User Manual

Software IDE Quick Start Guide

Eclipse / Visual Studio Code

Document: SW-IDE-QSG
July 2022, Version 1.3

Cobham Gaisler AB
Kungsgatan 12 | SE-41119 Gothenburg | Sweden
+46 31 7758650 | www.caes.com/Gaisler

Table of Contents

1. Introduction

This document describes how to setup a software editor or IDE together with compiler toolchains and GRMON or TSIM to be able to compile applications and/or enable C/C++ level debugging of LEON and NOEL-V applications.

1.1. Software tools

1.1.1. Editor/IDE

The following editors or IDEs are supported.

Eclipse IDE

The guide is based on Eclipse IDE for C/C++ Developers 2019-09, but will most likely work with other versions as well as long as all the necessary plugins are available.

The following Eclipse plugins are used in this guide, which are included in the Eclipse IDE for C/C++ Developers package.

- Eclipse C/C++ Development Tools (org.eclipse.cdt)
- CDT GCC Cross Compiler Support (org.eclipse.cdt.build.crossgcc)
- Eclipse GDB Hardware Debugging Plug-in (org.eclipse.cdt.debug.gdbjtag)

Visual Studio Code

The guide is based on Visual Studio Code 1.56.2, but will most likely work with other versions as well as long as all the necessary plugins are available.

The example project used in this guide is available for download.
https://gaisler.com/anonftp/lide/vscode_hello.zip

This example will use GNU Make to build the application and assumes that the user has knowledge about the GNU Make and Makefiles. It also requires that GNU Make is installed on the host computer. Other ways to build advanced applications are supported by Visual Studio Code, but not covered by this guide.

The following Visual Studio Code plugins are used in this guide:

- C/C++ by Microsoft (ms-vscode.cpptools)
- Native Debug by Webfreak (webfreak.debug)

1.1.2. GCC Toolchain for the LEON or the NOEL-V architecture

The GCC toolchain for the LEON architecture is required to be able to compile and link applications. Please see documentation of the toolchain for installation instructions.

The following toolchains are supported.

- Bare C cross-compiler (BCC 1.0.x, BCC 2.0.x)
- RTEMS LEON/ERC32 Cross-Compiler System (RCC 1.2.x, RCC 1.3.x)

1.1.3. GDB Server for the remote target

To be able to upload an application to a target system and run it, a GDB server compatible software must be available. Please see the user manual of the software for installation instructions.

The following programs are supported.

- GRMON 3 – a debug monitor for the LEON and NOEL-V processor
- GRMON 2 – a debug monitor for the LEON and NOEL-V processor
- TSIM 3 – a SPARC architecture simulator
- TSIM 2 – a SPARC architecture simulator

1.2. References

Eclipse website
[https://www.eclipse.org/]

Visual Studio Code website
[https://code.visualstudio.com/]

Make for Windows website
[http://gnuwin32.sourceforge.net/packages/make.htm]

GRMON download website
[https://www.gaisler.com/index.php/downloads/debug-tools]

TSIM download website
[https://www.gaisler.com/index.php/downloads/simulators]

BCC Download website
[https://www.gaisler.com/index.php/downloads/compilers]

RCC 1.2 (RTEMS 4.10) Download site
[https://www.gaisler.com/index.php/downloads/compilers]

RCC 1.3 (RTEMS 5) Download site
[https://www.gaisler.com/index.php/downloads/compilers]

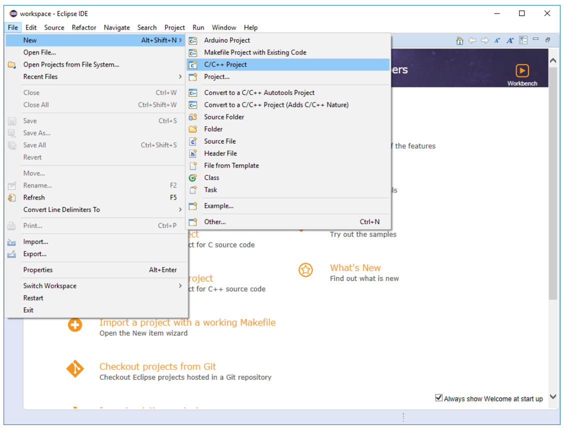
2. Projects

2.1. Eclipse IDE

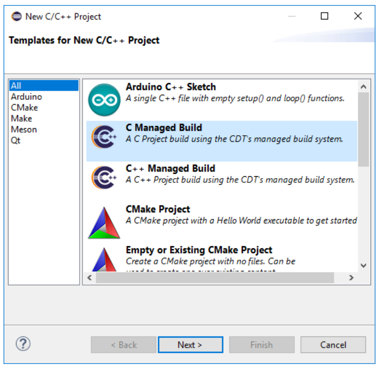
2.1.1. Creating a new project

This section describes the steps needed to create new project with a custom GCC based toolchain using the CDT GCC Cross Compiler plugin.

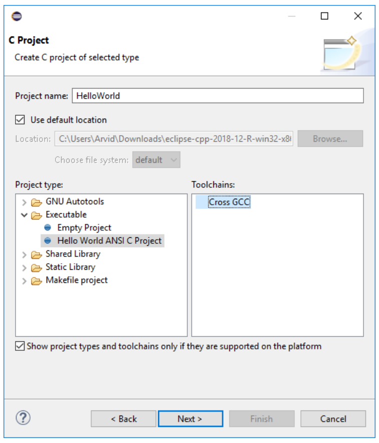
1. In the menu, select
   `File -> New -> C/C++ Project`



2. Select the **`C Managed Build`** template.
   Note that there are other valid templates at this stage, but they are not covered by this guide.
3. Press **`Next`** to continue.


New C/C++ Project
Templates for New C/C++ Project
All
Arduino
CMake
Make
Meson
Qt
Arduino C++ Sketch
A single C++ file with empty setup() and loop() functions.
C Managed Build
A C Project build using the CDT's managed build system.
C++ Managed Build
A C++ Project build using the CDT's managed build system.
CMake Project
A CMake project with a Hello World executable to get started
Empty or Existing CMake Project
Create a CMake project with no files. Can be
< Back
Next >
Finish
Cancel

4. Enter your the name of your project at the **`Project name`** field.
5. Under **`Project type`**, select **`Hello World ANSI C Project`**
6. Under Toolchains, select **`Cross GCC`**.
7. Press **`Next`** to continue.


C Project

Create C project of selected type

Project name: HelloWorld

☑ Use default location

Location: C:\Users\Arvid\Downloads\eclipse-cpp-2018-12-R-win32-x8

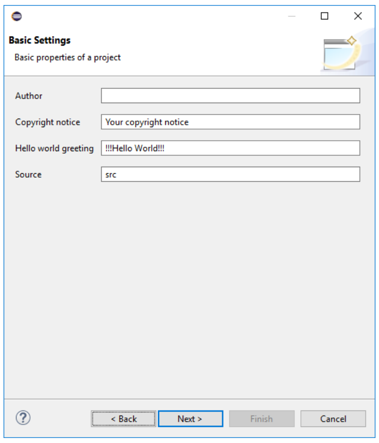
Choose file system: default

Project type: Hello World ANSI C Project

Toolchains: Cross GCC

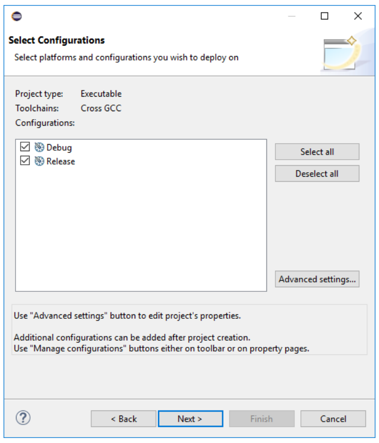
☑ Show project types and toolchains only if they are supported on the platform

8. Customize any inputs as you please.
9. Press **`Next`** to continue.


Basic Settings
Basic properties of a project

Author:
Copyright notice: Your copyright notice
Hello world greeting: !!!Hello World!!!
Source: src

10. Press **`Advanced settings...`**.


Select Configurations
Select platforms and configurations you wish to deploy on

Project type: Executable
Toolchains: Cross GCC
Configurations:
☑ Debug
☑ Release

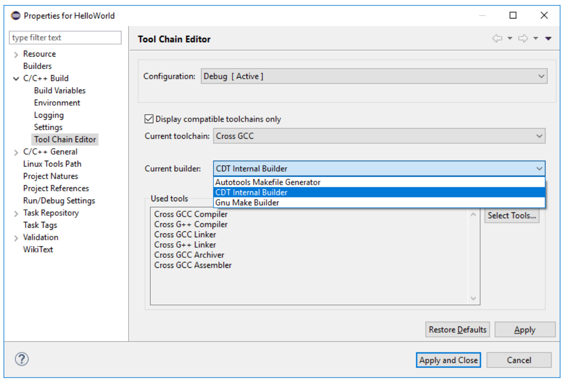
Use "Advanced settings" button to edit project's properties.

Additional configurations can be added after project creation.
Use "Manage configurations" buttons either on toolbar or on property pages.

The following steps are optional. They are recommended for Windows users and other systems where the program `make` is not available.

Jump to step 16 to skip these steps.

11. In the settings tree, select **`C/C++ Build -> Tool Chain Editor`**
12. Set the **`Configuration`** to **`Debug`**.
13. At the **`Current Builder`** option, select **`CDT Internal Builder`**.
14. Set the **`Configuration`** to **`Release`**.
15. At the **`Current Builder`** option, select **`CDT Internal Builder`**.


Properties for HelloWorld

Tool Chain Editor

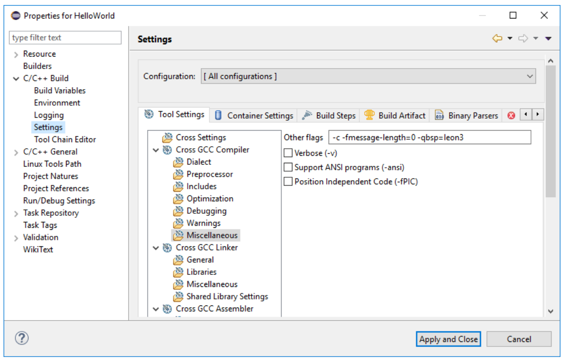
Configuration: Debug [Active]
☑ Display compatible toolchains only
Current toolchain: Cross GCC
Current builder: CDT Internal Builder

Used tools:
Cross GCC Compiler
Cross G++ Compiler
Cross GCC Linker
Cross G++ Linker
Cross GCC Archiver
Cross GCC Assembler

The following steps are optional. They are needed to setup the compiler and linker flags for the target hardware that are recommended by the toolchain documentation, i.e. -qbsp, -mfix-*, -mcpu etc. See toolchain documentation for more information.

Jump to step 21 to skip these steps.

16. In the settings tree, select **`C/C++ Build -> Settings`**
17. Set the **`Configuration`** to **`All configurations`**.
18. To add compiler flags, select **`Cross GCC Compiler -> Miscellaneous`** and the flags in the field **`Other flags`**.



Properties for HelloWorld

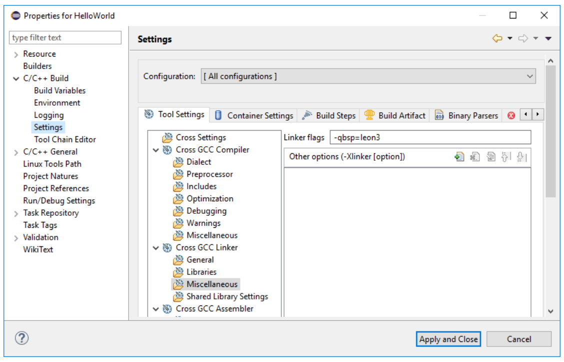
Settings

Configuration: [All configurations]

Other flags: -c -fmessage-length=0 -qbsp=leon3

☐ Verbose (-v)
☐ Support ANSI programs (-ansi)
☐ Position Independent Code (-fPIC)

19. To add linker specific flags, select **`Cross GCC Linker -> Miscellaneous`** and the flags in the field **`Linker flags`**.



Properties for HelloWorld

Settings

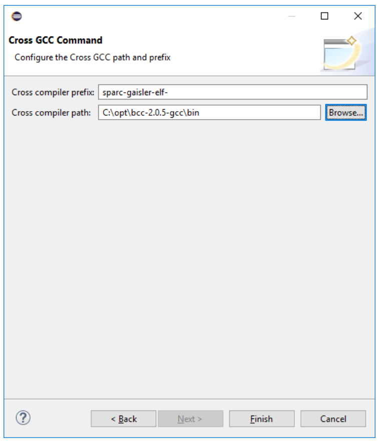
Configuration: [All configurations]

Linker flags: -qbsp=leon3

Other options (-Xlinker [option])

20. Press **`Apply and Close`** to close the advanced settings dialog.
21. Press **`Next`** to continue.

22. At the field **`Cross compiler prefix`**, enter the prefix of the toolchain, including the trailing hyphen.
    The prefix is the string that is in front of all the executable names in the `bin` folder of the toolchain.
23. Enter the **`Cross compiler path`** by pressing the **`Browse`**. Locate and choose the `bin` folder of the toolchain.
24. Press **`Finish`** to continue.



Cross GCC Command

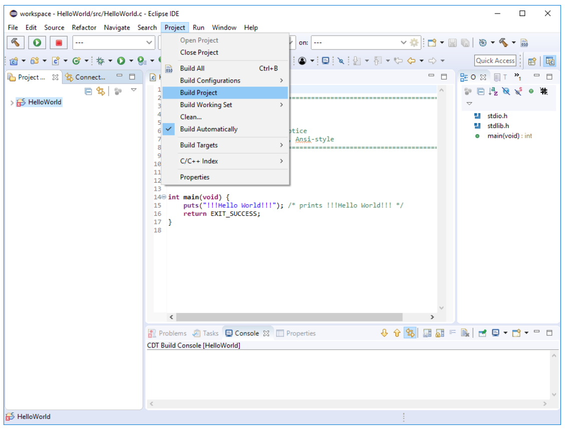
Configure the Cross GCC path and prefix

Cross compiler prefix: sparc-gaisler-elf-

Cross compiler path: C:\opt\bcc-2.0.5-gcc\bin

2.1.2. Building a project

To build the project, select menu **`Project -> Build Project`**.



```
int main(void) {
	puts("!!!Hello World!!!"); /* prints !!!Hello World!!! */
	return EXIT_SUCCESS;
}
```

2.2. Visual Studio Code

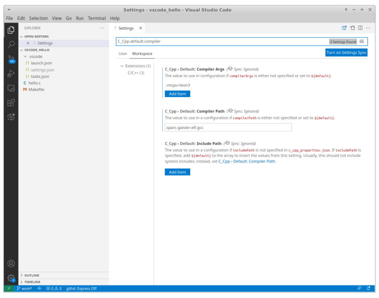
2.2.1. Creating a new project

1. Create a new folder using your regular OS tools. (i.e. Explorer in Windows).
2. Start Visual Studio Code.
3. Select the menu
   `File -> Open Folder` and locate the folder.

The following steps are optional. They will configure the Intellisense to find the system include paths using GCC.

Jump to step 9 to skip these steps.

4. In the menu, select
   `File -> Preferences -> Settings`
5. Select **`Workspace`** tab
6. Search for **`C_Cpp.default.compiler`**
7. Edit **`C_Cpp > Default: Compiler Path`** and add the path to GCC. The full path to GCC is needed if it hasn't been added to the environment variable PATH.
8. Edit **`C_Cpp > Default: Compiler Args`** and add compiler flags for the target hardware that are recommended by the toolchain documentation, i.e. -qbsp, -mfix-*, -mcpu etc. See toolchain documentation for more information.



C_Cpp.default.compiler

C_Cpp › Default: Compiler Args (Sync: Ignored)
The value to use in configuration if `compilerArgs` is either not specified or set to `${default}`.
-mcpu=leon3

C_Cpp › Default: Compiler Path (Sync: Ignored)
The value to use in a configuration if `compilerPath` is either not specified or set to `${default}`.
sparc-gaisler-elf-gcc

C_Cpp › Default: Include Path (Sync: Ignored)
The value to use in a configuration if `includePath` is not specified in `c_cpp_properties.json`. If `includePath` is specified, add `${default}` to the array to insert the values from this setting. Usually, this should not include system includes; instead, set C_Cpp › Default: Compiler Path.

9. In the menu, select
   `File -> New File`
10. Enter the "Hello World" code and save as **`hello.c`**

```
#include <stdio.h>

int main(void)
{
        puts("Hello World!");
        return 0;
}
```

This example will use GNU Make to build the application and assumes that the user has knowledge about the GNU Make and Makefiles. It also requires that GNU Make is installed on the host computer. Other ways to build advanced applications are supported by Visual Studio Code, but not covered by this guide.

11. In the menu, select
    `File -> New File`
12. Enter the make script code and save as
    `Makefile` (no extension)

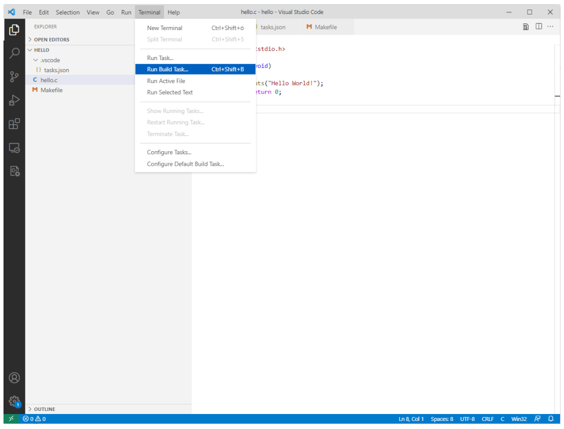
```
ifeq ($(OS),Windows_NT)
CC=sparc-gaisler-elf-gcc.exe
RM=del
else
CC=sparc-gaisler-elf-gcc
RM=rm -f
endif

CFLAGS=-g -O0 -Wall

all: hello.elf

clean:
 $(RM) hello.elf

hello.elf: hello.c
 $(CC) $(CFLAGS) -o $@ $<
```

13. Create a new, or open the existing, tasks.json by selecting the menu **`Terminal -> Configure tasks...`**
    The first time you open this menu in a new folder you will get a drop down list, select **`Create tasks.json from template`** and then **`Others`**.
14. Remove the automatically generated JSON code and replace it with the code below.
15. Verify that the **`command`** entry is correct. It should specify the shell command to start GNU make.
16. Save the file.

```
{
        // See https://go.microsoft.com/fwlink/?LinkId=733558
        // for the documentation about the tasks.json format
        "version": "2.0.0",
        "tasks": [
                {
                        "label": "Build All",
                        "type": "shell",
                        "windows": {
                                "command": "& 'C:/Program Files (x86)/GnuWin32/bin/make.exe'"
                        },
                        "linux": {
                                "command": "make"
                        },
                        "group": {
                                "kind": "build",
                                "isDefault": true
                        },
                        "problemMatcher": [
                                "$gcc"
                        ]
                }
        ]
}
```

2.2.2. Building a project

To build the project, select menu **`Terminal -> Run Build Task...`**.

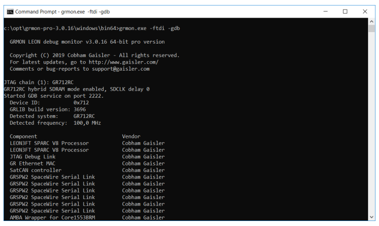


```
<stdio.h>
void)
uts("Hello World!");
eturn 0;
```

3. Debugging an application

3.1. Preparing the remote target

Before the debugging session in the IDE is started, you must start TSIM/GRMON and enable it to act as a GDB server. This is achieved by adding **`-gdb [port]`** to the command line, in addition to your normal command line options.



Command Prompt - grmon.exe -ftdi -gdb

```
c:\opt\grmon-pro-3.0.16\windows\bin64>grmon.exe -ftdi -gdb

  GRMON LEON debug monitor v3.0.16 64-bit pro version

  Copyright (C) 2019 Cobham Gaisler - All rights reserved.
  For latest updates, go to http://www.gaisler.com/
  Comments or bug-reports to support@gaisler.com

JTAG chain (1): GR712RC
GR712RC hybrid SDRAM mode enabled, SDCLK delay 0
Started GDB service on port 2222.
  Device ID:           0x712
  GRLIB build version: 3696
  Detected system:     GR712RC
  Detected frequency:  100,0 MHz

  Component                            Vendor
  LEON3FT SPARC V8 Processor           Cobham Gaisler
  LEON3FT SPARC V8 Processor           Cobham Gaisler
  JTAG Debug Link                      Cobham Gaisler
  GR Ethernet MAC                      Cobham Gaisler
  SatCAN controller                    Cobham Gaisler
  GRSPW2 SpaceWire Serial Link         Cobham Gaisler
  GRSPW2 SpaceWire Serial Link         Cobham Gaisler
  GRSPW2 SpaceWire Serial Link         Cobham Gaisler
  GRSPW2 SpaceWire Serial Link         Cobham Gaisler
  GRSPW2 SpaceWire Serial Link         Cobham Gaisler
  GRSPW2 SpaceWire Serial Link         Cobham Gaisler
  AMBA Wrapper for Core1553BRM         Cobham Gaisler
```

If you want to forward the UART input/output from the system you need to start GRMON3 with the commandline option -ucli [*]. GRMON will setup it's terminal to handle stdin/stdout, therefor the GRMON prompt will not be shown, nor will any commands be accepted.

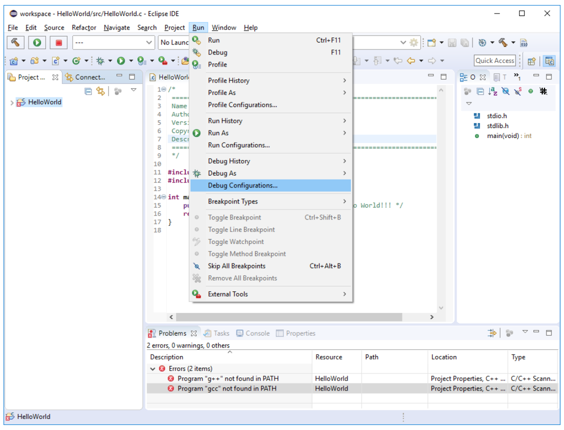
[*]The -ucli was added in GRMON 3.1.2. See FAQ Q4 for more information.

3.2. Eclipse IDE

3.2.1. Configuring a debug target

This section describes how to setup a new debug configuration for an application, which includes connecting and uploading the application to the target board, using the Eclipse GDB Hardware Debugging Plug-in.

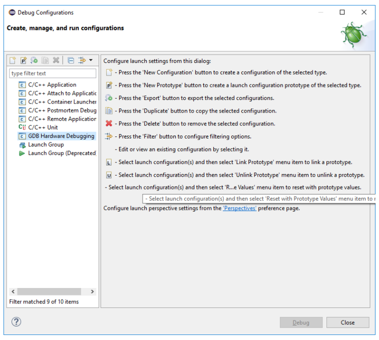
1. In the menu, select
   `Run -> Debug Configurations...`



2 errors, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (2 items)				
Program "g++" not found in PATH	HelloWorld		Project Properties, C++ ...	C/C++ Scann...
Program "gcc" not found in PATH	HelloWorld		Project Properties, C++ ...	C/C++ Scann...

2. Double-click on **`GDB Hardware Debugging`** to create a new configuration

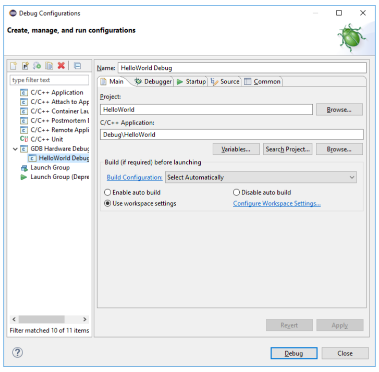


Configure launch settings from this dialog:
- Press the 'New Configuration' button to create a configuration of the selected type.
- Press the 'New Prototype' button to create a launch configuration prototype of the selected type.
- Press the 'Export' button to export the selected configurations.
- Press the 'Duplicate' button to copy the selected configuration.
- Press the 'Delete' button to remove the selected configuration.
- Press the 'Filter' button to configure filtering options.
- Edit or view an existing configuration by selecting it.
- Select launch configuration(s) and then select 'Link Prototype' menu item to link a prototype.
- Select launch configuration(s) and then select 'Unlink Prototype' menu item to unlink a prototype.
- Select launch configuration(s) and then select 'R...e Values' menu item to reset with prototype values.
- Select launch configuration(s) and then select 'Reset with Prototype Values' menu item to

Configure launch perspective settings from the 'Perspectives' preference page.

Filter matched 9 of 10 items

3. Verify that the **`Project`** name is correct.
4. Verify that the **`C/C++ Application`** path is set to the application that you want to debug.



Name: HelloWorld Debug

Project: HelloWorld

C/C++ Application: Debug\HelloWorld

Build (if required) before launching

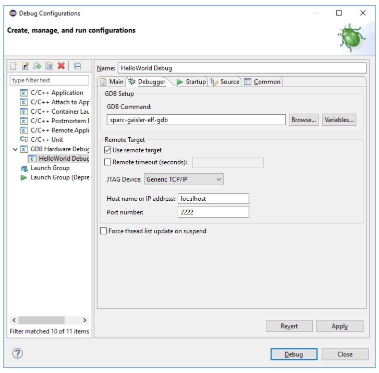
Build Configuration: Select Automatically

( ) Enable auto build ( ) Disable auto build

[✓ Use workspace settings] Configure Workspace Settings...

Filter matched 10 of 11 items

5. Open the **`Debugger`** tab.
6. Setup the path to the GDB executable at **`GDB command`**. If the environment variable PATH is setup for the toolchain, or if the project was created using the Cross GCC plug-in, then it should be sufficient to enter the name of the GDB executable. Otherwise press **`Browse`** and locate the GDB executable in the toolchain `bin` folder.
7. The drop-down list **`JTAG Device`** should be set to **`Generic TCP/IP`**.
8. Enter the **`Port number`** that GRMON/TSIM has opened.



Name: HelloWorld Debug

GDB Setup

GDB Command: sparc-gaisler-elf-gdb

Remote Target

[✓] Use remote target

[ ] Remote timeout (seconds):

JTAG Device: Generic TCP/IP

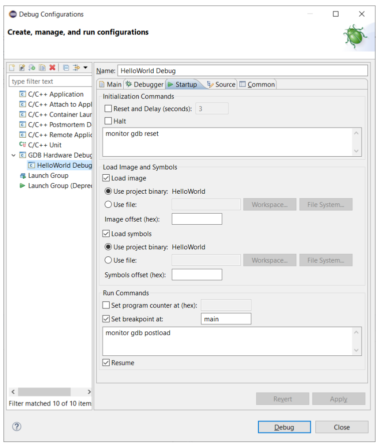
Host name or IP address: localhost

Port number: 2222

[ ] Force thread list update on suspend

Filter matched 10 of 11 items

9. Open the **`Startup`** tab.
10. Add "**`monitor gdb reset`**" to the first large text box [*]
11. Verify that **`Load Image`** is checked.
12. Verify that **`Load Symbols`** is checked.
13. Check **`Set breakpoint at`** and enter **`main`** in the field.
14. Add "**`monitor gdb postload`**" to the second large text box [†]
15. Check **`Resume`**
16. Press **`Debug`** to start the application.


Debug Configurations

Create, manage, and run configurations

Name: HelloWorld Debug

Main | Debugger | Startup | Source | Common

Initialization Commands
- ☐ Reset and Delay (seconds): 3
- ☐ Halt

```
monitor gdb reset
```

Load Image and Symbols
- ☑ Load image
  - ● Use project binary: HelloWorld
  - ○ Use file:
  - Image offset (hex):
- ☑ Load symbols
  - ● Use project binary: HelloWorld
  - ○ Use file:
  - Symbols offset (hex):

Run Commands
- ☐ Set program counter at (hex):
- ☑ Set breakpoint at: main

```
monitor gdb postload
```

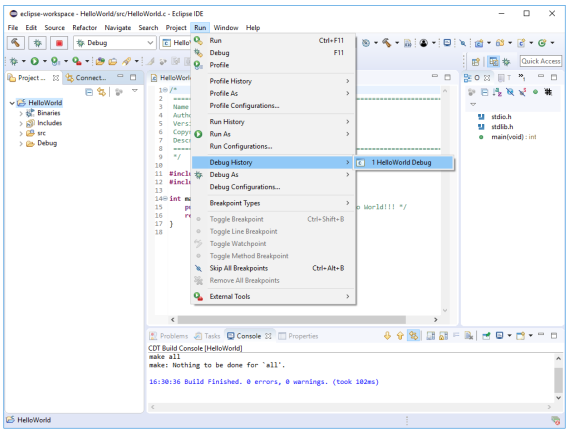
- ☑ Resume

[*]This command was added in GRMON 3.1.2, TSIM 2.0.65. For older versions. See FAQ Q6 for more information.
[†]This command was added in GRMON 3.1.2, TSIM 2.0.65. For older versions, See FAQ Q7 for more information.

3.2.2. Starting an application

After the debug configuration has been created and run once, it can be started again by selecting it under the menu **`Run -> Debug History`**.

If the debug configuration is missing in history, then open the menu **`Run -> Debug Configurations...`**. All previousely created configurations are listed as a subitem to **`GDB Hardware Debugging`**. Select your configuration and press **`Debug`** to start the application.



CDT Build Console [HelloWorld]

```
make all
make: Nothing to be done for `all'.

16:30:36 Build Finished. 0 errors, 0 warnings. (took 102ms)
```

3.3. Visual Studio Code

3.3.1. Configuring a debug target

The following Visual Studio Code plugins are used in this guide:

- C/C++ by Microsoft (ms-vscode.cpptools)
- Native Debug by Webfreak (webfreak.debug)

The example project used in this guide is available for download.
https://gaisler.com/anonftp/lide/vscode_hello.zip

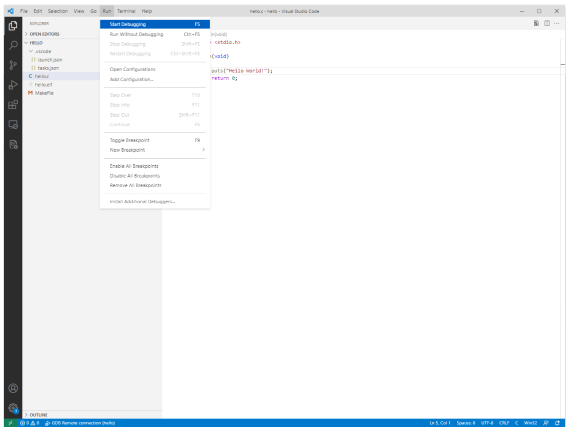
1. Create a new, or open the existing, launch.json file by selecting the menu **`Run -> Add Configuration...`**
   The first time you open this menu in a new folder you will get a drop down list, select **`GDB`**. *
2. Remove the automatically generated JSON code and replace it with the code below.
3. Verify that the **`executable`** entry is correct. It should specify the application that you want to debug.
4. Verify that the **`gdbpath`** entry is correct. It should specify the path to GDB.
5. Verify that the **`target`** entry is correct. It should specify the path to IP or hostname of the computer that runs GRMON/TSIM. The port number should also be the same as the one used by GRMON/TSIM.
6. Save the file.

```
{
        // Use IntelliSense to learn about possible attributes.
        // Hover to view descriptions of existing attributes.
        // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
        "version": "0.2.0",
        "configurations": [
                {
                        "name": "Hello",
                        "type": "gdb",
                        "request": "attach",
                        "cwd": "${workspaceFolder}",
                        "valuesFormatting": "parseText",
                        "executable": "${workspaceFolder}/hello.elf",
                        "windows": {
                                "gdbpath": "C:/opt/bcc-2.2.0-gcc/bin/sparc-gaisler-elf-gdb.exe"
                        },
                        "linux": {
                                "gdbpath": "/opt/bcc-2.2.0-gcc/bin/sparc-gaisler-elf-gdb"
                        },
                        "target": "localhost:2222",
                        "remote": true,
                        "autorun": [
                                "monitor gdb reset",
                                "load",
                                "monitor gdb postload",
                                "tbreak main"
                        ]
                }
        ]
}
```

*in older versions of VS Code you selected **`C++ (GDB/LLDB)`** and then **`Default Configuration`**).

3.3.2. Starting an application

Start the application by by select the menu **`Run -> Start Debugging`**.



```
<stdio.h>
(void)
    puts("Hello World!");
    return 0;
```

4. FAQ / TROUBLESHOOTING

Q1. After I have created my Cross GCC project, I get two errors in the Problems view: `Program "g++" not found in PATH` and `Program "gcc" not found in PATH`.

A This is because Eclipse defaults to using gcc for the indexing and the errors show up because you don't have any native GCC toolchain installed on your computer. These errors are harmless, as far as we know, but might cause Eclipse to report false positive problems.

Follow the steps below to remove the errors.

1. Open the menu **`Project -> Properties`**
2. Select **`C/C++ General -> Preprocessor Include Paths, Macros etc.`**
3. Open the tab **`Providers`**.
4. Uncheck **`CDT Cross GCC Built-in Compiler settings`**
5. Press **`Apply`**
6. Re-check **`CDT Cross GCC Built-in Compiler settings`**
7. Press **`Apply and close`**

Q2. When I build my application I get the error message `Error: Program "make" not found in PATH`.

A The Cross GCC plugins uses the GNU Make builder to build the application. The GNU Make builder requiers the program make to be installed on your computer. Make can be downloaded from from the GnuWin32 project [http://gnuwin32.sourceforge.net/packages/make.htm].

Another solution is to switch to the CDT Internal builder instead, which is independant of make. Follow the steps below to switch build system.

1. Open the menu **`Project -> Properties`**
2. Select **`C/C++ Build -> Tool Chain Editor`**
3. At **`Current Builder`** select **`CDT Internal Builder`** from the drop down list.
4. Repeat the previous step for each **`configuration`**.
5. Press the button **`Apply and Close`** to apply the changes and close the dialog.

Q3. Why don't I get any output in Eclipse when I start GRMON with "-u"?

A GRMON does forward all output to Eclipse, but Eclipse does not print it. A workaround is to start GRMON with "-ucli" instead, which print all the output in the GRMON console instead. See Q4 for additional information.

Q4. Why do I get the error message "`ERROR! Unknown option (-ucli)`" when starting GRMON?

A This option was added in GRMON 3.1.2. Earlier versions of GRMON does not support I/O forwarding when using Eclipse.

Q5. When launching an debug configuration I get the error message: `"Failed to execute MI command: -gdb-set auto-solib-add on "Error message from debugger back end: No symbol table is loaded. Use the "file" command.` What am I doing wrong?

A Your Eclipse version is not compatible with the GDB 6.8 version that you are using. Either you need to downgrade you Eclipse version or switch to GDB 8. GDB 8 is distributed in BCC 2.1.0 but requires GRMON3 or TSIM3.

Q6. When launching an debug configuration I get the error message: `"Failed to execute MI command: monitor gdb reset".` What am I doing wrong?

A This command is available in GRMON 3.1.2 or later and TSM 2.0.65 or later. A workaround for earlier versions of GRMON is to add "`monitor reset`" and for earlier versions of TSIM2 you can leave it empty. These workarounds works in most cases.

Q7. When launching an debug configuration I get the error message: `"Failed to execute MI command: monitor gdb postload".` What am I doing wrong?

A This command is only available in GRMON 3.1.2 or later and TSM 2.0.65 or later. For earlier versions that are running a single core application you can leave this field empty. It should work in most cases.

For an SMP application, when using earlier versions of GRMON, you should add this line instead: `monitor silent for {set c 1} {$c < [llength [lindex [cpu] 1]]} {incr c} {reg pc [reg pc cpu0] npc [reg npc cpu0] cpu[set c]}`

Q8. How can I get the output from my application in a console inside Eclipse?

A Open a Local Terminal, or a Command Shell console, in Eclipse and start GRMON/TSIM from there. To open a Local Terminal:

1. Select the Terminal view
2. Click on the **`Open a Terminal`** icon on the Terminal view toolbar
3. In the **`Choose terminal`** drop-down list, select **`Local Terminal`**
4. Press **`OK`**

Q9. How do I issue a TSIM/GRMON monitor command from within Eclipse?

A In Eclipse open the GDB debugger console through the console dropdown. Issue a “`monitor <command>`”. The output of the monitor command will appear in the console of the application being debugged.

Q10. I try to run TSIM/GRMON command with the monitor command, but it only works the first time.

A Eclipse sends all output from the target to a separate console window. When you run the GDB monitor command the output goes to that console window instead of the GDB console and Eclipse will automatically select it. To run a new command select the GDB console again by clicking on the GDB process or using the console selection button.

Q11. I get the error message: `"error creating session localhost:2222: Bad file descriptor".` What am I doing wrong?

A This error message is usually caused by GRMON/TSIM not being able to launch correctly. Make sure that your launch configuration settings are correct and that GRMON/TSIM is launched with the right settings.

Q12. When the debug sessions start the following messages are printed (or similar): `No symbol "auto" in current context. monitor symbol C:\workspace\rtems-tasks\Debug\rtems-tasks Previous frame identical to this frame (corrupt stack?) The target endianness is set automatically (currently big endian)`

A These are just informational messages from GDB and can be ignored.

Q13. Why do I get the error message `"Unable to open 'trap_table_mvt.S': Unable to read file '\opt\bcc-2.2.0-gcc\src\libbcc\shared\trap\trap_table_mvt.S' (Error: Unable to resolve non-existing file '\opt\bcc-2.2.0-gcc\src\libbcc\shared\trap\trap_table_mvt.S')."`

A You have not installed the target C library source code, or installed them to a different address then the recommended. This error message can safely be ignored.

To resolve this issue either install the target C library source code to the recommended location or setup GDB to translate the path using the GDB command "`set substitue-path <from> <to>`"

Q14. Why is my breakpoints not working?

A Have you enabled the GCC option to produce debugging information (`-g`) when compiling the source files. Also disable any GCC optimization options (`-O0`).

5. Support

Questions related to Eclipse usage is not covered by the support. Please contact the Eclipse Community Forums or other online resources.

Questions related to Visual Code Editor usage is not covered by the support. See Visual Code Editor FAQ [https://code.visualstudio.com/docs/supporting/faq#_technical-support] for more information

For support related to tools, e.g. GRMON, TSIM or this guide itself, contact the support team at support@gaisler.com.

When contacting support, please identify yourself in full, including company affiliation and site name and address. Please identify exactly what product that is used, specifying if it is an IP core (with full name of the library distribution archive file), component, software version, compiler version, operating system version, debug tool version, simulator tool version, board version, etc.

The support service is only for paying customers with a support contract.

Cobham Gaisler AB
Kungsgatan 12
411 19 Gothenburg
Sweden
www.caes.com/Gaisler
sales@gaisler.com
T: +46 31 7758650
F: +46 31 421407