



Gowin_EMPU_M1 IDE リファレンスマニュアル

IPUG536-2.4J, 2025-08-08

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** 及び **Gowin** 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
2019/02/18	1.0J	初版。
2019/07/18	1.1J	● MCU ハードウェア設計とソフトウェア設計では、周辺機器 CAN、Ethernet、SPI-Flash、RTC、TRNG、DualTimer、I2C、SPI、および SD-Card の拡張をサポート。 ● MCU はオフチップ SPI-Flash によるダウンロードと起動をサポート。
2019/08/18	1.2J	● MCU ハードウェア設計とソフトウェア設計では、DDR3 Memory の拡張をサポート。 ● 既知の ITCM、DTCM Size、および IDE の問題を修正。
2019/09/27	1.3J	MCU プログラミングソフトウェアの GOWIN MCU Designer のインターフェースと機能を更新および最適化。
2020/01/16	1.4J	● MCU ハードウェア設計とソフトウェア設計では、周辺機器 PSRAM の拡張をサポート。 ● MCU コンパイラ GMD V1.0 を更新。 ● RTOS リファレンス・デザインを更新。 ● AHB2 および APB2 拡張バスインターフェースハードウェアおよびソフトウェアのリファレンス・デザインを追加。
2020/03/05	1.5J	デバイス GW2A-18C/GW2AR-18C/GW2A-55C のサポートを追加。
2020/06/12	1.6J	● MCU が外部命令メモリをサポート。 ● MCU が外部データメモリをサポート。 ● AHB バスインターフェースを 6 個拡張。 ● APB バスインターフェースを 16 個拡張。 ● GPIO が複数のインターフェースタイプをサポート。 ● I2C が複数のインターフェースタイプをサポート。
2021/07/16	1.7J	MCU ソフトウェアのバージョン情報を更新。
2021/10/12	1.8J	GW2AN-9X/GW2AN-18X の、ITCM および DTCM Size の範囲を変更。
2023/05/11	1.9J	Arora V FPGA 製品のサポートを追加。
2023/07/21	2.0J	ソフトウェアのバージョン情報を更新。
2024/03/07	2.1J	● GW5AT-60(A バージョン)のサポートを追加。 ● リファレンス・デザインを更新。
2024/07/12	2.2J	● A バージョンの GW5ART-15 のサポートを追加。 ● リファレンス・デザインを更新。
2025/01/17	2.3J	オンラインデバッグの注意事項を追加。
2025/07/18	2.3.1J	一部のリンクを更新。
2025/08/08	2.4J	● GMD ソフトウェアのバージョン情報を更新。 ● GMD の 2025.01 バージョンに適合するようにソフトウェア・リファレンス・デザインを更新。 ● FAT32、32GB の SD-Card をサポートする SD_Host_v2 と SD_SPI_IF_v1 ソリューションを追加。

目次

目次	i
図一覧	ii
1 ARM Keil	1
1.1 ソフトウェアのインストール	1
1.2 プロジェクトテンプレート	1
1.2.1 プロジェクトの作成	1
1.2.2 プロジェクトの設定	2
1.2.3 プロジェクトのコンパイル	8
1.2.4 プロジェクトのダウンロード	9
1.2.5 デバッグ	9
1.3 リファレンス・デザイン	11
2 GMD ソフトウェア	12
2.1 ソフトウェアのインストールと構成	12
2.2 プロジェクトテンプレート	12
2.2.1 プロジェクトの新規作成	12
2.2.2 プロジェクトの設定	15
2.2.3 プロジェクトのコンパイル	23
2.2.4 プロジェクトのダウンロード	24
2.2.5 デバッグ	24
2.3 リファレンス・デザイン	29

図一覧

図 1-1 プロジェクトの新規作成.....	1
図 1-2 デバイスの設定	2
図 1-3 ROM と RAM の設定.....	4
図 1-4 出力されるファイルの形式の設定	5
図 1-5 ヘッダーファイルのパスの設定.....	5
図 1-6 JTAG デバッグポートの設定	6
図 1-7 SWD デバッグポートの設定.....	7
図 1-8 Flash オプションの設定.....	7
図 1-9 デバッグ初期化ファイルの設定.....	8
図 1-10 プロジェクトのコンパイル	9
図 1-11 デバッグの開始	10
図 2-1 プロジェクト・タイプとツールチェーンの選択.....	13
図 2-2 プラットフォームと構成の選択.....	14
図 2-3 ツールチェーンおよびそのパスの選択	15
図 2-4 Target Processor	16
図 2-5 GNU ARM Cross Assembler > Preprocessor.....	17
図 2-6 GNU ARM Cross C Compiler > Includes	18
図 2-7 GNU ARM Cross C Linker > General.....	19
図 2-8 GNU ARM Cross Create Flash Image	20
図 2-9 GNU ARM Cross Print Size.....	21
図 2-10 Optimization	22
図 2-11 Debugging	23
図 2-12 プロジェクトのコンパイル	24
図 2-13 デバッグレベルの設定	25
図 2-14 デバッグ・ビューの作成.....	26
図 2-15 Main タブの設定.....	26
図 2-16 Debugger タブ	27
図 2-17 Startup タブ.....	28
図 2-18 デバッグ・ビューの起動.....	29

1 ARM Keil

1.1 ソフトウェアのインストール

ARM Keil MDK のホームページ上の [Getting Started with MDK](#) を参照してください。

注記：

ARM Keil MDK(V5.26 以降)をお勧めします。

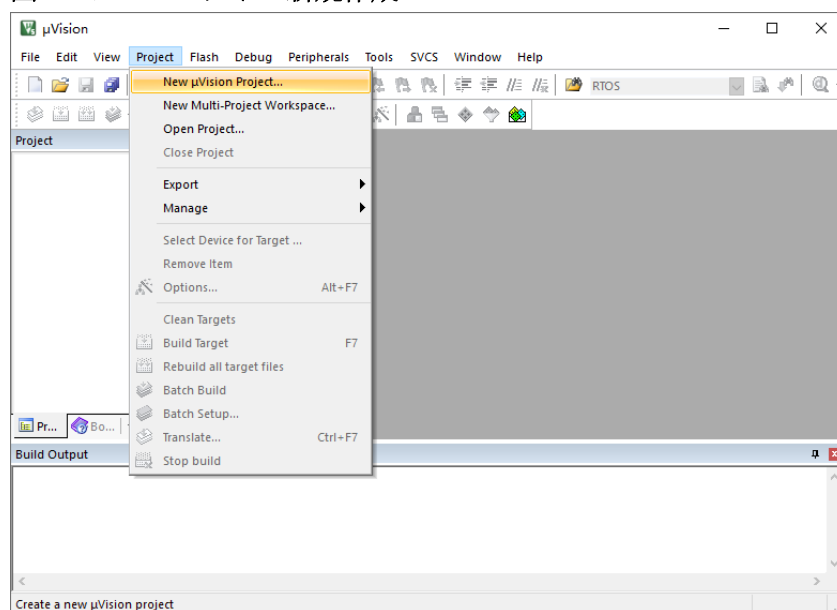
1.2 プロジェクトテンプレート

ARM Keil MDK を利用した Gowin_EMPU_M1 ソフトウェアプログラミングには、プロジェクトの作成、プロジェクトの設定、コーディング、コンパイル、ダウンロード、およびデバッグが含まれます。

1.2.1 プロジェクトの作成

ARM Keil MDK を開いたあと、メニューバーで「Project > New uVision Project…」を選択し、新しいプロジェクトを作成します(図 1-1)。

図 1-1 プロジェクトの新規作成

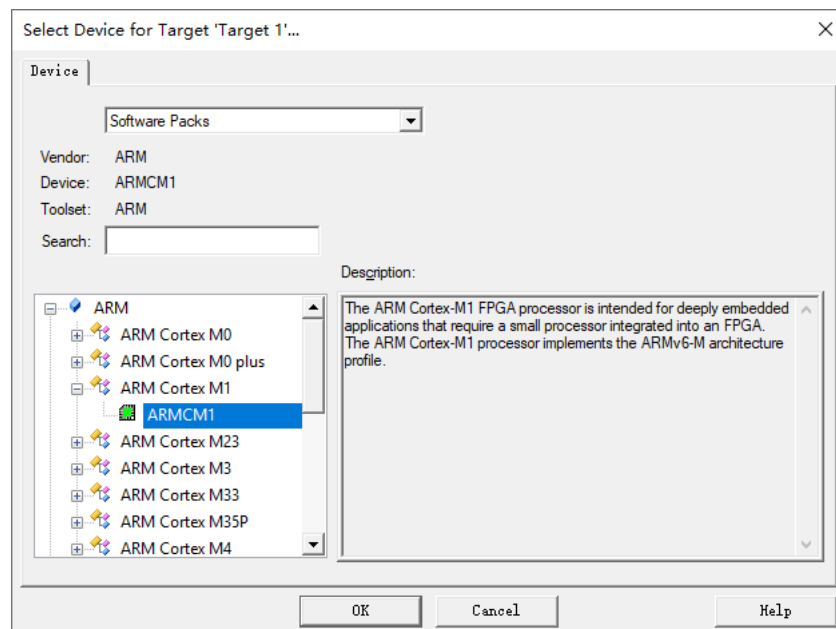


1.2.2 プロジェクトの設定

デバイスの設定

Gowin_EMPU_M1 には ARM Cortex-M1 コアが組み込まれているため、「ARM Cortex M1 > ARMCM1」が選択されます(図 1-2)。

図 1-2 デバイスの設定



ROM と RAM の設定

Gowin_EMPU_M1 の内部命令メモリまたは外部命令メモリが ROM として使用されます。

Gowin_EMPU_M1 の内部データメモリまたは外部データメモリが RAM として使用されます。

1. ROM(Internal Instruction Memory)および RAM(Internal Data Memory)の開始アドレスとサイズを設定します。

ROM の開始アドレスとサイズの設定

- オフチップ SPI-Flash Memory によるダウンロード&ブート：
 - ROM の開始アドレス：0x400。
 - ROM サイズ:ハードウェア設計の ITCM Size に応じて設定してください。例えば、ITCM Size が 32KB の場合、0x7C00 に設定します。
- オンチップ ITCM によるダウンロード&ブート：
 - ROM の開始アドレス：0x00000000。
 - ROM サイズ:ハードウェア設計の ITCM Size に応じて設定してください。例えば、ITCM Size が 32KB の場合、0x8000 に設定します。

RAM の開始アドレスとサイズの設定

- RAM の開始アドレス : 0x20000000。
- RAM サイズ : ハードウェア設計の DTCM Size に応じて設定してください。例えば、DTCM Size が 32KB の場合、0x8000 に設定します。

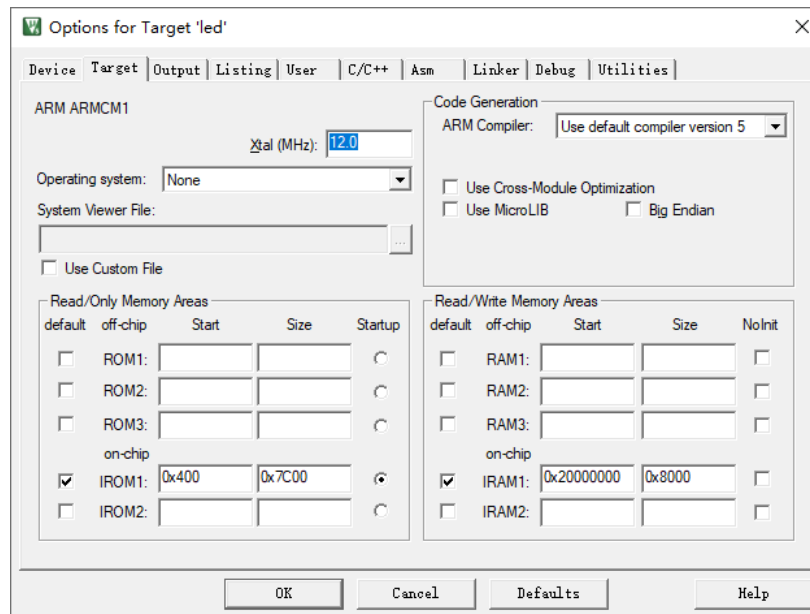
オンチップ Block RAM リソースの制限により、ITCM および DTCM のサイズはオンチップ Block SRAM の最大容量以下に構成する必要があります。

- GW1N/R-9 の場合、ITCM または DTCM のサイズは最大 32KB に構成できますが、ITCM または DTCM が 32KB に構成されている場合、もう 1 つのメモリは最大 16KB になります。
- GW2AN-9X/18X の場合、ITCM または DTCM のサイズは最大 32KB に構成できますが、ITCM または DTCM が 32KB に構成されている場合、もう 1 つのメモリは最大 16KB になります。
- GW2A/R/NR-18 の場合、ITCM または DTCM のサイズは最大 64KB に構成できますが、ITCM または DTCM が 64KB に構成されている場合、もう 1 つのメモリは最大 16KB になります。
- GW2A/N-55 の場合、ITCM または DTCM のサイズは最大 256KB に構成できますが、ITCM または DTCM が 256KB に構成されている場合、もう 1 つのメモリは最大 16KB になります。
- GW5A/T/ST/S-138 の場合、ITCM または DTCM のサイズは最大 512KB に構成できますが、ITCM または DTCM が 512KB に構成されている場合、もう 1 つのメモリは最大 128KB になります。
- GW5AT-75 の場合、ITCM または DTCM のサイズは最大 256KB に構成できますが、ITCM または DTCM が 256KB に構成されている場合、もう 1 つのメモリは最大 256KB になります。
- GW5A/R/S-25 の場合、ITCM または DTCM のサイズは最大 64KB に構成できますが、ITCM または DTCM が 64KB に構成されている場合、もう 1 つのメモリは最大 32KB になります。
- GW5A/T-60 の場合、ITCM または DTCM のサイズは最大 128KB に構成できますが、ITCM または DTCM が 128KB に構成されている場合、もう 1 つのメモリは最大 64KB になります。
- GW5AT/RT-15 シリーズの場合、ITCM または DTCM のサイズは最大 64KB に構成できますが、ITCM または DTCM が 64KB に構成されている場合、もう 1 つのメモリは最大 8KB になります。

ROM(Internal Instruction Memory)と RAM(Internal Data Memory)の設定を図 1-3 に示します。

例えば、DK_START_GW5A-LV25UG324 V2.0 開発ボードのリファレンス・デザインの場合、「ROM」の開始アドレスは「0x400」、「Size」は「0x7C00」に設定され、「RAM」の開始アドレスは「0x20000000」、「Size」は「0x8000」に設定されています。

図 1-3 ROM と RAM の設定



2. ROM(External Instruction Memory)および RAM(External Data Memory)の開始アドレスとサイズを構成します。

ROM の開始アドレスとサイズの設定

- ROM の開始アドレス : 0x00000000。
- ROM のサイズ : 実際の **Size** に応じて設定してください。
- RAM の開始アドレスとサイズの設定
- RAM の開始アドレス : 0x20100000。
- RAM のサイズ : 実際の **Size** に応じて設定してください。

出力されるファイルの形式の設定

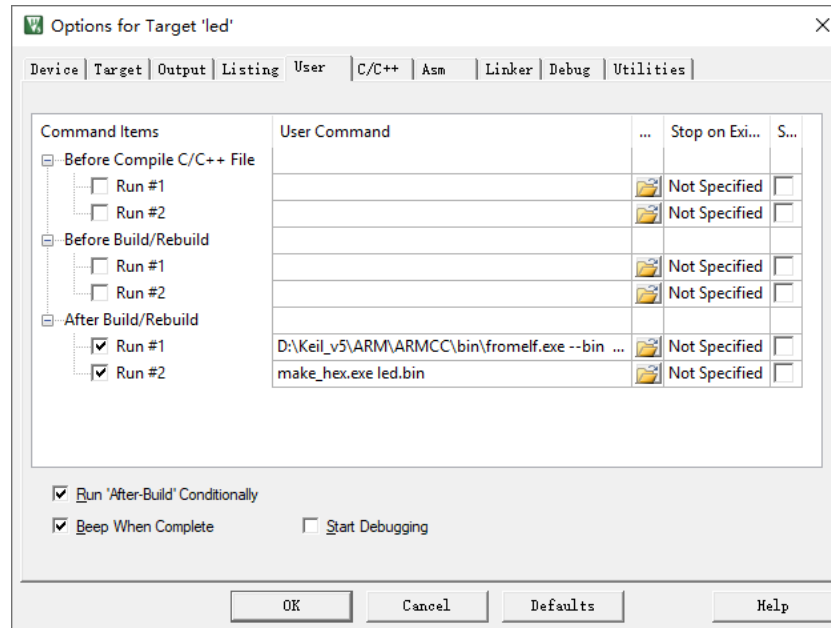
Gowin_EMPU_M1 ソフトウェア設計では実行ファイルを生成する必要があるため、.axf ファイルを.bin ファイルに変換する必要があります。

ITCM の初期値としてソフトウェア設計の実行ファイルを使用する場合、make_hex.exe ツールを使用して、実行ファイル.bin を 4 つの 16 進数のファイル(itcm0、itcm1、itcm2、および itcm3)に変換する必要があります。

ユーザーコマンドラインでファイル形式変換ツールを呼び出す方法は図 1-4 に示すとおりです

- Run #1: fromelf.exe --bin -o bin-file axf-file
- Run #2: make_hex.exe bin-file

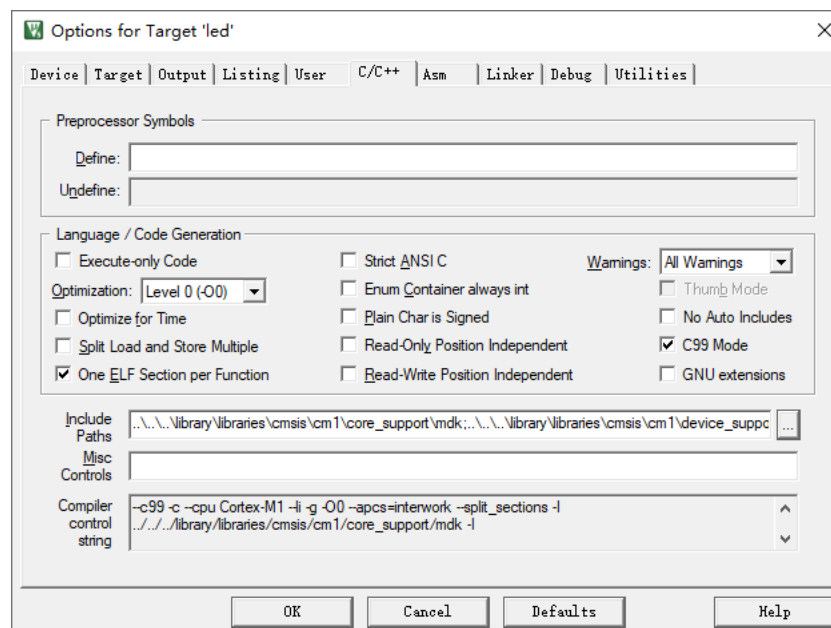
図 1-4 出力されるファイルの形式の設定



ヘッダーファイルのパスの設定

コンパイル中にCヘッダーファイルを呼び出すために使用されるCヘッダーファイルのパスを設定します(図 1-5)。

図 1-5 ヘッダーファイルのパスの設定



デバッグオプションの設定

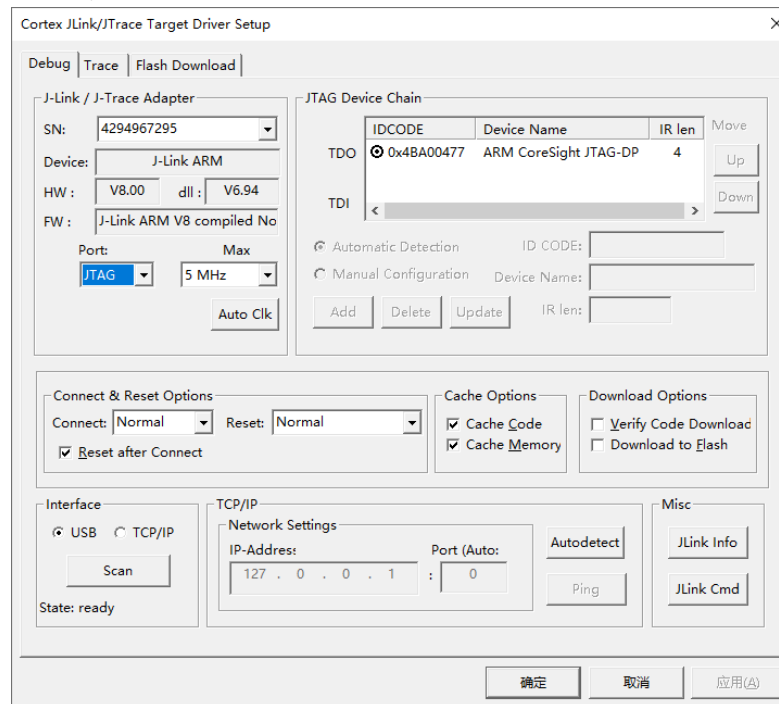
- エミュレータータイプの設定
- U-LINK エミュレーター

U-LINK エミュレーターを選択した場合、「ULNK2/ME Cortex Debugger」

を使用します。

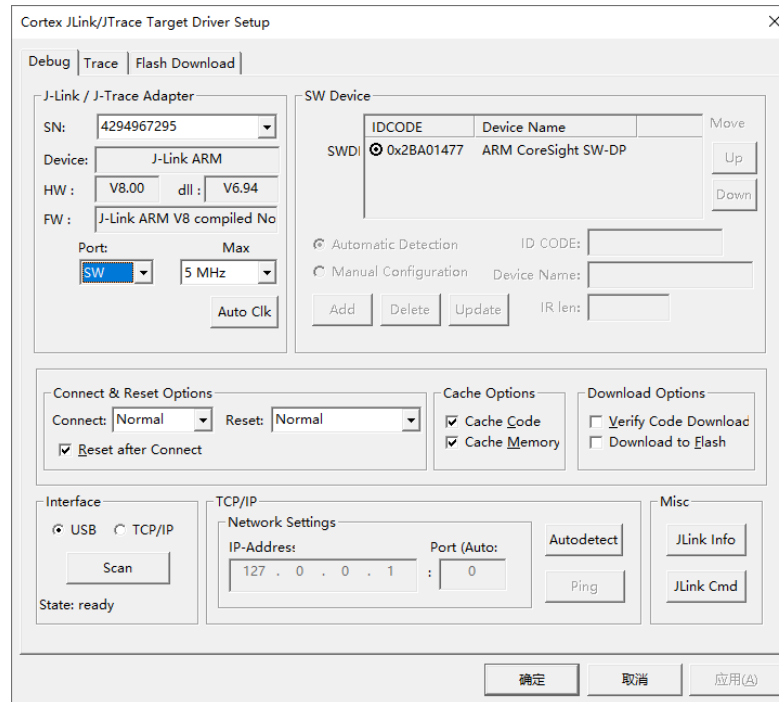
- J-LINK エミュレーター
J-LINK エミュレーターを選択した場合、「J-LINK/J-TRACE Cortex」を使用します。
- デバッグポートのタイプの設定
 - JTAG デバッグポート
図 1-6 は JTAG デバッグポートとして設定した場合のものです。

図 1-6 JTAG デバッグポートの設定



- SWD デバッグポート
図 1-7 は SWD デバッグポートとして設定した場合のものです。

図 1-7 SWD デバッグポートの設定



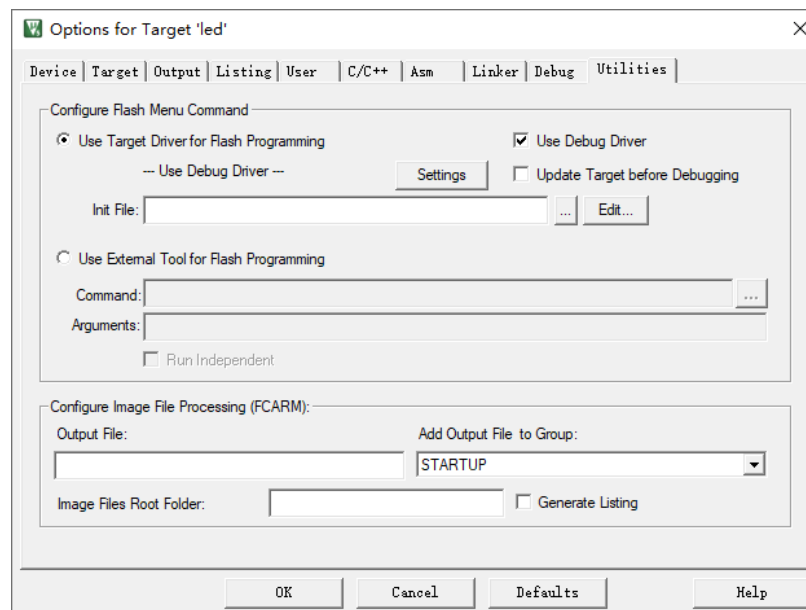
デバッグポートのタイプの設定では、

- 「Download Options > Verify Code Download」をオフにしてください。
- 「Download Options > Download to Flash」をオフにしてください。

Flash オプションの設定

デバッグする必要がある場合は、Update Target before Debugging オプションをオフにしてください(図 1-8)。

図 1-8 Flash オプションの設定

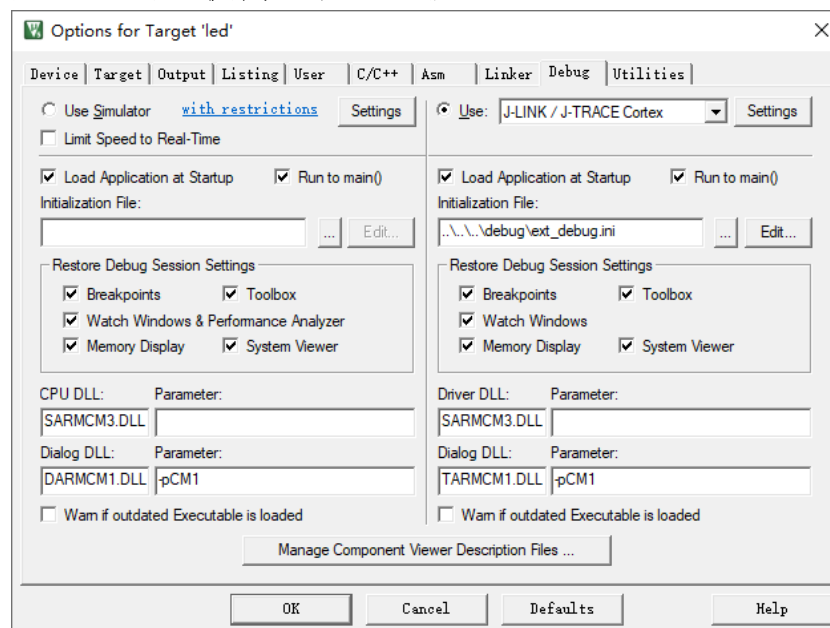


デバッグ初期化ファイルの設定

オフチップ SPI-Flash Memory によるダウンロード&ブートを選択した場合、デバッグする際にデバッグ初期化ファイルをロードする必要があります。「Debug > Initialization File」オプションで、デバッグ初期化ファイル `ext_debug.ini` をロードします(図 1-9)。

「`ext_debug.ini`」ファイルは `library¥debug` にあります。

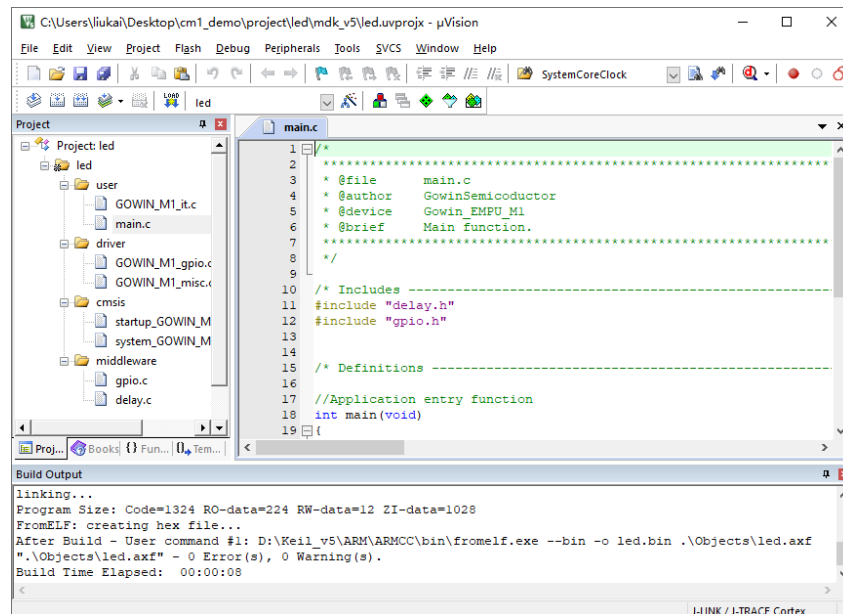
図 1-9 デバッグ初期化ファイルの設定



1.2.3 プロジェクトのコンパイル

プロジェクトの設定とコーディングが完了したら、ツールバーの **Build** 「」もしくは **Rebuild** 「」、またはメニューバーの「Project > Build Target」もしくは「Project > Rebuild all target files」を選択し、コンパイルして実行ファイルまたは 4 つの 16 進数のファイル(itcm0、itcm1、itcm2、および itcm3)を生成します(図 1-10)。

図 1-10 プロジェクトのコンパイル



1.2.4 プロジェクトのダウンロード

Gowin_EMPU_M1 ソフトウェア設計のダウンロードについては、『Gowin_EMPU_M1 ダウンロード ユーザーガイド([IPUG532](#))』を参照してください。

1.2.5 デバッグ

ハードウェア設計のビットストリームファイルとソフトウェア設計の実行ファイルのダウンロードが完了した後、ユーザーのソフトウェア設計に問題がある場合は、U-LINK または J-LINK エミュレーターに接続してデバッグすることができます。

ユーザーは、ハードウェア設計を再コンパイルすることなく、プロジェクトをダウンロードしてデバッグできます。

注記：

デバッグ中の ARM Keil ソフトウェアによる自動ダウンロードはサポートされていません。したがって、各デバッグの前に、ダウンロードツール **Programmer** を使用して、デバッグしたいソフトウェア設計の設計ファイルを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、各デバッグ対象が現在のソフトウェア設計であることが保証されます。

1. シミュレーターの接続

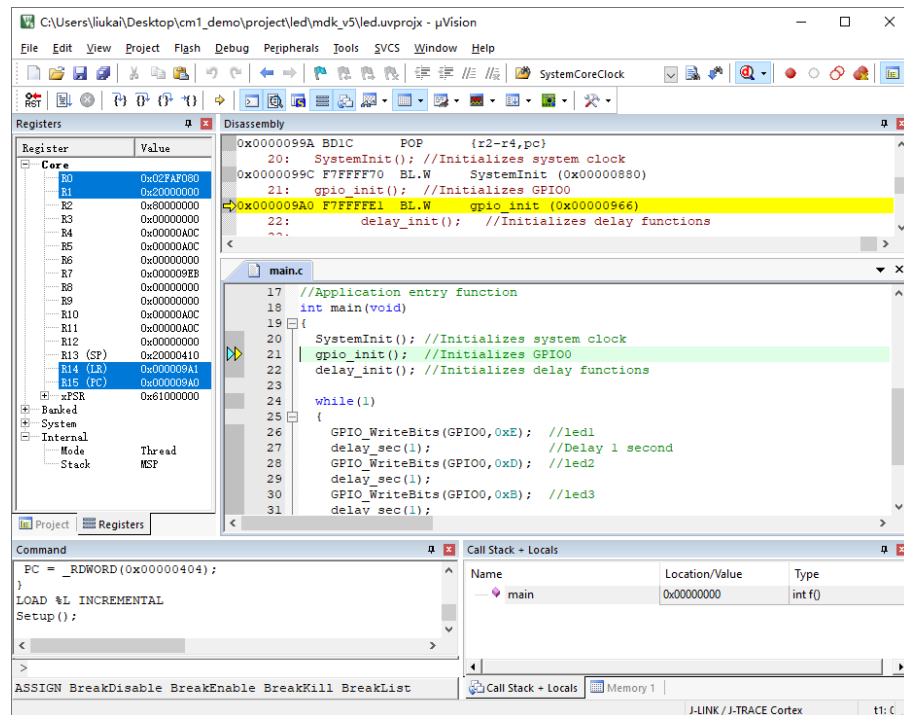
ハードウェア設計で FPGA IO に制約されてい Debug Access Port(JTAG の場合: JTAG_3~JTAG_18、VCC、GND。SWD の場合: JTAG_7、JTAG_9、VCC、GND)の位置に応じて、J-LINK または U-LINK に接続します。

2. デバッグの開始

U-LINK または J-LINK エミュレーターに接続し、ツールバーの Debug ボタン「」、またはメニューバーの「Debug > Start/Stop Debug Session」

を選択してデバッグモードに入ります。図 1-11 に示すように、ブレークポイントの設定、シングルステップ・デバッグ、リセット、実行などの操作を実行できます。

図 1-11 デバッグの開始



1.3 リファレンス・デザイン

ARM Keil ARM(テスト済みバージョン : V5.26)を使用した[リファレンス・デザイン](#) :

…¥ref_design¥MCU_RefDesign¥MDK_RefDesign¥cm1_demo、
cm1_fatfs、cm1_freertos、cm1_rtthread_nano、cm1_tcpip、cm1_ucos_iii

ARM Keil ARM(テスト済みバージョン : V5.26)を使用した[リファレンス・デザイン](#) :

…¥solution¥SD_Host_v2¥ref_design¥MCU_RefDesign¥apb_sd_host

…¥solution¥SD_SPI_IF_v1¥ref_design¥MCU_RefDesign¥cm1_fatfs

...

¥solution¥Embedded_Memory¥ref_design¥MCU_RefDesign¥cm1_demo

...

¥solution¥External_Memory¥ref_design¥MCU_RefDesign¥cm1_demo

2 GMD ソフトウェア

2.1 ソフトウェアのインストールと構成

GMD (GOWIN MCU Designer)のインストールパッケージは、GOWIN の[ホームページ](#)からダウンロードできます。

GMD のインストールと構成については、『GOWIN MCU Designer ユーザーガイド([SUG549](#))』を参照してください。

注記：

GMD 2025.01 以降のバージョンの使用をご推奨します。

2.2 プロジェクトテンプレート

GMD を利用した Gowin_EMPU_M1 ソフトウェアプログラミングには、プロジェクトの作成、プロジェクトの設定、コーディング、コンパイル、ダウンロード、およびデバッグが含まれます。

2.2.1 プロジェクトの新規作成

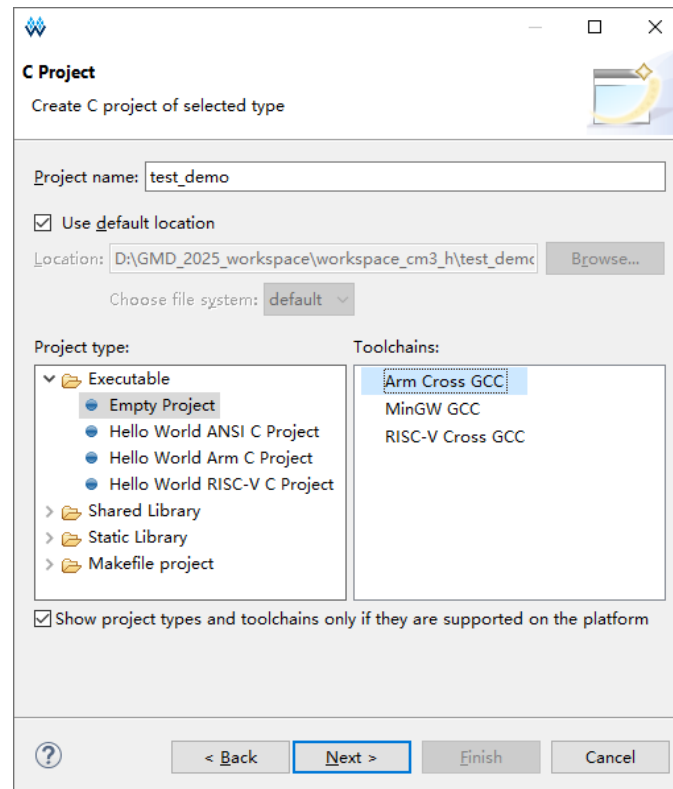
ツールバーの「New」() またはメニューバーの「File > New > C/C++ Project」を選択します。

プロジェクト・タイプとツールチェーンの選択

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 2-1)。

1. プロジェクトの名前とパスを構成します。例えば「Use default location」を選択し、つまり、現在のワークスペースを使用します。
2. プロジェクトのタイプ(Empty Project)を選択します。
3. コンパイルツールチェーン(Arm Cross GCC)を選択します。

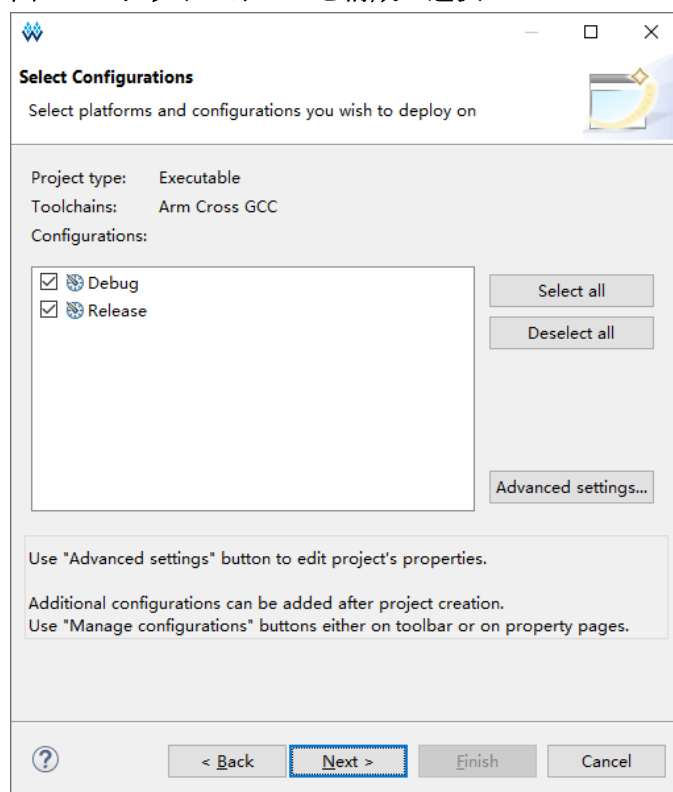
図 2-1 プロジェクト・タイプとツールチェーンの選択



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 2-2)。

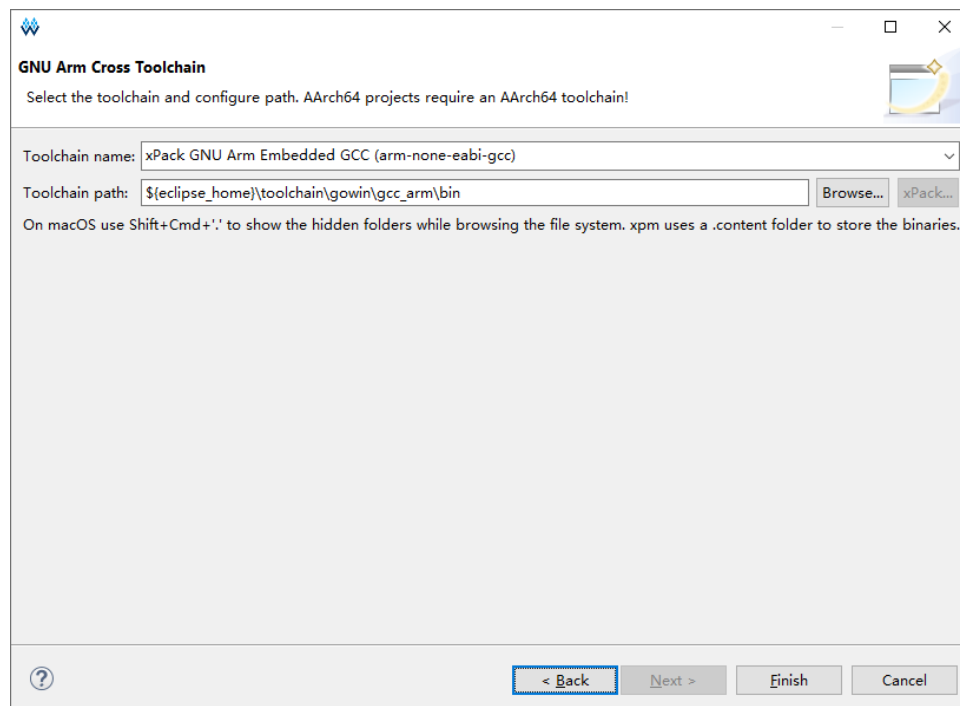
図 2-2 プラットフォームと構成の選択



ソフトウェア・ツールチェーンおよびそのパスの選択

特定の **Arm** 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。**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。たとえば、「**xPack GNU Arm Embedded GCC (arm-none-eabi-gcc)**」およびそのパスを選択します(図 2-3)。

図 2-3 ツールチェーンおよびそのパスの選択



プロジェクトの構造とファイルの作成

ソフトウェア・プロジェクトを新規作成した後、「Project Explorer」ビューで新規プロジェクトを選択してプロジェクトの構造とファイルを作成し、ソフトウェア開発を開始します。

コードの更新がある場合は、「Project Explorer」ビューで現在のプロジェクトを選択し、右クリックして「Refresh」オプションを選択すると、現在のプロジェクトの構造とファイルが自動的に更新されます。

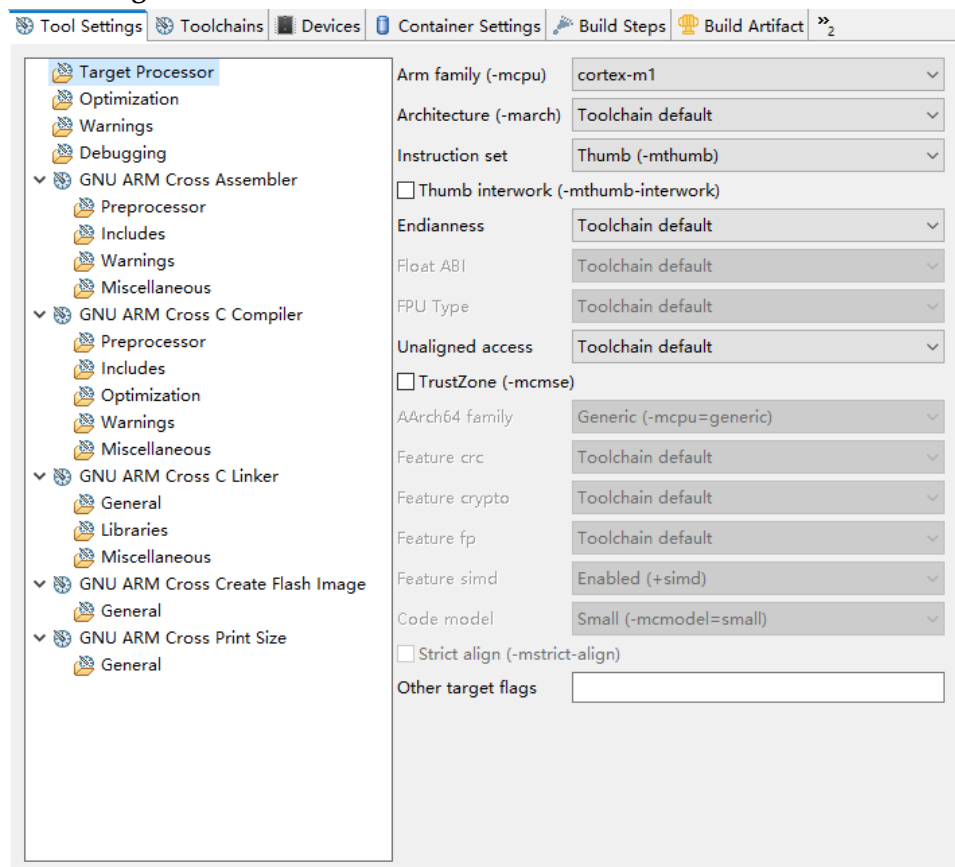
2.2.2 プロジェクトの設定

「Project Explorer」ビューで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。

Target Processor

「Target Processor」オプションは Arm ターゲットプロセッサを設定するために使用されます。例えば「Arm family (-mcpu)」を cortex-m1 に設定します(図 2-4)。

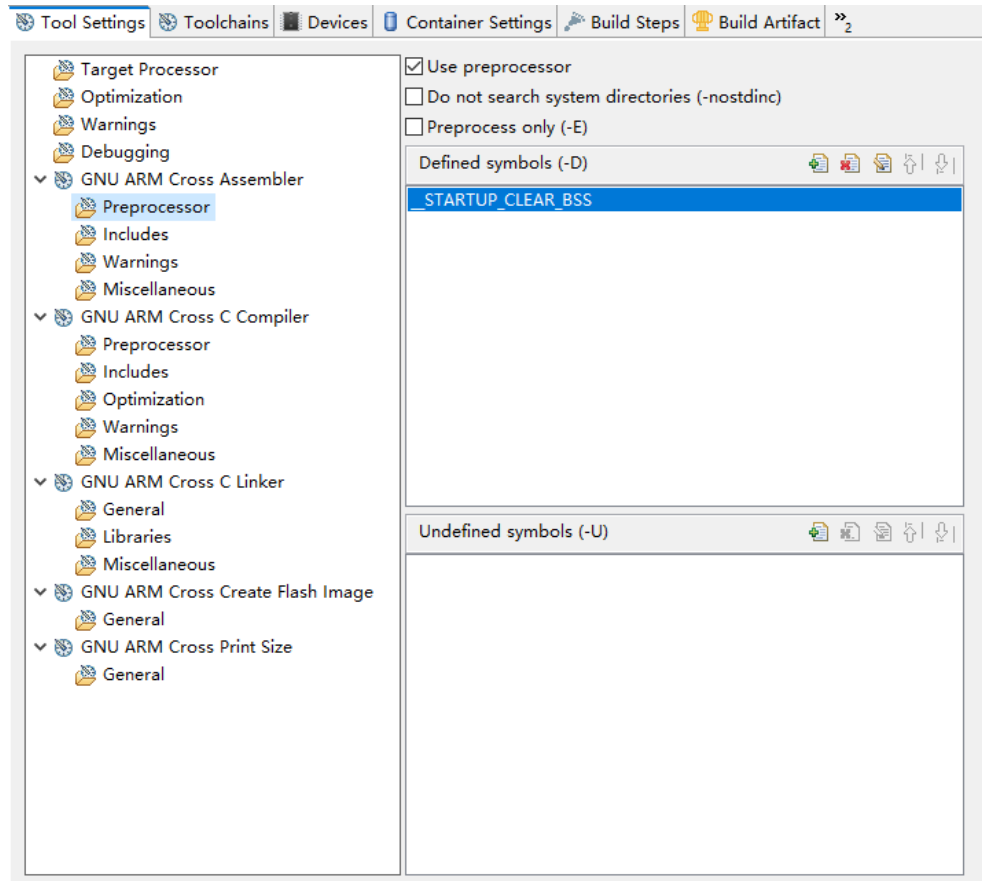
図 2-4 Target Processor



GNU ARM Cross Assembler > Preprocessor

「GNU ARM Cross Assembler > Preprocessor > Defined symbols (-D)」オプションは、マクロ定義を設定するために使用され、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。例えば、このオプションが「__STARTUP_CLEAR_BSS」として設定されています(図 2-5)。

図 2-5 GNU ARM Cross Assembler > Preprocessor



GNU ARM Cross C Compiler > Includes

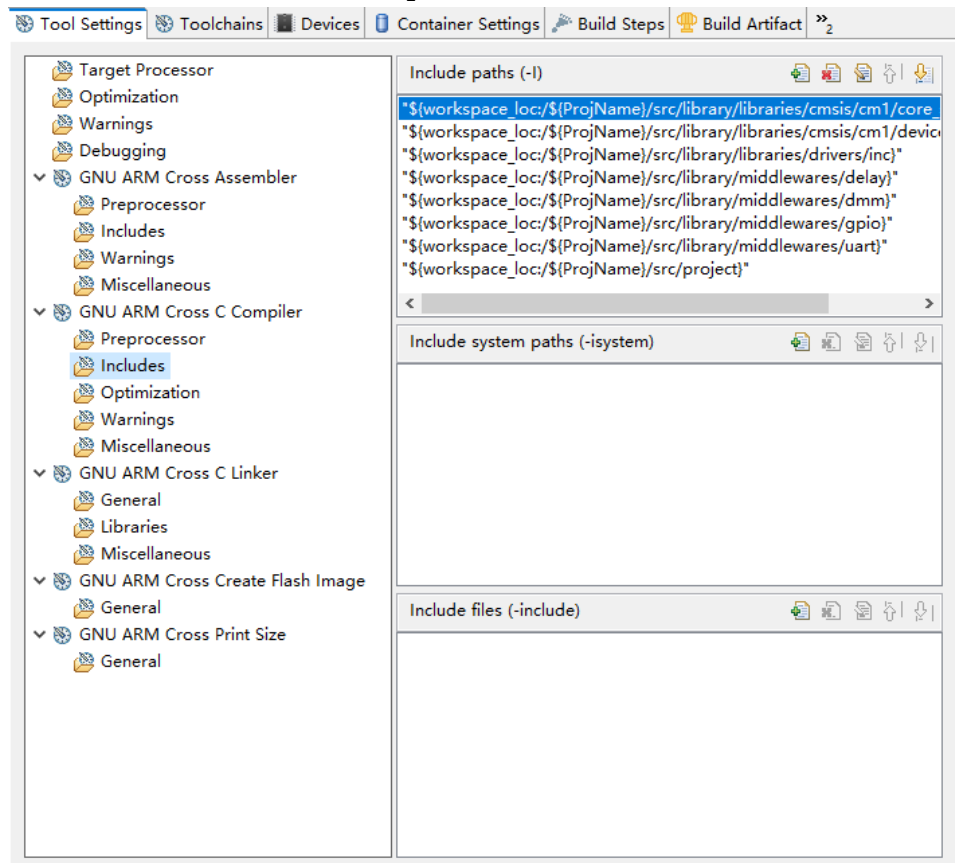
「GNU ARM Cross C Compiler > Includes > Include paths (-I)」オプションは、ヘッダーファイルの検索パスを指定するために使用されます。絶対パス、相対パスを入力できます(図 2-6)。

例えば、ソフトウェア・リファレンス・デザイン

GMD_RefDesign¥cm1_demo の場合、C ヘッダーファイルのパスの設定は以下ようになります：

- "\${workspace_loc}/\${ProjName}/src/library/libraries/cmsis/cm1/core_support/gmd}"
- "\${workspace_loc}/\${ProjName}/src/library/libraries/cmsis/cm1/device_support}"
- "\${workspace_loc}/\${ProjName}/src/library/libraries/drivers/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/uart}"
- "\${workspace_loc}/\${ProjName}/src/project}"

図 2-6 GNU ARM Cross C Compiler > Includes



GNU ARM Cross C Linker

「GNU Arm Cross C Linker > General > Script files (-T)」オプションは、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。例えば、「GOWIN_M1_flash_burn.ld」または「GOWIN_M1_flash_xip.ld」を Flash リンカー・スクリプト・ファイルとして構成します(図 2-7)。

例えば、ソフトウェア・リファレンス・デザイン

GMD_RefDesign¥cm1_demo の場合、Flash リンカー・スクリプト・ファイルを

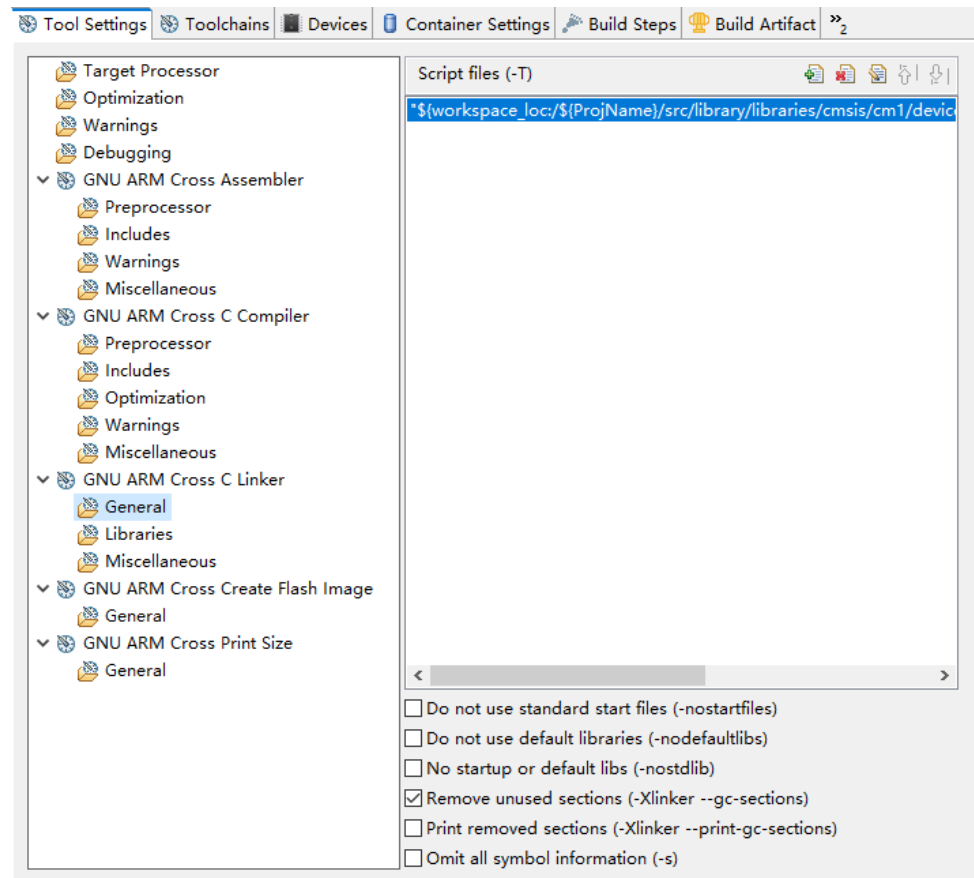
`${workspace_loc}/${ProjName}/src/library/libraries/cmsis/cm1/device_support/startup/gmd/linker/GOWIN_M1_flash_burn.ld`として設定することができます。

Flash リンカー・スクリプト・ファイル内の Flash 開始アドレス(FLASH ORIGIN)の設定方法は次のとおりです。

- Internal Instruction Memory
 - GOWIN_M1_flash_xip.ld : FLASH ORIGIN : 0x00000000。ITCM Initialization によるダウンロード&実行。
 - GOWIN_M1_flash_burn.ld : FLASH ORIGIN : 0x00000400。オフチップ SPI-Flash Memory によるダウンロード&ブート。
- External Instruction Memory

- GOWIN_M1_flash_xip.ld : FLASH ORIGIN : 0x00000000。

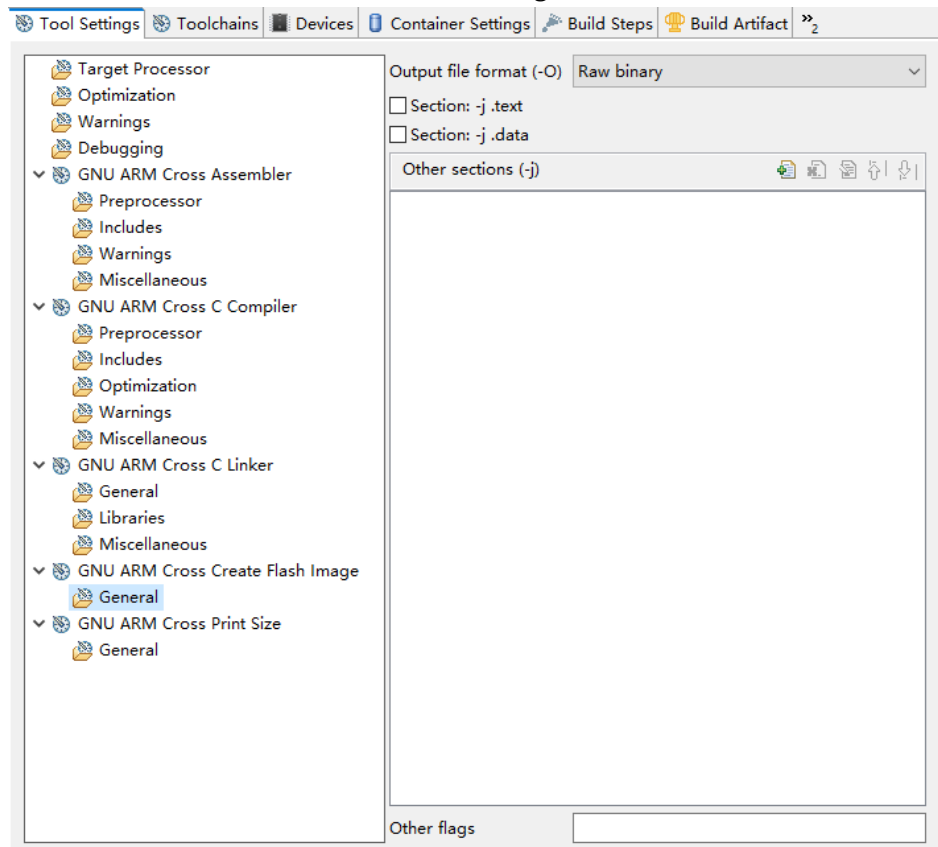
図 2-7 GNU ARM Cross C Linker > General



GNU ARM Cross Create Flash Image

「GNU ARM Cross Create Flash Image > General > Output file format (-O)」オプションは、Raw binary など Arm MCU 実行ファイルの出力形式を設定するために使用されます(図 2-8)。

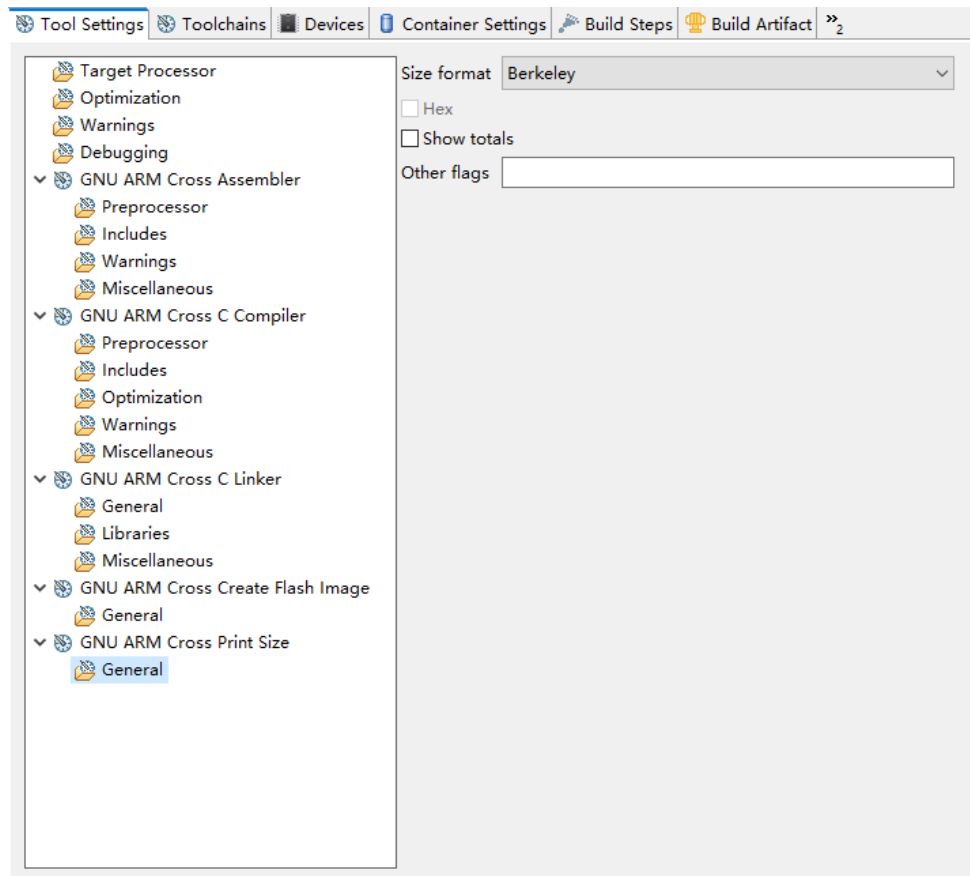
図 2-8 GNU ARM Cross Create Flash Image



GNU ARM Cross Print Size

Arm MCU 実行ファイル内の各セグメント (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。例えば、**Berkeley** として設定します(図 2-9)

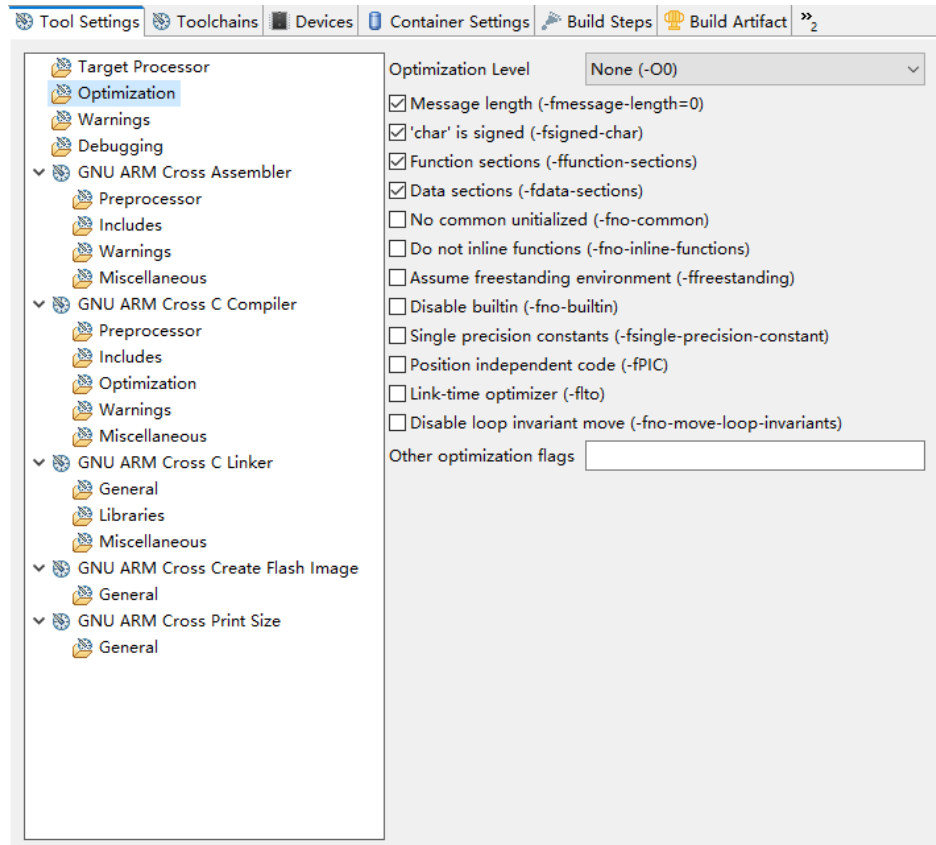
図 2-9 GNU ARM Cross Print Size



Optimization

「Optimization」オプションは、-O0、-O1、-O2、-O3、-Os、-Ofast、-Og、-Oz などの最適化レベルを設定してコンパイル速度、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使用されます(図 2-10)。

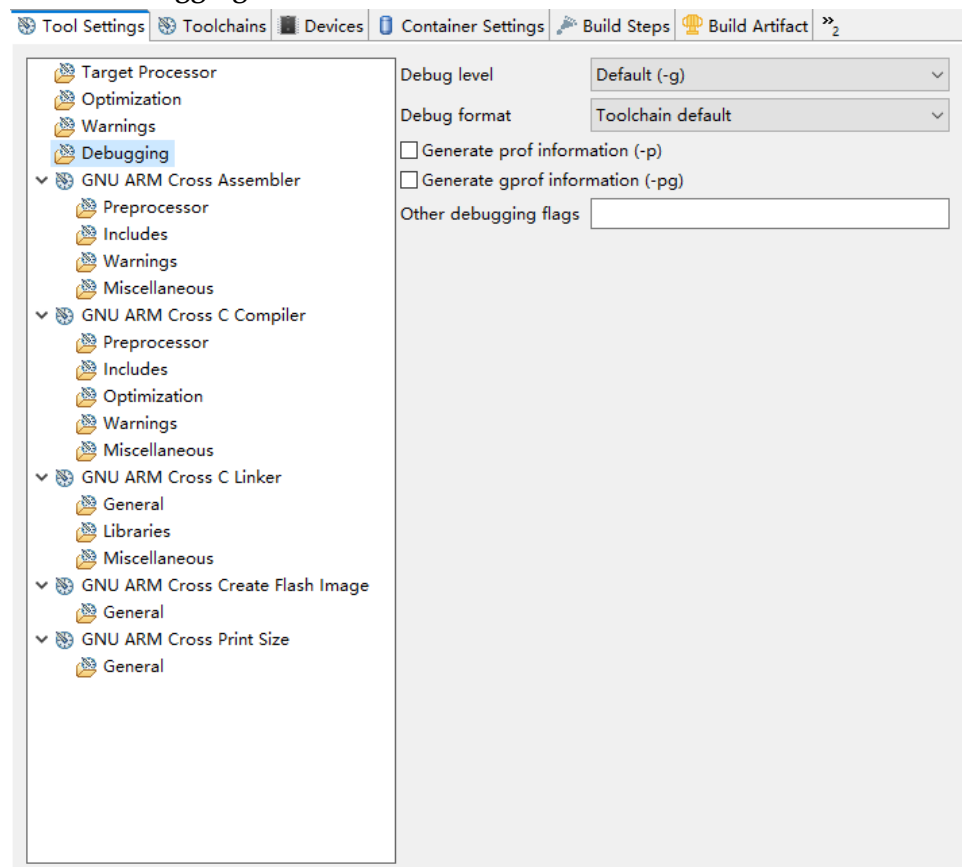
図 2-10 Optimization



Debugging

「Debugging」オプションは-g、-g1、-g3などのデバッグ・レベルを設定するために使用されます(図 2-11)。

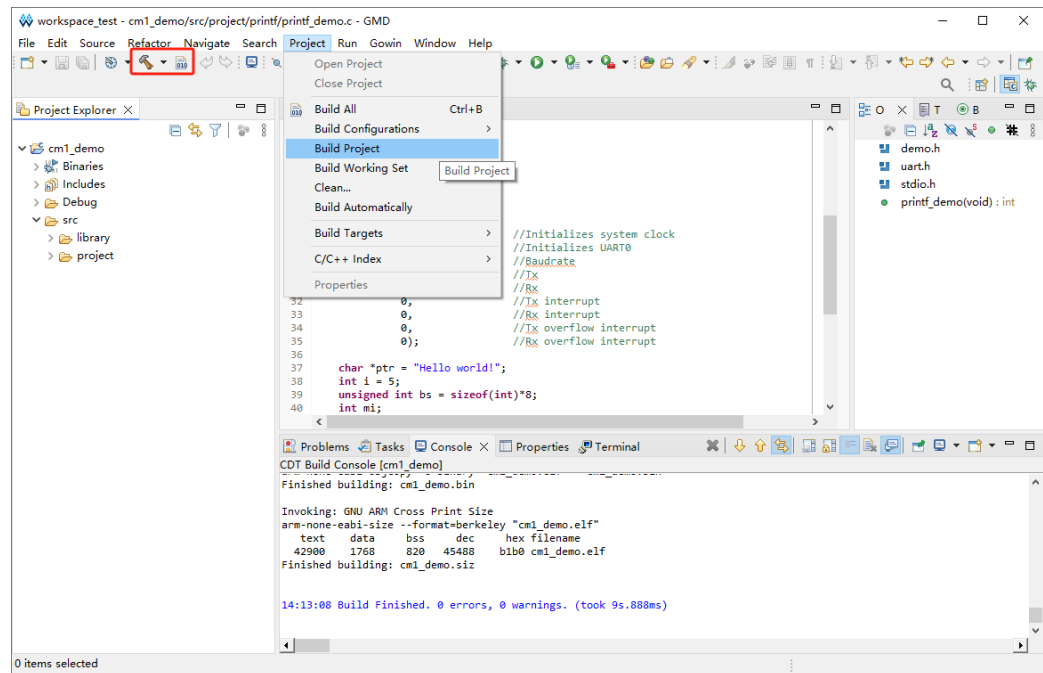
図 2-11 Debugging



2.2.3 プロジェクトのコンパイル

プロジェクトの設定とコーディングが完了した後、プロジェクトをコンパイルします。ツールバー上の **Build** 「🔧」 もしくは **Build All** 「📁」 、またはメニューバー上の「**Project > Build Project**」 もしくは「**Project > Build All**」 を選択してコンパイルして実行ファイルを生成します(図 2-12)。

図 2-12 プロジェクトのコンパイル



2.2.4 プロジェクトのダウンロード

Gowin_EMPU_M1 ソフトウェア設計のダウンロードについては、『Gowin_EMPU_M1 ダウンロード ユーザーガイド([IPUG532](#))』を参照してください。

2.2.5 デバッグ

Gowin_EMPU_M1 ソフトウェア設計の実行ファイルのダウンロードが完了した後、当該ソフトウェア設計に問題がある場合は、開発ボードを GWU2X、GWUSB、J-LINK などのエミュレーターに接続して当該ソフトウェア設計をデバッグすることができます。

注記：

デバッグ中の GMD ソフトウェアによる自動ダウンロードはサポートされていません。したがって、各デバッグの前に、ダウンロードツール **Programmer** を使用して、デバッグしたいソフトウェア設計の設計ファイルを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、各デバッグ対象が現在のソフトウェア設計であることが保証されます。

Gowin_EMPU_M1 のデバッグプロセス：

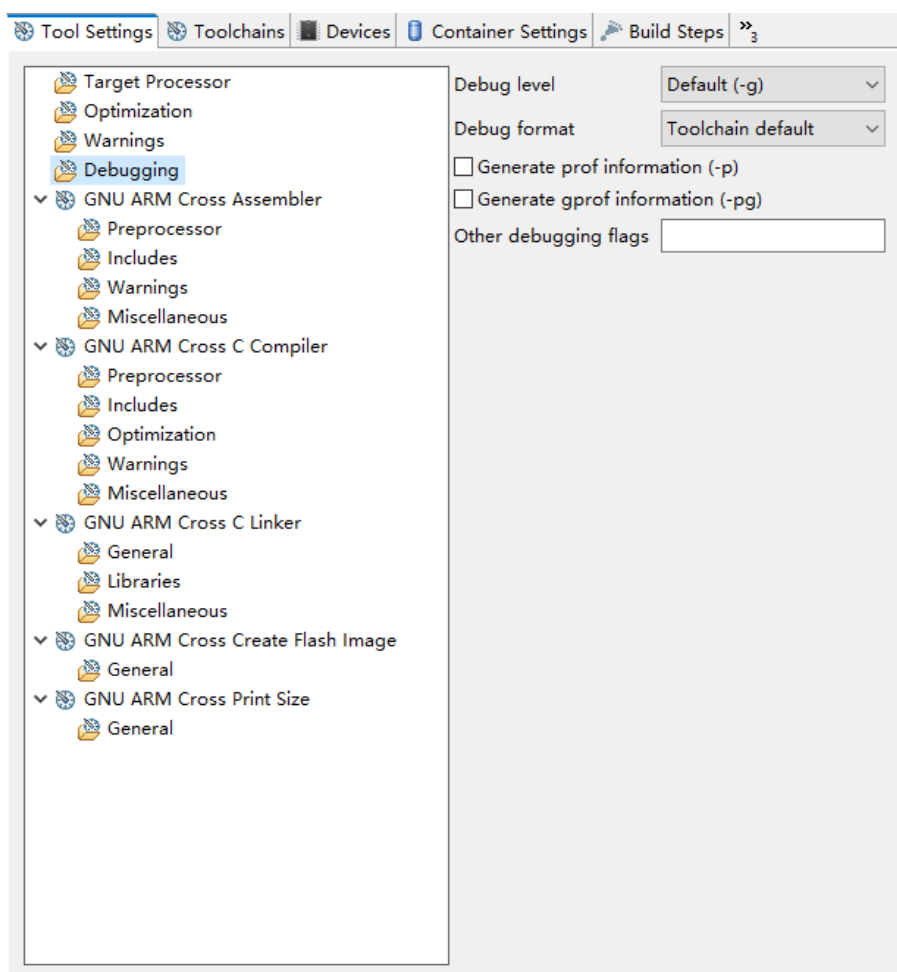
- デバッグレベルの設定
- デバッグモードの設定
- シミュレーターへの接続
- デバッグの開始

デバッグレベルの設定

Project Explorer ビューで、現在のプロジェクトの「Properties > C/C++ Build > Settings > Tool Settings > Debugging > Debug level」でデバッグレベルを選択します。「Default(-g)」または「Maximum(-g3)」に設定するこ

とをお勧めします(図 2-13)。

図 2-13 デバッグレベルの設定

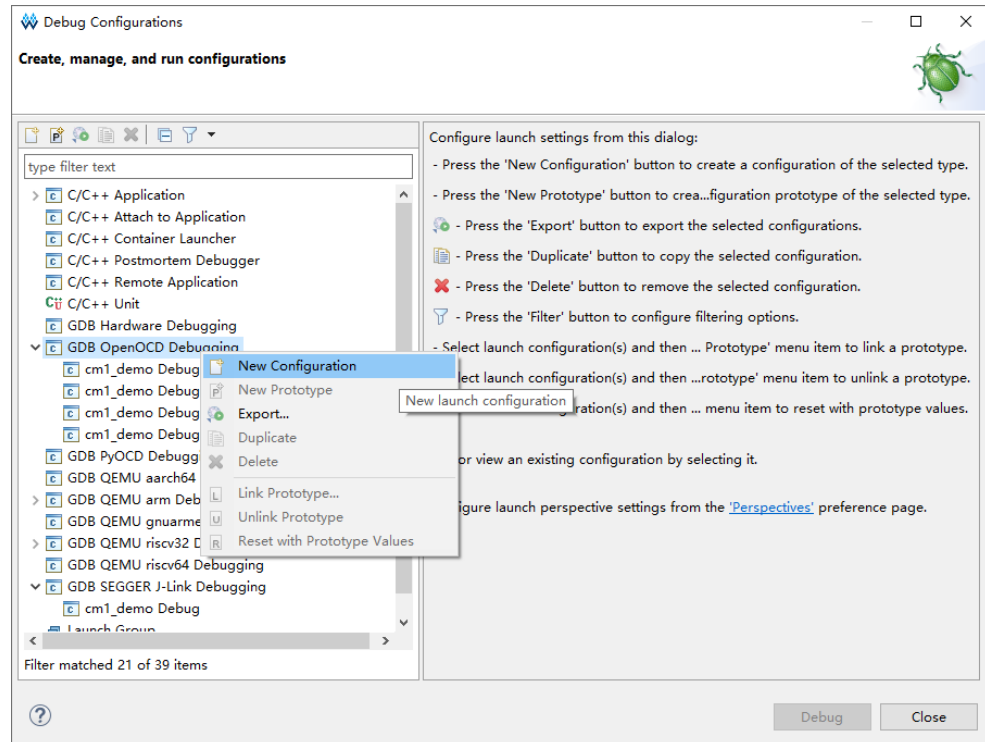


デバッグモードの設定

メニューバー > 「Run > Debug Configurations…」オプションを選択します Gowin_EMPU_M1 は GDB OpenOCD Debugging、GDB SEGGER J-Link Debugging デバッグモードをサポートしています。

例えば、GDB OpenOCD Debugging デバッグモードと GWU2X エミュレーターを選択します。「Debug Configurations > GDB OpenOCD Debugging」を右クリックして「New Configuration」を選択し、GDB OpenOCD Debugging デバッグモードのデバッグ・ビューを作成します(図 2-14)。

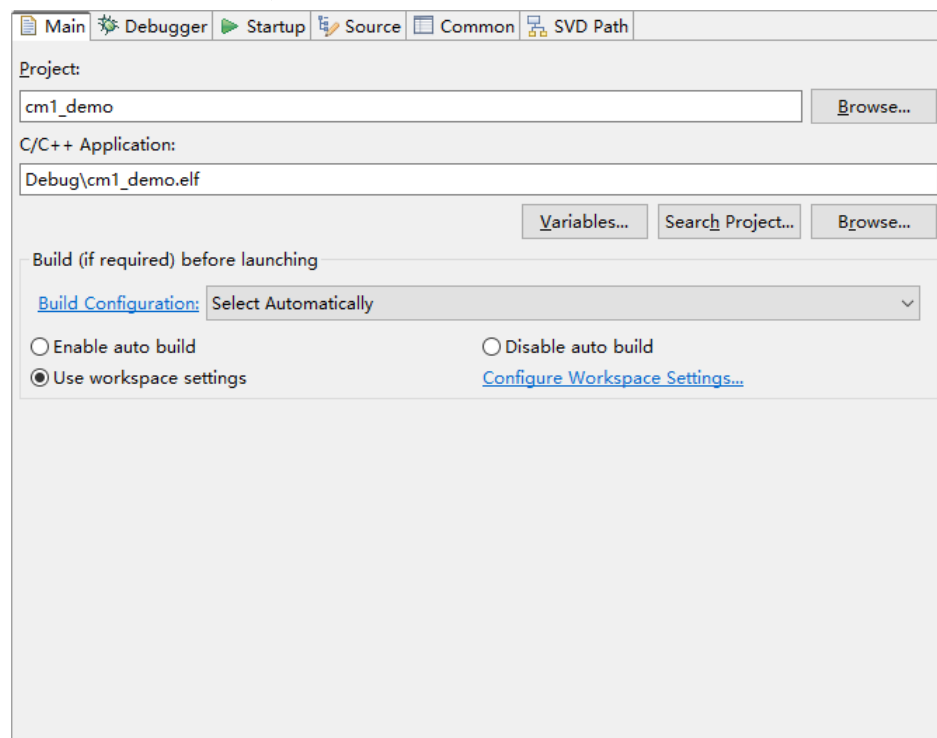
図 2-14 デバッグ・ビューの作成



作成されたデバッグ・ビューには、Main、Debugger、Startup などのタブがあります。各タブの設定は以下に示すとおりです。

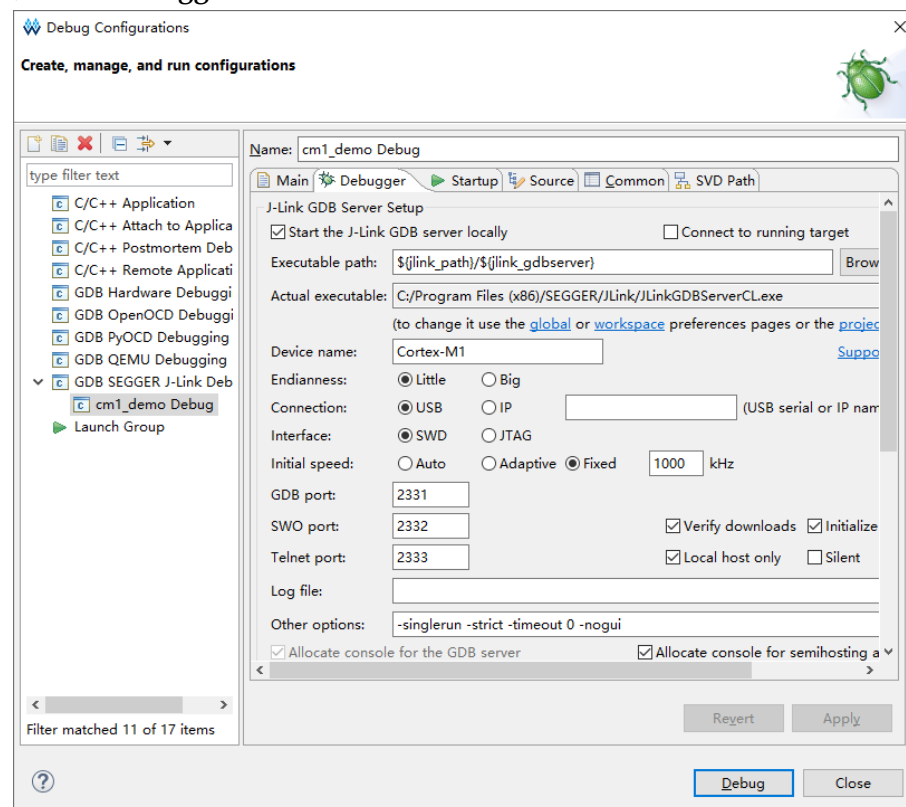
1. 「Main」タブは、Project と C/C++ Application などのオプションを設定するために使用されます(図 2-15)。

図 2-15 Main タブの設定



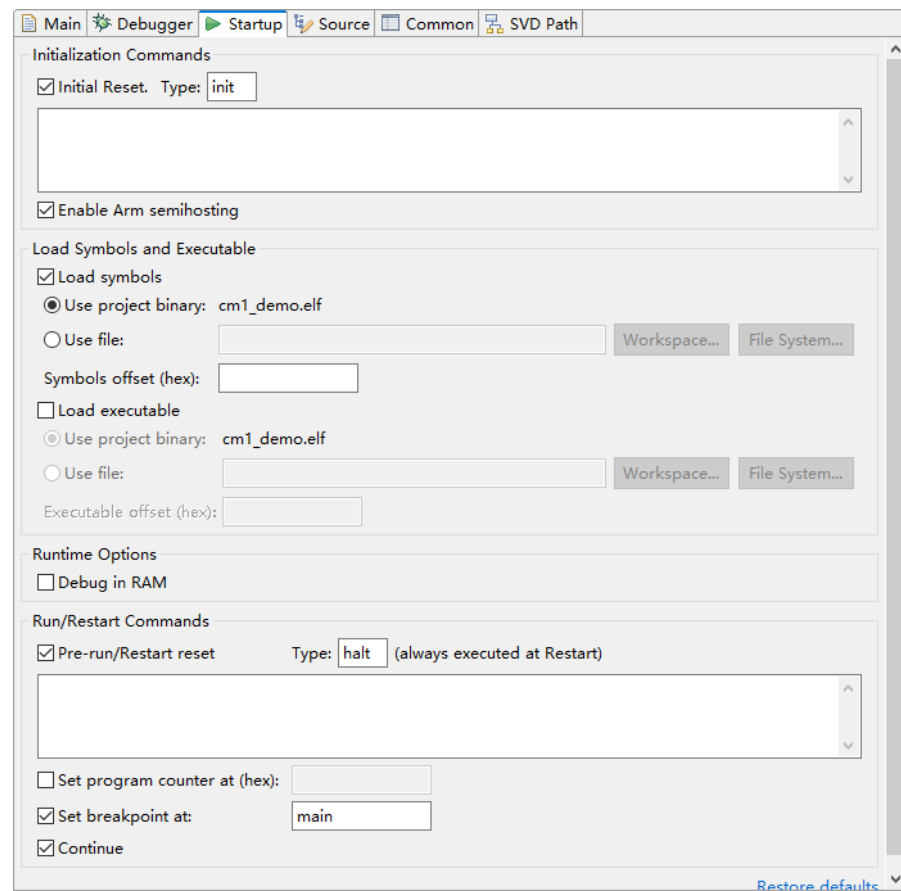
2. 「Debugger」タブは OpenOCD と GDB などのオプションを設定するために使用されます(図 2-16)。
- Start OpenOCD locally
 - GDB port: 3333
 - Telnet port: 4444
 - Tcl port: 6666
 - Config options: -f board/gowin_u2x_m1.cfg。その他の一般的なエミュレーターには、GWUSB (gowin_ftdi_dual_m1.cfg または gowin_ftdi_single_m1.cfg)、J-Link(WinUSB ドライバーを使用する場合は gowin_jlink_m1.cfg)などがあります。
 - Commands: set mem inaccessible-by-default off

図 2-16 Debugger タブ



3. 「Startup」ビューは、デバッグなどのオプションを設定するために使用されます(図 2-17)。
- Initial Reset. Type: init
 - Enable Arm semihosting
 - Load symbols
 - Pre-run/Restart rest
 - Set breakpoint at: main
 - Continue

図 2-17 Startup タブ

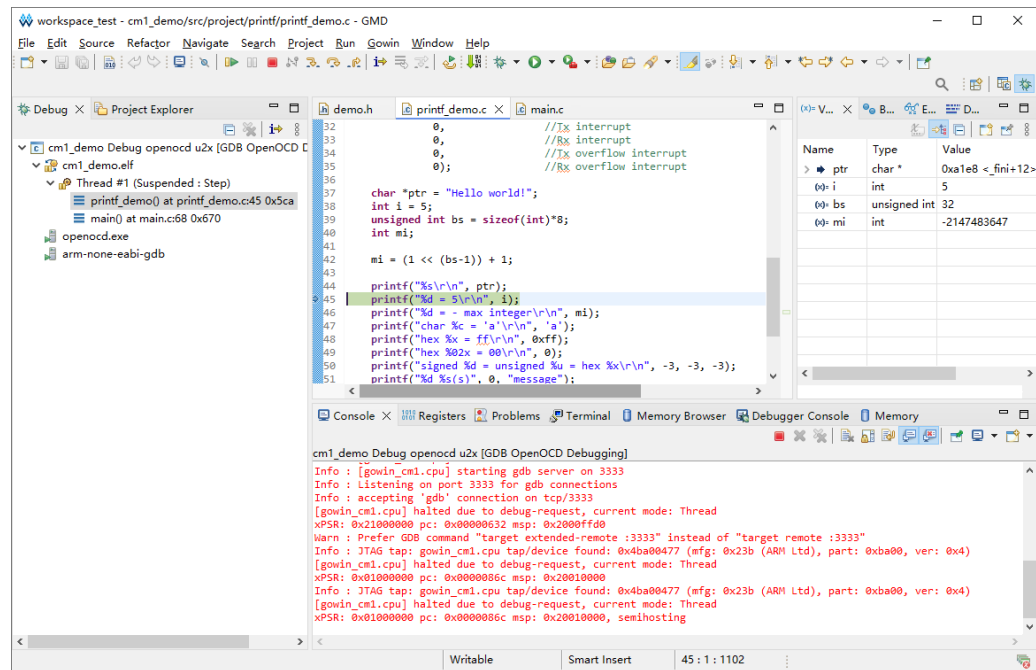


デバッグの開始

例えば、GWU2X シミュレーターに接続します。

ツールバー上の **Debug** (🔧) をクリックしてデバッグ・ビューを起動します(図 2-18)。

図 2-18 デバッグ・ビューの起動



デバッグ・ビューには次のものが含まれます。

- 変数ビュー：この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。
- 式ビュー：このビューでは、カスタム式を追加して値を監視できます。
- ブレークポイントビュー：このビューには、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を管理できます。
- 逆アセンブリビュー：このビューには、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。
- コンソールビュー：このビューにはプロセス出力が表示されます。
- メモリビュー：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。
- メモリブラウザビュー：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。
- レジスタビュー：このビューには、レジスタのビット・フィールド・レベルで MCU 情報が表示されます。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。

2.3 リファレンス・デザイン

GMD(テスト済みバージョン：2025.01)を使用した[リファレンス・デザイン](#)：

…¥ref_design¥MCU_RefDesign¥GMD_RefDesign¥cm1_demo、
cm1_fatfs、cm1_freertos、cm1_rtthread_nano、cm1_ucos_iii

