、**QEMU** でエミュレート方式で実行する場合、**GMD** ソフトウェアのコンソールに **Profiling** 情報が表示されます。開発ボードでデバッグする場合、シリアルポート経由で **Profiling** 情報が出力されます。

プロジェクトを実行して **Code Coverage** 機能を使用するには、以下の手順に従ってください。

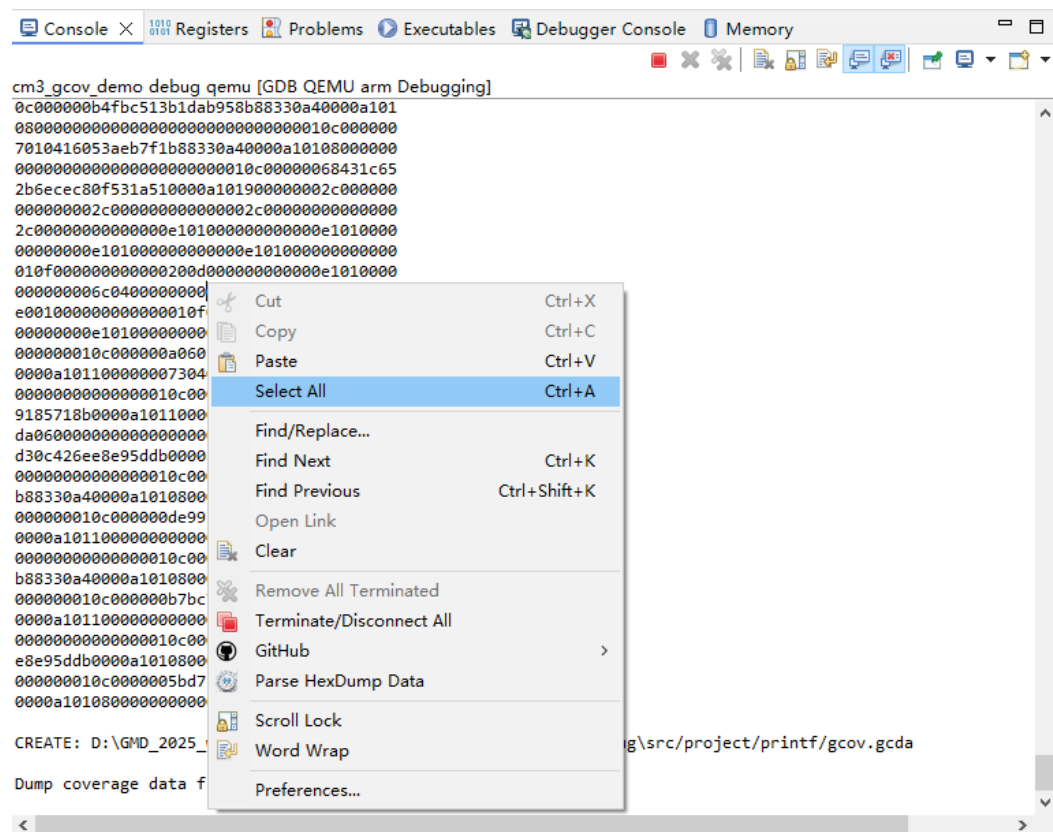
1. 「**Debug As > Debug Configurations...** > **GDB QEMU arm Debugging**」を選択し、**QEMU** でデバッグを開始します。プロジェクトの実行中に、コンソールに **Profiling** 情報が表示されます(図 12-6)。

図 12-6 Profiling 情報の表示



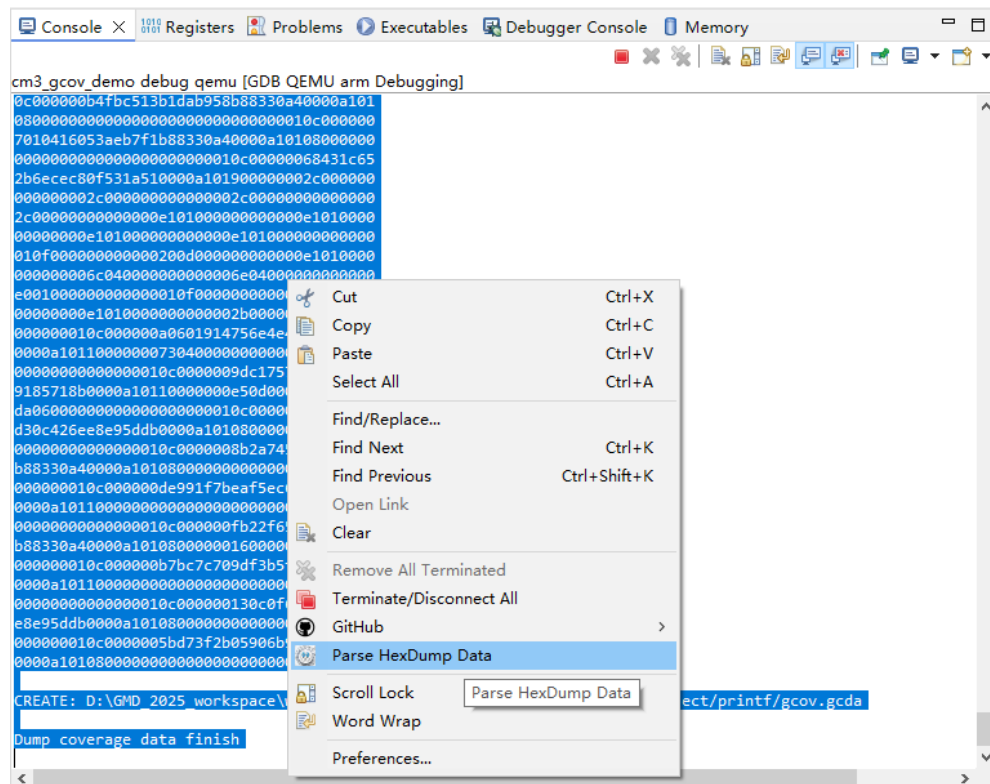
2. 出力された **Profiling** 情報は、**GMD** ソフトウェアが正しく読み取る前に解析する必要があります。コンソール内の任意の場所を右クリックし、ドロップダウン・メニューから「**Select All**」を選択してすべての出力情報を選択します(図 12-7)。

図 12-7 Select All



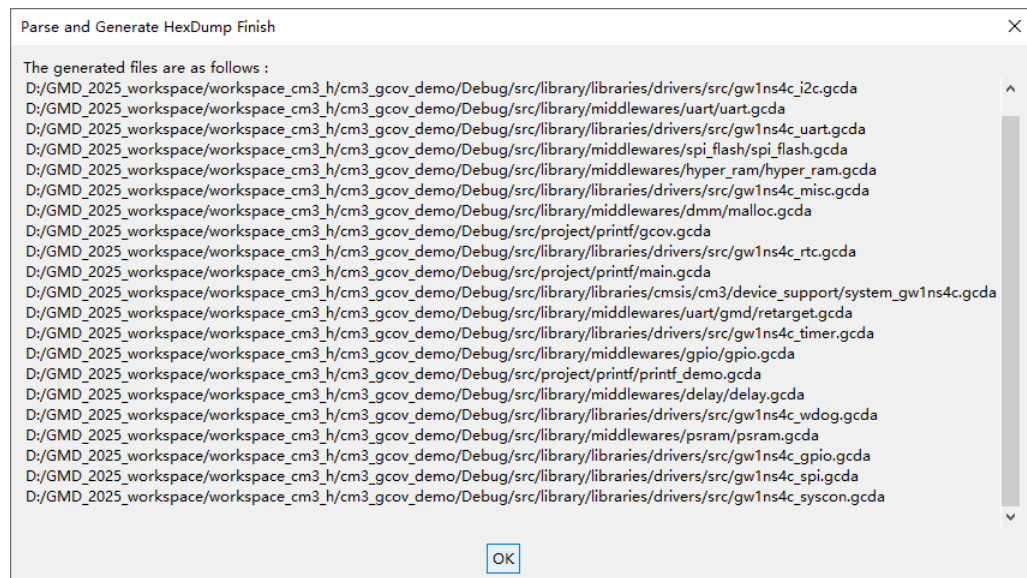
- 次に、コンソール内の任意の場所を再度右クリックし、「Parse HexDump Data」を選択します（図 12-8）。

図 12-8 Parse HexDump Data



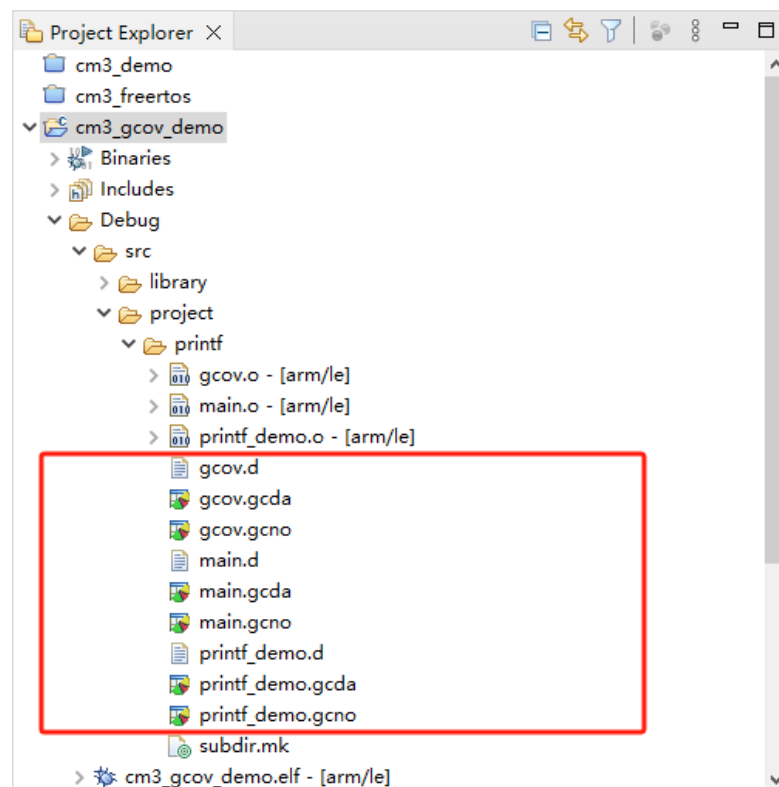
4. こうすると、GMD ソフトウェアは出力ファイルを解析し、結果を対応するファイルに保存します（図 12-9）。

図 12-9 Profiling 情報の解析と対応ファイルの生成



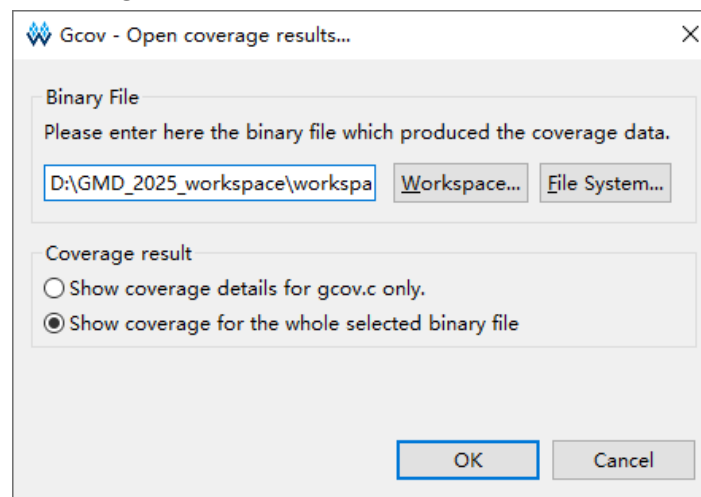
5. プロジェクトの Debug フォルダを再度確認すると、対応する .gcda ファイルが生成されていることがわかります（図 12-10）。

図 12-10 生成される対応ファイルの確認



6. Code Coverage を確認したい.gcda ファイルをダブルクリックして、Gcov ツールを開きます (図 12-11)。

図 12-11 gcda ファイルを開く

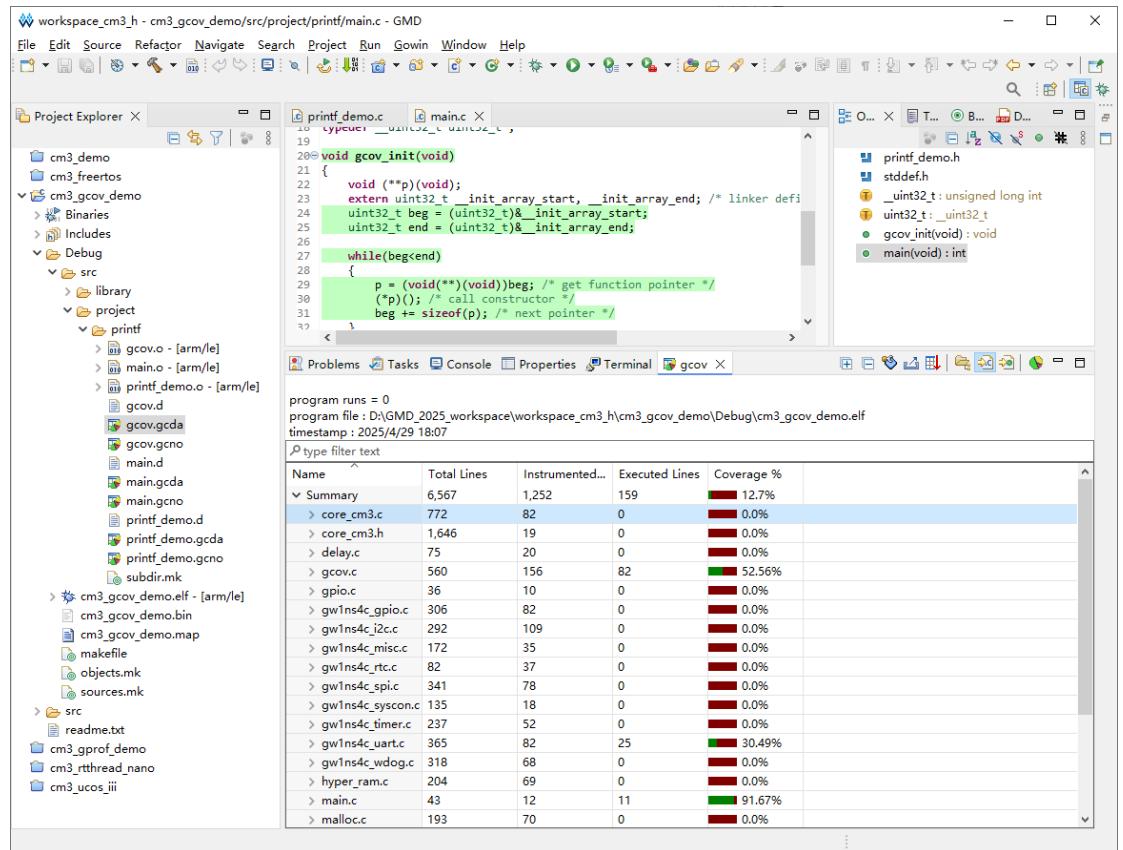


7. こうすると、アプリケーションの解析結果が表示されます。プログラム実行プロセス中に特定のファイルまたはメソッドが実行されたかどうか、コード実行カバレッジ率などのデータが表示されます。Gcov 内の行をダブルクリックすると、GMD ソフトウェアは対応するファ

イルを自動的に開き、ファイル内のコードに色を付けます。緑はプログラム実行プロセス中に実行されたことを意味し、赤はプログラム実行プロセス中に実行されていないことを意味します(図 12-12)。

開発者は **Gcov** の結果に応じてコードを最適化することができます。

図 12-12 Gcov



Gcov ツールの機能を表 12-1 に示します。

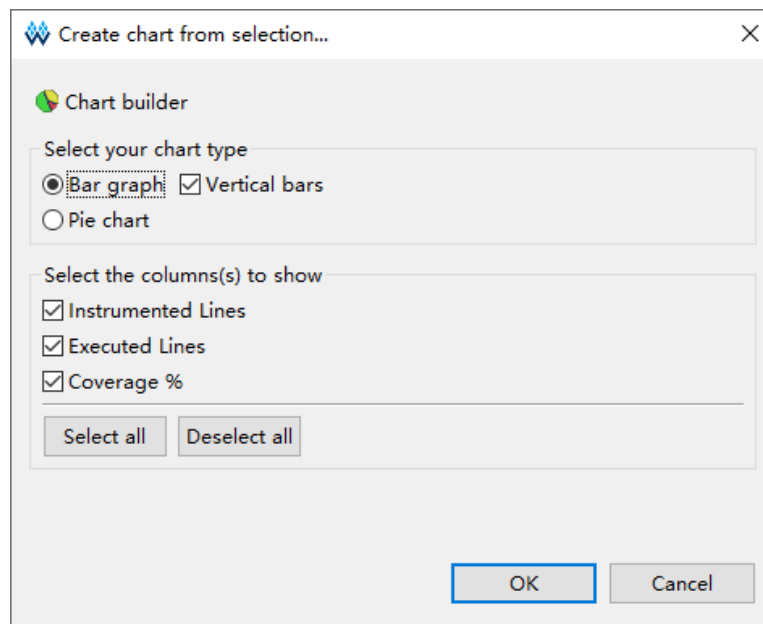
表 12-1 Gcov の機能

オプション	機能の説明
Show/Hide columns ()	表示する列を指定します
Export to CSV ()	解析データを.csv ファイルにエクスポートします
Sorting ()	解析データを並べ替えます
Sort coverage per file ()	解析データをファイル別に並べ替えます
Sort coverage per function ()	解析データを機能別に並べ替えます

オプション	機能の説明
Create chart… ()	指定した解析データを使用して棒グラフまたは円グラフを作成します

8. 棒グラフまたは円グラフを作成することができます。Gcov でグラフを作成するファイルを選択し、「Create chart…」() を選択します (図 12-13)。

図 12-13 Create chart…



作成された棒グラフと円グラフはそれぞれ図 12-14 と図 12-15 に示すとおりです。

図 12-14 棒グラフ

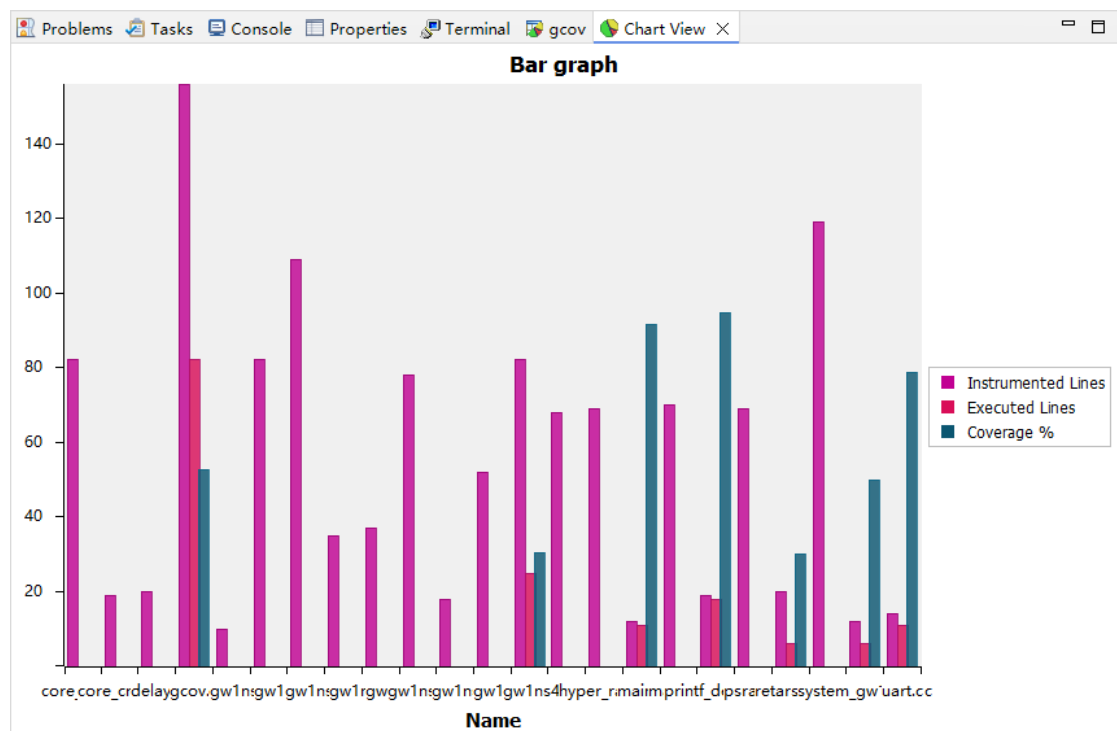
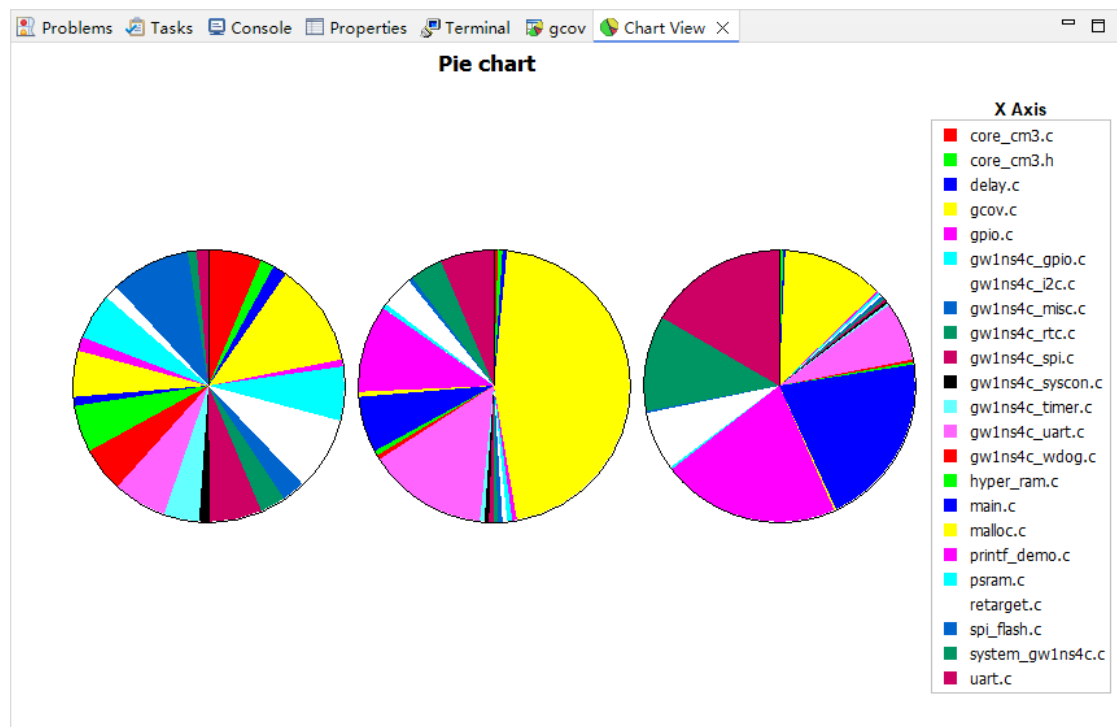


図 12-15 円グラフ



13 Profiling 機能

13.1 Profiling 機能について

GMD ソフトウェアは Profiling 機能をサポートしています。Profiling 機能により、GCC コンパイラの Binutils に含まれる Gprof ツールを使用して、指定されたファイル内の関数の実行時間と呼び出し回数、および呼び出し関係を確認することができます。Gprof は、プログラムのボトルネックを特定してパフォーマンスを最適化するために使用できます。Gprof は、プログラム実行中にデータを収集し、プログラム内の各関数の CPU 時間占有率と関数間の呼び出し関係を示すレポートを生成します。

Gprof ツールは、開発者が C/C++ プログラムの動作を理解するのに役立つパフォーマンス分析ツールです。Gprof を使用することにより、開発者はプログラム内の各関数の呼び出し情報、呼び出し回数、実行時間などを取得できるため、プログラムの最適化と実行効率の向上が容易になります。

GMD ソフトウェアでは、指定されたソースコード・ファイルを特定の `-pg` コンパイル・オプションでコンパイルすることで、実際の開発ボード上で実行し、`semihost` 機能を使用して必要な `gmon.out` ファイルを収集するか、`QEMU` 上で直接エミュレートして実行することができます。Gprof ツールと併用することにより、グラフィカルな方法で表示できます。

生成された `gmon.out` ファイルは、効果的な Profiling 機能を実装するには、コンパイラを利用して実際のボードまたは `QEMU` で実行する必要があり、動作環境がファイルの読み取りをサポートしている必要もあります。

13.2 Profiling 機能の使用

例えば、`QEMU` プラットフォーム上で `Gowin_EMPU (GW1NS-4C)` をエミュレート方式で実行することができます。

13.2.1 プロジェクトの新規作成

Profiling 機能用のソフトウェア・テスト・プロジェクトを作成します。従来のソフトウェア・プロジェクトと比較して、このプロジェクトにはいくつかの特別なコードセグメントが必要です。

- gmon.c、gmon.h、profile.c、profile.h ファイルを追加します。
- gmon 関連関数を呼び出します。
- Profiling 機能のために startup.S アセンブリ・ファイルと flash.ld リンク・ファイルを変更します。

具体的な実装については、Gowin_EMPU (GW1NS-4C)の関連リファレンス・デザイン (cm3_gprof_demo など) を参照してください。

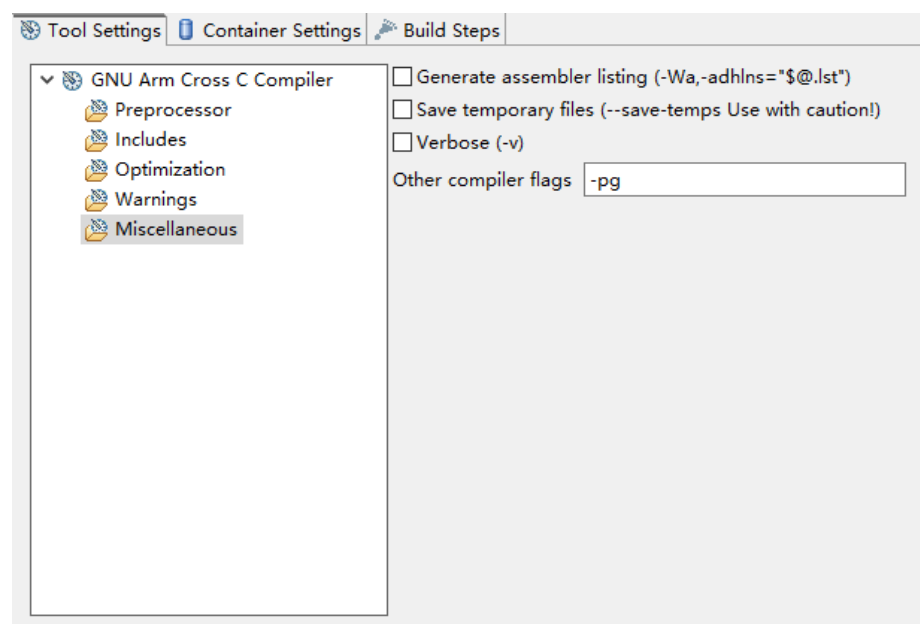
13.2.2 関連オプションの設定

コンパイル・オプションの設定

プロジェクトを作成後、コードを解析するファイルまたはフォルダに追加のコンパイル・オプション-pg を設定し、プロジェクトをコンパイルします。

特定のファイルを右クリックして「Properties > C/C++ Build > Settings > GNU Arm Cross C Compiler > Other compiler flags」を選択し、「-pg」を追加します(図 13-1)。

図 13-1 コンパイル・オプションの設定



Linux Tools Path の設定

Profiling 機能を使用する際は、Parse HexDump Data 機能呼び出す必要があります。ユーザー環境が異なるため、まず「Linux Tools Path」でコンパイル・ツールチェーンを指定することをお勧めします。

Windows の場合は、
「`${eclipse_home}\toolchain\gowin\gcc_arm\bin`」や「`arm-none-eabi-`」などのツールチェーン・パスとプレフィックス情報を手動で指定します(図 13-2)。

Linux の場合は、システムのデフォルトの環境パス（「Use the System Environment PATH」）を選択します(図 13-3)。

現在のプロジェクトを右クリックして「Properties > Linux Tools Path」を選択し、コンパイルツール・チェーンとそのパスを指定します。

図 13-2 Linux Tools Path の設定

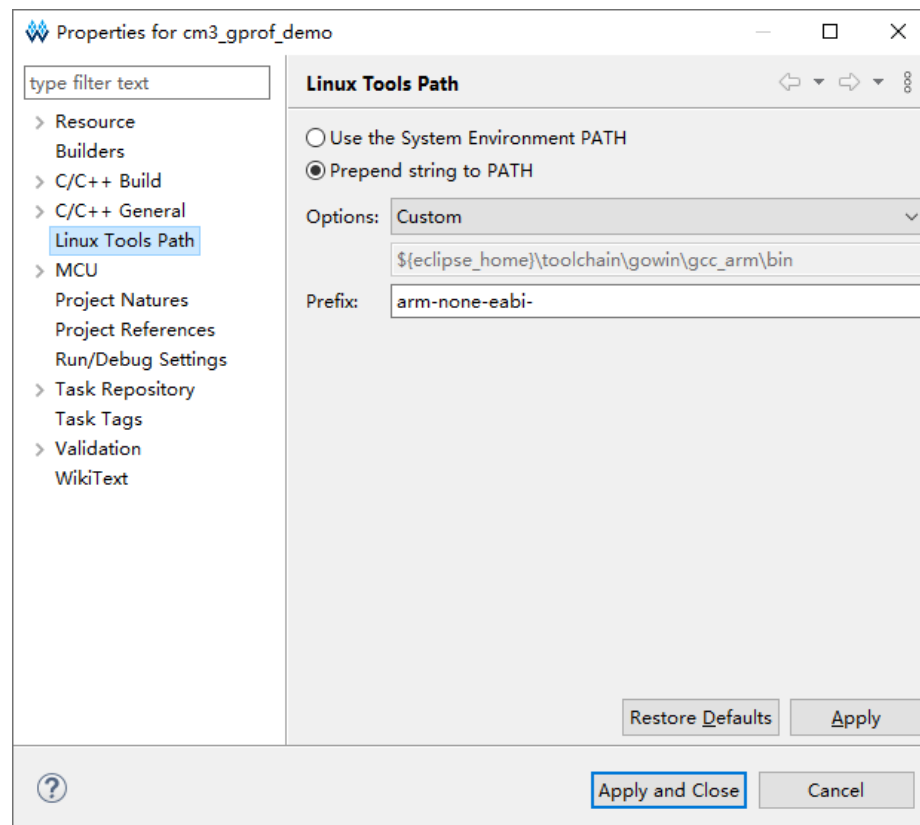
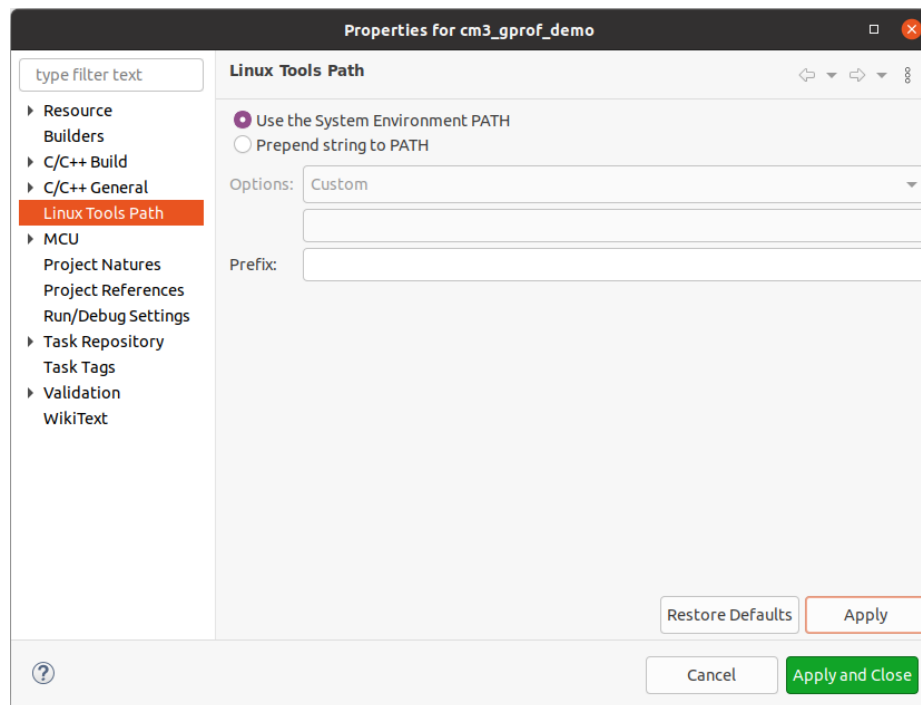


図 13-3 Linux Tools Path の設定



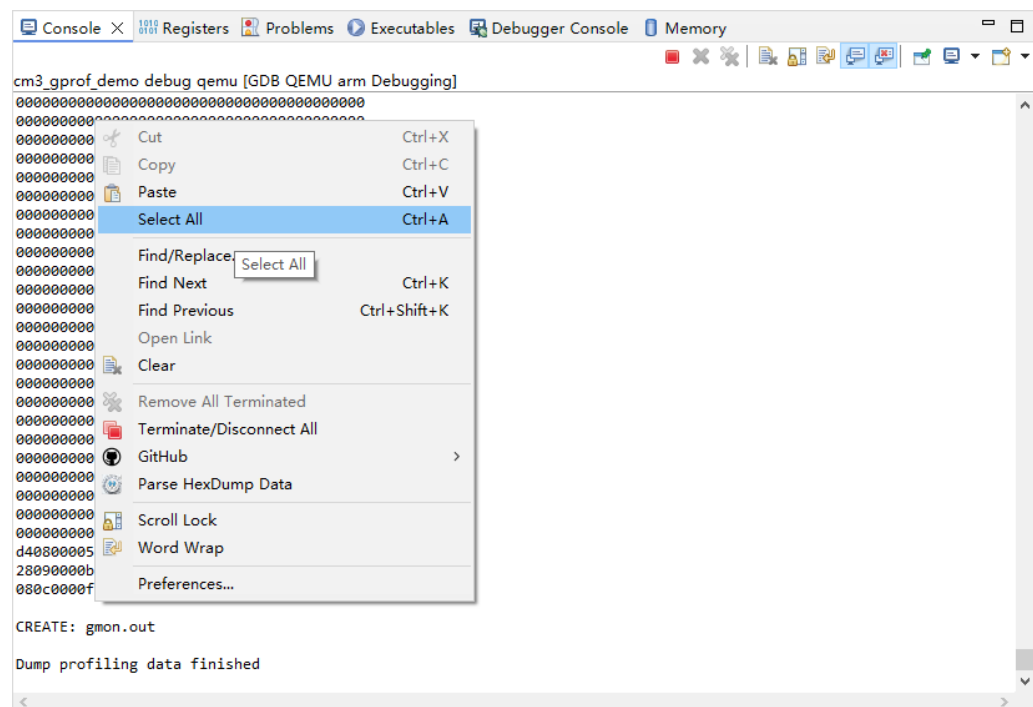
13.2.3 プロジェクトの実行および Profiling 機能の使用

プロジェクトをコンパイル後、プロジェクトを実行またはデバッグできます。**QEMU** でエミュレート方式で実行するか、実際の開発ボードでデバッグすることができます。例えば、**QEMU** でエミュレート方式で実行する場合、**GMD** ソフトウェアのコンソールに **Profiling** 情報が表示されます。開発ボードでデバッグする場合、シリアルポート経由で **Profiling** 情報が出力されます。

プロジェクトを実行して **Profiling** 機能を使用するには、以下の手順に従ってください。

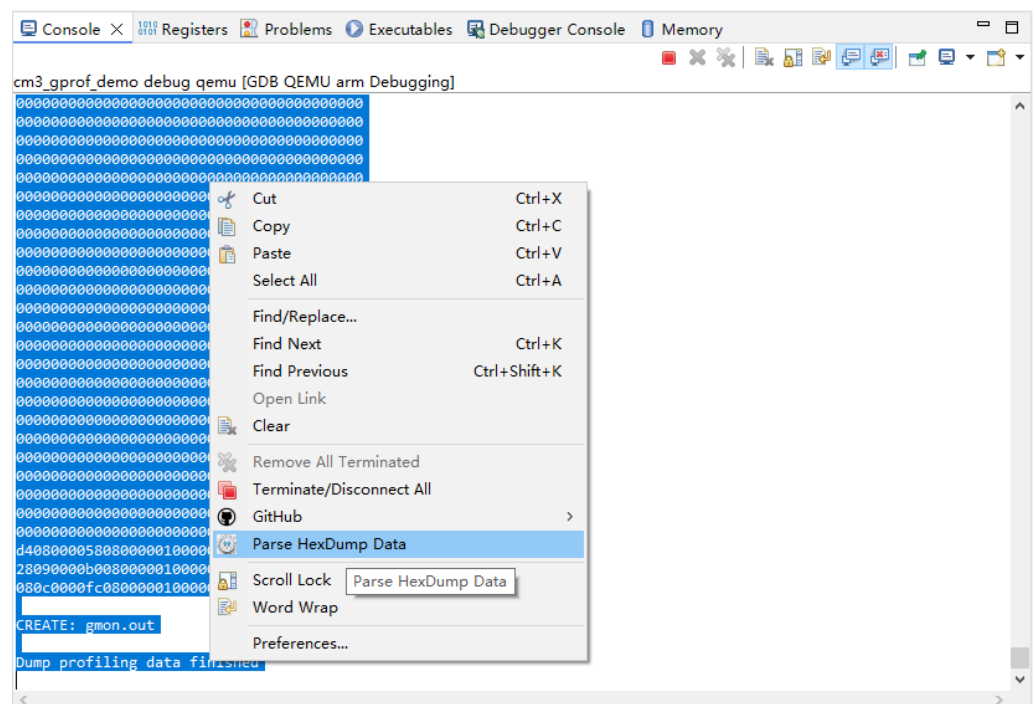
1. 「Debug As > Debug Configurations... > GDB QEMU arm Debugging」を選択し、**QEMU** でデバッグを開始します。プロジェクトの実行中に、コンソールに **Profiling** 情報が表示されます(図 13-4)。

図 13-5 Select All



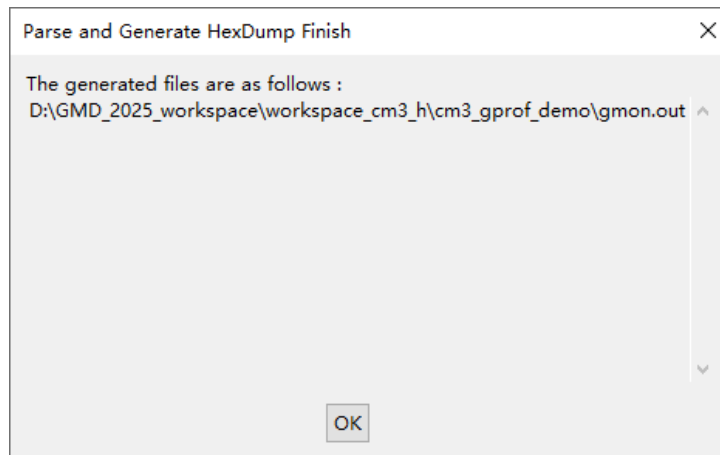
- 次に、コンソール内の任意の場所を再度右クリックし、「Parse HexDump Data」を選択します（図 13-6）。

図 13-6 Parse HexDump Data



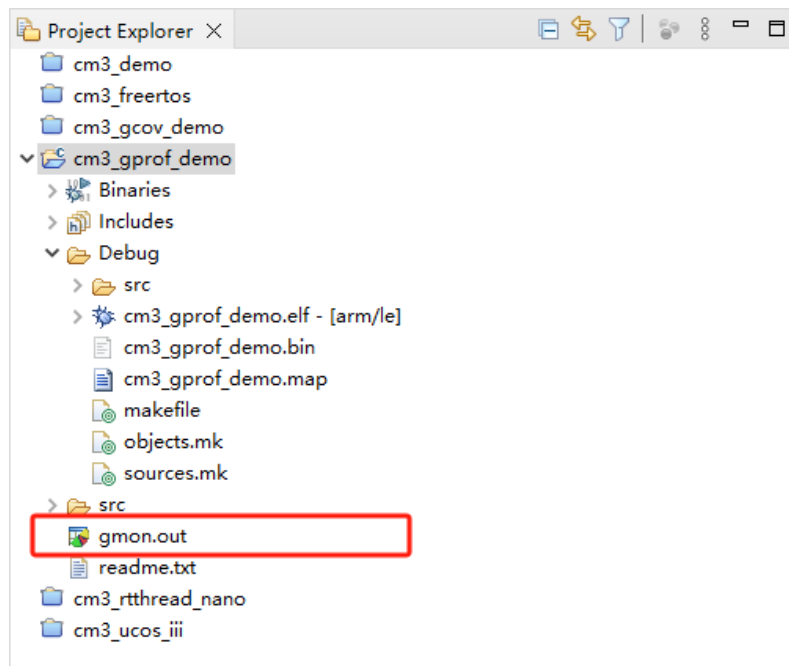
4. こうすると、GMD ソフトウェアは出力ファイルを解析し、結果を対応するファイルに保存します (図 13-7)。

図 13-7 Profiling 情報の解析と対応ファイルの生成



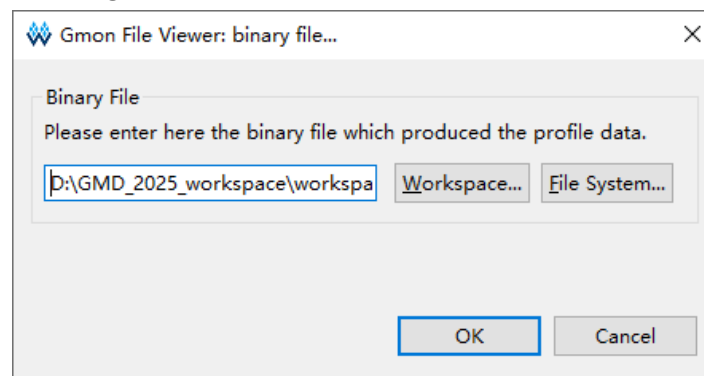
5. プロジェクトの **Debug** フォルダを確認すると、対応する **gmon.out** ファイルが生成されていることがわかります (図 13-8)。

図 13-8 生成される **gmon.out** ファイルの確認



6. **gmon.out** ファイルをダブルクリックして、**Gcov** ツールを開きます (図 13-9)。

図 13-9 gmon.out ファイルを開く



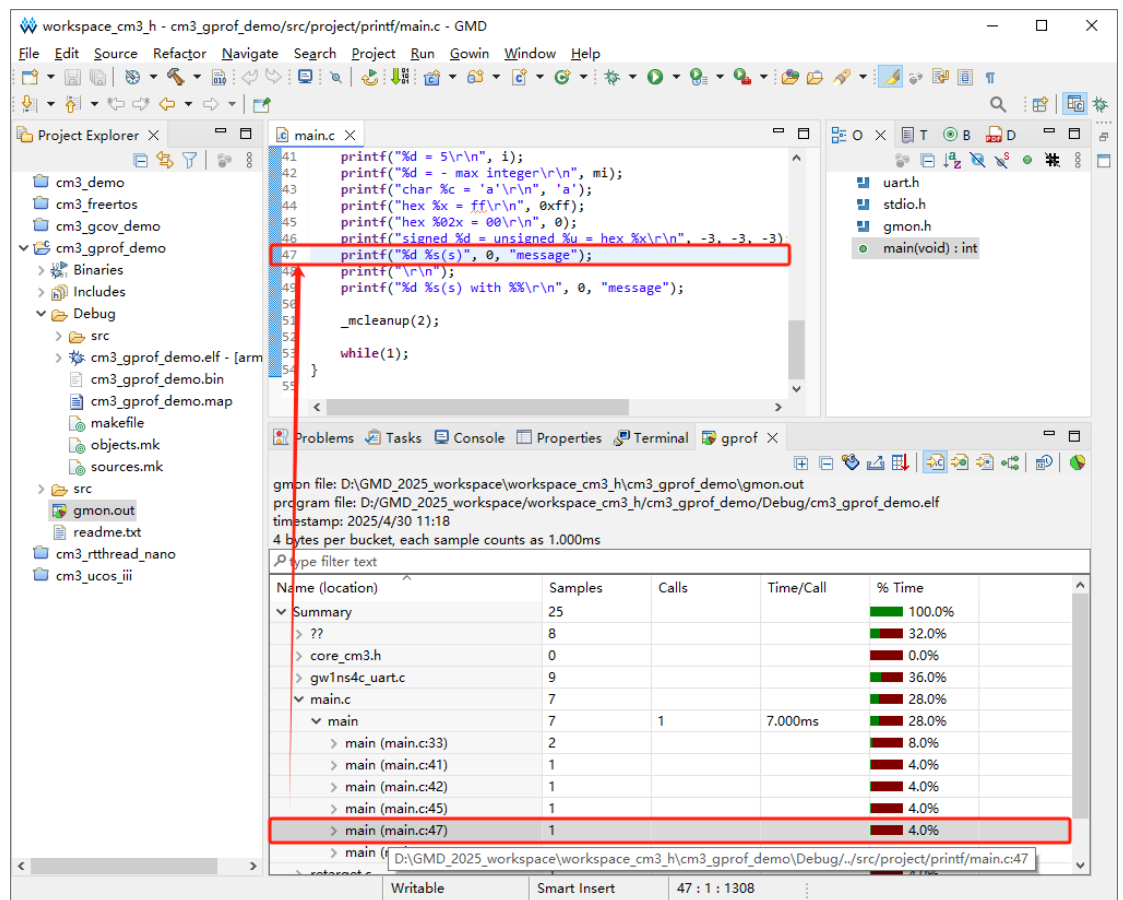
7. Gprof ツールを起動すると、アプリケーションの解析結果が表示され、ファイルとメソッドの呼び出し関係が表示されます(図 13-10)。

図 13-10 Gprof

Name (location)	Samples	Calls	Time/Call	% Time
Summary	25			100.0%
> ??	8			32.0%
> core_cm3.h	0			0.0%
> gw1ns4c_uart.c	9			36.0%
main.c	7			28.0%
main	7	1	7.000ms	28.0%
main (main.c:33)	2			8.0%
main (main.c:41)	1			4.0%
main (main.c:42)	1			4.0%
main (main.c:45)	1			4.0%
main (main.c:47)	1			4.0%
main (main.c:49)	1			4.0%
retarget.c	1			4.0%
startup_gw1ns4c.S	0			0.0%

8. Gprof 内の行をダブルクリックすると、GMD ソフトウェアは自動的に対応するソースファイルを開き、対応する行を特定します (図 13-11)。

図 13-11 Profiling 情報の表示



Gprof ツールの機能を表 13-1 に示します。

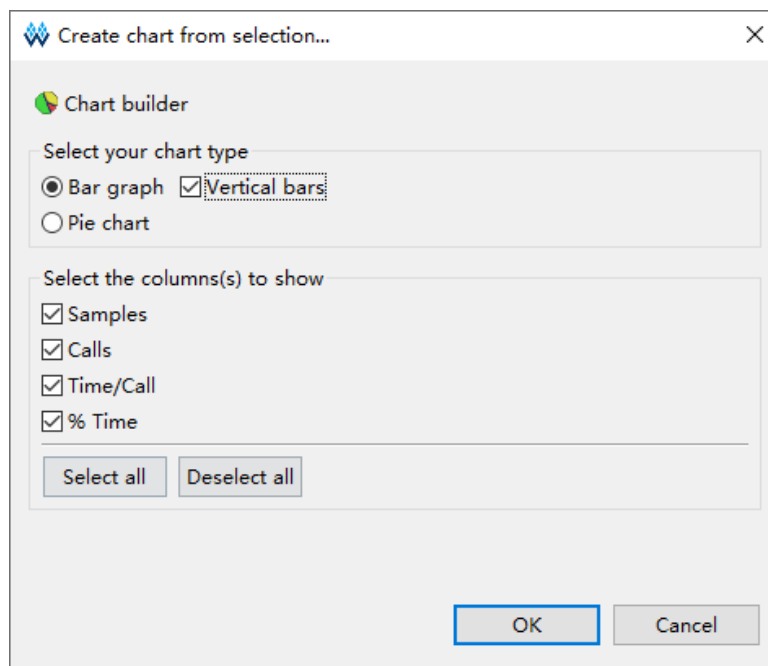
表 13-1 Gprof の機能

オプション	機能の説明
Show/Hide columns ()	表示する列を指定します
Export to CSV ()	解析データを.csv ファイルにエクスポートします
Sorting ()	解析データを並べ替えます
Sort samples per file ()	解析データをファイル別に並べ替えます
Sort samples per function ()	解析データを機能別に並べ替えます
Sort samples per line ()	解析データを行別に並べ替えます

オプション	機能の説明
Display function call graph ()	関数間の呼び出しグラフを表示します
Switch sample/time ()	サンプル数/時間パーセンテージビューを切り替えます
Create chart... ()	指定した解析データを使用して棒グラフまたは円グラフを作成します

9. 棒グラフまたは円グラフを作成することができます。Gprof でグラフを作成するファイルを選択し、「Create chart...」() を選択します (図 13-12)。

図 13-12 Create chart...



例えば、作成された棒グラフと円グラフはそれぞれ図 13-13 と図 13-14 に示すとおりです。

図 13-13 棒グラフ

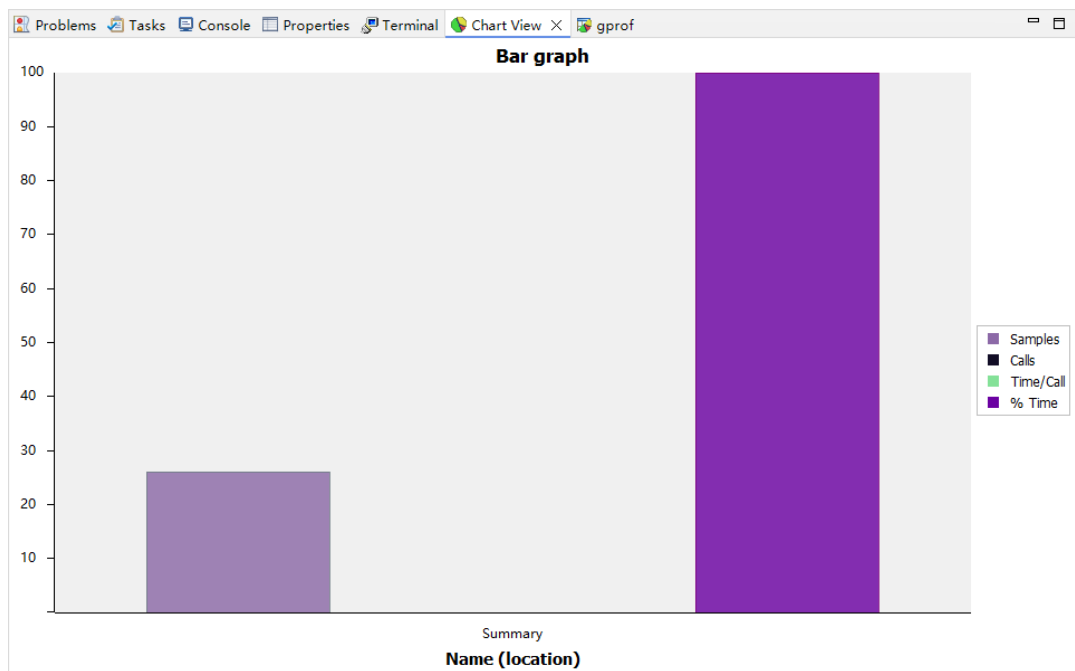


図 13-14 円グラフ

