

# Gowin PicoRV32 IDE リファレンスマニュアル

IPUG910-1.6J, 2025-09-26

## 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                                                                                                                                                                                                                                                                                                                |
|------------|--------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 2020/01/16 | 1.0J   | 初版。                                                                                                                                                                                                                                                                                                               |
| 2020/03/06 | 1.1J   | <ul style="list-style-type: none"><li>● MCU が Wishbone インターフェースの周辺機器 GPIO をサポート。</li><li>● MCU が拡張 AHB バスインターフェースをサポート。</li><li>● MCU がオフチップ SPI-Flash によるダウンロードをサポート。</li><li>● MCU がオフチップ SPI-Flash の読み出し、書き込み、および消去をサポート。</li><li>● MCU が Hardware Stack Protection および Trap Stack Overflow 機能をサポート。</li></ul> |
| 2020/06/01 | 1.2J   | <ul style="list-style-type: none"><li>● MCU のオンラインデバッグをサポート。</li><li>● MCU コアの割り込み処理機能を強化。</li><li>● MCU コアの命令を最適化。</li></ul>                                                                                                                                                                                     |
| 2021/07/16 | 1.3J   | MCU のソフトウェア・リファレンス・デザインを更新。                                                                                                                                                                                                                                                                                       |
| 2022/01/29 | 1.4J   | <ul style="list-style-type: none"><li>● MCU のソフトウェア・リファレンス・デザインを更新。</li><li>● IDE のオプション構成を最適化。</li><li>● IDE のオンラインデバッグプロセスを最適化。</li></ul>                                                                                                                                                                       |
| 2023/08/18 | 1.5J   | Arora V FPGA 製品のサポートを追加。                                                                                                                                                                                                                                                                                          |
| 2025/07/18 | 1.5.1J | 一部のリンクを更新。                                                                                                                                                                                                                                                                                                        |
| 2025/09/26 | 1.6J   | <ul style="list-style-type: none"><li>● GMD ソフトウェアのバージョン情報を更新。</li><li>● 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。</li></ul>                                                                                                                                                                              |

# 目次

|                                                    |           |
|----------------------------------------------------|-----------|
| 目次 .....                                           | i         |
| 図一覧 .....                                          | ii        |
| 表一覧 .....                                          | iii       |
| <b>1 GOWIN MCU Designer のインストールと構成 .....</b>       | <b>1</b>  |
| <b>2 プロジェクトテンプレート .....</b>                        | <b>2</b>  |
| 2.1 プロジェクトの作成 .....                                | 2         |
| 2.1.1 プロジェクト・タイプとツールチェーンの選択 .....                  | 2         |
| 2.1.2 プラットフォームと構成の選択 .....                         | 3         |
| 2.1.3 ソフトウェア・ツールチェーンおよびそのパスの選択 .....               | 3         |
| 2.1.4 プロジェクトの構造とファイルの作成 .....                      | 4         |
| 2.2 プロジェクトの設定 .....                                | 5         |
| 2.2.1 Target Processor .....                       | 5         |
| 2.2.2 Optimization .....                           | 8         |
| 2.2.3 Debugging .....                              | 11        |
| 2.2.4 GNU RISC-V Cross C Compiler > Includes ..... | 12        |
| 2.2.5 GNU RISC-V Cross C Linker .....              | 13        |
| 2.2.6 GNU RISC-V Cross Print Size .....            | 15        |
| 2.2.7 GNU RISC-V Cross Create Flash Image .....    | 16        |
| 2.3 プロジェクトのコンパイル .....                             | 17        |
| 2.4 プロジェクトのダウンロード .....                            | 18        |
| 2.5 プロジェクトのデバッグ .....                              | 18        |
| 2.5.1 デバッグレベルの設定 .....                             | 18        |
| 2.5.2 Flash リンカー・スクリプト・ファイルの設定 .....               | 19        |
| 2.5.3 デバッグモードの設定 .....                             | 19        |
| 2.5.4 デバッグの開始 .....                                | 24        |
| <b>3 リファレンス・デザイン .....</b>                         | <b>26</b> |

## 図一覧

|                                                    |    |
|----------------------------------------------------|----|
| 図 2-1 プロジェクト・タイプとツールチェーンの選択.....                   | 2  |
| 図 2-2 プラットフォームの選択.....                             | 3  |
| 図 2-3 ツールチェーンおよびそのパスの選択.....                       | 4  |
| 図 2-4 Target Processor .....                       | 6  |
| 図 2-5 Optimization .....                           | 9  |
| 図 2-6 Debugging .....                              | 12 |
| 図 2-7 GNU RISC-V Cross C Compiler > Includes ..... | 13 |
| 図 2-8 GNU RISC-V Cross C Linker .....              | 14 |
| 図 2-9 GNU RISC-V Cross Print Size .....            | 16 |
| 図 2-10 GNU RISC-V Cross Create Flash Image .....   | 17 |
| 図 2-11 プロジェクトのコンパイル .....                          | 18 |
| 図 2-12 デバッグレベルの設定 .....                            | 19 |
| 図 2-13 デバッグ・ビューの作成.....                            | 20 |
| 図 2-14 Main タブの設定.....                             | 21 |
| 図 2-15 Debugger タブの設定.....                         | 22 |
| 図 2-16 Startup タブの設定.....                          | 23 |
| 図 2-17 デバッグ・ビューの起動.....                            | 24 |

## 表一覧

|                                                |   |
|------------------------------------------------|---|
| 表 2-1 32 ビットの RISC-V アーキテクチャのプロセッサのデータ型の幅..... | 7 |
|------------------------------------------------|---|

# 1 GOWIN MCU Designer のインストールと構成

GOWIN MCU Designer(GMD)は、Gowin\_PicoRV32 のソフトウェア設計、コンパイル、ダウンロード、およびデバッグなどをサポートしています。

GMD 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 のドライバーインストール、および Gowin\_PicoRV32 用のデバッグドライバ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(バージョン 2025.01)の使用をお勧めします。

# 2 プロジェクトテンプレート

## 2.1 プロジェクトの作成

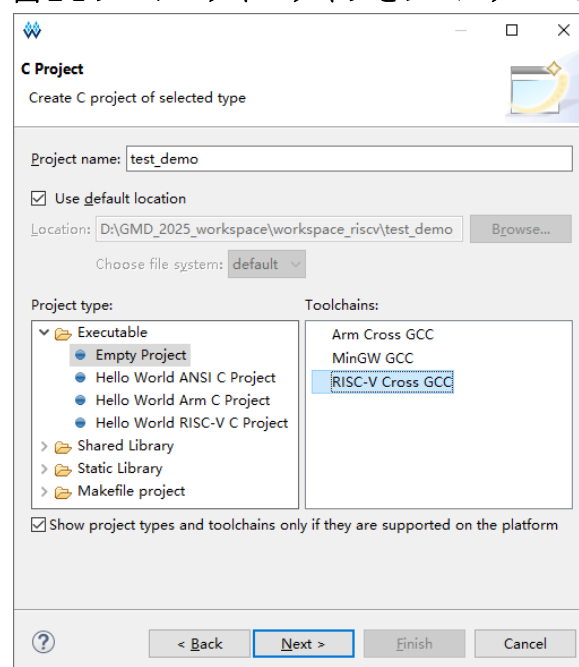
ツールバー上の「New」(📁) またはメニューバー上の「File > New > C/C++ Project > C Managed Build」を選択します。

### 2.1.1 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. ツールチェーン(RISC-V Cross GCC)を選択します。
4. 「Next」をクリックします。

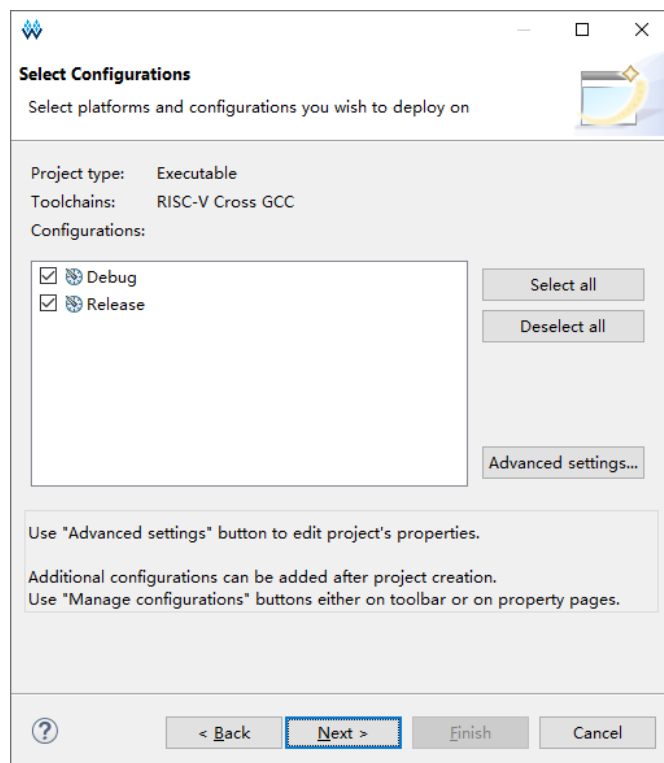
図 2-1 プロジェクト・タイプとツールチェーンの選択



## 2.1.2 プラットフォームと構成の選択

プラットフォームと構成(Debug と Release を含む)を選択して、「Next」をクリックします(図 2-2)。

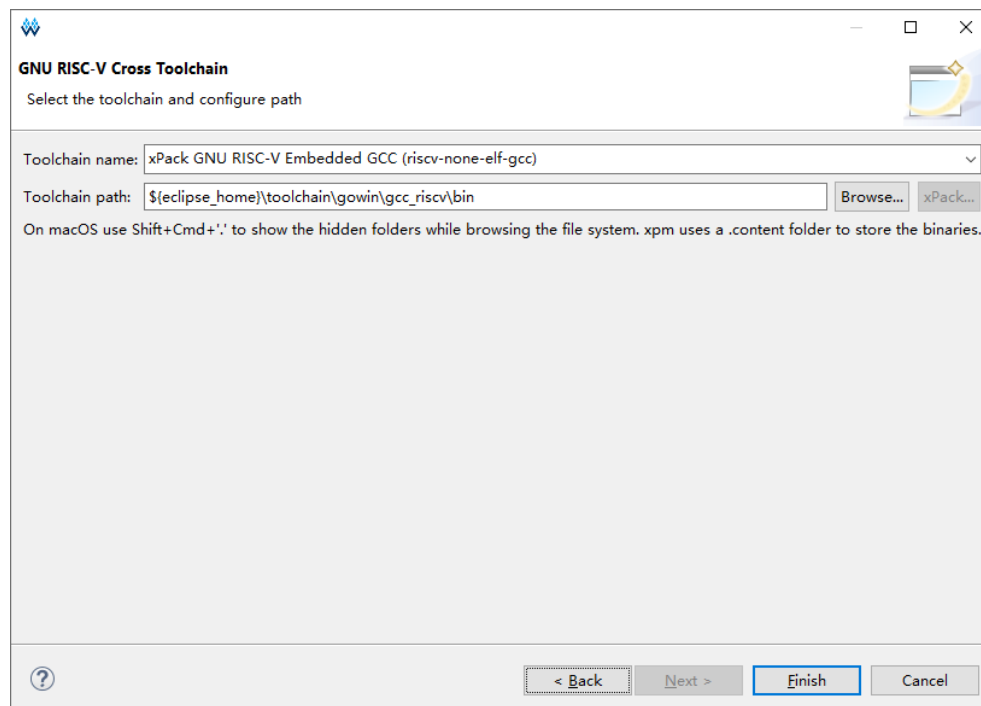
図 2-2 プラットフォームの選択



## 2.1.3 ソフトウェア・ツールチェーンおよびそのパスの選択

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。例えば、`riscv-none-elf-gcc` とそのパスは図 2-3 に示すとおりです。

図 2-3 ツールチェーンおよびそのパスの選択



### 2.1.4 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、ソフトウェア設計を開始します。

例えば、SDK 内のリファレンス・デザインのプロジェクトの構造とコードは以下のようになります。

- **bsp** : ペリフェラル・ドライバ関数の定義
  - **simpleuart.h**  
simpleuart.c : ペリフェラル Simple UART ドライバ関数の定義
  - **wbgpio.h**  
wbgpio.c : ペリフェラル GPIO ドライバ関数の定義
  - **wbi2c.h**  
wbi2c.c : ペリフェラル I2C ドライバ関数の定義
  - **wbspi.h**  
wbspi.c : ペリフェラル SPI ドライバ関数の定義
  - **wbspiflash.h**  
wbspiflash.c : ペリフェラル SPI-Flash ドライバ関数の定義
  - **wbuart.h**  
wbuart.c : ペリフェラル UART ドライバ関数の定義
- **lib** : システム情報の定義
  - **firmware.h**  
firmware.c : システム情報の定義
  - **printf.c** : printf 関数の再定義

- config.h : 起動および実行方法の構成
- custom\_ops.S : ユーザー命令の定義
- irq.h  
irq.c : 割り込みハンドラ
- loader.c : 起動方法の定義
- main.c : main 関数
- picorv32.h : コアレジスタとアドレスマッピングの定義
- start.S : スタートアッププログラムの定義
- sections\_debug.lds : デバッグ用のリンカー・スクリプト・ファイル
- sections\_xip.lds : 外部 SPI-Flash XIP モードでのプログラム実行用のリンカー・スクリプト・ファイル
- sections.lds : 内部 ITCM でのプログラム実行用のリンカー・スクリプト・ファイル

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」を選択すると、現在のプロジェクトが自動的に更新されます。

## 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。

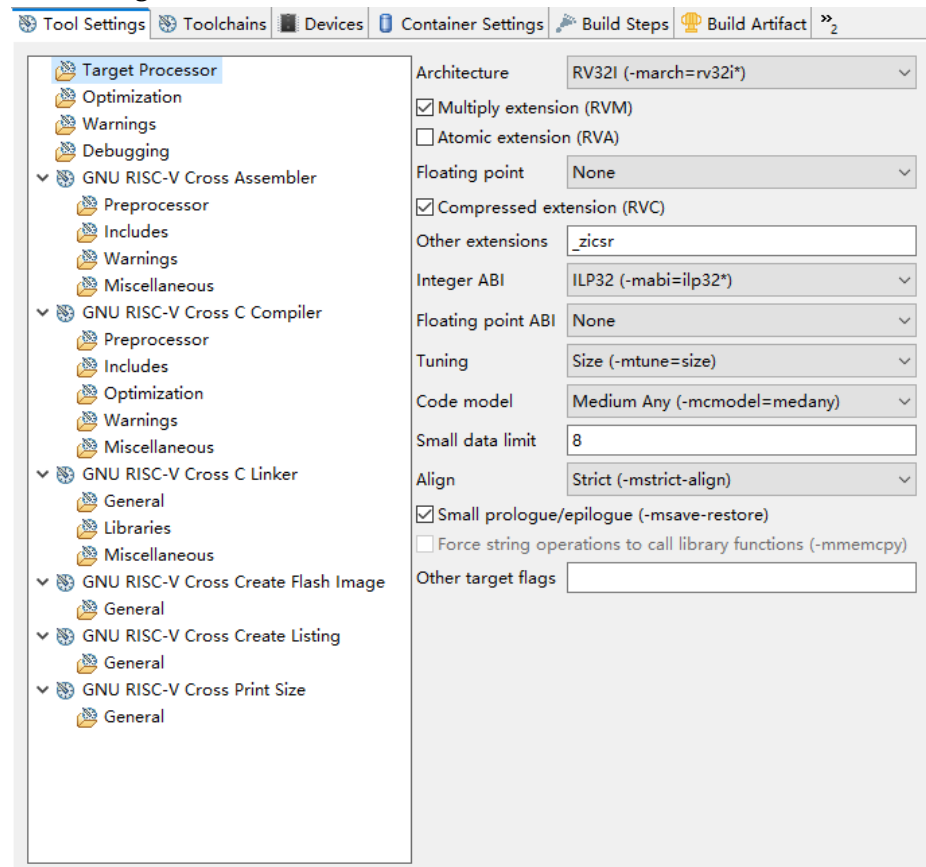
Gowin\_PicoRV32 のオプション :

- Target Processor
- Optimization
- Debugging
- GNU RISC-V Cross C Compiler
  - Includes
- GNU RISC-V Cross C Linker
  - General
- GNU RISC-V Cross Print Size
- GNU RISC-V Cross Create Flash Image
  - General

### 2.2.1 Target Processor

「Target Processor」オプションは RISC-V ターゲットプロセッサを設定するために使用されます(図 2-4)。

## 図 2-4 Target Processor



- **Architecture** : RV32I(-march=rv32i\*)を選択してください。  
Gowin\_PicoRV32 は RISC-V 32 ビット整数命令セットをサポートするため、「RV32I」を選択します。
- **Multiply extension (RVM)** : RV32M 拡張機能を使用したい場合は、Multiply extension(RVM)オプションをチェックしてください。また、Gowin\_PicoRV32 ハードウェア設計の際、IP Core Generator で Gowin PicoRV32 Core の構成オプション「Support RV32M Extends」をチェックする必要があります。そうしないと Gowin\_PicoRV32 が正しく動作できなくなります。
- **Atomic extension (RVA)** : Gowin\_PicoRV32 はアトミック命令拡張をサポートしていないため、このオプションをチェックしないでください。
- **Floating point** : Gowin\_PicoRV32 は浮動小数点命令をサポートしていないため、None を選択します。
- **Compressed extension (RVC)** :  
RV32C 拡張を使用したい場合は、「Compressed extension (RVC)」をチェックしてください。また、Gowin\_PicoRV32 ハードウェア設計の際、IP Core Generator で Gowin PicoRV32 Core の構成オプション「Support RV32C Extends」をチェックする必要があります。そうしないと Gowin\_PicoRV32 が正しく動作できなくなります。

- **Other extensions** : `_zicsr` を追加します。**Zicsr** は、RISC-V 命令セットアーキテクチャの標準拡張機能であり、制御およびステータス・レジスタ(CSR)に対するアクセスと操作に使用されます。
- **Integer ABI** : RISC-V のターゲットプラットフォームでサポートされる ABI 関数の呼び出し規則を設定します。**Gowin\_PicoRV32** は 32 ビットの RISC-V アーキテクチャのプロセッサプラットフォームであり、ハードウェア浮動小数点命令をサポートしていないため、**ILP32** (`-mabi=ilp32*`)を選択します。32 ビットの RISC-V アーキテクチャのプロセッサのデータ型の幅を表 2-1 に示します。

表 2-1 32 ビットの RISC-V アーキテクチャのプロセッサのデータ型の幅

| C 言語のデータ型   | 32 ビットの RISC-V アーキテクチャのデータ型の幅<br>(単位: バイト) |
|-------------|--------------------------------------------|
| char        | 1                                          |
| short       | 2                                          |
| int         | 4                                          |
| long        | 4                                          |
| long long   | 8                                          |
| void *      | 4                                          |
| float       | 4                                          |
| double      | 8                                          |
| long double | 16                                         |

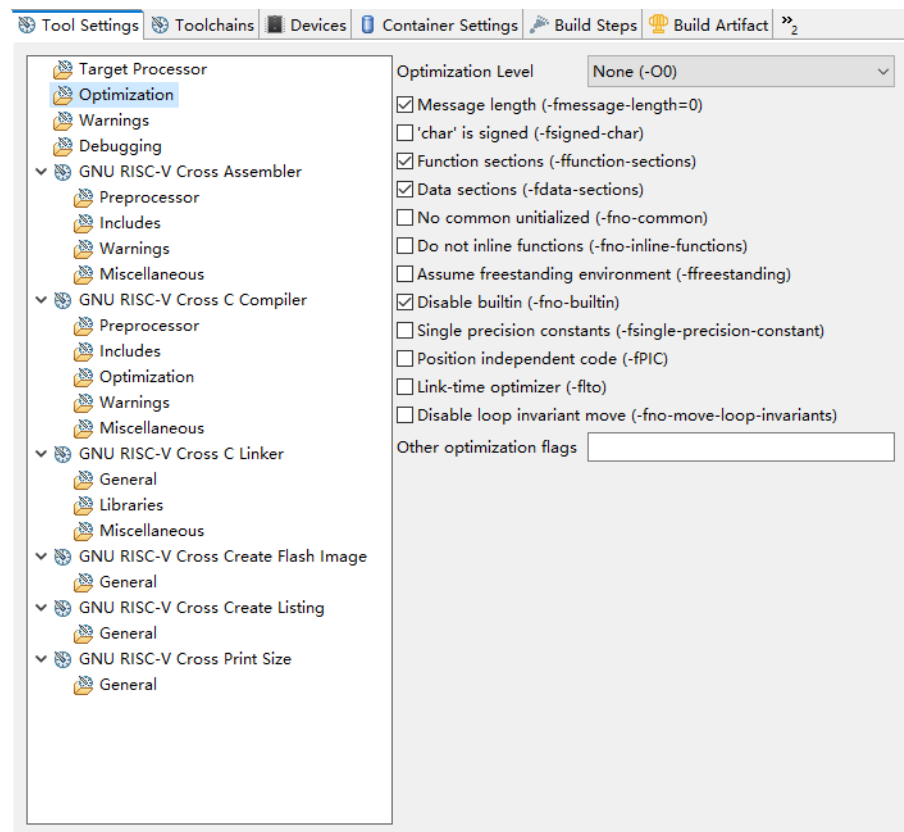
- **Tuning** : GCC がコードパフォーマンスを調整するターゲットプロセッサの名前を指定します。**Gowin\_PicoRV32** がこのオプションをサポートしないため、**Size**(`-mtune=size`)を選択します。
- **Code model** : プログラムのアドレス範囲を指定するパラメータ `-mcmodel` を構成します。**Medium Any** (`-mcmodel=medany`)を選択します。(`-mcmodel = medany`)の場合、プログラムのアドレス範囲が任意の 4GB のスペースで、アドレス空間は事前に決定されていません。
- **Small data limit** : ここでは、スモールデータ領域に配置できるグローバル変数および静的変数の最大サイズ(バイト単位)を指定する、パラメータ `-msmall-data-limit` を構成します。このパラメータは 8 に構成されます。
- **Align** : アンアラインド・メモリアクセスを回避するかどうかを設定します。**Gowin\_PicoRV32** が高速のアンアラインド・メモリアクセスをサポートしないため、**Strict**(`-mstrict-align`)を選択することをお勧めします。
- **Small prologue/epilogue**(`-msave-restore`) : このオプションをチェックすると、比較的遅い最小サイズの開始コードおよびリターンコードのライブラリ関数が呼び出されます。デフォルトでは、高速のインラインコードが使用されます。

- **Other target flags** オプションでは、次の項目を手動で追加できます。
  - **Allow use of PLTs(-mplt)** : この項目を追加すると、**PLT** を使用して割り込み制御コードが生成されます。
  - **Integer divide instructions(-mdiv)** : この項目を追加すると、整数除算ハードウェア命令が使用されます。そうするには、**RV32M** 命令セット拡張をサポートするプロセッサが必要です。  
**Gowin\_PicoRV32** が **RV32M** 命令セット拡張をサポートしているため、「**Multiply extension(RVM)**」オプションを構成することができます。また、**Gowin\_PicoRV32** ハードウェア設計の際、**IP Core Generator** で **Gowin PicoRV32 Core** の構成オプション「**Support RV32M Extends**」をチェックする必要があります。
  - **-mpreferred-stack-boundary=num**  
スタック境界は、**2num** バイト境界に揃えるように設定します。デフォルトは **24**(つまり **16** バイトまたは **128** ビット)アライメントです。この項目を追加すると、すべてのモジュール(ライブラリ、システムライブラリ、開始モジュールを含む)を構築するときこの項目を使用する必要があるようになります。
  - **-mexplicit-relocs / -mno-explicit-relocs**  
記号アドレスを処理する場合、リダイレクト演算子を使用するかどうかを決定します。または、最適化を制限する可能性のあるアセンブリマクロを使用します。
  - **-mrelax**
  - **-mno-relax**  
リンカー・リラクゼーションを使用して、記号アドレスの実装に必要な命令の数を減らします。デフォルトでは、リンカー・リラクゼーションを使用します。
  - **-memit-attribute / -mno-emit-attribute**  
**RISC-V** 属性情報を **ELF** オブジェクトにエクスポートするかどうかを決定します。この機能には、**2.32** バージョンの **binutils** が必要です。
  - **-malign-data=type**  
**GCC** が配列、構造体、共用体などのタイプの変数と定数をどのように調整するかを制御します。サポートされているタイプは「**xlen**」および「**natural**」です。その中で、「**xlen**」の場合はアライメント値としてレジスタ幅を使用し、「**natural**」の場合はナチュラルアライメントを使用します。デフォルトでは「**xlen**」です。

## 2.2.2 Optimization

「**Optimization**」オプションは、コンパイルの最適化レベルを設定するために使用されます(図 2-5)。

## 図 2-5 Optimization



- **Optimization Level:** 最適化レベルの設定は、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。オプションには、**-O0**、**-O1**、**-O2**、**-O3**、**-Os**、**-Ofast**、**-Og**、**-Oz** があります。**-O0/-O1/-O2/-O3** の最適化レベルは順番に増加します。
  - **-O0** : 最適化しません。
  - **-Os** : **-O2** に基づいて、コードのサイズを最適化するために、コードのサイズを増加させるオプションをオフにします。
  - **-Og** : **-O0** に基づいて、デバッグに適用するいくつかのコンパイル・オプションを追加します。
- **Message length (-fmessage-length=n) :**  
console ウィンドウに 1 行あたり n 文字の形式でエラーメッセージを表示します。0 に設定すると、改行機能がオフになり、エラーメッセージが 1 行で表示されます。デフォルトは **-fmessage-length = 0** です。このオプションをチェックすることをお勧めします。
- **'char' is signed (-fsigned-char) :**  
char 型データを符号付き数に設定します。
- **Function sections (-ffunction-sections)/Data sections (-fddata-sections)**
  - ターゲットが任意のセグメンテーションをサポートしている場合、

各関数またはデータ項目が、関数名またはデータ項目名を出力セグメントの名前として使用した、出力ファイル内の個別のセグメントを作成するようにします。

- このオプションは、リンカーが命令スペース内の参照の局所を改善する最適化を実行できる場合に有効にできます。
- リンカーのガベージコレクション(**linker -gc-sections** オプション)を併用すると、実行ファイルが最終的に生成されるときに、使用されていない関数セクションとデータ項目セクションが自動的に削除されます。したがって、コンパイルサイズが小さくなります。

**注記：**

このオプションは、大きな効果を得る可能性がある場合にのみチェックしてください。また、「GNU RISC-V Cross C Linker> General」で「Remove unused sections (-Xlinker --gc-sections)」オプションをチェックして、コードのサイズを減らしてください。

- **No common uninitialized (-fno-common)：**

**fno-common** オプションは、コンパイラーがオブジェクトファイルの **BSS** セクションに初期化されていないグローバル変数を配置することを指定します。これにより、リンカーが仮定義をマージできなくなります。したがって、同じ変数が複数のコンパイル単位内で定義されている場合、複数定義エラー(**multiple definitions error**)が発生します。このオプションをチェックしないことをお勧めします。

- **Do not inline functions (-fno-inline-functions)：**

このオプションをチェックした場合、**always\_inline** 属性でマークされた関数以外のインライン関数は展開されません。これは、最適化がオフの場合のデフォルト設定です。また、関数のインライン化を回避するために、**noinline** 属性で単一の関数をマークすることもできます。

- **Assume freestanding environment (-ffreestanding)：**

コンパイラターゲットをアサートする際のフリースタンディング環境です。フリースタンディング環境には標準ライブラリが存在しない場合があります、プログラムの起動が **main** 関数ではない場合があります。例として、OS のカーネルが挙げられます。**-fno-hosted** と同等です。

- **Disable builtin (-fno-builtin)**

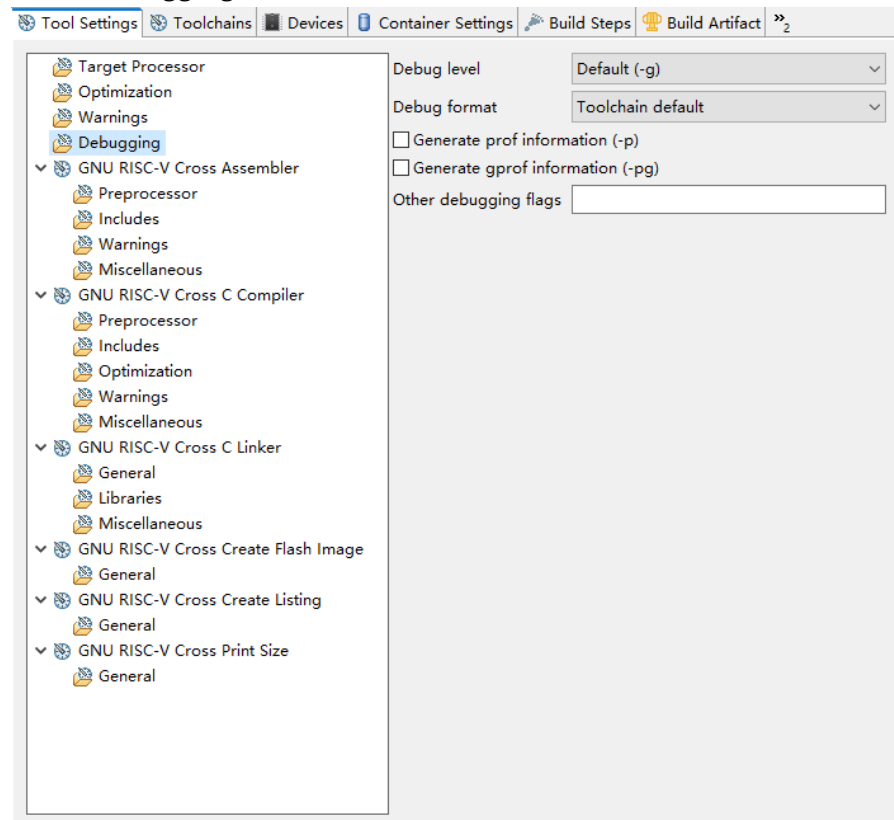
- 「**\_builtin\_**」が前に付いていない組み込み関数は認識されません。具体的には、**-ansi** または **-std** オプション(厳密な ISO C 一貫性のため)を使用するときに組み込み関数ではない関数は認識されません。
- **GCC** は通常、特定の組み込み関数をより効率的に処理するために特別なコードを生成します。たとえば、**alloca** の呼び出しはスタックを直接調整する単一の命令になり、**memcpy** の呼び出しはインラインコピーループになります。通常、このようなコードは小さくて高速ですが、関数はこの方法で呼び出されないため、これらの呼び出しにブレークポイントを設定できず、異なるライブラリをリンクして関数の動作を変更することもできません。

- さらに、関数が組み込み関数として識別されると、GCC は関数に関する情報を利用して、関数の呼び出しについて警告するか、より効率的なコードを生成する場合があります。たとえば、`printf` が組み込みで、`strlen` がグローバルメモリを変更しない場合、GCC は `-Wformat` を使用して `printf` の不正な呼び出しについて警告します。
- **Single precision constants (-fsingle-precision-constant) :**  
浮動小数点定数を単精度データとして処理します。
- **Position independent code (-fPIC) :**  
ターゲットマシンでサポートされている場合は、共有ライブラリでの使用に適した位置独立コード(PIC)を生成します。Gowin\_PicoRV32 が PIC をサポートしないため、このオプションをチェックしないことをお勧めします。
  - **Link-time optimizer (-flto)**  
このオプションは、標準のリンク時オブティマイザを実行します。
  - ソースコードで呼び出されると、GIMPLE(中間表現)が生成され、オブジェクトファイルの特別な ELF セクションに書き込まれます。オブジェクトファイルが互いにリンクされると、すべての関数が同じ翻訳単位の一部であるように ELF ファイルから読み出されてインスタンス化されます。
  - リンク時オブティマイザを使用するには、コンパイル時および最終リンク中に `-flto` および最適化オプションを指定する必要があります。同じリンクのすべてのファイルを同じオプションでコンパイルし、リンク時にこれらのオプションを指定することをお勧めします。
- **Disable loop invariant move (-fno-move-loop-invariants) :**  
RTL ループオブティマイザのループ不変の転送をキャンセルするかどうかを選択します。最適化レベルが `-O1` 以上に設定されている場合(`-Og` を除く)、RTL ループオブティマイザのループ不変の転送は自動的にオンになります。

## 2.2.3 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 2-6)。

### 図 2-6 Debugging



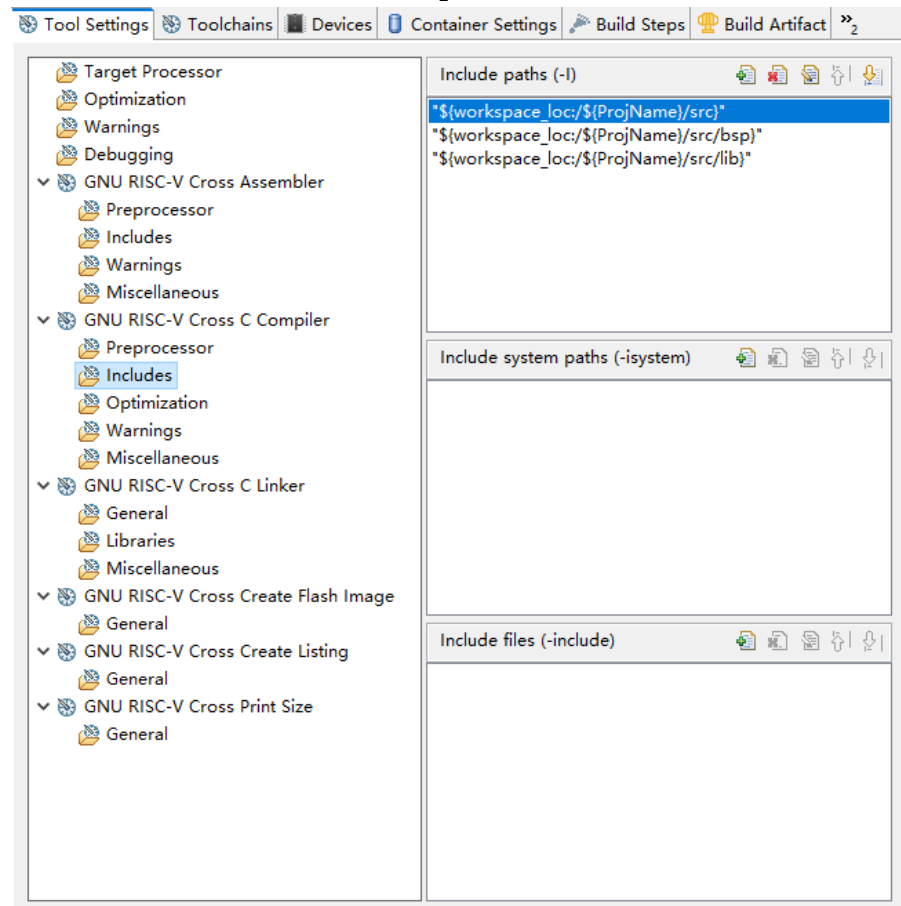
## 2.2.4 GNU RISC-V Cross C Compiler > Includes

「GNU RISC-V 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、相対パスなどを入力できます(図 2-7)。

例えば、SDK 内のリファレンス・デザインの場合、C ヘッダーファイルのパスの設定は以下のようになります：

- "\${workspace\_loc}/\${ProjName}/src"
- "\${workspace\_loc}/\${ProjName}/src/bsp"
- "\${workspace\_loc}/\${ProjName}/src/lib"

図 2-7 GNU RISC-V Cross C Compiler &gt; Includes



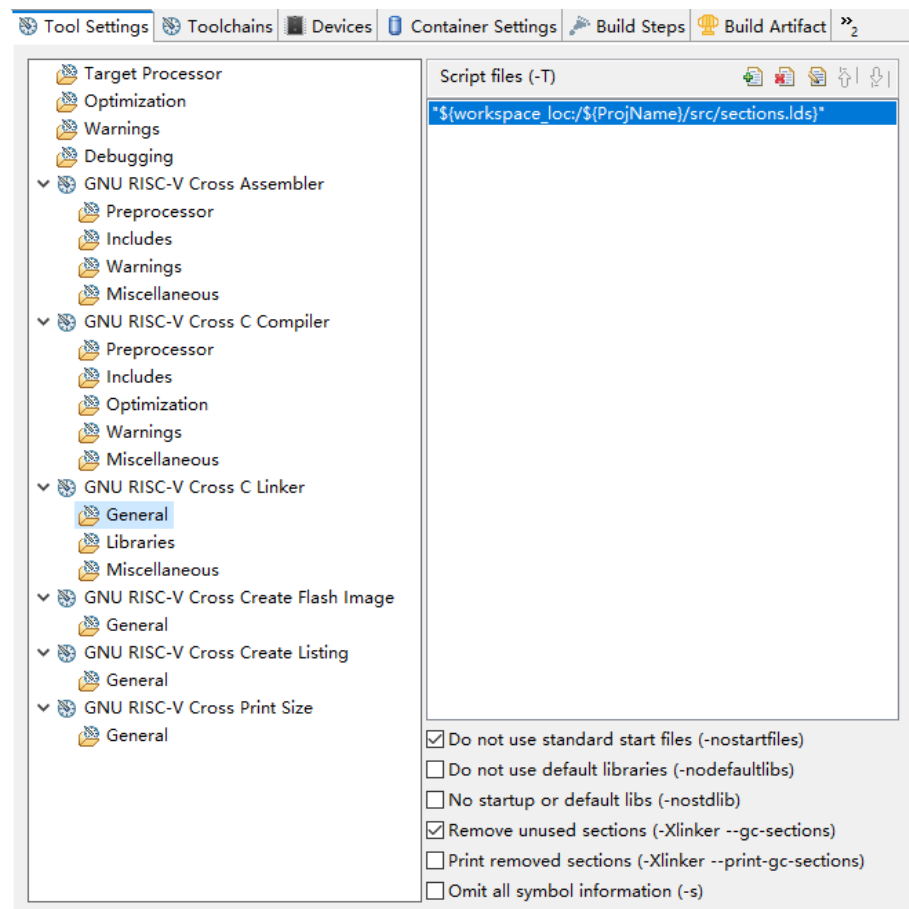
## 2.2.5 GNU RISC-V Cross C Linker

「GNU RISC-V 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、「sections.lds」、「sections\_xip.lds」、または「sections\_debug.lds」を Flash リンカー・スクリプト・ファイルとして構成します(図 2-8)。

Gowin\_PicoRV32 ハードウェア設計際の IP Core Generator 上の Gowin\_PicoRV32 > ITCM の「Boot Mode」について説明します：

- Boot Mode が「MCU boot from external Flash and run in ITCM」または「MCU boot and run in ITCM」の場合、Flash リンカー・スクリプト・ファイルとして sections.lds を選択します。例：  
`"${workspace_loc}/${ProjName}/src/sections.lds"`
- Boot Mode が「MCU boot and run in external Flash」の場合、Flash リンカー・スクリプト・ファイルとして sections\_xip.lds を選択します。例：  
`"${workspace_loc}/${ProjName}/src/sections_xip.lds"`
- Boot Mode が「MCU boot from external Flash and run in ITCM」または「MCU boot and run in ITCM」で、ソフトウェア設計をデバッグする場合、Flash リンカー・スクリプト・ファイルとして sections\_debug.lds を選択します。例：  
`"${workspace_loc}/${ProjName}/src/sections_debug.lds"`

図 2-8 GNU RISC-V Cross C Linker



- **Do not use standard start files (-nostartfiles) :**  
リンク時に標準の開始ファイルを使用しないようにするオプションです。Gowin\_PicoRV32 はカスタム開始ファイルを使用するため、このオプションをチェックしなければなりません。
- **Do not use default libraries (-nodefaultlibs) :**  
リンク時に標準のシステムライブラリを使用しないようにするオプションです。システムライブラリのリンク(-static-libgcc や-shared-libgcc など)を指定するオプションは無視されます。**-nostartfiles** を使用しない限り、通常は標準のシステムスタートアップファイルを使用します。コンパイラは、**memcpy**、**memset**、**memcpy**、および **memmove** の呼び出しを生成する場合があります。
- **No startup or default libs (-nostdlib)**
  - リンク時に標準のシステムスタートアップファイルまたはライブラリを使用しないオプションです。
  - このオプションを有効にする場合、スタートアップファイルがなく、ユーザー指定のライブラリのみがリンカーに渡され、システムライブラリのリンクを指定するオプション(-static-libgcc や-shared-libgcc など)は無視されます。

- このオプションをチェックしないでください。
- **Remove unused sections (-Xlinker -gc-sections)**
  - 呼び出されていないセクションを削除するオプションです。
  - このオプションと、コンパイラの最適化オプション **-ffunction-sections** および **-fdata-sections** オプションを併用することにより、リンク時に呼び出されていない関数と変数を削除し、コードのサイズをさらに削減できます。
- **Print removed sections (-Xlinker -print-gc-sections) :**

「Remove unused sections (-Xlinker -gc-sections)」がチェックされた場合、本オプションをチェックすることで、コンパイル時に削除されたセクションの名前を出力し、削除されたセクションにマークを付けることができます。
- **Omit all symbol informations (-s)**

実行可能ファイルからすべてのシンボルテーブルと再配置情報を削除します。

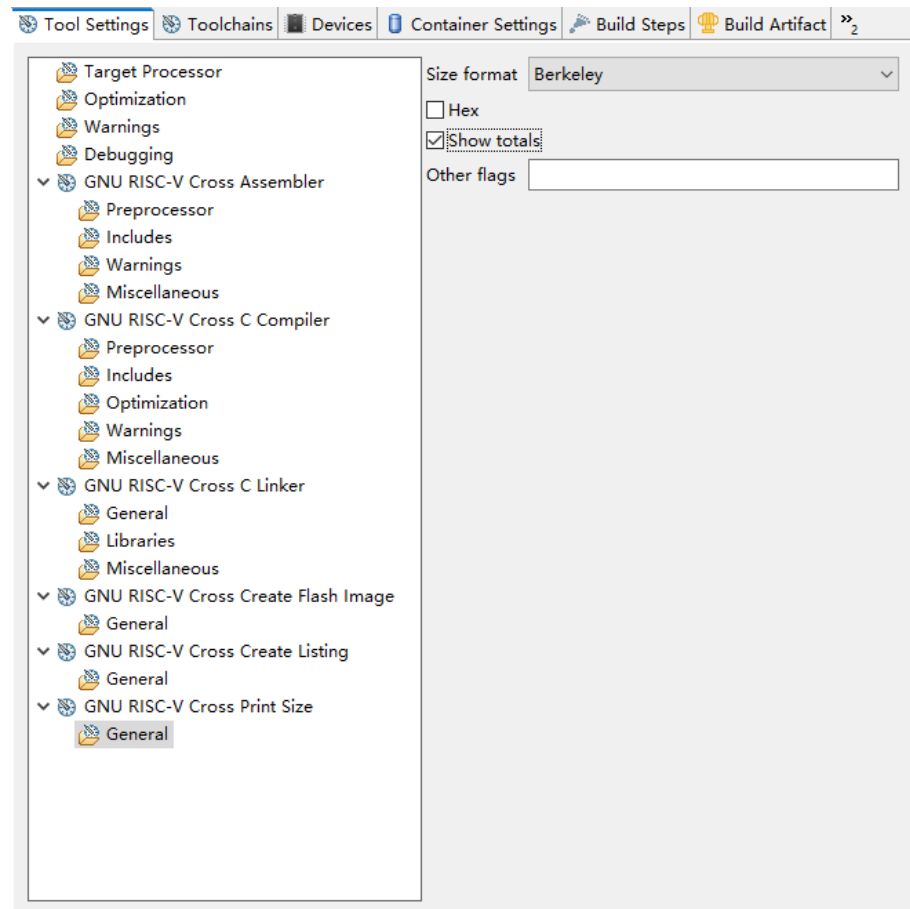
注記 :

デフォルトでは、「Do not use standard start files (-nostartfiles)」と「Remove unused sections (-Xlinker --gc-sections)」の2つのオプションをチェックしてください。

## 2.2.6 GNU RISC-V Cross Print Size

「GNU RISC-V Cross Print Size > General」オプションは、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影響しません (図 2-9)。

### 図 2-9 GNU RISC-V Cross Print Size



## 2.2.7 GNU RISC-V Cross Create Flash Image

「GNU RISC-V Cross Create Flash Image > General」オプションは、RISC-V 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 2-10)。

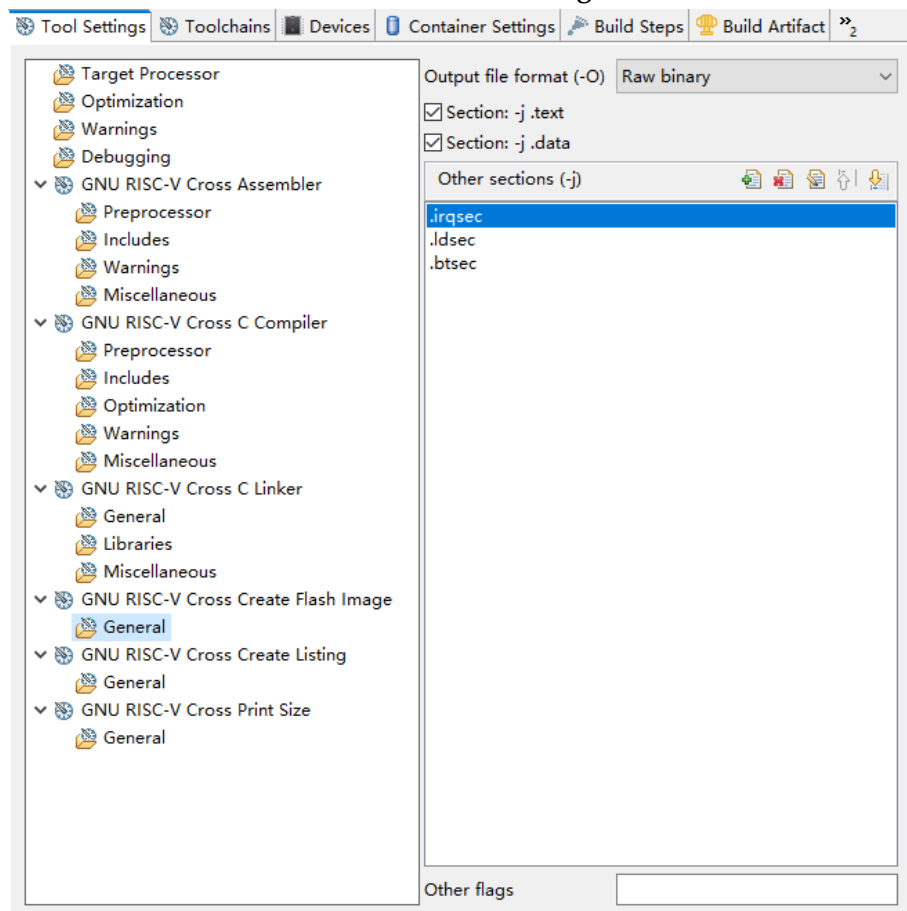
ソフトウェア開発キット内のリファレンス・デザインを例に説明します。

- Output file format (-O)オプションでは、**Raw binary** を選択します。
- Section: -j .text オプションをチェックしてください。
- Section: -j .data オプションをチェックしてください。

ソフトウェア設計にカスタムセクションがあり、関数または変数をカスタムセクションにマップする場合、これらのカスタムセクションを Other sections (-j)に追加する必要があります。例えば：

- .irqsec
- .ldsec
- .btsec

図 2-10 GNU RISC-V Cross Create Flash Image



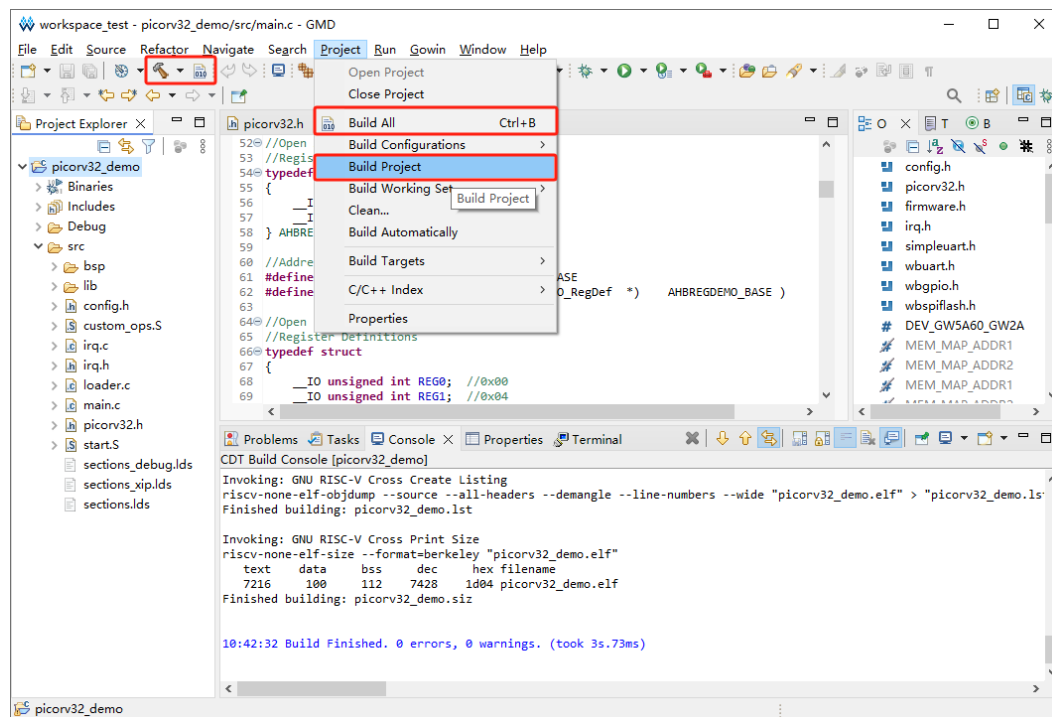
## 2.3 プロジェクトのコンパイル

Gowin\_PicoRV32 ハードウェア設計際の IP Core Generator での Gowin\_PicoRV32 > ITCM > Boot Mode の構成を参照しながら、config.h を変更して BUILD\_MODE のマクロ定義を行います。

- MCU boot and run in ITCM : #define BUILD\_MODE BUILD\_LOAD
- MCU boot from external Flash and run in ITCM : #define BUILD\_MODE BUILD\_BURN
- MCU boot and run in external Flash : #define BUILD\_MODE BUILD\_XIP

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 2-11)。

図 2-11 プロジェクトのコンパイル



## 2.4 プロジェクトのダウンロード

Gowin\_PicoRV32 ソフトウェア設計のダウンロードについては、『Gowin PicoRV32 ソフトウェアダウンロード ユーザーガイド([IPUG913](#))』を参照してください：

## 2.5 プロジェクトのデバッグ

Gowin\_PicoRV32 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、当該設計に問題がある場合は、開発ボードを GWU2X、GWUSB、J-Link、Olimex などのデバッガーに接続して当該プロジェクトをデバッグすることができます。

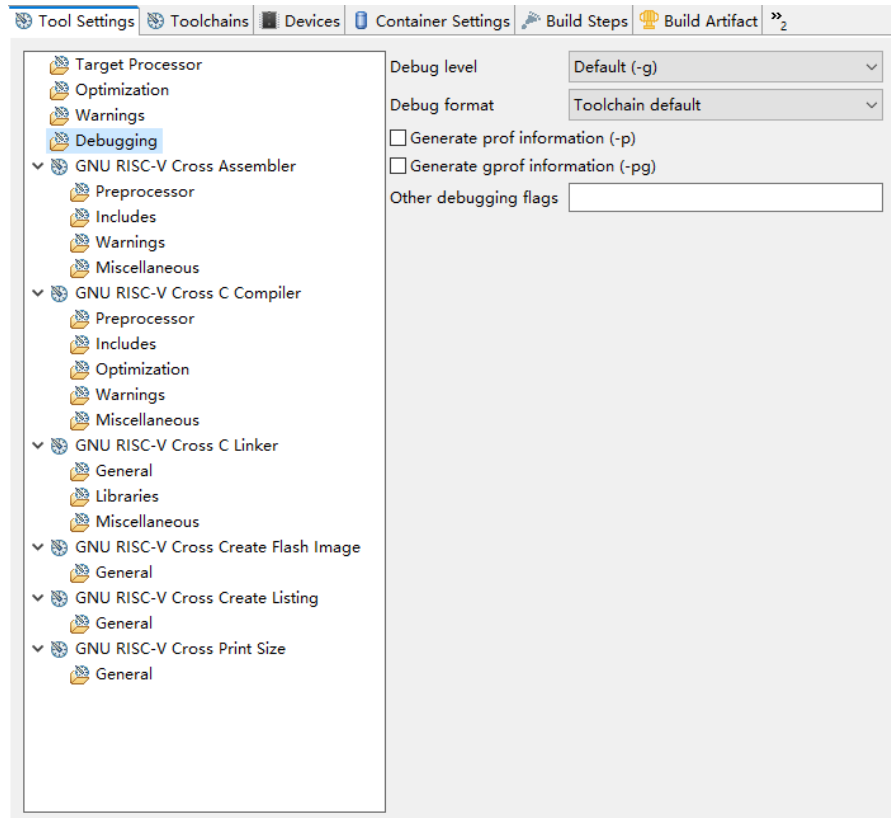
Gowin\_PicoRV32 のデバッグプロセス：

- デバッグレベルの設定
- Flash リンカー・スクリプト・ファイルの設定
- デバッグモードの設定
- デバッガーへの接続
- デバッグの開始

### 2.5.1 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を右クリックして、「Properties > C/C++ Build > Settings > Tool Settings > Debugging > Debug level」オプションを選択してデバッグレベルを構成します(図 2-12)。

図 2-12 デバッグレベルの設定



Debug level には次のオプションがあります：

- None：デバッグ最適化なし
- Minimal (-g1)：最小デバッグレベル
- Default (-g)：デフォルトのデバッグレベル
- Maximum (-g3)：最大デバッグレベル

## 2.5.2 Flash リンカー・スクリプト・ファイルの設定

Gowin\_PicoRV32 ソフトウェア設計をデバッグする場合は、デバッグ用の Flash リンカー・スクリプト・ファイル「**sections\_debug.lds**」を使用してください。Flash リンカー・スクリプト・ファイルの設定方法については、[2.2.5 GNU RISC-V Cross C Linker](#) のを参照してください。

「GNU RISC-V Cross C Linker > General > Script files (-T)」オプションを「`${workspace_loc}/${ProjName}/src/sections_debug.lds`」に変更します。

Flash リンカー・スクリプト・ファイルの構成が完了したら、[2.3 プロジェクトのコンパイル](#)を参照して、プロジェクトを再度コンパイルしてください。

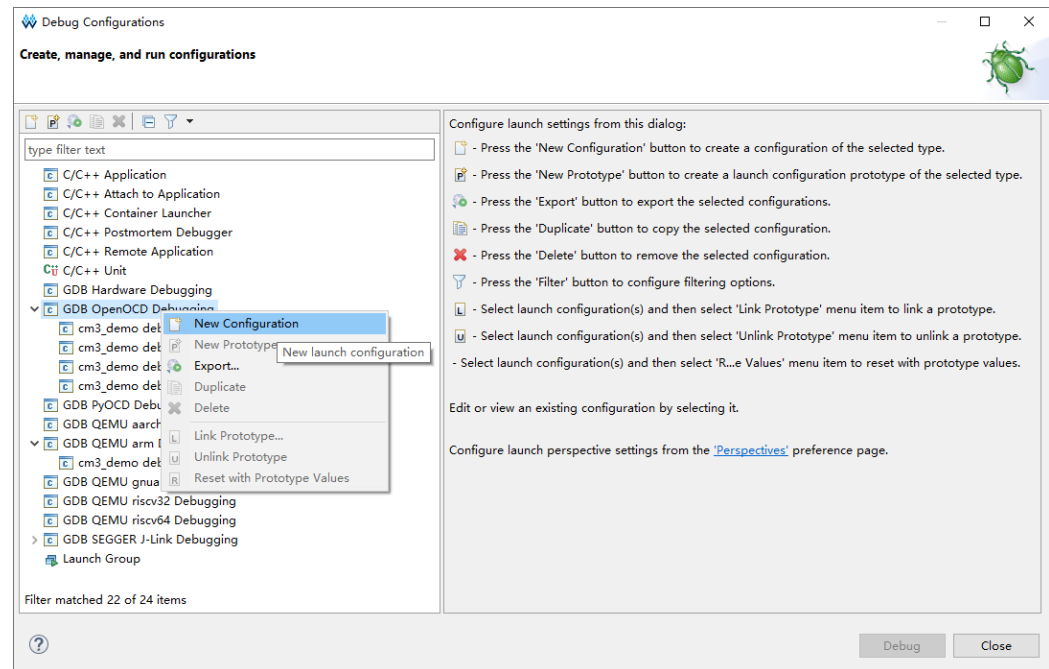
## 2.5.3 デバッグモードの設定

メニューバー > 「Run > Debug Configurations…」 オプションを選択

します。Gowin\_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。

例えば、GDB OpenOCD Debugging デバッグモード選択します。「Debug Configurations」、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 2-13)。

図 2-13 デバッグ・ビューの作成

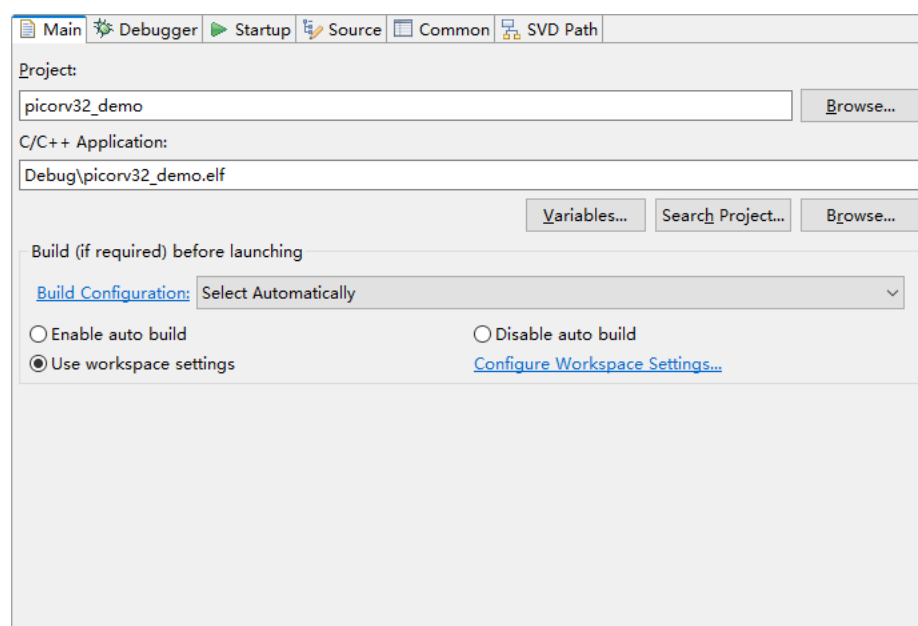


作成されたデバッグ・ビューには、Main、Debugger、Startup などのタブがあります。

### Main タブの設定

「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-)。

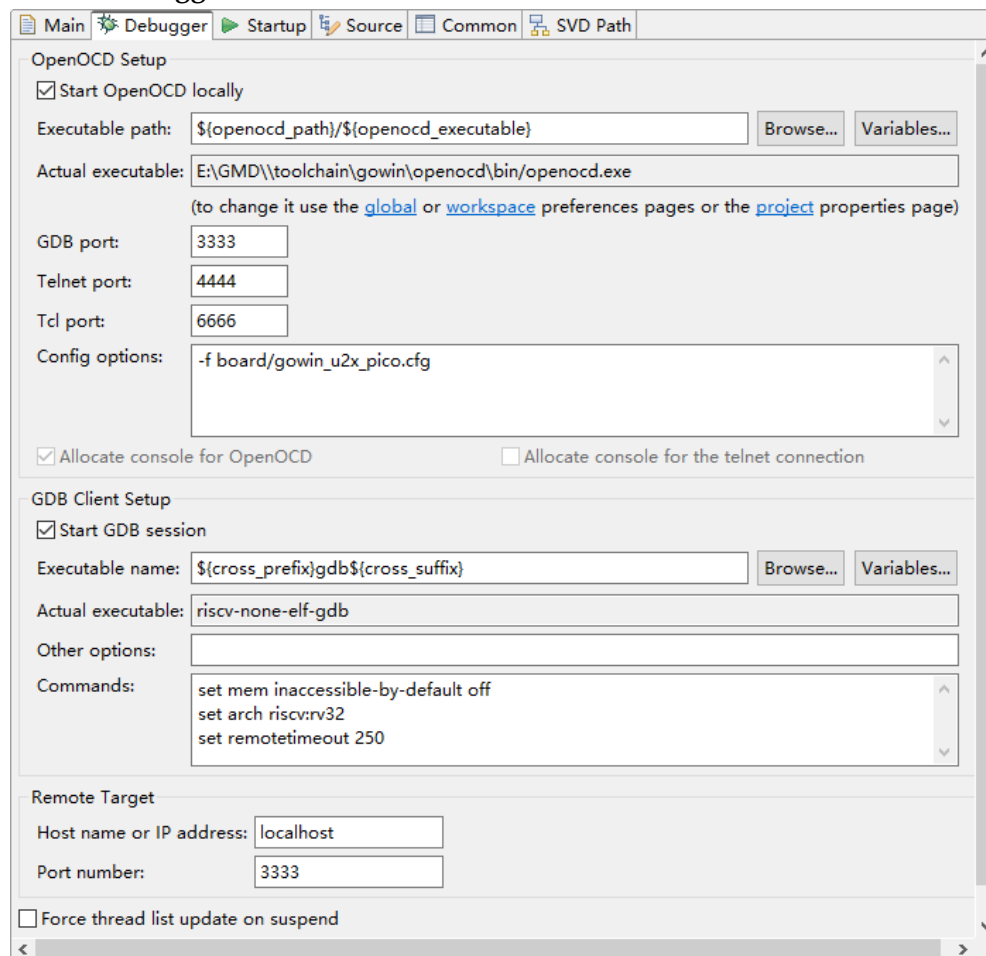
図 2-14 Main タブの設定



### Debugger タブの設定

「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 2-15)。

図 2-15 Debugger タブの設定

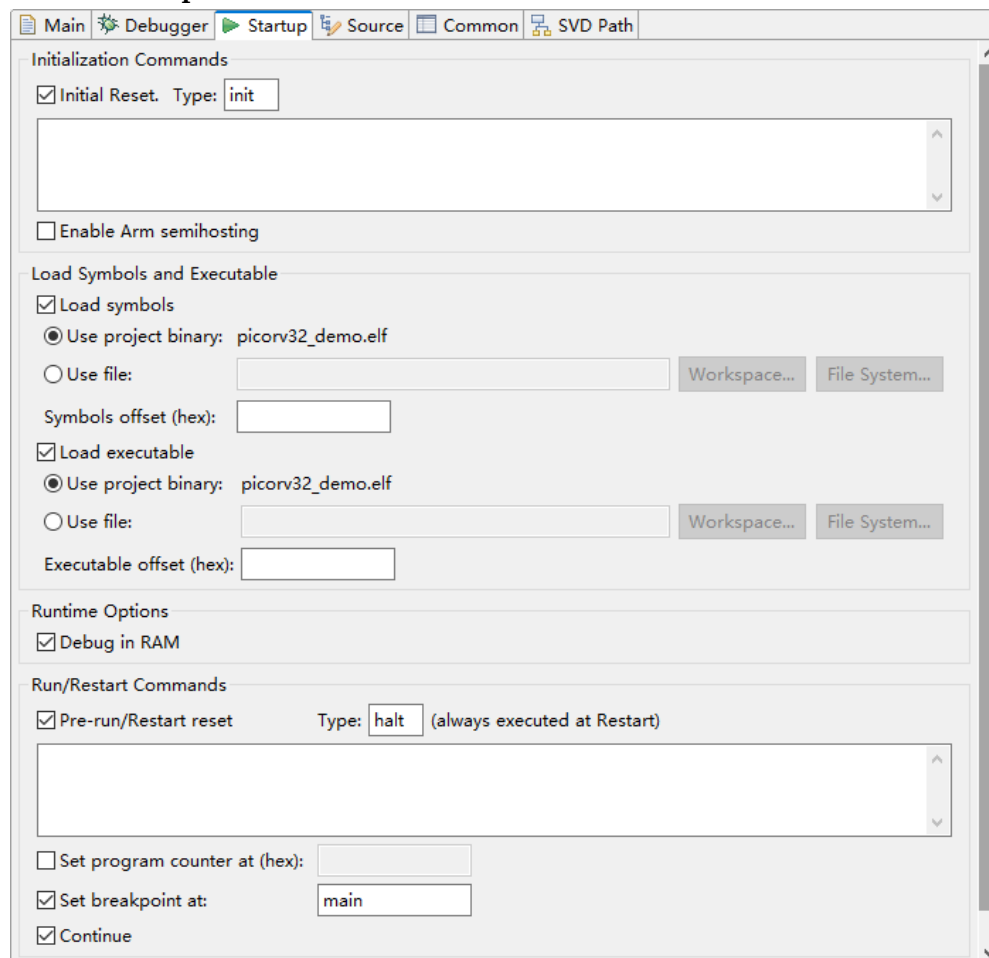


- **Config options** : デバッガーの構成ファイルを指定します。例えば、GWU2X の場合、「-f board/gowin\_u2x\_pico.cfg」を追加します。その他の一般的なデバッガーには、GWUSB (gowin\_ftdi\_dual\_pico.cfg または gowin\_ftdi\_single\_pico.cfg)、J-Link (WinUSB ドライバーを使用する場合は gowin\_jlink\_pico.cfg)、Olimex (gowin\_olimex\_pico.cfg) などがあります。
- **Commands**
  - set mem inaccessible-by-default off
  - set arch riscv:rv32(RISC-V アーキテクチャを指定)
  - set remotetimeout 250(タイムアウトを防ぐ)

### Startup タブの設定

「Startup」タブは、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 2-16)。

図 2-16 Startup タブの設定



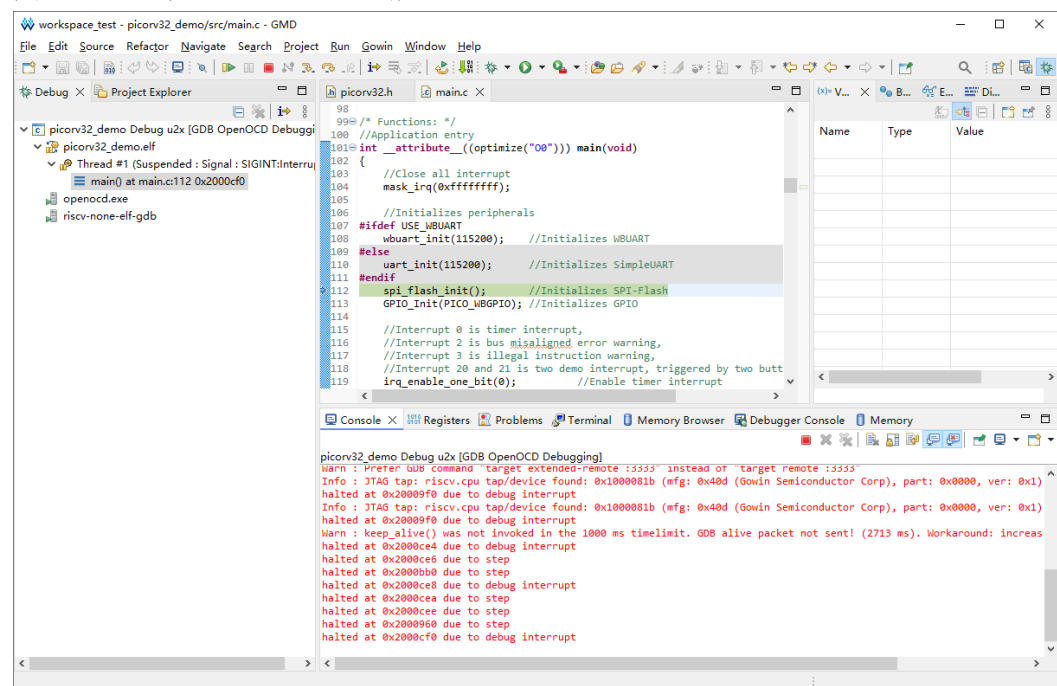
- Initialization Commands
  - Initial Reset オプションをチェックし、Type を「init」に設定します。これにより、デバッグの際に自動的に初期化が実行されます。
  - Gowin\_PicoRV32 が Enable ARM semihosting オプションをサポートしないため、このオプションをチェックしないでください。
- Load symbols と Load executable
  - Load symbols オプションをチェックして「Use project binary」を選択します。
  - Load executable オプションをチェックして「Use project binary」を選択します。
- Runtime Options : 「Debug in RAM」オプションをチェックしてください。これにより、デバッグの際にソフトウェアの実行ファイルが読み出され、命令メモリ ITCM に書き込まれます。
- Run/Restart Commands
  - Pre-run/Restart reset オプションをチェックし、Type を「halt」に設定します。これにより、デバッグの実行時に自動的に一時停止できます。

- **set breakpoint at** オプションをチェックし、「main」と入力します。これにより、デバッグの実行時に、メイン関数の最初のステートメントの位置にブレークポイントを自動的に設定できます。
- **Continue** オプションをチェックしてください。これにより、デバッグの実行時に、ブレークポイントの位置まで自動的に実行して一時停止できます。

## 2.5.4 デバッグの開始

例えば、GWU2X デバッガーに接続します。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 2-17)。

図 2-17 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。

### 割り込みハンドラーのデバッグ

**Gowin\_PicoRV32** は、領域が最適化された **RISC-V** 命令セットアーキテクチャプロセッサであり、その割り込み制御システムはシンプルで使いやすく、ハードウェア割り込みネストと割り込み優先度の設定をサポートしていません。

そのため、デバッグ実行中にプログラムが一時停止状態(シングルステップやブレークポイントによる一時停止)の場合、**MCU** は外部割り込み要求に応答できません。

外部割り込みハンドラーのオンラインデバッグを行う必要がある場合は、割り込みハンドラーの該当箇所にブレークポイントを設定する必要があります。

ツールバーの「Resume」()をクリックして、プログラムを連続実行状態にします。ブレークポイントが検出されない場合は、連続実行状態が維持されます。この状態で、外部デバイスが割り込み要求を出すと、**MCU** は割り込みハンドラーをブレークポイントまで実行し、一時停止状態になります。このとき、割り込みハンドラーでシングルステップデバッグ、ブレークポイントデバッグなどを実行できます。

# 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¥picorv32\_demo

…¥ref\_design¥MCU\_RefDesign¥picorv32\_qemu\_demo

