



Gowin PicoRV32 IDE Software **Reference Manual**

IPUG910-1.4E, 01/29/2022

Copyright © 2022 Guangdong Gowin Semiconductor Corporation. All Rights Reserved.

GOWIN and GOWIN 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. 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">● 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	<ul style="list-style-type: none">● 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	<ul style="list-style-type: none">● MCU software reference design updated.● IDE software options configuration optimized.● IDE software online debug process improved.

Contents

Contents	i
List of Figures	ii
List of Tables	iii
1 GMD Installation&Configuration	1
2 Software Programming Template	2
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 Template Project Configuration	5
2.2.1 Target Processor Configuration	5
2.2.2 Optimization Configuration	8
2.2.3 Configure GNU RISC-V Cross C Compiler > Includes	10
2.2.4 Configure GNU RISC-V Cross C Linker	11
2.2.5 Configure GNU RISC-V Cross Create Flash Image	12
2.3 Build	13
2.4 Download	14
2.5 On-line Debug	14
2.5.1 Configure Software Debug Level	14
2.5.2 Configure and Debug Flash Linker	15
2.5.3 Configure Software Debug Option	15
2.5.4 Connect Debug Emulator	18
2.5.5 Start On-line Debug	18
3 Reference Design	20

List of Figures

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

List of Tables

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

1 GMD Installation&Configuration

GOWIN MCU Designer, the MCU compiling software, supports the compiling of Gowin_PicoRV32 software design, compile, download, and debug.

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

<https://www.gowinsemi.com/en/support/arm/>.

For the installation and configuration of Gowin MCU Designer and Olimex Debugger Driver, the driver software of debug emulator for Gowin_PicoRV32, see [SUG549](#), *GOWIN MCU Designer User Guide*.

Note!

It is recommended to use GOWIN MCU Designer (V1.1).

2 Software Programming Template

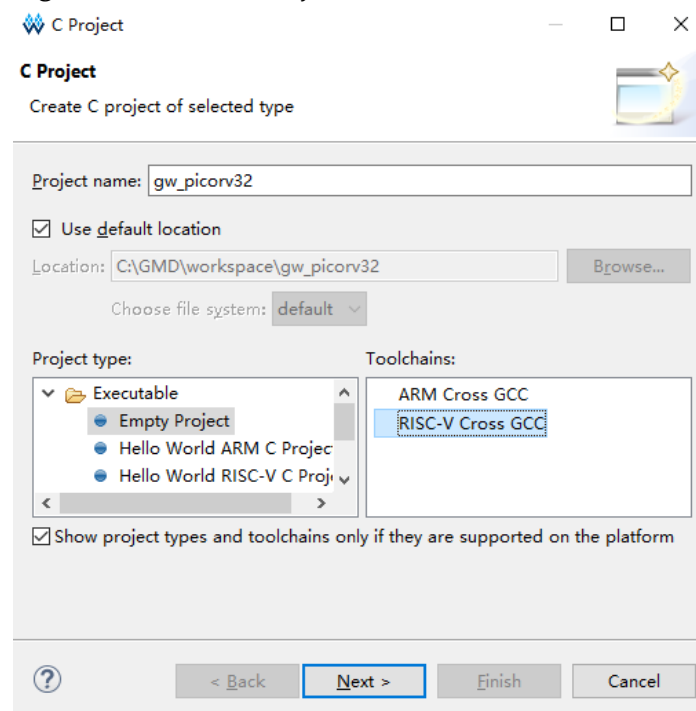
2.1 Template Project Creation

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 project type “Empty Project”;
3. Select the tool chain “RISC-V Cross GCC”;
4. Click “Next”.

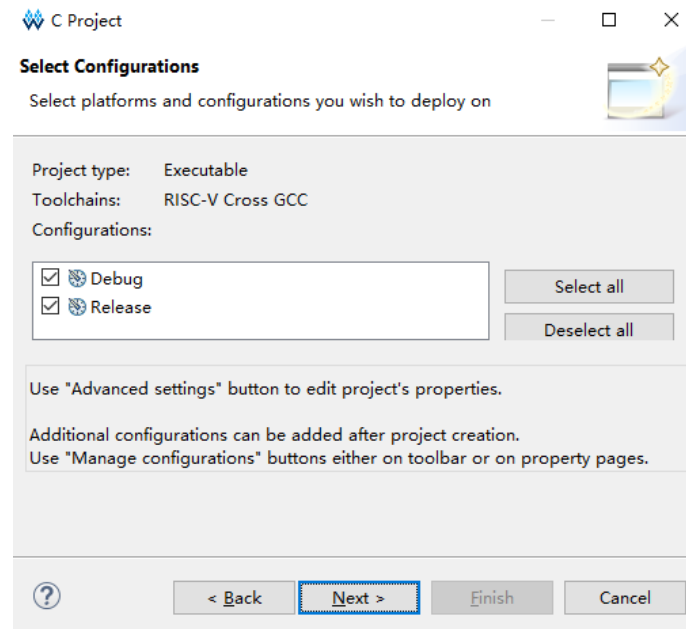
Figure 2-1 Create a Project



2.1.2 Platform Type Configuration

Check “Debug” and “Release”, then click “Next” to configure the platform type, as shown in Figure 2-2.

Figure 2-2 Select Platform and Configuration

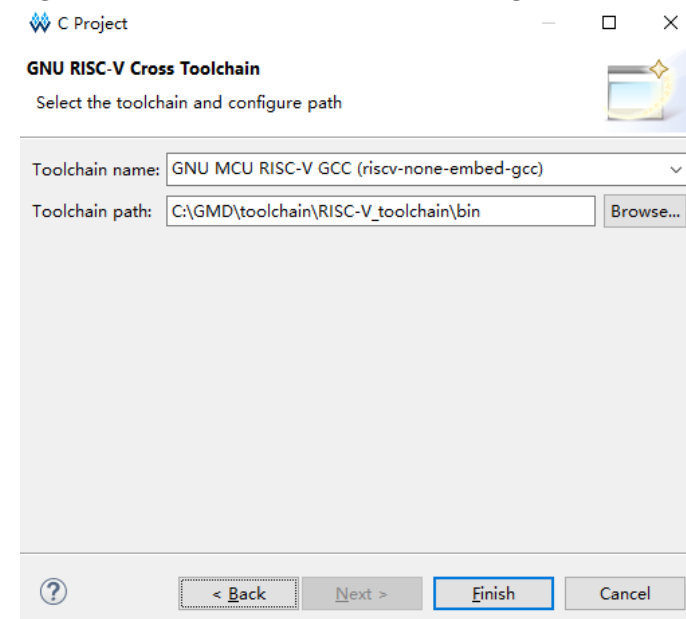


2.1.3 Toolchain Compilation and Path Configuration

Select the “riscv-none-embed-gcc” cross compilation tool chain 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 you need self-defined RISC-V cross compilation tool chain, manually modify the specified tool chain 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 created project in the workspace and import the software programming design.

Take reference design in SDK for an instance, the structure of software programming design contents and the definition of codes are listed 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.ld: the link for debug
- Sections_xip.ld: the link for external SPI-Flash to run program instructions
- sections.ld: the link for internal ITCM to run program instructions

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

2.2 Template Project Configuration

In GOWIN MCU Designer, select the current project in the view “Project Explorer”, right click “Properties”, and select “C/C++ Build > Settings > Toolchains” to configure the parameters of Gowin_PicoRV32 template project.

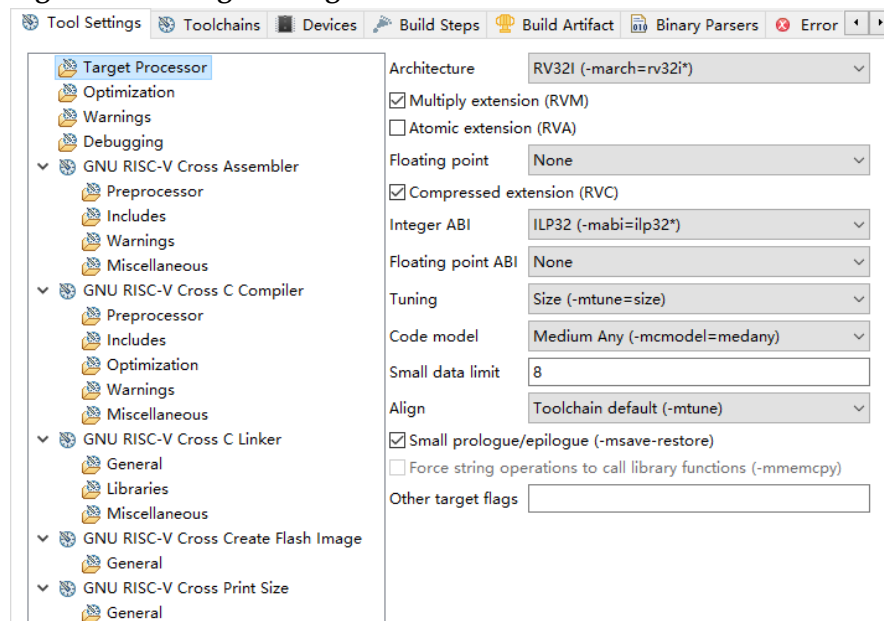
Gowin_PicoRV32 is required to configure the following parameters.

- Target Processor
- Optimization
- 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

Configure “Target Processor”, as shown in Figure 2-4.

Figure 2-4 Configure Target Processor



- Architecture
Select “RV32I(-march=rv32i*)”, because Gowin_PicoRV32 supports RISC-V 32-bit integer instructions only.
- Multiply extension (RVM)
If you need to use RV32M extension, check “Multiply extension (RVM)”. At the same time, check “Support RV32M Extends” in the Gowin PicoRV32 Core parameter configuration of the Gowin_PicoRV32 Hardware Design IP Core Generator. Otherwise, when RVM is

compiled, an error will occur in Gowin_PicoRV32.

- **Atomic extension (RVA)**
Gowin_PicoRV32 does not support atomic instruction extension, so do not check this option.
- **Floating point**
Gowin_PicoRV32 does not support floating-point extension, so select "None".
- **Compressed extension (RVC)**
If you need to use RV32C extension, check "Compressed extension (RVC)". At the same time, check "Support RV32C Extends" in the Gowin PicoRV32 Core parameter configuration of the Gowin_PicoRV32 Hardware Design IP Core Generator. Otherwise, when 16-bit RVC is compiled, an error will occur in Gowin_PicoRV32.
- **Integer ABI**
Set call 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 "-mcmmodel". This parameter specifies the program's addressing range, and 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. This parameter specifies the

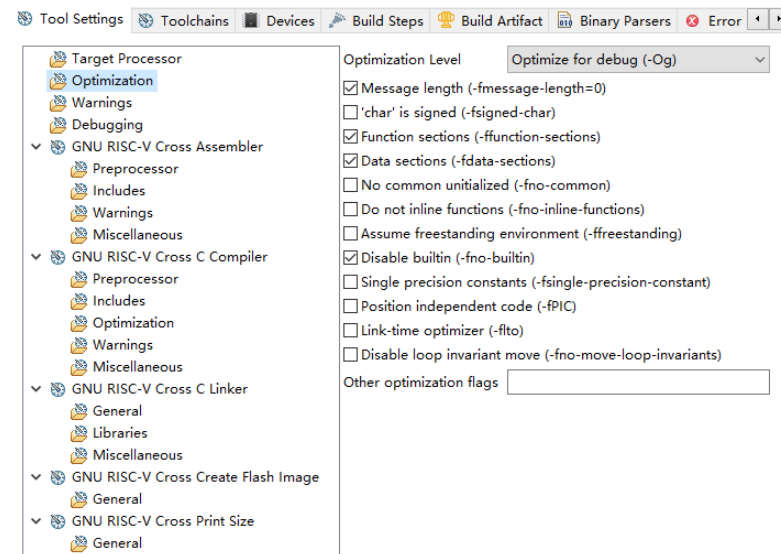
maximum size in bytes of global and static variables that can be placed into small data areas. This parameter is set to 8.

- **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 check this option, set to call the library functions of the smallest size, but slower, start and return code. Fast inline code is used by default.
- You can manually add the following configuration 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 configured, use integer division hardware instructions, requiring processor support for RV32M instruction set extensions. Gowin_PicoRV32 supports RV32M instruction set extensions, so the “Multiply extension(RVM)” option can be configured. And check the “Support RV32M Extends” option in the Gowin PicoRV32 Core parameter configuration of Gowin_PicoRV32 hardware design IP Core Generator.
 - **-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-explicit-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 Configuration

Configure “Optimization” option, as shown in Figure 2-5.

Figure 2-5 Configure Optimization



- **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, optimize the compile size by turning off the options that lead to an increase in the compile size.
 - -Og: Add some compile options suitable for debug on the basis of -O0. Gowin_PicoRV32 does not support on-chip debug, so it is recommended to adopt other optimization strategies.
- **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 check 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!

Check this option only if it works well. And check the “Remove unused sections (-Xlinker --gc-sections)” option in the “GNU RISC-V Cross C Linker > General” configuration to reduce the compile size.

- No common uninitialized (-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 not to check this option.
- Do not inline functions (-fno-inline-functions)
If you check 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 not to check 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.

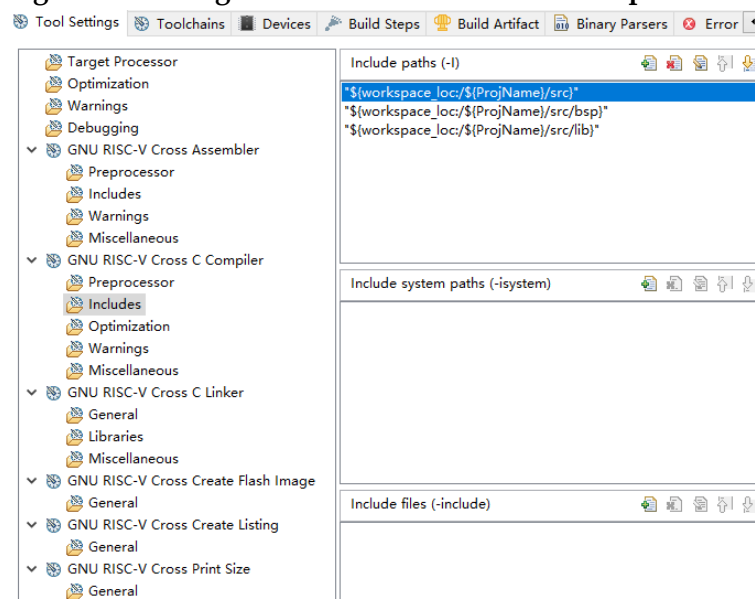
2.2.3 Configure GNU RISC-V Cross C Compiler > Includes

Select “GNU RISC-V Cross C Compiler > Includes > Include paths (-I)” to configure C header file path, as shown in Figure 2-6.

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

- “\${workspace_loc}/\${ProjName}/src”
- “\${workspace_loc}/\${ProjName}/src/bsp”
- “\${workspace_loc}/\${ProjName}/src/lib”

Figure 2-6 Configure GNU RISC-V Cross C Compiler > Includes



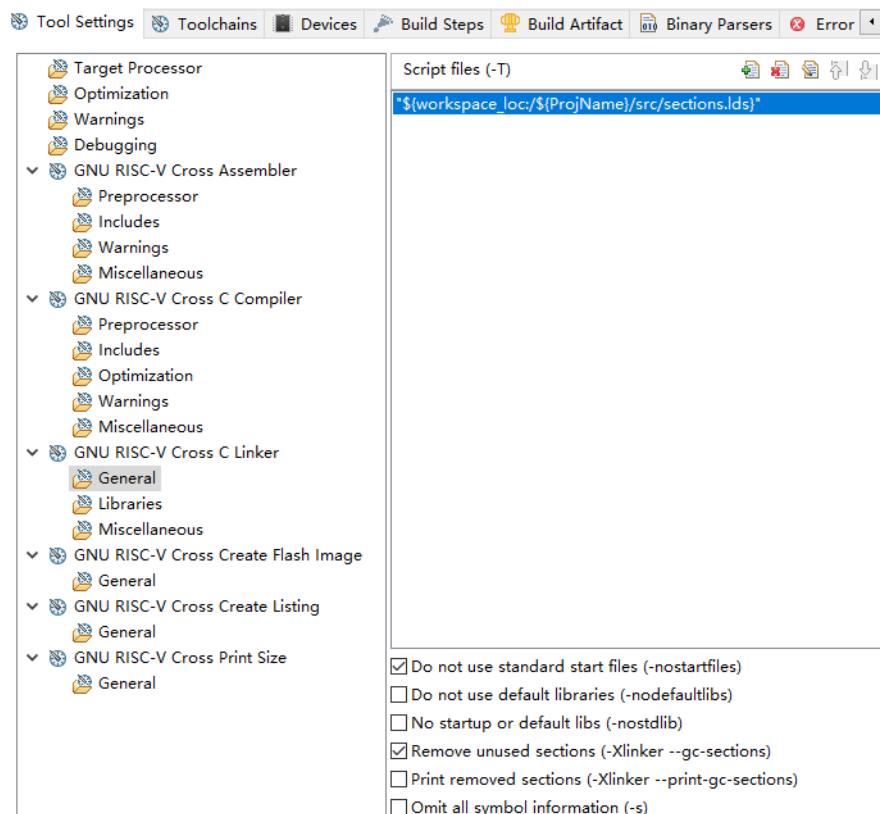
2.2.4 Configure GNU RISC-V Cross C Linker

Select “GNU RISC-V Cross C Linker > General > Script files (-T)” to configure Gowin_PicoRV32 Flash linker “sections.lds”, “sections_xip.lds”, or “sections_debug.lds”, as shown in Figure 2-7.

Select “ITCM > Boot Mode” in the IP Core Generator of Gowin_PicoRV32 hardware design:

- If Boot Mode is configured as “MCU boot from external Flash and run in ITCM” or “MCU boot and run in ITCM”, then select “sections.lds” as Flash linker, for example, “\${workspace_loc}/\${ProjName}/src/sections.lds”.
- If Boot Mode is configured as “MCU boot and run in external Flash”, then select “sections_xip.lds” as Flash linker, for example, “\${workspace_loc}/\${ProjName}/src/sections_xip.lds”.
- If Boot Mode is configured as “MCU boot from external Flash and run in ITCM” or “MCU boot and run in ITCM” and implemented software to debug on-line, then select “sections_debug.lds” as Flash linker, for example, “\${workspace_loc}/\${ProjName}/src/sections_debug.lds”.

Figure 2-7 Configure 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 check.
- 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 “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 linker, and options for specifying system library connections (such as “-static-libgcc” or “-shared-libgcc”) are ignored.
 - Please do not check 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 compile size.
- Print removed sections (-Xlinker-print-gc-sections)

When “Remove unused sections (-Xlinker-gc-sections)” is checked, 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!

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

2.2.5 Configure GNU RISC-V Cross Create Flash Image

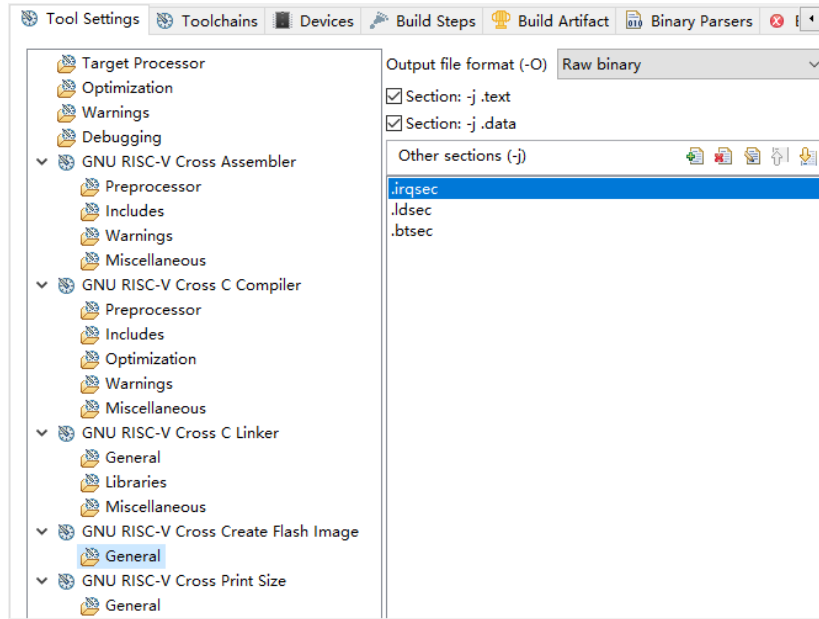
Select “GNU RISC-V Cross Create Flash Image > General” options to configure, as shown in Figure 2-8.

Taking reference design in SDK for an instance, the configuration is as follows.

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

If you have custom sections in your project 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

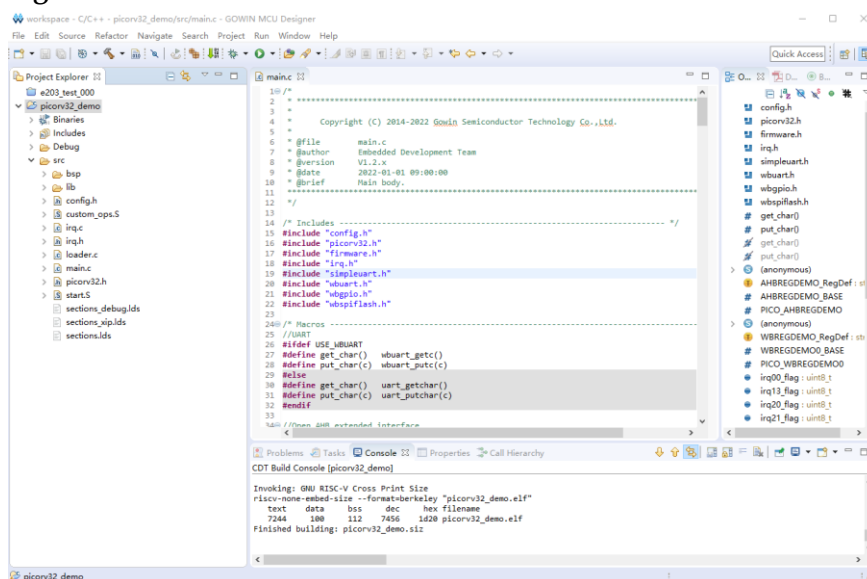
Figure 2-8 Configure GNU RISC-V Cross Create Flash Image

2.3 Build

Select "ITCM > Boot Mode" in the IP Core Generator of Gowin_PicoRV32 hardware design to configure "config.h" to startup parameter macro definition 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 template project configuration, user configuration, and encoding, select the build button "🔧" on the tool bar to build BIN file, as shown in Figure 2-9.

Figure 2-9 Build

2.4 Download

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 On-line Debug

After the Gowin_PicoRV32 software design BIN file is downloaded, if you have problems with the software application design, you can connect the development board with Olimex debug emulator (or other debug emulator supported by RISC-V instruction set architecture processor) to debug the current MCU software design on-line.

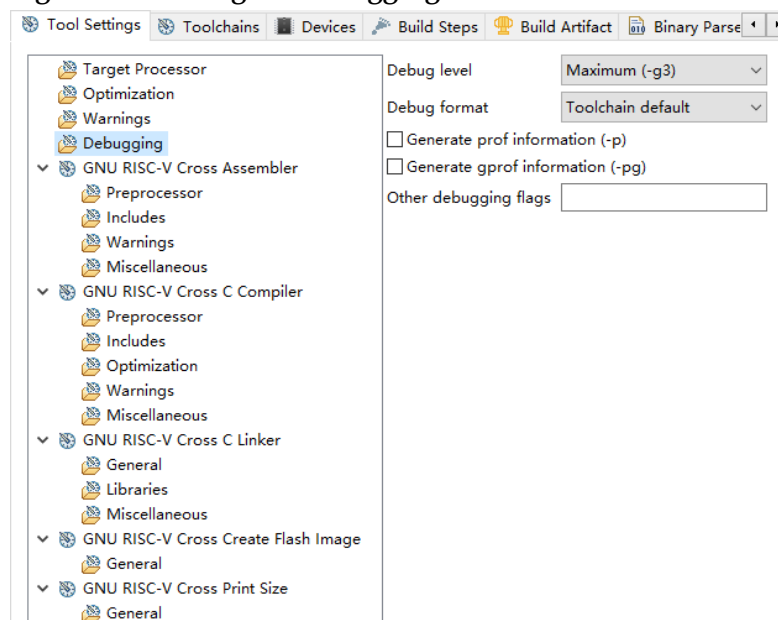
The software debug flow for Gowin_PicoRV32 includes:

- Configure software debug level
- Configure and debug Flash Linker
- Configure software debug option
- Connect debug emulator
- Start software debug

2.5.1 Configure Software Debug Level

In the “Project Explorer” view, select the current project, right click to select “Properties > Debugging > Debug level” option, as shown in Figure 2-10.

Figure 2-10 Configure Debugging



Debug level configuration includes the following options:

- None: no debug level configuration
- Minimal (-g1): minimal debug level configuration

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

2.5.2 Configure and Debug Flash Linker

When debugging Gowin_PicoRV32 software design on-line, use the Flash linker "functions_debug.lids" for debugging instead. See [2.2.4 Configure GNU RISC-V Cross C Linker](#) for the configuration of Flash linker. Select "GNU RISC-V Cross C Linker > General > Script files (-T)" to configure to "\${workspace_loc}/\${ProjName}/src/sections_debug.lids".

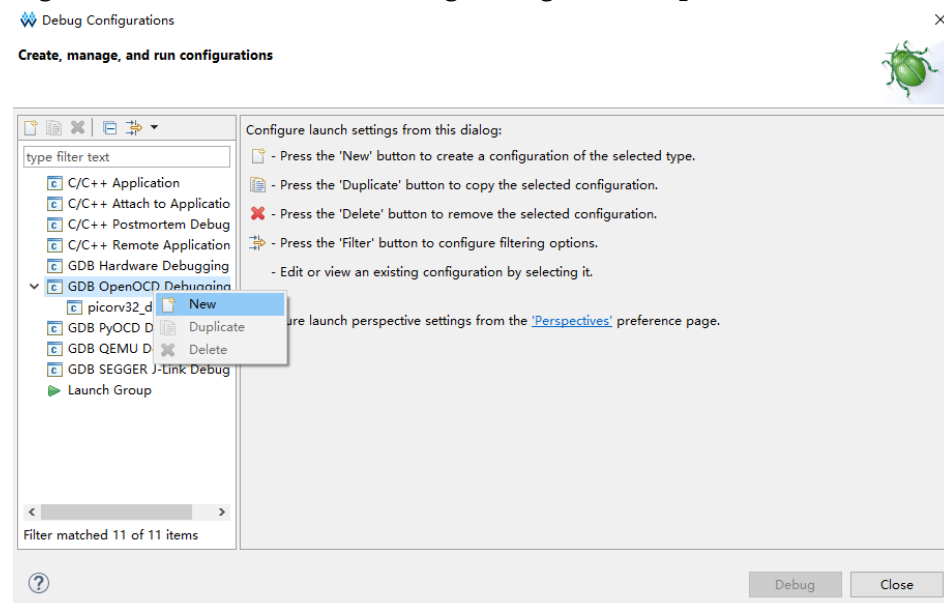
After Flash linker is configured and debugged, see [2.3 Build](#) to build software design BIN file.

After the software is built, see [2.4 Download](#) to download the software design BIN file.

2.5.3 Configure Software Debug Option

Select "Run > Debug Configurations > GDB OpenOCD Debugging" option and right-click "New" option to create the debug configuration options of current project, as shown in Figure 2-11.

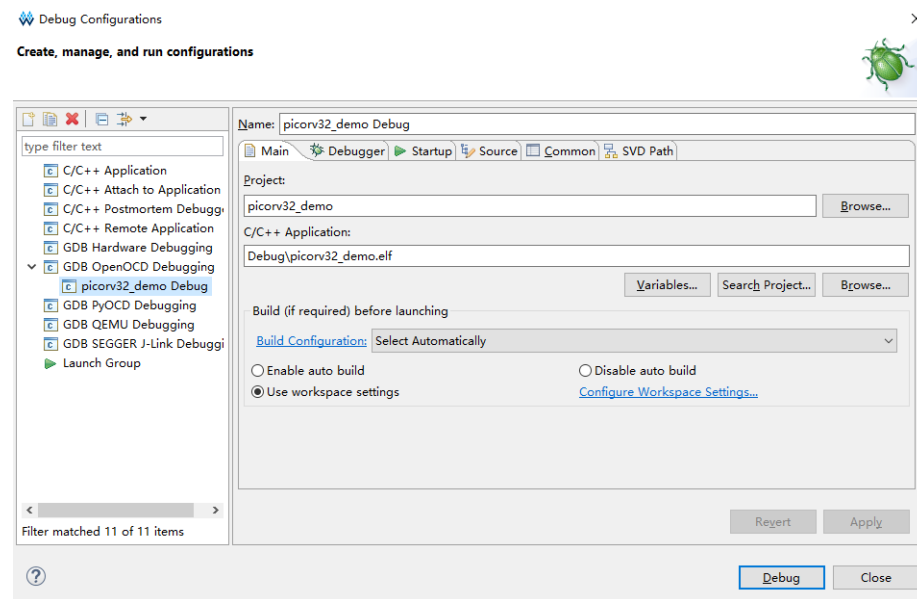
Figure 2-11 Create Software Debug Configuration Option



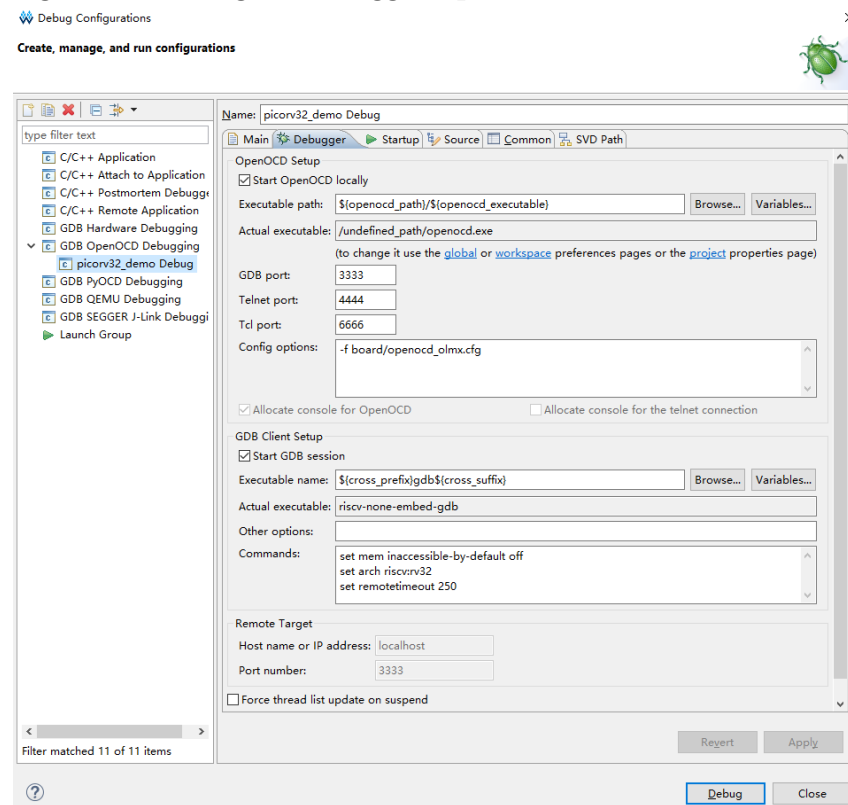
Select the created debug configuration options and configure "Main", "Debugger", "Startup" options.

Configure Main Option

Select "Main" option to configure the "Project" and "C/C++ Application" parameters of current project, as shown in Figure 2-12.

Figure 2-12 Configure Main Option**Configure Debugger Option**

Select “Debugger” option to configure the “OpenOCD” and “GDB” parameters, as shown in Figure 2-13.

Figure 2-13 Configure Debugger Option

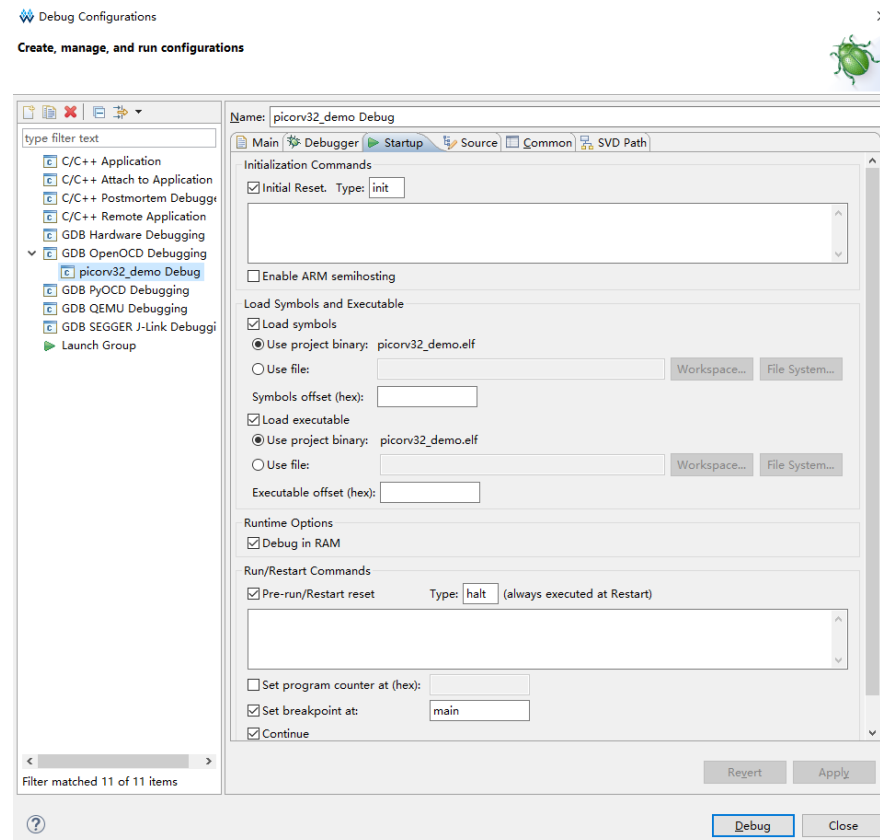
- **Configure OpenOCD Config Option**
This option is configured as “-f board/openocd_olmx.cfg”, which specifies the debug emulator configuration file to be used, and GMD uses Olimex emulator by default.

- Configure GDB Commands Option
 - set mem inaccessible-by-default off
 - set arch riscv:rv32 (to specify RISC-V instruction set architecture)
 - set remotetimeout 250 (to prevent protocol timeout)

Configure Startup Option

Select “Startup” option to configure on-line debug options, as shown in Figure 2-14.

Figure 2-14 Configure Startup Option



- Configure Initialization Commands Option
 - Please check the “Initial Reset” option and configure the “Type” as "init", so that the initialization can be executed automatically when debugging on-line.
 - Please do not check the “Enable ARM semihosting” function which Gowin_PicoRV32 does not support.
- Configure Load Symbols and Executable Option
 - Please check the “Load symbols” option and select “Use project binary” option.
 - Please check the “Load executable” option and select “Use project binary” option.
- Configure Runtime Option

Please check the “Debug in RAM” option to read the software design BIN file and write it to the instruction memory ITCM when executing

on-line debugging.

- **Configure Run/Restart Commands Option**
 - Please check the “Pre-run/Restart reset” option and configure the “Type” as "halt", so that it can pause automatically when debugging on-line.
 - Please check 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 check the “Continue” option to automatically run to the breakpoint position and pause when executing on-line debug.

2.5.4 Connect Debug Emulator

Take Olimex debug emulator for an instance.


Connect the development board with Olimex debug emulator (or other debug emulator supported by RISC-V instruction set architecture processor)

Select “Olimex arm-usb-tiny-h” for Olimex debug emulator and the link is: <https://www.olimex.com/Products/ARM/JTAG/ARM-USB-TINY-H/>.

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

Connect the TDI, TDO, TMS, and TCK pins of Olimex debug emulator to the development board in the order of the standard JTAG interface. VREF pin is connected to 3.3V, one of the 4/6/8/.../20 GND pin need to be connected only.

2.5.5 Start On-line Debug

Click debug “” on the tool bar to execute on-line debug. The debug interface includes:

- **Code area:** you can view C code and compiled code, set breakpoints, run programs, etc.
- **Variable area:** you can view the values of general-purpose registers or the values of program variables.
- **Compiled instruction area:** you can view the instructions in the current instruction memory ITCM and the currently executing instructions.
- **Control area:** you can control the process of on-line debug, set breakpoints, etc.

Debug Interrupt handler

Gowin_PicoRV32 is an area-optimized RISC-V instruction set architecture processor with an easy-to-use interrupt control system design not supporting hardware interrupt nesting and interrupt priority setting.

Therefore, MCU cannot respond to external interrupt requests when on-line debug is being executed and the program is in a paused state where operations such as single-step running, breakpoint running, etc. are executed to run the program to a certain line or instruction and pause at

that position.

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 GOWIN MCU Designer (V1.1) software environment. Access the following reference designs via the [link](#).

Gowin_PicoRV32\ref_design\MCU_RefDesign\GMD_RefDesign\picorv32_demo.

