




# Gowin PicoRV32 IDE Software **Reference Manual**

IPUG910-1.3E, 07/16/2021

**Copyright © 2021 Guangdong Gowin Semiconductor Corporation. All Rights Reserved.**

**GOWIN**, , Gowin and GOWINSEMI are trademarks of Guangdong Gowin Semiconductor Corporation and are registered in China, the U.S. Patent and Trademark Office, and other countries. All other words and logos identified as trademarks or service marks are the property of their respective holders. No part of this document may be reproduced or transmitted in any form or by any denotes, electronic, mechanical, photocopying, recording or otherwise, without the prior written consent of GOWINSEMI.

#### **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. All information in this document should be treated as preliminary. 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	<ul style="list-style-type: none"><li>● MCU supports GPIO of Wishbone bus interface;</li><li>● MCU supports extension AHB bus interface;</li><li>● MCU supports off-chip SPI-Flash download and startup;</li><li>● MCU supports the read, write and erasure SPI-Flash;</li><li>● MCU supports Hardware Stack Protection and Trap Stack Overflow.</li></ul>
06/01/2020	1.2E	<ul style="list-style-type: none"><li>● MCU software online debugging function supported;</li><li>● MCU core interrupt handling function enhanced;</li><li>● MCU core instructions optimized.</li></ul>
07/16/2021	1.3E	MCU software reference design updated.

# Contents

<b>Contents .....</b>	<b>i</b>
<b>List of Figures .....</b>	<b>ii</b>
<b>List of Tables .....</b>	<b>ii</b>
<b>1 GMD Installation and Configuration .....</b>	<b>1</b>
<b>2 Software Programming Template .....</b>	<b>2</b>
2.1 Template Project Creation .....	2
2.1.1 Project Creation .....	2
2.1.2 Platform Type Configuration .....	3
2.1.3 Toolchain Compilation and Path Configuration .....	3
2.1.4 Import Software Programming Design .....	4
2.2 Tempalte Project Configuration .....	4
2.2.1 Target Processor Configuration .....	5
2.2.2 Optimization Configuration .....	8
2.2.3 Configure GNU RISC-V Cross Assembler > Includes .....	10
2.2.4 Configure GNU RISC-V Cross C Compiler > Includes .....	11
2.2.5 Configure GNU RISC-V Cross C Linker .....	12
2.2.6 Configure GNU RISC-V Cross Create Flash Image .....	13
2.3 Build .....	14
2.4 Download .....	15
2.5 Online Debugging .....	15
2.5.1 Software Debugging Levels Configuration .....	15
2.5.2 Software Debugging Options Configuration .....	16
2.5.3 Connect the Debugging Emulator .....	19
2.5.4 Start Online Debugging .....	19
<b>3 Reference Design .....</b>	<b>21</b>

# List of Figures

Figure 2-1 Project Creation .....	2
Figure 2-2 Platform Type Configuration .....	3
Figure 2-3 Select Tool Chain and Configure Path .....	3
Figure 2-4 Target Processor Configuration .....	5
Figure 2-5 Optimization Configuration .....	8
Figure 2-6 Configure GNU RISC-V Cross Assembler > Includes .....	11
Figure 2-7 Configure GNU RISC-V Cross C Compiler > Includes .....	11
Figure 2-8 Configure GNU RISC-V Cross C Linker .....	12
Figure 2-9 Configure GNU RISC-V Cross Create Flash Image .....	14
Figure 2-10 Build .....	14
Figure 2-11 Debugging Configuration .....	15
Figure 2-12 Create Debugging Configuration Options .....	16
Figure 2-13 Main Configuration .....	16
Figure 2-14 Debugger Option Configuration .....	17
Figure 2-15 Startup Option Configuration .....	18

# List of Tables

Table 2-1 Data Type Width of 32-bit RISC-V Architecture Processor .....	6
---	---

# 1 GMD Installation and Configuration

GOWIN MCU Designer, the MCU compiling software, supports the compiling of Gowin\_PicoRV32 software design, compile, download and debugging.

The installation package of GOWIN MCU software Designer is available at Gowinsemi website:

<http://www.gowinsemi.com.cn/prodshow.aspx>.

For the installation and configuration of Gowin MCU Designer and Olimex Debugger Driver for Gowin\_PicoRV32, please refer to [SUG549](#), *GOWIN MCU Designer User Guide*.

**Note!**

GOWIN MCU Designer V1.1 and above is recommended to use.

# 2 Software Programming Template

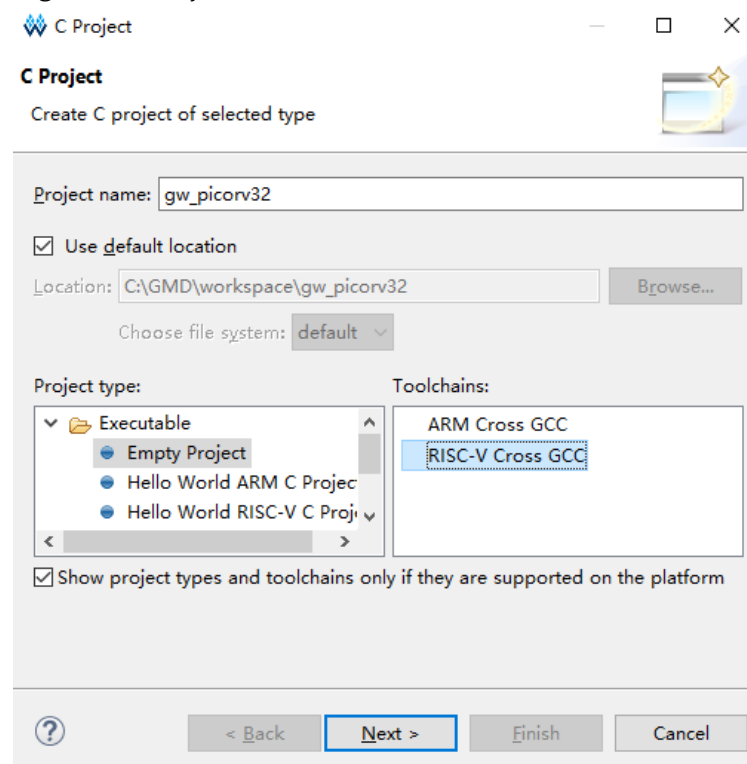
## 2.1 Template Project Creation

### 2.1.1 Project Creation

Click "File > New > C Project" on the menu bar, as shown in Figure 2-1.

1. Create a project name and location;
2. Select the "Empty Project" project type;
3. Select "RISC-V Cross GCC" tool chains;
4. Click "Next".

**Figure 2-1 Project Creation**

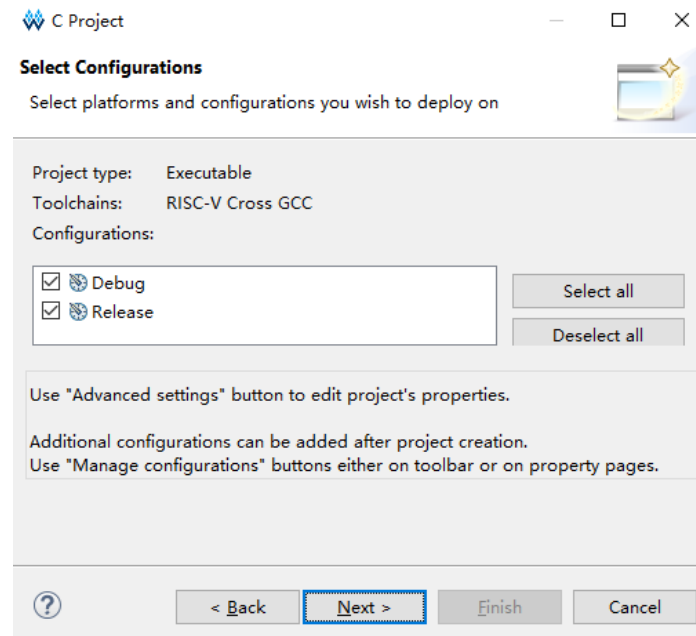




## 2.1.2 Platform Type Configuration

Configure platform type. Select "Debug" and "Release". Click "Next", as shown in Figure 2-2.

**Figure 2-2 Platform Type Configuration**

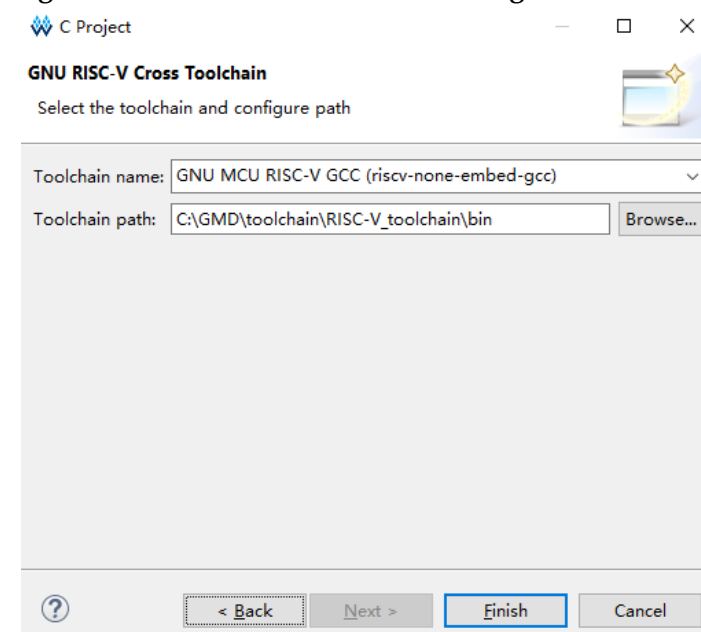


## 2.1.3 Toolchain Compilation and Path Configuration

Select the "riscv-none-embed-gcc" cross compilation toolchain and its path and click "Finish" to finish creating the software programming template of Gowin\_PicoRV32, as shown in Figure 2-3.

The default configuration is the installation path of RISC-V cross compilation tool. If users need self-defined RISC-V cross compilation tool chain, manually modify the specified toolchain location.

**Figure 2-3 Select Tool Chain and Configure Path**



### 2.1.4 Import Software Programming Design

After the software programming project is created, select the newly-created project in the workspace and import the software programming design.

Take reference design in SDK for an instance, the software programming design content structures and codes are listed as follows.

- Gowin\_PicoRV32: Gowin\_PicoRV32 core startup files
- SYSTEM: Gowin\_PicoRV32 core and system definition
- PERIPHERAL: Peripheral driving function library
- CONFIG: Download method profile
- LINKER: Flash linker
- USER: User application Code

If codes under the software programming design content are updated, right-click on the current project in "Project Explorer" view and select "Refresh" to update the file and codes in GMD project template.

## 2.2 Tempalte Project Configuration

In GOWIN MCU Designer, select the current project, right click "Properties", and select "C/C++ Build >Setting > Toolchains" to configure Gowin\_PicoRV32 template project parameters.

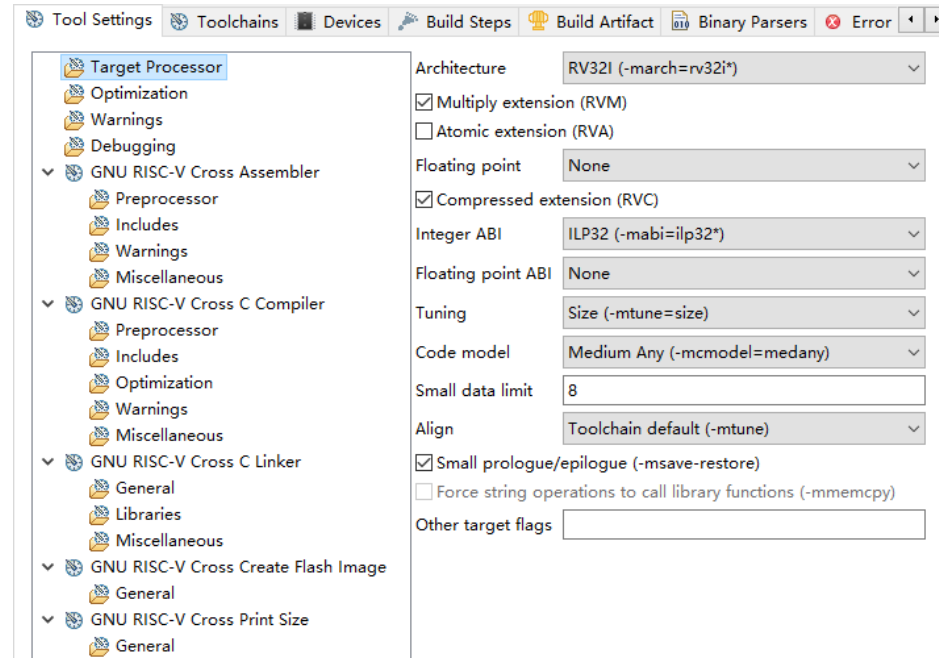
Gowin\_PicoRV32 must be configured with the following parameter options:

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

## 2.2.1 Target Processor Configuration

The "Target Processor" option is configured as shown in Figure 2-4.

Figure 2-4 Target Processor Configuration



- **Architecture**  
RV32I(-march=rv32i\*), Gowin PicoRV32 supports RISC-V32 integer instruction only, so the option of RV32I is selected.
- **Multiply extension (RVM)**  
If RV32M extension is required, enable "Multiply extension(RVM)". At the same time, enable "Support RV32M Extends" in IP Core Genrator parameters of Gowin PicoRV32 hardware design, or when fast multiply instruction is compiled, Gowin\_PicoRV32 run-time error will happen.
- **Atomic extension (RVA)**  
Disable this option, for Gowin PicoRV32 does not support RVA.
- **Floating point**  
Set this option as "None", for Floating point does not support floating point.
- **Compressed extension (RVC)**  
If RV32C extension is required, enable "Compressed extension (RVC)". At the same time, enable "Support RV32C Extends" in IP Core Genrator Gowin PicoRV32 IP parameters of Gowin PicoRV32 hardware design, or when RISC-V 16-bit compressed instruction is compiled, Gowin\_PicoRV32 run-time error will happen.
- **Integer ABI**  
Set ABI function call rules supported by RISC-Vtarget platform. PicoRV32 is 32-bit RISC-V architecture processor platform and does not support hardware floating instruction, so this option is configured as 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 (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. Configure this option as `Size(-mtune=size)`, for it does not support Gowin\_PicoRV32.

- Code model

Set the "-mcmmodel" parameter. This parameter specifies the addressing range of the programmer. Select Medium Any (-mcmmodel=medany). The (-mcmmodel=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, which 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 `Strict(-mstrict-align)` is recommended.

- Small prologue/epilogue(-msave-restore)

If enabled, set to call the library functions of the smallest size, but slower, start and return code. Fast inline code is used by default.

- Other target flags options can be added manually follows:

- Allow use of PLTs(-mplt)

If this option is enabled, the interrupt control code is generated using PLT; otherwise, PLT is not used.

- Floating-point divide/sqrt instructions(-mfdiv)

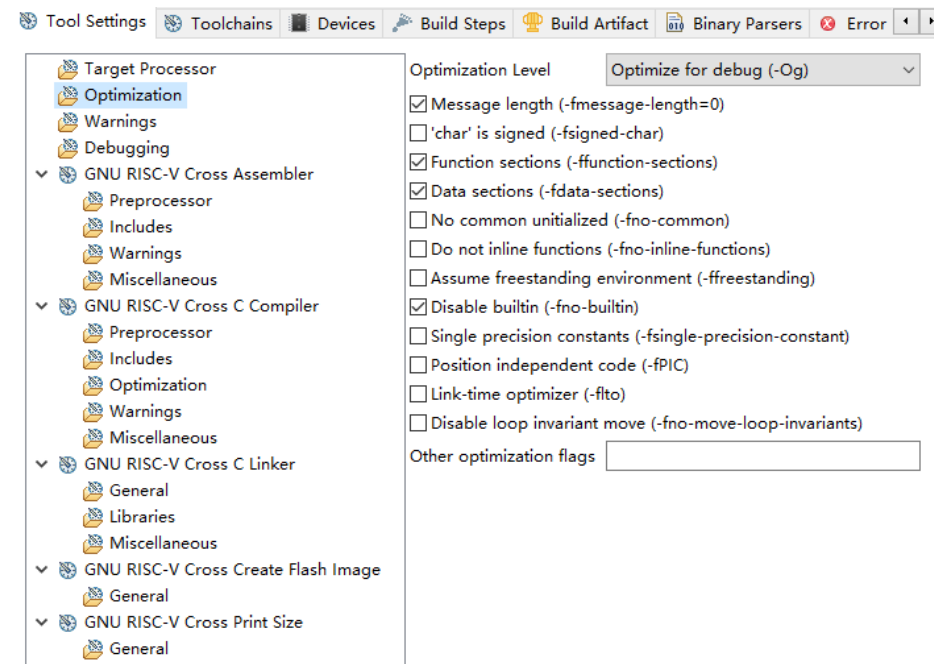
If enabled, hardware floating-point division and square root instructions are used, requiring processor support for RV32F and RV32D instruction set extensions and floating-point registers. Disable this option, for Gowin\_PicoRV32 does not support RV32F and RV32D instruction set extensions.

- Integer divide instructions(-mdiv)
- If enabled, integer division hardware instructions are used, requiring processor support for RV32M instruction set extensions. Gowin\_PicoRV32 supports RV32M instruction set extensions, so the Multiply extension(RVM) option can be enabled and enable the "Support RV32M Extends" option in the Gowin PicoRV32 IP parameter configuration of IP Core Generator designed for Gowin PicoRV32 hardware.
- -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 relocation operator. Another configuration is to use assembly macros, which can limit optimization.
- -mrelax/-mno-relax  
Use FLASH Linker relaxation to reduce the number of instructions required to implement symbolic addresses. The default is to use connector 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 values, and 'natural' USES natural alignment. The default value is 'xlen'.

## 2.2.2 Optimization Configuration

The “Optimization” option is configured as shown in Figure 2-5.

Figure 2-5 Optimization Configuration



- **Optimization Level**  
 "-O" level is used to set the optimization level to optimize compile Size, run speed, compile time, etc. The options include -O0, -O1, -O2, -O3, -Os, -Og. The optimization level of -O0/-O1/-O2/-O3 is improved step by step.
  - -O0: No optimization;
  - -Os: Based on -o2, turn off the option that causes compilation Size to increase to achieve the optimization of compilation Size.;
  - -Og: Add some compilation options suitable for debugging on the basis of -o0. Gowin PicoRV32 does not support on-chip debugging, so other optimization strategies are recommended.
- **-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 disable this option.
- **'char' is signed (-fsigned-char)**  
 Set char type data to 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.

- This option can be enabled if the connector can perform the referenced localized optimizations in the improved instruction space.
- When used with connector 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!

Enable this option only if it has a significant effect. When this option is specified, the compiler and connector create larger Size objects and executables, reducing compilation speed and affecting code generation. This option places the compiler and assembler optimized using relative positions within the translation unit, as these positions are unknown before the connection. One example of this optimization is the relaxation of calls to short call instructions. It is recommended to disable this option and enable the -gc-sections option in the connector settings to reduce compilation 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 connector 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 enabled, 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, iwhere 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 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.
  - To 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 RTL 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.

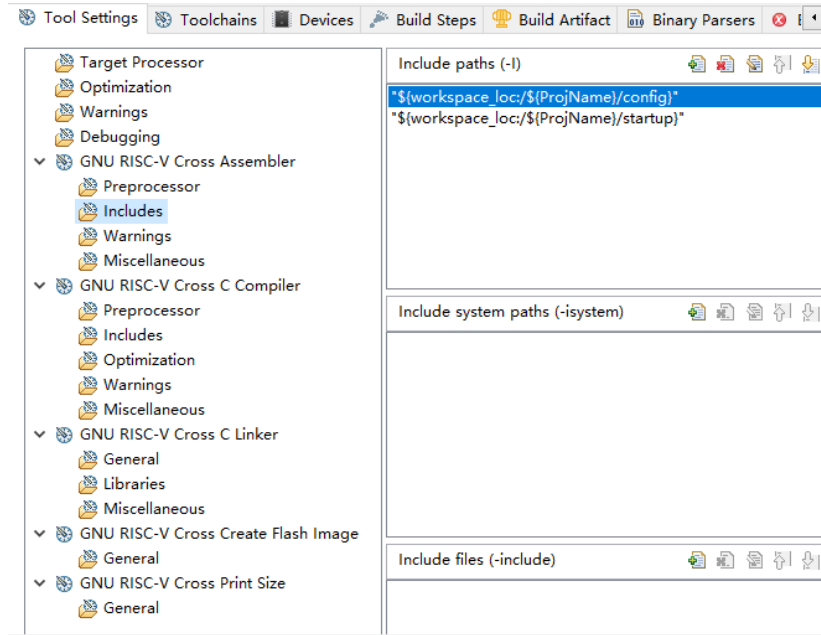
### 2.2.3 Configure GNU RISC-V Cross Assembler > Includes

Select “GNU RISC-V Cross Assembler > Includes > Include paths (-I)” to configure the assembly file reference path, as shown in Figure 2-6.

Take reference design in SDK for an instance, assembly reference file paths are listed as follows.

- "\${workspace\_loc}/\${ProjName}/STARTUP}"
- "\${workspace\_loc}/\${ProjName}/ CONFIG}"



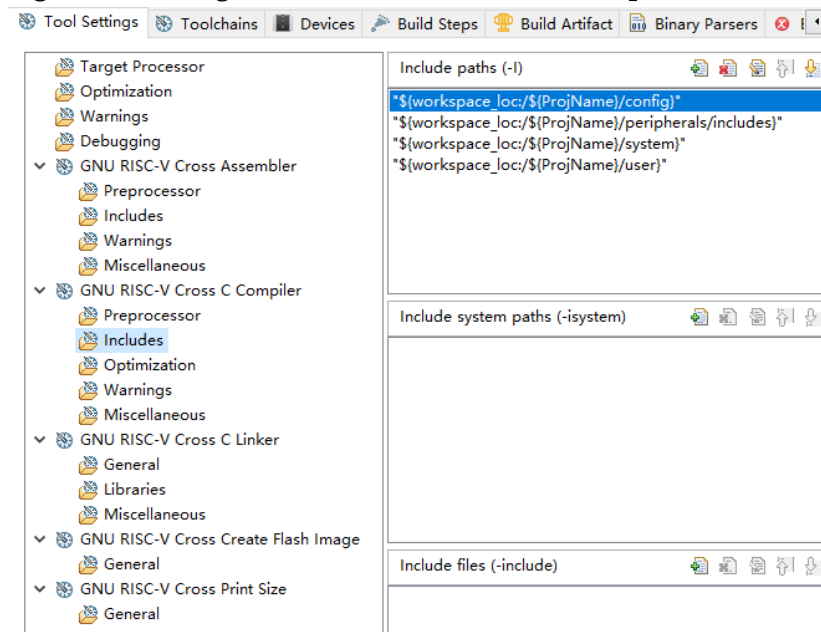
**Figure 2-6 Configure GNU RISC-V Cross Assembler > Includes**

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

Select "GNU ARM I, GNU C Compiler > Includes > Include paths> Include paths (-I)" to configure the C header file reference path of the current project, as shown in Figure 2-7.

Take reference design in SDK for an instance, the C header file paths are listed as follows.

- "\${workspace\_loc:\${ProjName}/ CONFIG}"
- "\${workspace\_loc:\${ProjName}/PERIPHERALS/Includes}"
- "\${workspace\_loc:\${ProjName}/SYSTEM}"
- "\${workspace\_loc:\${ProjName}/USER}"

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

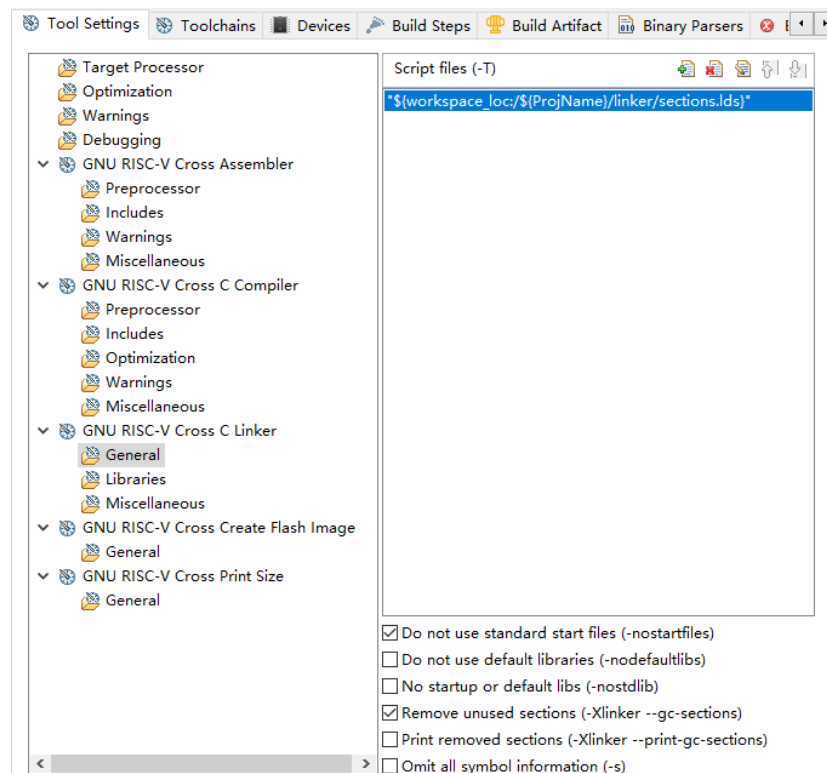
## 2.2.5 Configure GNU RISC-V Cross C Linker

Select "GNU RISC-V Cross C Linker > General > Script files (-T)" to configure Flash linker gw1ns4c\_flash.ld, as shown in Figure 2-8.

According to Gowin\_PicoRV32 hardware design in IPCore Generator ITCM > Boot Mode configuration method:

- If Boot Mode is configured as MCU boot from external Flash and run in ITCM or MCU boot and run in ITCM, select sections.ld as Flash linker, such as "\${workspace\_loc}/\${ProjName}/linker/sections.ld}"
- If Boot Mode is configured as MCU boot and run in external Flash, select sections\_xip.ld as Flash linker, such as "\${workspace\_loc}/\${ProjName}/linker/sections\_xip.ld}"
- If Boot Mode is configured as MCU boot from external Flash and run in ITCM or MCU boot and run in ITCM and perform software online debugging, select sections\_debug.ld as Flash linker, such as "\${workspace\_loc}/\${ProjName}/linker/sections\_debug.ld}"

Figure 2-8 Configure GNU RISC-V Cross C Linker



- Do not use standard start files (-nostartfiles)  
Standard startup files are not used when setting up a connection. Gowin\_PicoRV32 must use custom start files, so this option must be enabled.
- Do not use default libraries (-nodefaultlibs)  
Standard system libraries are not used when configuring linking, only the selected libraries are passed to the connector. 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 `memcpy`, `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 connector, and options for specifying system library connections (such as `-static-libgcc` or `-shared-libgcc`) are ignored.
  - This option is set to disabled status.
- 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 compilation size.
- Print removed sections (`-Xlinker -print-gc-sections`)  
This option can be enabled when "Remove unused sections (`-Xlinker -gc-sections`)" is enabled, 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.
  - Enable Do not use standard start files (`-nostartfiles`) and Remove unused sections (`-Xlinker -gc-sections`) is recommended.

## 2.2.6 Configure GNU RISC-V Cross Create Flash Image

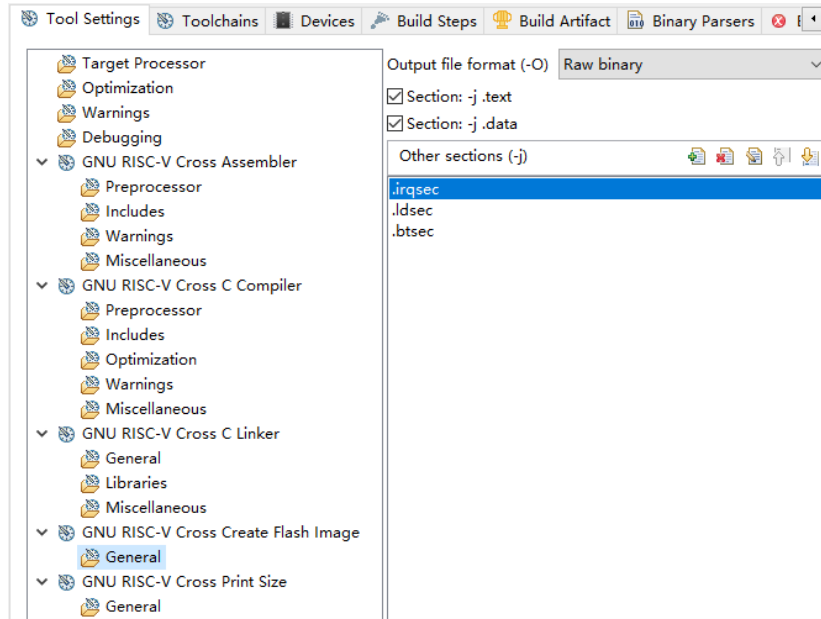
The "NU RISC-V Cross Create Flash Image > General" is configured as shown in Figure 2-9.

Take reference design in SDK for an instance. The option is configured as follows.

- The "Output file format (`-O`) option" configures the output file format to Raw. Binary
- Enable Section: `-j .text`
- Enable Section: `-j .data`

If you have custom sections in your project that map functions or variables to custom sections, add these custom sections in Other sections (`-j`).

- `.irqsec`
- `.ldsec`
- `.btsec`

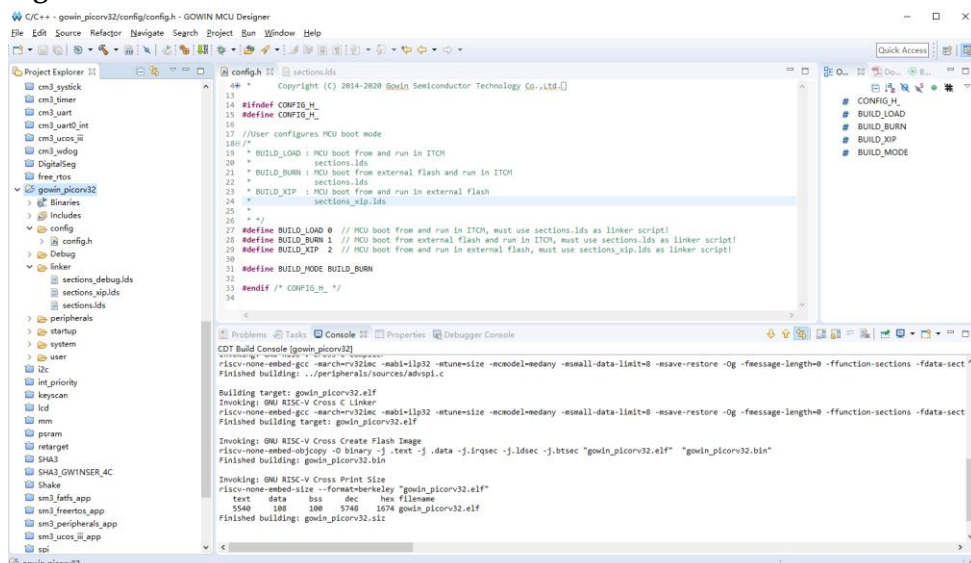
**Figure 2-9 Configure GNU RISC-V Cross Create Flash Image**

## 2.3 Build

According to the IP Core Generator configuration method of “ITCM > Boot Mode” in Gowin\_PicoRV32 hardware design, modify the config.h startup parameter macro to define BUILD\_MODE.

- 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 project configuration and encoding, select the "🔧" compile button on the tool bar to start compiling to generate Software design binary BIN files, as shown in Figure 2-10.

**Figure 2-10 Build**

## 2.4 Download

After the software programming design of Gowin\_PicoRV32 is finished, please refer to [IPUG913](#), *Gowin\_PicoRV32 Software Downloading Reference Manual* for the download method of software programming design.

## 2.5 Online Debugging

After downloading the Gowin\_PicoRV32 software design BIN file, if there is a problem with the user's software application design, you can connect the development board and Olimex debugger (or other debugger supported by the RiscV instruction set architecture processor) to debug the current MCU software design online.

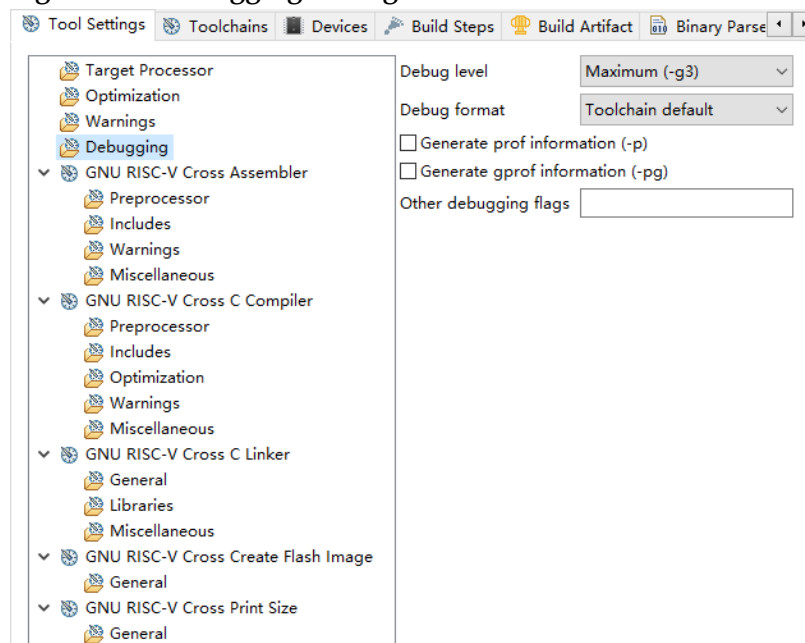
The debugging flow for Gowin\_PicoRV32 Software includes:

- Software Debugging Levels Configuration
- Software debugging options configuration
- Connect the debugging emulator
- Start Software Debugging

### 2.5.1 Software Debugging Levels Configuration

In the Project Explorer view, select the current project and right-click the "Properties > Debugging > Debugging level" option, which is configured as shown in Figure 2-11.

Figure 2-11 Debugging Configuration



Debugging level configuration includes the following options.

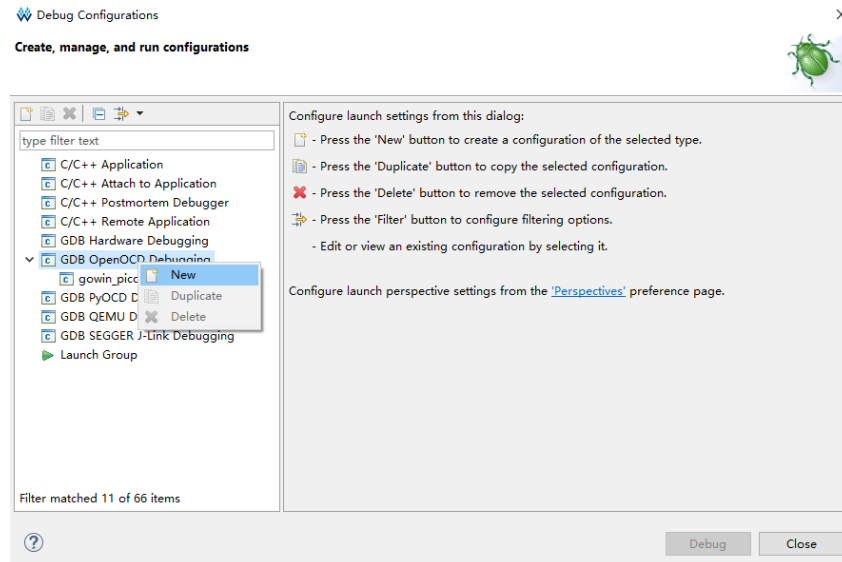
- None: debugging levels configuration
- Minimal (-g1): minimum debugging level configuration

- Default (-g): default debugging level configuration
- Maximum (-g3): maximum debugging level configuration

## 2.5.2 Software Debugging Options Configuration

Select “Run > Debugging Configurations > GDB OpenOCD Debugging” and right click on “New” to create the project debugging configuration options, as shown in Figure2-12 .

**Figure 2-12 Create Debugging Configuration Options**

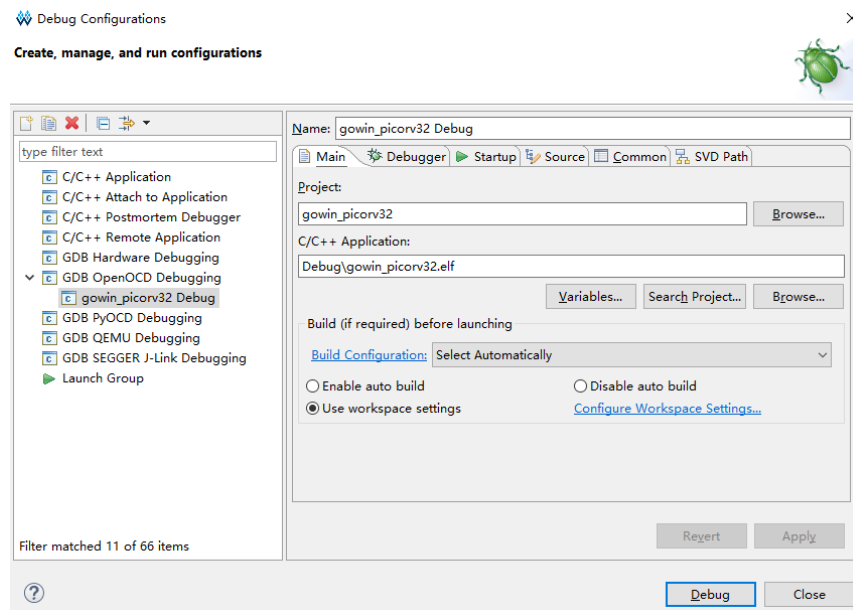


Select the established debugging configuration options. Configure debugging options such as Main, Debugger, Startup, etc.

### “Main” options configuration

Select “Main” to configure theParameters such as Project and C/C++ Application of current project, as shown in Figure 2-13.

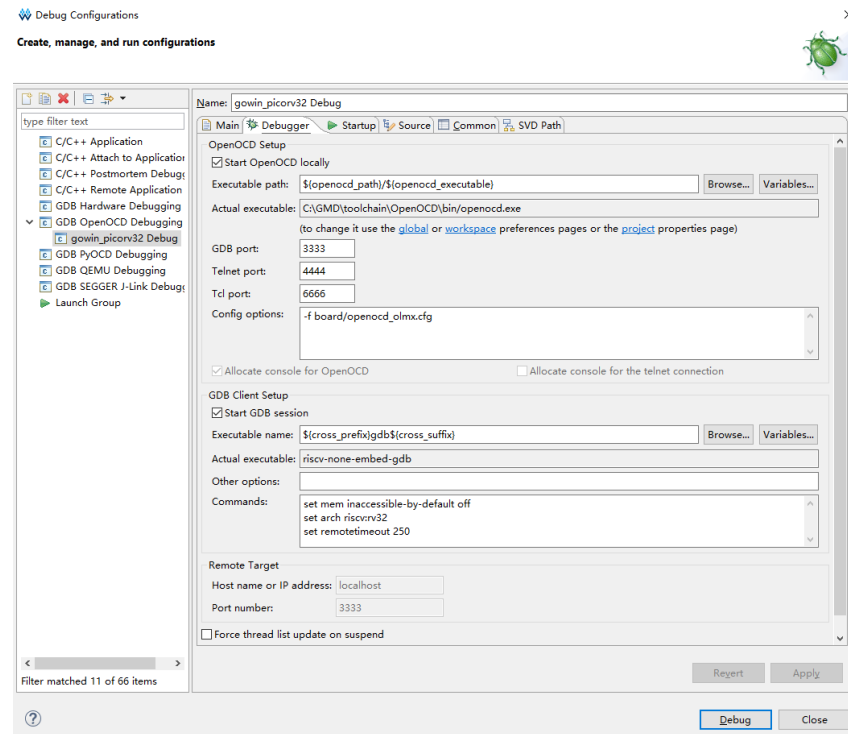
**Figure 2-13 Main Configuration**



## “Debugger” options Configuration

Select "Debugger" to configure GDB and interface, as shown in Figure 2-14.

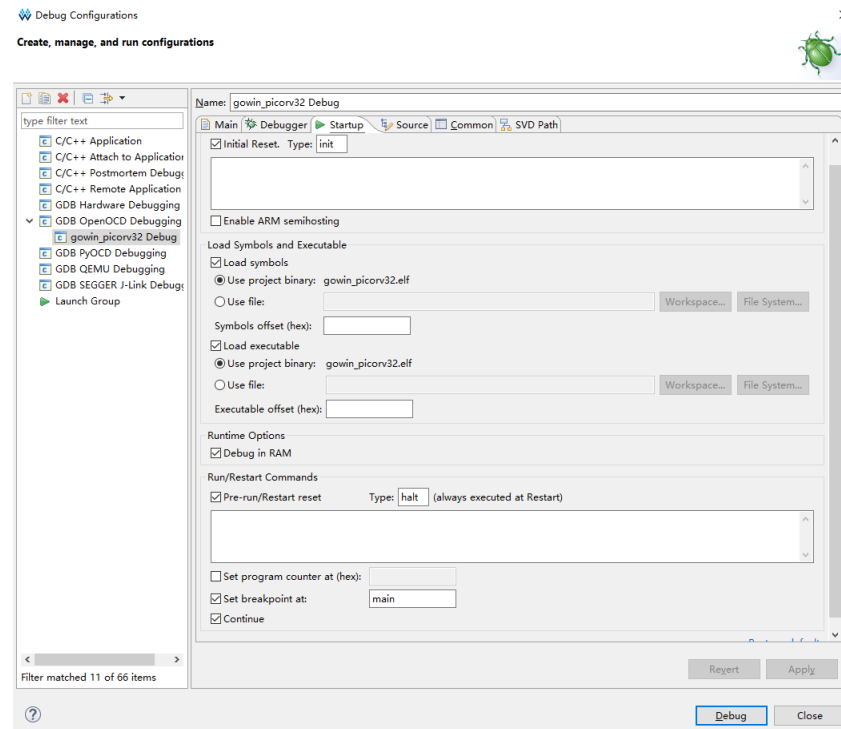
**Figure 2-14 Debugger Option Configuration**



- “OpenOCD Config Options” option configuration  
This option is configured as "-f board/openocd\_olmx.cfg" and specifies the debugging emulator used. Configuration file, GMD uses the Olimex emulator by default.
- “GDB Commands Options” option Configuration
  - Set mem inaccessible-by-default off
  - Set arch riscv:rv32 (specifying the RiscV instruction set architecture)
  - Set remotetimeout 250 (to prevent protocol timeouts)

## “Startup” option Configuration

Select the "Startup" option to configure the option to perform online debugging, etc., as shown in Figure 2-15.

**Figure 2-15 Startup Option Configuration**

- “Initialization Commands” option Configuration
  - Enable the Initial Reset option and configure the Type as "init" to execute the initialization automatically when executing online debugging.
  - Disable the Enable ARM semihosting option for it is not supported by Gowin\_PicoRV32.
- “Load Symbols and Executable” option Configuration
  - Enable “Load symbols” option. Select “Use project binary”.
  - Enable “Load executable” option. Select “Use project binary”.
- “Runtime Options” option Configuration
 

Enable the Debugging in RAM option to read software design binaries and write to command memory ITCM when performing in-circuit debugging.
- “Run/Restart Commands” option Configuration.
  - Enable the Pre-run/Restart reset option and configure the type to "halt" to automatically pause the online debugging;
  - Enable the set breakpoint at option, configured as "main", to automatically set the breakpoint at the first statement position of main function when executing online debugging;
  - Enable the Continue option to automatically run to a breakpoint and pause when performing online debugging.



## 2.5.3 Connect the Debugging Emulator

Take Olimex debugging emulator for an instance.


Connect the development board to the Olimex debugging emulator (or other debugging emulator supported by the RiscV instruction set architecture processor).

Olimex debugging emulator, using Olimex arm-usb-tiny-h, link is: <https://www.olimex.com/Products/ARM/JTAG/ARM-USB-TINY-H/>

For the software installation and configuration of Olimex Debugging Emulator Driver, please refer to [SUG549](#), *GOWIN MCU Designer User Guide*.

The Olimex debugging emulator's TDI, TDO, TMS, TCK pins are in order with the standard JTAG interface. Development board connection, VREF pins to connect 3.3V, 4/6/8/.../20 GND pins need only one connection.

## 2.5.4 Start Online Debugging

Click "Debugging" on the toolbar  to execute the software online debugging, the debugging pages include:

- Code area: you can view C code and assembly code, set breakpoints, and run programs;
- Variable area: you can view the values of general registers or program variables;
- Assembly instruction area: you can view the instructions in the current instruction memory ITCM and the instructions currently being executed;
- Control area: you can control the online debugging process, set breakpoints etc.

### Interrupt handler

The Gowin\_PicoRV32 is an area-optimized RiscV instruction set architecture processor with an easy-to-use interrupt control system design that does not support hardware interrupt nesting or interrupt prioritization.

Therefore, when in-circuit debugging is being performed and the program is in a suspended state (performing operations such as single-step runs, breakpoint runs, etc., that cause the program to run to a certain line or instruction and pause at that location), the MCU cannot respond to an external interrupt request.

If you need to perform in-circuit debugging of an external interrupt handler function, you need to set a breakpoint at the corresponding position in the interrupt handler function to execute the program into a continuous run state.

Clicking "Resume" on the toolbar will put the program into a continuous run state, and it will remain continuous if no breakpoints are encountered. In this state, when the external device sends an interrupt

request, the MCU executes the interrupt processing function and runs to the breakpoint position in the interrupt processing function to enter the pause state, at this time you can perform single-step debugging, breakpoint debugging, variable viewing and other operations in the interrupt processing function.

# 3 Reference Design

Gowin\_PicoRV32 supports GOWIN MCU Designer (V1.1 and above version ) of the software environment via links to the following [reference designs](#):

Gowin\_PicoRV32\ref\_design\MCU\_RefDesign\GMD\_RefDesign\gowin\_picorv32

