

# NuEclipse User Manual

*The information described in this document is the exclusive intellectual property of Nuvoton Technology Corporation and shall not be reproduced without permission from Nuvoton.*

*Nuvoton is providing this document only for reference purposes of NuMicro microcontroller based system design. Nuvoton assumes no responsibility for errors or omissions.*

*All data and specifications are subject to change without notice.*

For additional information or questions, please contact: Nuvoton Technology Corporation.

[www.nuvoton.com](http://www.nuvoton.com)

## Table of Contents

<b>1 OVERVIEW .....</b>	<b>6</b>
1.1 Supported Chips .....	6
<b>2 REQUIREMENTS .....</b>	<b>7</b>
<b>3 QUICK START .....</b>	<b>8</b>
3.1 Installation.....	8
3.1.1 Performing the NuEclipse Installer on Microsoft Windows .....	8
3.1.2 Extracting the NuEclipse Tar File on GNU/Linux.....	9
3.1.3 Verifying the Eclipse Preferences .....	10
3.2 Running Eclipse .....	12
<b>4 DEVELOPMENT TUTORIAL .....</b>	<b>13</b>
4.1 Select Workspace .....	13
4.2 New Project Wizard.....	14
4.3 Import Existing Projects .....	16
4.4 Build Settings.....	17
4.5 Debug Configuration .....	18
4.5.1 Debugger Tab .....	19
4.5.2 Startup Tab.....	20
4.6 Debug Views.....	21
4.6.1 Registers View.....	21
4.6.2 Memory View .....	22
4.6.3 Disassembly View .....	23
4.6.4 Peripheral Registers View .....	24
4.7 Watchpoints .....	30
4.8 Debug in RAM.....	32
4.9 Debug Executable Files Only .....	36
4.10 Configuration of Cortex-A, Multi-core SMP and AMP system .....	42
4.10.1 Preliminary Preparation.....	42
4.10.2 Configuration of debugging Single-core Cortex-A .....	43
4.10.3 Configuration of debugging Dual-core Cortex-A in SMP mode.....	47
4.10.4 Configuration of debugging Single-core Cortex-A and Cortex-M in AMP mode ...	47
4.10.5 Configuration of debugging Dual-core Cortex-A and Cortex-M in AMP mode.....	49
4.10.6 Configuration of debugging Cortex-A Core in AARCH32 mode.....	50
4.11 Configuration of NuMicro 8051 1T.....	53
4.11.1 Import and Build Project.....	53
4.11.2 Debug Configuration.....	53
4.11.3 Monitor different Memory Space.....	54
<b>5 Q&amp;A .....</b>	<b>57</b>

6 REVISION HISTORY .....61

## List of Figures

Figure 3-1 NuEclipse Setup Wizard .....	8
Figure 3-2 Install.sh Script .....	9
Figure 3-3 Preferences for Global Tools Paths .....	10
Figure 3-4 Preferences for OpenOCD Nu-Link .....	11
Figure 3-5 Eclipse.exe and Related Folders .....	12
Figure 4-1 Selecting a Workspace .....	13
Figure 4-2 New Project Wizard .....	14
Figure 4-3 Target Processor Settings .....	15
Figure 4-4 Importing Projects .....	16
Figure 4-5 Build Settings .....	17
Figure 4-6 Debug Configuration .....	18
Figure 4-7 Configuring the Debugger Tab .....	19
Figure 4-8 Configuring the Startup Tab .....	20
Figure 4-9 Registers View .....	21
Figure 4-10 Memory View .....	22
Figure 4-11 Clicking the Instruction Stepping Mode Button .....	23
Figure 4-12 Disassembly View .....	23
Figure 4-13 Opening the Packs Perspective .....	24
Figure 4-14 How to Download Packages .....	25
Figure 4-15 Locations of Repositories .....	26
Figure 4-16 Installing SFR Files .....	27
Figure 4-17 Device Selection .....	28
Figure 4-18 Peripheral Registers View .....	29
Figure 4-19 Toggle Watchpoint .....	30
Figure 4-20 Properties for C/C++ Watchpoint .....	31
Figure 4-21 Added Watchpoint in the Breakpoints View .....	31
Figure 4-22 Memory Layout .....	32
Figure 4-23 Modifying the Id Script .....	33
Figure 4-24 Debug Configuration Settings .....	34
Figure 4-25 Debugging in RAM .....	35
Figure 4-26 Importing Executable for Debugging .....	36
Figure 4-27 Selecting Executable .....	37
Figure 4-28 Choosing GDB Nuvoton Nu-Link Debugging .....	38
Figure 4-29 Locating the GDB Executable .....	39
Figure 4-30 Choosing the ELF File to Download .....	40
Figure 4-31 Adding Source Lookup Path .....	41

Figure 4-32 Configuring the Startup Tab for MA35D1 .....	43
Figure 4-33 How to Disable Command Timeout .....	44
Figure 4-34 Disable CPU1 Setting .....	45
Figure 4-35 Configuring the Debugger Tab for Single-core Debugging on MA35D1 .....	46
Figure 4-36 Configuring the Debugger Tab for Multi-core Debugging on M4 core .....	47
Figure 4-37 Configuring the Launch Group .....	48
Figure 4-38 How to Disable SMP .....	49
Figure 4-39 Multi-core Debugging .....	49
Figure 4-40 Locating the AARCH64 GDB Executable .....	51
Figure 4-41 The First Time to Enter the Debug Mode Settings .....	51
Figure 4-42 The Second Time to Enter the Debug Mode Settings .....	52
Figure 4-43 AARCH32 Program Debugging .....	52
Figure 4-44 Build NuMicro 8051 1T project.....	53
Figure 4-45 NuMicro 8051 1T Debug Configuration in Debugger Tab .....	54
Figure 4-46 Access Memory Space with Memory Browser.....	55
Figure 4-47 Access Memory Space with Memory View .....	55
Figure 4-48 Access Variable with Watch Expressions View .....	56
Figure 5-1 Adding Udev Rules .....	58
Figure 5-2 Preferences for String Substitution .....	59
Figure 5-3 More Options for NuLink2 .....	60

## 1 OVERVIEW

The **NuEclipse** is designed for cross-platform embedded MCU development. It includes a series of Eclipse plug-ins and tools. The plug-ins allow the user to create, build, and debug ARM-based and NuMicro 8051 1T projects within the Eclipse framework. Its features are listed below:

- Creating projects by the New Project Wizard: The New Project Wizard provides several templates for different target chips.
- Building projects by the GNU ARM Toolchain: The toolchain contains the ARM Embedded GCC compiler. The user can use it to build projects without restriction.
- Debugging projects by GDB: The user can halt, step, run, and monitor target chips. Accessing memory and flash is allowed. Setting hardware breakpoints and watchpoints is supported. In addition, the user can erase target chips and program the user configuration.

Through the **NuEclipse**, the user can develop projects of the NuMicro® Family within the Eclipse framework.

### 1.1 Supported Chips

To see the list of supported chips, please refer to **Supported\_chips.htm** in the folder of user manual.

## 2 REQUIREMENTS

The following table lists system requirements for the user to run the **NuEclipse**.

	Minimum Requirements	Recommended Specifications
<b>Operating System</b>	Windows®7 x64 or GNU/Linux	Windows®10 x64 or Ubuntu 18.04 LTS
<b>GNU ARM Embedded Toolchain</b>	10.3-2021.10	The latest version

**Note:** To have a fully usable and pleasant experience on Linux, the recommended Linux distribution is Ubuntu 18.04 LTS (64-bit).

### 3 QUICK START

#### 3.1 Installation

To make the **NuEclipse** ready for work, perform the following steps based on your operating system:

1. Performing the NuEclipse installer on Microsoft Windows.
2. Extracting the NuEclipse tar file on GNU/Linux.

##### 3.1.1 Performing the NuEclipse Installer on Microsoft Windows

On Windows, it is very easy to install the NuEclipse only by performing the NuEclipse installer. The installer will ask the user to install the **GNU ARM Eclipse Windows Build Tools** and **GNU ARM Embedded Toolchain** because they are required by NuEclipse.

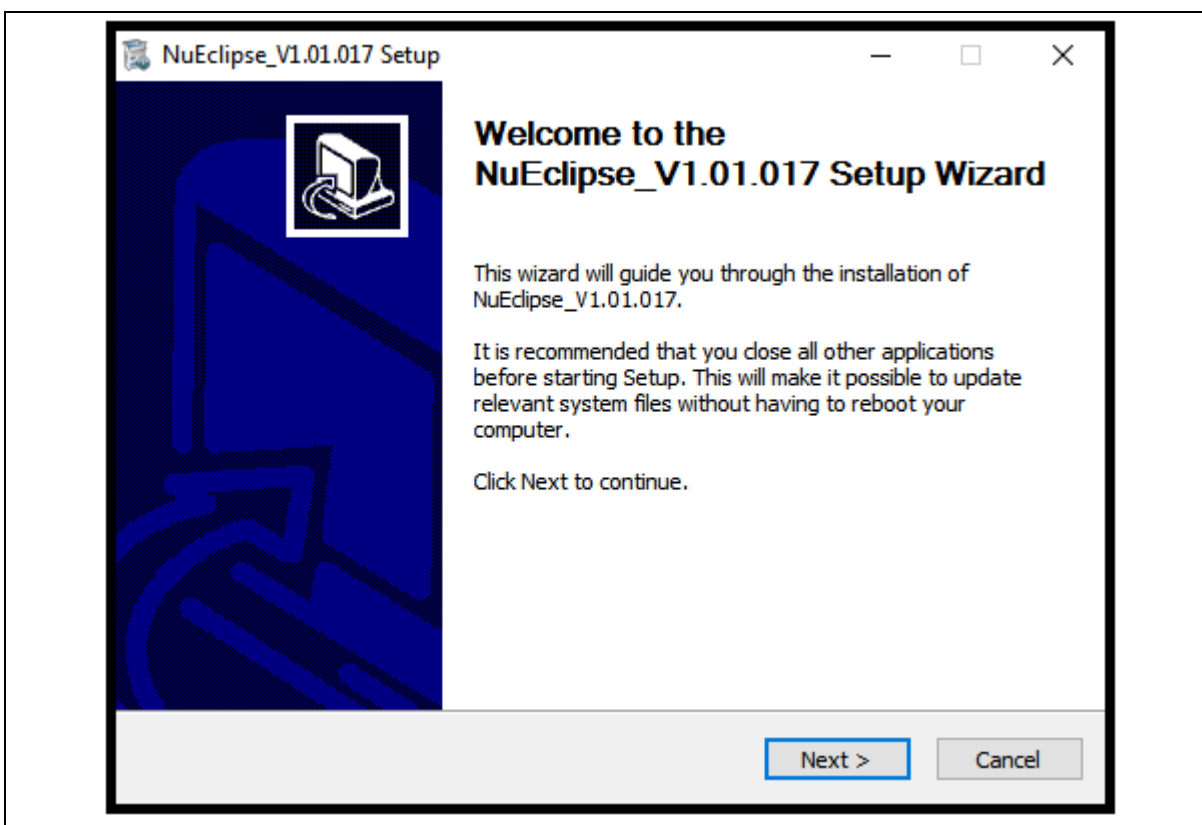


Figure 3-1 NuEclipse Setup Wizard



3.1.2     Extracting the NuEclipse Tar File on GNU/Linux

On GNU/Linux, it is very easy to install the NuEclipse only by extracting the NuEclipse tar file. After that, **run the install.sh script and re-login** to complete the installation process. Please do not use the **sudo** command to run the script.

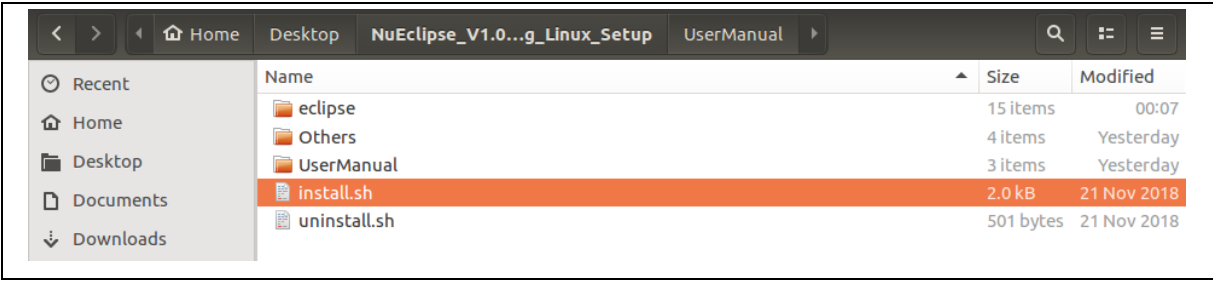


Figure 3-2 Install.sh Script

### 3.1.3 Verifying the Eclipse Preferences

After the installation, the Eclipse preferences are automatically written on Windows. To verify them, click **Window > Preferences**, the Preferences wizard appears. Go to **C/C++ > Build > Global Tools Paths** and make sure the Build tools and Toolchain folder be correctly configured to what the installer has installed in the previous step. Click the **Apply** button to take effect. On GNU/Linux, Build tools folder path is not required. The path should be empty.

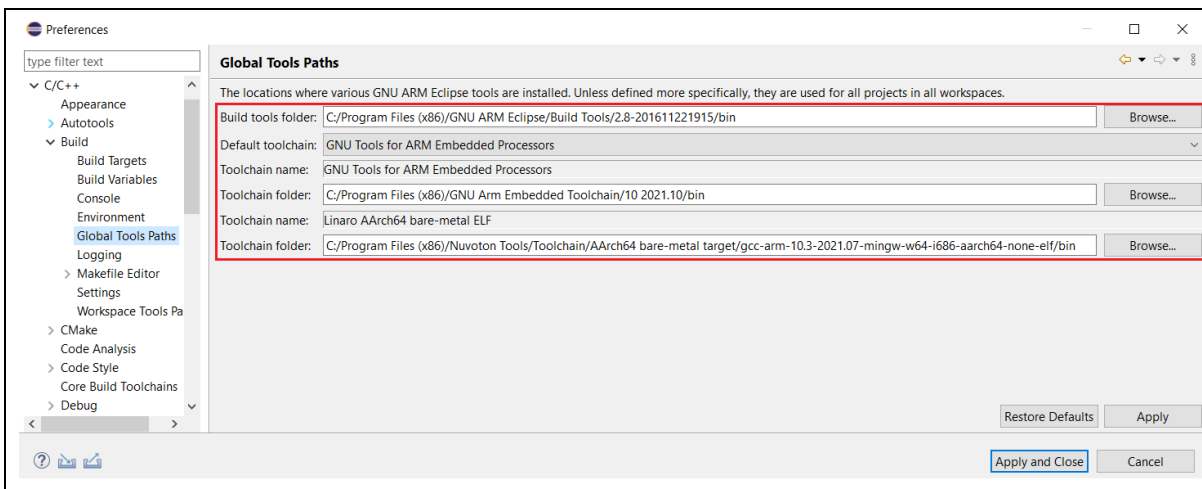


Figure 3-3 Preferences for Global Tools Paths

Subsequently, go to **Run/Debug > OpenOCD Nu-Link** and make sure the OpenOCD folder be configured to where the installer has put the OpenOCD executable. On Microsoft Windows, for example, the path of OpenOCD folder could be C:/Program Files (x86)/Nuvoton Tools/OpenOCD. Similarly, on GNU/Linux it could be /usr/local/OpenOCD. The OpenOCD executable provided by Nuvoton is customized for Nu-Link. If the user tries to use other OpenOCD executable, OpenOCD and Nu-Link may not cooperate well. Click the **Apply** button to take effect.

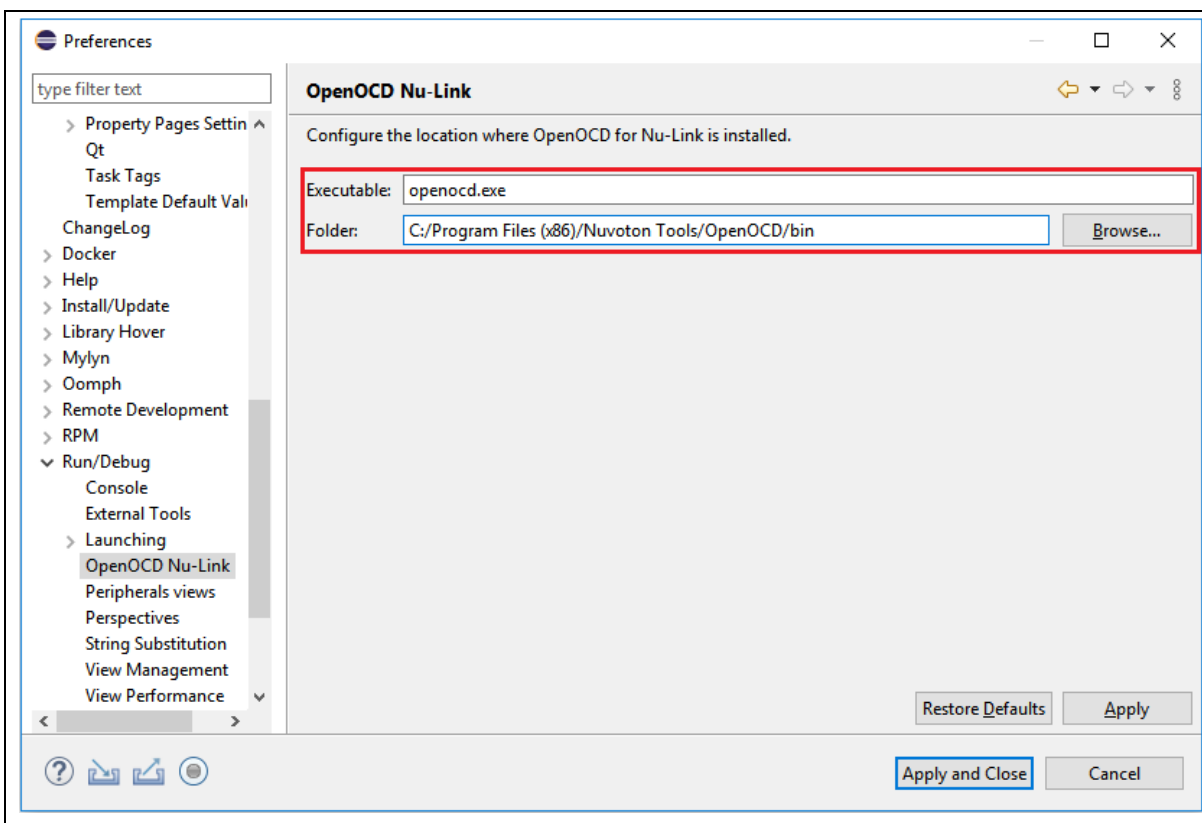


Figure 3-4 Preferences for OpenOCD Nu-Link

## 3.2 Running Eclipse

To run **NuEclipse**, double-click the **eclipse.exe**. Note that the .exe file and the related folders, such as the OpenOCD folder, should stay in the same directory; otherwise, the application will not work properly.

Name	Date modified	Type	Size
configuration	2020/10/22 下午 05:41	File folder	
dropins	2020/10/22 下午 05:41	File folder	
features	2020/10/22 下午 05:41	File folder	
OpenOCD	2020/10/22 下午 05:41	File folder	
p2	2020/11/30 下午 03:48	File folder	
Packages	2020/10/22 下午 05:41	File folder	
plugins	2020/11/17 下午 04:26	File folder	
readme	2020/10/22 下午 05:46	File folder	
.eclipseproduct	2020/9/2 下午 10:06	ECLIPSEPRODUCT...	1 KB
artifacts.xml	2020/10/22 下午 05:23	XML Document	170 KB
<b>eclipse.exe</b>	2020/9/10 上午 11:05	Application	417 KB
eclipse.ini	2020/10/20 上午 10:51	Configuration sett...	1 KB
eclipsesec.exe	2020/9/10 上午 11:05	Application	129 KB
notice.html	2020/9/7 下午 09:35	HTML Document	10 KB

Figure 3-5 Eclipse.exe and Related Folders

## 4 DEVELOPMENT TUTORIAL

### 4.1 Select Workspace

When Eclipse launches, we have to select a workspace which groups a set of related projects together that usually make up an application. In addition, some configuration settings for Eclipse and projects are stored here, too. For different computers, the configuration settings may change. We should create our own workspace rather than copying another user's workspace. Only one workspace can be active at one time. The current workspace for Eclipse can be switched by clicking **File->Switch Workspace**.

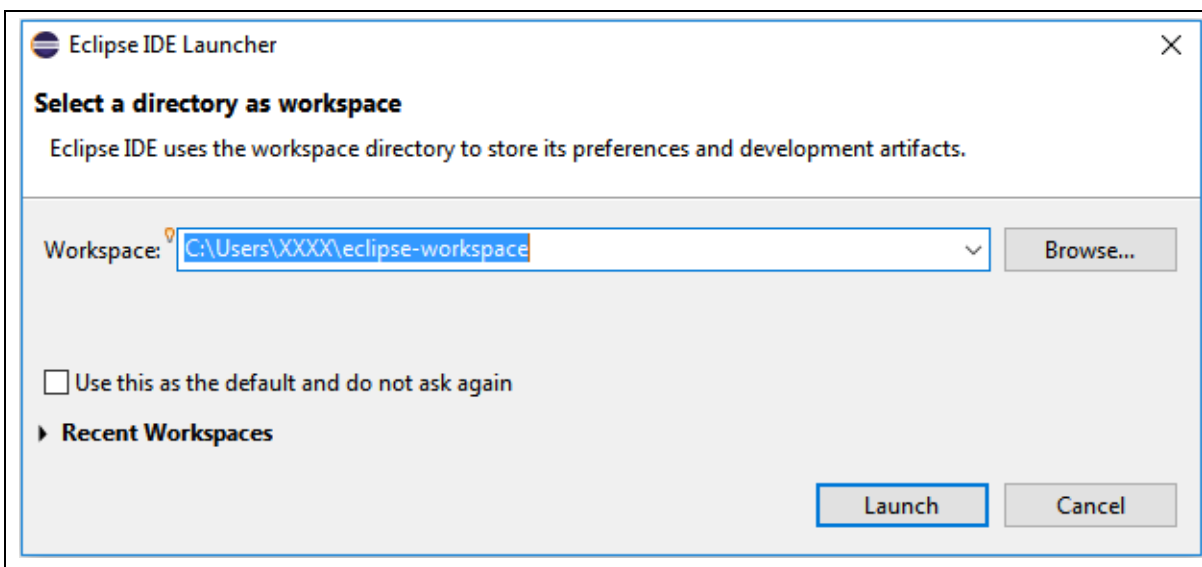


Figure 4-1 Selecting a Workspace

## 4.2 New Project Wizard

As a beginner, the fastest way to create a C/C++ project is using the New Project Wizard. For instance, to create a C project, click **File > New > C Project**. The New Project Wizard appears. Here we choose **Hello World Nuvoton Cortex-M C Project** for Project type. Input the project name and click the **Next >** button to continue.

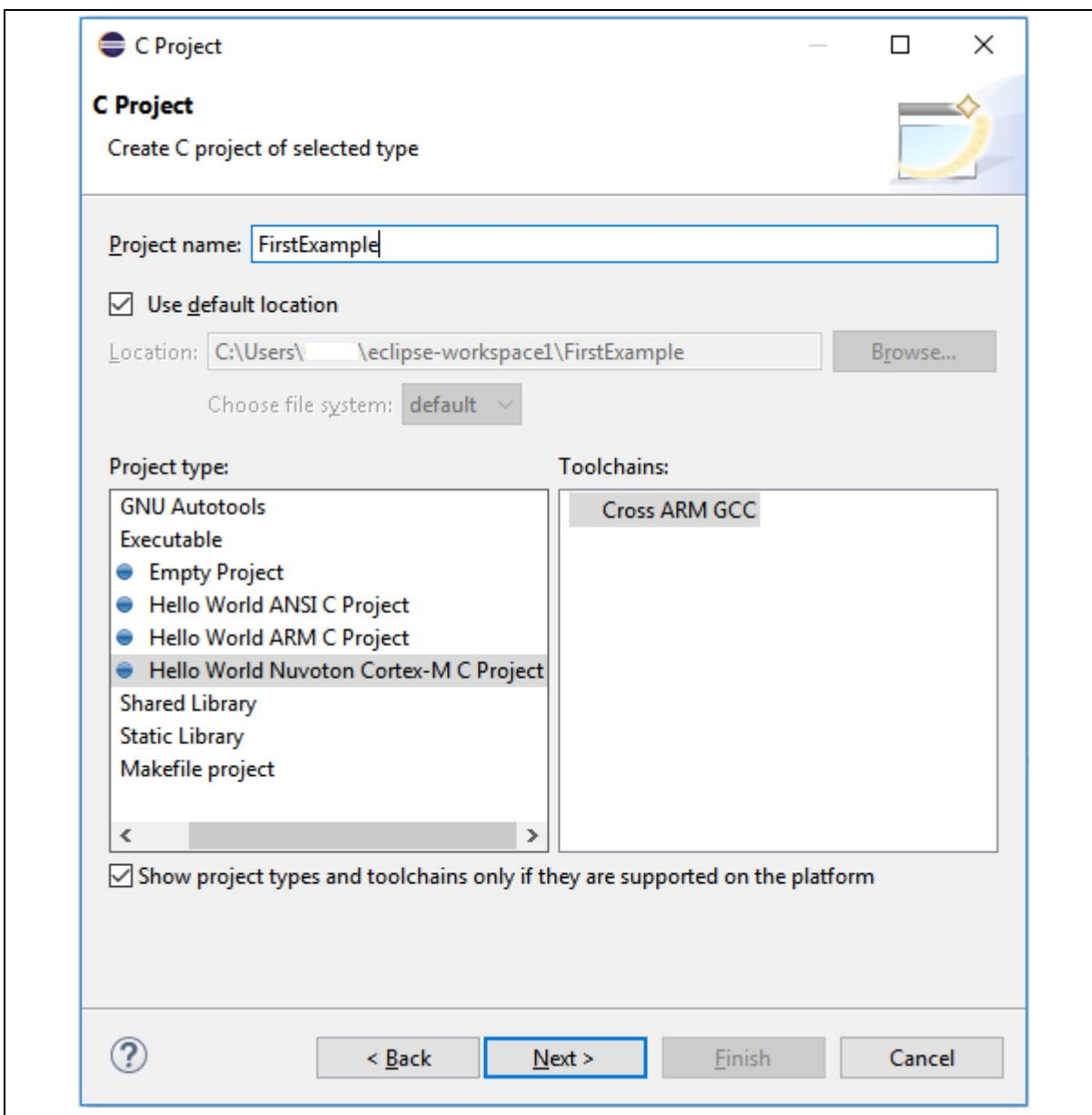
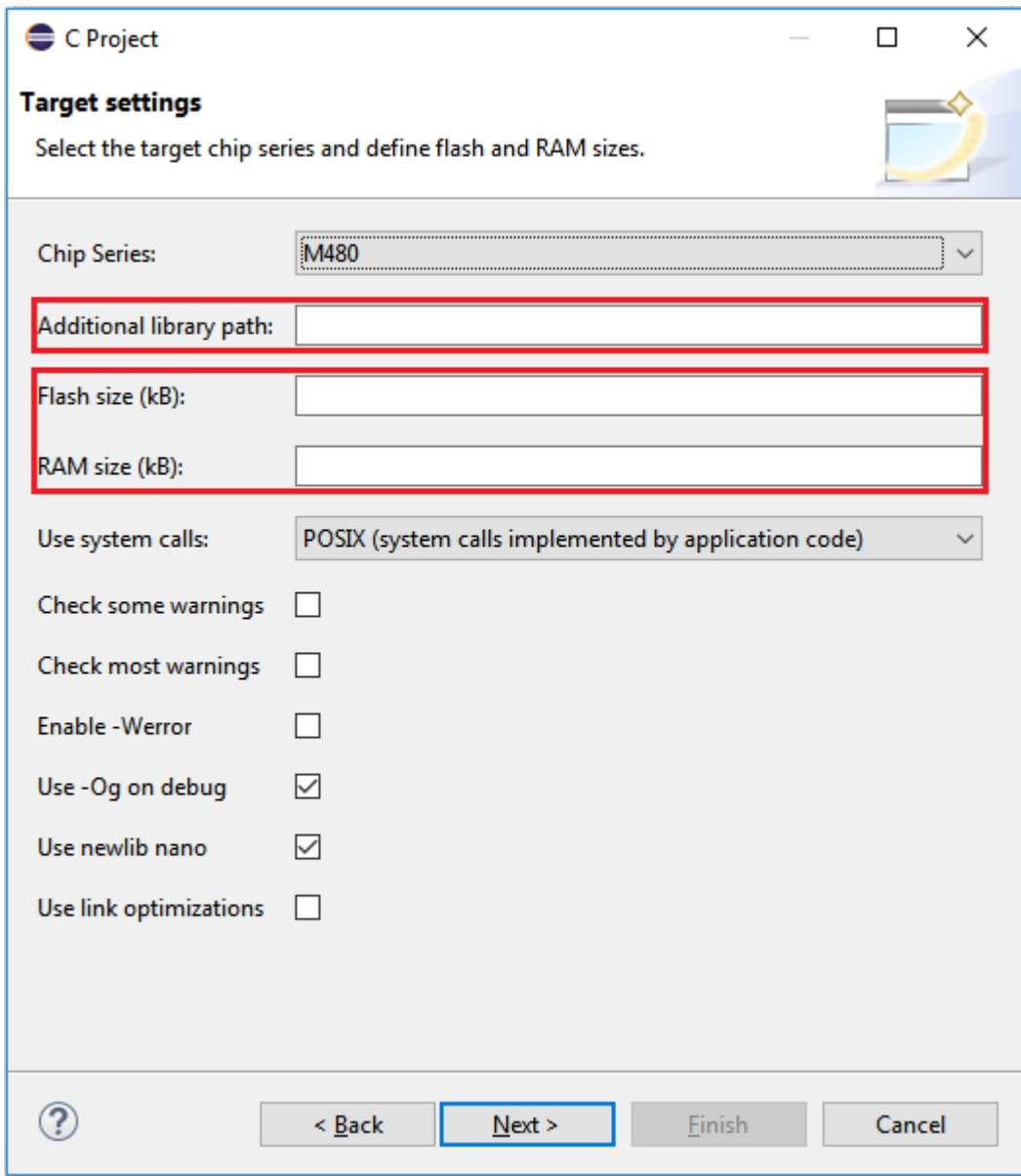


Figure 4-2 New Project Wizard

Based on the actual target chip, we select the corresponding chip series. For some chip series, e.g., M2351\_NonSecure, we need to input the additional library path. If not, the build process may fail. In addition, input the real values to Flash and RAM size. If not, the default values will be used. When all the settings are done, click the **Next >** buttons until clicking the **Finish** button.



**C Project**

**Target settings**

Select the target chip series and define flash and RAM sizes.

Chip Series:

Additional library path:

Flash size (kB):

RAM size (kB):

Use system calls:

Check some warnings ☐

Check most warnings ☐

Enable -Werror ☐

Use -Og on debug ☒

Use newlib nano ☒

Use link optimizations ☐

Figure 4-3 Target Processor Settings

### 4.3 Import Existing Projects

When BSP projects are available, we can import them into the workspace using the following steps:

1. From the main menu bar, select **File > Import**. The Import wizard shows up.
2. Select General > Existing Project into Workspace and click Next.
3. Choose either **Select root directory** or **Select archive file** and click the associated **Browse** to locate the directory or file containing the projects. In the Nuvoton BSP, the Eclipse projects are stored in the GCC folder.
4. Under Projects select the project or projects which you would like to import and click **Finish**.

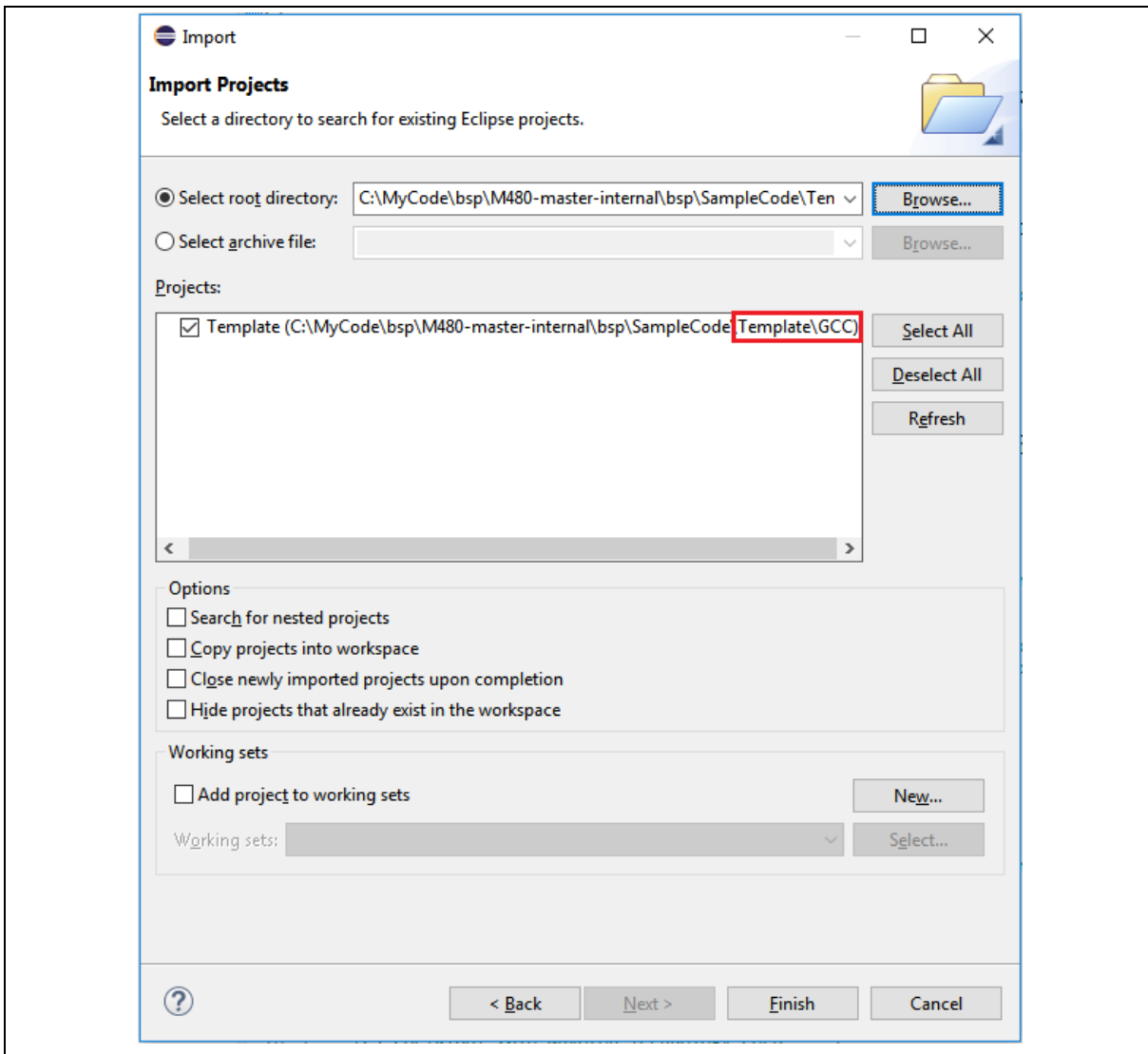


Figure 4-4 Importing Projects



## 4.4 Build Settings

After projects have been created, we still have a chance to alter the build settings by clicking **Project > Properties**. The Properties wizard shows up. Then go to **C/C++ Build > Settings**. From there, we can alter the build settings according to the actual target chip. Then click the **Apply** button to take effect. After applying build settings, we should be able to build projects successfully.

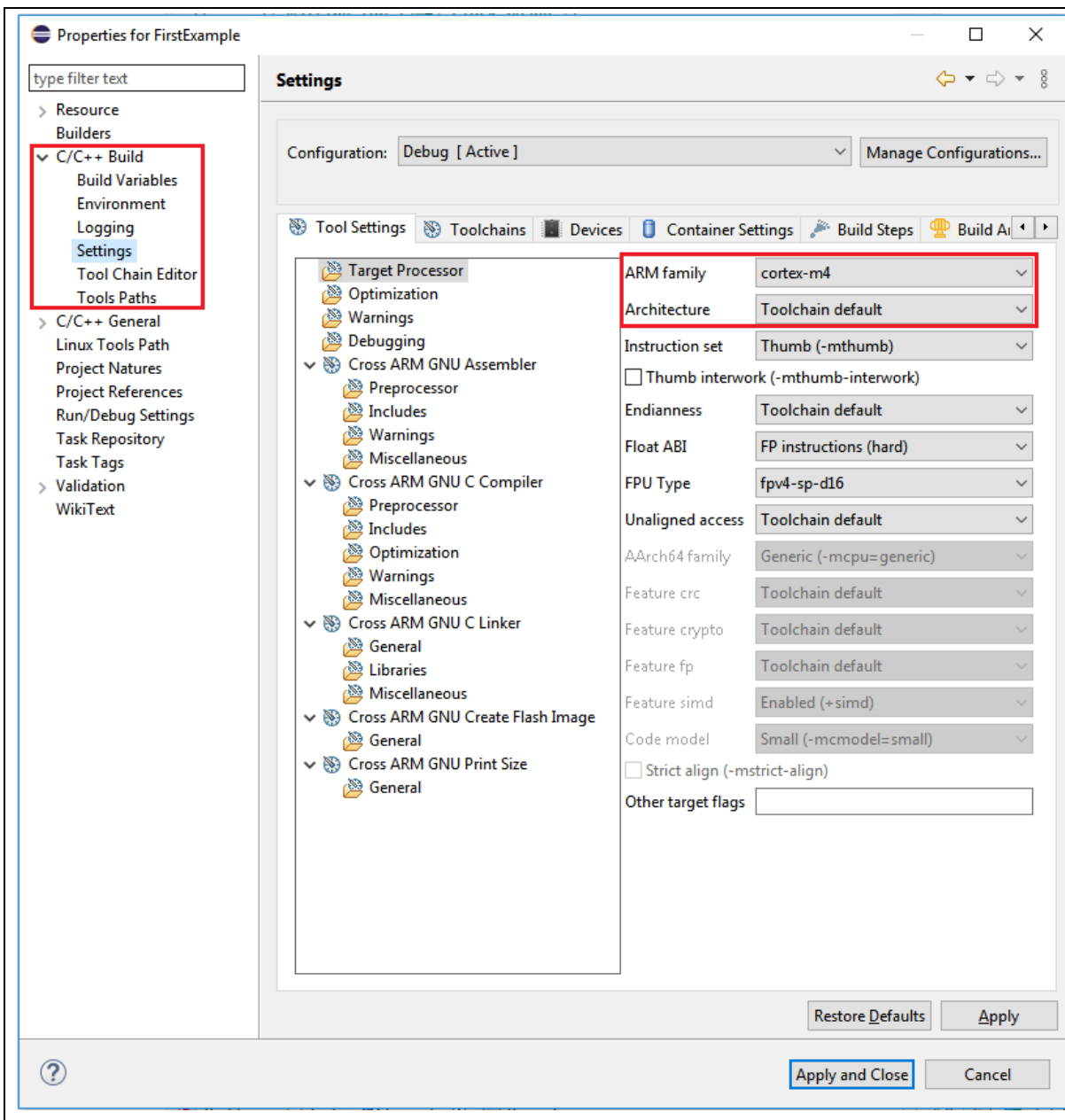


Figure 4-5 Build Settings

## 4.5 Debug Configuration

Before launching an application into the debug mode, we have to prepare a debug configuration, which contains all the necessary information about the debug mode. Click **Run > Debug Configuration...** to open the debug configuration dialog. Double click on the **GDB Nuvoton Nu-Link Debugging** group. The Nuvoton Nu-Link debug configuration appears on the right-hand side. In the Main tab, the name of Project should coincide with the project name. The C/C++ Application should point to the elf application generated by the build process. If the project name or C/C++ Application is incorrect, please select the expected project first in the project view, build the project to generate the executable, and expand the tree to make sure the existence of the generated executable. Then repeat the former operations again.

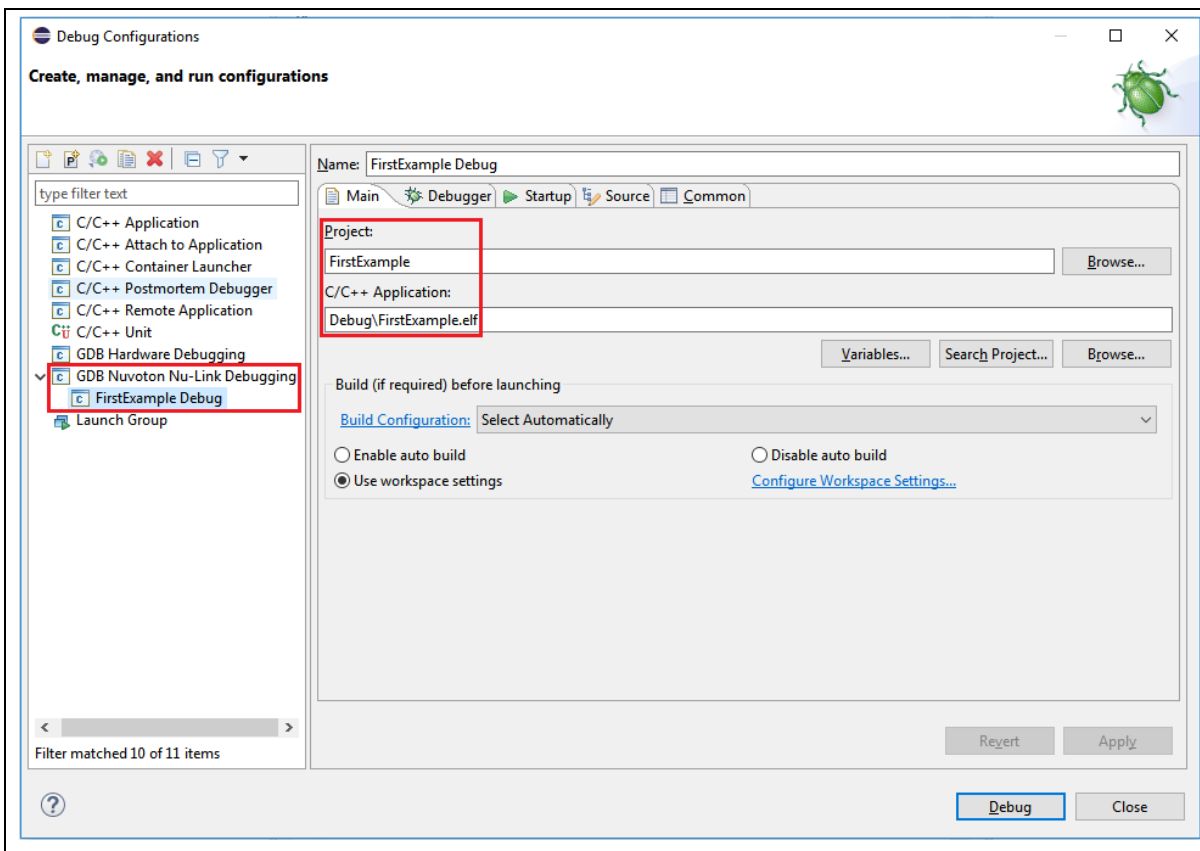


Figure 4-6 Debug Configuration

### 4.5.1 Debugger Tab

The Debugger tab is used to provide the OpenOCD and GDB Client setup. OpenOCD requires correct configuration files to know how to work with adapters and target chips. The configuration files are specified in the **Config options** field. Nuvoton's adapter is Nu-Link, which uses the interface configuration file named **nulink.cfg**. In addition, Nuvoton has three different ARM families, such as M0, M4, and M23. The corresponding target configuration files are **numicroM0.cfg**, **numicroM4.cfg**, and **numicroM23.cfg**. For M23 2<sup>nd</sup> development, the target configuration file would be **numicroM23\_NS.cfg**. For MA35D1 development, see Section 4.10.

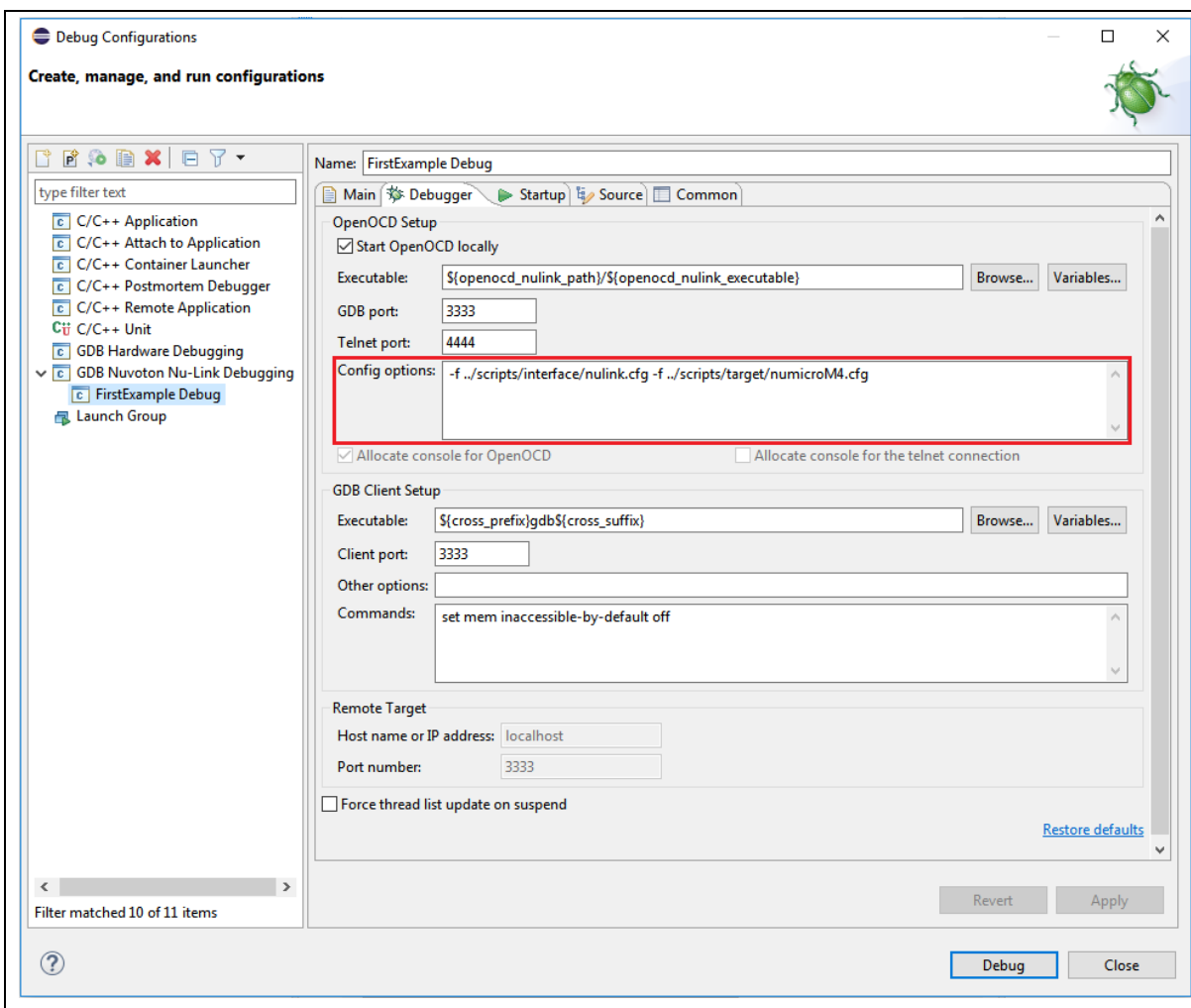


Figure 4-7 Configuring the Debugger Tab

## 4.5.2 Startup Tab

As the first step, we should **choose the right Chip Series** in the Startup tab. When done, the corresponding target configuration file will be automatically written in the **Config options** field of the Debugger tab. To load executable to flash, we need to select the **Load executable to flash** checkbox. To load executable to RAM, we need to select the **Load executable to SRAM** checkbox. When all the settings are done, click the **Apply** button to take effect. To launch the application into the debug mode, click the **Debug** button.

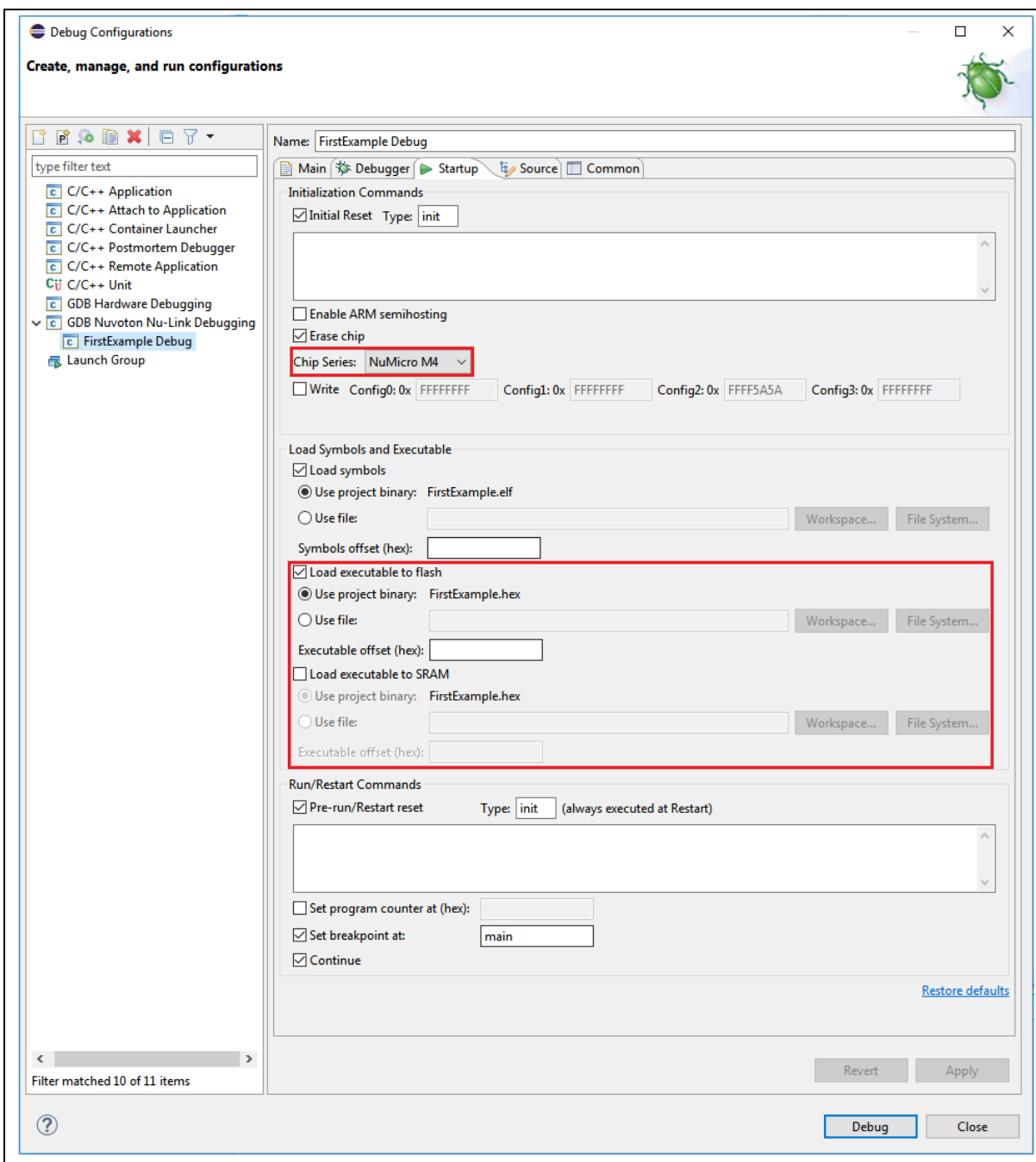


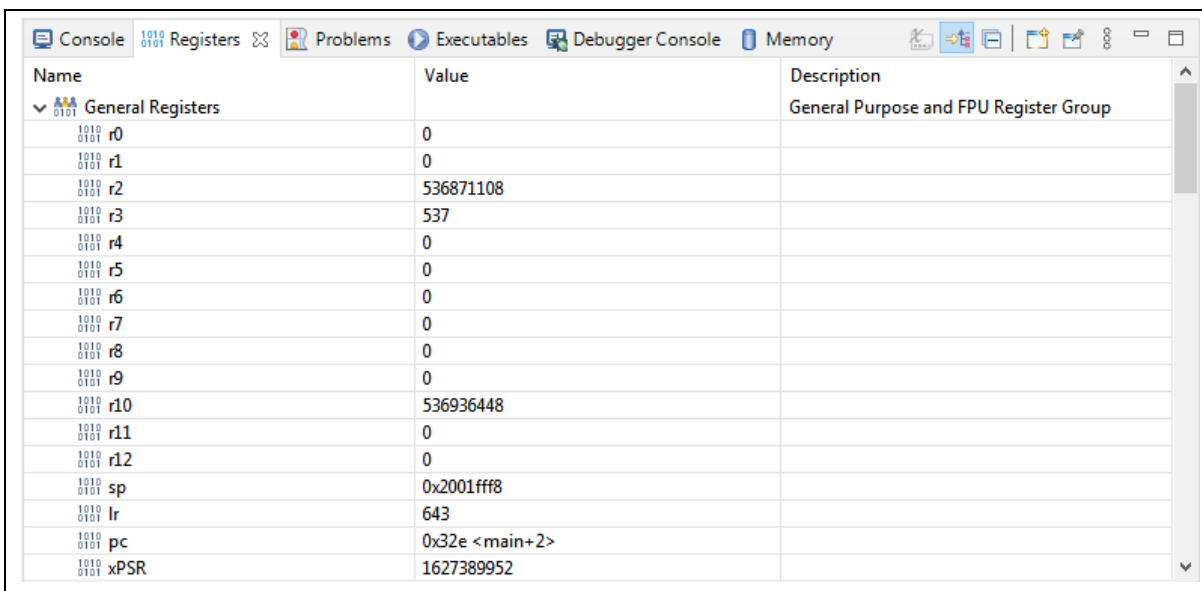
Figure 4-8 Configuring the Startup Tab

## 4.6 Debug Views

Eclipse provides many debug views. Each of them contains specific information for debugging.

### 4.6.1 Registers View

When entering the debug mode, we can open the **Registers view** in the bottom of Debug perspective. The Registers view lists information about the registers in a selected stack frame.



Name	Value	Description
General Registers		General Purpose and FPU Register Group
r0	0	
r1	0	
r2	536871108	
r3	537	
r4	0	
r5	0	
r6	0	
r7	0	
r8	0	
r9	0	
r10	536936448	
r11	0	
r12	0	
sp	0x2001fff8	
lr	643	
pc	0x32e <main+2>	
xPSR	1627389952	

Figure 4-9 Registers View

## 4.6.2 Memory View

The **Memory view** of the Debug perspective is used to monitor and modify the process memory. The process memory is presented as a list of so called **memory monitors**. Each monitor represents a section of memory specified by its location called **base address**. To open it, click the Memory tab on the lower side of Debug perspective.

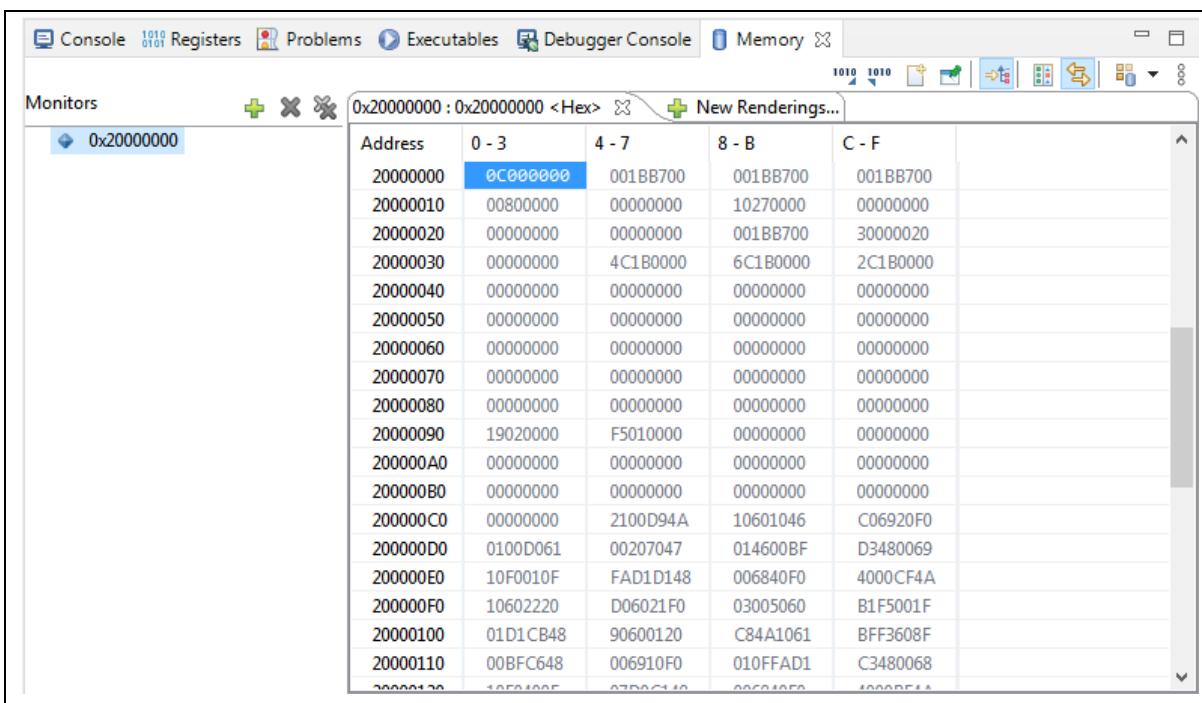


Figure 4-10 Memory View

### 4.6.3 Disassembly View

The Disassembly view shows the loaded program as assembler instructions mixed with source code for comparison. We can do the following tasks in the Disassembly view:

1. Setting breakpoints at the start of any assembler instruction.
2. Enabling and disabling breakpoints.
3. Stepping through the disassembly instructions of the program.
4. Jumping to specific instructions in the program.

To open it, we need to click the **Instruction Stepping Mode** button on the upper toolbar, as follows:

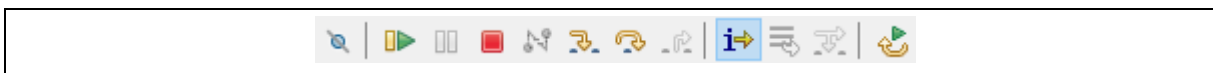


Figure 4-11 Clicking the Instruction Stepping Mode Button

Then the Disassembly view will appear on the right-hand side.

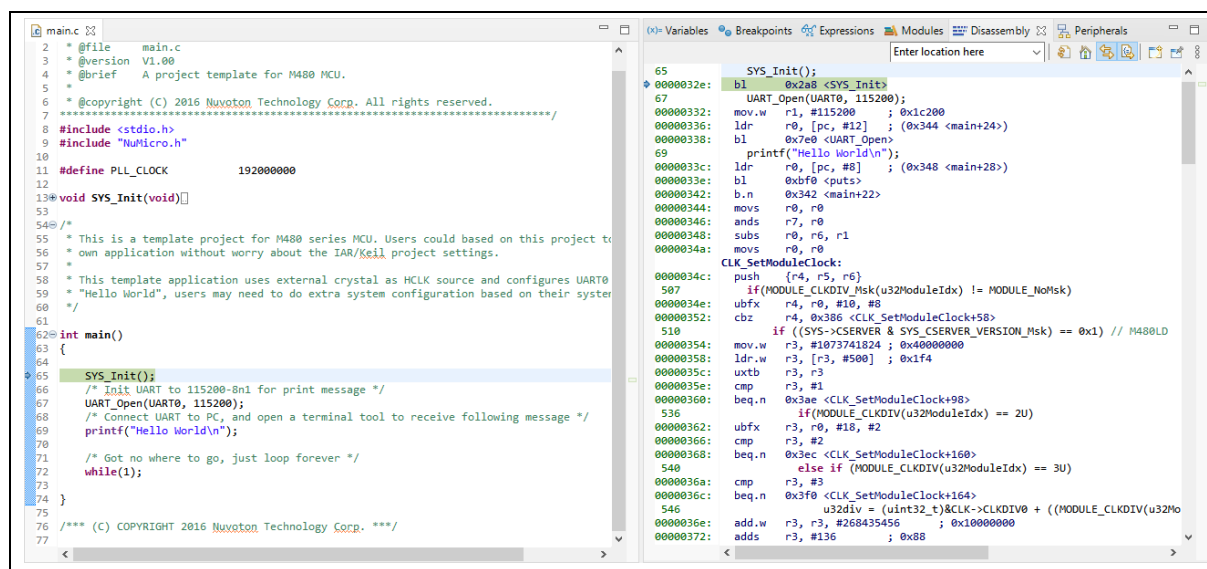


Figure 4-12 Disassembly View

#### 4.6.4 Peripheral Registers View

To display the Peripheral Registers view, we need to utilize **Packs** mechanism. Packs can help the user download special function register (SFR) files from the Keil repository. Firstly, we open the Packs perspective by choosing it in the **Open Perspective** dialog.

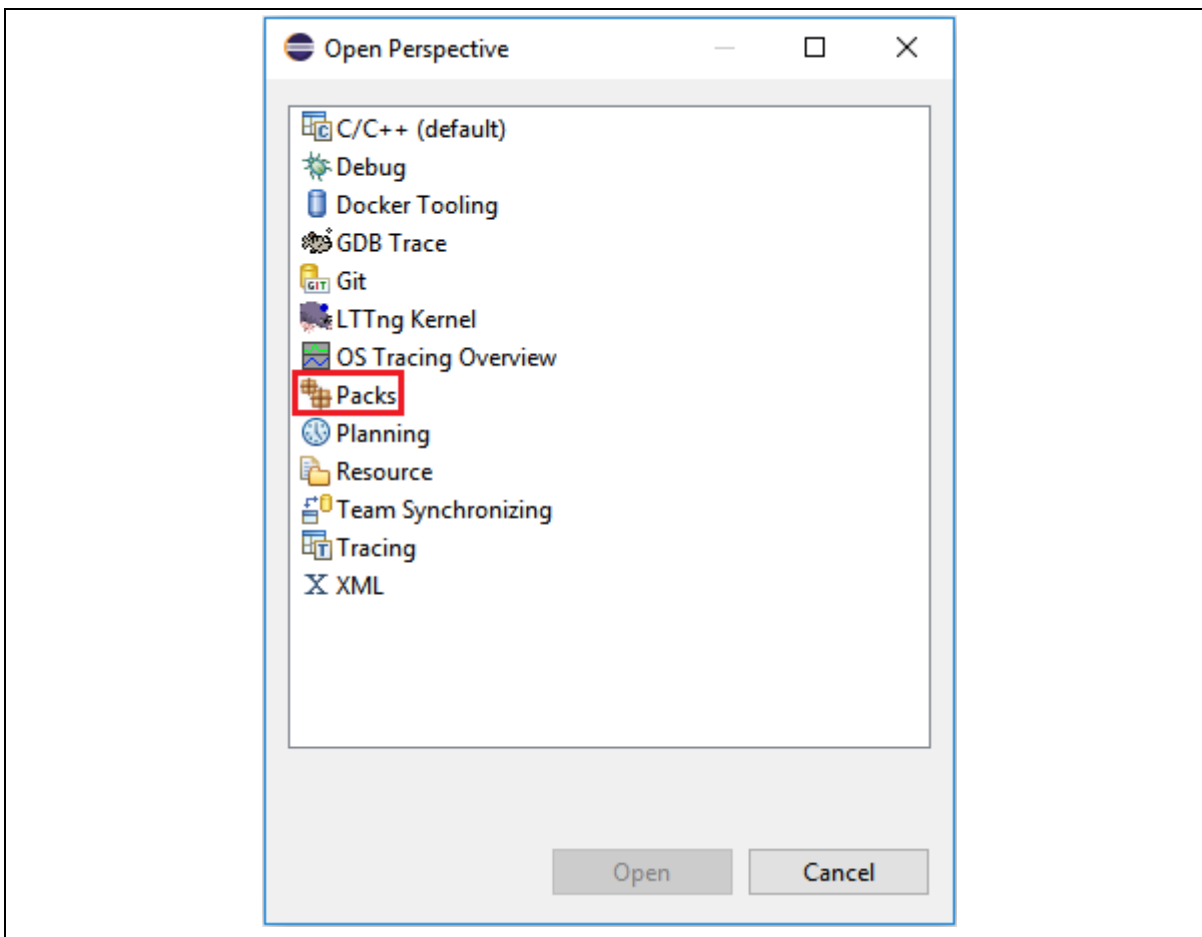


Figure 4-13 Opening the Packs Perspective



For the first time we see the Packs perspective, its content is provided by the NuEclipse installer. If the default content is missing, please switch to a new workspace and try again. To get the latest version, click the **Update the packages definitions from all repositories** button at the upper-right corner. After clicking, Eclipse begins downloading all the SFR files from an online repository.

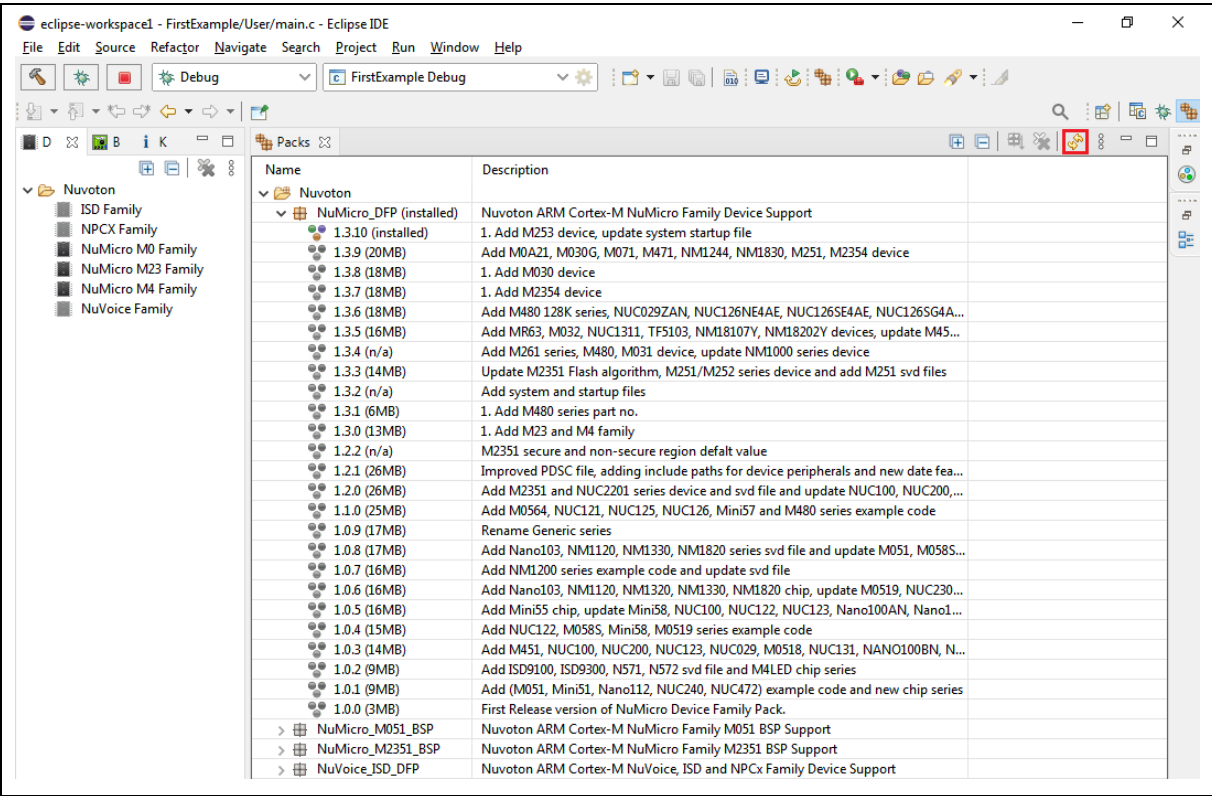


Figure 4-14 How to Download Packages

The locations of repositories are specified in the **Window > Preferences > C/C++ > Packages > Repositories**. The default is from Keil's CMSIS Pack.

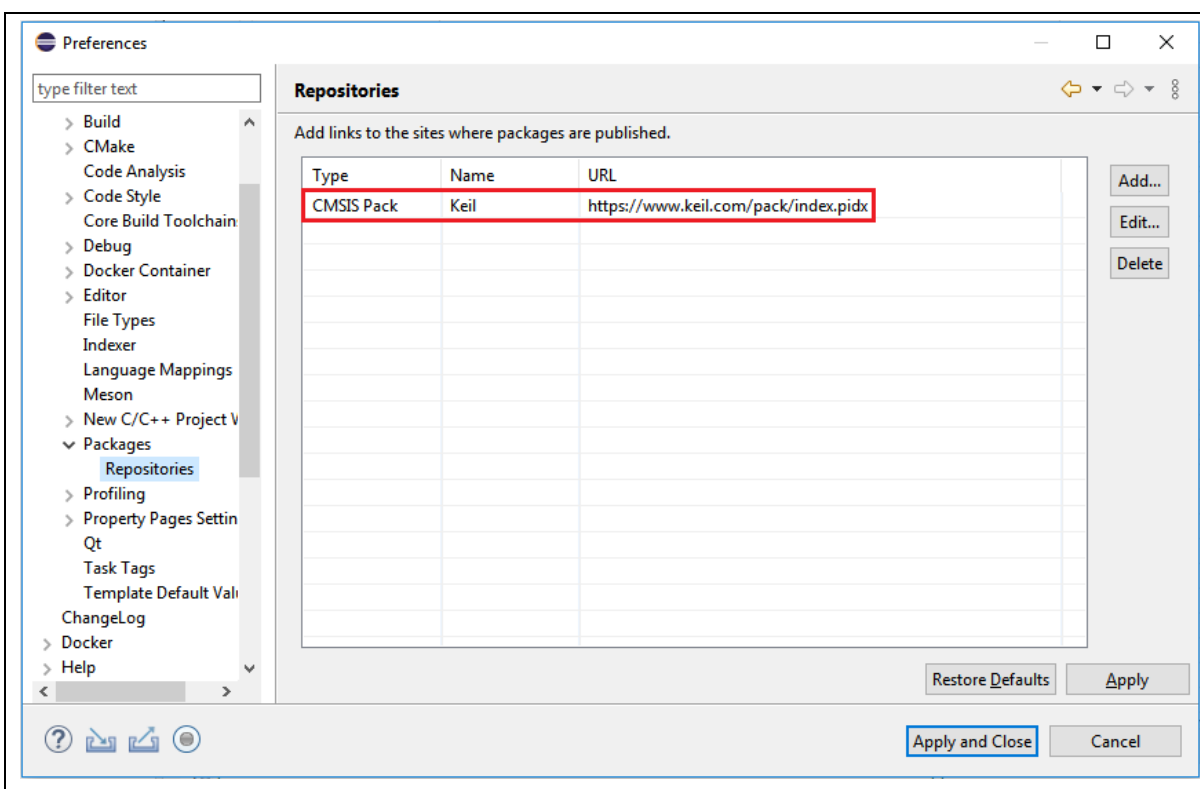


Figure 4-15 Locations of Repositories

When the download is completed, we can find the Nuvoton SFR files and install them on Eclipse if needed.

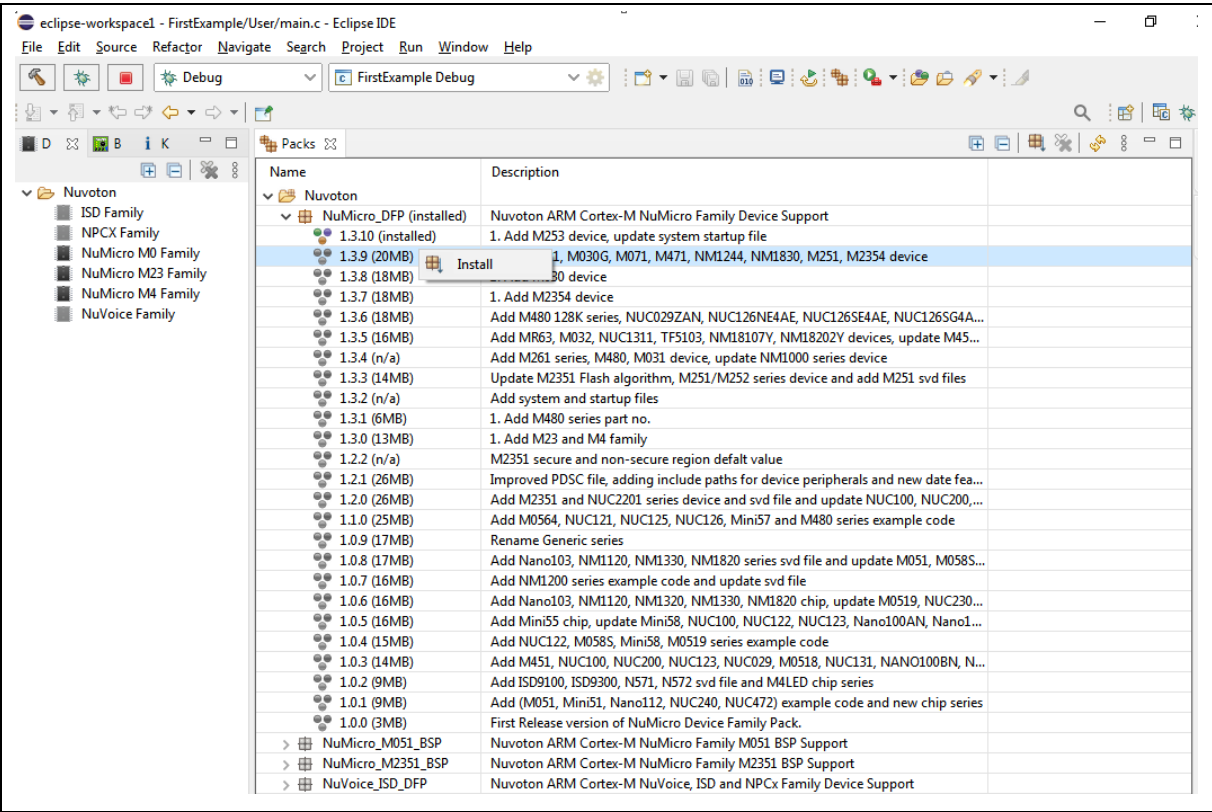


Figure 4-16 Installing SFR Files

To use the specific SFR file, open the project's properties dialog and go to the **C/C++ Build > Settings**. From there, we should choose the specific device matching the real one. In this case, it is M487JIDAE. Click the **Apply** button to take effect.

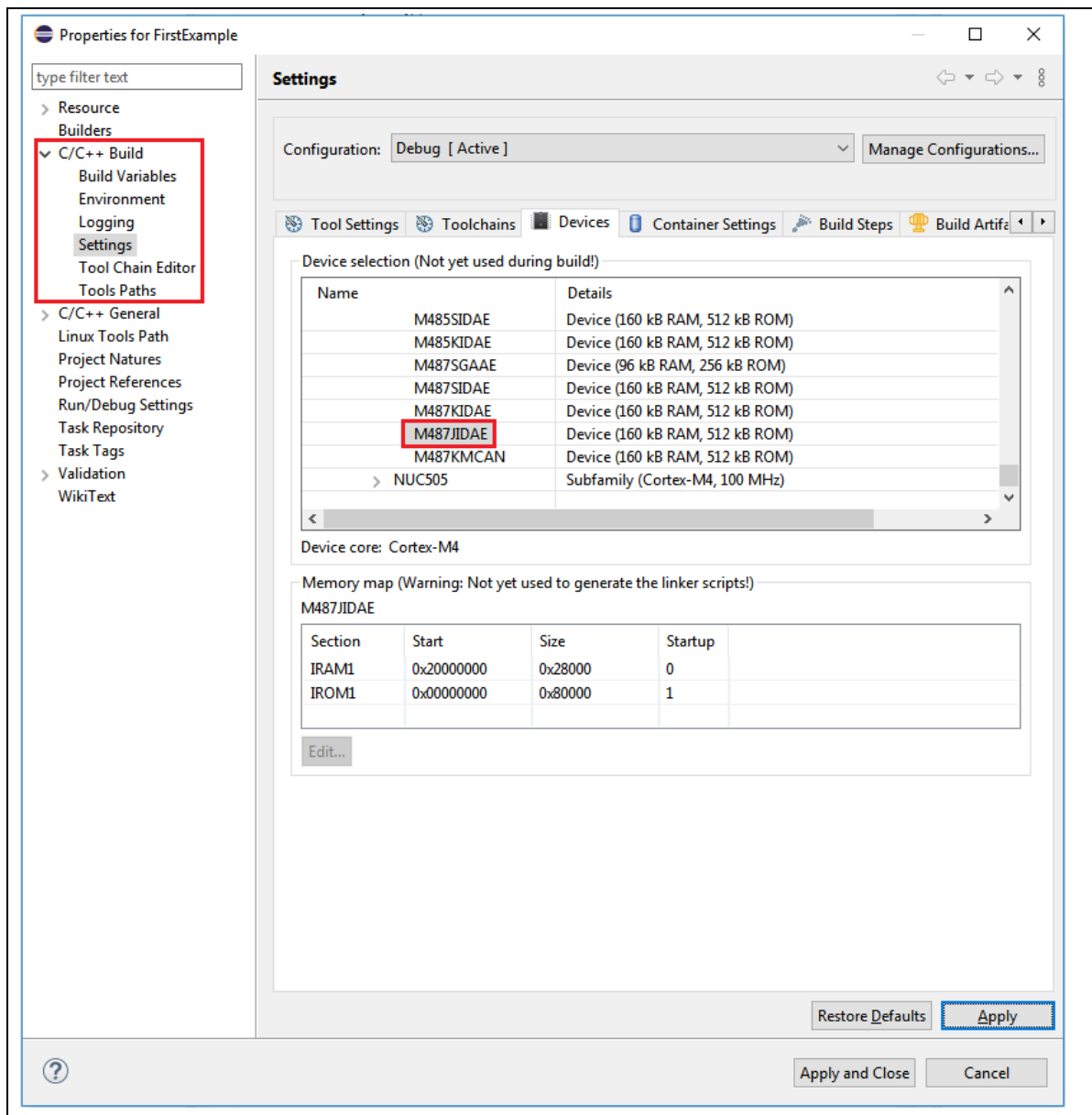


Figure 4-17 Device Selection

As a result, we can monitor the peripheral registers when debugging.

The screenshot displays the NuEclipse IDE interface. The main editor shows the `main.c` file with the following code:

```

1 //
2 * @file    main.c
3 * @version V1.00
4 * @brief    A project template for M480 MCU.
5 *
6 * @copyright (C) 2016 Nuvoton Technology Corp. All rights reserved.
7
8 #include <stdio.h>
9 #include "NuMicro.h"
10
11 #define PLL_CLOCK      192000000
12
13 void SYS_Init(void)
14 {
15     /*----- System Clock -----*/
16     /* Init System Clock */
17     /*----- System Clock -----*/
18     /* Unlock protected registers */
19     SYS_UnlockReg();
20
21     /* Set XT1_OUT(PF.2) and XT1_IN(PF.3) to input mode */
22     PF->MODE &= ~(GPIO_MODE_MODE2_Msk | GPIO_MODE_MODE3_Msk);
23
24     /* Enable External XTAL (4~24 MHz) */
25     CLK_EnableXtalRC(CLK_PWRCTL_HXTEN_Msk);
26
27     /* Waiting for 12MHz clock ready */
28     CLK_WaitClockReady(CLK_STATUS_HXTSTB_Msk);
29
30     /* Set core clock as PLL_CLOCK from PLL */
31     CLK_SetCoreClock(PLL_CLOCK);
32     /* Set PCLK0/PCLK1 to HCLK/2 */
33     CLK->PCLKDIV = (CLK_PCLKDIV_APB0DIV_DIV2 | CLK_PCLKDIV_APB1DIV_DIV2);
34
35     /* Enable UART clock */
36     CLK_EnableModuleClock(UART0_MODULE);
37
38     /* Select UART clock source from HXT */
39     CLK_SetModuleClock(UART0_MODULE, CLK_CLKSEL1_UART0SEL_HXT, CLK_CLKSEL0_UART0SEL_HXT);
40
41     /* Update System Core Clock */
42     /* User can use SystemCoreClockUpdate() to calculate SystemCoreClock */
43     SystemCoreClockUpdate();
44
45     /* Set GPB multi-function pins for UART0 RXD and TXD */
46     SYS->GPB_MFPH = ~(SYS_GPB_MFPH_PB12MFP_Msk | SYS_GPB_MFPH_PB13MFP_Msk);
47     SYS->GPB_MFPH |= (SYS_GPB_MFPH_PB12MFP_UART0_RXD | SYS_GPB_MFPH_PB13MFP_UART0_TXD);
48     /* Lock protected registers */
49     SYS_LockReg();
50 }
51
52 /*
53 * This is a template project for M480 series MCU. Users could based on
54 * own application without worry about the IAR/Keil project settings.
55 * This template application uses external crystal as HCLK source and
56 * "Hello World", users may need to do extra system configuration base
57 */
58
59
60
61

```

The Peripheral Registers View is open on the right side, showing a list of peripherals and their addresses. The GPIO peripheral is selected, and its registers are displayed in the bottom panel.

Peripheral	Address	Description
ECAP1	0x400B5000	ECAP Register Map
EMAC	0x40008000	EMAC Register Map
EPWM0	0x40058000	EPWM Register Map
EPWM1	0x40059000	EPWM Register Map
FMC	0x4000C000	FMC Register Map
GPIO	0x40004000	GPIO Register Map
HSOTG	0x4004F000	HSOTG Register Map
HSUSBD	0x40019000	USB Register Map
HSUSBH	0x4001A000	USB Register Map
I2C0	0x40080000	I2C Register Map
I2C1	0x40081000	I2C Register Map
I2C2	0x40082000	I2C Register Map
I2S	0x40048000	I2S Register Map
NMI	0x40003000	NMI Register Map
NVIC	0xE000E100	NVIC Register Map
OPA	0x40046000	OPA Register Map
OTG	0x4004D000	OTG Register Map
PDMA	0x40008000	PDMA Register Map
QEI0	0x40080000	QEI Register Map
QEI1	0x40081000	QEI Register Map
QSPI0	0x40060000	QSPI Register Map
QSPI1	0x40069000	QSPI Register Map
RTC	0x40041000	RTC Register Map
SC0	0x40090000	SC Register Map
SC1	0x40091000	SC Register Map
SC2	0x40092000	SC Register Map
SCS	0xE000E000	SYST_CSR Register Map
SDH0	0x4000D000	SDH Register Map
SDH1	0x4000E000	SDH Register Map
SPI0	0x40061000	SPI Register Map
SPI1	0x40062000	SPI Register Map
SPI2	0x40063000	SPI Register Map
SPIM	0x40007000	SPIM Register Map
SYS	0x40000000	SYS Register Map
TMR01	0x40050000	TIMER Register Map
TMR02	0x40051000	TIMER Register Map
TRNG	0x40089000	TRNG Register Map
UART0	0x40070000	UART Register Map
UART1	0x40071000	UART Register Map
UART2	0x40072000	UART Register Map
UART3	0x40073000	UART Register Map

The bottom panel shows the GPIO peripheral registers. The selected register is `GPIO: 0x40004000`. The registers are listed as follows:

Register	Address	Value
PA_DINOFF	0x40004004	0x00000000
PA_DOUT	0x40004008	0x0000FFFF
DOUT0	[0]	0x1: 1
DOUT1	[1]	0x1: 1
DOUT10	[10]	0x1: 1
DOUT11	[11]	0x1: 1
DOUT12	[12]	0x1: 1
DOUT13	[13]	0x1: 1
DOUT14	[14]	0x1: 1
DOUT15	[15]	0x1: 1
DOUT2	[2]	0x1: 1
DOUT3	[3]	0x1: 1
DOUT4	[4]	0x1: 1
DOUT5	[5]	0x1: 1
DOUT6	[6]	0x1: 1
DOUT7	[7]	0x1: 1
DOUT8	[8]	0x1: 1
DOUT9	[9]	0x1: 1

Figure 4-18 Peripheral Registers View

## 4.7 Watchpoints

To add watchpoints on Eclipse, we need to do the following steps:

1. Selecting a **global variable**, i.e. `g_seconds`, in the Outline view.
2. Right-clicking on the global variable and choosing **Toggle Watchpoint**.

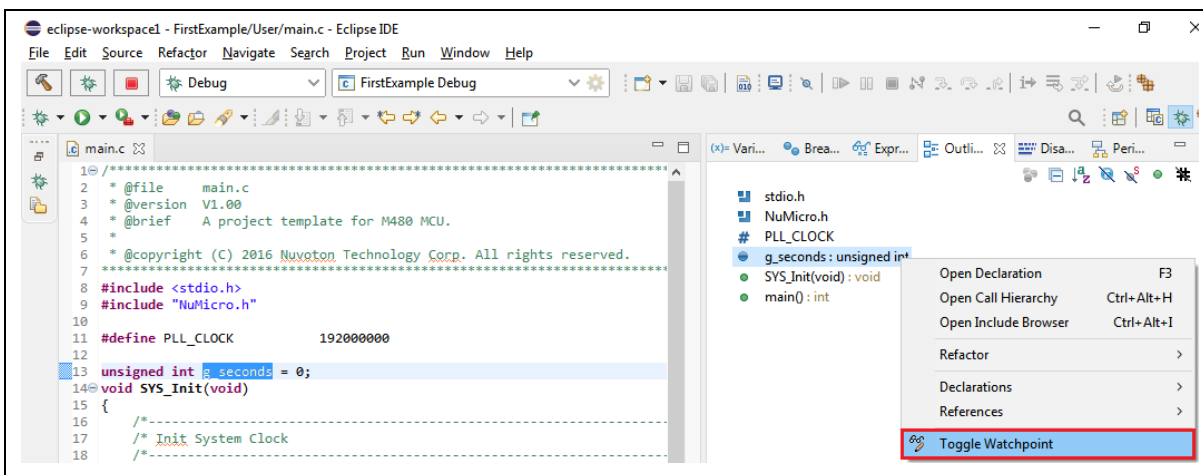


Figure 4-19 Toggle Watchpoint

3. Configuring the settings for watchpoints. To stop execution when the watch expression is read, select the **Read** checkbox. To stop execution when the watch expression is written to, select the **Write** checkbox.

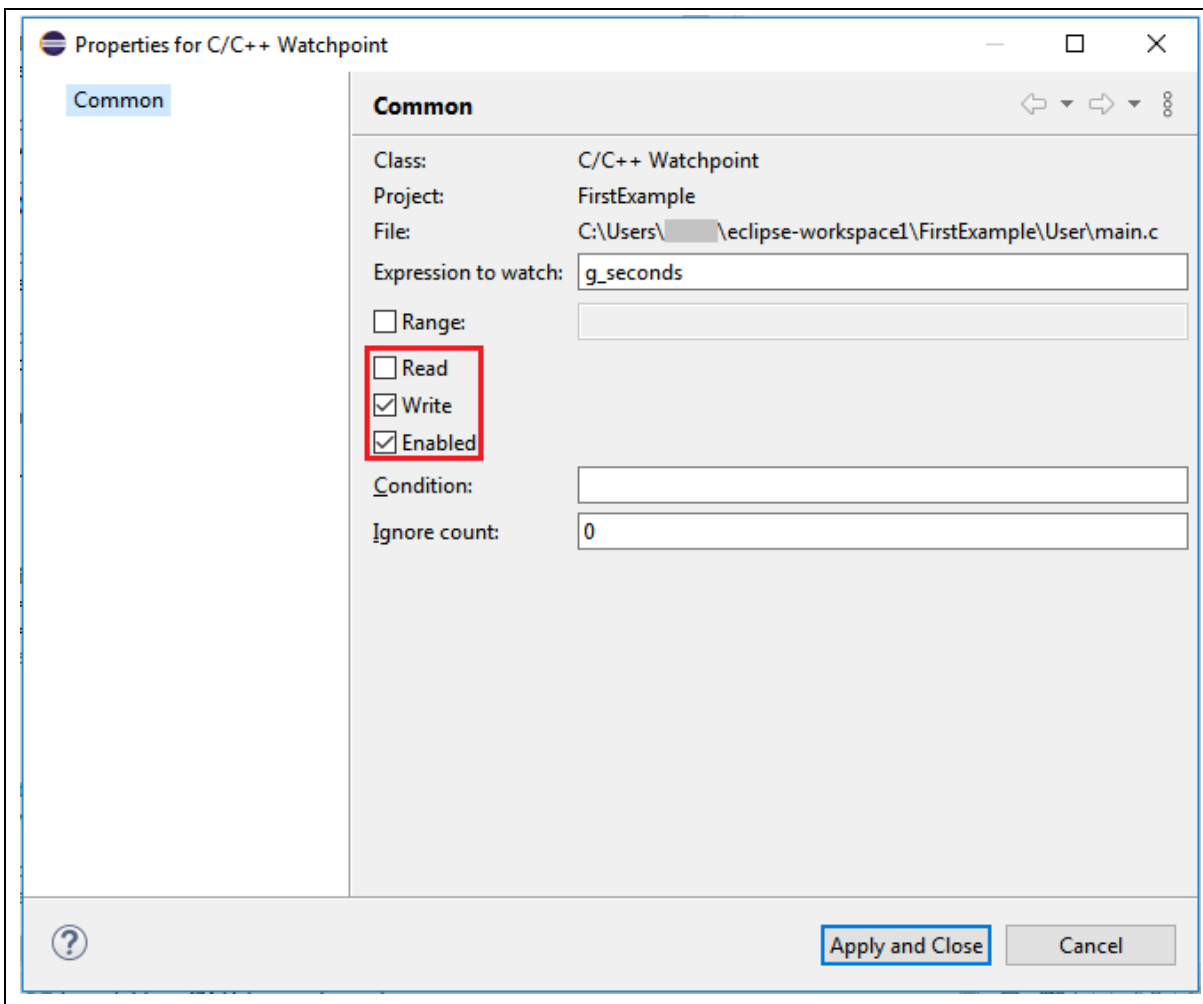


Figure 4-20 Properties for C/C++ Watchpoint

When the watchpoint is added, it appears in the **Breakpoints view**.

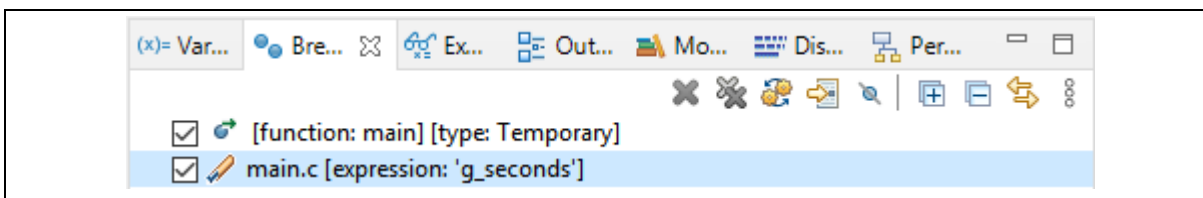


Figure 4-21 Added Watchpoint in the Breakpoints View

## 4.8 Debug in RAM

To debug in RAM, there are several steps to follow:

1. Modifying the ld script.
2. Assigning PC to the specific RAM address.
3. Assigning SP to the specific RAM address.
4. Downloading the binary file to RAM.

The ld script is responsible for telling the linker the layout of the compiled executable. For example, the memory layout looks like:

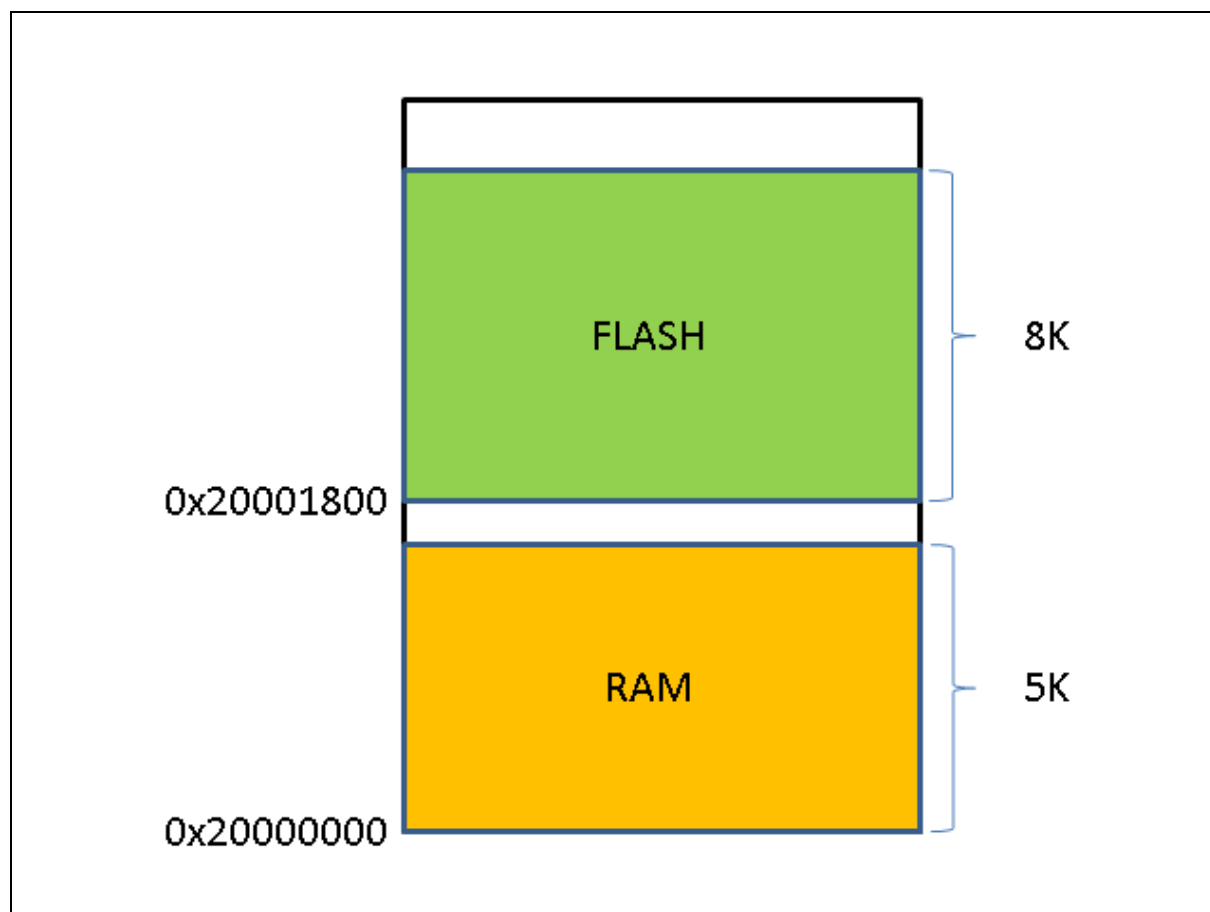


Figure 4-22 Memory Layout



The modified Id script should meet the memory layout design.

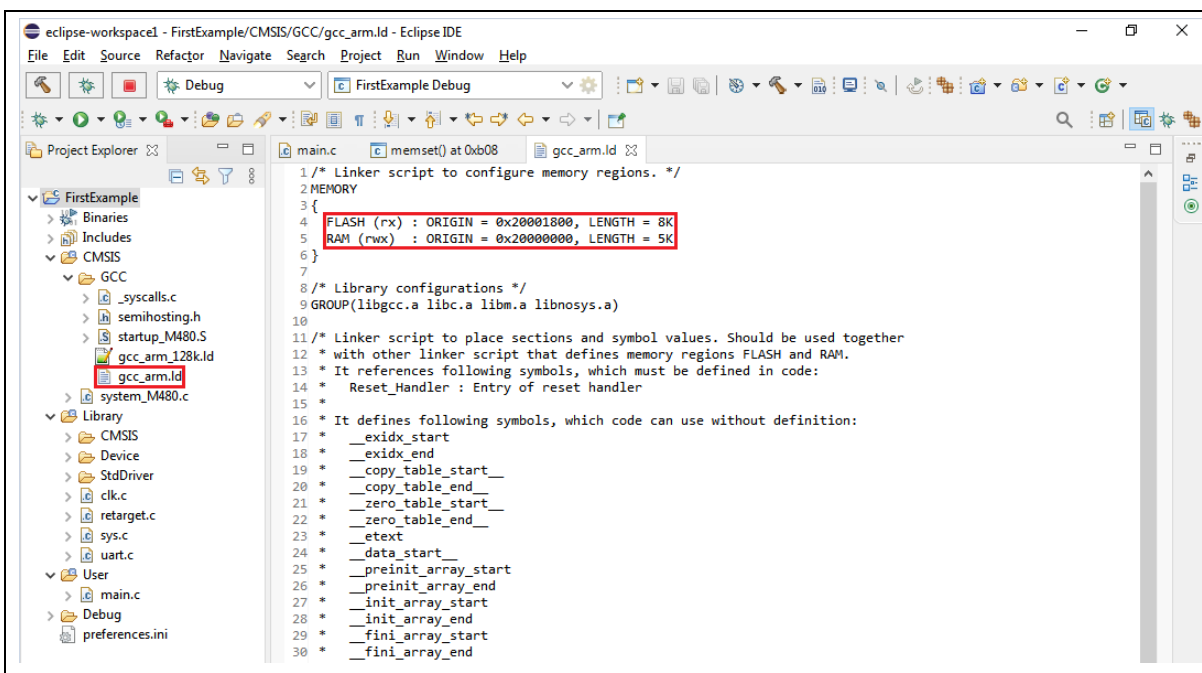


Figure 4-23 Modifying the Id Script

To assign PC and SP to the specific addresses, we need to input them in the debug configuration, as follows. Based on the previous memory layout, the PC and SP addresses should be Reset\_Handler and 0x20001400, respectively. In addition, set Vector Table Offset Register (0xE000ED08) should be 0x20000000 and unselect the **Pre-run/Restart reset** Button. To download the binary to RAM, we select the **Load executable to SRAM** button and unselect the **Load executable to flash** button. Click the **Debug** button to start a debug session.

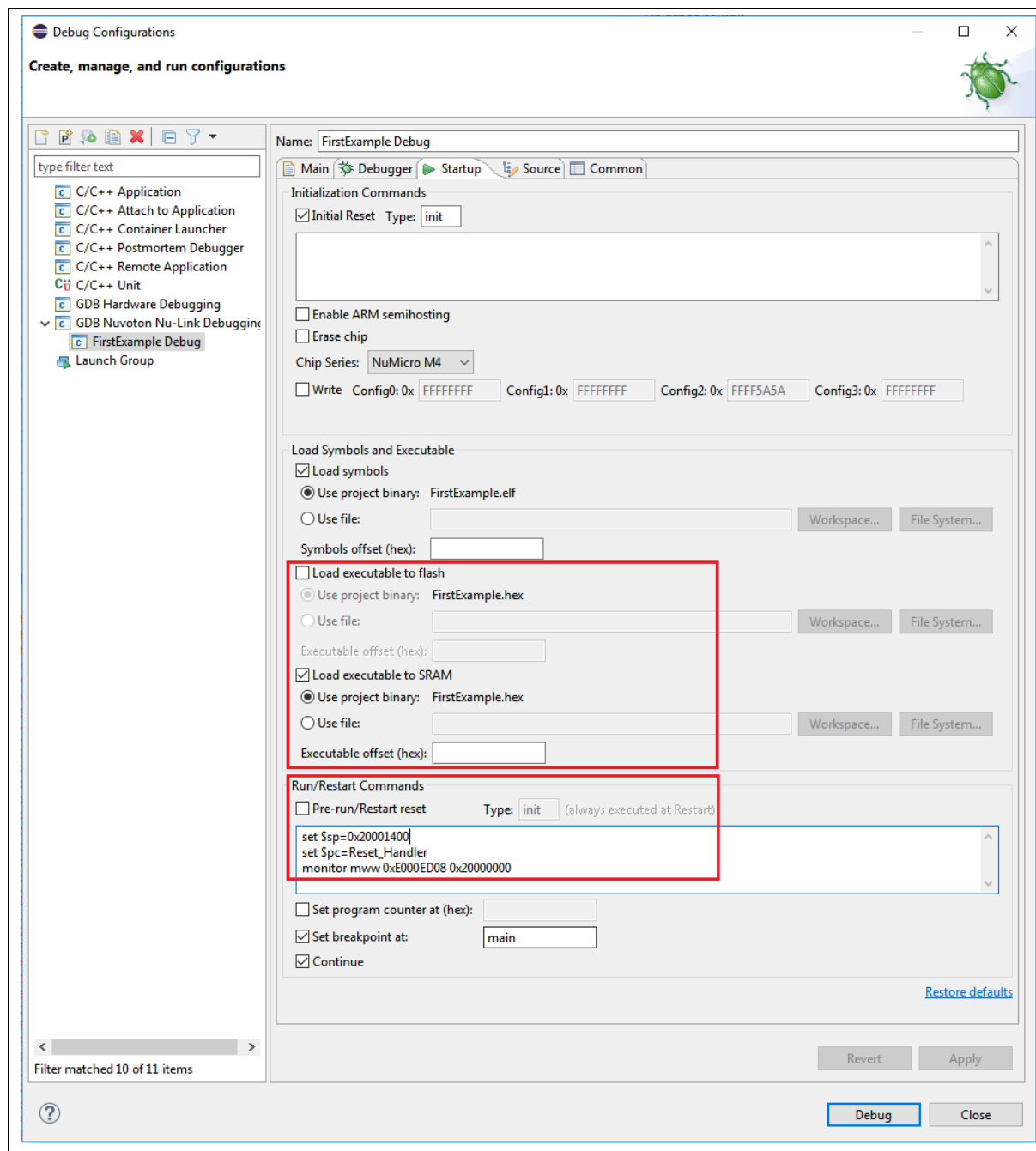


Figure 4-24 Debug Configuration Settings

When the program stops in the main function, we open the Memory view. From there, we can verify that the binary file is successfully downloaded into RAM. The first word denotes the SP address. The following words denote the addresses of handlers.

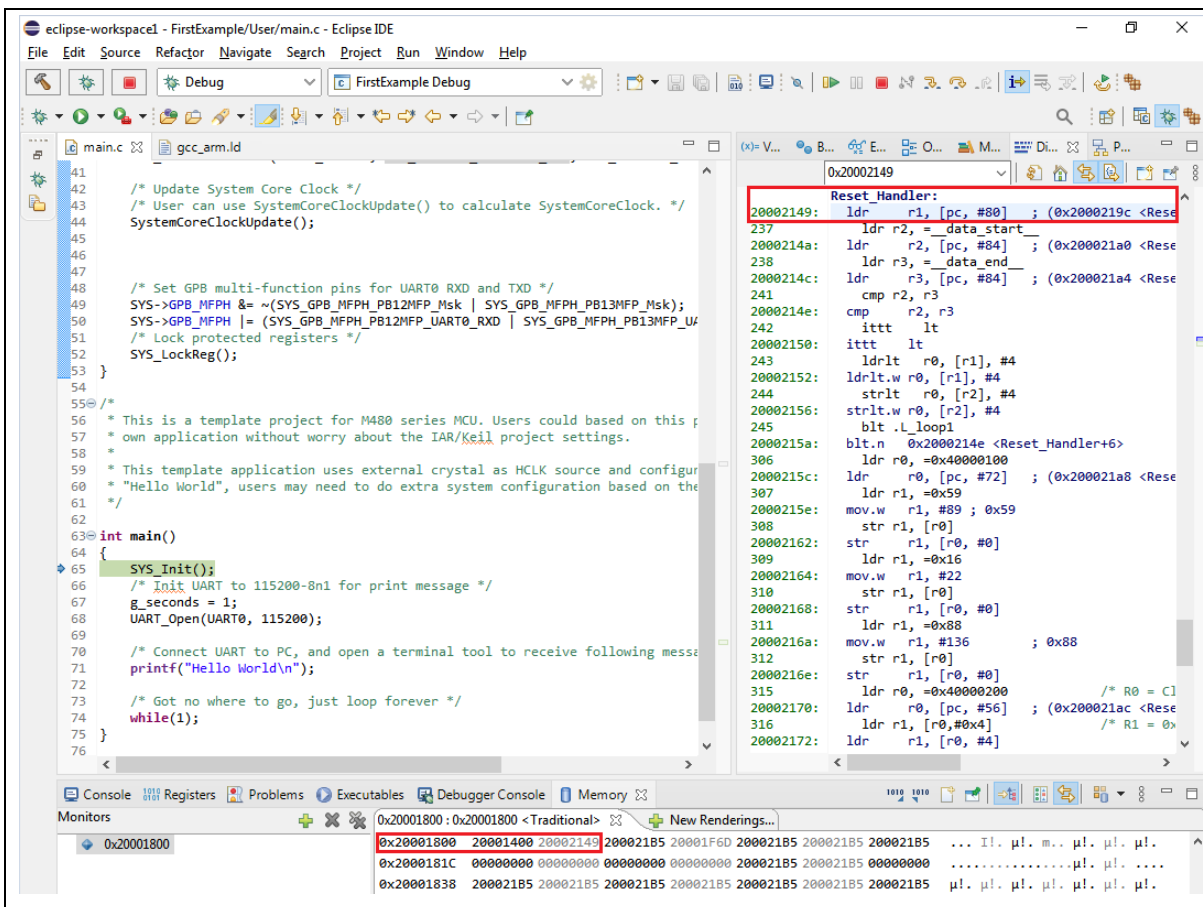


Figure 4-25 Debugging in RAM

## 4.9 Debug Executable Files Only

If the user has an executable built with **debug symbols** but may not have the relevant environment used to build the executable, he still is able to debug the executable by the following steps:

1. Import an executable for debugging (referring to Figure 4-26).
2. Click Browse following Select executable, then select an executable (referring to Figure 4-27).
3. Choose GDB Nuvoton Nu-Link Debugging as a Launch Configuration (referring to Figure 4-28).
4. Locate the GDB executable in the debug configuration (referring to Figure 4-29).
5. Choose the ELF file to download in the debug configuration (referring to Figure 4-30).
6. Add source lookup path relative to source folders (referring to Figure 4-31).
7. Press on the Debug button.

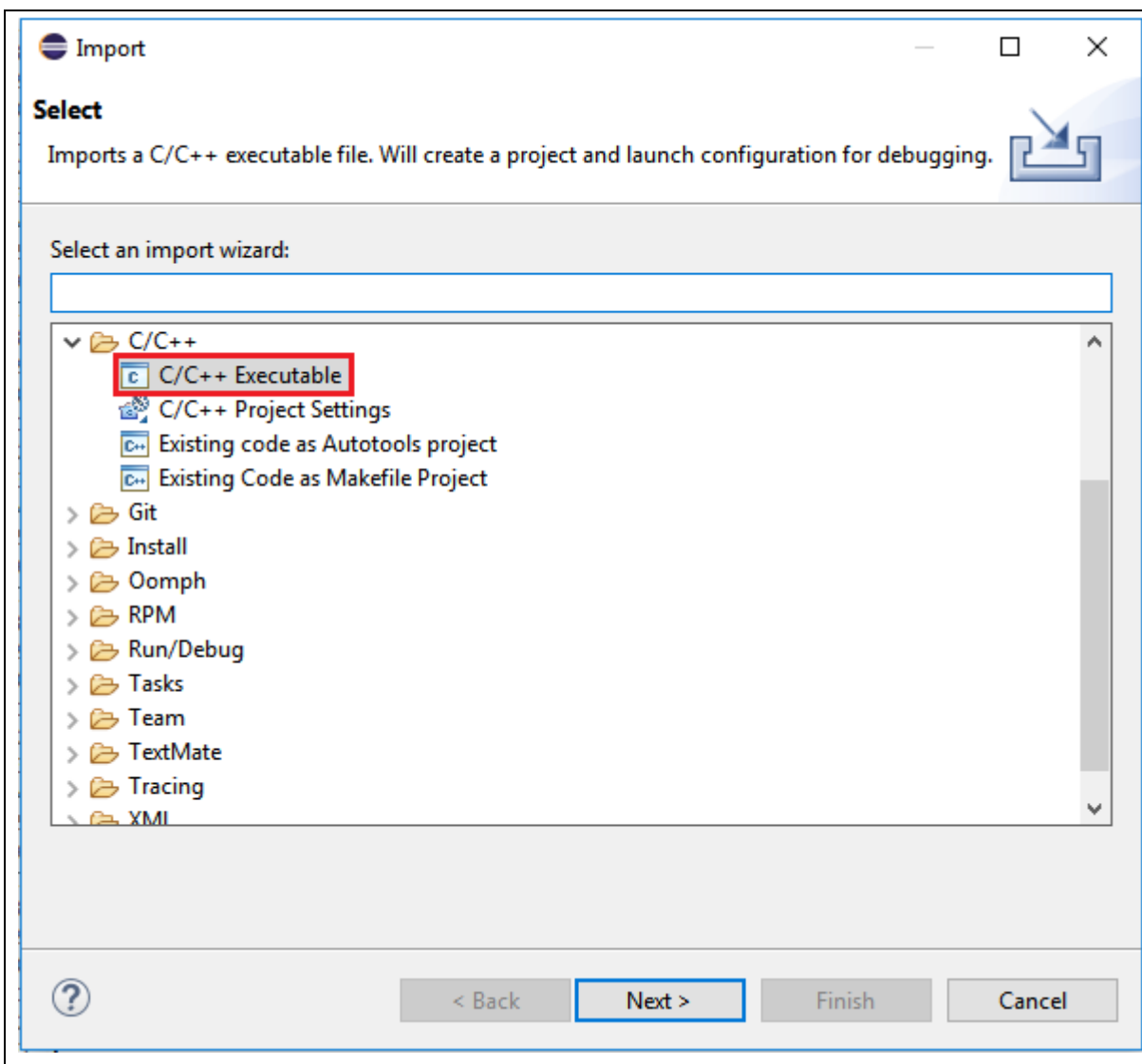


Figure 4-26 Importing Executable for Debugging

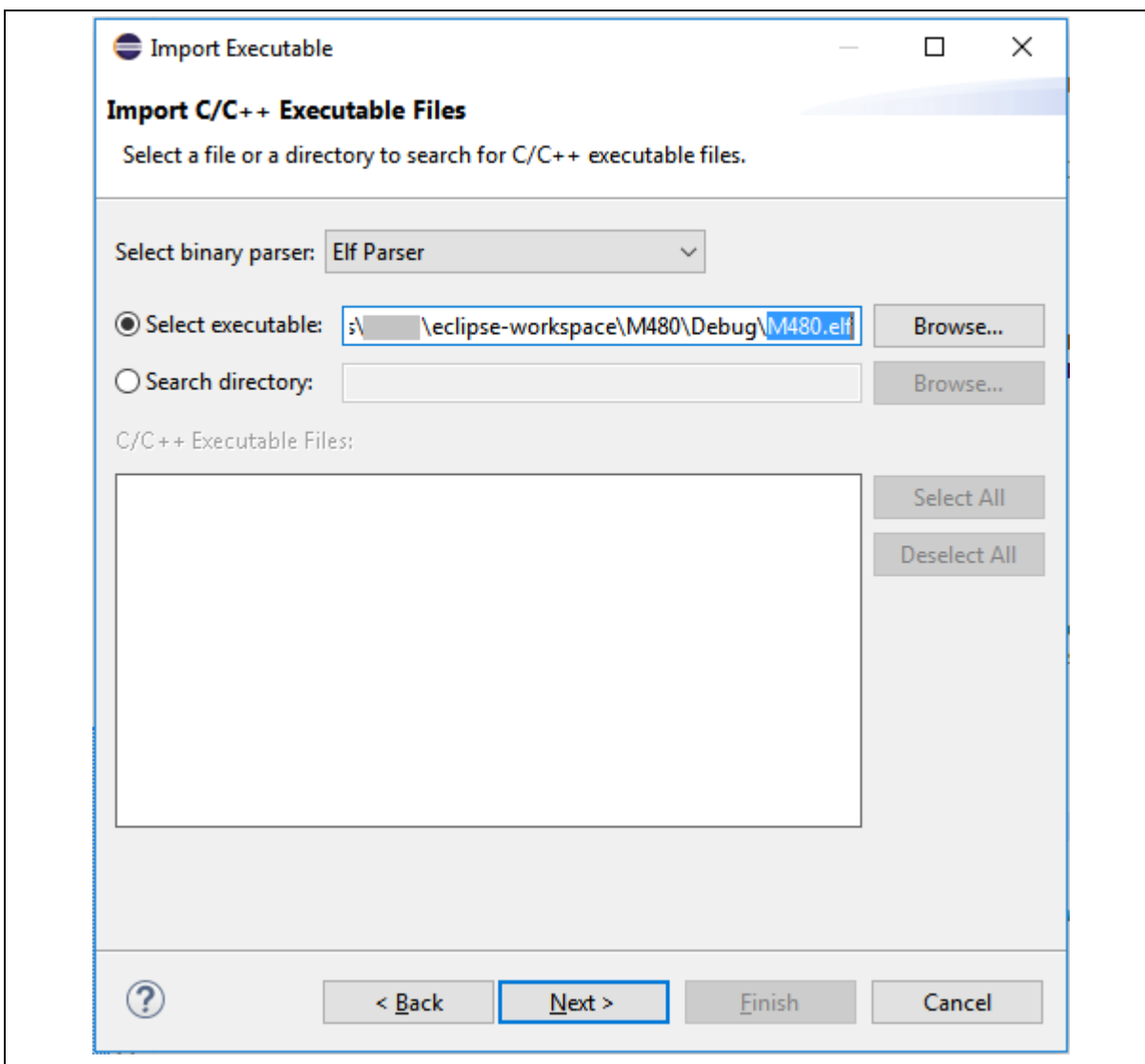


Figure 4-27 Selecting Executable

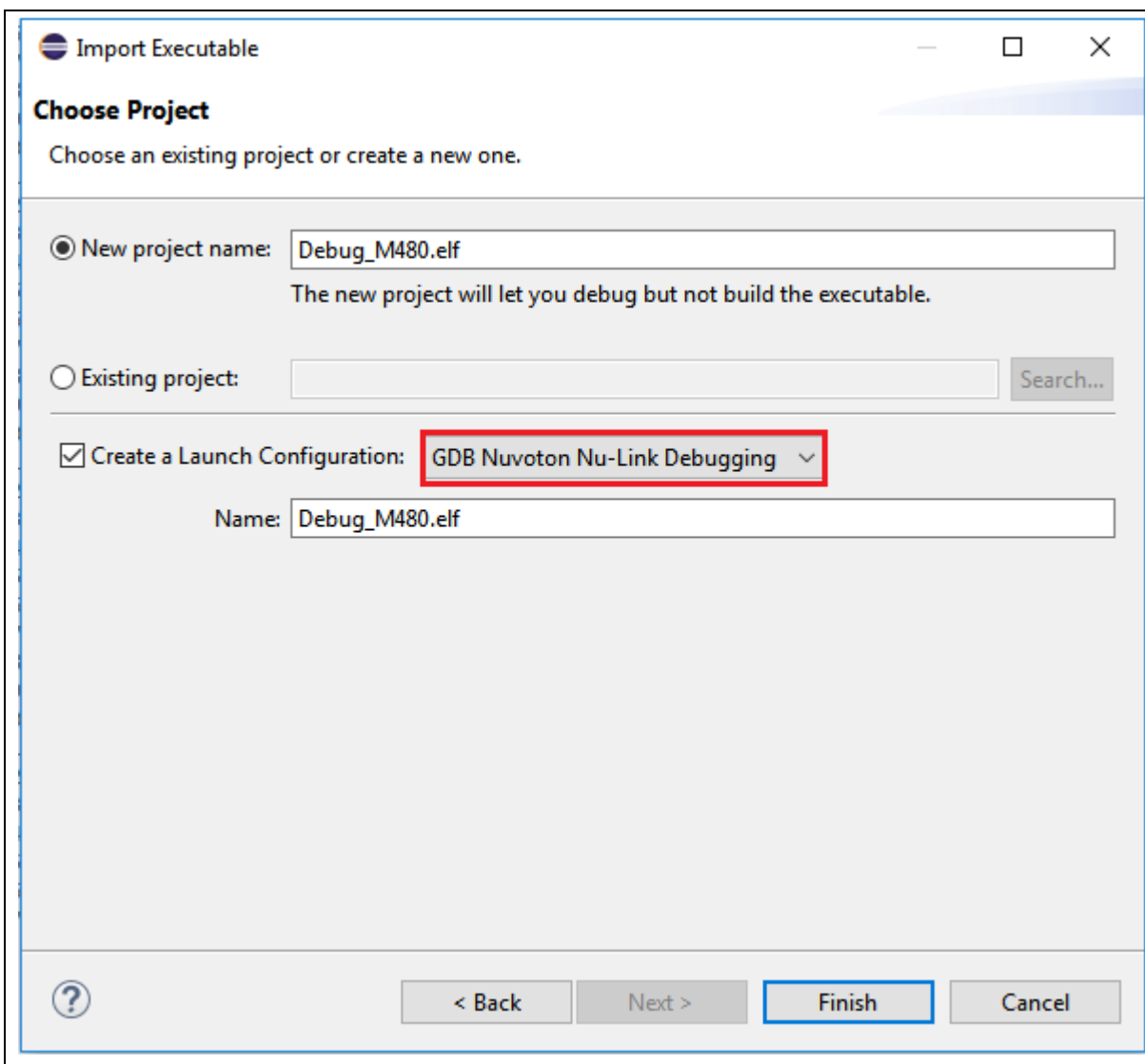


Figure 4-28 Choosing GDB Nuvoton Nu-Link Debugging

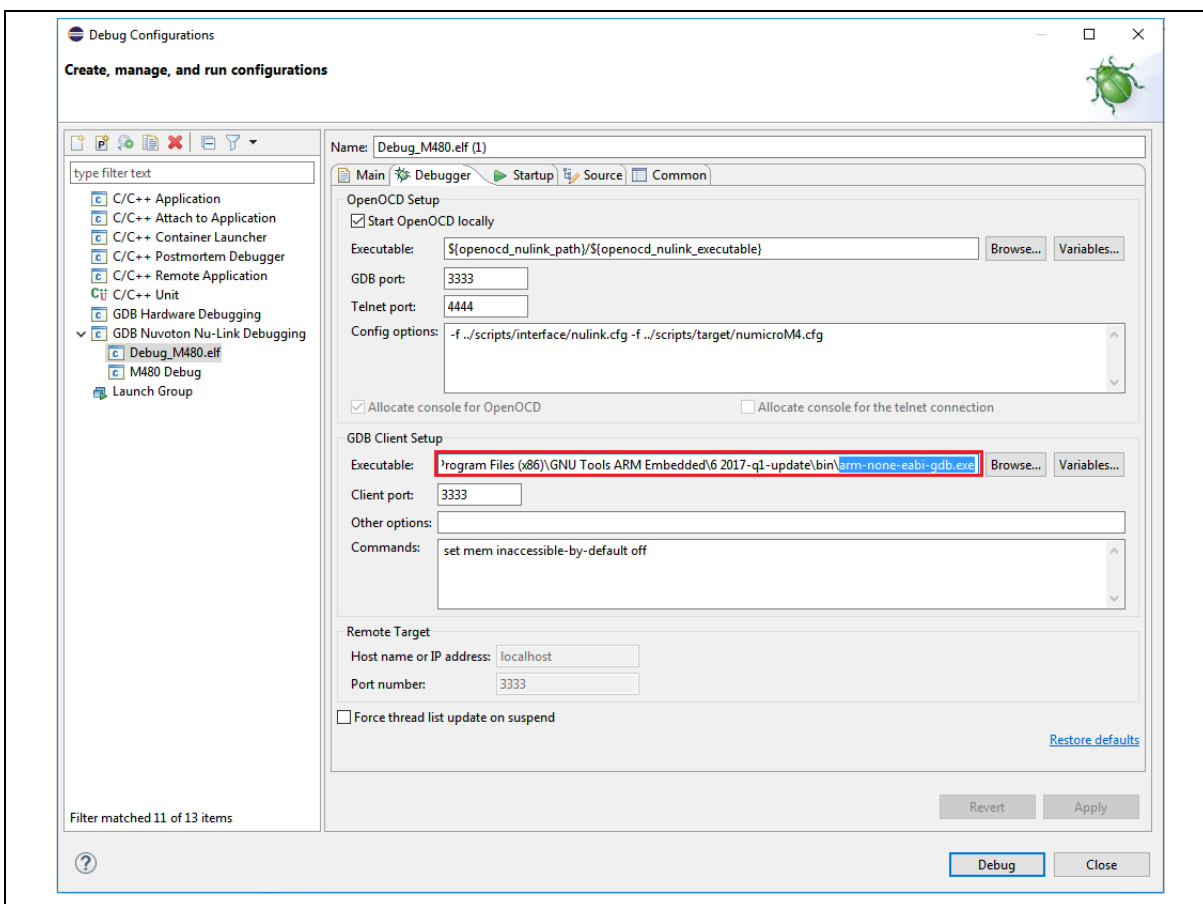


Figure 4-29 Locating the GDB Executable

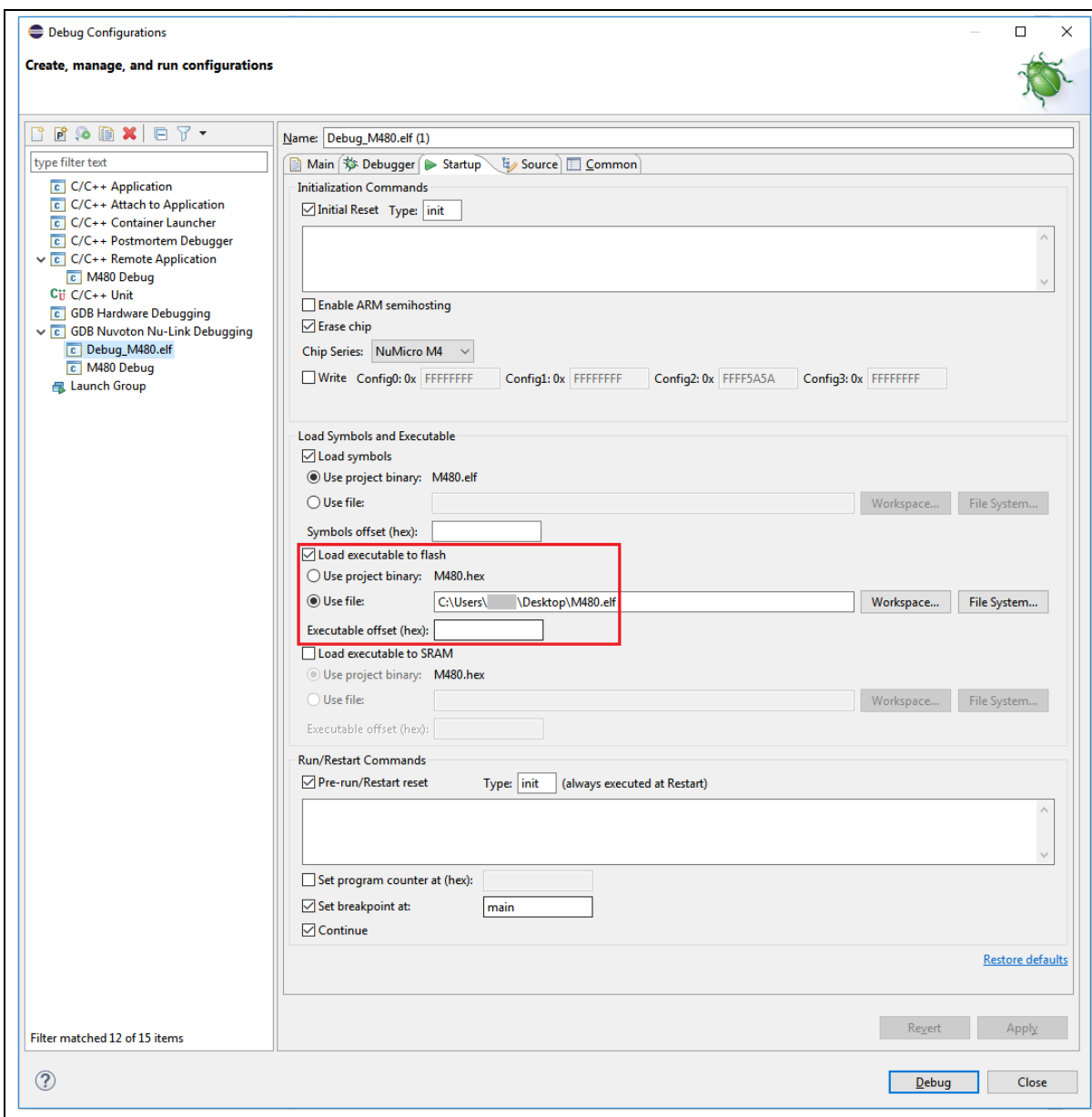


Figure 4-30 Choosing the ELF File to Download



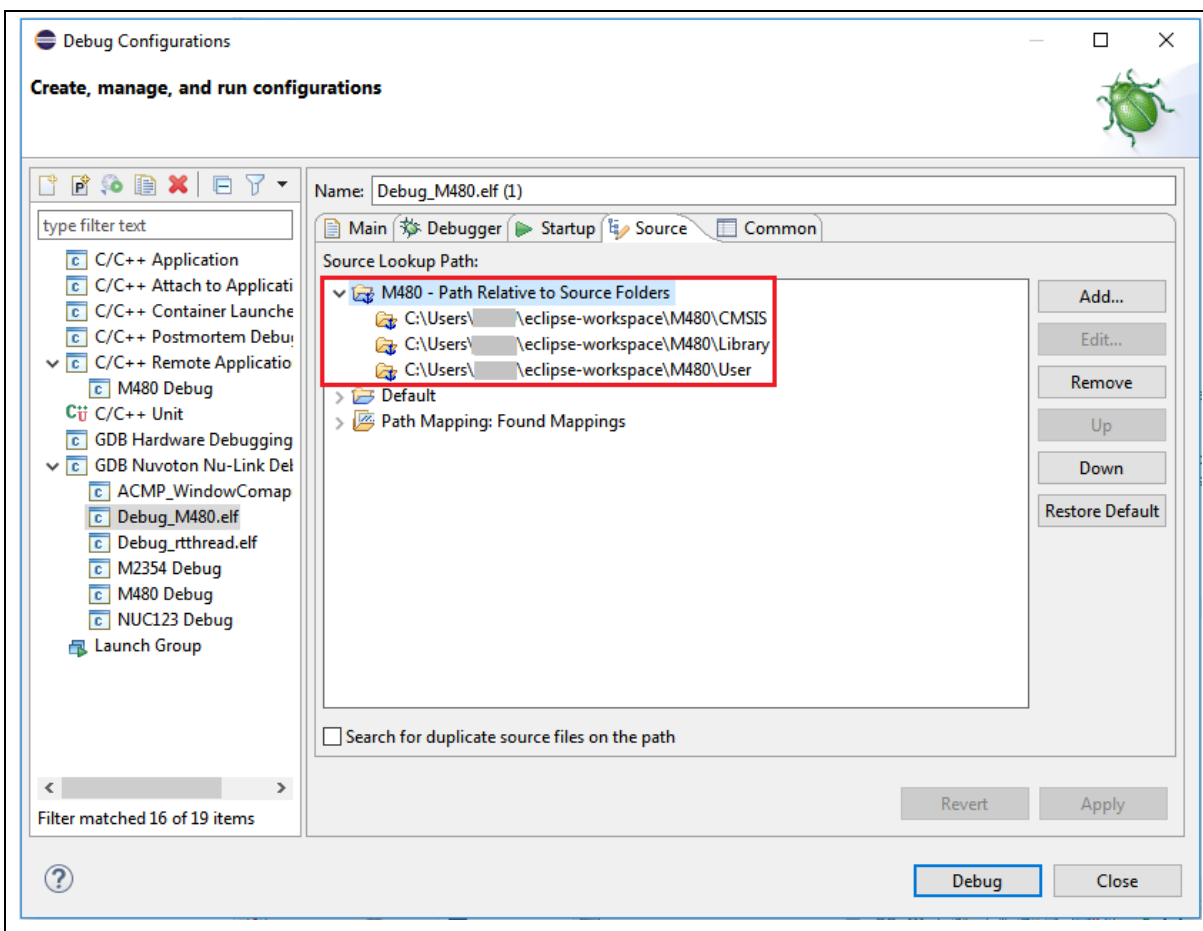


Figure 4-31 Adding Source Lookup Path

## 4.10 Configuration of Cortex-A, Multi-core SMP and AMP system

### 4.10.1 Preliminary Preparation

Suggest using **Nu-Link2** family as debugger, and being sure to upgrade Nu-Link2 firmware version higher than **v3.09.7380r**.

The term SMP stands for symmetric multiprocessing, AMP is asymmetric multiprocessing.

Nu-Link2 had built-in CMSIS-DAP interface from **v3.09.7380r**, and this CMSIS-DAP interface is used to support Single Cortex-A, Multi-Core SMP and AMP (Cortex-A + Cortex-M).

Besides, NuEclipse provides GNU toolchain for the Cortex-A family which resides in **NuEclipse/Others** folder. After configuring this toolchain (see Section 3.1.3), we can build MA35D1's projects.

### 4.10.2 Configuration of debugging Single-core Cortex-A

This section describes how to configure a single Cortex-A core debug.

Take Nuvoton MA35D1 as an example, which has dual Arm Cortex-A35 cores and one Arm Cortex-M4 core. To debug it, we should **choose NuMicro A35** and **resume** the target at the position where the executable starts running in the Startup tab, as follows. For more details, please refer to Section 4.5. To download other images, the command would be: **monitor load\_image filename address**.

To load other script, the command would be: **monitor script filename**.

E.g. load DDR initial script, as follows.

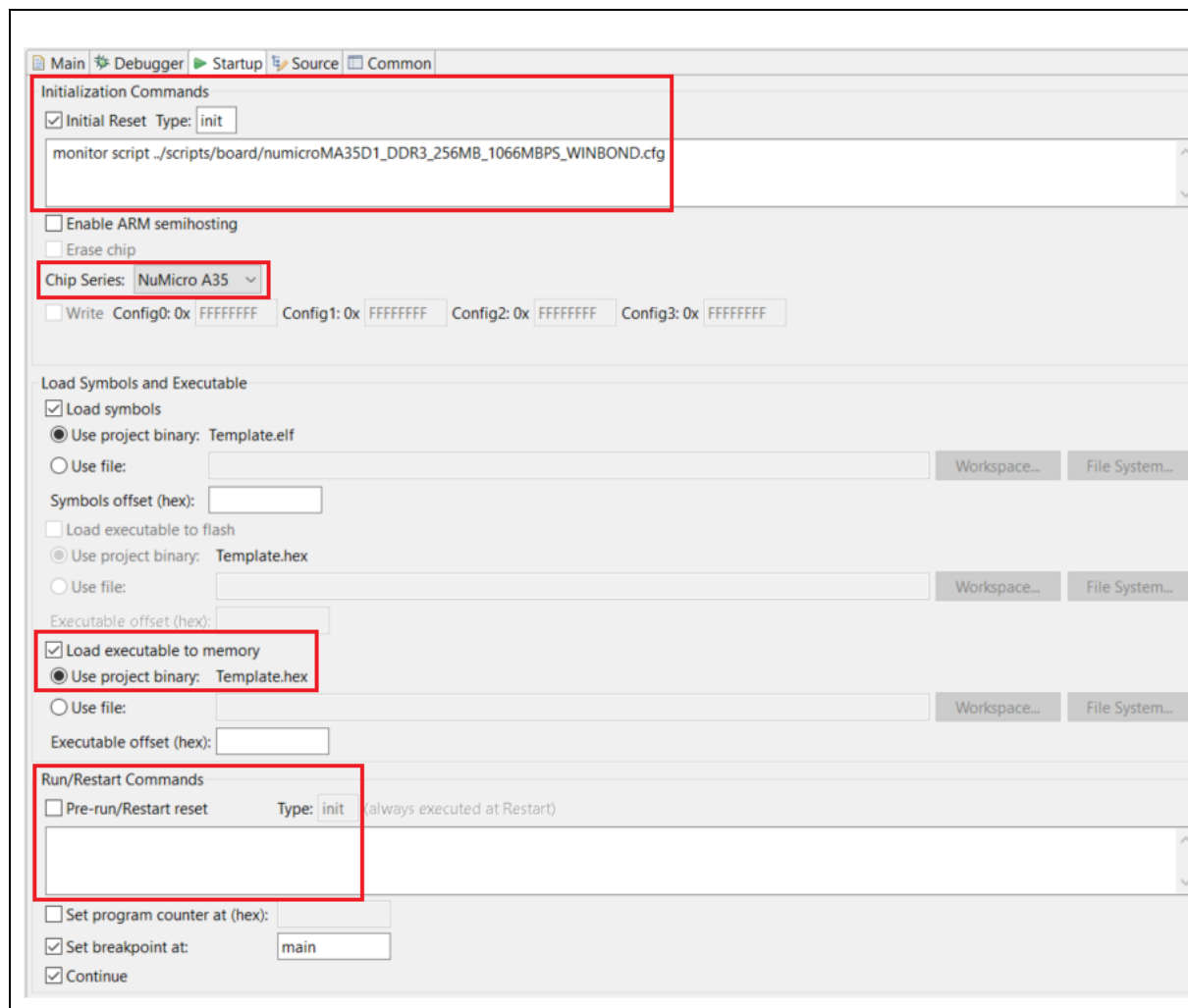


Figure 4-32 Configuring the Startup Tab for MA35D1

When the images are too large, the download command may face a timeout. To avoid that, go to **Window > Preferences > C/C++ > Debug > GDB** and disable the Command timeout or increase it.

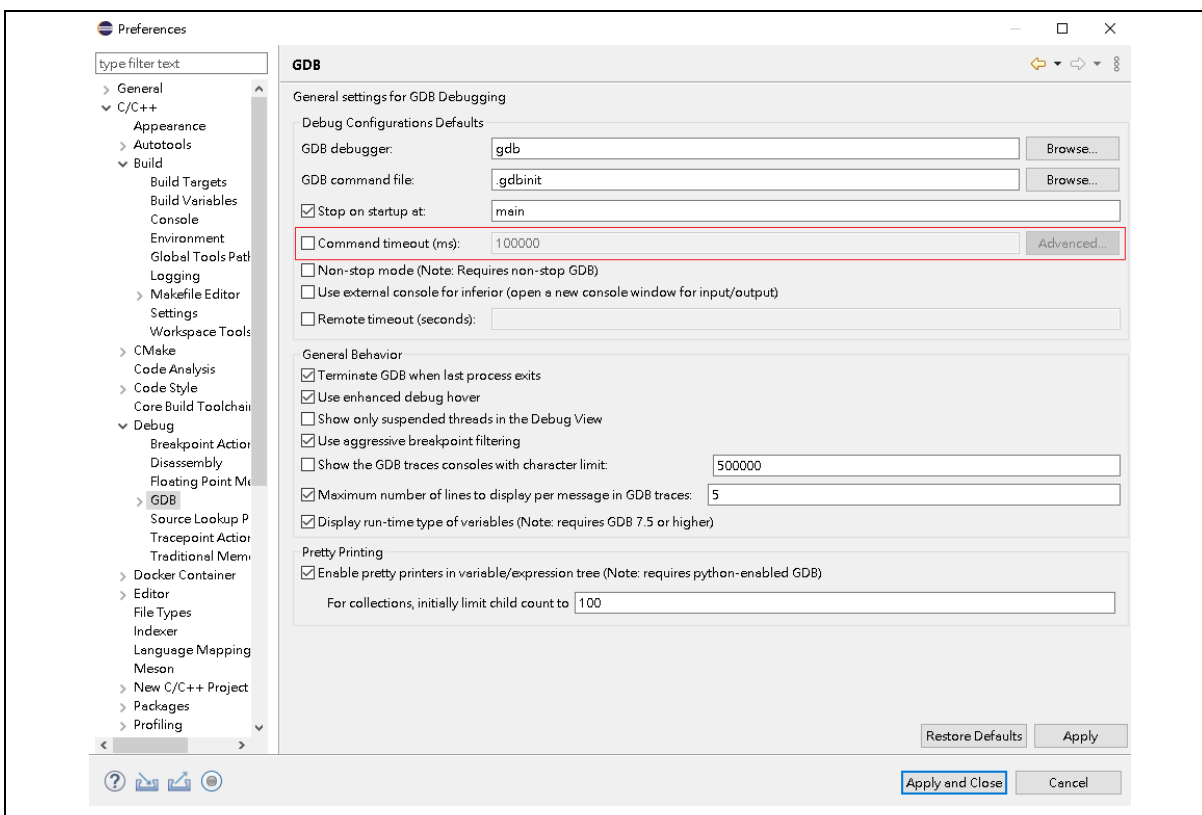


Figure 4-33 How to Disable Command Timeout

To debug single A35 core, we have to disable CPU1 setting by commenting out the following code in the numicroMA35D1.cfg. On Windows, the location where numicroMA35D1.cfg resides could be C:\Program Files (x86)\Nuvoton Tools\ OpenOCD\scripts\target. On GNU/Linux, it could be /usr/local/OpenOCD/scripts/target.

```
cti create $_CHIPNAME.cti.sys -dap $_CHIPNAME.dap -ap-num 1 -baseaddr 0xE0801000
cti create $_CHIPNAME.cti.cpu0 -dap $_CHIPNAME.dap -ap-num 1 -baseaddr 0xE0C20000
#cti create $_CHIPNAME.cti.cpu1 -dap $_CHIPNAME.dap -ap-num 1 -baseaddr 0xE0D20000
cti create $_CHIPNAME.cti.cm4 -dap $_CHIPNAME.dap -ap-num 2 -baseaddr 0xE0042000

# so defer-examine it until the reset framework get merged
# NOTE: keep ap-num and dbgbase to speed-up examine after reset
# NOTE: do not change the order of target create
target create $_CHIPNAME.axi mem_ap -dap $_CHIPNAME.dap -ap-num 0
target create $_CHIPNAME.cpu0 aarch64 -dap $_CHIPNAME.dap -ap-num 1 -coreid 0 -dbgbase 0xE0C10000 -cti $_CHIPNAME.cti.cpu0
#target create $_CHIPNAME.cpu1 aarch64 -dap $_CHIPNAME.dap -ap-num 1 -coreid 1 -dbgbase 0xE0D10000 -cti $_CHIPNAME.cti.cpu1
target create $_CHIPNAME.cm4 cortex_m -dap $_CHIPNAME.dap -ap-num 2

targets $_CHIPNAME.cpu0

#target smp $_CHIPNAME.cpu0 $_CHIPNAME.cpu1
$_CHIPNAME.cpu0 aarch64 maskisr on
#$_CHIPNAME.cpu1 aarch64 maskisr on

adapter srst delay 200
reset_config srst only
adapter speed 1000

proc cpu0_init {} {
    $_CHIPNAME.cpu0 aarch64 dbginit
}

#proc cpu1_init {} {
#    $_CHIPNAME.cpu1 aarch64 dbginit
#}

$_CHIPNAME.cpu0 configure -rtos hwthread
#$_CHIPNAME.cpu1 configure -rtos hwthread
$_CHIPNAME.cpu0 configure -event reset-assert {cpu0_init}
#$_CHIPNAME.cpu1 configure -event reset-assert {cpu1_init}
$_CHIPNAME.cm4 configure -event reset-assert {halt}

$_CHIPNAME.cpu0 configure -event gdb-detach { shutdown }
```

Figure 4-34 Disable CPU1 Setting

After choosing NuMicro A35 in the Startup tab, the **numicroMA35D1.cfg** will be automatically updated in the **Config options** field of the Debugger tab. Because of single-core debugging, the OpenOCD GDB port and GDB client port should be the same value, e.g., **3333**. When all the settings are done, click the **Apply** button to take effect. To launch the application into the debug mode, click the **Debug** button.

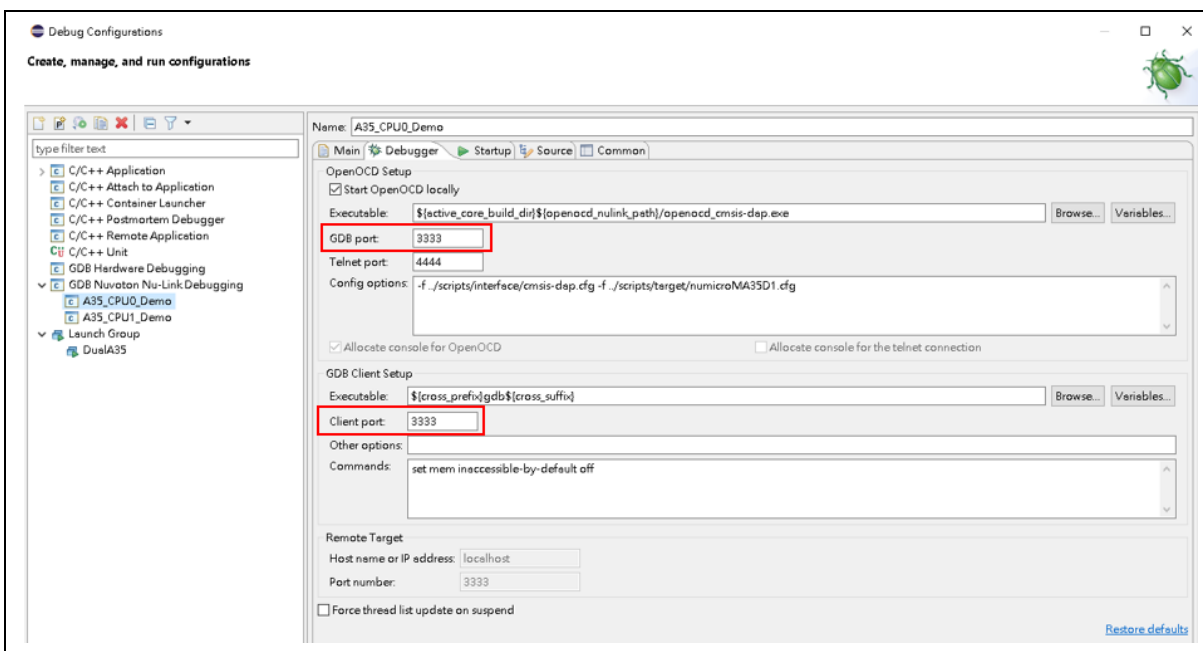


Figure 4-35 Configuring the Debugger Tab for Single-core Debugging on MA35D1

### 4.10.3 Configuration of debugging Dual-core Cortex-A in SMP mode

To debug SMP, e.g. dual A35 core, please refer to Section 4.10.2, but bypass the step that disabling CPU1 setting.

### 4.10.4 Configuration of debugging Single-core Cortex-A and Cortex-M in AMP mode

To debug AMP, e.g. one A35 core and M4 simultaneously, we build the A35 and M4 projects and create the corresponding debug configurations (see Section 4.5). The debug configuration of A35 follows previous section to complete settings. However, the debug configuration of **M4** differs from the Debugger Tab. Uncheck the **Start OpenOCD locally** checkbox. The value of GDB Client port and Remote Target port is set to **3334**.

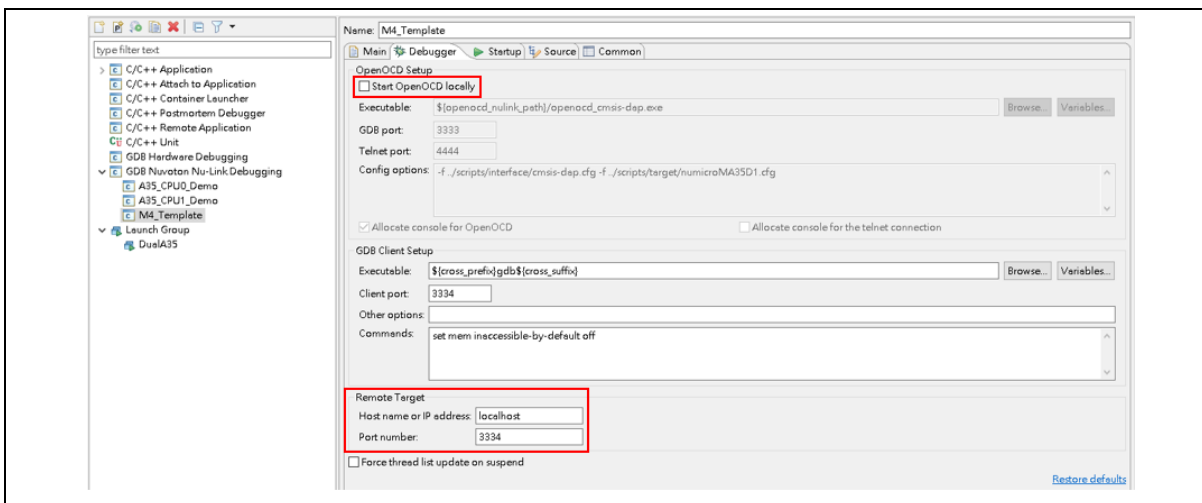


Figure 4-36 Configuring the Debugger Tab for Multi-core Debugging on M4 core

To launch A35 and M4 applications simultaneously, we use **Launch Group** by the following steps:

1. Give a name to the newly created launch group.
2. Press on the Add button.
3. Add the expected launch configurations to the launch group.
4. Press on the OK button.
5. Move the A35 launch configuration in the first sequence order.
6. Press on the Apply button to take effect.
7. Press on the Debug button to run the group launch.

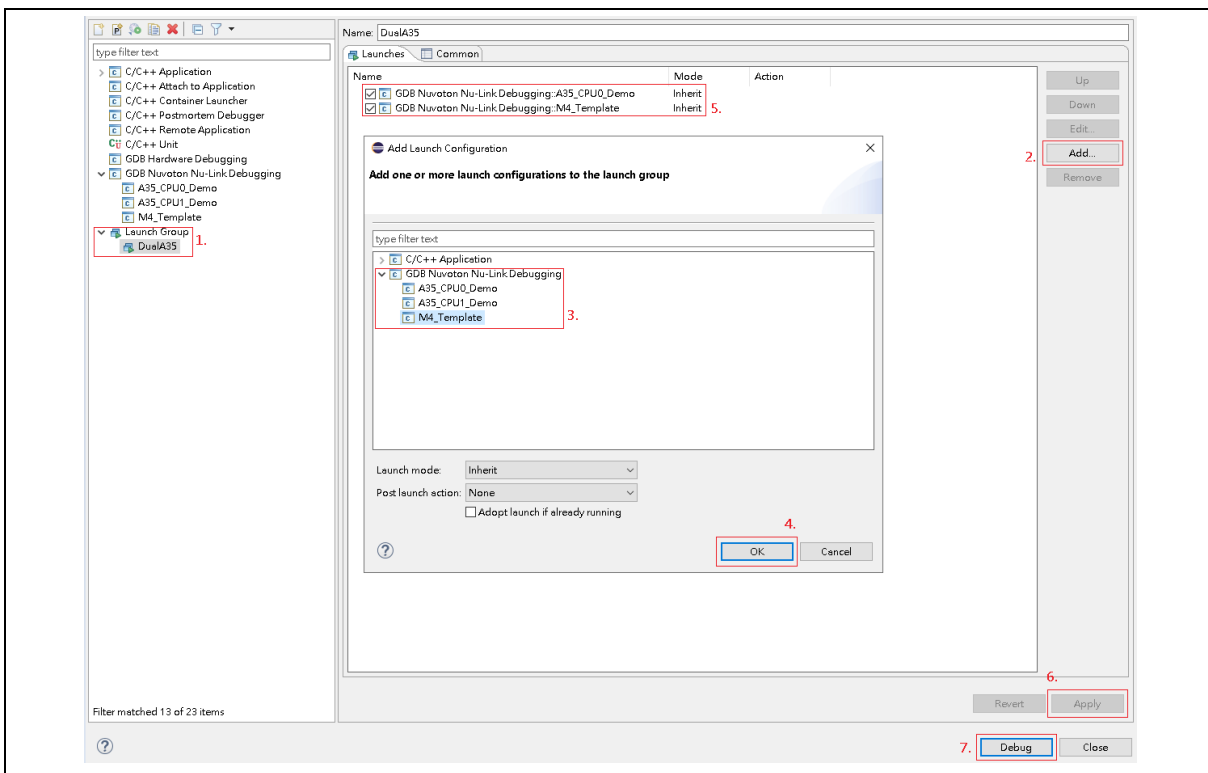


Figure 4-37 Configuring the Launch Group



#### 4.10.5 Configuration of debugging Dual-core Cortex-A and Cortex-M in AMP mode

To debug AMP, e.g. dual A35 core and M4 simultaneously, we have to disable the CPU Symmetric Multi-Processors (SMP) by commenting out the following code in the numicroMA35D1.cfg. On Windows, the location where numicroMA35D1.cfg resides could be C:\Program Files (x86)\Nuvoton Tools\OpenOCD\scripts\target. On GNU/Linux, it could be /usr/local/OpenOCD/scripts/target.

```
targets $_CHIPNAME.cpu0

#target smp $_CHIPNAME.cpu0 $_CHIPNAME.cpu1
$_CHIPNAME.cpu0 aarch64 maskisr on
$_CHIPNAME.cpu1 aarch64 maskisr on
```

Figure 4-38 How to Disable SMP

The debug configuration of A35 CPU0 follows Section 4.10.2 to complete settings. The debug configuration of A35 CPU1 and M4 follows Section 4.10.3 to complete settings, but the value of GDB Client port and Remote Target port differs. That of **A35 CPU1** and **M4** is **3334** and **3335**, respectively. The Launch Group should include the launch configurations of A35 CPU0, CPU1 and M4. Make sure that the **A35 CPU0** launch configuration is in the **first** sequence order, then press on the Debug button to run the group launch. While entering the debug mode, we can switch different CPU core to debug by clicking on the corresponding execution of threads in the Debug view.

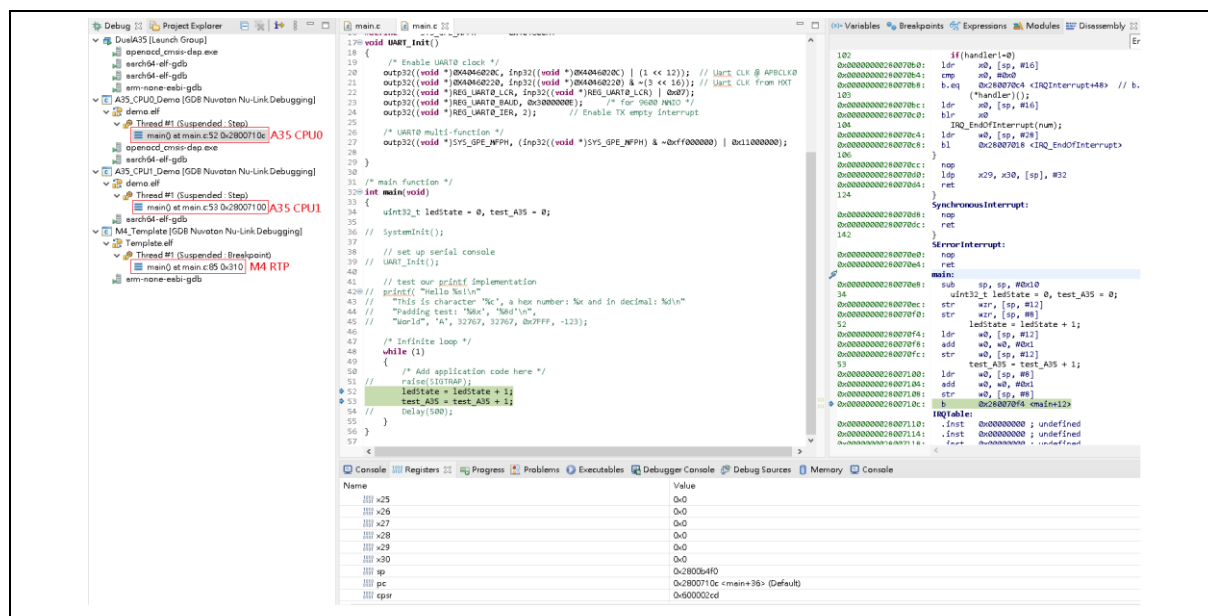


Figure 4-39 Multi-core Debugging

#### 4.10.6 Configuration of debugging Cortex-A Core in AARCH32 mode

The settings in chapter 4.10 are mostly for ARM v8a AARCH64 mode by default. In this section, we describe how to debug a AARCH32 application.

For example, to debug AARCH32 program on Cortex-A35, we first enter debug mode to run AARCH64 code and leave debug mode. And then re-enter debug mode switching to AARCH32 code for debugging. We import executable files (see Section 4.9) and create the corresponding debug configurations (see Section 4.5). On Windows, the location where numicroMA35D1.cfg resides could be

C:\Program Files (x86)\Nuvoton Tools\OpenOCD\scripts\target. On GNU/Linux, it could be /usr/local/OpenOCD/scripts/target. There are several steps to follow:

1. Import an executable for debugging (referring to Figure 4-26).
2. Click browse following select executable, then select an executable (referring to Figure 4-27).
3. Choose GDB Nuvoton Nu-Link Debugging as a Launch Configuration (referring to Figure 4-28).
4. Locate the GDB executable in the debug configuration (referring to Figure 4-40).
5. Choose the ELF file to download and initial setting in the debug configuration (referring to Figure 4-41). Please based on DDR of dev-board to select the corresponding DDR initial file.
6. These files could be found in C:\Program Files (x86)\Nuvoton Tools\OpenOCD\scripts\board folder.
7. (monitor Script ../scripts/board/numicroMA35D1\_DDR3\_256MB\_1066MBPS\_WINBOND.cfg)
8. Press on the Debug button.
9. Set breakpoint at the last address of aarch64 program and resume to the address.
10. Press run button and then leave debug mode.
11. Uncheck the reset settings and re-enter debug mode (referring to Figure 4-42).
12. Debug AARCH32 program (referring to Figure 4-43).

The above description is for debugging AARCH32 program in dual core SMP mode. AARCH32 program debugging can also be used in the following two settings

1. Refer to Figure 4-34 to disable CPU1 setting for single core mode.
2. Refer to Figure 4-38 to disable the CPU SMP for AMP mode.

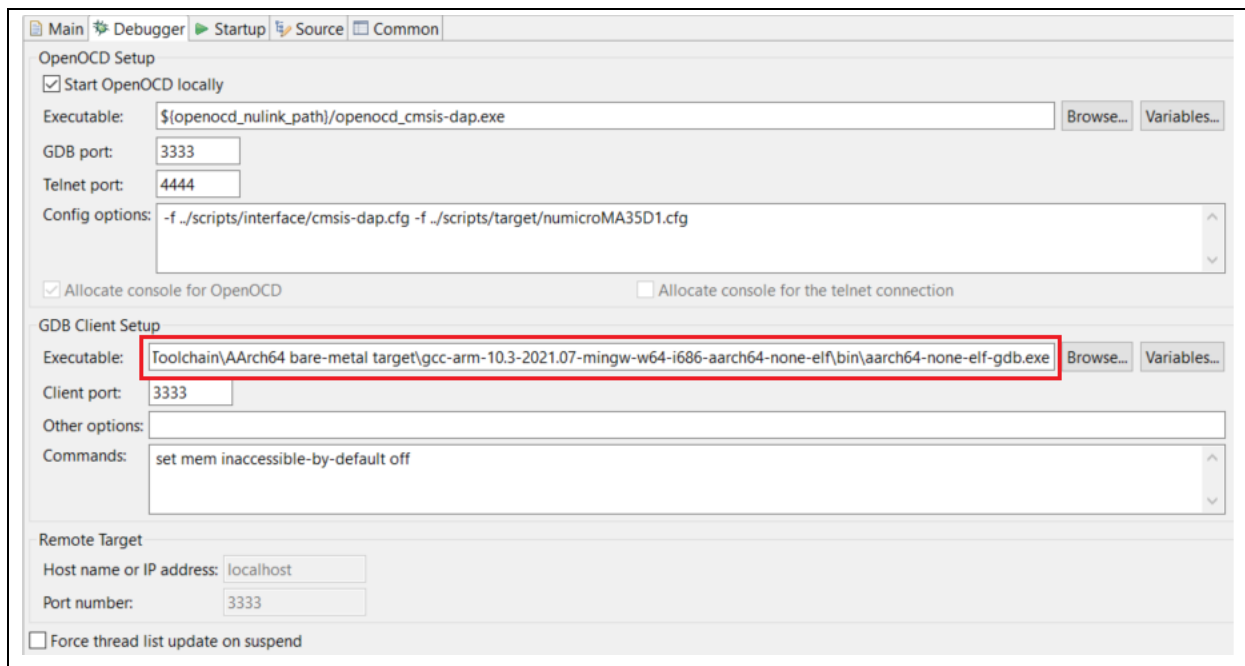


Figure 4-40 Locating the AARCH64 GDB Executable

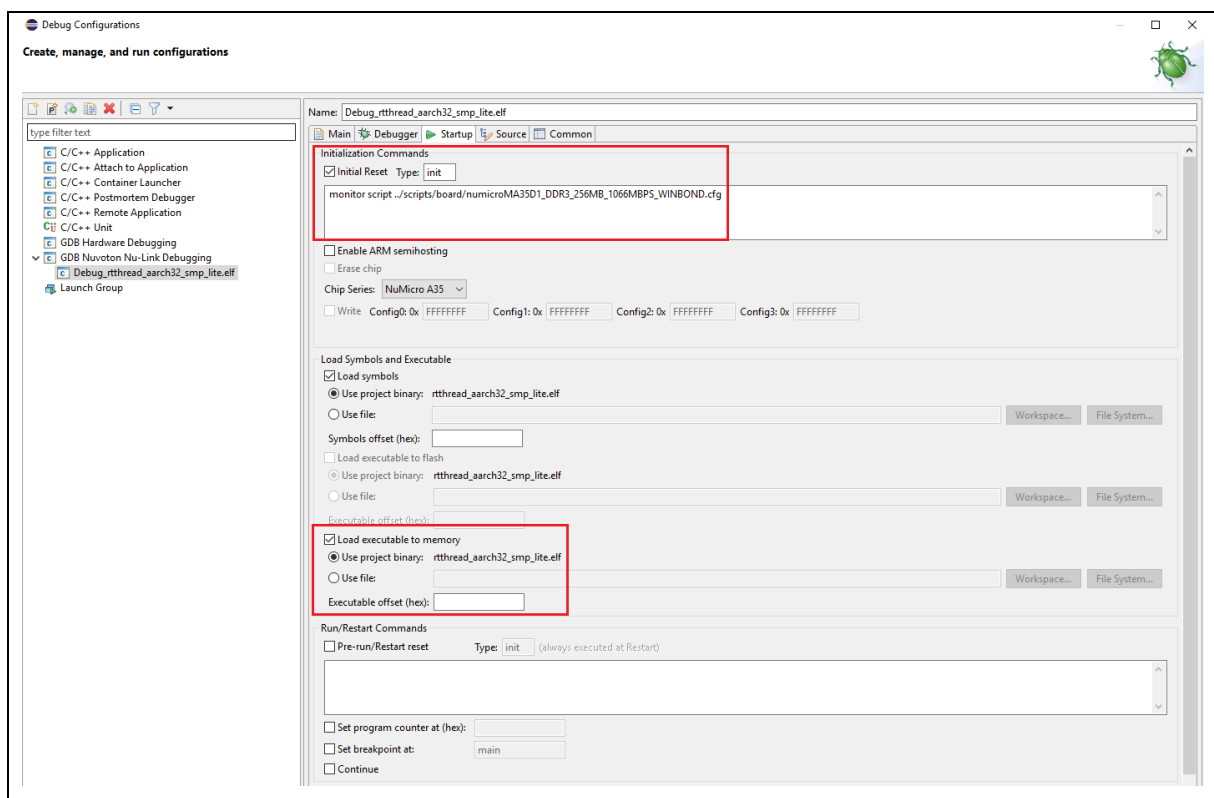


Figure 4-41 The First Time to Enter the Debug Mode Settings

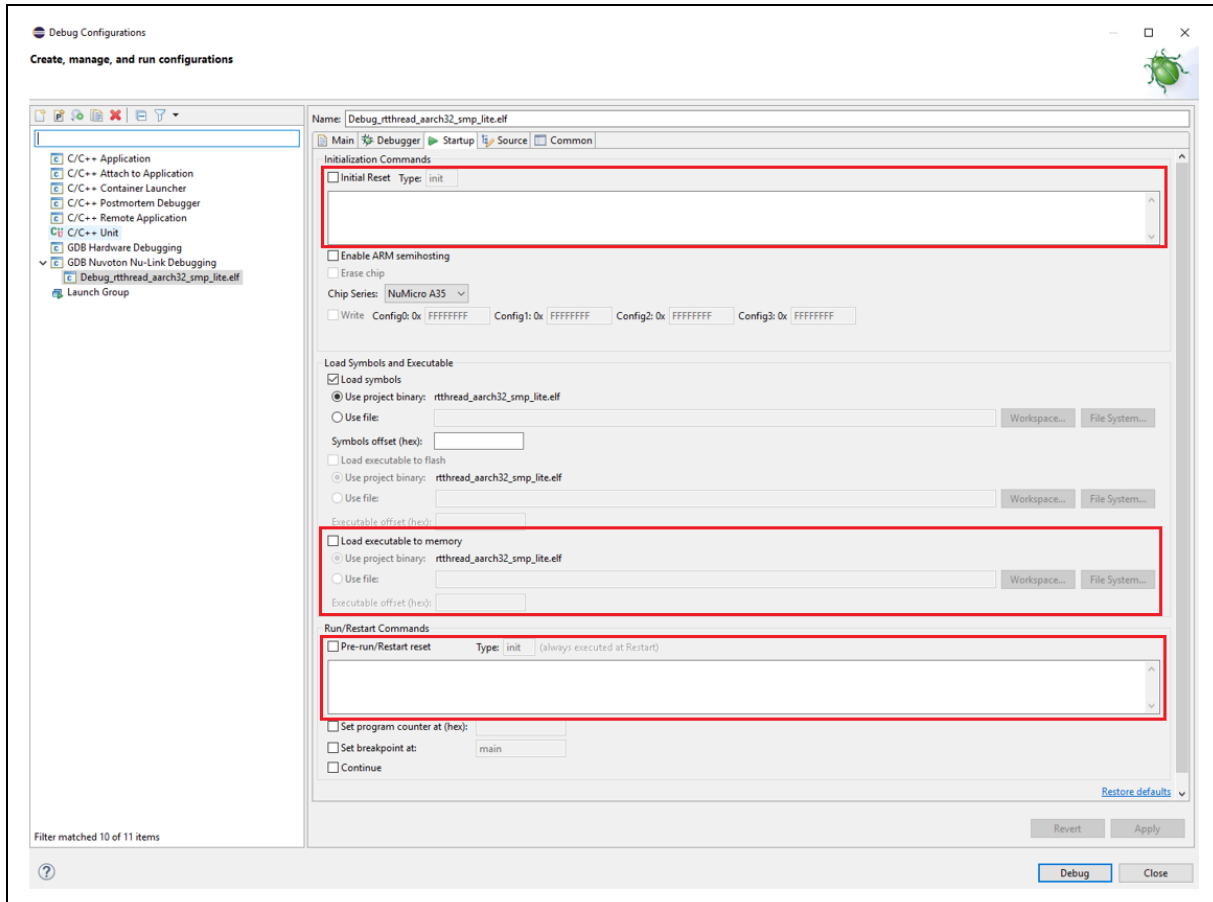


Figure 4-42 The Second Time to Enter the Debug Mode Settings

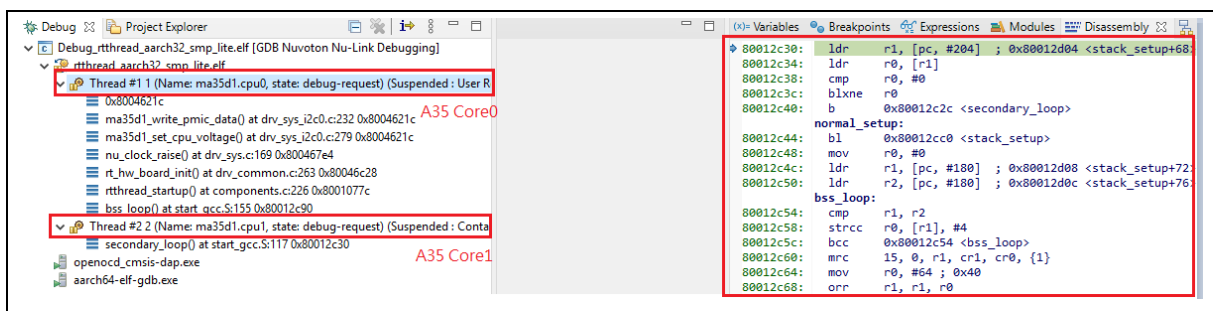


Figure 4-43 AARCH32 Program Debugging

## 4.11 Configuration of NuMicro 8051 1T

Please download NuEclipse\_Windows (For NuMicro 8051) from Nuvoton website.

### 4.11.1 Import and Build Project

When **NuMicro 8051 1T** BSP projects are available, we can import them into the workspace. For more details, please refer to Section 4.3.

Refer to Section 4.4 to build project. After building project successfully, **Console** View would show checksum information and generate bin file.

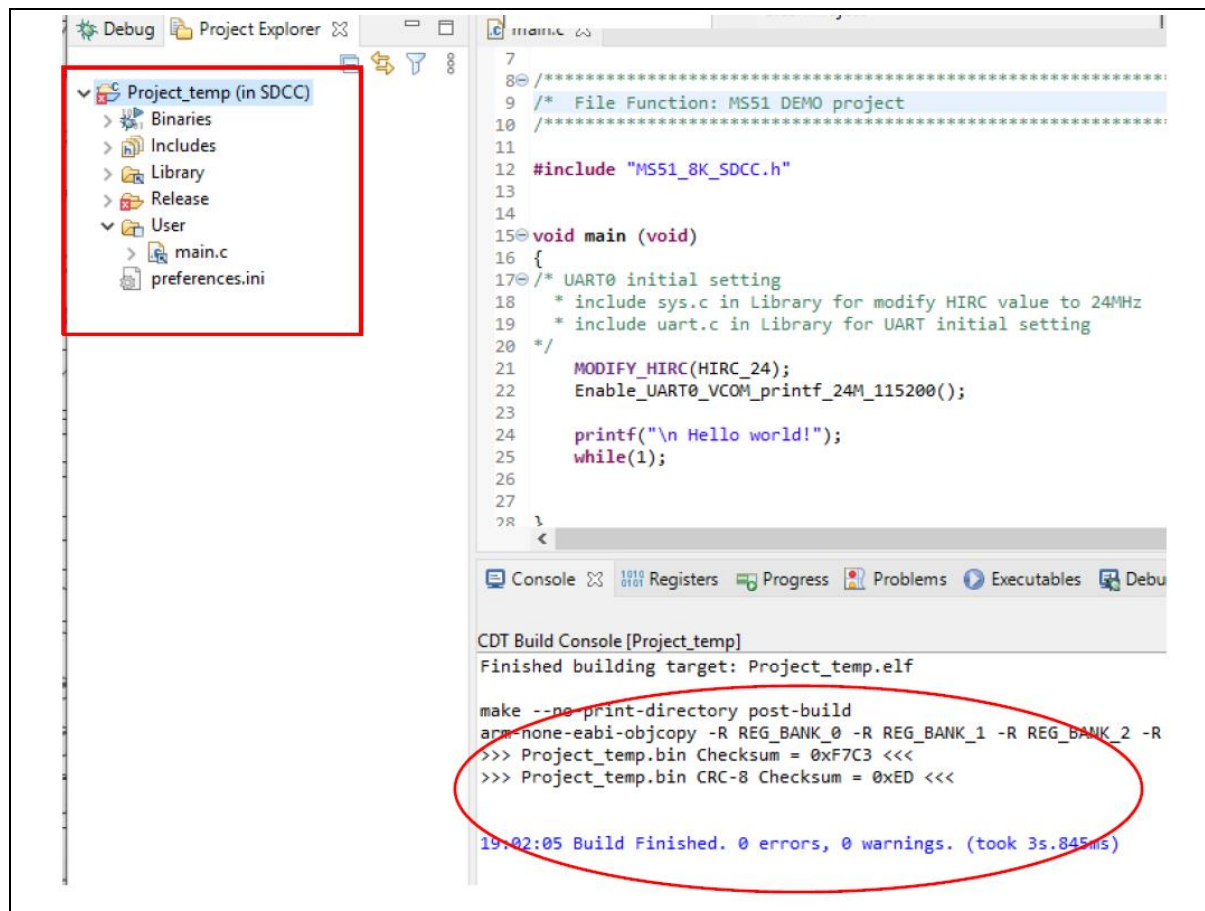


Figure 4-44 Build NuMicro 8051 1T project

### 4.11.2 Debug Configuration

This section describes how to configure NuMicro 8051 1T debugging.

In debugger tab, the configuration files are **numicro8051\_1T.cfg** in the **Config options** field. The install path of gdb.exe must be set in **GDB Client Setup Executable** field. Refer to Figure 4-45.

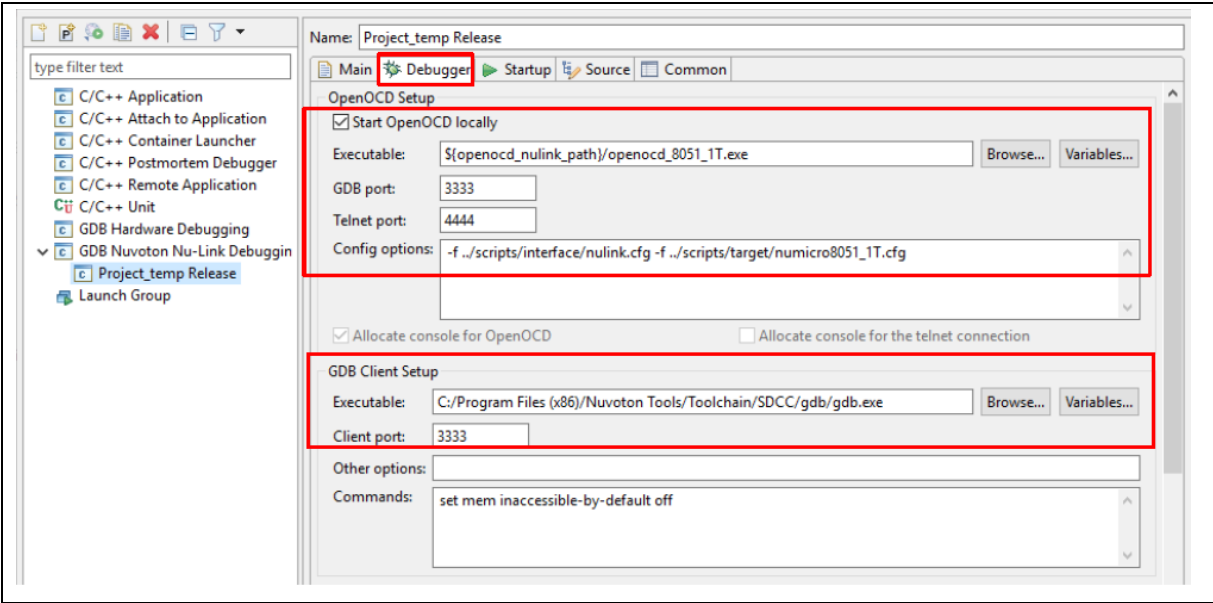


Figure 4-45 NuMicro 8051 1T Debug Configuration in Debugger Tab

4.11.3 Monitor different Memory Space

NuEclipse provides different 8051 memory space access. Variables may be explicitly assigned to a specific memory space (by including a memory space specifier in the declaration) or implicitly assigned (based on the memory model).

According to different memory space, the corresponding offset position must be specified in order to access specific memory space data. Please refer to the table below.

Memory space of SDCC declaration	Data access address
__code	(CODE address) + 0
__data	(DATA address) + 0xF0000000
__idata	(IDATA address) + 0xF1000000
__xdata	(XDATA address) + 0xF2000000

Table 4-1 Access Memory Space with Offset

For example, to access the 0x10 location of **\_\_xdata** memory, user need to fill 0xF2000010 into the **Memory Browser** or **Memory View**. Refer to Figure 4-46.

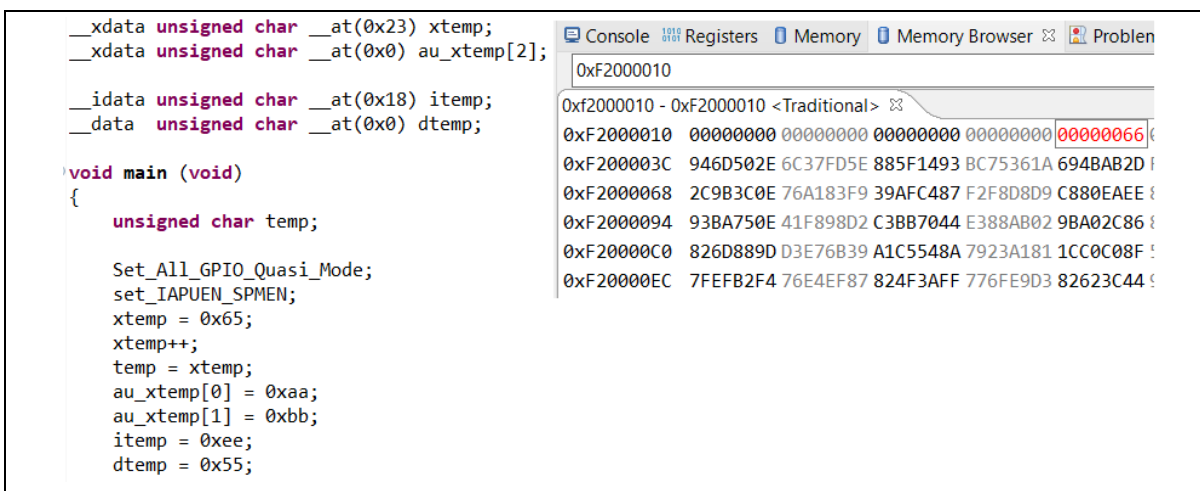


Figure 4-46 Access Memory Space with Memory Browser

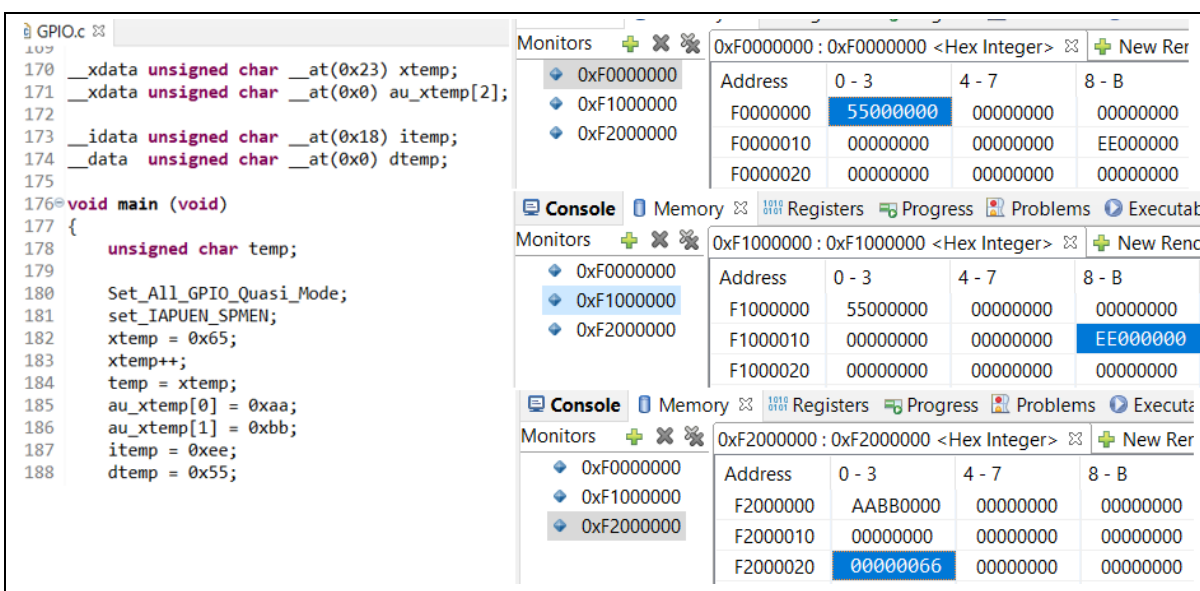


Figure 4-47 Access Memory Space with Memory View

To monitor the variable value, user can observe the variable through the **Watch Expressions View** and add the specified offset address to access the variable value in different memory space.

Figure 4-48 Access Variable with Watch Expressions View



## 5 Q&A

1. Q: Can we simultaneously debug on Eclipse, Keil and Iar?

A: No, we must stop the debug mode on Eclipse first. Then we can debug on another IDE.

2. Q: Can we simultaneously debug on Eclipse and use the Nuvoton development tools, such as ICP Programming tool?

A: No, we must stop the debug mode on Eclipse first. Then we can use them and vice versa.

3. Q: How many breakpoints and watchpoints are supported?

A: It depends on the hardware. For M0 chips, the supported number of breakpoints and watchpoints is 4 and 2, respectively. For M4 chips, the supported number of breakpoints and watchpoints is 6 and 4, respectively. For M23 chips, the supported number of breakpoints and watchpoints is 4 and 4, respectively. For 8051 1T chips, the supported number of breakpoints is 8. For now, we do not support flash breakpoints.

4. Q: How to update firmware for Nu-Link?

A: Please use ICP Programming tool or Keil to update firmware. Visit [https://github.com/OpenNuvoton/Nuvoton\\_Tools](https://github.com/OpenNuvoton/Nuvoton_Tools) website to get further information.

5. Q: How to change Flash and RAM size after projects are created?

A: Please find and open the Id script in the Id scripts folder. From there, we can change Flash and RAM size.

6. Q: Why can the application not enter the debug mode?

A: Firstly, we must install all the stuff by following the previously mentioned steps. Then check the following list:

- Leave the previous debug mode first if exists.
- Flash and RAM size must be correct.
- OpenOCD should not be launched before debugging. To check that, please go to Windows Task Manager or System Monitor. If an OpenOCD process has already been running, please end the process.
- The target chip should not be held by other tools or IDE.
- The **Config options** field of the Debugger tab must be correct.
- The **GDB Client Setup Executable** field of the Debugger tab must be correct.
- In the Startup tab, the **Initial Reset** type should be **init**. The **Pre-run/Restart reset** type should be **init**.
- The Eclipse preferences must be correct. Please refer to the previous discussion.
- **Upgrade Nu-Link firmware and USB driver to the latest one.**
- Check whether the hex file is generated under the project successfully.
- Check whether the executable has been downloaded to the target chip correctly.
- Check SP. If it is wrong, please assign it to the correct location.
- Write a correct Config value into the target chip.
- If the operating system is Windows 7, please use USB 2.0 port, instead of USB 3.0 port.
- The project path must not contain any special character or whitespace, such as # \$ & ' . { } .
- Restart the computer.

## 7. Q: How to add udev rules for Nu-Link on GNU/Linux?

A: When accessing target chips via Nu-Link, GNU/Linux requires the USB permission. We can get the permission by adding udev rules for Nu-Link. Here are the steps to do that:

- Add the User to the plugdev Group. Type the command in the Terminal:  
`sudo useradd -G plugdev $USER`
- Add Nu-Link to udev. Type the commands in the Terminal:  
`cd /etc/udev/rules.d` and `sudo gedit 10-openocd-nulink.rules`
- Add the following text to the file

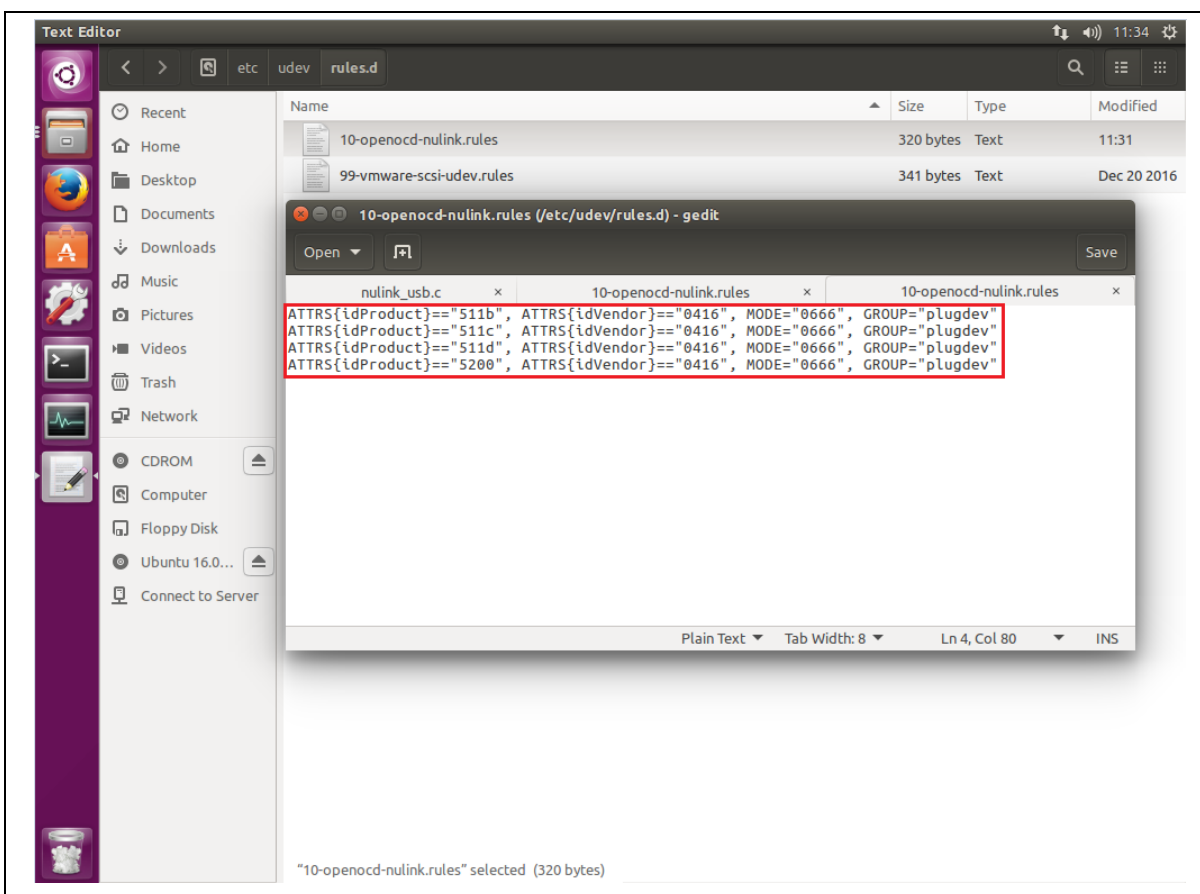


Figure 5-1 Adding Udev Rules

- Reloaded the new udev rules by entering the command in the Terminal:  
`sudo udevadm trigger`

8. Q: How to edit string substitution for openocd\_nulink\_path?

A: The **openocd\_nulink\_path** string stores where the OpenOCD executable resides. After upgrading NuEclipse, the string may keep the previous OpenOCD path. To fix it, click **Window > Preferences**, the Preferences wizard appears. Go to **Run/Debug > String Substitution**. Find and edit the openocd\_nulink\_path to the new OpenOCD path.

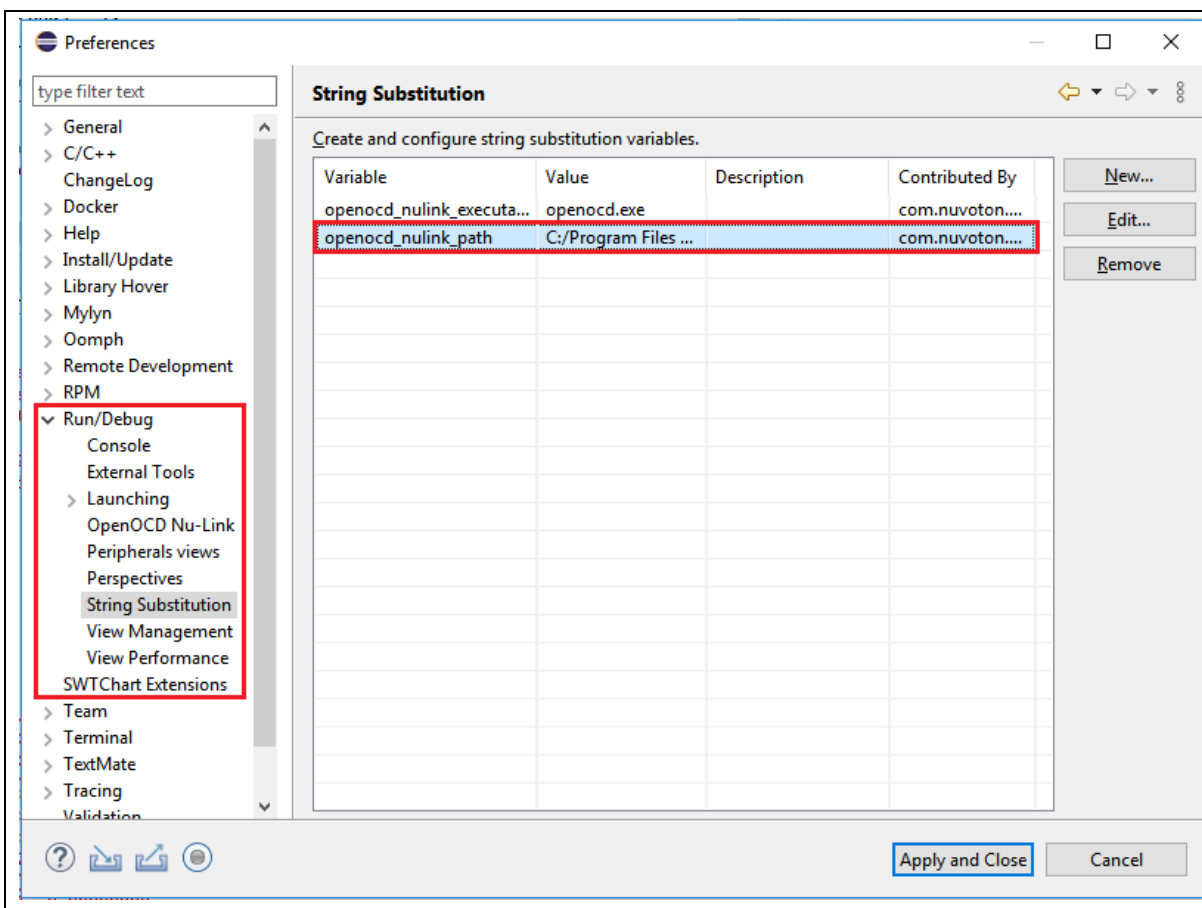


Figure 5-2 Preferences for String Substitution

9. Q: Why does Eclipse fail to update or install new software packs on Windows?

A: One of possible reasons is that the write permission for Windows folders is denied. We need to find the correct folder and allow the write permission. On Windows, the location where we place software packs is C:\Program Files (x86)\Nuvoton Tools\Packages.

10. Q: How to adjust output voltage of Nu-Link2?

A: Upgrade the NuLink2 firmware greater than version v3.08.7249, open NU\_CFG.TXT file in pop-up "NuMicro MCU" disk, you will see some options in NU\_CFG.TXT.

Set CMSIS-DAP=1 then re-plug in USB cable, it presents one more interface HID\_CMSIS-DAP, this is handy if you want to use CMSIS-DAP protocol.

Set LEVEL=3 then re-plug in USB cable, it presents one of output voltage, this is handy if you want to use 5V voltage.

```
[Power Control]
LEVEL=3
; Default I/O voltage (Level0: 1.8V, Level1: 2.5V, Level2: 3.3V, Level3: 5V)

[Interface configuration]
CMSIS-DAP=1
; 0 = disable
; 1 = enable
```

Figure 5-3 More Options for NuLink2

## 6 REVISION HISTORY

Date	Revision	Description
2017.03.31	0.01.000	Alpha version released.
2017.06.30	1.01.000	Beta version released.
2018.09.15	1.01.013	Official version released.
2018.11.30	1.01.014	1. Supported NUC505. 2. Updated the new project wizard.
2019.08.09	1.01.015	Supported M031 ,M261 and M480LD.
2020.03.06	1.01.016	Supported M031BT, NUC1311, M2354 and M479.
2020.09.30	1.01.017	Supported M030G, M071, M0A21, M251 and M471.
2021.03.12	1.01.018	Supported M030GM031G and NUC1262.
2021.12.01	1.01.019	Supported M460, I94100 and KM1M7.
2022.04.25	1.02.019	Supported MA35D1.
2023.02.07	1.02.022	Supported 8051 1T.
2023.05.26	1.02.023	Supported M2L31.
2023.08.18	1.02.024	Supported MG51, N76S003 and MUG51.

### Important Notice

Nuvoton Products are neither intended nor warranted for usage in systems or equipment, any malfunction or failure of which may cause loss of human life, bodily injury or severe property damage. Such applications are deemed, "Insecure Usage".

Insecure usage includes, but is not limited to: equipment for surgical implementation, atomic energy control instruments, airplane or spaceship instruments, the control or operation of dynamic, brake or safety systems designed for vehicular use, traffic signal instruments, all types of safety devices, and other applications intended to support or sustain life.

All Insecure Usage shall be made at customer's risk, and in the event that third parties lay claims to Nuvoton as a result of customer's Insecure Usage, customer shall indemnify the damages and liabilities thus incurred by Nuvoton.

---

*Please note that all data and specifications are subject to change without notice.  
All the trademarks of products and companies mentioned in this datasheet belong to their respective owners.*