

Tutorial

Installation and setup for running and debugging RISC-V applications (Windows)

Author:

Saul García Rodríguez

saul.garciaarodriguez@fh-dortmund.de

Contents

1	Introduction	5
2	Target Hardware	5
2.1	Digilent Nexys A7 development board	5
2.2	Olimex ARM-USB-Tiny-H	6
2.2.1	Drivers installation	7
2.2.2	Connections to Nexys A7	7
3	Software Tools	8
3.1	Xilinx Vivado Design Suite 2023.2	8
3.1.1	Caeco files	9
3.1.2	MEM and COE files	9
3.1.2.1	Convert ELF file to MEM file with data2mem	10
3.1.2.2	Format MEM file with new-mem tool	10
3.1.2.3	Format COE file with new-coe tool	11
3.1.3	RISC-V files	11
3.1.3.1	Design Sources	11
3.1.3.2	IP Cores	12
3.1.3.3	Simulation Sources	16
3.1.3.4	Constraints	17
3.1.4	Program Device	17
3.2	Eclipse IDE for Embedded C/C++ Developers 2023-09	18
3.2.1	xPack GNU RISC-V Embedded GCC	18
3.2.1.1	Prerequisites	19
3.2.1.2	Installation	19
3.2.2	xPack Windows Build Tools	20
3.2.3	xPack OpenOCD	21
3.3	HTerm	21
4	Debugging in Eclipse IDE for Embedded C/C++ Developers	21

4.1	Create Project	22
4.2	Project Properties Configuration	28
4.2.1	C/C++ Build Settings Configuration	28
4.2.1.1	Target Processor	28
4.2.1.2	Debugging	29
4.2.1.3	GNU RISC-V Cross C Linker	30
4.2.2	MCU Paths Selection	32
4.2.2.1	Build Tools Path	32
4.2.2.2	OpenOCD Path	33
4.2.2.3	RISC-V Toolchain Path	34
4.3	Debug Configuration	35
4.3.1	Adding target to OpenOCD	36
4.3.2	Debugger Configuration	36
4.3.3	Main Configuration	37
4.4	Debugging	38
4.5	Optional: Debugging on Linux server	40
5	Verification with Python	41
	Bibliography	47

List of Figures

1	Nexys A7 components	5
2	USB-UART bridge connections	6
3	Pmod connectors	6
4	JA Pmod port pin assignment	6
5	ARM-USB-Tiny-H OLIMEX OPENOCD ARM JTAG DEBUGGER	7
6	ARM-USB-Tiny-H drivers installation with Zadig	7
7	ARM-USB-Tiny-H drivers check	7
8	Olimex JTAG connector layout	8
9	Clocking options	13
10	Output clocks configuration	14
11	Basic BRAM configuration	15
12	Port A configuration	15
13	Port B configuration	16
14	Memory initialization	16
15	Project structure	17
16	Download Eclipse IDE for Embedded C/C++ Developers	18
17	Selection of directory as workspace	22
18	Create new Embedded C/C++ project	23
19	C Managed Build selection	23
20	C Managed Build selection	24
21	Adding author name and copyright	25
22	Select Configurations	25
23	GNU	26
24	Source files in project	26
25	Linking new file to project	27
26	Selecting file	27
27	Project structure	28
28	Target Processor configuration	29
29	Debugging configuration	30

30	GNU RISC-V Cross C Linker General configuration	31
31	GNU RISC-V Cross C Linker Miscellaneous configuration	32
32	Build Tools Path selection	33
33	OpenOCD Path selection	33
34	RISC-V Toolchains Path selection	34
35	Building the project	35
36	Add target configuration file <i>airi5c.cfg</i> to OpenOCD	36
37	Configure the debbuger	37
38	Select ELF file	38
39	Debug perspective	39
40	HTerm terminal during debugging	39
41	Select the toolchain and path	40
42	Debug configuration for debug on Remote Target	41
43	Data flow in the RV32IM system architecture	42
44	Accessing to the memory using General OpenOCD commands	43
45	Calling <i>read_ecg</i> and plotting ECG data	44
46	Calling <i>caeco_dmi</i> to directly write and read to CAECO	45
47	Calling <i>caeco_mcu</i> to write and read to CAECO via RISC-V	46

1 Introduction

In this tutorial the hardware and software tools required to program a Nexys A7 and debug the RAIFES RV32IM platform is explained, as well as the installation and setup of the software tools on Windows. The RAIFES core is based on a RISC-V RV32IM processor core developed by Fraunhofer IMS. The main objective of this report is to provide information on the first steps for further analysis and development of future projects.

As use case, the Configurable Accelerator Engine for Convolution Operations (CAECO) [1] has been integrated into the Raifes RISC-V RV32IM processor as an ML intellectual property (IP) core for the evaluation of Electrocardiogram (ECG) data [2]. However, the Raifes core can be used or expanded for other purposes, e.g. cryptographic accelerators.

2 Target Hardware

The hardware required for this project is a Nexys A7 board, which is a development platform based on the Artix-7 FPGA provided by Xilinx [3], and an ARM-USB-Tiny-H programmer/debugger provided by Olimex [4].

2.1 Digilent Nexys A7 development board

The Nexys A7 board is conformed by components, in addition to the FPGA, that allow a convenient development of embedded applications. The board components used for communication between the board and the PC are circled in Figure 1. These are the USB-UART bridge shown in Figure 2 and the JA Pmod port shown in Figure 3, which is used for the JTAG interface. In section 4.4 the results of the data transmission from the FPGA to the PC through the USB-UART bridge are presented.

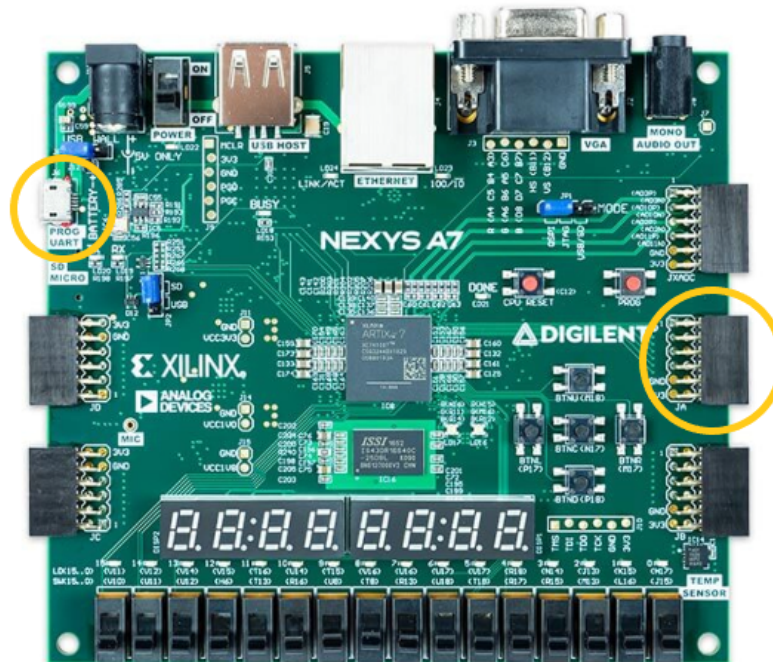


Figure 1: Nexys A7 components

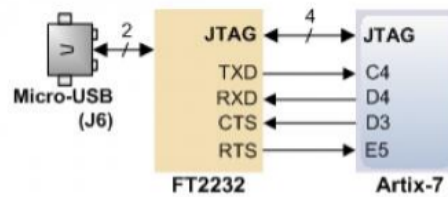


Figure 2: USB-UART bridge connections

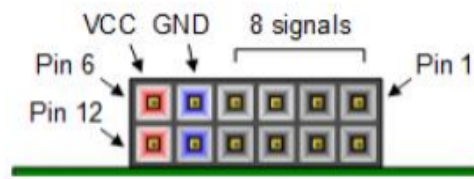


Figure 3: Pmod connectors

The JA Pmod port is the port used with the JTAG debugger for debugging in Eclipse IDE. The pin assignment for this port is shown in Figure 4

Pmod JA
JA1: C17
JA2: D18
JA3: E18
JA4: G17
JA7: D17
JA8: E17
JA9: F18

Figure 4: JA Pmod port pin assignment

2.2 Olimex ARM-USB-Tiny-H

The ARM-USB-Tiny-H is a USB FT2232-based ARM JTAG programmer/debugger controlled by a PC via OpenOCD under Windows. Eclipse IDE is the platform used in this project for debugging using the OpenOCD plug-in to control the ARM-USB-Tiny-H.



Figure 5: ARM-USB-Tiny-H OLIMEX OPENOCD ARM JTAG DEBUGGER

2.2.1 Drivers installation

In order to establish communication with the ARM-USB-Tiny-H, first the proper drivers must be installed with the *Zadig* program as presented in Figure 6 [5]. Install the WinUSB drivers for Interface 0 and Interface 1.

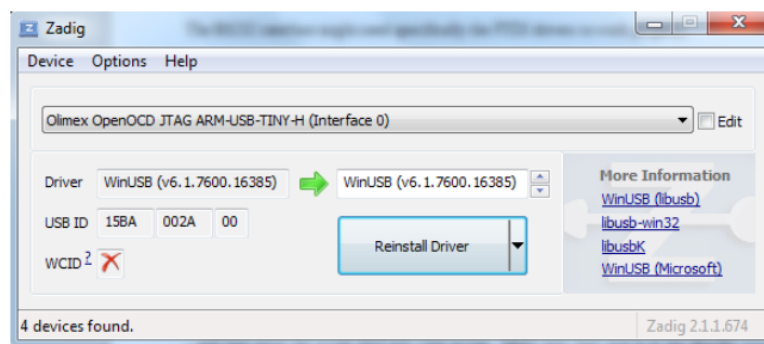


Figure 6: ARM-USB-Tiny-H drivers installation with Zadig

To verify that the drivers have been installed successfully open the Device Manager and check that both interfaces are recognised as Universal Serial Bus devices as shown in Figure 7

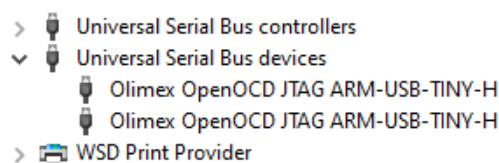


Figure 7: ARM-USB-Tiny-H drivers check

2.2.2 Connections to Nexys A7

Following the JTAG connector layout presented in Figure 8 the connections between the ARM-USB-Tiny-H with the Nexys JA Pmod port are shown in Figure 1.

The 20-pin JTAG connector of:
Olimex ARM-JTAG-TINY
Olimex ARM-JTAG-TINY-H
Olimex ARM-USB-OCD
Olimex ARM-USB-OCD-H

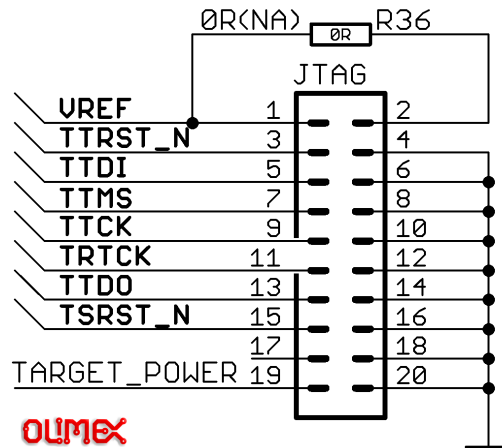


Figure 8: Olimex JTAG connector layout

JA Pmod port		ARM-USB-Tiny-H	
Pin#	Name	Name	Pin#
1	C17	TTDI	5
2	D18	TTDO	13
3	E18	TTMS	7
4	G17	TTCK	9
5	GND	GND	4
6	VCC	VREF	1
11	GND	GND	6
12	VCC	VREF	2

Table 1: Connections of JA Pmod port with ARM-USB-Tiny-H

3 Software Tools

The two main software tools used are Vivado Design Suite and Eclipse IDE. The first one is used for synthesis, implementation and to program the FPGA, whereas Eclipse is used for compilation and debugging of the code that targets the RISC-V within the FPGA.

3.1 Xilinx Vivado Design Suite 2023.2

Vivado Design Suite is a design software for AMD adaptive SoCs and FPGAs that includes design entry, synthesis, place and route, verification and simulation tools. The first step to program the FPGA is to create a new project by selecting the correct board (Nexys A7-100T for this project), if the board is missing in the catalogue follow instructions in [6].

Several files from probably different sources are required to successfully synthesize the design and program the FPGA. The files can be divided into files from the CAECO project and files from the RISC-V project.

3.1.1 Caeco files

The required RTL CAECO files to add as design sources are:

- `caeco.vhd`
- `caeco_data_buffer.vhd`
- `caeco_data_storage.vhd`
- `caeco_pack.vh`
- `caeco_pack.vhd`
- `caeco_pe.vhd`
- `caeco_weight_storage.vhd`
- `conv_pkg.vhd`
- `caeco_data_storage_debug.vhd`
- `caeco_weight_storage_debug.vhd`
- `caeco_debug_paths.vhd`

The last three files might be inside a folder called *"impl"*. In `caeco_weight_storage_debug.vhd` in line 30 a file is opened in read mode, however constant `debug_mem_dir` might not be correct. For this reason, modify the constant `debug_mem_dir` in file `caeco_debug_paths.vhd` to specify the relative path from file `caeco_weight_storage_debug.vhd`, which is the file where the constant is used. Additionally, in file `caeco_pack.vhd` there is a constant called *"LOGGING"*, make sure it does not have the value *"debug"* to prevent errors related to finding files to write.

3.1.2 MEM and COE files

Important files that are required in this project to indicate how the RISC-V must behave are the MEM and COE file, these files are filled with instructions that the RISC-V will execute. However, the method employed to load the data from the files to the memory (BRAM) of the RISC-V is different for both cases.

On the one hand, the MEM file is called in the testbench for simulation purposes. A *task* is defined in the testbench to simulate writing the BRAM via the JTAG interface. On the other hand, the BRAM configuration allows a COE file to be loaded as initialization file. Therefore, the COE file is considered during synthesis, as opposed to the MEM file, which is only used for simulation. Additionally, even though both files load data to the BRAM, the MEM file can be used during simulation to overwrite the data in BRAM loaded initially from the COE file.

The process to obtain the MEM file required by the JTA task and the COE file to initialize the BRAM is explained in this section. This process involves 3 steps:

- Convert ELF file to MEM file with `data2mem`
- Format MEM file with `new-mem` tool
- Convert formatted MEM file to COE file with `new-coe` tool

3.1.2.1 Convert ELF file to MEM file with `data2mem`

The ELF file contains the program that the RISC-V executes, this file is generated by the GNU toolchain explained in section 3.2.1. However, the ELF file cannot be used neither for simulation nor synthesis. The first step to convert the ELF file into a file accepted by Vivado is using the tool `data2mem`. This tool can be downloaded from source `data2mem`. Once downloaded, the `data2mem.exe` file must be placed in the folder: "C:\Xilinx\Vivado\2023.2\bin\unwrapped\win64.o\". Finally, on the Tcl Console in Vivado enter the following command to generate the new MEM file:

```
> data2mem -bd <absolute path/FILE.elf> -d e -o m <absolute path/MEM_file.mem>
```

Listing 1: `data2mem` command syntax

Use / instead of \ to run the `data2mem` command. For my specific case, the paths used were the following:

```
> data2mem -bd C:/Users/sgarcia/Documents/Eclipse/workspace_tutorial/Tutorial/Debug/Tutorial.elf -d e -o m C:/Users/sgarcia/Documents/Vivado/POMAA_tutorial/Tutorial_X.mem
```

Listing 2: Generation of MEM file with `data2mem` command

3.1.2.2 Format MEM file with `new-mem` tool

Although a MEM file now exists, the data in it cannot be used by the JTAG task due to its current format. For this reason, the tool `new-mem` is required. The tool formats the initial MEM file to make it usable. The tools `new-mem` and `new-coe` were created using RUST. Executable files in RUST projects can be found under `target/release`. However, it might be the case that the `release` folder had been moved out from the `target` folder, as probably in this case. If the `release` folder contains the `new-mem.exe` compressed executable file instead of the EXE file, first the file must be expanded. This is achieved just by modifying its extension from `.ex_` to `.exe`. If in the terminal you are located where the EXE file is, the command syntax is the following:

```
> .\new-mem.exe <absolute path/MEM_file.mem> <absolute path/new_MEM_file.mem>
```

Listing 3: `new-mem` command syntax

Otherwise, remember to add the absolute path to the EXE file to run it as follows:

```
> <absolute path/new-mem.exe> <abs path/MEM_file.mem> <abs path/new_file.mem>
```

Listing 4: `new-mem` command syntax

The paths used for my case were the following:

```
> .\new-mem.exe C:/Users/sgarcia/Documents/Vivado/POMAA_tutorial/Tutorial_X.mem C:/Users/sgarcia/Documents/Vivado/POMAA_tutorial/Tutorial.mem
```

Listing 5: Generation of MEM file with `new-mem` command

The new MEM file generated can now be used for simulation in Vivado.

3.1.2.3 Format COE file with new-coe tool

Generating a COE file from the formatted MEM file generated by *new-mem* is achieved using the *new-coe* tool. To execute the application, first the expansion of the EX_ file must be done. This file can be found under `tools/new-coe/target/release/`. The command syntax to use it is the following:

```
> .\new-coe.exe <absolute path/MEM_file.mem> <absolute path/COE_file.coe>
```

Listing 6: *new-coe* command syntax

The paths used for my case were:

```
> .\new-coe.exe C:/Users/sgarcia/Documents/Vivado/POMAA_tutorial/Tutorial.mem C:/Users/sgarcia/I
```

Listing 7: Generation of COE file with *new-coe* command

3.1.3 RISC-V files

The RISC-V files are conformed by the RTL designs sources, IP cores provided by Xilinx, simulation files and the constraints file.

3.1.3.1 Design Sources

Add as design sources the files listed below:

- `caecointerface.v`
- `jtag_tasks.vh`
- `POMAA_constants.vh`
- `raifes_alu.v`
- `raifes_alu_ops.vh`
- `raifes_arch_options.vh`
- `raifes_core.v`
- `raifes_csr_addr_map.vh`
- `raifes_csr_file.v`
- `raifes_ctrl.v`
- `raifes_ctrl_constants.vh`
- `raifes_debug_module.v`
- `raifes_debug_rom.v`
- `raifes_dmi_constants.vh`
- `raifes_dtm.v`

- raifes_fpga_wrapper.v
- raifes_gpio.v
- raifes_hasti_bridge.v
- raifes_hasti_constants.vh
- raifes_imm_gen.v
- raifes_md_constants.vh
- raifes_mul_div.v
- raifes_PC_mux.v
- raifes_pipeline.v
- raifes_platform_constants.vh
- raifes_regfile.v
- raifes_src_a_mux.v
- raifes_src_b_mux.v
- raifes_sync_to_hasti_bridge.v
- raifes_top.v
- raifes_uart.v
- rv32_opcodes.vh
- UART_module.v

3.1.3.2 IP Cores

IP cores can be created and configured in Vivado using the GUI or using the TCL Shell. Add the XCI files for the IP cores as design sources, if there are no such files then create them. To create the IP Clocking Wizard select *Project Manager* → *IP Catalog* → *FPGA Features and Design* → *Clocking* → *Clocking Wizard* and configure it as shown in Figure 9 and 10.

Board

Clocking Options

Output Clocks

Port Renaming

MMCM Settings

Summary

Clock Monitor

☐ Enable Clock Monitoring

Primitive

☒ MMCM
 ☐ PLL

Clocking Features

☒ Frequency Synthesis
 ☐ Minimize Power
 ☐ Phase Alignment
 ☐ Spread Spectrum
 ☐ Dynamic Reconfig
 ☐ Dynamic Phase Shift
 ☐ Safe Clock Startup

Jitter Optimization

☒ Balanced
 ☐ Minimize Output Jitter
 ☐ Maximize Input Jitter filtering

Dynamic Reconfig Interface Options

☒ AXI4Lite
 ☐ DRP
 ☐ Phase Duty Cycle Config
 ☐ Write DRP registers

Input Clock Information

	Input Clock	Port Name	Input Frequency(MHz)		Jitter Options	Input Jitter	Source
	Primary	clk_in1	100.000	10.000 - 800.000	UI	0.010	Single ended clock capable pir
<input type="checkbox"/>	Secondary	clk_in2	100.000	65.753 - 131.507		0.010	Single ended clock capable pi

Figure 9: Clocking options

Board
Clocking Options
Output Clocks
Port Renaming
MMCM Settings
Summary

The phase is calculated relative to clk_out1.

Output Clock	Port Name	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Drives	Use Fine PS	Max Freq. of buffer
		Requested	Actual	Requested	Actual	Requested	Actual			
<input checked="" type="checkbox"/> clk_out1	clk_out1	25.000	25.00000	0.000	0.000	50.000	50.0	BUFG	<input type="checkbox"/>	464.037
<input type="checkbox"/> clk_out2	clk_out2	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	<input type="checkbox"/>	464.037
<input type="checkbox"/> clk_out3	clk_out3	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	<input type="checkbox"/>	464.037
<input type="checkbox"/> clk_out4	clk_out4	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	<input type="checkbox"/>	464.037
<input type="checkbox"/> clk_out5	clk_out5	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	<input type="checkbox"/>	464.037
<input type="checkbox"/> clk_out6	clk_out6	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	<input type="checkbox"/>	464.037
<input type="checkbox"/> clk_out7	clk_out7	100.000	N/A	0.000	N/A	50.000	N/A	BUFG	<input type="checkbox"/>	464.037

☐ USE CLOCK SEQUENCING

Output Clock	Sequence Number
clk_out1	1
clk_out2	1
clk_out3	1
clk_out4	1
clk_out5	1
clk_out6	1
clk_out7	1

Enable Optional Inputs / Outputs for MMCM/PLL
Reset Type

☒ reset
☐ power_down
☐ input_clk_stopped
☒ Active High
☐ Active Low

☐ locked
☐ clkfbstopped

Figure 10: Output clocks configuration

Additionally, an IP Block Memory Generator is also required. Another alternative to create it is by searching *Block Memory Generator* in *Project Manager* → *IP Catalog*. Configure it following Figures 11, 12, 13 and 14. The BRAM must be initialized with a COE file, which can be generated from an ELF file.

IP Symbol Power Estimation

☐ Show disabled ports

Component Name: blk_mem_gen_0

Basic Port A Options Port B Options Other Options Summary

Interface Type: Native ☒ Generate address interface with 32 bits

Memory Type: True Dual Port RAM ☒ Common Clock

ECC Options

ECC Type: No ECC

☐ Error Injection Pins Single Bit Error Injection

Write Enable

☒ Byte Write Enable

Byte Size (bits): 8

Algorithm Options

Defines the algorithm used to concatenate the block RAM primitives. Refer datasheet for more information.

Algorithm: Minimum Area

Primitive: 8x2

Figure 11: Basic BRAM configuration

Basic Port A Options Port B Options Other Options Summary

Memory Size

Write Width: 32 Range: 32 to 1024 (bits)

Read Width: 32

Write Depth: 65536 Range: 2 to 1048576

Read Depth: 65536

Operating Mode: Write First Enable Port Type: Always Enabled

Port A Optional Output Registers

☐ Primitives Output Register ☐ Core Output Register

☐ SoftECC Input Register ☐ REGCEA Pin

Port A Output Reset Options

☐ RSTA Pin (set/reset pin) Output Reset Value (Hex): 0

☐ Reset Memory Latch Reset Priority: CE (Latch or Register Enable)

READ Address Change A

☐ Read Address Change A

Figure 12: Port A configuration

Basic | **Port A Options** | **Port B Options** | Other Options | Summary

Memory Size

Write Width: 32
Read Width: 32
Write Depth: 65536
Read Depth: 65536

Operating Mode: Write First
Enable Port Type: Always Enabled

Port B Optional Output Registers

☐ Primitives Output Register ☐ Core Output Register
☐ SoftECC Output Register ☐ REGCEB Pin

Port B Output Reset Options

☐ RSTB Pin (set/reset pin) Output Reset Value (Hex): 0
☐ Reset Memory Latch Reset Priority: CE (Latch or Register Enable)

READ Address Change B

☐ Read Address Change B

Figure 13: Port B configuration

Basic | **Port A Options** | **Port B Options** | Other Options | Summary

Pipeline Stages within Mux: 0 Mux Size: 16x1

Memory Initialization

☒ Load Init File
Coe File: J:\J.J.J.\Documents\Vivado\POMAA_tutorial\POOMAv1_0.coe [Browse] [Edit]
☒ Fill Remaining Memory Locations
Remaining Memory Locations (Hex): 0

Structural/UniSim Simulation Model Options

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.
Collision Warnings: All

Behavioral Simulation Model Options

☐ Disable Collision Warnings ☐ Disable Out of Range Warnings

Figure 14: Memory initialization

3.1.3.3 Simulation Sources

The simulation sources that can be added for simulation are the `masterthesis_tb.v` file, which contains the testbench, and the ECG file `00019fb0-6b6a-4ccf-b818-b52221ec524c.ecg` and a MEM file e.g. `P00MAv1_0Sim.mem` both used by the testbench.

3.1.3.4 Constraints

Finally, the XDC constraint file `Nexys-A7-100T-Master.xdc`, which defines the pins to be used must be added to the project. The final structure of the project is shown in Figure 15.



Figure 15: Project structure

3.1.4 Program Device

The final steps are running synthesis, then implementation and generating the bitstream. To program the device from the *Tool Navigator* select *Program and Debug* → *Open Hardware Management* →

Autoconnect and in the top green bar select *Programm Device*. Once it has been programmed press the **CPU RESET** button on the board and now it is ready for debugging.

3.2 Eclipse IDE for Embedded C/C++ Developers 2023-09

As source-code editor and to build and debug code the *Eclipse IDE for Embedded C/C++ Developers* is used. This IDE packs together the *Eclipse IDE for C/C++ Developers* standard distribution with the *Eclipse Embedded CDT* managed cross build plug-ins for Arm and RISC-V, as well as debug plug-ins, such as OpenOCD, SEGGER J-Link, pyocd, and QEMU [7].

The *Eclipse Embedded CDT* (Eclipse Embedded C/C++ Development Tools), formerly GNU MCU/ARM Eclipse, is an open source project that includes a family of Eclipse plug-ins and tools for multi-platform embedded cross (Arm and RISC-V) development, based on GNU toolchains. These plug-ins provide extensions for GNU embedded toolchains (xPack GNU RISC-V Embedded GCC, xPack GNU Arm Embedded GCC, etc.) to create, build, debug and in general to manage Arm & RISC-V projects (executables and static/shared libraries, in both 32 and 64-bit versions) with the Eclipse framework [8].

Eclipse IDE for Embedded C/C++ Developers can be downloaded from source [9] to be installed as shown in Figure 16.

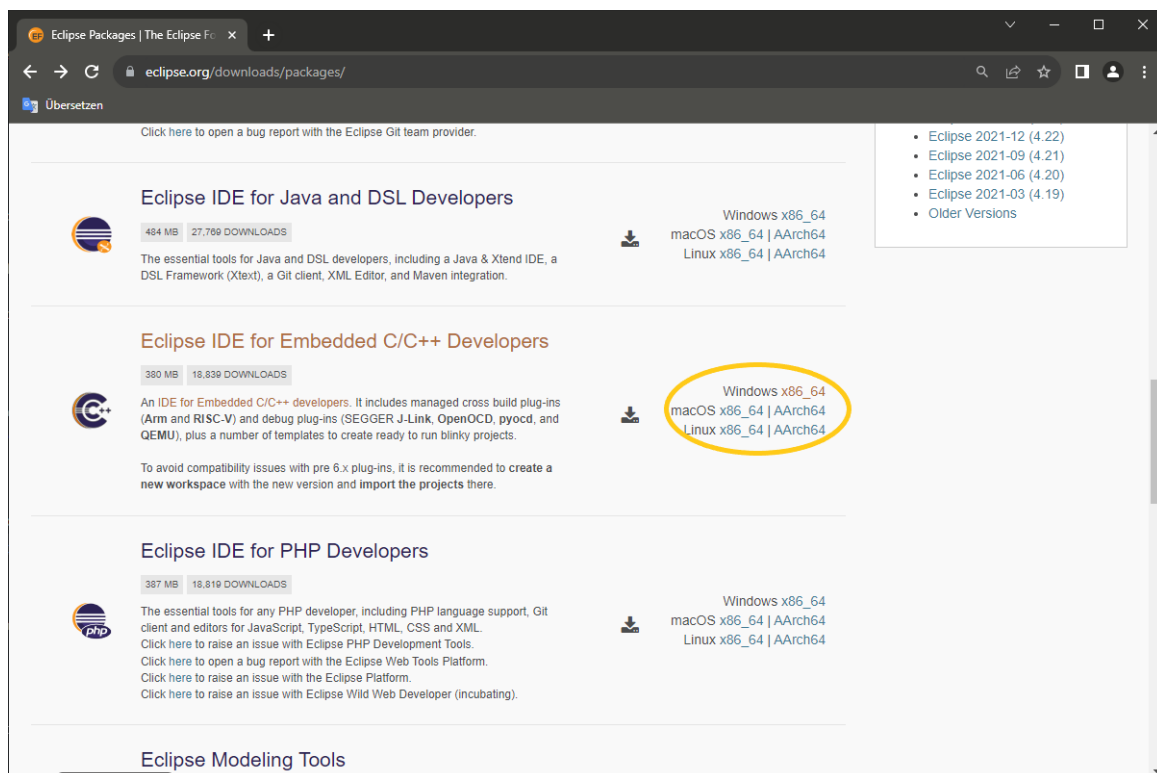


Figure 16: Download Eclipse IDE for Embedded C/C++ Developers

3.2.1 xPack GNU RISC-V Embedded GCC

Apart from the main Eclipse plug-ins, the package *xPack GNU RISC-V Embedded GCC* must be installed to run the RISC-V C and C++ cross-compiler and generate the code that runs on the RISC-V. The package contains a

binary distribution of *GNU RISC-V Embedded GCC*, which is the new distribution of the bare metal version of *RISC-V GCC*. The distribution is conformed by a collection of programming tools produced by the GNU Project that forms the RISC-V C and C++ cross-compiler toolchain [10].

3.2.1.1 Prerequisites

To install the package a recent *xpm* is necessary, and to install it a recent *Node.js* and a recent *npm* are required.

- xPack GNU RISC-V Embedded GCC requires:
 - a recent *xpm* that requires:
 - * a recent *Node.js* (≥ 16.14)
 - * a recent *npm*

The most recent *Node.js* version can be downloaded from [11]. After installation, the version of *Node.js* can be displayed with the following command:

```
> node --version  
v20.9.0
```

Listing 8: *Node.js* version command

The *npm* that comes with *Node.js* is recommended to be updated as follows:

```
> npm install --global npm@latest  
  
changed 14 packages in 5s  
  
28 packages are looking for funding  
  run 'npm fund' for details  
  
> npm --version  
10.2.4
```

Listing 9: *npm* installation and version check

The *xpm* can now be installed with the next command:

```
> npm install --global xpm@latest  
  
removed 35 packages, and changed 42 packages in 11s  
  
56 packages are looking for funding  
  run 'npm fund' for details  
  
> xpm --version  
0.18.0
```

Listing 10: *xpm* installation and version check

3.2.1.2 Installation

After installing the prerequisites, the *xPack GNU RISC-V Embedded GCC* can be installed [12] and once it has been installed the version can be displayed using the following commands:

```
> xpm install --global @xpack-dev-tools/riscv-none-embed-gcc@latest

@xpack-dev-tools/riscv-none-embed-gcc@10.2.0-1.2.1...
warning: package @xpack-dev-tools/riscv-none-embed-gcc@10.2.0-1.2.1 already installed
, use --force to overwrite

> AppData\Roaming\xPacks\@xpack-dev-tools\riscv-none-embed-gcc\10.2.0-1.2.1\content\
bin\riscv-none-embed-gcc.exe --version

riscv-none-embed-gcc.exe (xPack GNU RISC-V Embedded GCC x86_64) 10.2.0
Copyright (C) 2020 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Listing 11: xPack GNU RISC-V Embedded GCC installation and version check

The *xpack-dev-tools/riscv-none-embed-gcc-xpack* toolchain is now end-of-life and there will be no more releases, thus, its replacement is the new toolchain *xpack-dev-tools/riscv-none-elf-gcc-xpack* [13]. However, for our project the *xpack-dev-tools/riscv-none-embed-gcc-xpack* toolchain is required, since the new version is not compatible with extensions used in the *crt0.S* file. For future projects the new toolchain can be installed with the following command:

```
> xpm install --global @xpack-dev-tools/riscv-none-elf-gcc@latest --verbose

xPack manager - install package(s)

Processing @xpack-dev-tools/riscv-none-elf-gcc@13.2.0-2.1...
warning: package @xpack-dev-tools/riscv-none-elf-gcc@13.2.0-2.1 already installed, use
--force to overwrite

'xpm install' completed in 771 ms.

> AppData\Roaming\xPacks\@xpack-dev-tools\riscv-none-elf-gcc\13.2.0-2.1\content\bin\
riscv-none-elf-gcc.exe --version

riscv-none-elf-gcc.exe (xPack GNU RISC-V Embedded GCC x86_64) 13.2.0
Copyright (C) 2022 Free Software Foundation, Inc.
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Listing 12: xPack GNU RISC-V Embedded GCC installation with new toolchain and version check

3.2.2 xPack Windows Build Tools

To perform builds on Windows the *xPack Windows Build Tools* project includes the additional tools required. This package contains a Windows binary distribution of GNU Make required for the Eclipse Embedded CDT managed build projects [14].

```
> xpm install --global @xpack-dev-tools/windows-build-tools@latest --verbose

xPack manager - install package(s)

Processing @xpack-dev-tools/windows-build-tools@4.4.1-2.1...
warning: package @xpack-dev-tools/windows-build-tools@4.4.1-2.1 already installed, use
--force to overwrite
```

```
'xpm install' completed in 850 ms.

> AppData\Roaming\xPacks\@xpack-dev-tools\windows-build-tools\4.4.1-2.1\content\bin\
make --version

GNU Make 4.4.1
Built for x86_64-w64-mingw32
Copyright (C) 1988-2023 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <https://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
```

Listing 13: xPack Windows Build Tool installation and version check

3.2.3 xPack OpenOCD

For a better/more convenient integration with the Eclipse *OpenOCD Debugging* plug-in when debugging on the device the *xPack OpenOCD* package is installed. This package contains a binary distribution of *OpenOCD* [15].

```
> xpm install --global @xpack-dev-tools/openocd@latest --verbose

xPack manager - install package(s)

Processing @xpack-dev-tools/openocd@0.12.0-2.1...
warning: package @xpack-dev-tools/openocd@0.12.0-2.1 already installed, use --force t
o overwrite

'xpm install' completed in 829 ms.

> AppData\Roaming\xPacks\@xpack-dev-tools\openocd\0.12.0-2.1\content\bin\openocd.exe
--version

xPack Open On-Chip Debugger 0.12.0+dev-01312-g18281b0c4-dirty (2023-09-04-22:32)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
```

Listing 14: xPack OpenOCD installation and version check

3.3 HTerm

HTerm is used for debugging as the terminal program for the serial communication between the FPGA and the PC. In this tutorial, only the FPGA sends data at a baud rate of 115200 bits per second. The HTerm can be downloaded from source [16].

4 Debugging in Eclipse IDE for Embedded C/C++ Developers

By building a project in Eclipse IDE with the correct configuration, an ELF file is generated. This ELF file is then used as the input for the debugger tool to debug the code on the FPGA. The steps to debug are:

- Create a project

- Configure project properties
- Configure the debugger
- Debug

4.1 Create Project

When Eclipse IDE is opened the user must select the directory that will be used as workspace.

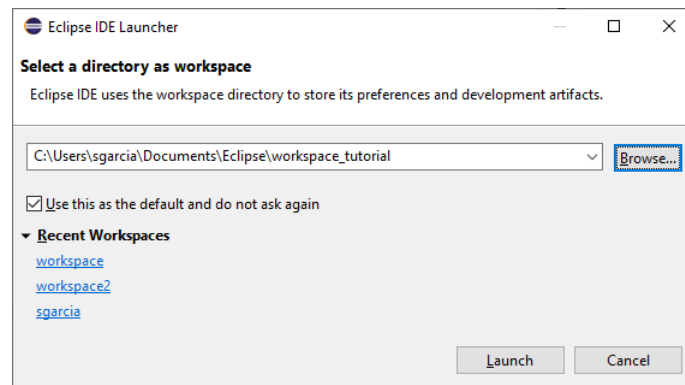


Figure 17: Selection of directory as workspace

Create a new Eclipse project for Embedded C/C++ source code.

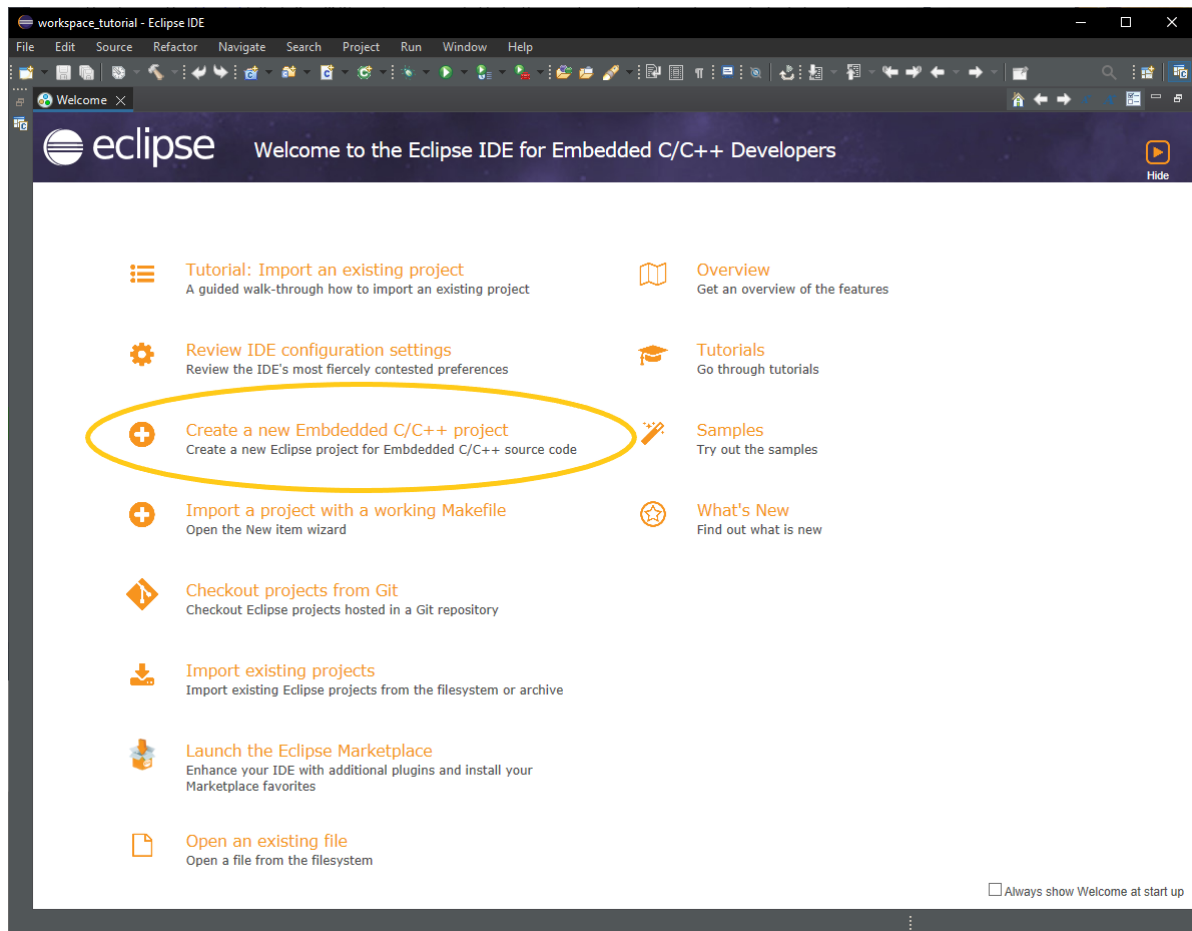


Figure 18: Create new Embedded C/C++ project

Select *C Managed Build* to use the CDT's managed build system to build C projects.

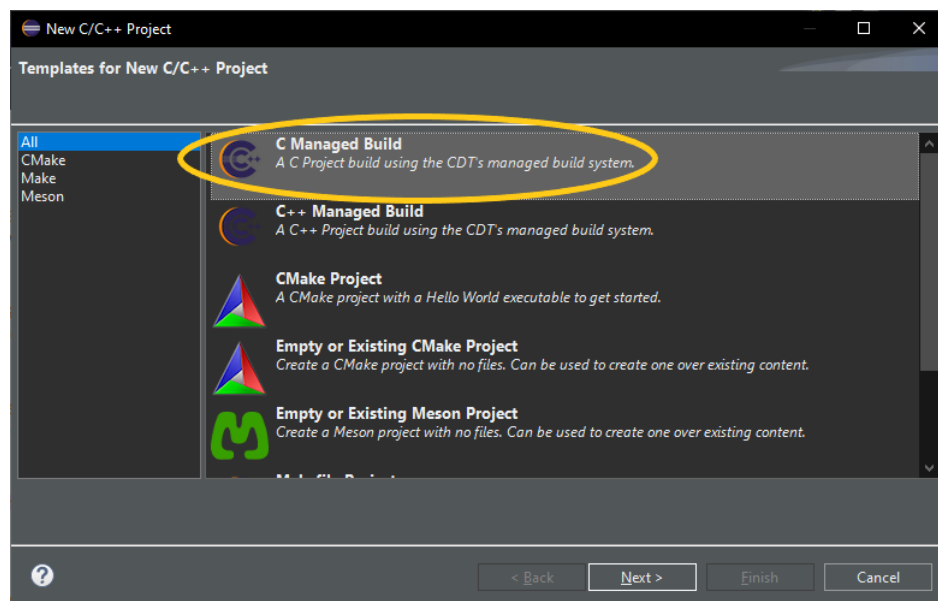


Figure 19: C Managed Build selection

Name the project, select a project type and a toolchain. For the *Hello World RISC-V C Project* type only the toolchain *RISC-V Cross GCC* is available.

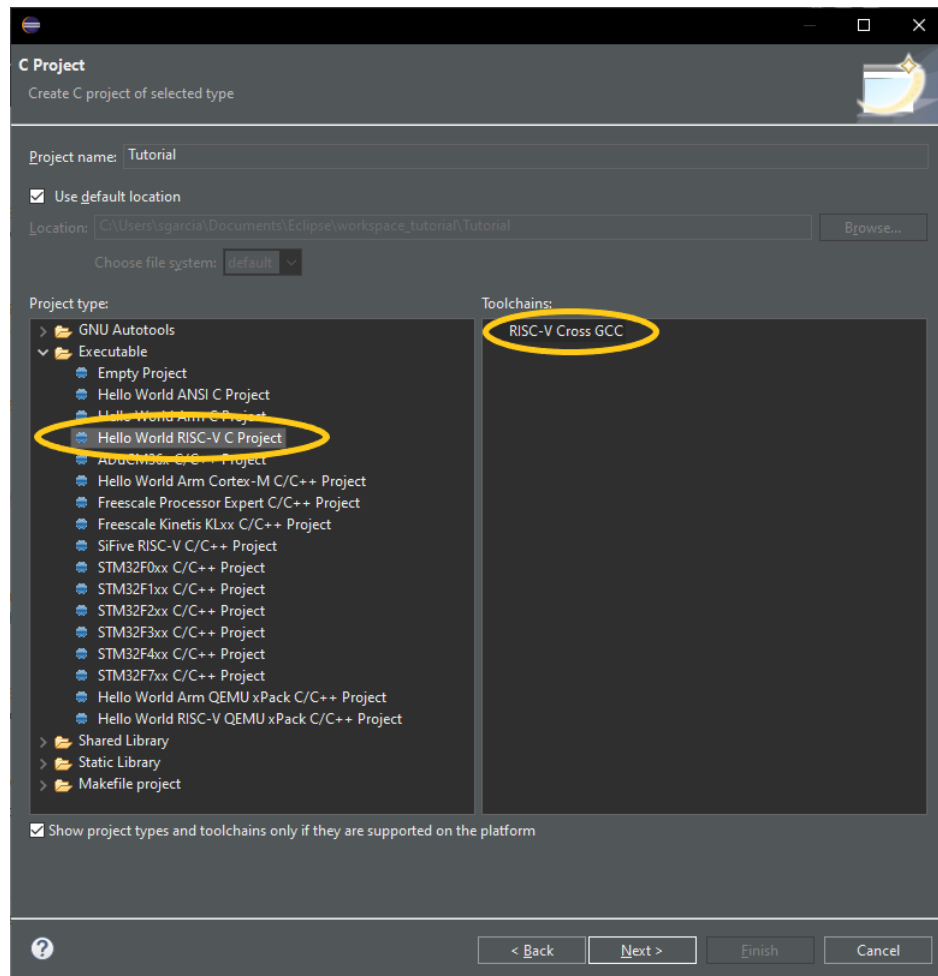


Figure 20: C Managed Build selection

Add the author name and modify the copyright if necessary.

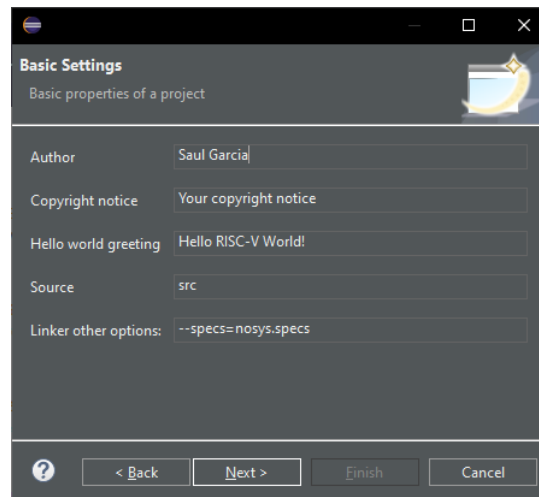


Figure 21: Adding author name and copyright

Enable the *Debug* and *Release* configurations.

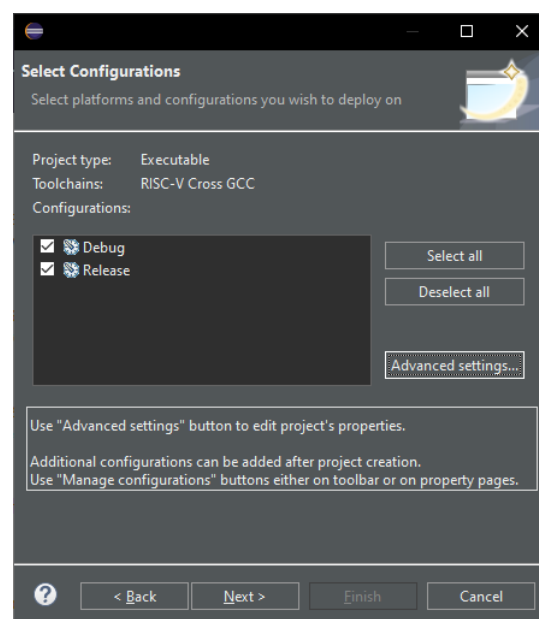


Figure 22: Select Configurations

Select the GNU RISC-V Cross toolchain and toolchain path. The required toolchain for our project is the *riscv-none-embed-gcc* toolchain, which is considered "*DEPRECATED*" due to the new available toolchain as explained in section 3.2.1.2.

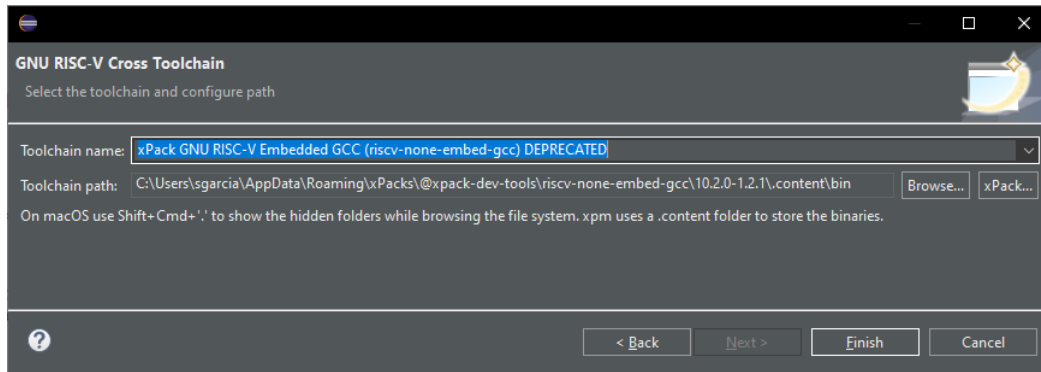


Figure 23: GNU

Eclipse adds automatically the header files from the selected toolchain. These are shown in the *Includes* section of the project. The source files that must be added to the project are:

- `airi5c_syscalls.c`: System calls
- `airi5c_syscalls.h`: System calls
- `crt0.S`: Defines the program sequence before the `main` function is called.
- `link.ld`: Link script used for the linker configuration
- `main.c`: Source code

To add source files to the project the user only needs to copy and paste the files to the `src` folder, which is inside the projects folder. The structure of the project after adding the source files is shown in Figure 24.

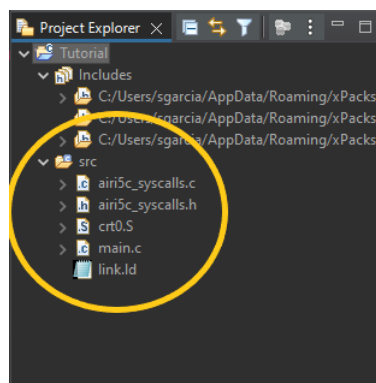


Figure 24: Source files in project

However, if the user wants to use files stored in an external folder, e.g. repository, these files can be linked to the project without duplicating them by following the steps presented in Figure 25 and Figure 26. This is helpful when we are using a version control system like git and we do not want many copies or different versions of our project on our local computer. Right click on the `src` folder, then *New* → *File*.

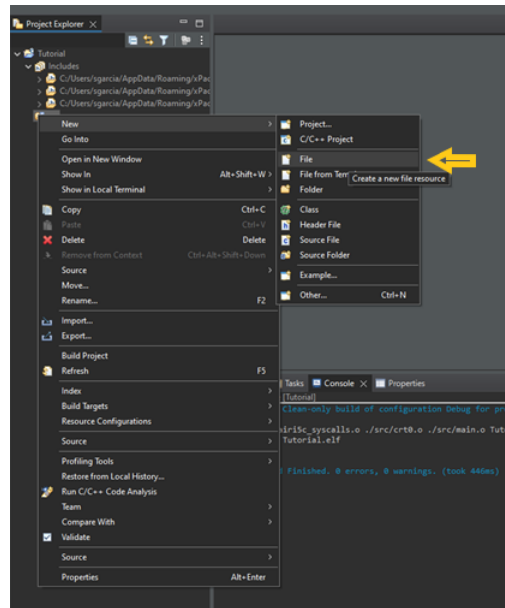


Figure 25: Linking new file to project

Click on "Advanced >>", select the "Link to file in the file system" option and click on "Browse" to select the file to link.

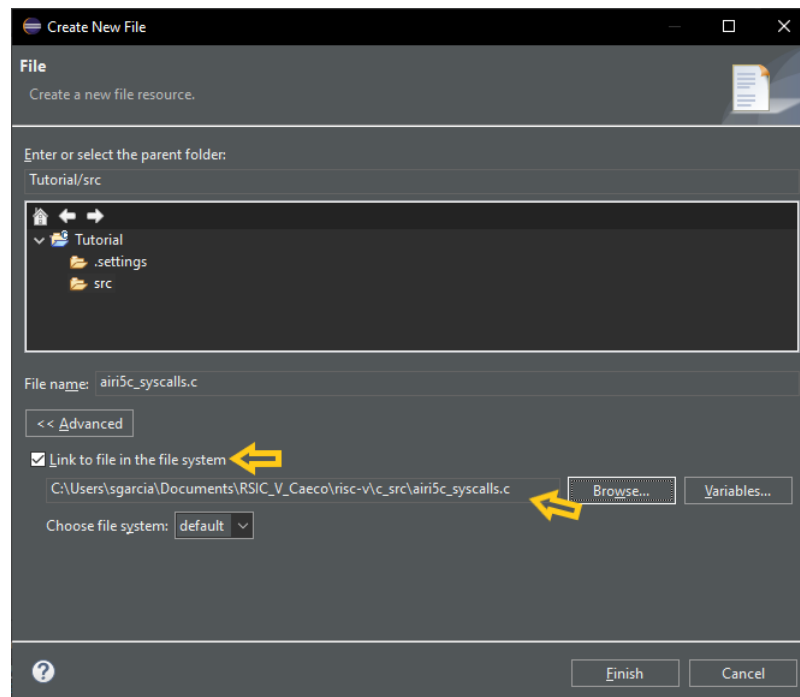


Figure 26: Selecting file

After linking all the required files the structure of the project looks as in Figure 27. The linked files are displayed with a small arrow compared to those files copied to the project.

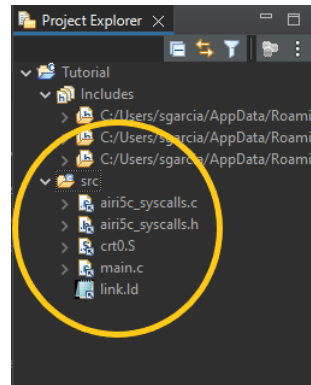


Figure 27: Project structure

4.2 Project Properties Configuration

Before building the project to generate the ELF file for the debugger, it is important to follow the next steps:

- Configure C/C++ Build settings
- Select MCU paths

To configure these parameters, from the menu bar select *Project* → *Properties*.

4.2.1 C/C++ Build Settings Configuration

4.2.1.1 Target Processor

From the options listed in the left select *C/C++ Build* → *Settings* and in the *Tool Setting* bar select *Target Processor*. The configuration that needs to be set is:

- Architecture: RV32I (-march=rv32i*) - It controls the set of instructions that GCC uses when generating assembly code. The option `rv32i*` specifies a load-store ISA with 32, 32-bit general-purpose integer registers.
- Enable Multiply extension (RVM) - checks if the target supports the M (hardware multiply/divide) extension.
- Integer ABI: ILP32 (-mabi=ilp32*) - It defines the calling convention and layout of objects in memory. The option `ilp32*` establishes that “int”, “long”, and pointers are all 32-bit long. “long long” is a 64-bit type, “char” is 8-bit, and “short” is 16-bit.

A graphical representation is presented in Figure 28.

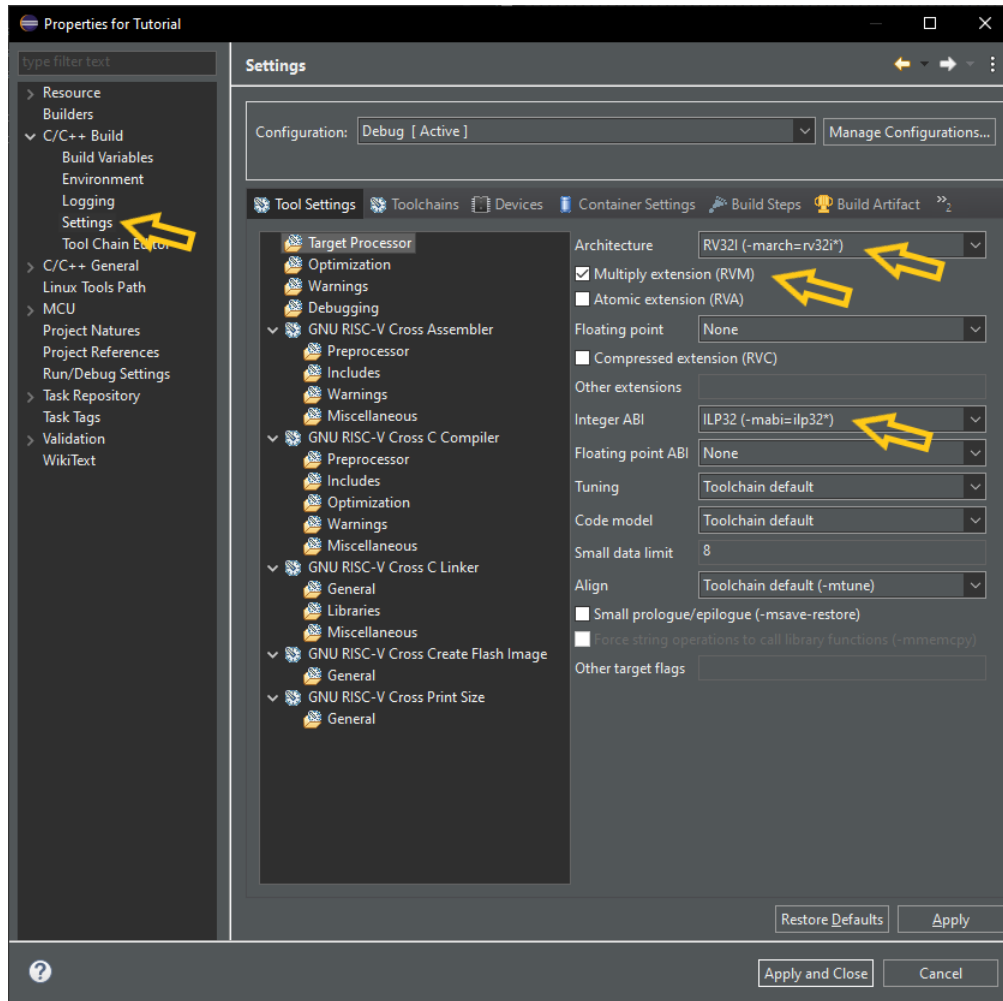


Figure 28: Target Processor configuration

4.2.1.2 Debugging

Select the *Debugging* option and make sure that the "Debug level" is set to "Maximum (-g3)" as shown in Figure 29. This option produces all possible information that the debugger can produce, such as descriptions of functions, variables, line number tables, and all the macro definitions present in the program.

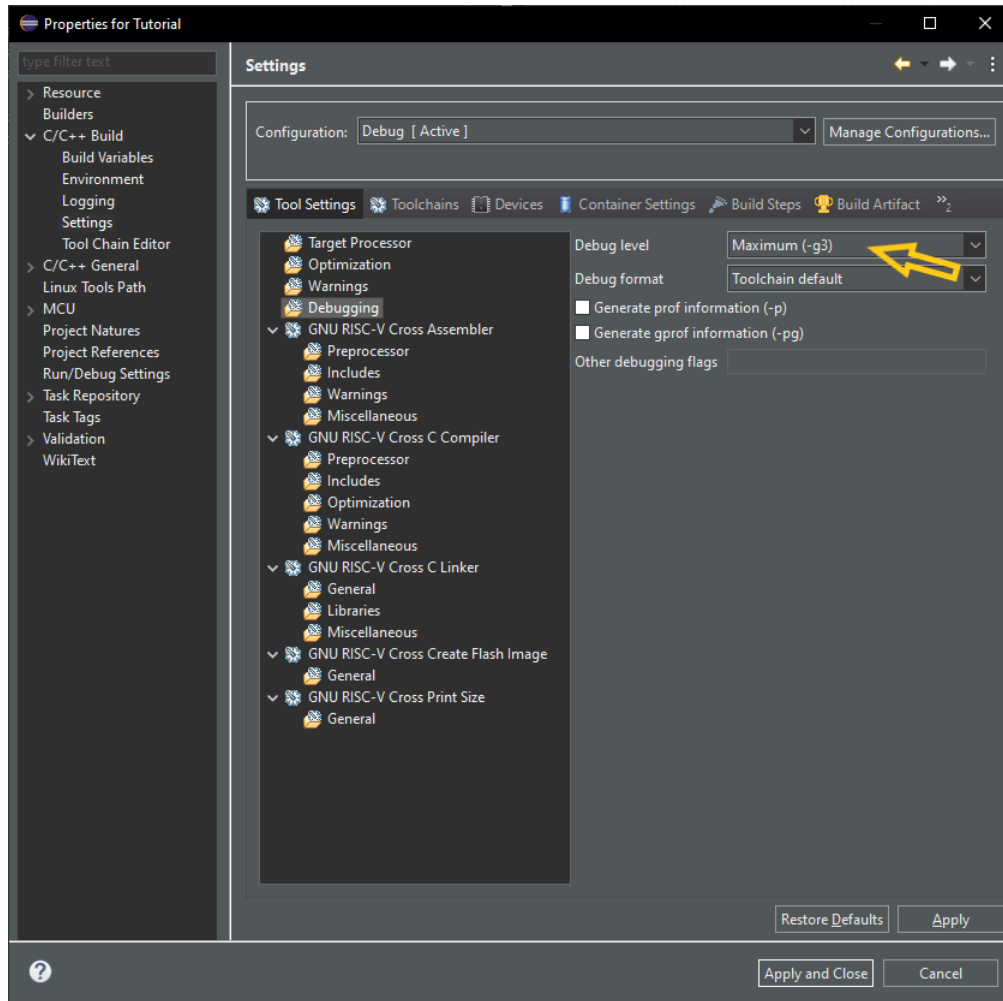


Figure 29: Debugging configuration

4.2.1.3 GNU RISC-V Cross C Linker

Select *GNU RISC-V Cross C Linker* → *General*, add the `link.ld` script file and enable the option "Do not use standard start files (-nostartfiles)" as presented in Figure 30. The last configuration avoids using standard system startup files when linking.

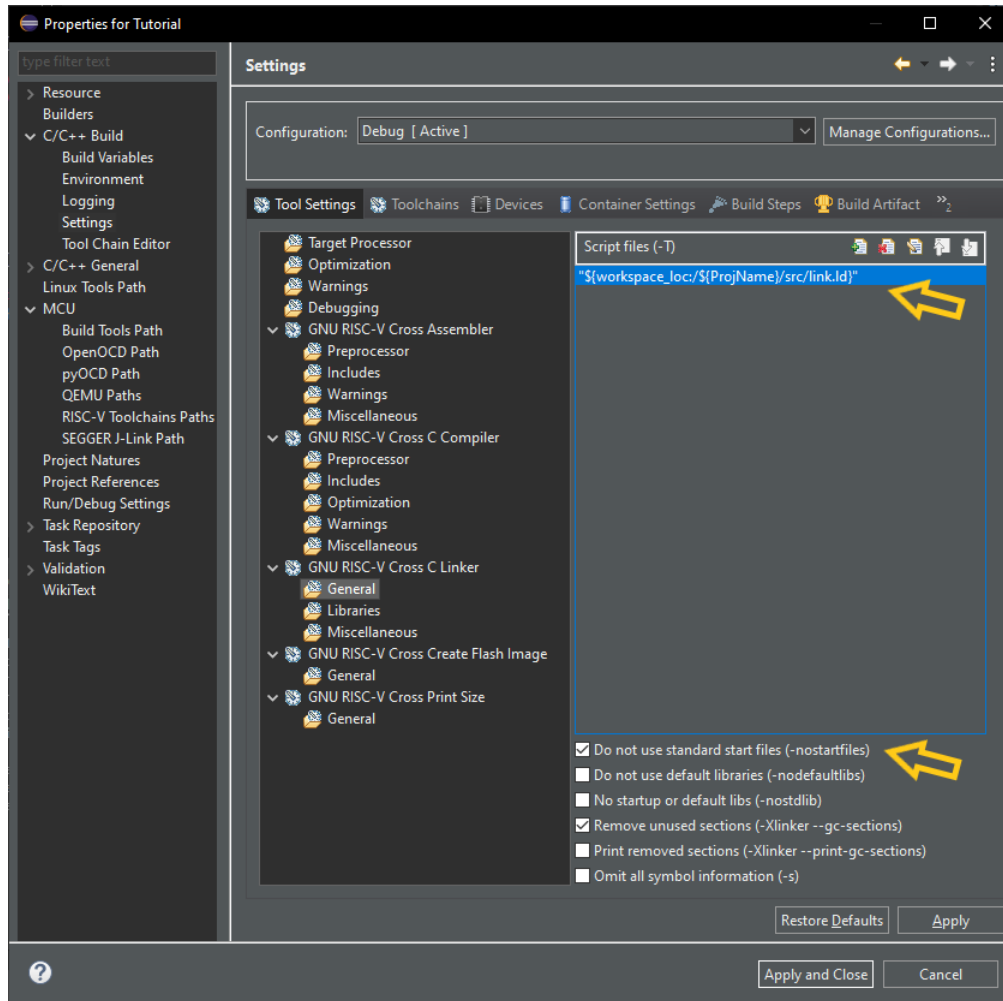


Figure 30: GNU RISC-V Cross C Linker General configuration

Select *GNU RISC – V Cross C Linker* → *Miscellaneous*, enable the option "Cross reference (-Xlinker -cref)" and Use *newlib-nano* (-specs=nano.specs) as presented in Figure 31. --cref prints to a generated linker map file the cross reference table. However, since this option is a system-specific linker option that GCC does not recognize the -Xlinker command is also added before --cref. The --specs option specifies a spec file containing directives that define how programs, invoked by GCC, do the work of compiling, assembling and linking. In this case, the nano.spec argument modifies the link options since we are in the GNU RISC-V Cross C Linker settings. nano.spec uses the newlib-nano open source C library, which targets embedded microcontrollers. It focuses on optimization and removal of non-MCU features for code and data size reduction.

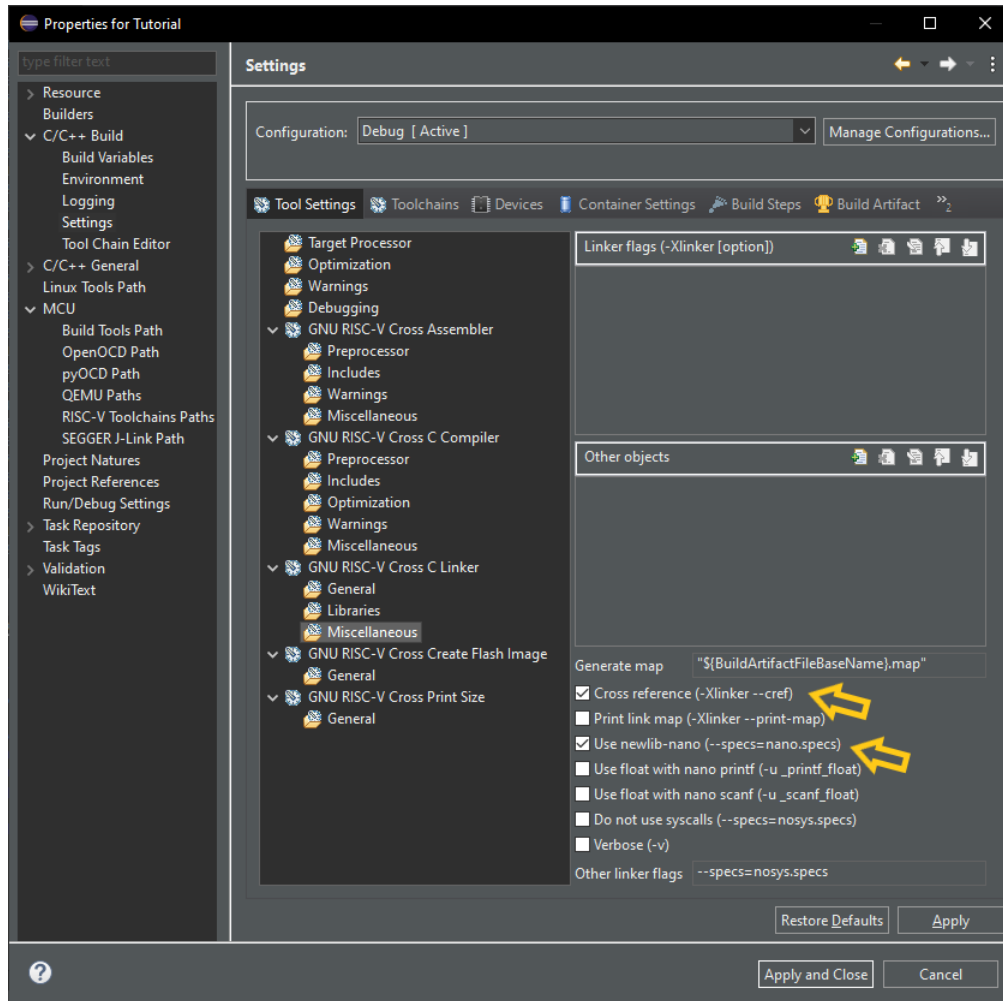


Figure 31: GNU RISC-V Cross C Linker Miscellaneous configuration

4.2.2 MCU Paths Selection

From the options listed in the left select *MCU*. The paths that need to be specified are:

- Build Tools Path
- OpenOCD Path
- RISC-V Toolchain Path

4.2.2.1 Build Tools Path

Select *Build Tools Path* and specify the build tools folder as shown in Figure 32.

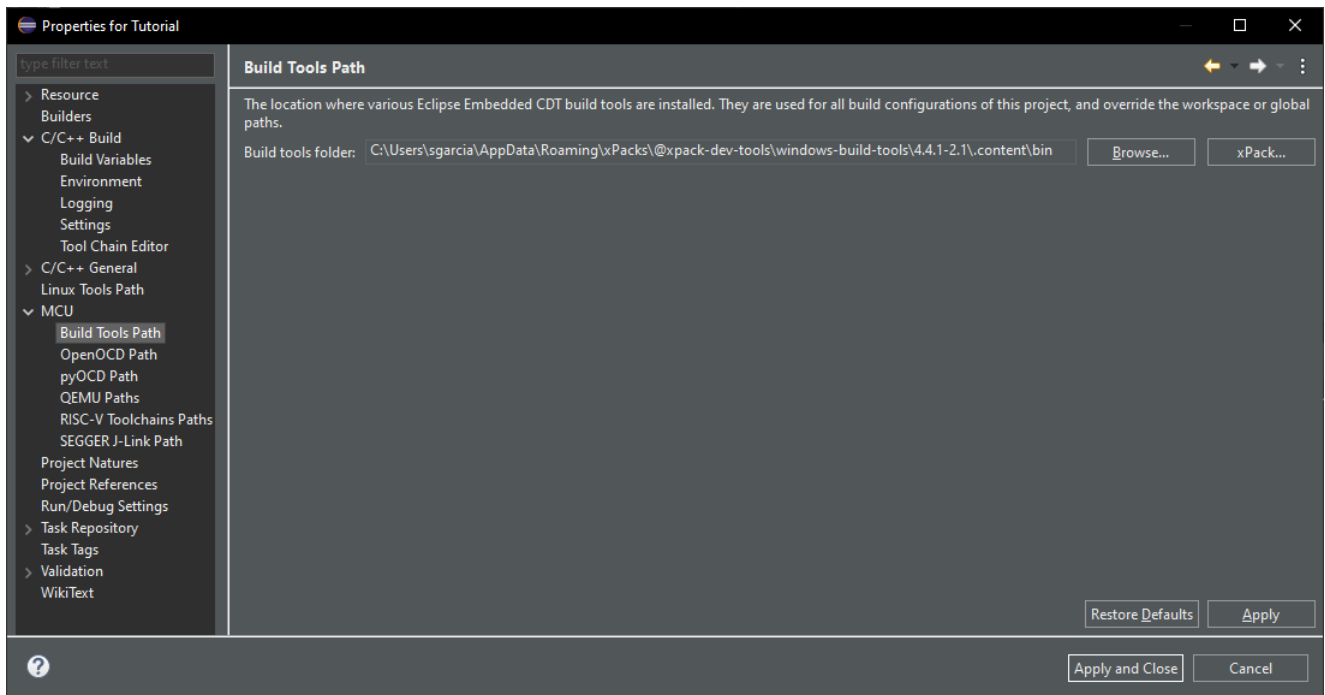


Figure 32: Build Tools Path selection

4.2.2.2 OpenOCD Path

Select *OpenOCD Path* and specify the OpenOCD folder as shown in Figure 33.



Figure 33: OpenOCD Path selection

4.2.2.3 RISC-V Toolchain Path

Select *RISC – V Toolchains Path* and specify the toolchain folder as shown in Figure 34.

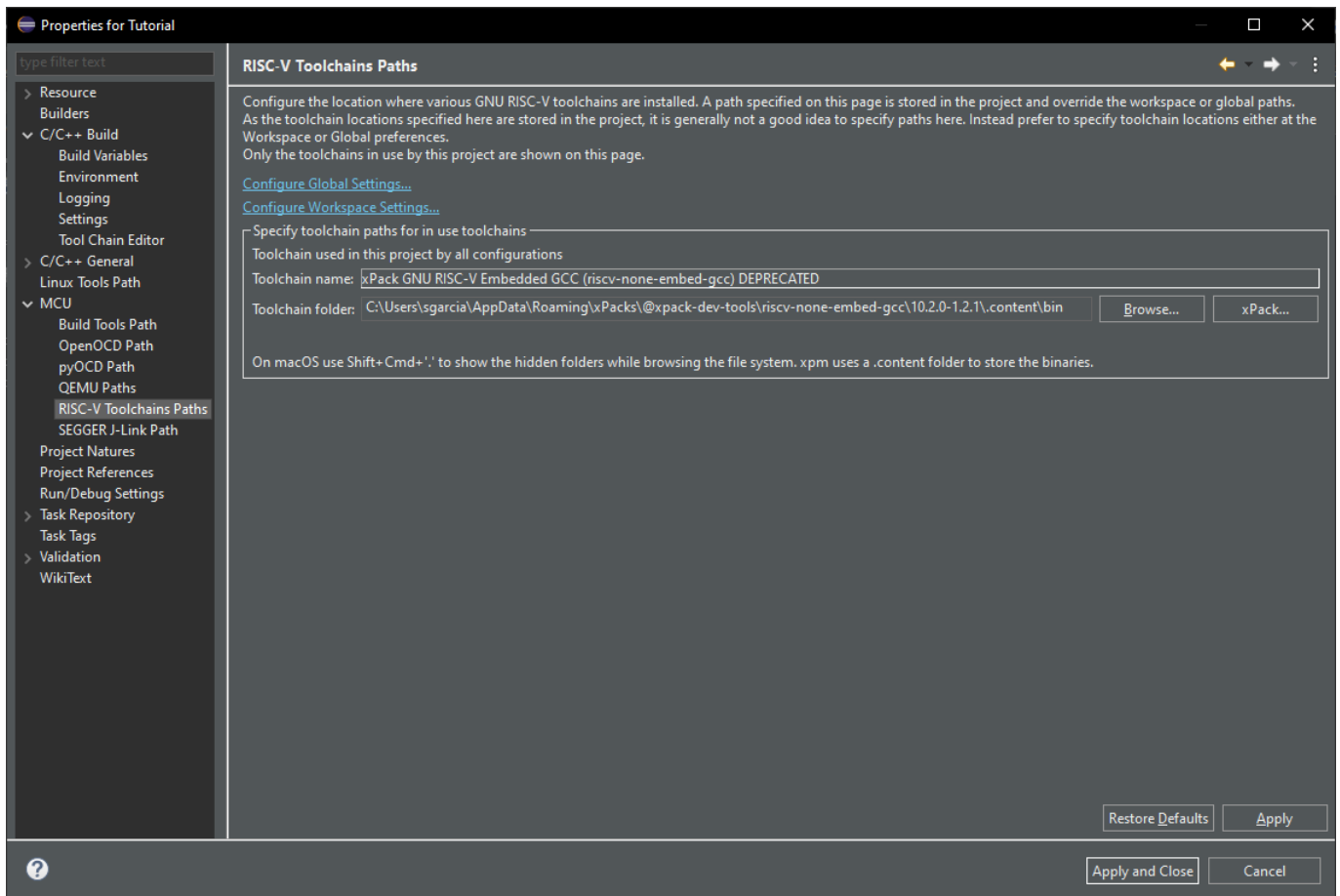


Figure 34: RISC-V Toolchains Path selection

Finally, select *Apply and Close* and build the project with *Project* → *Build Project* or with the build icon as shown in Figure 35. The *Console* view shows the building process generating the ELF file shown now in the *Project Explorer* view in the *Debug* folder.

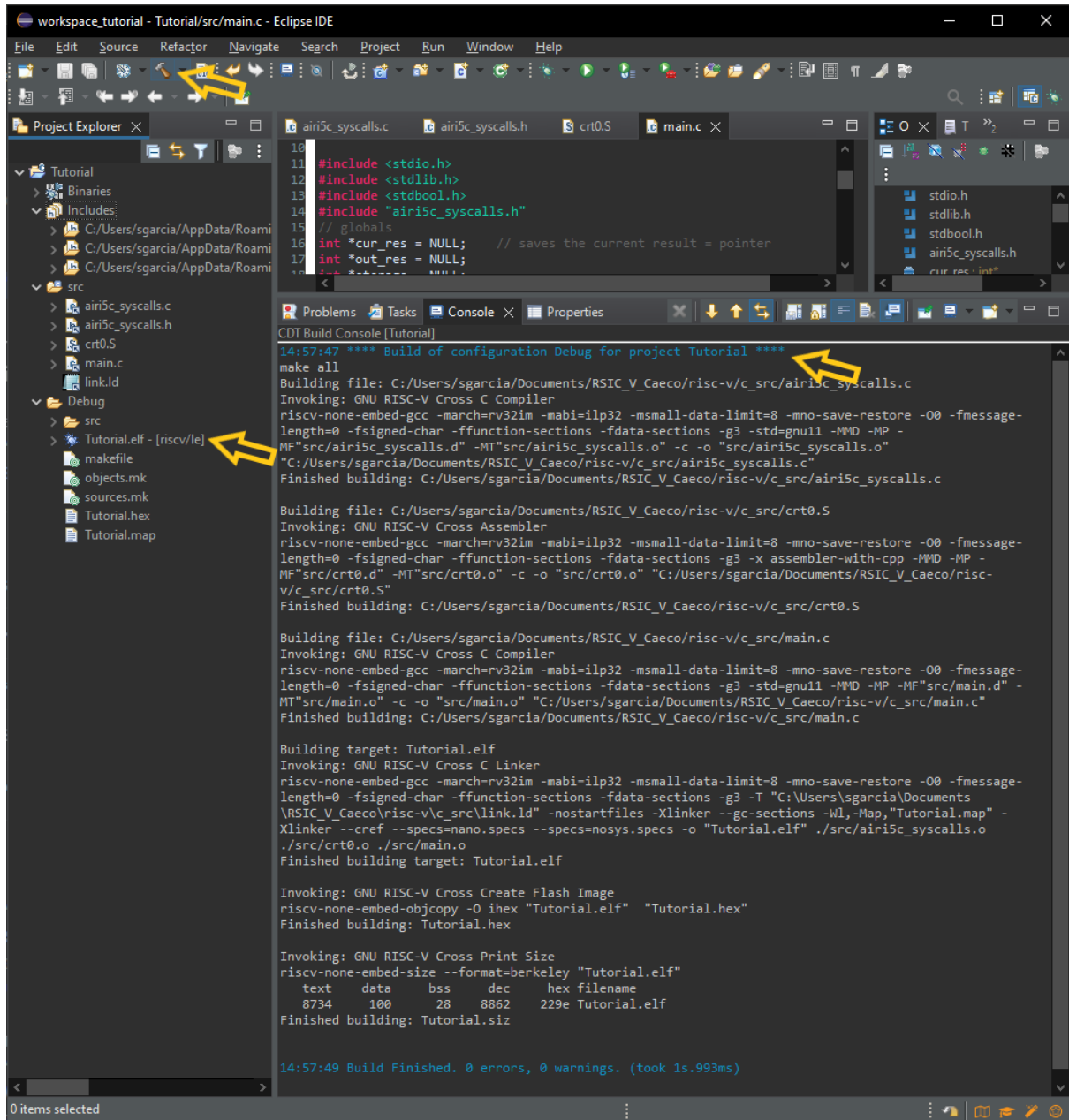


Figure 35: Building the project

4.3 Debug Configuration

To configure the debug settings the next steps must be followed:

- Add target configuration file to OpenOCD
- Configure the debugger options
- Configure main settings

4.3.1 Adding target to OpenOCD

To establish a connection between the debugger and the processor the configuration file `airi5c.cfg` must be added to the target directory in OpenOCD as shown in Figure 36. This file is loaded when the debugger starts. The name of the chip `airi5c` is specified in the first five lines in `airi5c.cfg`, then in line eight the TAP for the JTAG interface to be used as a DTM and finally the debug target is defined in line nine. Further details regarding *Config File Guidelines* can be found in [17].

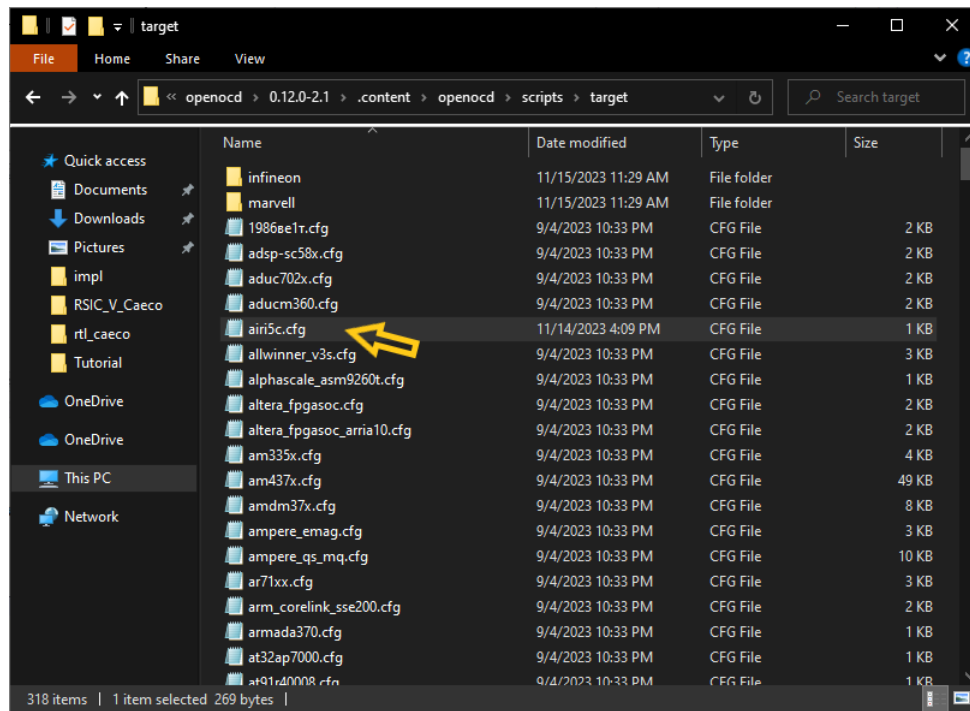


Figure 36: Add target configuration file `airi5c.cfg` to OpenOCD

4.3.2 Debugger Configuration

To configure the debugger double click *GDB OpenOCD Debugging* to create a new launch configuration with the name `< project name > Debug`. Add the lines to call the `olimex-arm-usb-tiny-h.cfg` and `airi5c.cfg` files in the *Config options* box. The first file contains the configuration for OpenOCD to debug using the adapter. The second file is the one added in the previous section. In the *GDB Client Setup* section the GDB, which is the GNU debugger, can be configured as shown in Figure 37. The first line in the *Commands* box refuses accesses to memory that is not explicitly described to prevent undesired effects. The second line specifies the architecture.

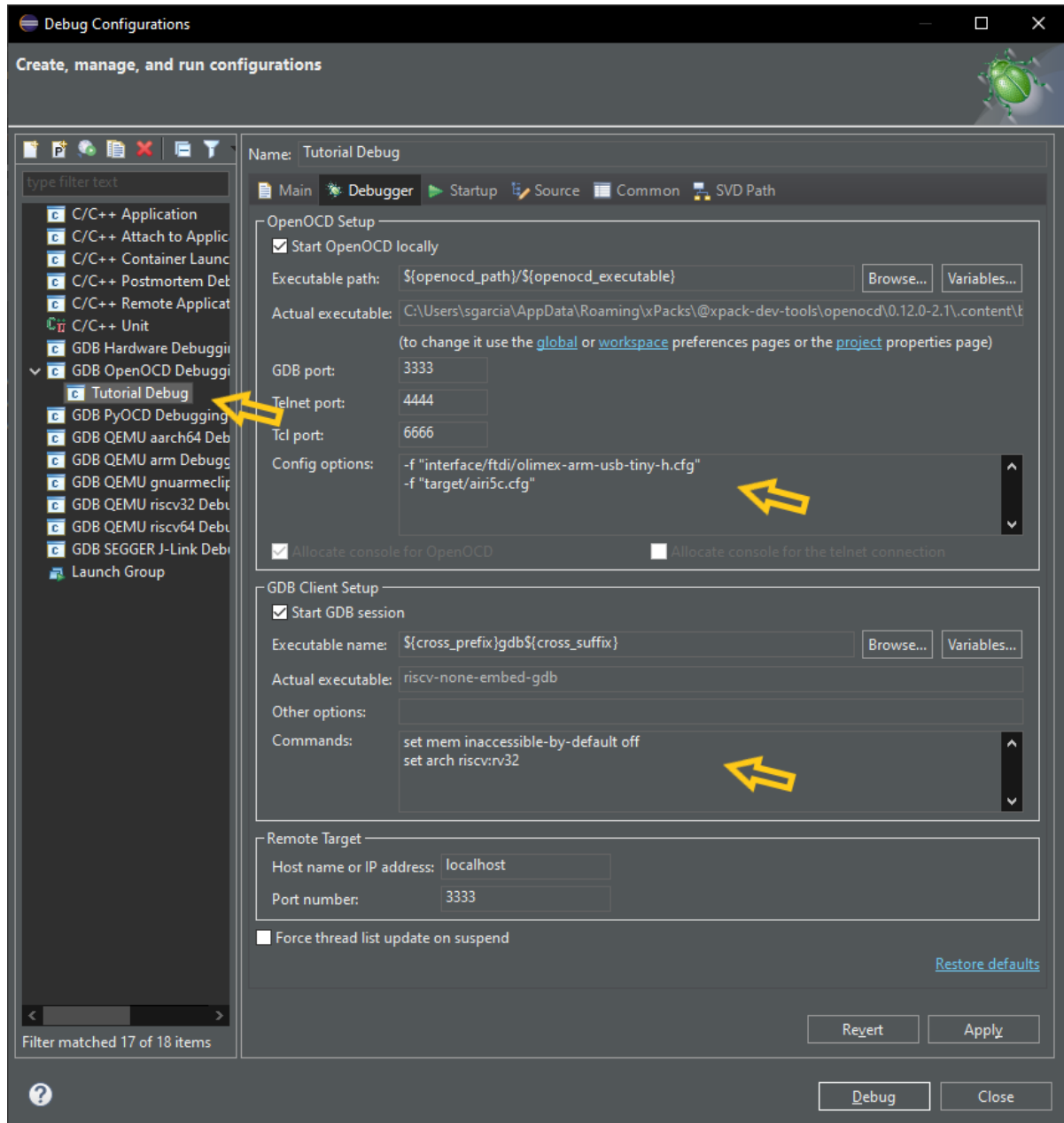


Figure 37: Configure the debbuger

4.3.3 Main Configuration

Make sure the ELF file is specified as the *C/C++ Application*, select *Apply* and *Debug*.

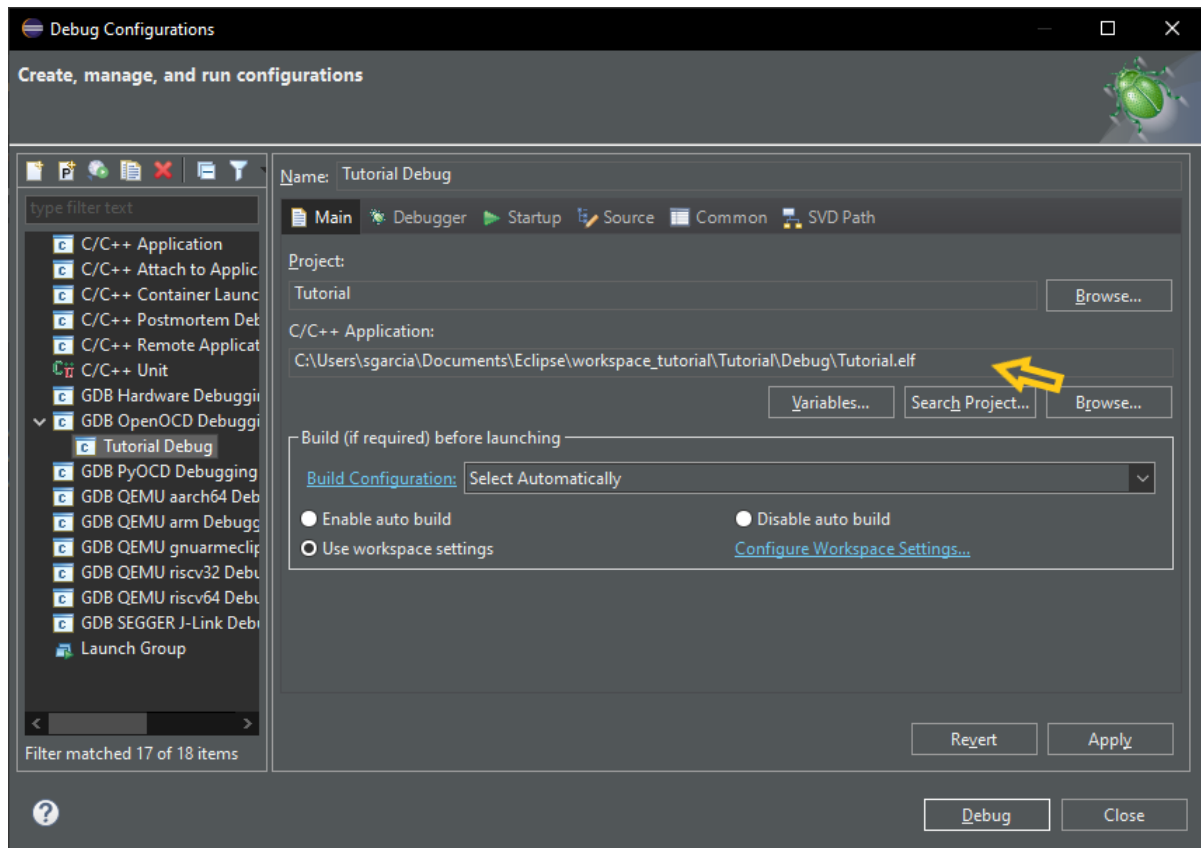


Figure 38: Select ELF file

4.4 Debugging

The IDE switches to the *Debug* perspective where the user can select different option for debugging from the toolbar, e.g. *Step Into* as shown in Figure 39. The *Disassembly* view can be shown selecting *Window* → *Show View* → *Disassembly*.

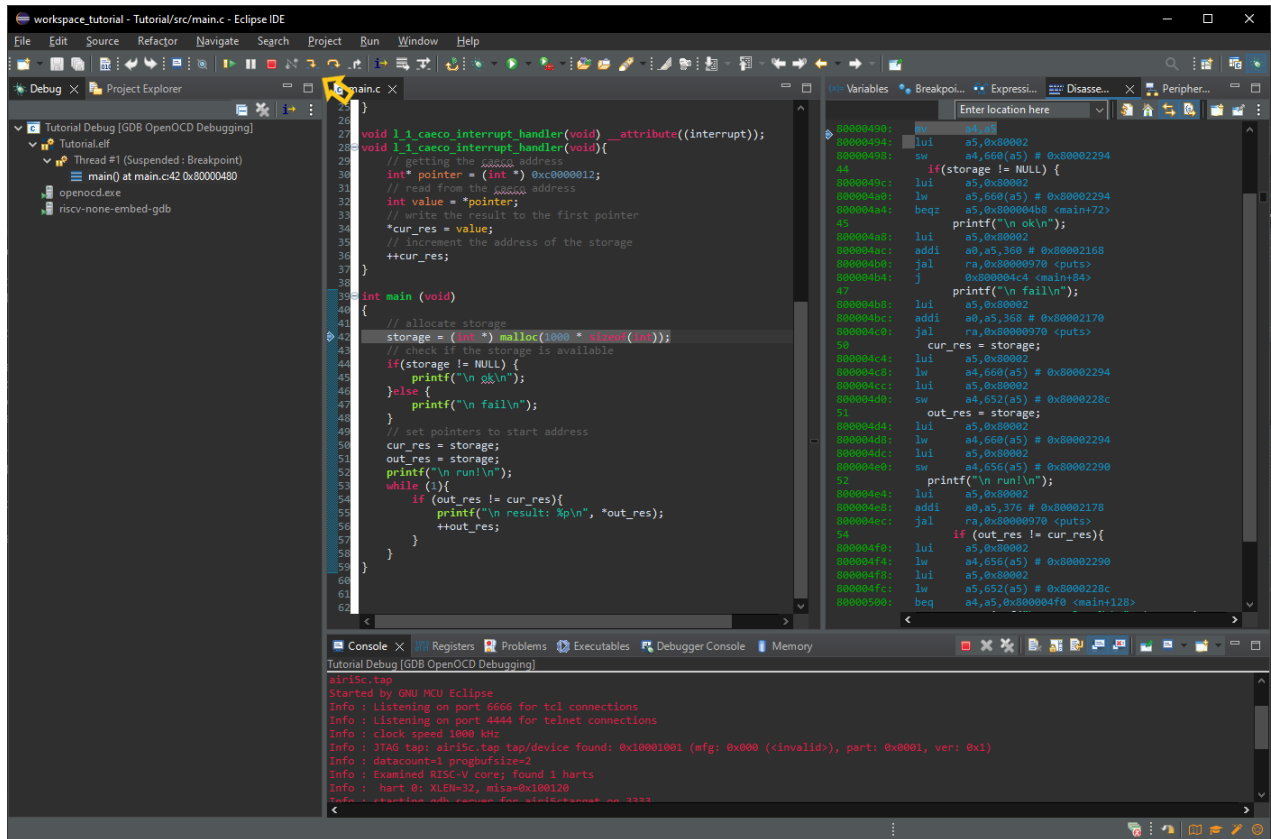


Figure 39: Debug perspective

In the HTerm terminal after connecting to the correct port and defining a data transmission of 115200 bauds, the printf messages will be visible in the terminal when debugging as shown in Figure 40.

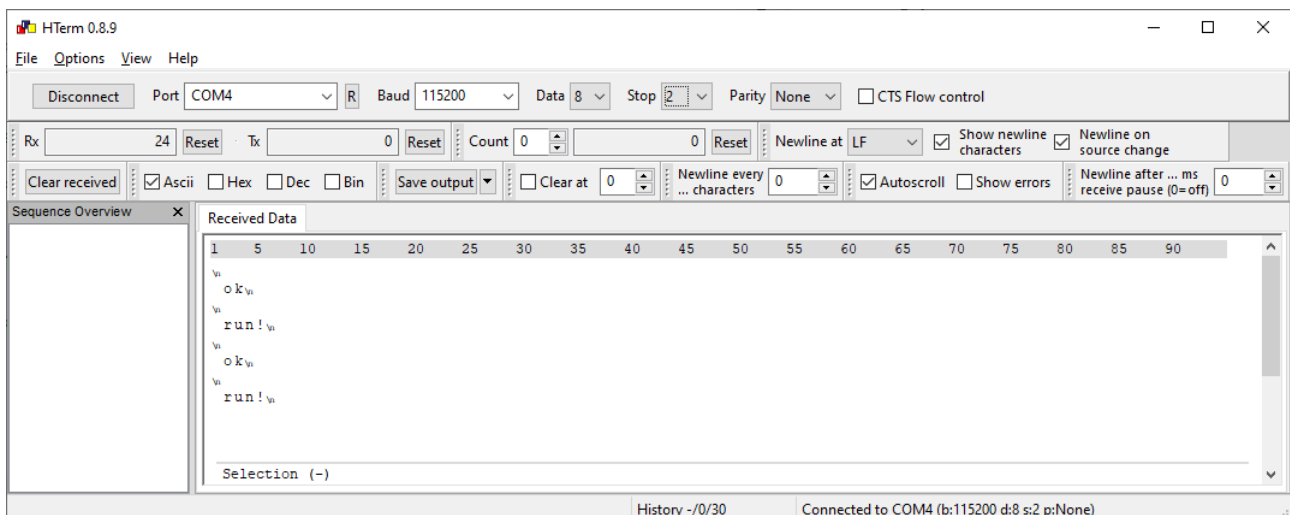


Figure 40: HTerm terminal during debugging

4.5 Optional: Debugging on Linux server

Logging to the server provided by the lab makes easier the debugging process since all the tools are already installed in form of modules. The first step is loading the modules and run Eclipse:

```
$ module load Eclipse/gnu-mcu-eclipse/2019-12
$ module load GCC/riscv-none-embed-gcc/8.3.0-1.2.1
$ module load OpenOCD/0.10.0-14.3
$ eclipse
```

Listing 15: Loading modules on lab server

Create a new C/C++ project and select the same configuration as when working on Windows. Select the Toolchain name and configure the Toolchain path as shown in Figure 41.

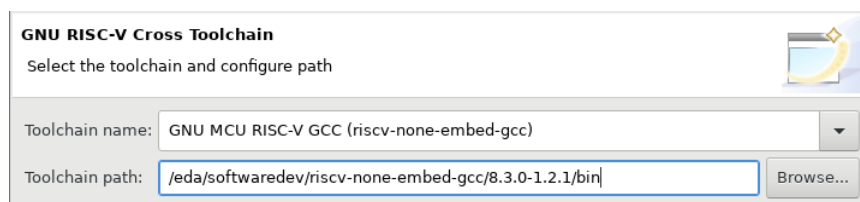


Figure 41: Select the toolchain and path

Add the files to the *src* folder, and configure the *Project Properties* related to *C/C++ Build* in the same way as on Windows. However, in this case no paths must be specified in the *MCU configuration* (Build Tools, OpenOCD, RISC-V Toolchain).

Since the board is connected to the PC and not to the server, a remote connection for debugging is required. First, on the PC run the command to execute OpenOCD adding to the command the configuration files and the option to listen for incoming TCP/IP connections (*bindto*). The syntax of the command is shown below:

```
<path\openocd.exe> -c "adapter speed 10000" -c "bindto 0.0.0.0" -f <path\olimex-arm-usb-tiny-h.cfg> -f <path\airi5c.cfg> -c "init"
```

Listing 16: Syntax to run OpenOCD with configuration files and listening to TCP/IP connections

The full command in my case for my paths is the following:

```
C:\Users\sgarcia\AppData\Roaming\xPacks\@xpack-dev-tools\openocd\0.12.0-2.1\content\bin\openocd.exe -c "adapter speed 10000" -c "bindto 0.0.0.0" -f C:\Users\sgarcia\AppData\Roaming\xPacks\@xpack-dev-tools\openocd\0.12.0-2.1\content\openocd\scripts\interface\ftdi\olimex-arm-usb-tiny-h.cfg -f C:\Users\sgarcia\AppData\Roaming\xPacks\@xpack-dev-tools\openocd\0.12.0-2.1\content\openocd\scripts\target\airi5c.cfg -c "init"
```

Listing 17: Execute OpenOCD specifying configuration files and listening to TCP/IP connections

Finally, to debug on the FPGA the *Debug Configurations* must be set. In the *Debugger Tab* for your *GDB OpenOCD Debugging* configurations disable the option *Start OpenOCD locally*, add the additional command for the *GDB Client Setup* and define the PC IP address to connect remotely as presented in 42. With the command *ipconfig* you can display the IP address for your PC, this is found under the name *IPv4 Address*.

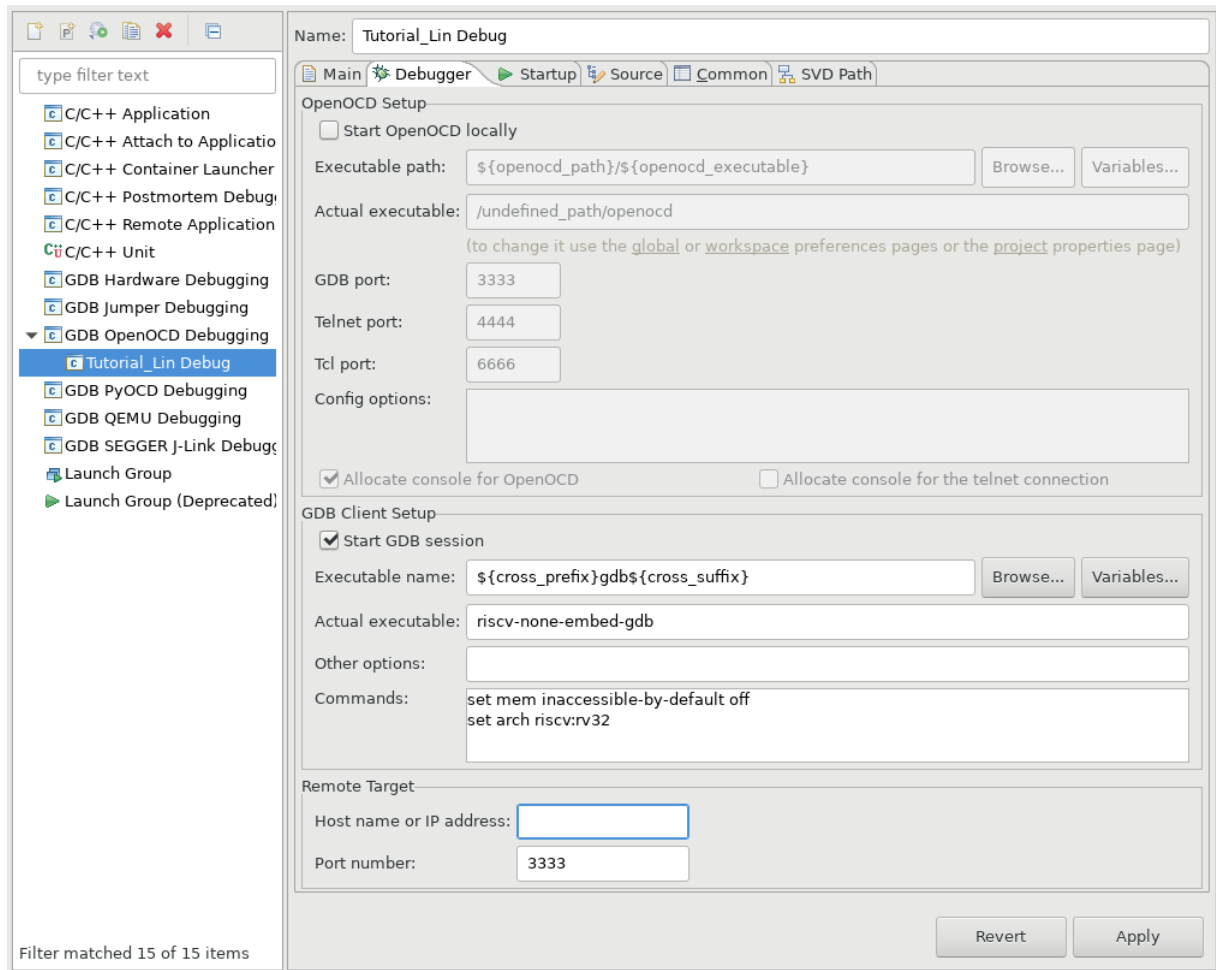


Figure 42: Debug configuration for debug on Remote Target

5 Verification with Python

In this section, the behaviour of the RISC-V core is verified with a Python script and the JTAG adapter. The software tool JupyterLab is a web-based interactive development environment that allows the user to code and analyse data section by section making the workflow more understandable [18].

The file with the extension *ipynb* is the file used by JupyterLab. The first cell is not represented as code, instead describes as a *raw cell* the syntax of the OpenOCD command required to connect to the FPGA and debug the python code using the JTAG adapter.

```
Init OpenOCD
<path\openocd.exe> -c "adapter speed 10000" -f <path\olimex-arm-usb-tiny-h.cfg>
-f <path\airi5c.cfg>

C:\Users\sgarcia\AppData\Roaming\xPacks\@xpack-dev-tools\openocd\0.12.0-2.1\content
\bin\openocd.exe -c "adapter speed 10000" -f "C:\Users\sgarcia\AppData\Roaming\xPacks
\@xpack-dev-tools\openocd\0.12.0-2.1\content\openocd\scripts\interface\ftdi\olimex-
arm-usb-tiny-h.cfg" -f "C:\Users\sgarcia\AppData\Roaming\xPacks\@xpack-dev-tools\
openocd\0.12.0-2.1\content\openocd\scripts\target\airi5c.cfg"
```

Listing 18: Run OpenOCD with configuration files

In the next cell, the necessary packages are imported and functions are defined. The function `read_ecg` reads the ECG file and returns the data as a list. The `progressbar` function displays a representation of the data written. Before the explanation of the following functions, it is important to remark the two available alternatives to interact with the CAECO accelerator.

The first option is accessing directly to CAECO to write the ECG data, start the calculation and read the result. This is achieved writing and reading to specific addresses defined in `raifes_dmi_constants.vh`. When these addresses are received by the Debug Module Interface (DMI), which is part of the raifes Debug Module (DM), the `caeco_interface` module handles the data specified for these addresses for a direct access to CAECO and without requiring to access to the RISC-V before using the CAECO. A graphical representation of the data flow for this option is presented in Figure 43 showing with red arrows the way in which the CAECO is accessed.

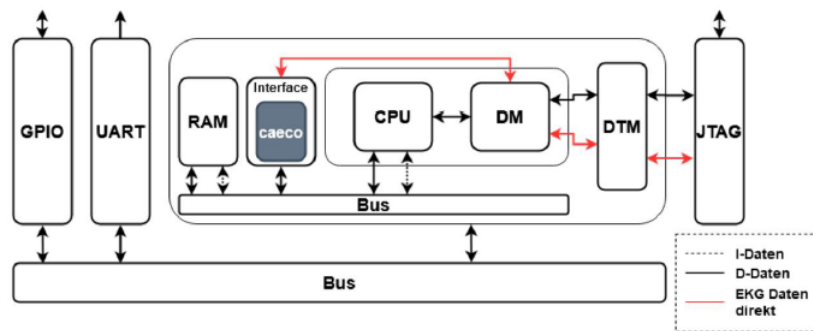


Figure 43: Data flow in the RV32IM system architecture

The second option to access to CAECO is through the RISC-V. By interacting with the RISC-V in debug mode using the DMI, it is possible to write and read to registers in the RAM. The RAM serves as memory for the instructions and data of the RISC-V. Once the data has been written to the memory, the CAECO gets the data from the RAM through the BUS as shown in Figure 43. The addresses used to access to CAECO via the RISC-V are different than those used for a direct access with the DM as shown in Tables 2 and 3.

Address	Function
0x22	Write ECG data
0x23	Write CMD
0x24	Read Result

Table 2: Addresses to write and read directly to CAECO

Address	Function
0xc0000010	Write ECG data
0xc0000011	Write CMD
0xc00000c0	Read Result

Table 3: Addresses to write and read to CAECO via RISC-V

The address 0x24 used to directly read from the CAECO does not really represent a memory address, but is only used to notify that the `RESULT` signal of the CAECO module has to be read by the DM and send via JTAG. However, if the result of CAECO is read through the RISC-V, then the 0xc00000c0 is a memory address containing the last result of CAECO.

The sequence of commands required to process the ECG data with CAECO are as shown in 4. To choose between the two options to access to CAECO, a first command is sent. Depending on the address specified for the

first command the `caecointerface` module determines if the CAECO is accessed directly or via the RISC-V. After the first command, the following commands control the CAECO. The second command resets CAECO and the third command starts the computation.

Command	Function
0x0	Initialize
0x10	Reset
0x11	Start Computation

Table 4: Commands to control CAECO

Now that the two available options have been clarified, the functions in the python script can be described. The two main functions are: `caeco_dmi` and `caeco_mcu`. Two arguments must be provided to both functions, the `openocd` object and the ECG data desired to be sent to CAECO. The function `caeco_dmi` writes the ECG data directly to the CAECO, whereas `caeco_mcu` uses the RISC-V. Both functions read the result to their respective address, display an image corresponding to the given result and return the result.

Each function handles the process of writing and reading via the JTAG in different manners. Function `caeco_dmi` uses functions `openocd_dmi_write` and `openocd_dmi_read` to build write/read OpenOCD RISC-V DMI commands [19]. These strings are sent using the `send` method of `openocd` object. In contrast, function `caeco_mcu` uses functions `openocd_write_mem` and `openocd_read_mem` to send the instructions and data to the RISC-V in debug mode to write and read to the desired addresses in the memory.

Accessing directly to the memory using General OpenOCD Commmands(`mdw`) is not possible unless the core is halted as shown in Figure 44. For this reason the only way to write and read to memory is through the DMI.

```
Open On-Chip Debugger
> mdw 0x80000000
Failed to read memory via system bus.
Buggy aampostincrement! Address not incremented correctly.
0x80000000: 00000000
> halt
> mdw 0x80000000
0x80000000: 00000197
> resume
```

Figure 44: Accessing to the memory using General OpenOCD commands

Once we have defined the functions and OpenOCD has been executed specifying the configuration files, a connection can be established using the `connect` method as shown in the next cell of the Python script. The data set is then defined specifying the ECG files paths and finally in the next cells, the `read_ecg` function is called, ECG data is plotted and the two different functions to access CAECO are defined as presented in Figures 45, 46 and 47.



Figure 45: Calling *read_ecg* and plotting ECG data

```
""" ----- W/R directly to CAECO ----- """
res = caeco_dmi(o, chs)

Writing data directly to CAECO
Initial Caeco result: 0x0

Writing CMD commands

Writing ECG data...
[=====]
Writing CMD to indicate last ECG value

Calculating...
New Caeco result: 0x1
Result: Krank
```

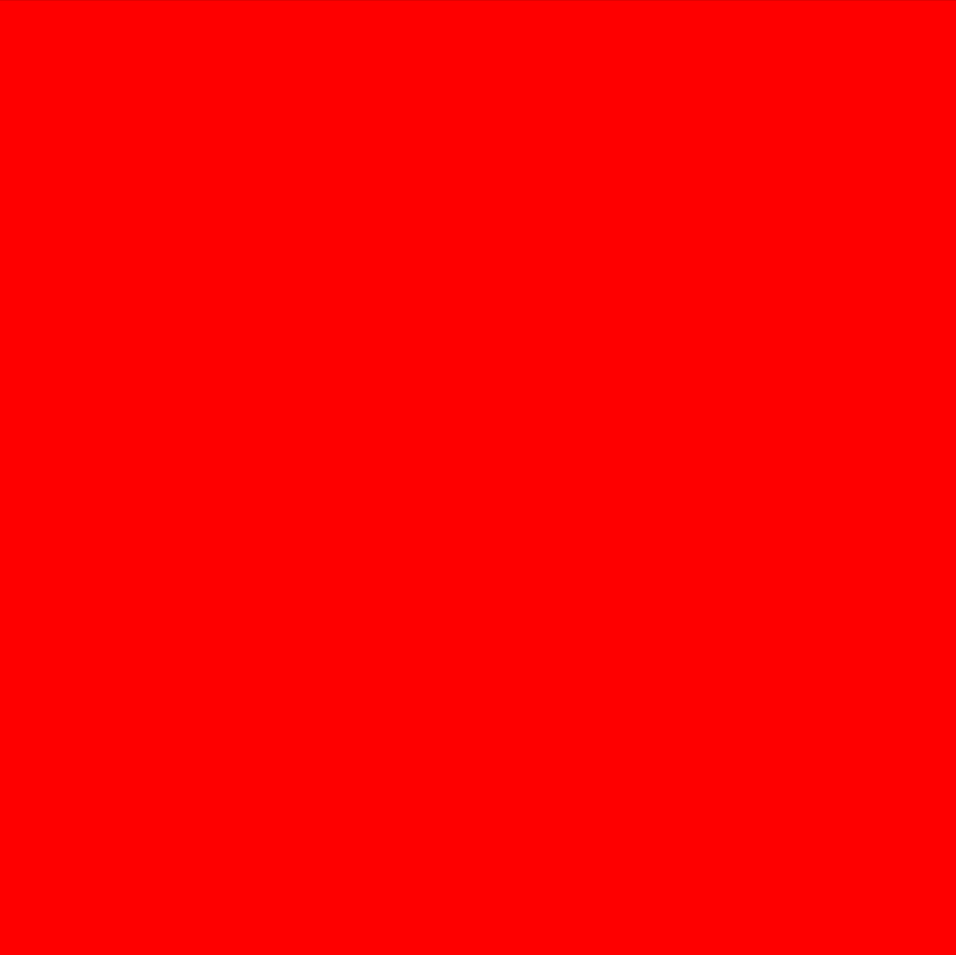


Figure 46: Calling *caeco_dmi* to directly write and read to CAECO

```
""" ----- W/R to CAECO via RISC-V ----- """
res = caeco_mcu(o, chs)

Writing data via RISC-V
Reading address: 0xc00000c0
Initial Caeco result: 0x0

Writing CMD commands

Writing ECG data...
[=====]
Writing CMD to indicate last ECG value

Calculating...
Reading address: 0xc00000c0
New Caeco result: 0x1
Result: Krank
```

Figure 47: Calling *caeco_mcu* to write and read to CAECO via RISC-V

Bibliography

- [1] Hendrik Wöhrle et al. "Surrogate Model based Co-Optimization of Deep Neural Network Hardware Accelerators". In: *2021 IEEE International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2021, pp. 40–45. DOI: [10.1109/MWSCAS47672.2021.9531708](https://doi.org/10.1109/MWSCAS47672.2021.9531708).
- [2] Fabian Brünger. *Integration eines Hardwarebeschleunigers für Maschinelles Lernen in einen RISC-V RV32IM Prozessor über Memory-Mapped Register*. Master's thesis. Aug. 2020.
- [3] DIGILENT. Nexys A7. URL: <https://digilent.com/reference/programmable-logic/nexys-a7/start>. (accessed: 24.11.2023).
- [4] OLIMEX. ARM-USB-TINY-H. URL: <https://www.olimex.com/Products/ARM/JTAG/ARM-USB-TINY-H/>. (accessed: 24.11.2023).
- [5] Zadig. Zadig. URL: <https://zadig.akeo.ie/>. (accessed: 24.11.2023).
- [6] Digilent. *Install Digilent's Board Files*. URL: <https://digilent.com/reference/programmable-logic/guides/install-board-files>. (accessed: 24.11.2023).
- [7] Eclipse Foundation. *Eclipse IDE for Embedded C/C++ Developers*. URL: <https://www.eclipse.org/downloads/packages/release/2023-09/r/eclipse-ide-embedded-cc-developers>. (accessed: 22.11.2023).
- [8] Eclipse Embedded CDT. *Eclipse Embedded CDT*. URL: <https://eclipse-embed-cdt.github.io/>. (accessed: 22.11.2023).
- [9] Eclipse Foundation. *Eclipse IDE 2023-09 R Packages*. URL: <https://www.eclipse.org/downloads/packages/>. (accessed: 21.11.2023).
- [10] The xPack Project. *The xPack GNU RISC-V Embedded GCC*. URL: <https://xpack.github.io/dev-tools/riscv-none-elf-gcc/>. (accessed: 22.11.2023).
- [11] node.js. *Download Node.js®*. URL: <https://nodejs.org/en>. (accessed: 21.11.2023).
- [12] The xPack Project. *The xPack GNU RISC-V Embedded GCC*. URL: <https://github.com/xpack-dev-tools/riscv-none-embed-gcc-xpack>. (accessed: 22.11.2023).
- [13] The xPack Project. *The xPack GNU RISC-V Embedded GCC*. URL: <https://github.com/xpack-dev-tools/riscv-none-elf-gcc-xpack>. (accessed: 22.11.2023).
- [14] The xPack Project. *The xPack Windows Build Tools*. URL: <https://xpack.github.io/dev-tools/windows-build-tools/>. (accessed: 22.11.2023).
- [15] The xPack Project. *The xPack OpenOCD*. URL: <https://xpack.github.io/dev-tools/openocd/>. (accessed: 22.11.2023).
- [16] der-hammer. *HTerm*. URL: <https://www.der-hammer.info/pages/terminal.html>. (accessed: 22.11.2023).
- [17] OpenOCD. *6 Config File Guidelines*. URL: <https://openocd.org/doc/html/Config-File-Guidelines.html#Target-Config-Files>. (accessed: 22.11.2023).
- [18] Jupyter. *Jupyter*. URL: <https://jupyter.org/>. (accessed: 11.12.2023).
- [19] OpenOCD. *16 Architecture and Core Commands*. URL: <https://openocd.org/doc/html/Architecture-and-Core-Commands.html>. (accessed: 11.12.2023).