Gowin PicoRV32 IDE Software

Reference Manual

IPUG910-1.6E, 09/26/2025

Disclaimer

GOWINSEMI assumes no liability and provides no warranty (either expressed or implied) and is not responsible for any damage incurred to your hardware, software, data, or property resulting from usage of the materials or intellectual property except as outlined in the GOWINSEMI Terms and Conditions of Sale. GOWINSEMI may make changes to this document at any time without prior notice. Anyone relying on this documentation should contact GOWINSEMI for the current documentation and errata.

Revision History

Date	Version	Description
01/16/2020	1.0E	Initial version published.
03/06/2020	1.1E	● MCU supports GPIO of Wishbone bus interface; ● MCU supports extension AHB bus interface; ● MCU supports off-chip SPI-Flash download and startup; ● MCU supports the read, write and erasure SPI-Flash; ● MCU supports Hardware Stack Protection and Trap Stack Overflow.
06/01/2020	1.2E	● MCU on-line debug function supported; ● MCU core interrupt handler function enhanced; ● MCU core instruction optimized.
07/16/2021	1.3E	MCU software reference design updated.
01/29/2022	1.4E	● MCU software reference design updated. ● IDE software options configuration optimized. ● IDE software online debug process improved.
08/18/2023	1.5E	Arora Ⅴ FPGA products supported.
07/18/2025	1.5.1E	The link of GMD installation package in "1 GMD Software Installation & Configuration" updated.
09/26/2025	1.6E	● GMD software version information updated. ● Software programming reference designs updated to be compatible with GMD 2025.01.

Contents

List of Figures

List of Tables

1 GMD Software Installation & Configuration

GMD software supports the compiling of Gowin_PicoRV32 software design, compile, download, and debug, etc.

The GMD software package is available at GOWINSEMI website.

For the installation and configuration of the GMD software, as well as the installation and configuration of the software driver for the emulator used to debug Gowin_PicoRV32, see *SUG549, GMD User Guide*.

Note!

It is recommended to use GMD 2025.01.

2 Software Project Template

2.1 Software Project Creation

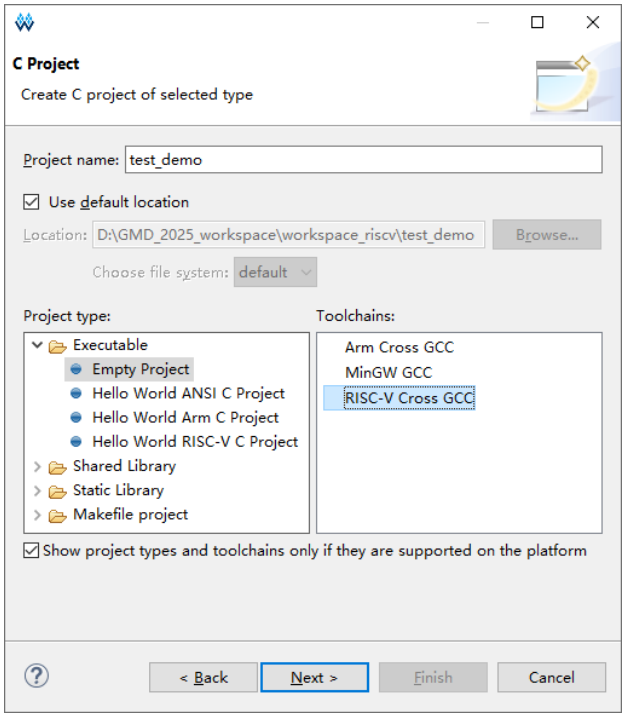
In the toolbar, select "New" ( ) or navigate through the menu "File > New > C/C++ Project > C Managed Build" to create a project.

2.1.1 Select Project Type and Software Toolchain

Create the project name and location, and choose the project type and software toolchain, as shown in Figure 2-1

1. Set the project name and location. For example, select "Use default location" to use the current workspace.
2. Select the project type "Empty Project".
3. Select the toolchain "RISC-V Cross GCC"
4. Click "Next"

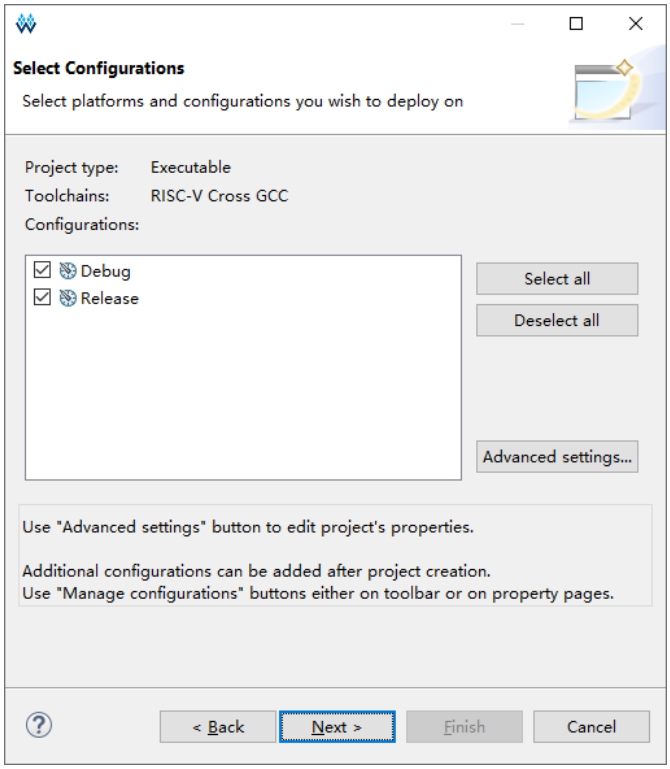
Figure 2-1 Select Project Type and Software Toolchain



2.1.2 Select Platform and Configuration

Choose the platform and configuration, which include "Debug" and "Release", then click "Next", as shown in Figure 2-2.

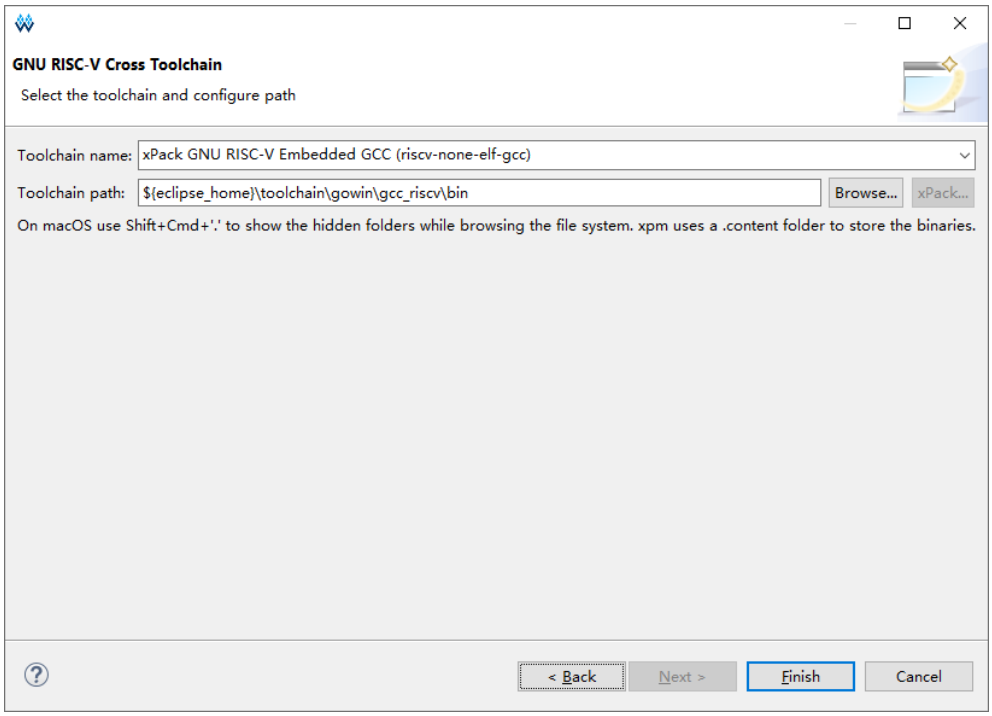
Figure 2-2 Select Platform



2.1.3 Select Toolchain and Path

Select the specific software toolchain and its path. The GMD software already integrates the toolchain and preconfigures the path, so in most cases, the default selection can be used, such as riscv-none-elf-gcc and its corresponding path. Click "Finish", as shown in Figure 2-3.

Figure 2-3 Select Toolchain and Path



2.1.4 Create Project Structure and Files

After the project has been created, select the newly created project in the "Project Explorer" view to establish the project structure and files, and begin software programming and design.

Taking the SDK reference design as an example, the project structure and code definitions are as follows.

- bsp: the definition of peripheral driver function
  - simpleuart.h
    simpleuart.c: the definition of peripheral Simple UART driver function
  - wbgpio.h
    wbgpio.c: the definition of peripheral GPIO driver function
  - wbi2c.h
    wbi2c.c: the definition of peripheral I2C driver function
  - wbspi.h
    wbspi.c: the definition of peripheral SPI driver function
  - wbspiflash.h
    wbspiflash.c: the definition of peripheral SPI-Flash driver function
  - wbuart.h
    wbuart.c: the definition of peripheral UART driver function
- lib: the definition of system information
  - firmware.h
    firmware.c: the definition of system information
  - printf.c: the redefinition of printf function

- Config.h: the start-up and operation mode of user configurations
- custom_ops.S: the definition of user instructions
- irq.h
  irq.c: interrupt handler
- loader.c: the definition of start-up
- main.c: user-defined main function
- Picorv32.h: the definition of kernel register and address mapping
- start.S: the definition of startup program
- sections_debug.lds: Linker script file for debugging
- sections_xip.lds: Linker script file for running program instructions in external SPI-Flash XIP mode
- sections.lds: Linker script file for running program instructions in internal ITCM

If there are code updates during software development, select the current project in the "Project Explorer" view, right-click "Refresh" to update the project.

2.2 Set the Project

In the "Project Explorer" view, select the current project, right-click and choose "Properties > C/C++ Build > Settings > Tool Settings" to set the project.

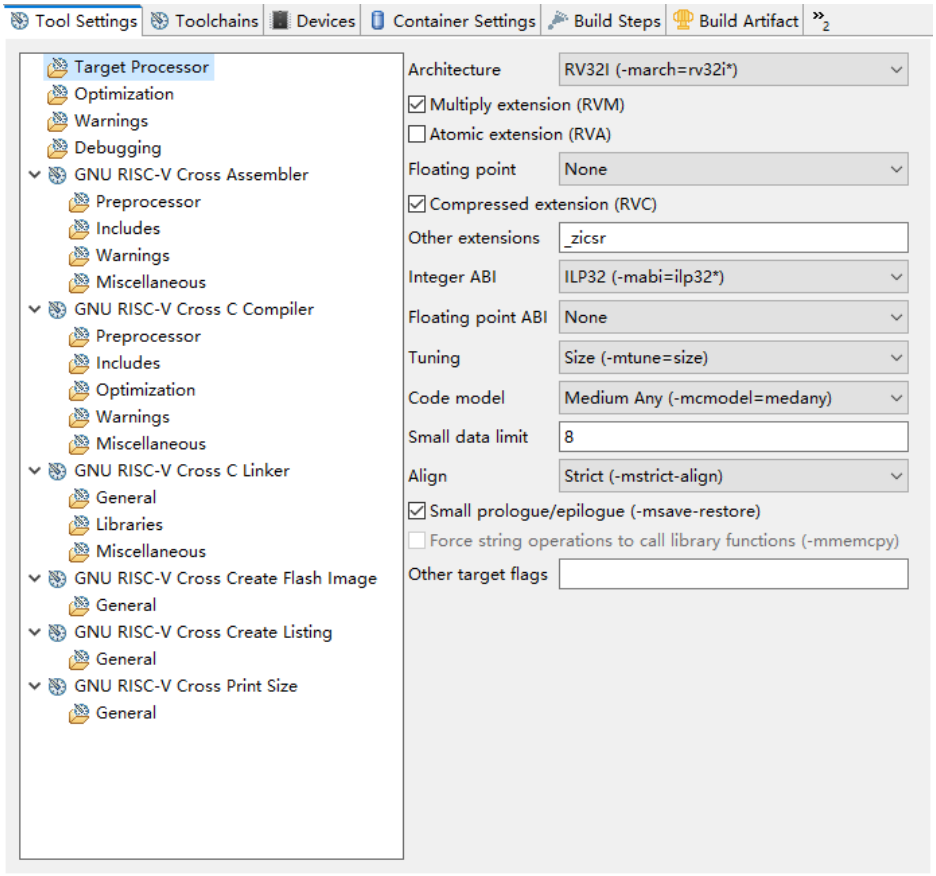
For Gowin_PicoRV32, the following options need to be set:

- 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

The “Target Processor” option is used to configure the RISC-V target processor, as shown in Figure 2-4.

Figure 2-4 Target Processor



- Architecture: Select "RV32I(-march=rv32i*)", Since Gowin_PicoRV32 supports the RISC-V 32-bit integer instruction set, the "RV32I" option must be selected.
- Multiply extension (RVM): If the RV32M extension is required, enable the "Multiply extension (RVM)" option. At the same time, when configuring Gowin_PicoRV32 in the hardware design using the IP Core Generator tool, select the "Support RV32M Extends" option under the CORE configuration of Gowin PicoRV32. Otherwise, it may cause Gowin_PicoRV32 to run incorrectly.
- Atomic extension (RVA): Gowin_PicoRV32 does not support atomic instruction extension, so disable this option.
- Floating point: Gowin_PicoRV32 does not support floating-point extension, so select "None".
- Compressed Extension (RVC): If the RV32C extension is required, enable the "Compressed extension (RVC)" option. At the same time, when configuring Gowin_PicoRV32 in the hardware design using the IP Core Generator tool, select the "Support RV32C Extends" option under the CORE configuration of Gowin PicoRV32. Otherwise, it may cause Gowin_PicoRV32 to run incorrectly.
- Other Extensions: Add _zicsr, a standard extension in the RISC-V instruction set architecture (Zicsr) that enables access to and operations on Control and Status Registers (CSR).

- Integer ABI: Set the calling rule of ABI function supported by RISC-V target platform. Gowin_PicoRV32 is a 32-bit RISC-V architecture processor platform and does not support hardware floating-point instruction, so select "ILP32 (-mabi=ilp32*)". The data type width of 32-bit RISC-V architecture processor is as shown in Table 2-1.

Table 2-1 Data Type Width of 32-bit RISC-V Architecture Processor

C Language Data Types	Data Type Width of 32-bit RISC-V Architecture (Unit: Byte)
char	1
short	2
int	4
long	4
long long	8
void *	4
float	4
double	8
long double	16

- Tuning: Specify the compiling toolchain GCC as the name of target processor for improving code performance. Gowin_PicoRV32 does not support this option, so select "Size(-mtune=size)".
- Code model: Set the parameter "-mcmodel". This parameter specifies the program’s addressing range, and select "Medium Any (-mcmodel=medany)". The "(-mcmodel=medany)" option is used to specify that the program's addressing range can be in any 4GB space. The addressing space is not predetermined, and the application is relatively flexible.
- Small data limit: Set the "-msmall-data-limit" parameter. This parameter specifies the maximum size in bytes of global and static variables that can be placed into small data areas. This parameter is set to 8.
- Align: Set whether to avoid operations that result in unaligned memory access. Gowin_PicoRV32 does not support fast unaligned access, so it is recommended to select Strict(-mstrict-align).
- Small prologue/epilogue(-msave-restore): If this option is selected, library functions are used to generate the smallest possible call prologue and epilogue code, but with slower execution speed. By default, faster inline code is used.
- You can manually add the following configurations in the "Other target flags" option.
  - Allow use of PLTs(-mplt)
    If this option is configured, use PLT to generate interrupt control code.
  - Integer divide instructions(-mdiv)
    If this option is selected, integer division hardware instructions will

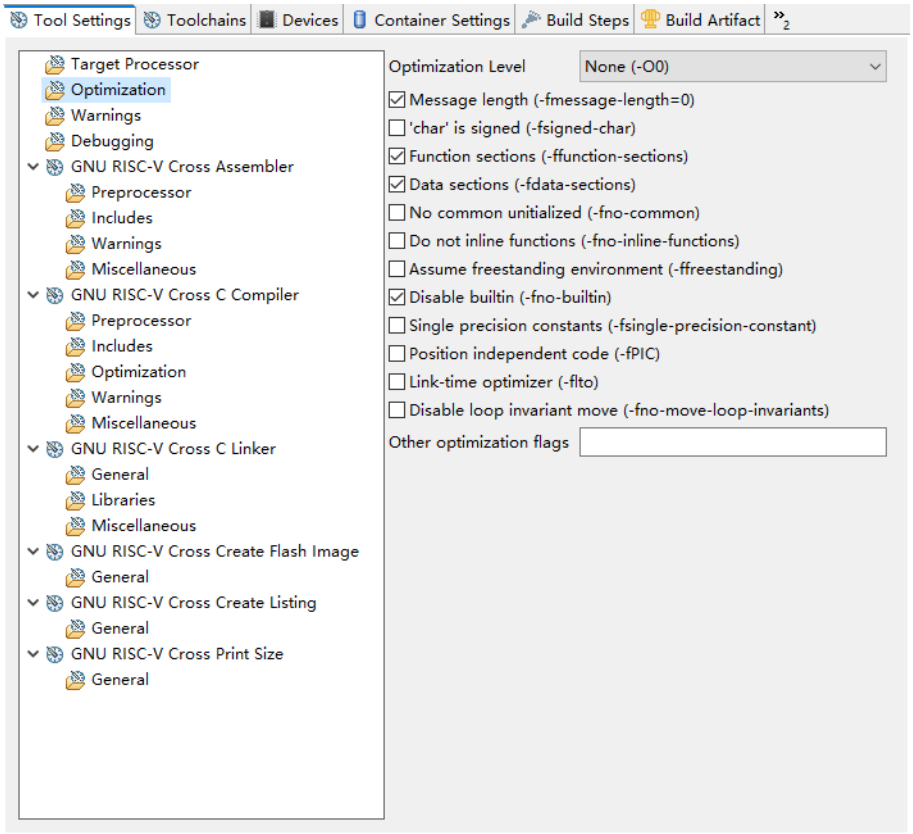
be used, which require processor support for the RV32M instruction set extension. Since Gowin_PicoRV32 supports the RV32M extension, you can enable the "Multiply extension (RVM)" option. At the same time, in the hardware design for Gowin_PicoRV32, configure the CORE options in the IP Core Generator tool and select the "Support RV32M Extends" option.

- -mpreferred-stack-boundary=num: The stack boundary is aligned to 2num bytes. If not specified, the default value is 24, that is, 16 bytes or 128 bits. If this option is configured, it needs to be used when building all modules (including libraries, system libraries, and start modules).
- -mexplicit-relocs / -mno-exlicit-relocs: When dealing with symbolic addresses, use or do not use the assembler redirection operators. Another configuration is to use assembly macros, which can limit optimization.
- -mrelax
- -mno-relax: Use the linker relaxation to reduce the number of instructions required to implement symbolic addresses. The default is to use linker relaxation.
- -memit-attribute / -mno-emit-attribute: Output or no output of RISC-V attribute information to ELF objects. This feature applies to binutils 2.32.
- -malign-data=type: Control how GCC aligns variables and constants of array, struct, union, and so on. The supported type values are "xlen" and "natural". "xlen" uses register widths as alignment value, and "natural" uses natural alignment. The default value is "xlen".

2.2.2 Optimization

The “Optimization” option is used to set the build optimization level, as shown in Figure 2-5.

Figure 2-5 Optimization



- Optimization Level: The optimization level is set using the -O options, which balance build speed, code performance, and code size. The selectable levels include -O0, -O1, -O2, -O3, -Os, -Ofast, -Og, and -Oz, with -O0/-O1/-O2/-O3 providing progressively higher levels of optimization.
  - -O0: No optimization.
  - -Os: Based on -O2, but disables options that increase code size to achieve optimal code size reduction.
  - -Og: Based on -O0, but enables some optimization options that are suitable for debugging.
- Message length (-fmessage-length=n)
  Displays the error message in the console window as n characters per line. If set to 0, the newline function is turned off and an error message is displayed as a line. The default is -fmessage-length=0. It is recommended to select this option.
- ‘char’ is signed (-fsigned-char)
  Set "char" type data as signed number.
- Function sections (-ffunction-sections)/Data sections (-fdata-sections)
  - If the target supports arbitrary segmentation, have each function or data item create a separate segment in the output file, using the name of the function or data item as the name of the output segment.

- Enable this option if the linker can perform the referenced localized optimizations in the improved instruction space.
- When used with linker garbage collection (linker -- gc-sections option), unused function segments and data item segments are automatically deleted during the final generation of the executable file, resulting in a smaller compilation Size.

Note!

This option should be selected only when it provides a significant effect, and it should be used together with selecting “Remove unused sections (-Xlinker --gc-sections)” under "GNU RISC-V Cross C Linker > General" to reduce code size.

- No common unitialized (-fno-common): The fno-common option specifies that the compiler places uninitialized global variables in the BSS segment of the target file. This prevents the linker from merging the tentative definitions, so if the same variables are defined in more than one compilation unit, a multi-definition error occurs. It is recommended to disable this option.
- Do not inline functions (-fno-inline-functions): If you select this option, no inline functions are expanded except those marked with the "always_inline" property. This is the default setting when optimization is turned off. Users can also mark a single function with the noinline property to avoid inlining of the function.
- Assume freestanding environment (-ffreestanding): Assume a freestanding environment during target compiling, where the standard library may not exist and the program launch may not be the main function. One case is the operating system kernel. It is equivalent to -fno-hosted.
- Disable builtin (-fno-builtin)
  - Built-in functions not prefixed with "_builtin_" are not recognized. The affected functions include functions that are not built-in when using "-ansi" or "-std" options (for strict ISO C consistency) because there is no standard ISO meaning.
  - GCC typically generates special code to handle certain built-in functions more efficiently. For example, a call to alloca may become a single instruction that directly adjusts the stack, and a call to "memcpy" may become an inline copy loop. The generated code is usually smaller and faster, but because function calls no longer appear that way, users can't set breakpoints on them or change the behavior of a function by connecting to different libraries.
  - In addition, when a function is identified as a built-in function, GCC may use information about the function to warn of problems calling the function or to generate more efficient code, even if the generated code still contains calls to the function. For example, when "printf" is built in and "strlen" is known not to modify global memory, an error call to "printf" is warned in "-wformat".
- Single precision constants (-fsingle-precision-constant): Floating-point

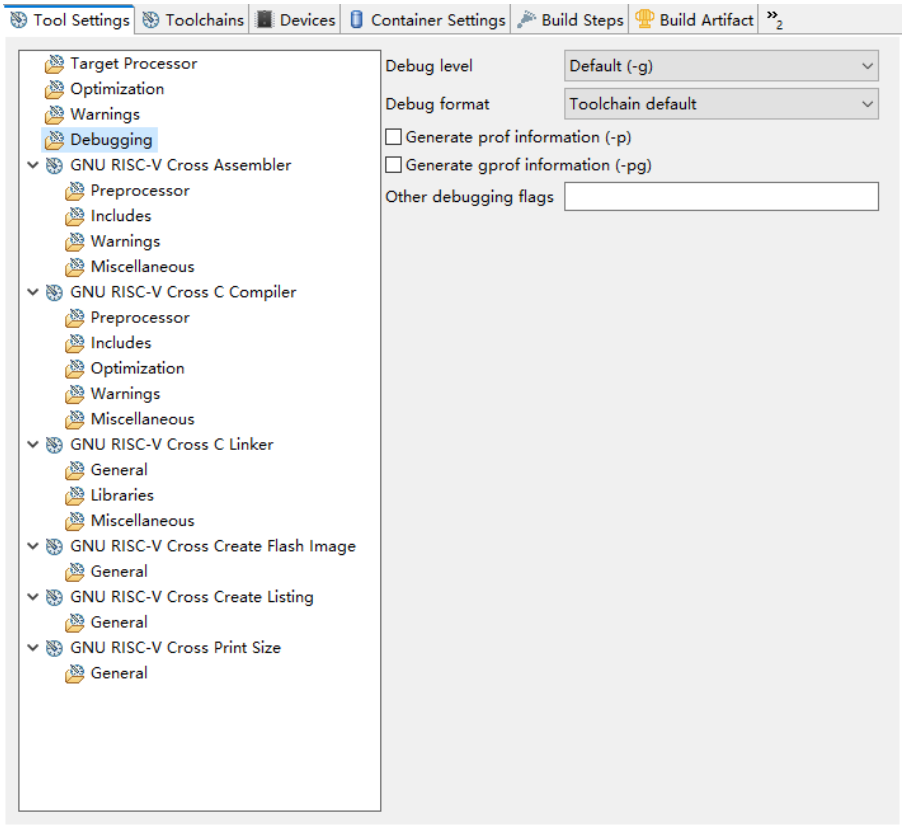
constants are treated as single-precision data.

- Position independent code (-fPIC): If the target side supports, generate position-independent code (PIC) suitable for use in the shared library. Gowin_PicoRV32 does not support PIC, so it is recommended to disable this option.
  - Link-time optimizer (-flto)
    This option runs standard link-time optimizer.
  - When users use a source code to call, generate GIMPLE (one of the internal representations of GCC) and write it to a special ELF section in the object file. When the object files are joined together, all the function bodies are read from the ELF section and instantiated as if they were part of the same translation unit.
  - When you use the link-time optimizer, specify "-flto" and optimization options at compile time and during the final connection. It is recommended to compile all files that participate in the same linking using the same options, and specify these options at link-time.
- Disable loop invariant move (-fno-move-loop-invariants): Select whether to cancel the loop invariant action transfer in the RTL loop optimizer. If the optimization level is set to -O1 or higher (except -Og), the loop invariant action transfer in the RTL loop optimizer will be automatically started.

2.2.3 Debugging

The "Debugging" option is used to set the debugging level. For example, -g, -g1, and -g3 generate debugging information with different levels of detail, as shown in Figure 2-6.

Figure 2-6 Debugging



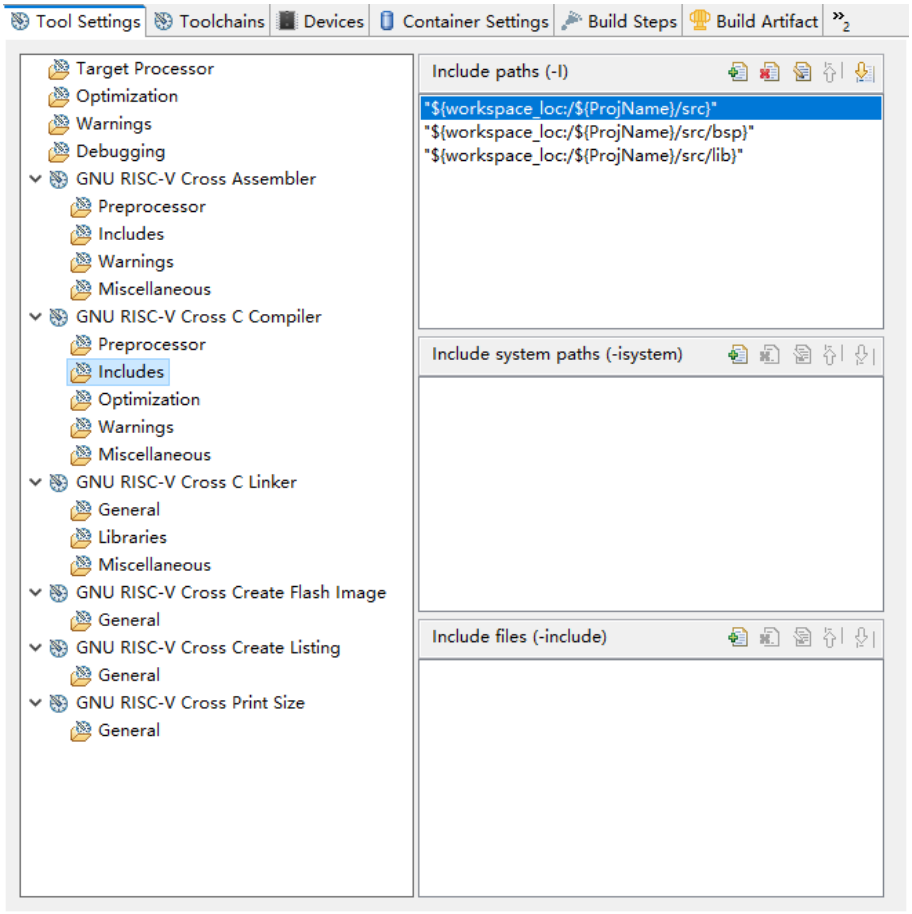
2.2.4 GNU RISC-V Cross C Compiler > Includes

The "GNU RISC-V Cross C Compiler > Includes > Include paths (-I)" option is used to specify the search paths for C header files. Both absolute and relative paths can be used, as shown in Figure 2-7.

Taking the SDK reference design as an example, the C header file search paths are configured as follows:

- "${workspace_loc:/${ProjName}/src}"
- "${workspace_loc:/${ProjName}/src/bsp}"
- "${workspace_loc:/${ProjName}/src/lib}"

Figure 2-7 GNU RISC-V Cross C Compiler > Includes



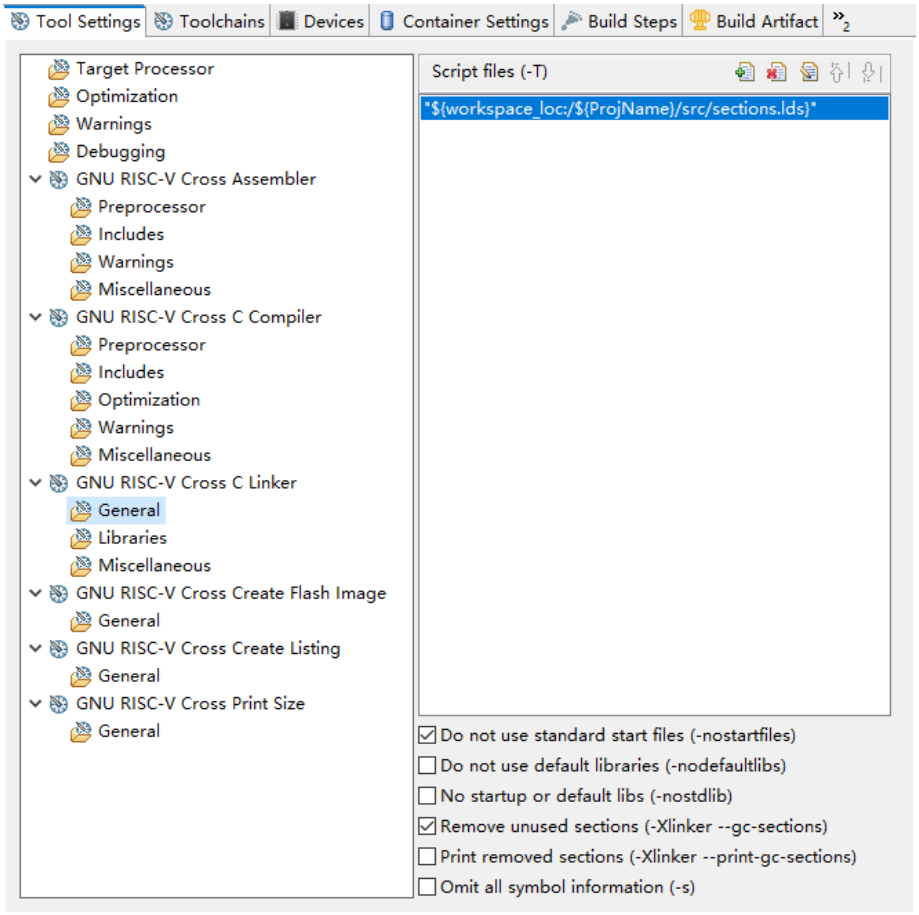
2.2.5 GNU RISC-V Cross C Linker

The "GNU RISC-V Cross C Linker > General > Script files (-T)" option is used to specify the Flash linker script file. You can set sections.lds, sections_xip.lds, or sections_debug.lds as the Flash linker script file for Gowin_PicoRV32, as shown in Figure 2-8.

Referring to the hardware design of Gowin_PicoRV32, the "Boot Mode" option in the ITCM settings of Gowin_PicoRV32 within the IP Core Generator tool is described as follows:

- If Boot Mode is set to "MCU boot from external Flash and run in ITCM" or "MCU boot and run in ITCM", then select "sections.lds" as Flash linker script file, such as "${workspace_loc:/${ProjName}/src/sections.lds}".
- If Boot Mode is set to "MCU boot and run in external Flash", then select "sections_xip.lds" as Flash linker script file, such as "${workspace_loc:/${ProjName}/src/sections_xip.lds}".
- If Boot Mode is set to "MCU boot from external Flash and run in ITCM" or "MCU boot and run in ITCM", and you are debugging the software design, select sections_debug.lds as the Flash linker script file, such as "${workspace_loc:/${ProjName}/src/sections_debug.lds}"

Figure 2-8 GNU RISC-V Cross C Linker



- Do not use standard start files (-nostartfiles): Standard start files are not used when configuring linking. Gowin_PicoRV32 must use custom start files, so this option is required to be selected.
- Do not use default libraries (-nodefaultlibs): Standard system libraries are not used when configuring linking, only the selected libraries are transferred to the linker. The options that specify the system library linker, such as "-static-libgcc" or "-shared-libgcc", are ignored. Normal use of standard startup files, except using "-nostartfiles". The compiler may generate calls to "memcmp", "memset", "memcpy", and "memmove".
- No startup or default libs (-nostdlib)
  - Standard system startup files or libraries are not used when linking.
  - There are no startup files, only user-specified libraries are passed to the linker, and options for specifying system library connections (such as "-static-libgcc" or "-shared-libgcc") are ignored.
  - Please disable this option.
- Remove unused sections (-Xlinker-gc-sections)
  - This option removes segments that are not called.
  - This option works with the "-ffunction-sections" and "-fdata-sections" options set by the compiler optimization to

remove uncalled functions and variables at linking time, further reducing code size.

- Print removed sections (-Xlinker-print-gc-sections): When "Remove unused sections (-Xlinker-gc-sections)" is selected, this option can be selected at the same time, and the name of the deleted section is printed at compile time to mark the deleted section.
- Omit all symbol informations (-s)

  Remove all symbol tables and relocation information from the executable file.

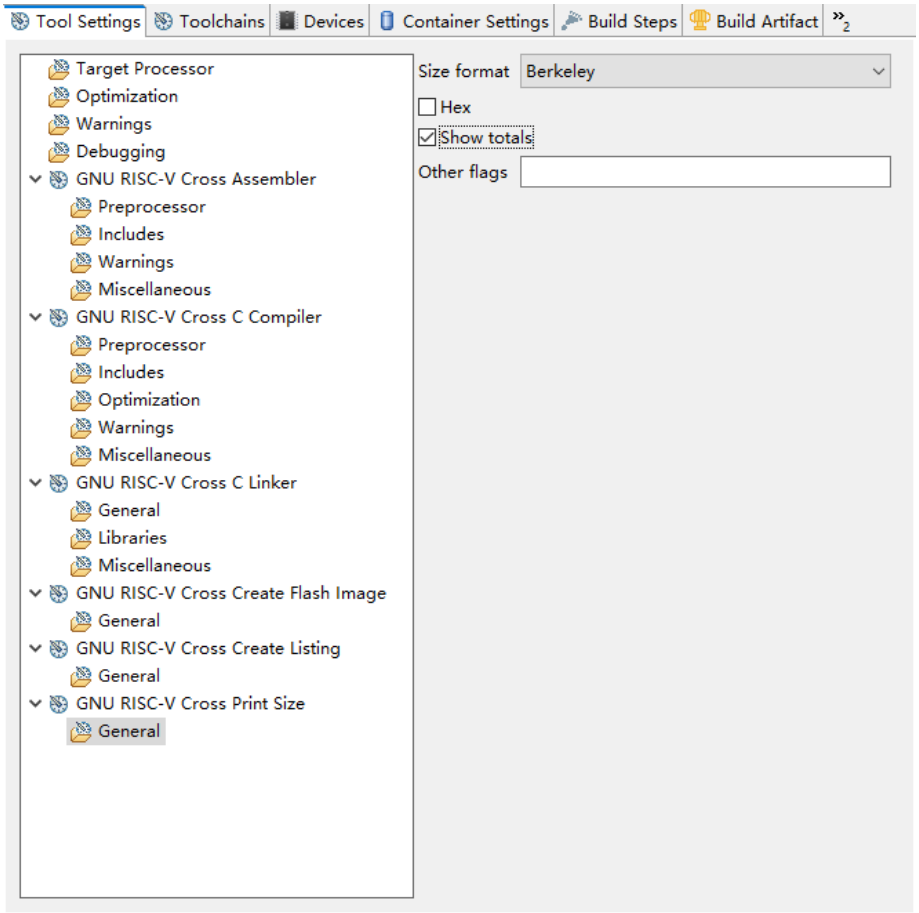
Note!

Select "Do not use standard start files (-nostartfiles)" and "Remove unused sections (-Xlinker --gc-sections)" options by default.

2.2.6 GNU RISC-V Cross Print Size

The "GNU RISC-V Cross Print Size > General" option is used to output the size statistics of each segment (text, data, bss) and the total size of the RISC-V MCU executable file. It supports Berkeley and SysV formats; different formats only affect the output style and do not change the actual segment size calculations, as shown in Figure 2-10.

Figure 2-9 GNU RISC-V Cross Print Size



2.2.7 GNU RISC-V Cross Create Flash Image

The "GNU RISC-V Cross Create Flash Image > General" option is used to set the output format of the RISC-V MCU executable file, as shown in Figure 2-10.

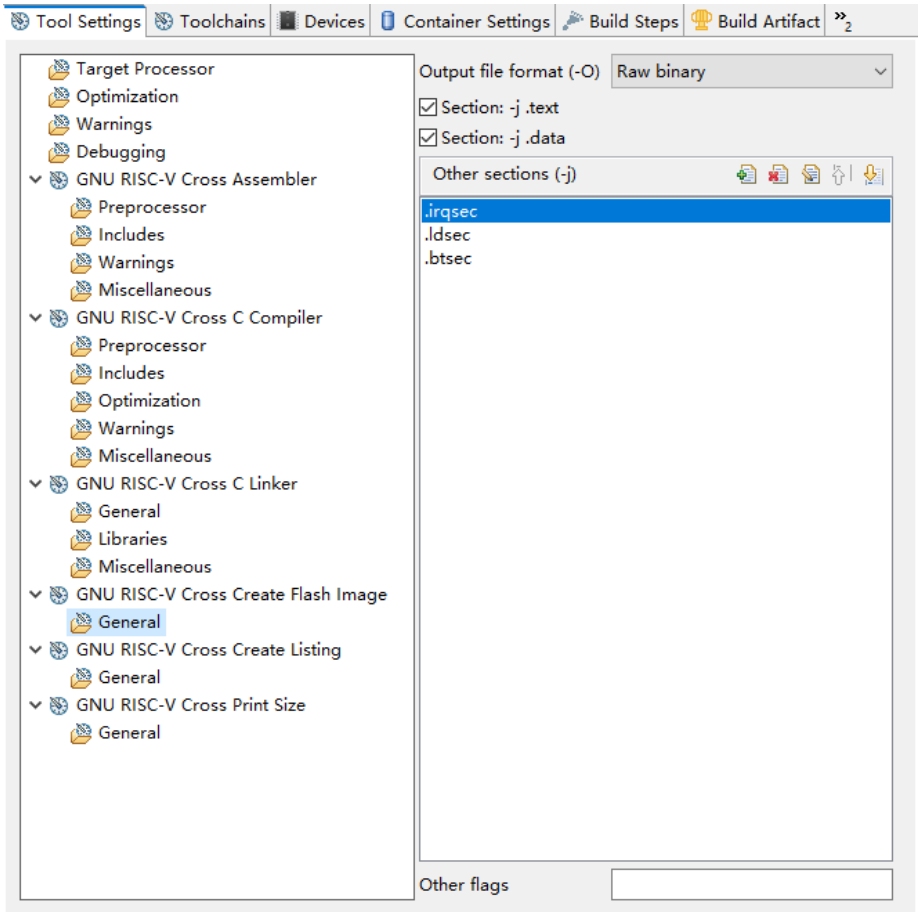
Taking the SDK reference design as an example, this option is configured as follows:

- In the "Output file format (-O)" option, select the output file format as "Raw binary".
- Please select "Section": "-j .text" option
- Please select "Section": "-j .data" option

If you have custom sections in your software programming design that map functions or variables to custom sections, add these custom sections in "Other sections (-j)". For example, Gowin_PicoRV32 customizes the following sections:

- .irqsec
- .ldsec
- .btsec

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



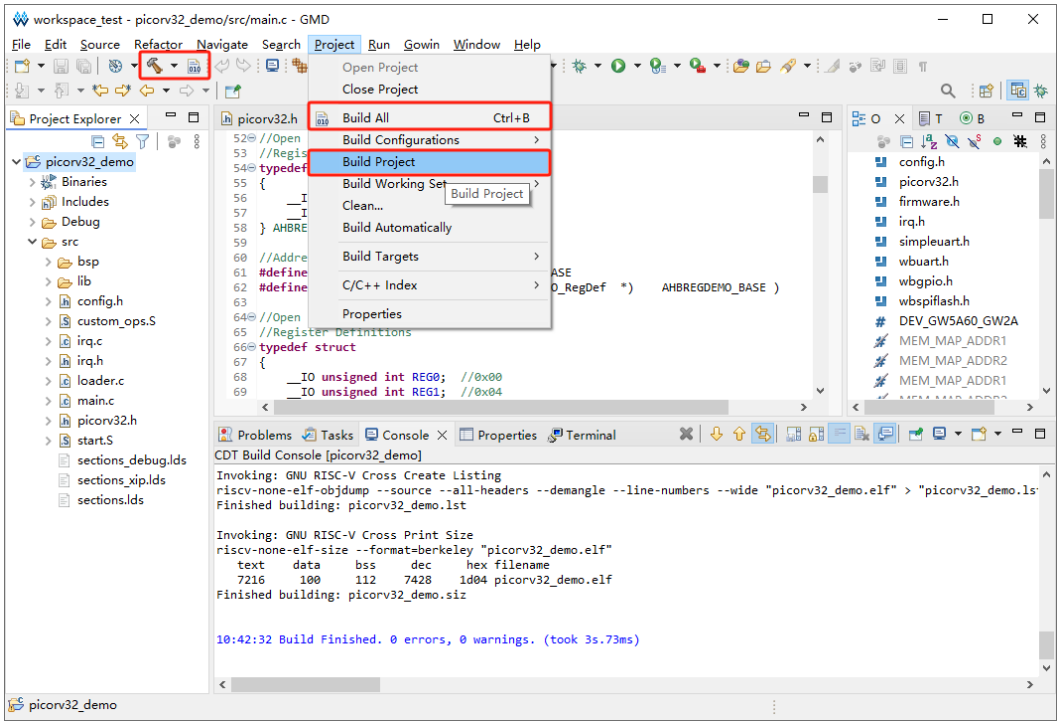
2.3 Build Projects

Define the BUILD_MODE startup parameter macro in config.h according to the configuration of the "Boot Mode" setting in the ITCM options of Gowin_PicoRV32 within the IP Core Generator tool, as referenced from the Gowin_PicoRV32 hardware design.

- 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

After completing the project configuration and code development, build the project by selecting "Build" ( ) or "Build All" ( ) in the toolbar, or "Project > Build Project / Project > Build All" in the menu. This will generate the software executable file, as shown in Figure 2-11.

Figure 2-11 Build



2.4 Download Projects

After the software programming design of Gowin_PicoRV32 is built, see *IPUG913, Gowin_PicoRV32 Software Downloading Reference Manual* for the download method of the software programming design.

2.5 Debug and Run Projects

After downloading the Gowin_PicoRV32 software executable file, if issues occur in the user design, the project can be debugged by connecting the development board with common debugging emulators such as GWU2X, GWUSB, J-Link, or Olimex.

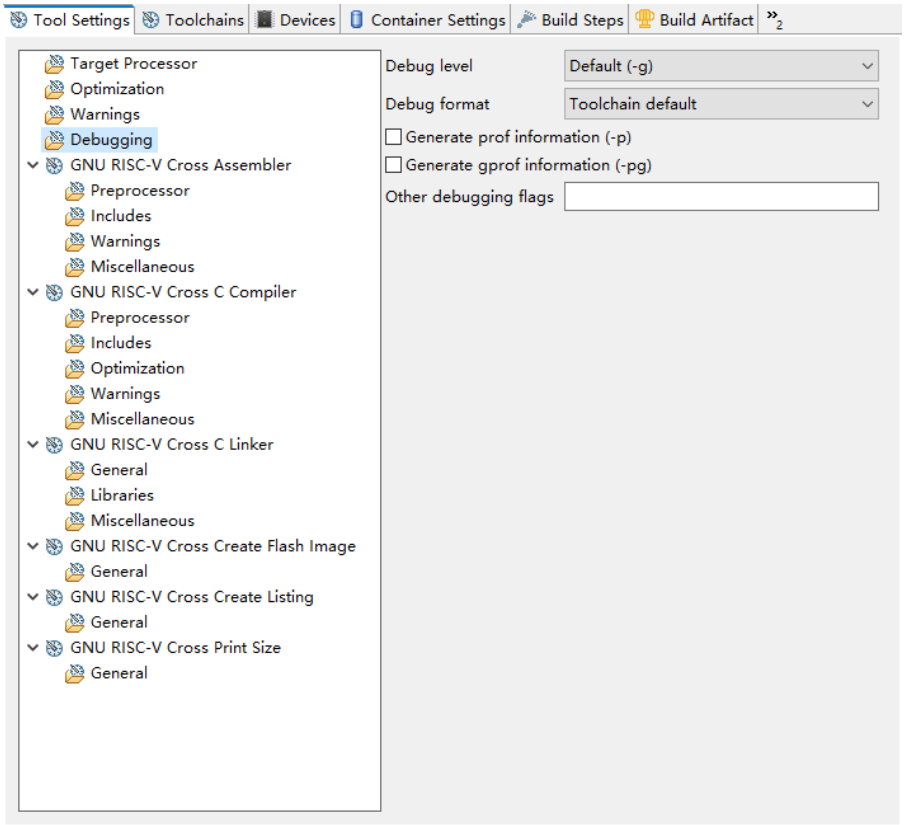
The Gowin_PicoRV32 software debugging process includes:

- Set the debugging level
- Set the Flash linker script file
- Set the debugging mode
- Connect the debugging emulator
- Start the debugging

2.5.1 Set Debug Level

In the "Project Explorer" view, select the current project, right click to select "Properties > C/C++ Build > Settings > Tool Settings > Debugging > Debug level" to set the debug level. For example, -g, -g1, and -g3 generate debugging information with different levels of detail, as shown in Figure 2-12.

Figure 2-12 Set Debug Level



Debug level configuration includes the following options:

- None: No debugging information
- Minimal (-g1): Lowest debugging level
- Default (-g): Default debugging level
- Maximum (-g3): Highest debugging level

2.5.2 Set Flash Linker Script File

When debugging the Gowin_PicoRV32 software design, specify the Flash linker script file sections_debug.lds. For details on configuring the Flash linker script file, you can see 2.2.5 GNU RISC-V Cross C Linker.

Modify the "GNU RISC-V Cross C Linker > General > Script files (-T)" option to "${workspace_loc:/${ProjName}/src/sections_debug.lds}".

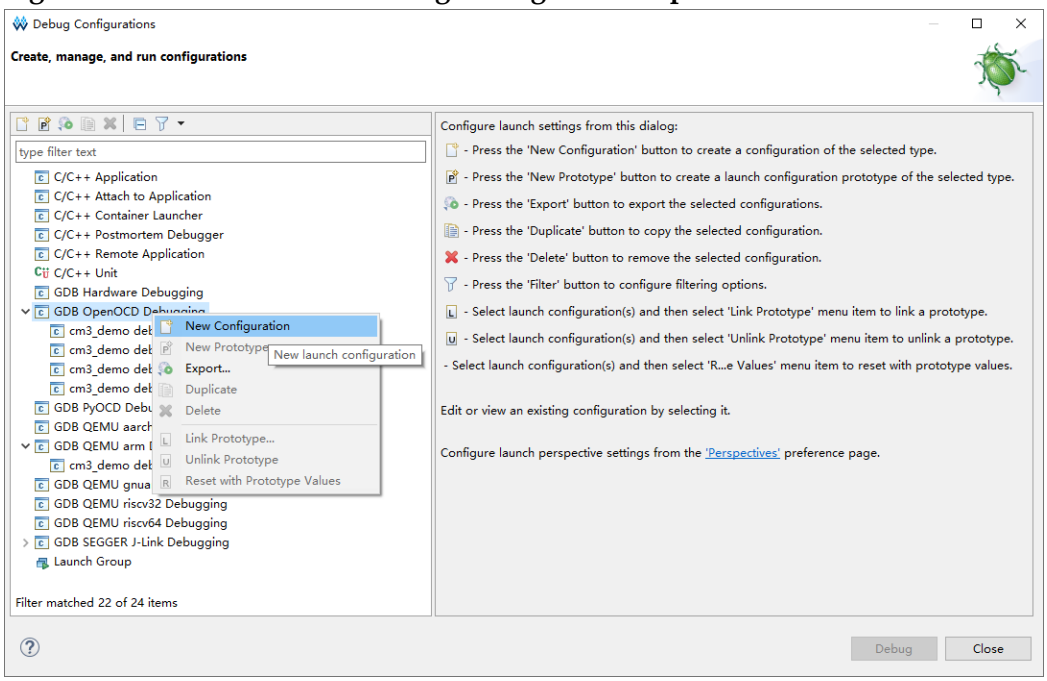
After completing the Flash linker script file configuration, rebuild the project as described in 2.3 Build Project.

2.5.3 Set Debug Mode

From the menu bar, select "Run > Debug Configurations….". Gowin_PicoRV32 supports both GDB OpenOCD Debugging and GDB QEMU riscv32 Debugging modes.

For example, if you choose GDB OpenOCD Debugging, create a debug configuration under "Debug Configurations" for this mode, as shown in Figure 2-13.

Figure 2-13 Create Software Debug Configuration Option

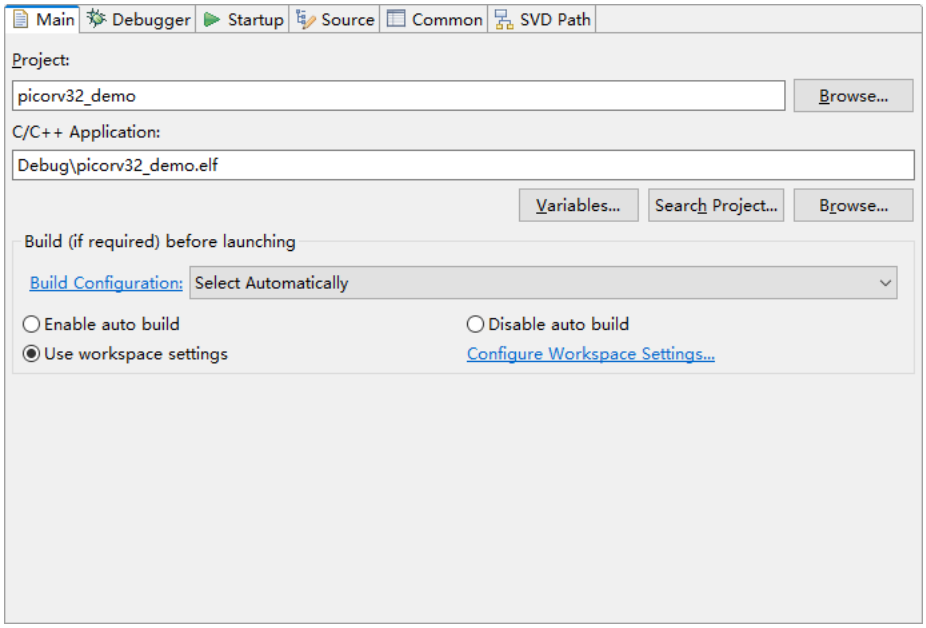


The established debug configuration mainly consists of the Main, Debugger, and Startup tabs.

Main Tab

The "Main" tab is used to set options such as the current project (Project) and the C/C++ Application, as shown in Figure 2-14.

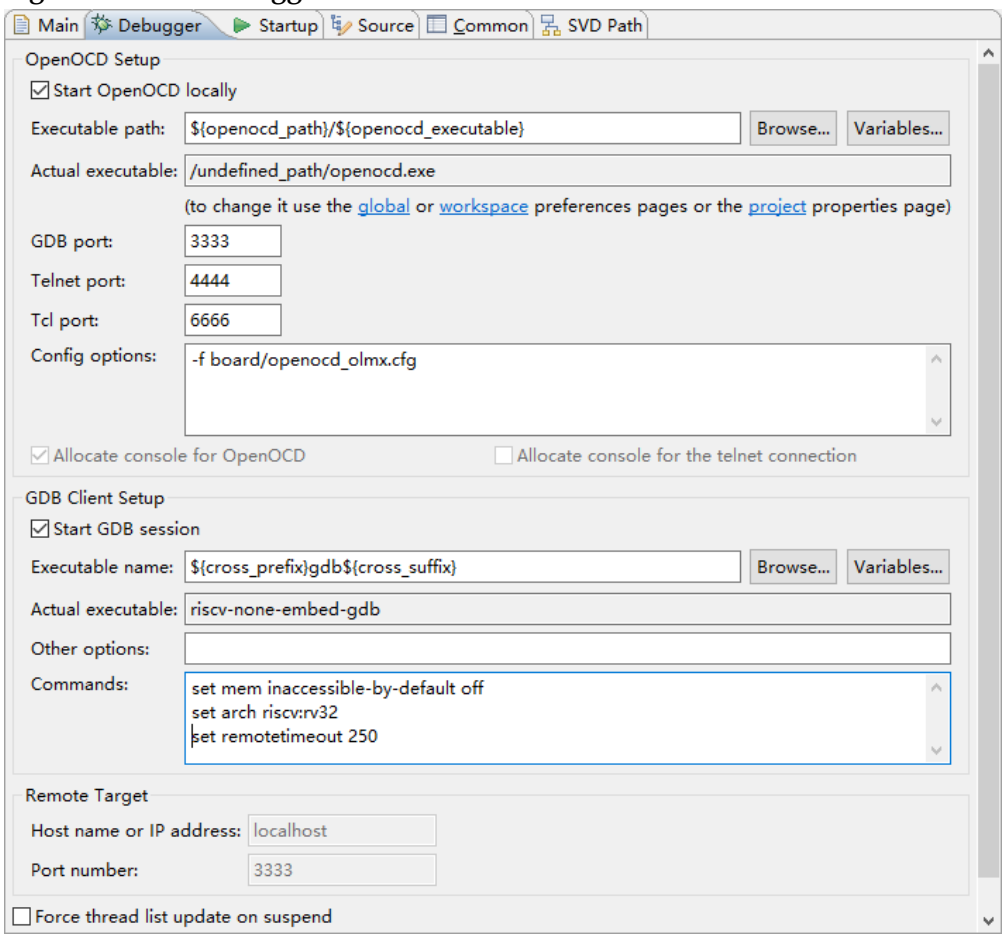
Figure 2-14 Set Main Tab



Debugger Tab

"Debugger" tab is used to set the "OpenOCD" and "GDB" options, as shown in Figure 2-15.

Figure 2-15 Set Debugger Tab

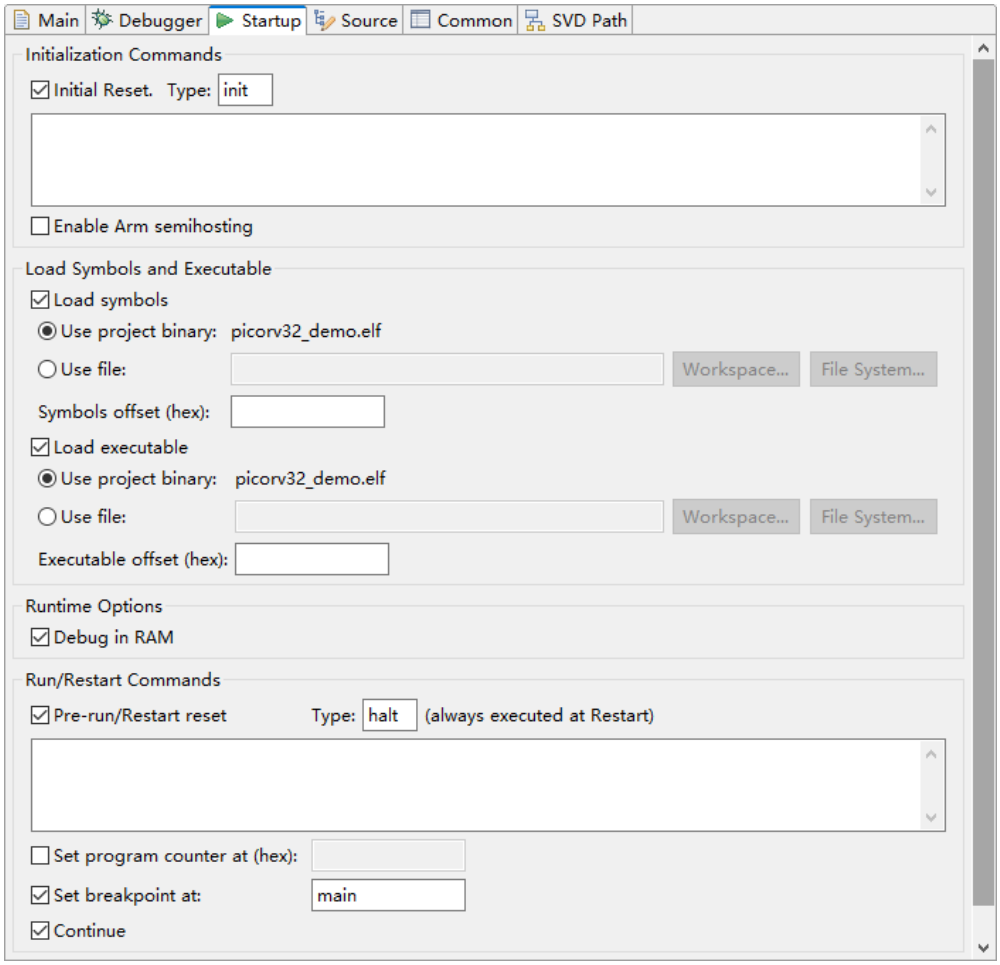


- Config options: Specifies the configuration file for the debugging emulator. For example, when using GWU2X, add "-f board/gowin_u2x_pico.cfg". Other commonly used debugging emulators include:
  - GWUSB: gowin_ftdi_dual_pico.cfg or gowin_ftdi_single_pico.cfg
  - J-Link: gowin_jlink_pico.cfg (when using the WinUSB driver)
  - Olimex: gowin_olimex_pico.cfg
- Commands:
  - set mem inaccessible-by-default off
  - set arch riscv:rv32 (specifies the RISC-V instruction set architecture)
  - set remotetimeout 250 (prevents protocol timeout)

Startup Tab

"Startup" tab is used to set debug and execution, as shown in Figure 2-16.

Figure 2-16 Set Startup Tab



- Initialization Commands
  - Please select the "Initial Reset" option and configure the "Type" as "init", so that the initialization can be executed automatically when

debugging.

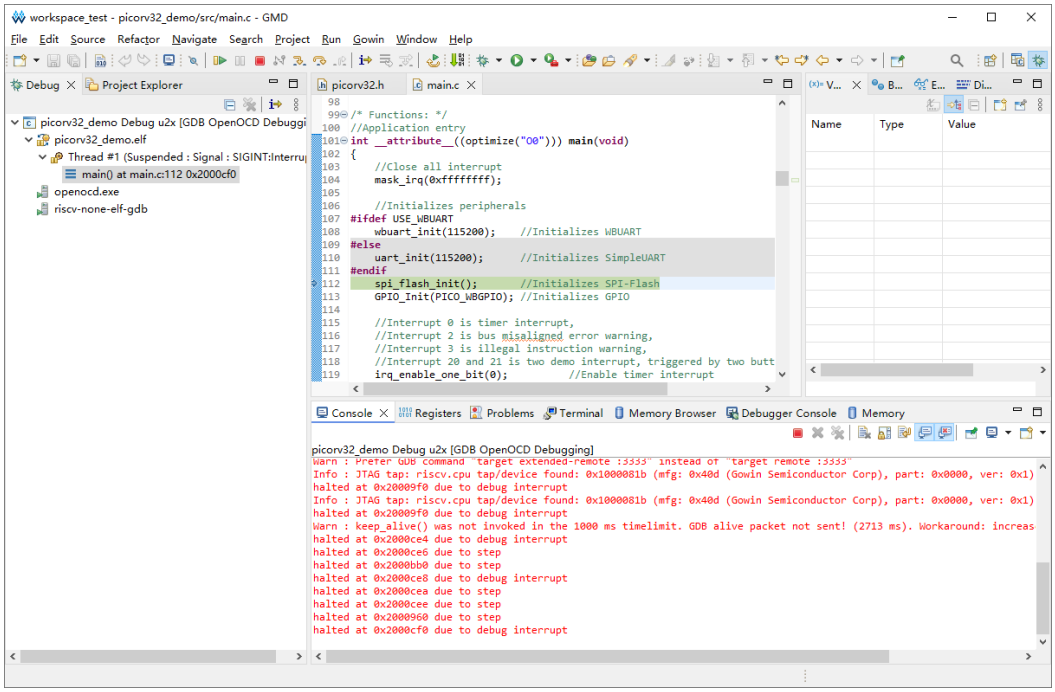
- Please disable the "Enable ARM semihosting" function which Gowin_PicoRV32 does not support.

- Load symbols and Load executable
  - Please select the "Load symbols" option and select "Use project binary" option.
  - Please select the "Load executable" option and select "Use project binary" option.
- Runtime Options: Select the "Debug in RAM" option. During debugging, the software executable is loaded and written into the Instruction Memory (ITCM).
- Run/Restart Commands
  - Please select the "Pre-run/Restart reset" option and configure the "Type" as "halt", so that it can pause automatically when debugging on-line.
  - Please select the "set breakpoint at" option and configure it as "main", so that it can automatically set a breakpoint at the first statement of main function when executing on-line debug.
  - Please select the "Continue" option to automatically run to the breakpoint position and pause when debugging.

2.5.4 Start Debug

For example, when connecting the GWU2X debug emulator, select Debug "  " on the toolbar to launch the Debug view and run the program, as shown in Figure 2-17.

Figure 2-17 Debug View



The Debug view mainly includes the following sub-views:

- Variables sub-view: Displays the names and values of all variables within the current scope.
- Expressions sub-view: Allows adding and monitoring the values of custom expressions.
- Breakpoints sub-view: Lists all breakpoints set in the software project and allows users to manage them.
- Disassembly sub-view: Displays the assembly code corresponding to high-level language, useful for locating instruction-level issues during debugging.
- Console sub-view: Shows process output.
- Memory sub-view: Provides a memory monitor list to inspect and modify program memory.
- Memory Browser sub-view: An alternative to the Memory sub-view for inspecting and modifying process memory.
- Registers sub-view: Displays MCU information at the register bit-field level, allowing direct modification of registers or bit fields in the cells.

Debug Interrupt Handler

Gowin_PicoRV32 is an area-optimized RISC-V ISA processor with a simple and easy-to-use interrupt control system. It does not support hardware interrupt nesting or interrupt priority configuration.

Therefore, during debugging, when the program is in a paused state (such as during single-step execution, breakpoint execution, or when halted at a specific line or instruction), the MCU cannot respond to external interrupt requests.

If you want to execute on-line debug of the external interrupt handler, you need to set a breakpoint at the corresponding location in the interrupt handler and execute the program into a continuous running state.

Click "Resume" () on the tool bar to put the program into a continuous running state, and if no breakpoint is met, it will remain in a continuous running state. In this state, when the external device issues an interrupt request, MCU executes the interrupt handler and enter the pause state when it runs to the breakpoint position in the interrupt handler. At this time, you can execute single-step debug, breakpoint debug, viewing variables, and other operations in the interrupt handler.

3 Reference Design

Gowin_PicoRV32 supports the reference designs for the GMD (2025.01, tested). Access the following reference designs via the link.

…\ref_design\MCU_RefDesign\picorv32_demo

…\ref_design\MCU_RefDesign\picorv32_qemu_demo


GOWIN
PROGRAMMING FOR THE FUTURE