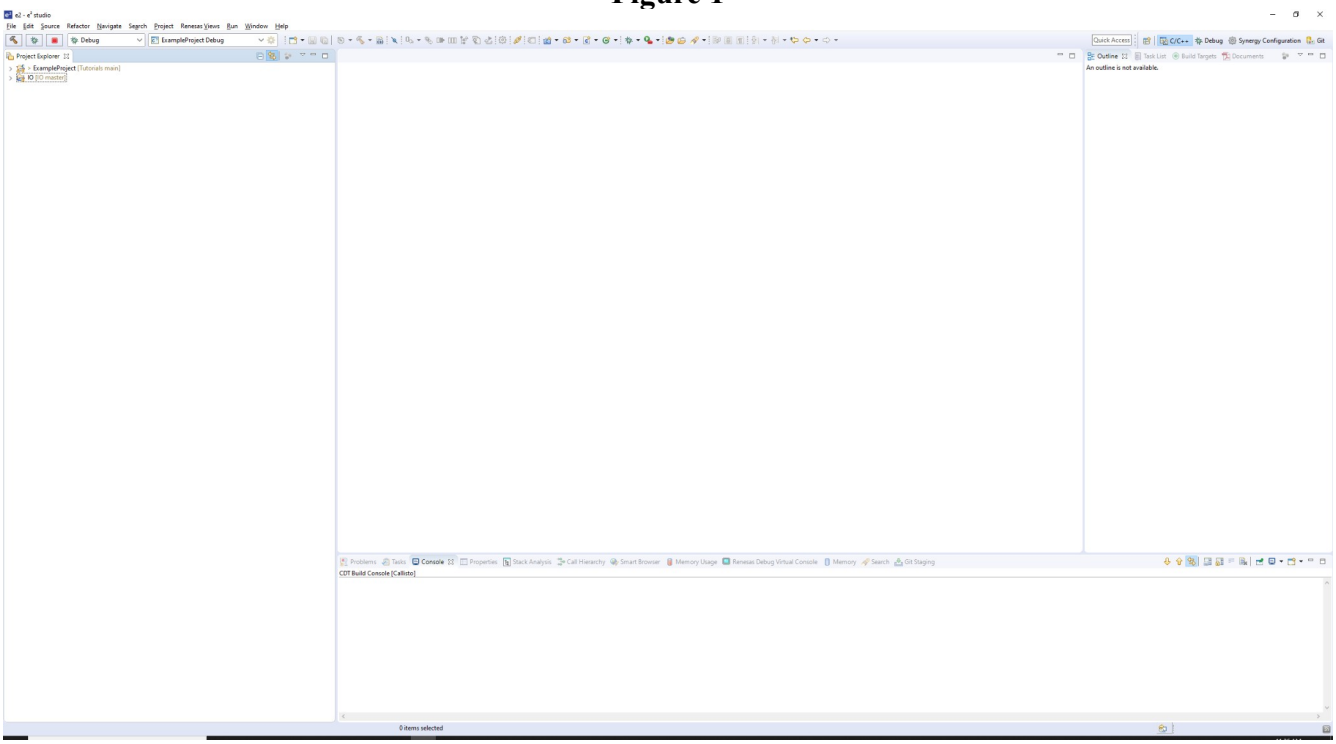
Table of Contents

Project Creation Tutorial

Introduction

This document explains how to create a new project and begin working with the Ganymede interface libraries. To begin, you should have cloned the Tutorials repository along with any library repositories you wish to work with, and imported the enclosed projects into e2 studio. You should see the ExampleProject and library projects in the Project Explorer view as shown below:

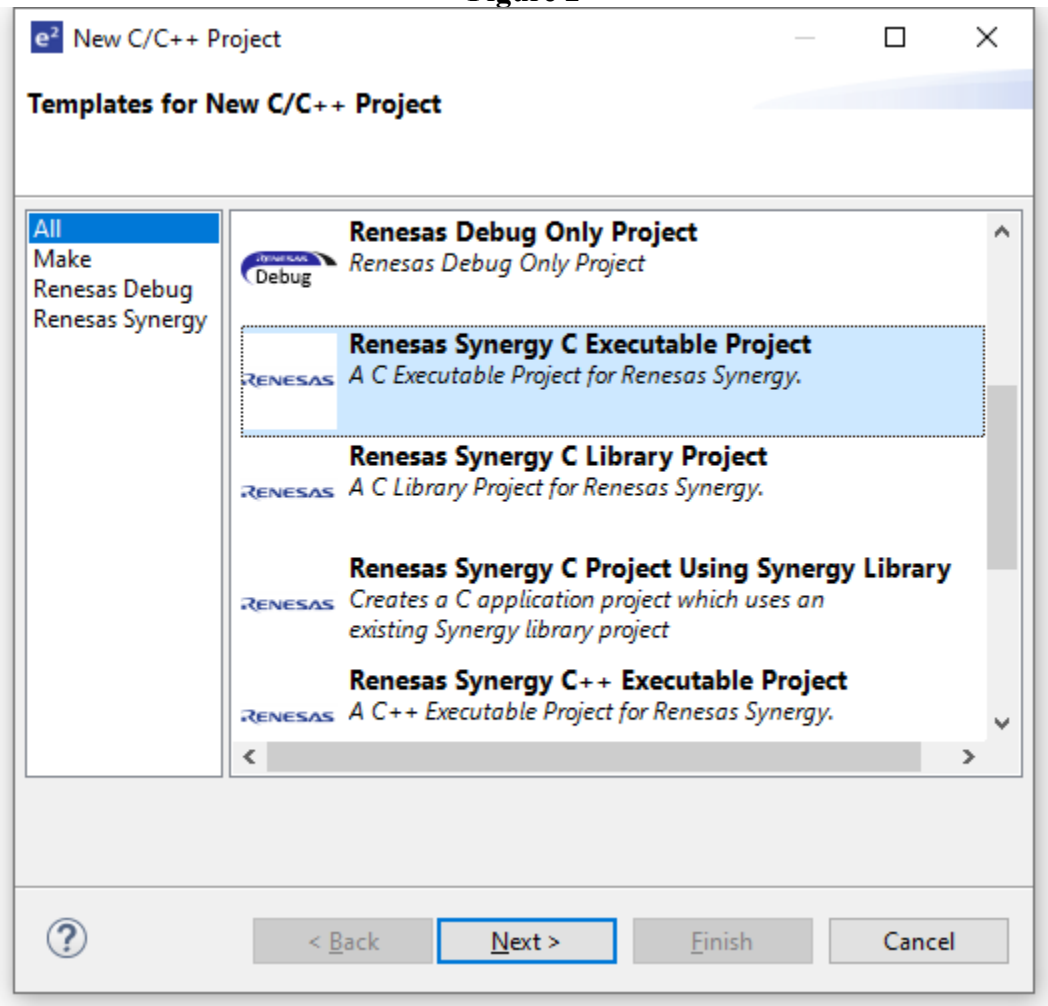
Figure 1



Project Creation and Setup

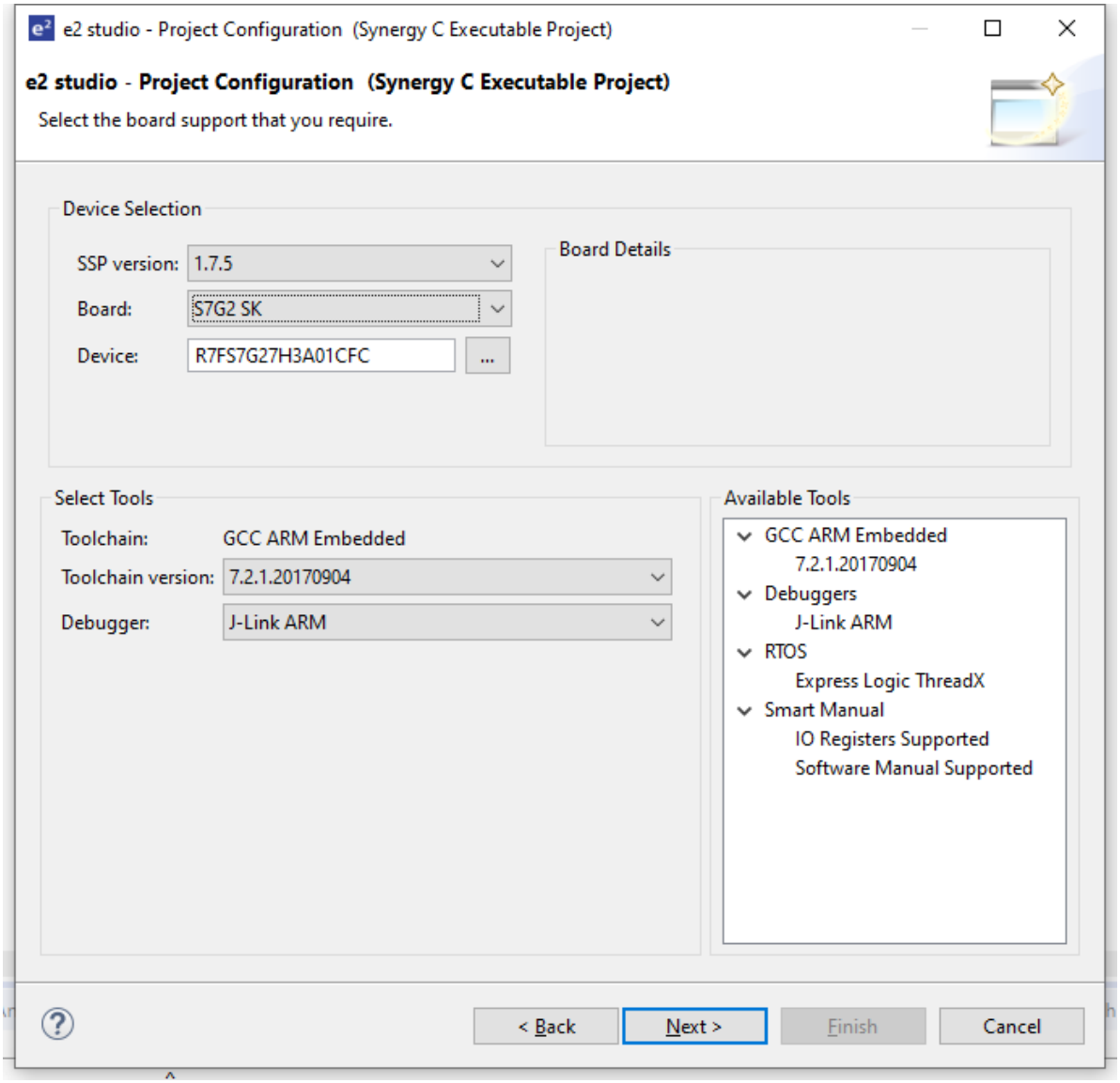
1. Create a Renesas Synergy C Executable project. Click Next, and give it a name. Next…

Figure 2



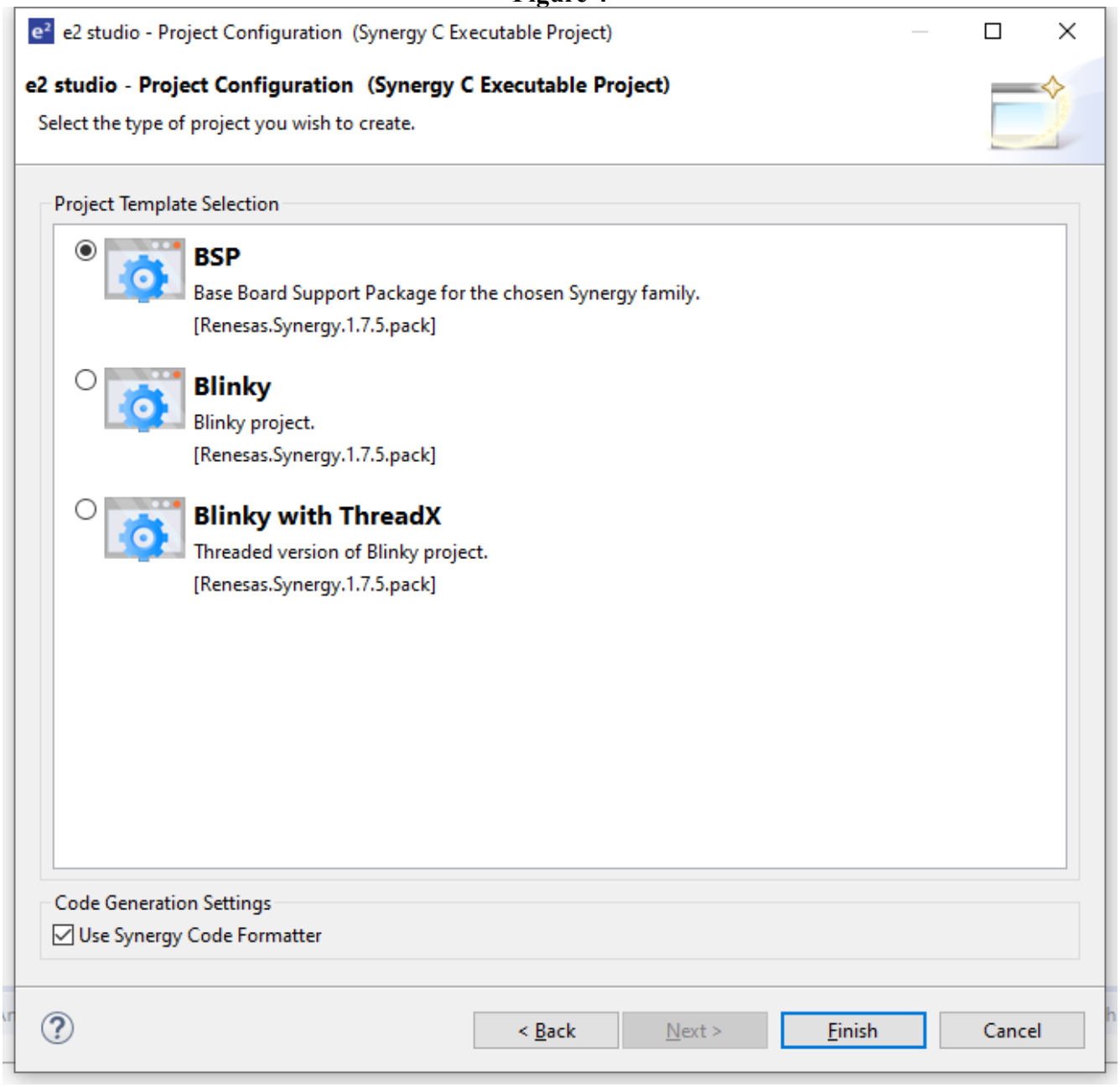
2. Select the S7G2 SK board. Next...

Figure 3



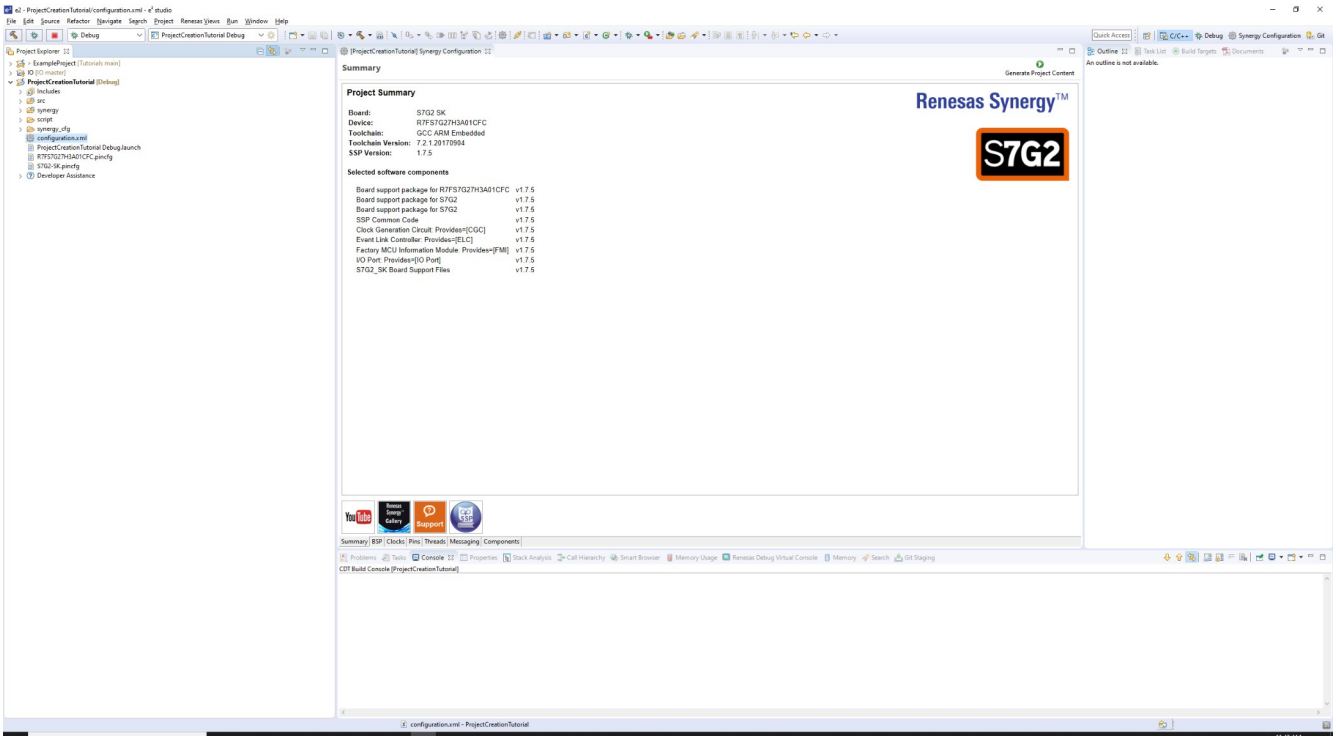
3. Select the BSP Template. Finish...

Figure 4



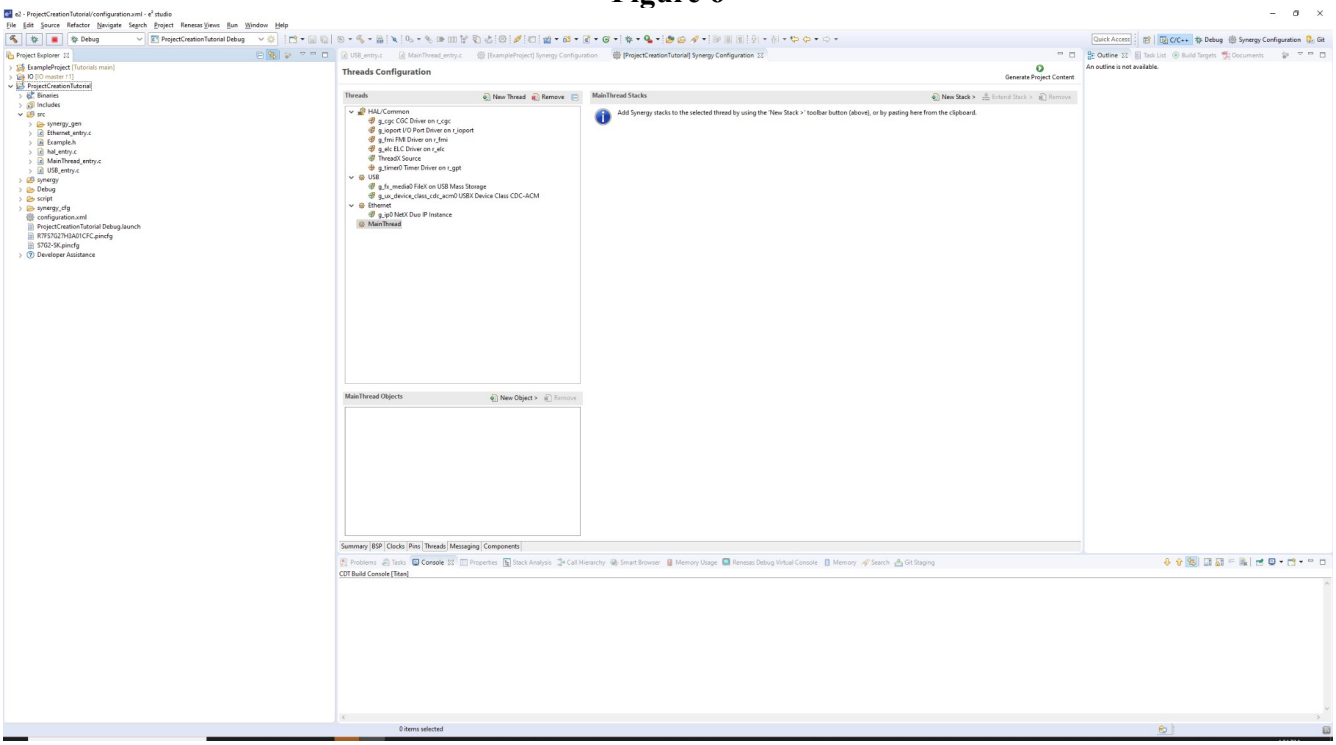
4. Your IDE should look similar to the figure below:

Figure 5



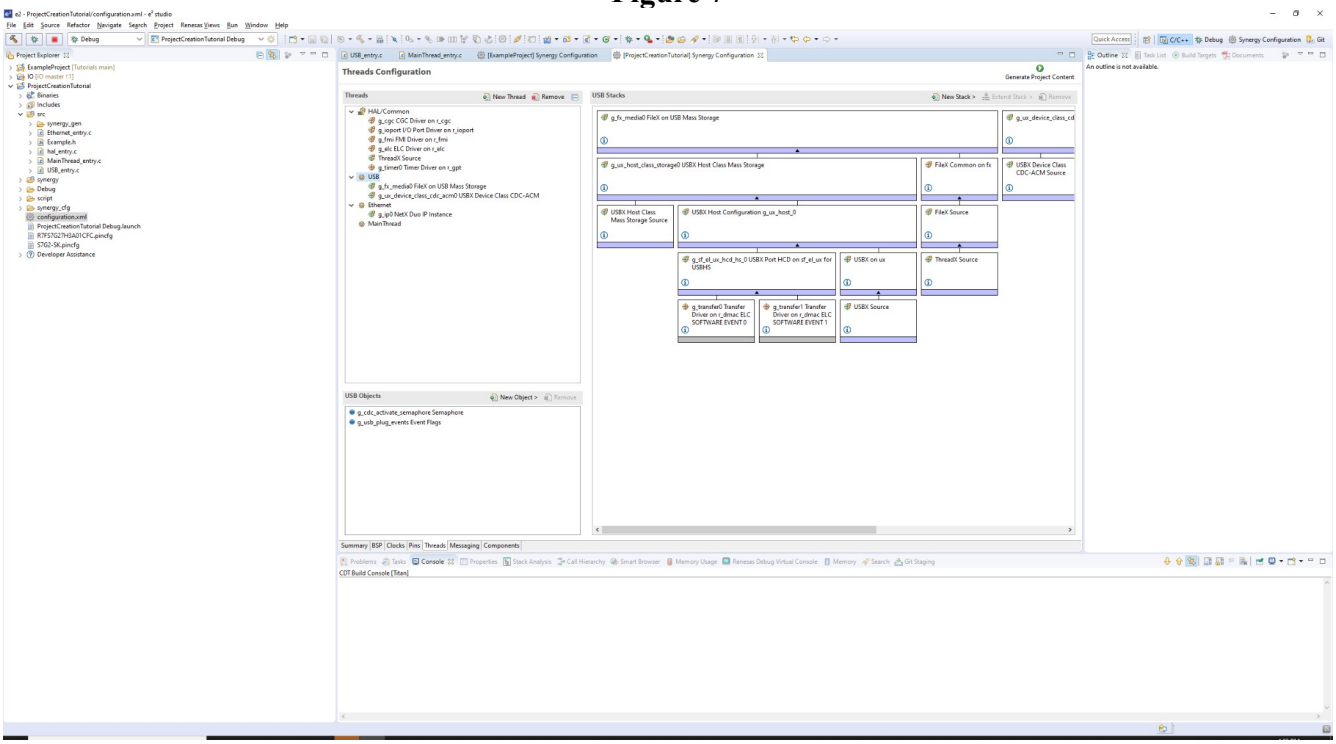
5. E2 studio will open the configuration.xml file by default. Close this file, and copy the configuration.xml file from the root directory of the ExampleProject folder into the root directory of your new project, overwriting the existing one. Next, copy all the files located in the “src” directory of ExampleProject in the same directory of your new project. Then, re-open the configuration.xml file and go to the Threads tab. You should see the following:

Figure 6



5. Now, you will notice that there are Ethernet and USB threads. Each of the interfaces has a dedicated thread which includes setup code and a short example. If you click on the USB thread, you should see the following:

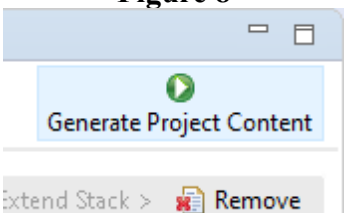
Figure 7



The items shown here are the thread stack elements which are added to a thread and configured to produce the required headers and API components needed to use the interface.

7. At this point what you do will depend on your project requirements. Every interface is given a dedicated thread. In the case of USB Host and Device, these are combined into a single thread. If your project does not require the use of every interface, you can free up resources by deleting the threads or thread stacks associated with these. Alternatively, you can leave everything as it is. For most applications, the additional unused interfaces will make no noticeable difference. The details of these thread stack settings will be left for other tutorials, or for the reader to review on their own.
8. Once you are finished adjusting the configuration.xml, click the Generate Project Content button in the upper-right corner.

Figure 8



Next we will add the libraries and adjust appropriate project settings.

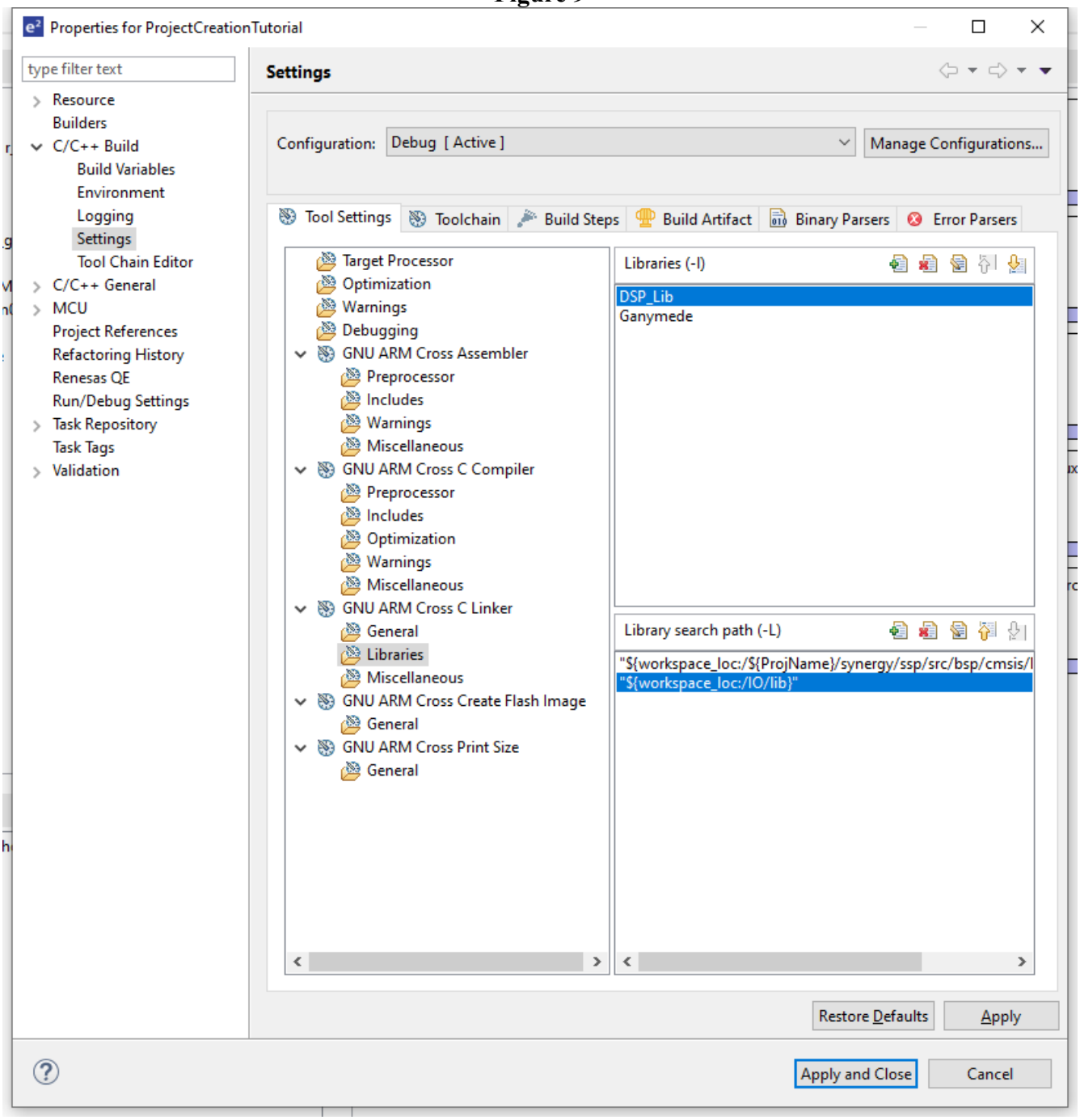
Adding the Ganymede Library and Headers

The Ganymede library contains a variety of functions necessary for the operation of each interface. These include functions for setup and initialization, callback functions, communications, etc... In the case of our USB Device interface the peripheral requires callback functions in order to properly interact with another device, like a PC. When the MCU is connected to a PC with Windows 10, for example, Windows should automatically recognize it as a USB Serial Device, install the correct drivers, and create a COM port device which can be used for communication. The callback functions facilitate this.

To include the Ganymede library into the project, follow the following steps:

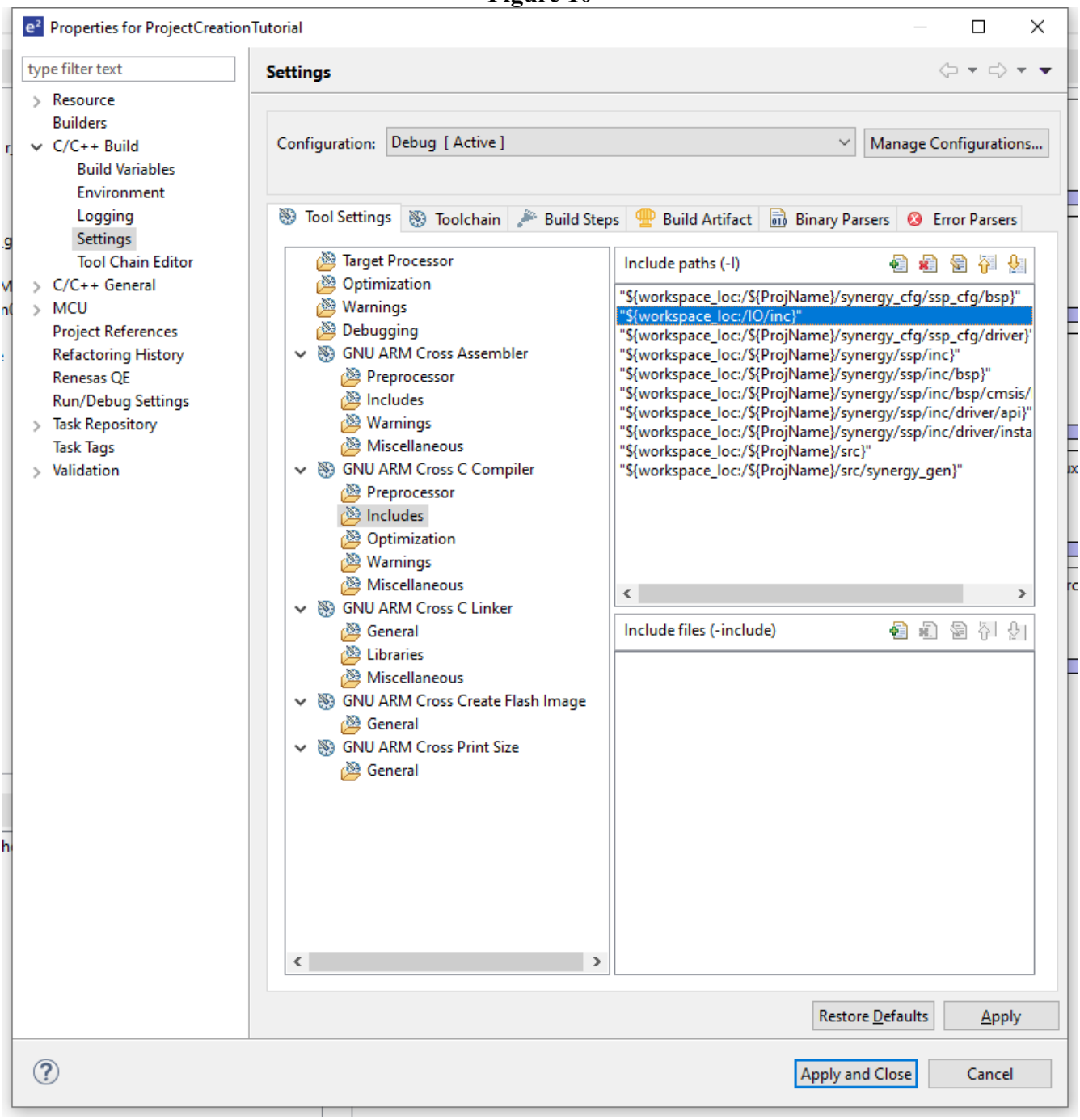
1. Right-click on the project, and go to Properties → C/C++ Build → Settings → GNU Arm Cross C Linker → Libraries. Add a new library named “Ganymede”. Next, add the “IO/lib” workspace directory to the list of search paths. The following figure is what should be seen after this:

Figure 9



2. Go to the GNU ARM Cross C Compiler → Includes, and add the “IO/inc” workspace directory, as shown in the following figure:

Figure 10

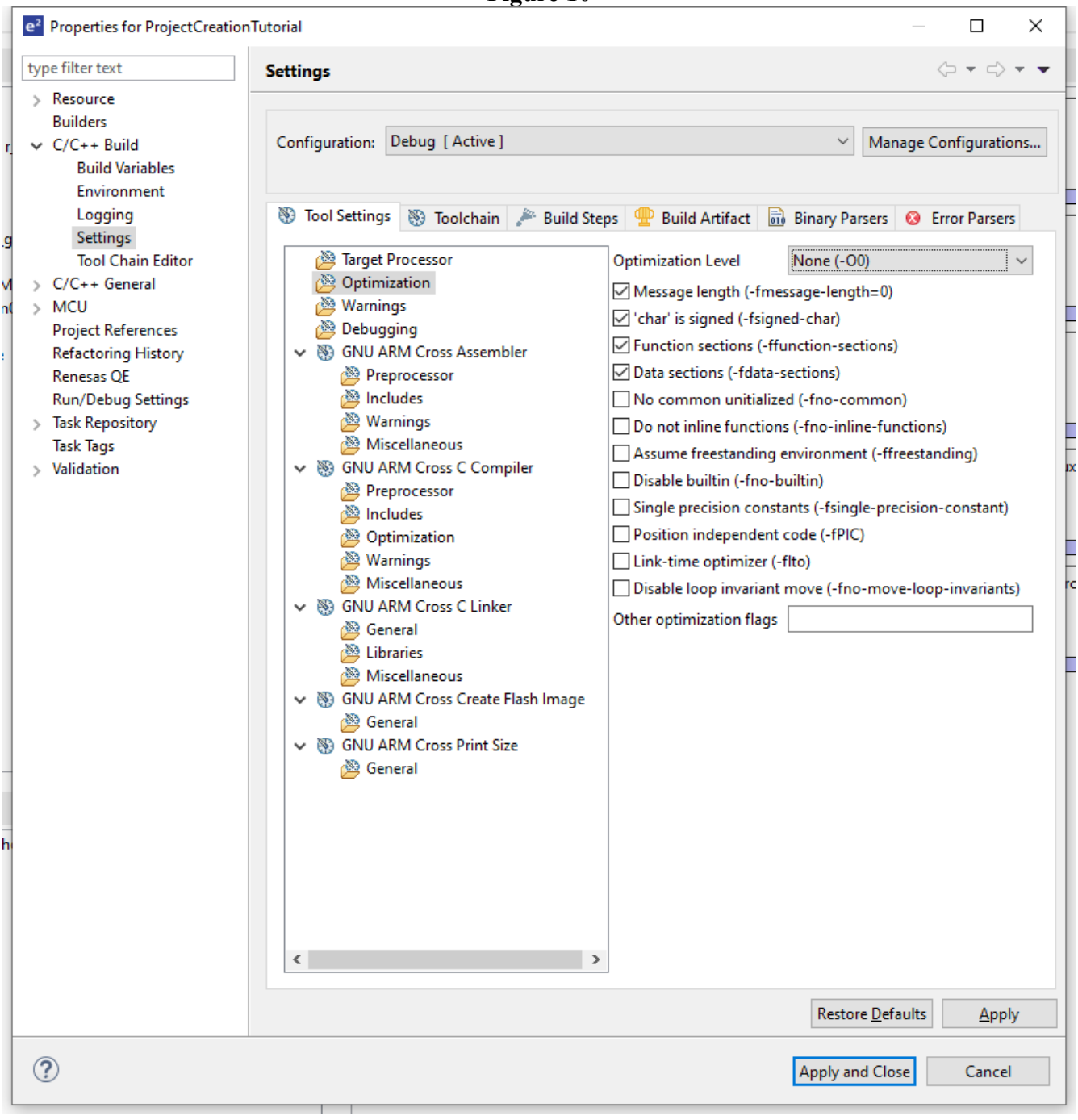


This is all that's required to make the Ganymede libraries available to your compiler. Next, there are some additional project settings that must be reviewed and adjusted.

1. Situations have been encountered where the Optimization setting has had an affect on the controller's functioning. This should not be the case, generally, but during debugging periods it has been found on some occasions that changing the Optimization level to None has solved a problem, so it's recommended that developers keep this set to None during debugging stages. Aside from potential crashes, this also allows the developer to monitor variables that would

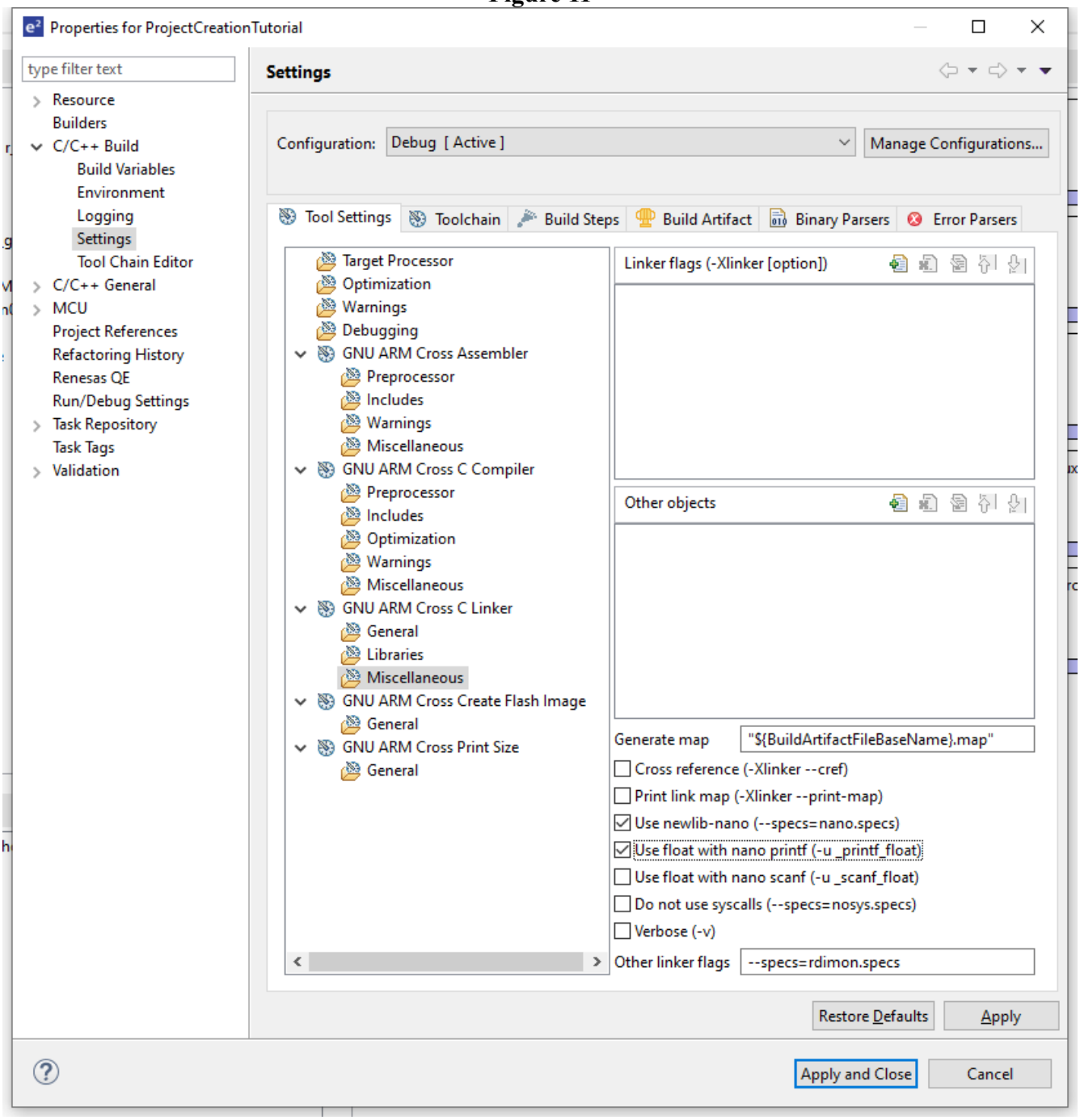
otherwise be optimized out. Optimization can be turned back on prior to production in order to reduce the code size, if desired. To change the optimization setting, go to the Optimization view and select None from the dropdown, as shown:

Figure 10



2. If you would like to include floating point numbers in printf outputs, you will need to enable the appropriate library, which is disabled by default. Go to GNU ARM Cross C Linker → Miscellaneous and check the box labeled “use float with nano printf” as shown:

Figure 11

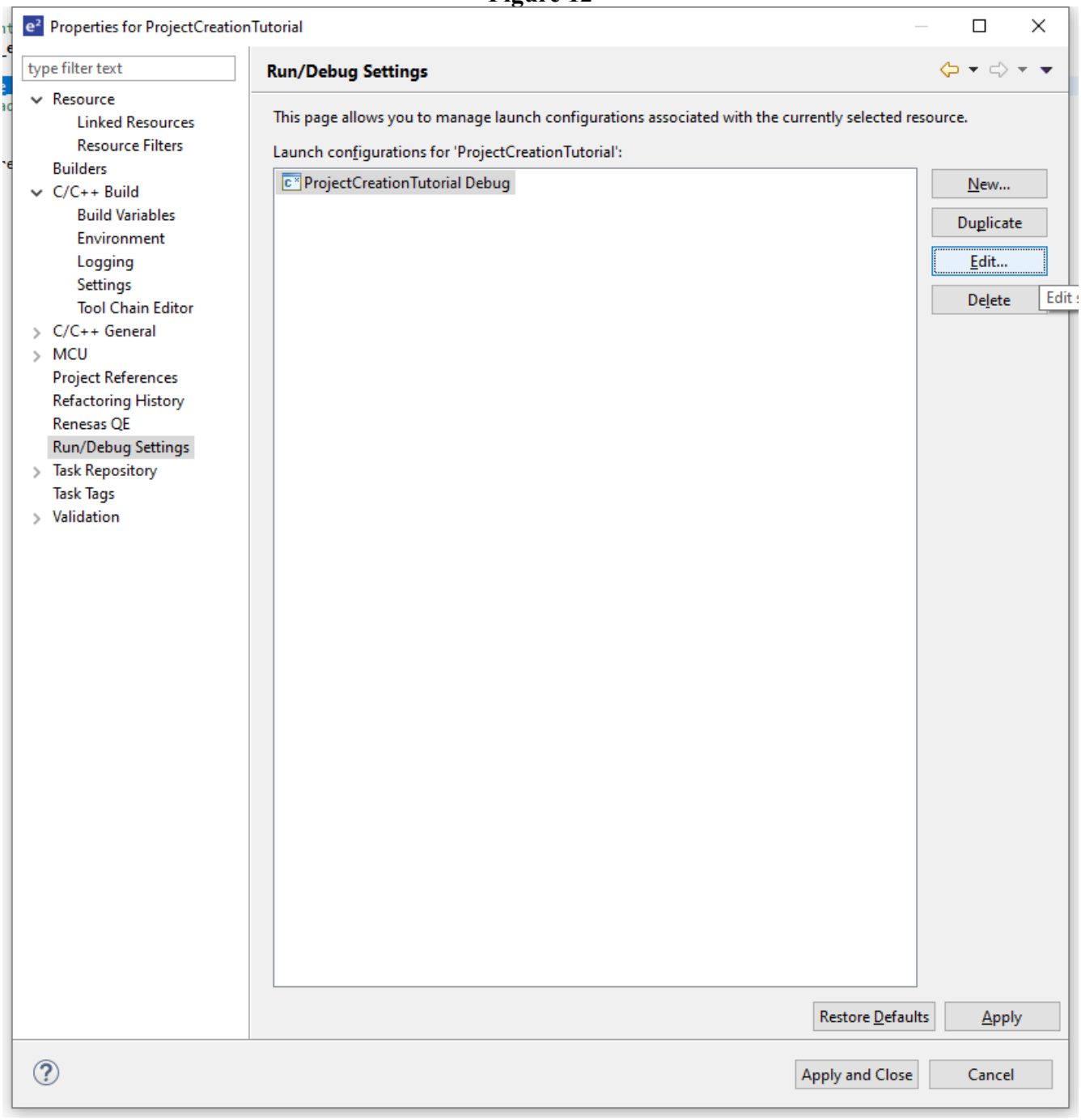


3. If you would like to have the MCU auto-run your program when it is powered on, there are some considerations to keep in mind. On the same view, you will see in the “Other linker flags” field, the line “--specs=rdimon.specs”. This is related to semi-hosting. If the MCU is programmed to output printf data, this will cause a crash if not connected to the debugger while semi-hosting is enabled. Removing this line will disable semi-hosting, and allow the MCU to safely auto-run when powered on. During debugging stages, this linker flag setting should be left as-is to allow semi-hosting and printf output to the console.
4. In addition to the linker flag setting, a call to "initialize_monitor_handles()” must be run prior to any printf statements in order to enable output to the console. Failure to do this will likely result

in a crash, with an error related to "_swi" reported. Along with removing the previously mentioned linker flag, the call to initialize_monitor_handles must also be removed to allow the MCU to auto-run.

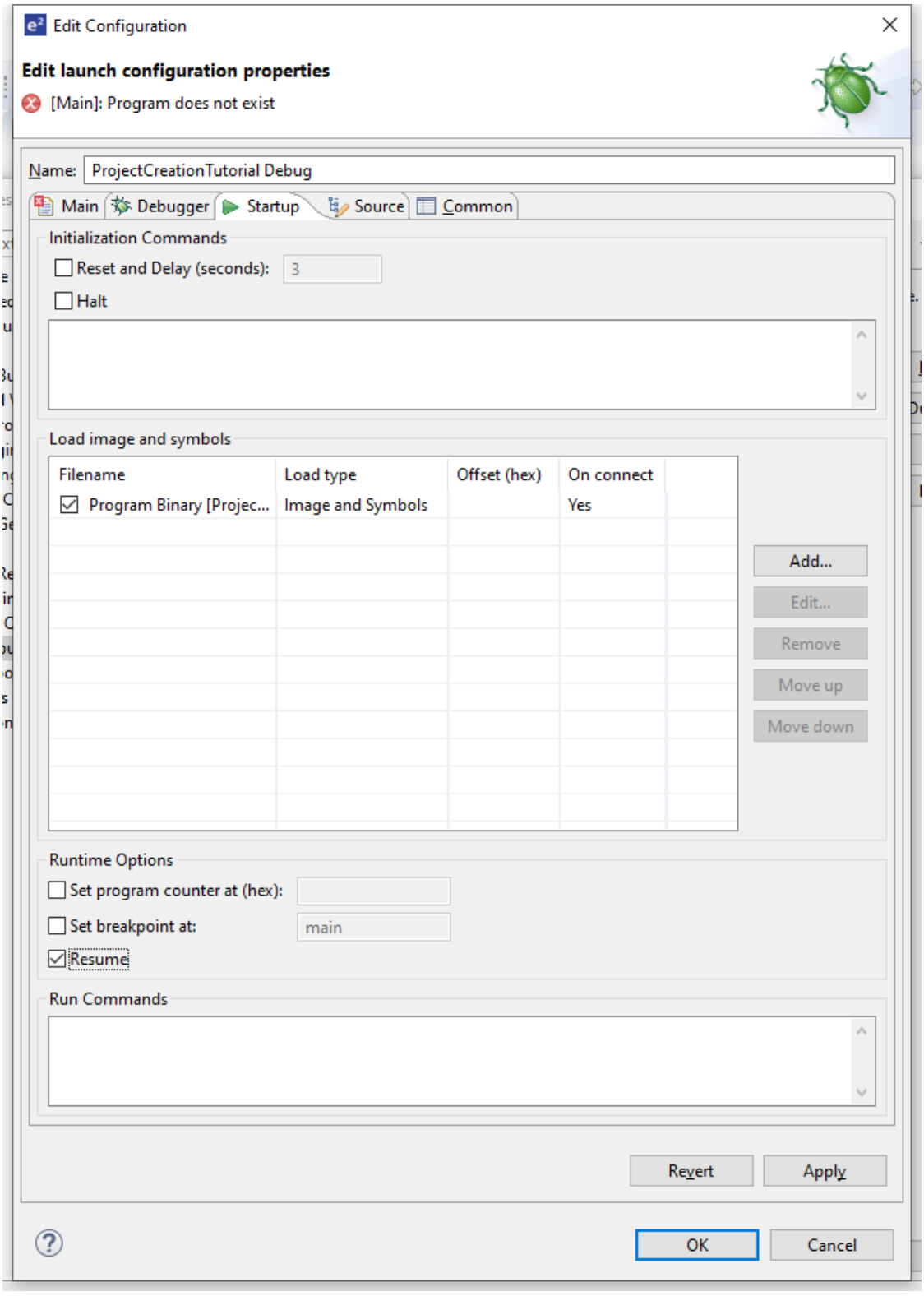
5. Finally, to allow the MCU to auto-run, there are two more additional project settings which must be modified. These are located in the Debug launch configuration. Go to Run/Debug Settings and edit the debug configuration settings, as shown:

Figure 12



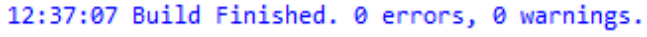
6. Go to the Startup tab and uncheck "set breakpoint at" and check "resume" as shown below:

Figure 13


Edit Configuration
Edit launch configuration properties
[Main]: Program does not exist
Name: ProjectCreationTutorial Debug
Main
Debugger
Startup
Source
Common
Initialization Commands
Reset and Delay (seconds): 3
Halt
Load image and symbols
Filename
Load type
Offset (hex)
On connect
Program Binary [Projec...
Image and Symbols
Yes
Add...
Edit...
Remove
Move up
Move down
Runtime Options
Set program counter at (hex):
Set breakpoint at: main
Resume
Run Commands
Revert
Apply
OK
Cancel

7. Click Ok and then Apply and Close. The project is now ready to be compiled. Go to Project → Build Project or use the Ctrl-B hotkey to build the project. After some time, the console view should report zero errors.

Figure 14



This is the end of the Project Creation Tutorial. Please refer to the rest of the tutorials in the Documents folder to learn about the available interfaces and library functions.