

## Scope

---

This document contains the user guide of Integrated Development Environment for CC100/150/200 processor class. Installation requirements and basic usage are described. The document covers project creation, compilation and debugging with build-in simulator as well as the FPGA or ASIC hardware.

# Contents

<b>1. Installation</b>	<b>3</b>
1.1 Download	3
1.2 Requirements	4
1.2.1 Operating System	4
1.2.2 Java 8	4
1.2.3 Python	5
1.2.4 Python pyserial module	5
<b>2. Using CCIDE</b>	<b>6</b>
2.1 Starting and closing IDE	7
2.2 Creating a project	9
2.3 Building a project	12
2.3.1 Perspectives	13
2.4 Project properties	14
2.5 Debugging	18
2.6 IDE preferences	22
<b>3. Troubleshooting CCIDE</b>	<b>23</b>
<b>4. Revision History</b>	<b>24</b>



# 1. Installation

## 1.1 Download

CCIDE is free and available at <https://github.com/chipcraft-ic>. It contains software development kit (CCSDK) with cycle-accurate CC150-S processor model and graphical user interface based on Eclipse CDT (Eclipse for C++ Developers).



## 1.2 Requirements

### 1.2.1 Operating System

CCIDE is distributed for Linux (64-bit CCIDE version) and for Windows (32-bit CCIDE version).

Supported Linux distributions: CentOS 6, Ubuntu 18.04.

Supported Windows versions: 7, 8, 8.1, 10.

CCIDE should work on different versions of Linux and Windows, however ChipCraft company can not guarantee that.

### 1.2.2 Java 8

CCIDE requires Java 8 runtime environment.

On Windows you can install 32-bit Oracle JRE manually after downloading it from Oracle website:

<https://www.oracle.com/technetwork/java/javase/downloads/jre8-downloads-2133155.html>

On Linux you can install OpenJDK 8 using your favorite package manager. In Ubuntu 18.04 run in terminal:

```
sudo apt-get install openjdk-8-jre
```

Please make sure you use version 8 of JRE software. CCIDE can work on newer versions however ChipCraft company can not guarantee that. To verify version of JRE run:

```
java -version
```



## 1.2.3 Python

CCIDE uses Python module for debugging programs running in ChipCraft devices or in a simulator. It is recommended to use latest version of Python 2.7 (2.7.15 as time of writing).

On Windows you can install it manually after downloading from official Python website:

<https://www.python.org/downloads/>

On Linux install it using your favorite package manager. On Ubuntu 18.04:

```
sudo apt-get install python2
```

## 1.2.4 Python pyserial module

CCIDE uses pyserial Python module to connect with device through serial port. Install it using your favorite Python package manager, e.g.

```
pip install pyserial
```

If pip command cannot be found try:

```
python2 -m pip install pyserial
```



## 2. Using CCIDE

CCIDE is an integrated development environment provided by ChipCraft for developing software for hardware produced by the company. It is based on Eclipse CDT. Basic usage and concepts are the same. This manual is describing custom elements added over Eclipse CDT and basic features important for embedded software development for ChipCraft's products.

Eclipse CDT documentation can be found on: <https://www.eclipse.org/cdt/documentation.php>.



## 2.1 Starting and closing IDE

To start CCIDE run `ccide.exe` (Windows) or `ccide` (Linux) executable file in `ide` subdirectory.

During the first start IDE will ask for selecting a workspace directory as can be seen on figure 2.1. This directory is used as a default location for all new projects. Please select some empty directory outside of CCSDK. You can change it later using `File` menu. If you are going to work with this directory for a longer time it is recommended to check *Use this as the default and do not ask again* option.

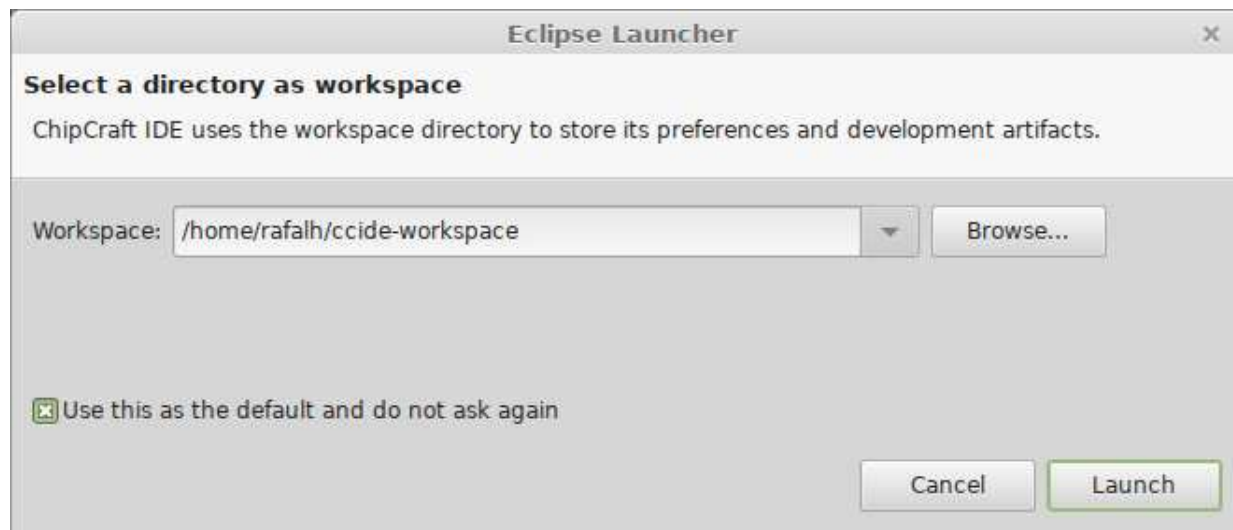


Figure 2.1. Workspace directory selection window.

After starting the IDE you should see empty IDE window (figure 2.2)

To close the IDE select `File` -> `Exit` from menu or click a cross symbol on the window titlebar. In the latter case you will be asked for confirmation (figure 2.3).



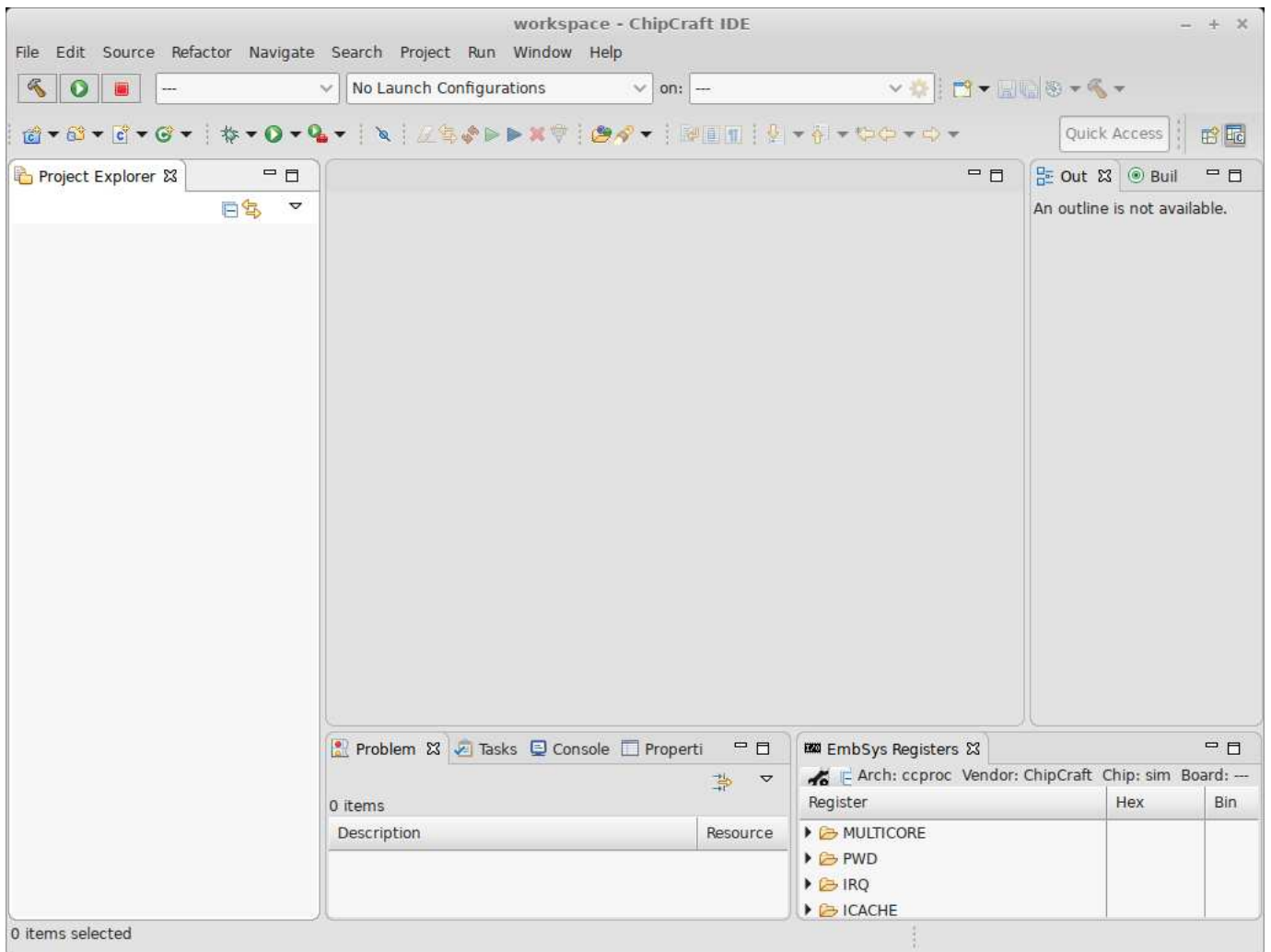


Figure 2.2. Empty CCIDE window.

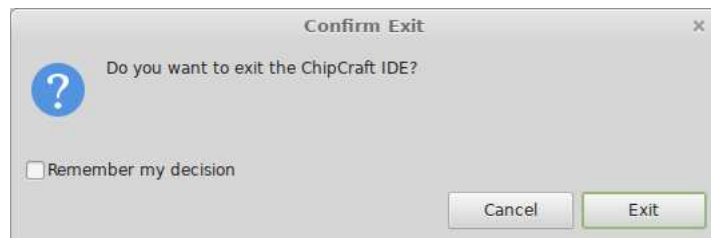


Figure 2.3. Confirmation on closing CCIDE.





## 2.2 Creating a project

To create a new project select `File -> New -> CCSDK Project` from main window menu (figure 2.4).

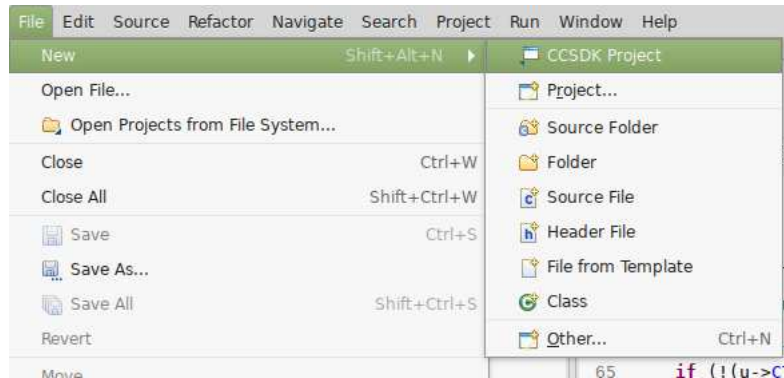


Figure 2.4. File -> New menu.

First page of New Project wizard will show up (figure 2.5). Specify a project name. Project location is auto-generated using workspace path and project name. If you want you can override it by unselecting **Use default location** checkbox.

After filling the fields click **Next**.

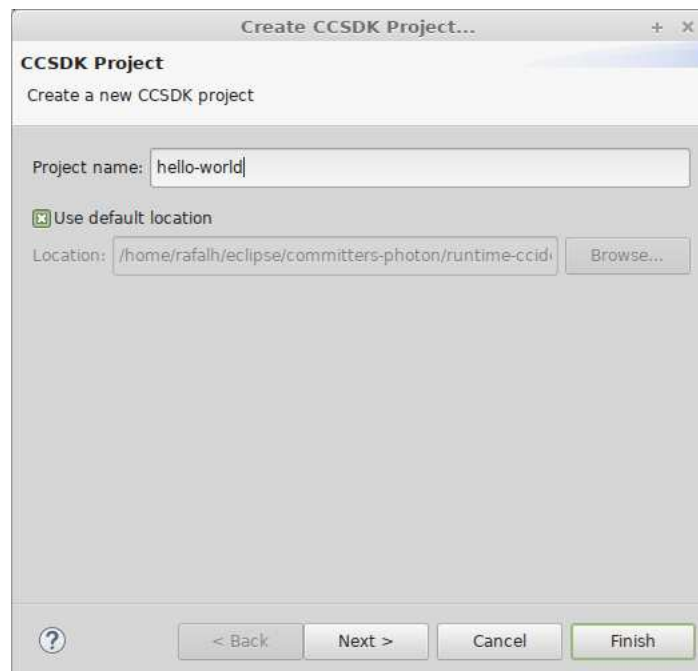


Figure 2.5. First page of New Project wizard.

In the second page of New Project wizard (figure 2.6) you can provide basic configuration of the project. Option specified on this page are persisted so next wizard invocation will restore all lastly used values.

**CCSDK Location** field allows customization of CCSDK location. By default parent directory of CCIDE directory is used.



Target board and connection options are also configured in this page. If you have a hardware ASIC or FPGA board please select its name in **Board** combo-box. If you have no dedicated hardware or would like to use build-in simulator please select special board named *sim*.

For physical board select proper debug and UART port. The first one is used for sending debugger commands and receiving results. The second one is used as STDIO stream. On Linux ports usually follow a pattern: `/dev/ttyUSBn`. On Windows ports follow a different pattern: `COMn`. CCIDE will show all connected ports as combo-box suggestions.

**C++ Support** option makes the project C++ compatible. Enable it if you are considering usage of C++ source files.

**Built Type** allows to specify how build process looks. Select **Managed Build** if you want CCIDE to always generate Makefiles from configuration provided in graphical user interface. If you want to have control over Makefiles select **Makefile Build** option.

After filling all fields click **Next**.

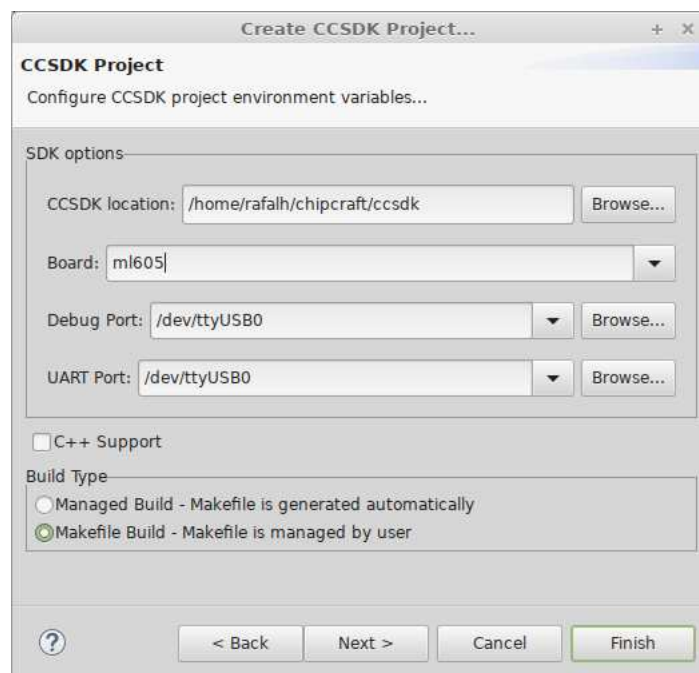


Figure 2.6. Second page of New Project wizard.

In the third page of New Project wizard (2.7) user can configure initial code generation. Select peripherals required by your project in a tree on the left part of the wizard window. Selecting a peripheral allows to customize its options on the right panel.

After configuring peripherals click **Finish**. New project is being generated.

After project is created IDE opens *main.c* file (figure 2.8). Peripherals selected in New Project wizard have initialization routines generated and called from *main()* function.



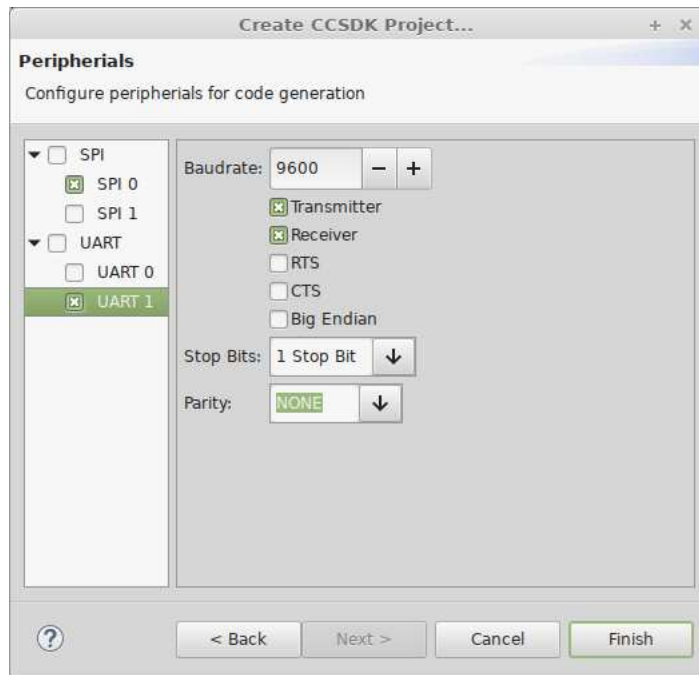


Figure 2.7. Third page of New Project wizard.

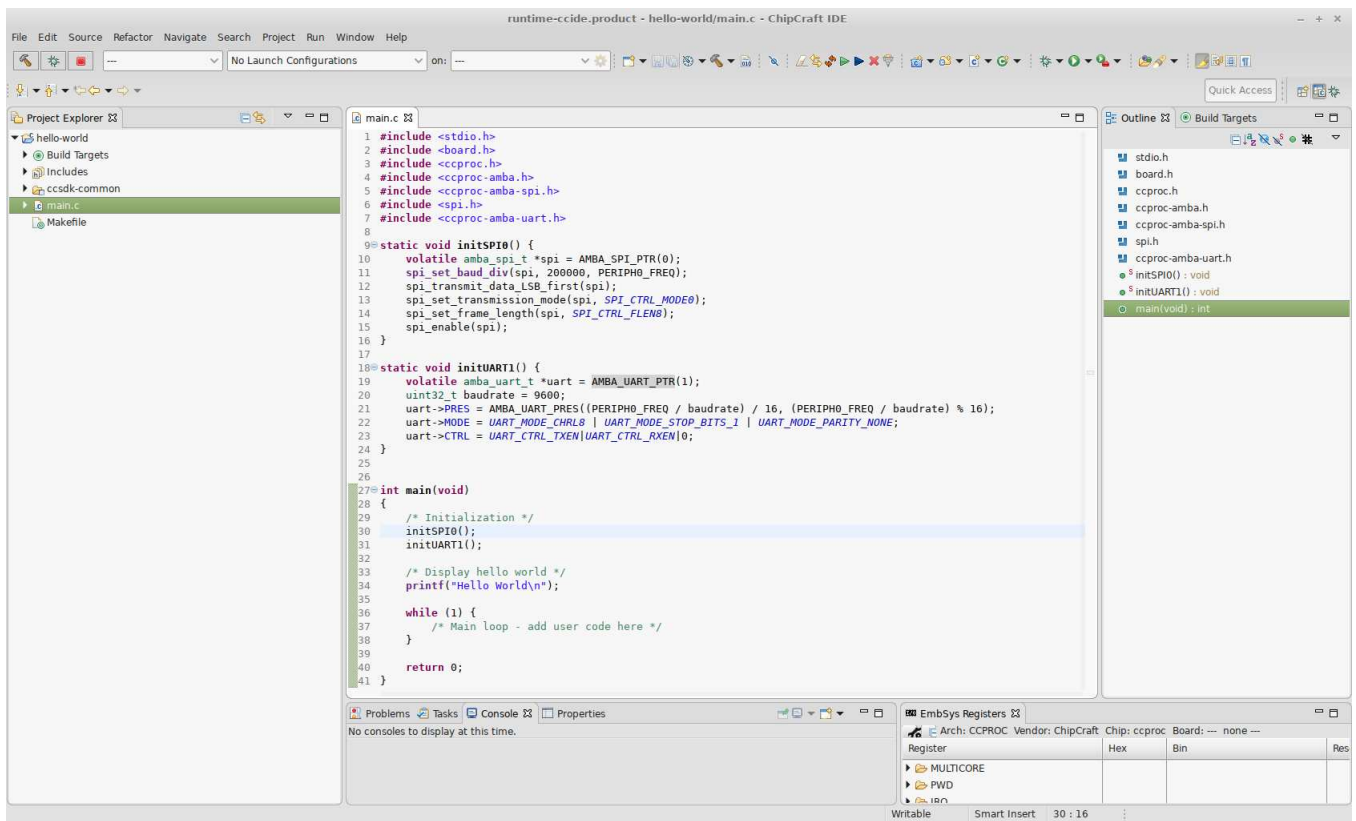



Figure 2.8. CCIDE main window after project creation.



## 2.3 Building a project

To build a project click on a build button . You can also select option `Project -> Build All` from menu.

CCIDE has dedicated toolbar for most used actions when working with CCSDK projects. It is visible on figure 2.9.

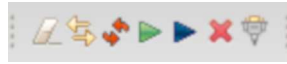


Figure 2.9. CCIDE Toolbar.

Toolbar has following buttons:

- Clean - cleans build artifacts,
- Build - builds the project,
- Rebuild - cleans and builds the project,
- Save in RAM,
- Save in Flash memory,
- Reset device,
- Serial terminal.

All toolbar buttons work in context of current project. It is determined from selected element in project tree or from file currently open in editor window (depending on input focus).

Project created by CCSDK Project wizard contains `ccsdk_common` virtual folder (visible on figure 2.10). All files in this folder are symbolic links to corresponding files in CCSDK. You should not modify them because all projects share them. If you need to make changes you should copy a file to your project and remove from `ccsdk_common` folder.

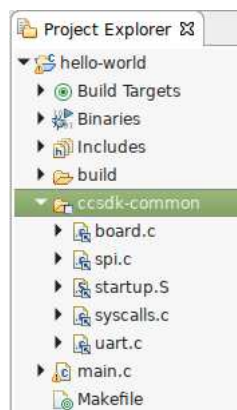


Figure 2.10. Linked source files.

Custom Makefile actions can be configured in CCIDE as Build Target items. They are visible in Project Explorer (figure 2.11) and a dedicated view (figure 2.12). By default new project wizard adds Build Targets for resetting and programming a device.



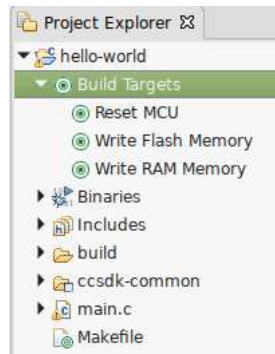


Figure 2.11. Build targets in project tree.

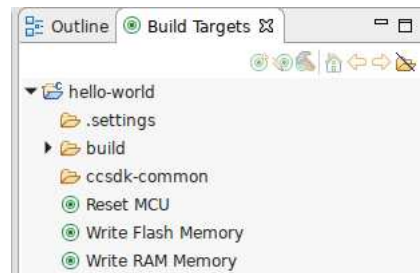


Figure 2.12. Build targets view.

### 2.3.1 Perspectives

CCIDE supports perspective concept. User can customize the main IDE window by adding views and changing their position. All such actions happen in context of a perspective. IDE can have multiple perspectives optimized for different tasks e.g. code development, debugging etc. By default CCIDE uses two perspectives: C/C++ and Debug. User can create more if needed. List of all visible views and their positions are persisted in perspective so when IDE is restarted they are in the same place.



## 2.4 Project properties

Project properties can be changed by clicking RMB<sup>1</sup> on the project in the Project Explorer view and selecting *Properties* from a context menu.

All project properties are grouped in categories visible in a tree control in the left side of the window. Categories can be filtered by providing a query text inside field above the category tree. To save changed properties and close the window click **Apply and Close** button at the bottom.

Options directly responsible for building the project are in *C/C++ Build* category.

**C/C++ Build -> Build Variables** page allows to set custom variables accessible from some CCIDE input fields. They can be used in a form of placeholders: `${xxx}`. They are useful if the same string is needed to be repeated multiple times in configuration. To determine if input field supports variables check if *Variables...* button is nearby.

**C/C++ Build -> Environment** page (figure 2.13) allows to set environment variables used during the build process. They are accessible from *Makefile* scope. New Project Wizard configures multiple environment variables starting with `CCSDK_` prefix. They should not be changed nor removed using this page. To change `CCSDK` options use *CCSDK Options* properties page.

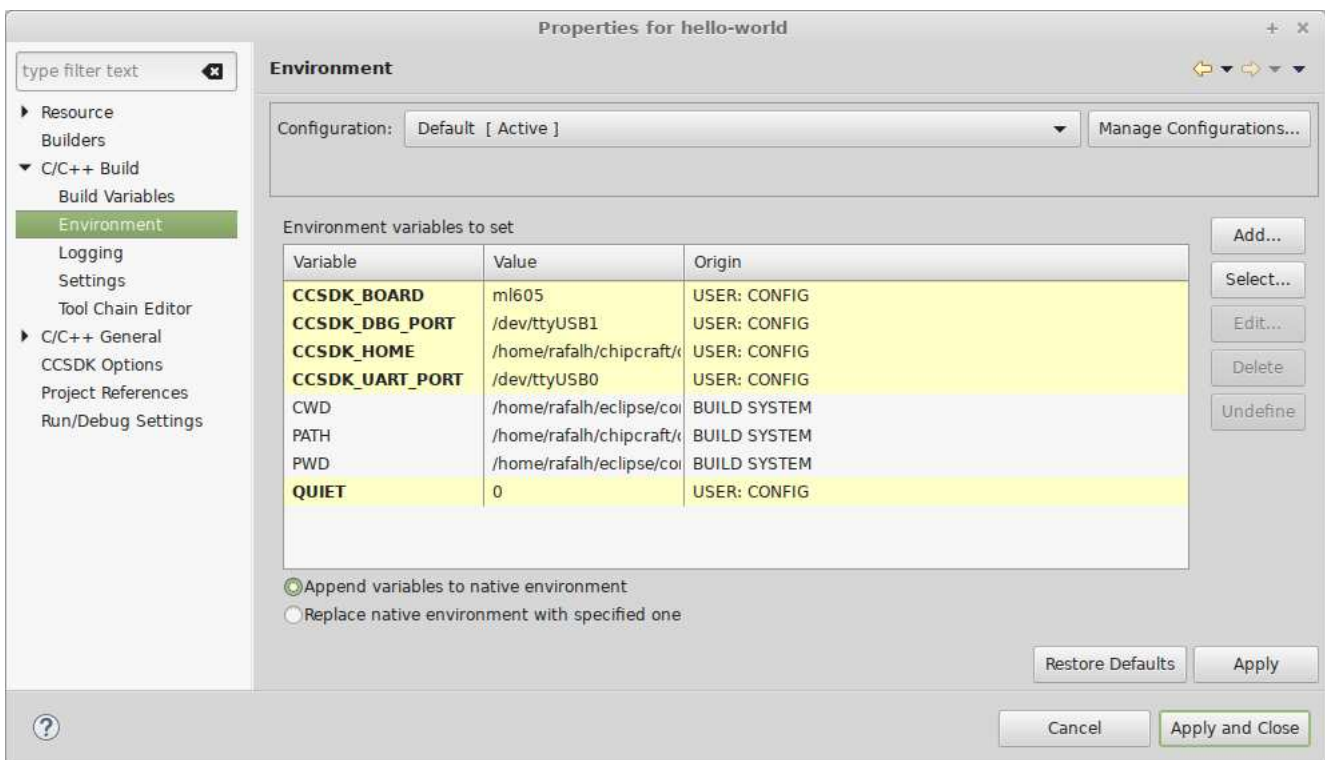


Figure 2.13. Environment variables configuration.

Toolchain options in a project of *Managed Build* type can be configured in **C/C++ Build -> Settings** page (figure 2.14). All options are configuration specific (they can affect Debug or Release project configuration).

**Tool settings** tab allows to set optimization level for C and C++ compiler, debugging flags, warnings and other flags for the compiler and linker. Include directories and preprocessor definitions can be configured too (it is recommended

<sup>1</sup> right mouse button



to use *C/C++ General -> Paths and Symbols* page for that). Every tool used by the project has separate configuration. To configure a tool find its name in tools tree (e.g. Cross GCC Compiler) and select one of options categories belonging to the tool.

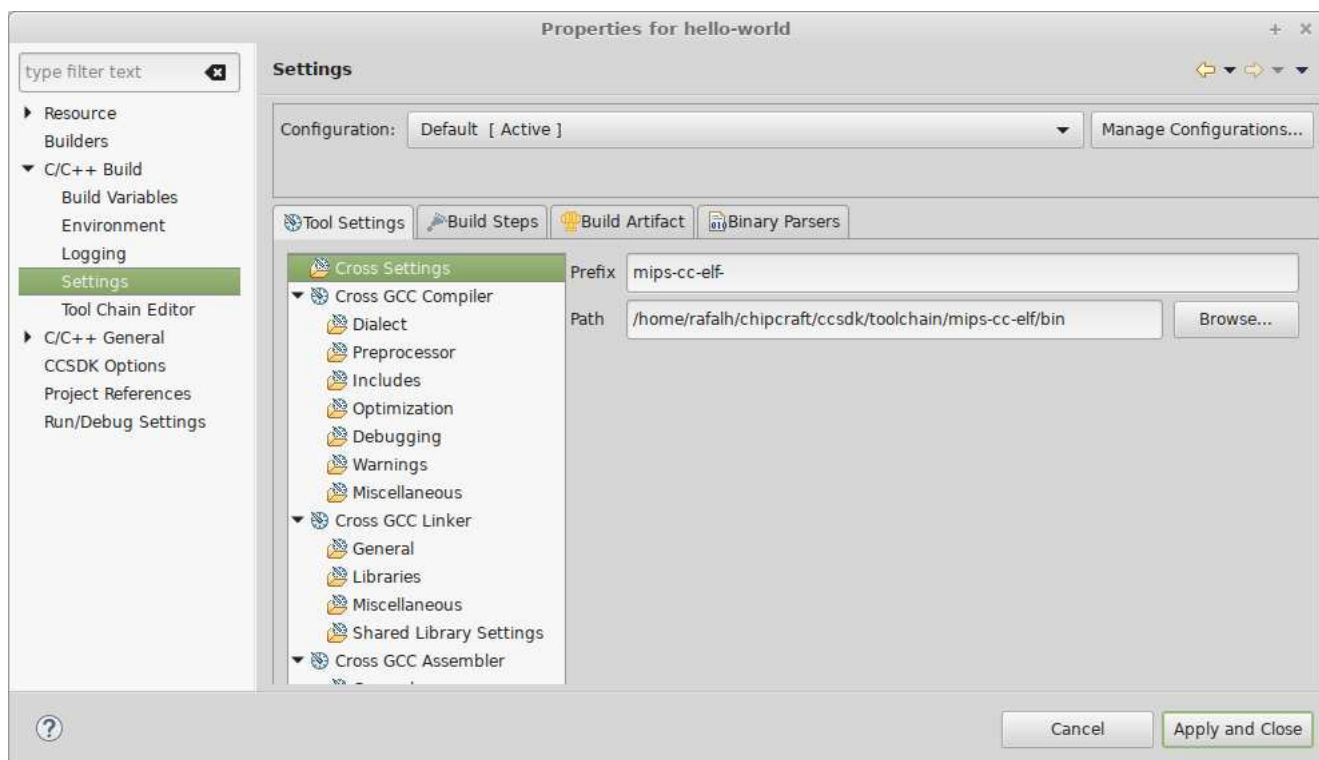


Figure 2.14. Toolchain settings.

**Build Steps** tab allows to configure additional commands to run before and after the build process (figure 2.15). Command running before build process is entered in *Pre-build steps* section. Commands running after build process are entered in *Post-build steps*. In Managed Build project CCIDE uses post-build step to print size of resulting binary and generate SREC file. You can add multiple commands in build steps by concatenating them using && operator.

In *Makefile* project *C/C++ Build -> Settings* page is simplified and contains only parsers configuration. User is supposed to provide compiler and linker flags hard-coded inside a *Makefile*.

**C/C++ General -> Paths and Symbols** page is used to configure include directories, C preprocessor definitions and libraries used by the linker.

**Includes** tab (figure 2.16) allows configuration of include paths. Paths are added independently for used programming languages (e.g. C, C++, Assembly). Environment and build variables can be used in path definition.

CCIDE adds multiple paths when generating the project (they start with `${CCSDK_HOME}`). They should never be deleted.

**Symbols** tab (figure 2.17) allows configuration of C preprocessor definitions. Similarly to *Includes* tab you can use environment and build variables during symbol definition. Definitions are independently defined for each language (C, C++).

CCIDE add multiple symbols when generating the project (e.g. `BOARD`, `BOARD_<board_name>`). Please do not delete them from configuration.





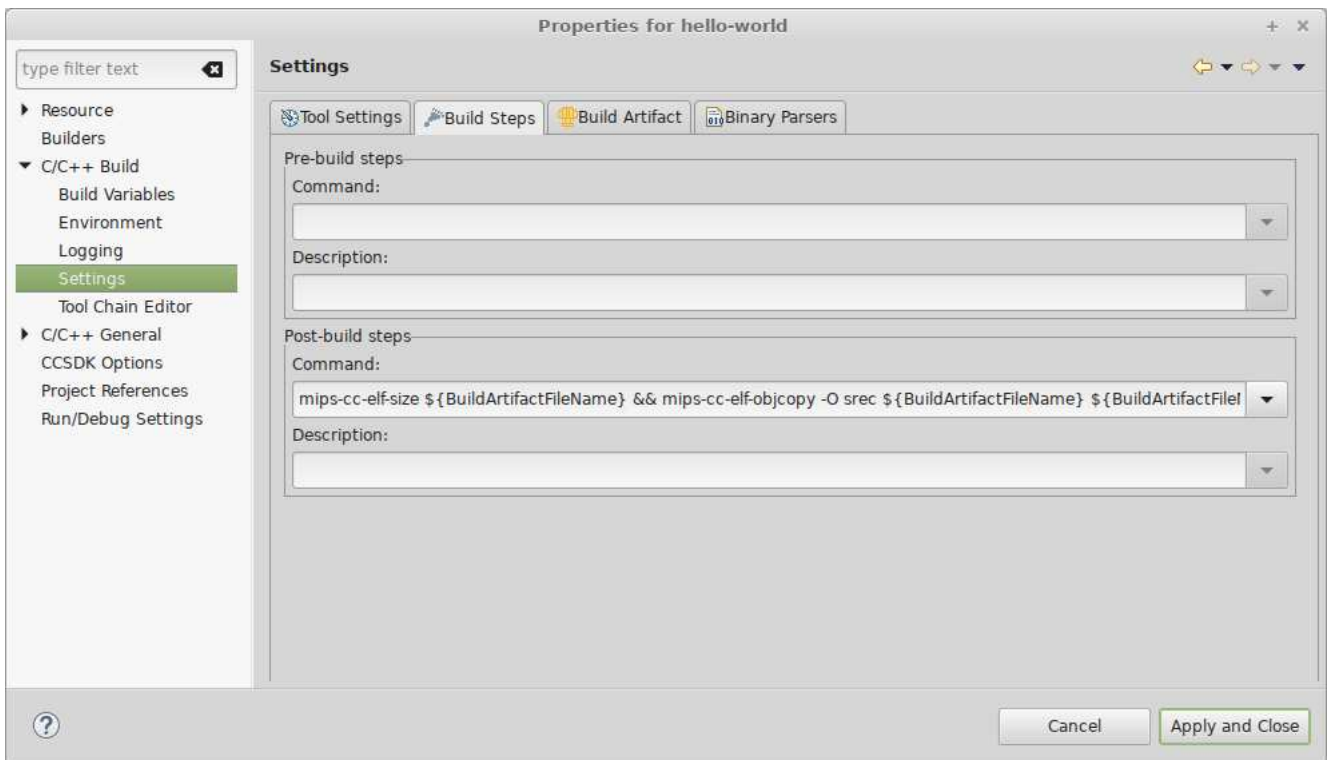


Figure 2.15. Build steps configuration.

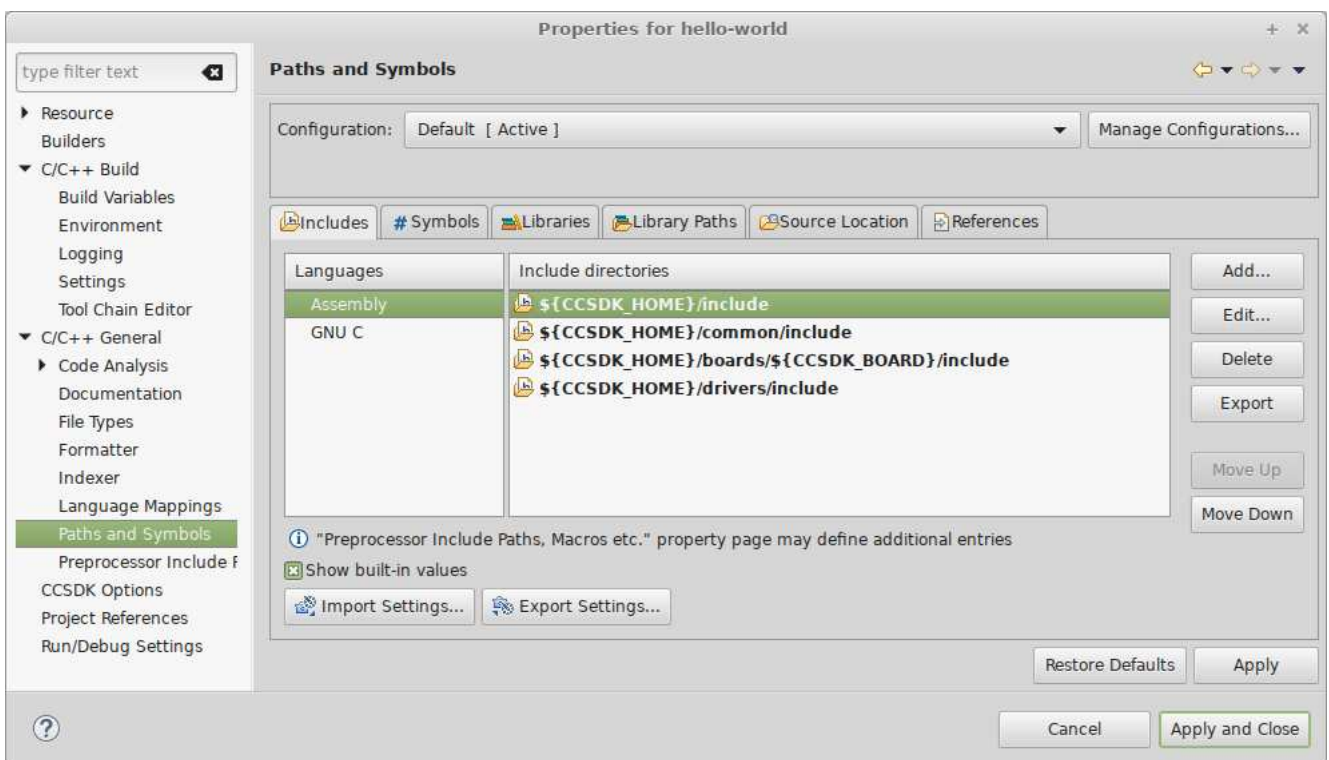


Figure 2.16. Include directories configuration.

CCSDK specific configuration can be found in **CCSDK Options** page (figure 2.18). It has the same options as you encounter during *New CCSDK Project* wizard. CCSDK home location, board, debug and UART (stdio) ports can be configured. It is useful if configuration provided during project generation needs to be changed.





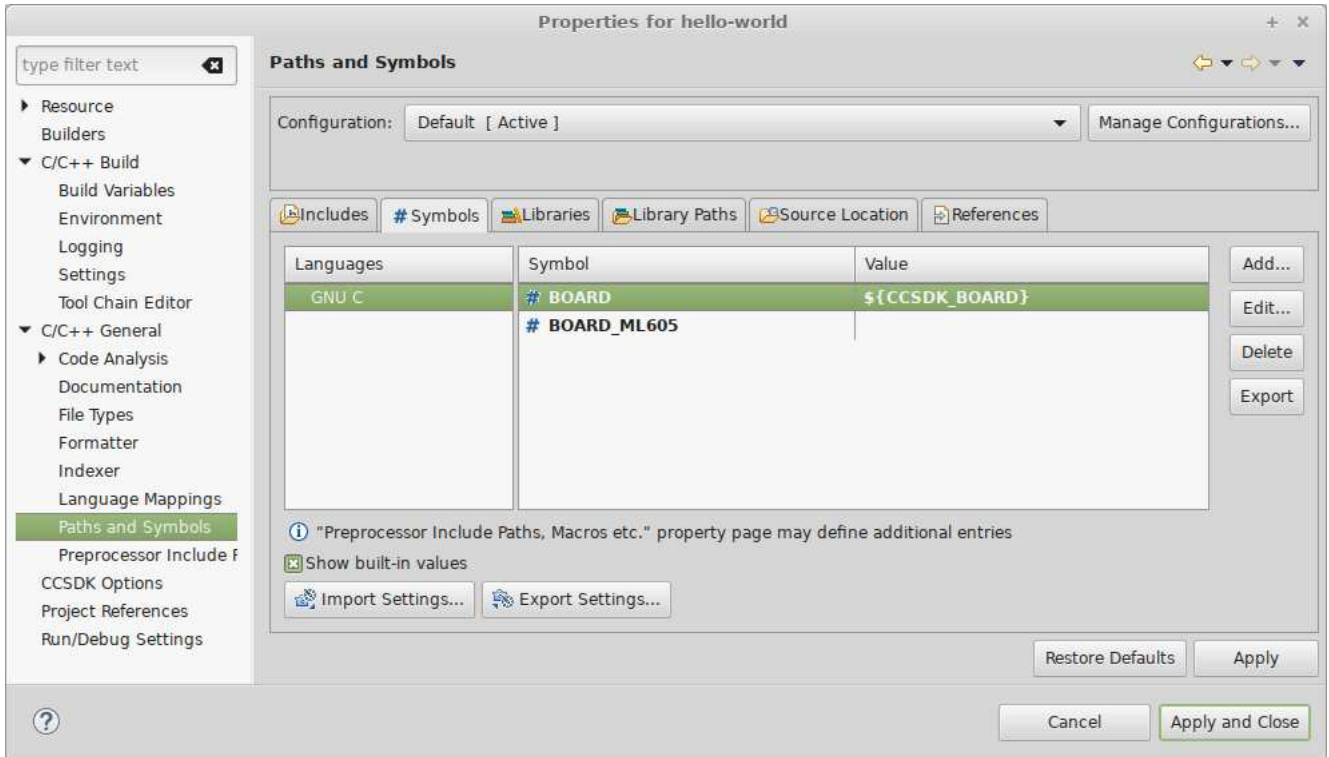


Figure 2.17. Preprocessor symbols configuration.

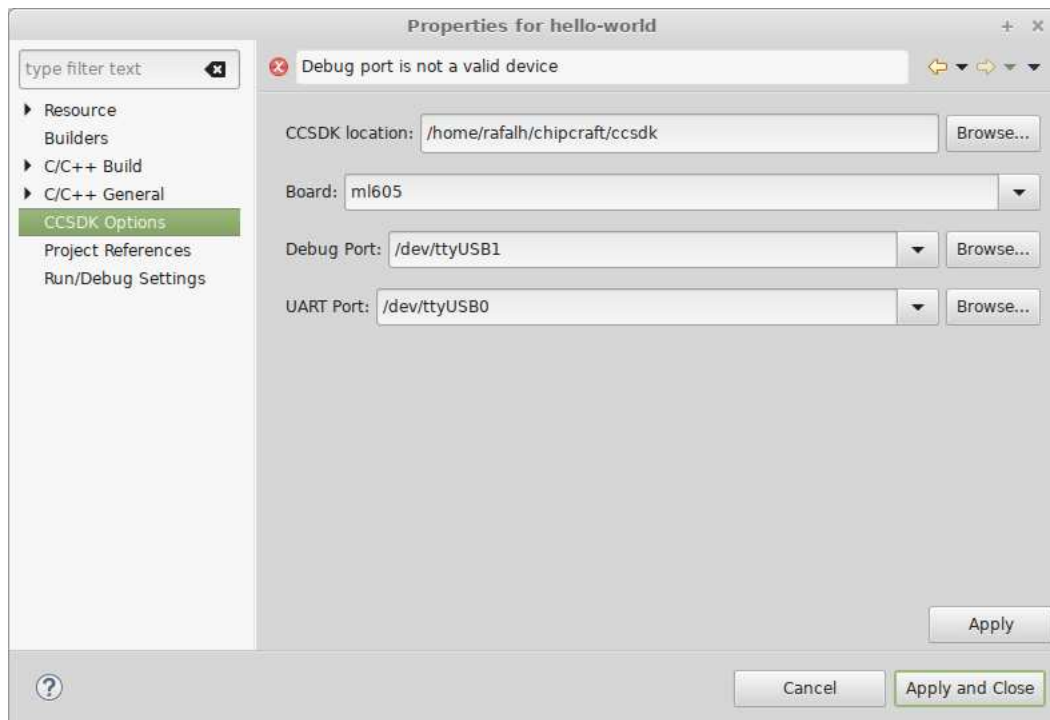


Figure 2.18. Project CCSDK specific properties.



## 2.5 Debugging

To debug an application in CCIDE launch configuration has to be created. There are two supported configuration types: *CCSDK Hardware* and *CCSDK Simulator*. The first one is supposed to be used with a dedicated ASIC or FPGA board connected to the computer with a USB cable. The second one uses simulator software included in CCSDK package.

Launch configuration can be run in one of two defined modes: **Run** (for quick running without attaching a debugger program) and **Debug** (full featured debug mode). Depending on launch mode different perspective is used.

Launch configuration can be created by clicking RMB on **Run** or **Debug** toolbar button and selecting *Debug Configurations...*. In next window (figure 2.19) all launch configuration for given launch mode are displayed. New launch configuration can be created by clicking RBM on configuration type in left part of the window and selecting *New*.

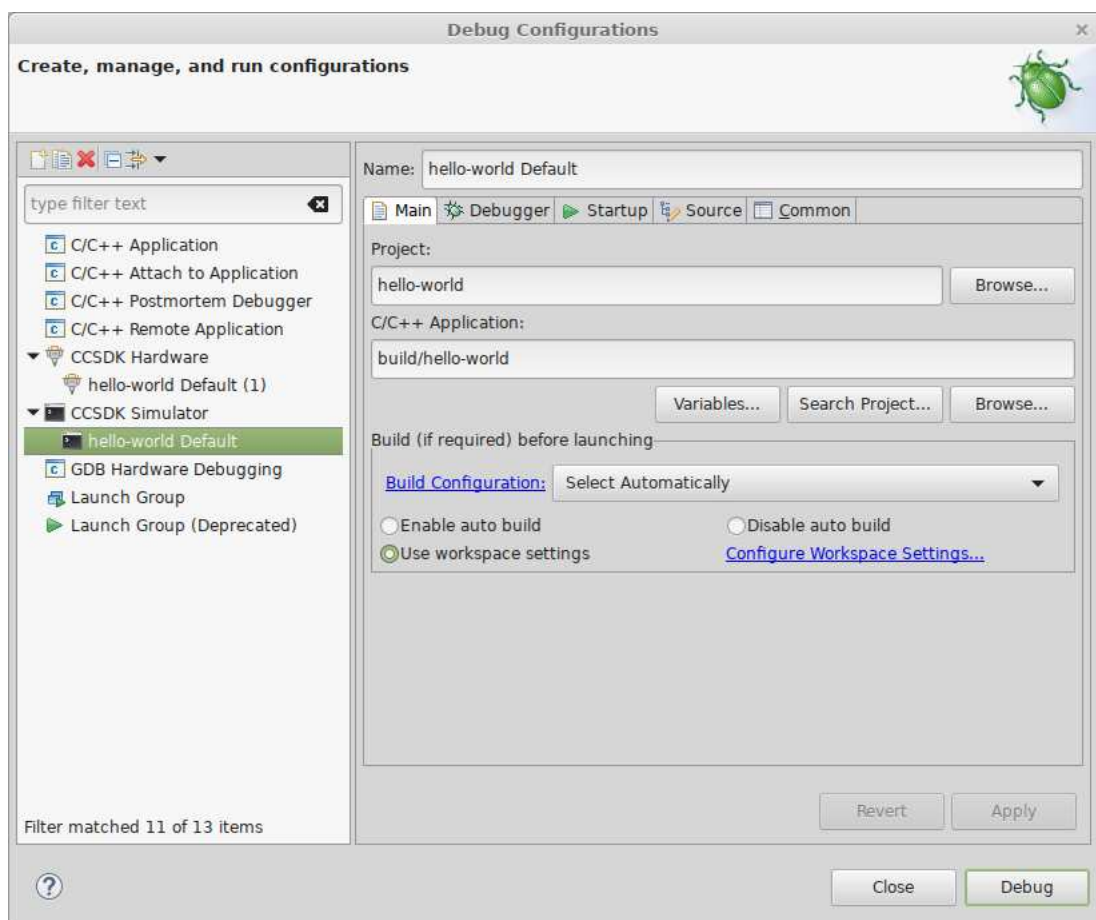


Figure 2.19. Launch configuration.

After launch configuration is created it can be executed by selecting **Debug/Run** button in launch configuration window or by selecting created launch configuration from context menu of **Debug/Run** buttons in the toolbar.

Launch configuration with default options can be started by using a shortcut. To do so click RMB on **Debug/Run** toolbar button, then highlight **Debug As/Run As** submenu and select suitable configuration type (figure 2.20). If any configuration of selected type already exists for current project it is started. If not a new configuration is created with default options and it is started instead.



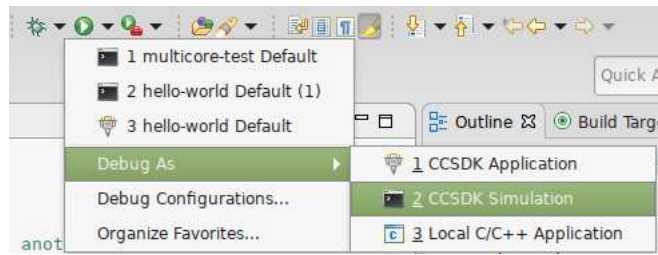


Figure 2.20. Debug launch shortcut.

It is important to not create launch configurations not corresponding to project board. If project uses *sim* board *CCSDK Simulator* configuration should be used. If project uses ASIC or FPGA board *CCSDK Hardware* configuration should be used. Binary files produced for simulator are different than binary files produced for other boards so both launch configurations cannot be used interchangeably.

After debug configuration is launched debug perspective is opened (figure 2.21). In **Debug** view (figure 2.22) all launches are visible in a tree. Applications utilizing multiple cores are displayed with a list of threads instead of cores. Every thread has associated stack-trace (if program is halted).

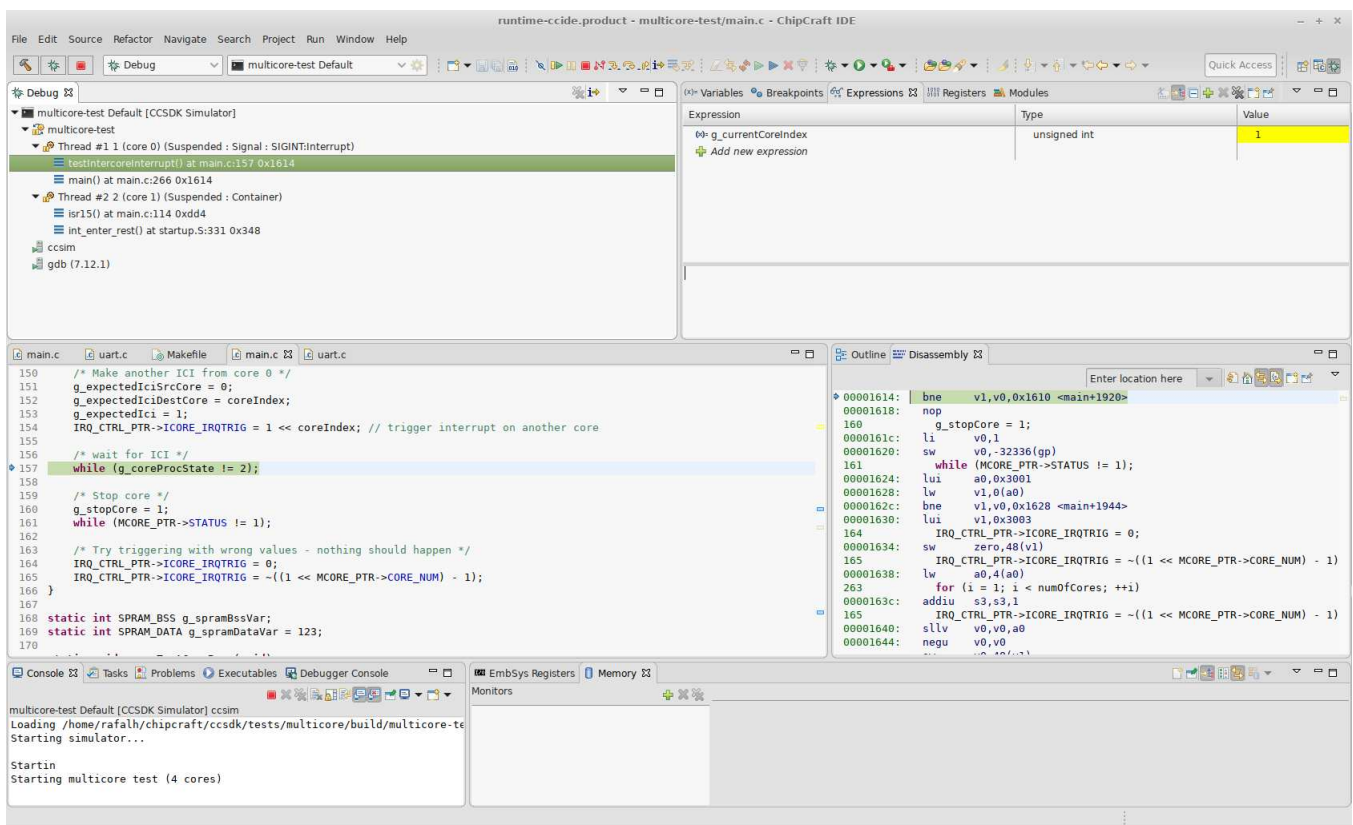


Figure 2.21. Main window during debugging.

To access standard input/output streams from debugged program use **Console** view. In case of *CCSDK Simulator* configuration type console with simulator output should already be visible. In case of *CCSDK Hardware* configuration serial terminal has to be opened manually by clicking **Serial Terminal** button in CCSDK toolbar (figure 2.9). Console can be used both for reading program output and sending text to program input (figure 2.23).





Figure 2.22. Debug process tree.

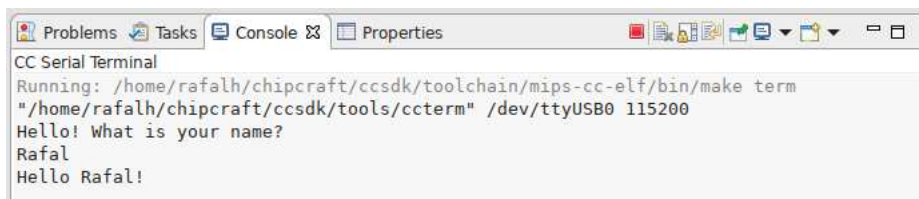


Figure 2.23. Debug UART console

**EmbSys Registers** view displays all registers of debugged device. Each register and in case of some registers groups of bits in registers can be modified independently.

Before using this view user should manually configure it by clicking a wrench icon. Then in EmbSysRegView plugin properties a proper board shall be selected.

Register	Hex	Bin	Reset	Access	Address	Description
UART						UART Controller
UART0						UART Controller
STATUS	0x0000006	00000000000000000000000000000000	0x00000000	RO	0x80000100	Status Register
RXC (bit 0)	0x0	0				Reception Complete
TXC (bit 1)	0x1	1				Transmission Complete
TXDRE (bit 2)	0x1	1				Transmission Data Register Empty
PERR (bit 3)	0x0	0				Parity Error
FRERR (bit 4)	0x0	0				Framing Error
OVERR (bit 5)	0x0	0				Overrun Error
RXBRK (bit 6)	0x0	0				Break Reception
CTS (bit 7)	0x0	0				CTS (Clear To Send) Status
PRES	0x0009000D	000000000000100100000000	0x00000000	RW	0x80000104	Prescaler Register (lo mantisa, hi fraction)
CTRL	0x00000003	00000000000000000000000000000000	0x00000000	RW	0x80000108	Control Register
MODE	0x00000000	00000000000000000000000000000000	0x00000000	RW	0x8000010c	Mode Register
TDR	0x0000006F	00000000000000000000000000000000	0x00000000	RW	0x80000110	TX data Register
RDR	0x00000000	00000000000000000000000000000000	0x00000000	RO	0x80000114	RX data Register
IRQM	0x00000000	00000000000000000000000000000000	0x00000000	RW	0x80000118	Interrupt mask Register
IRQF	0x00000000	00000000000000000000000000000000	0x00000000	RW	0x8000011c	Interrupt flags Register
IRQMAP	0x00000002	00000000000000000000000000000000	0x00000000	RW	0x80000120	Interrupt mapping
TMNG	0x00000000	00000000000000000000000000000000	0x00000000	RW	0x80000124	RS485 timings Register
UART1						UART Controller
UART2						UART Controller

Figure 2.24. Debug peripherals view.



To load contents of a register user has to double click on the register itself or a containing element (e.g. entire peripheral). Register values are displayed in red font if they changed after last program execution. To modify register content double click on register value cell and enter a new value. Note: it only works if program is already halted.



## 2.6 IDE preferences

To open CCIDE preferences window select menu *Window -> Preferences*. Most of preferences are the same as in Eclipse CDT.

In **General -> Keys** page (figure 2.25) you can configure hotkeys for all CCIDE commands. CCIDE specific commands starts with *CC* prefix.

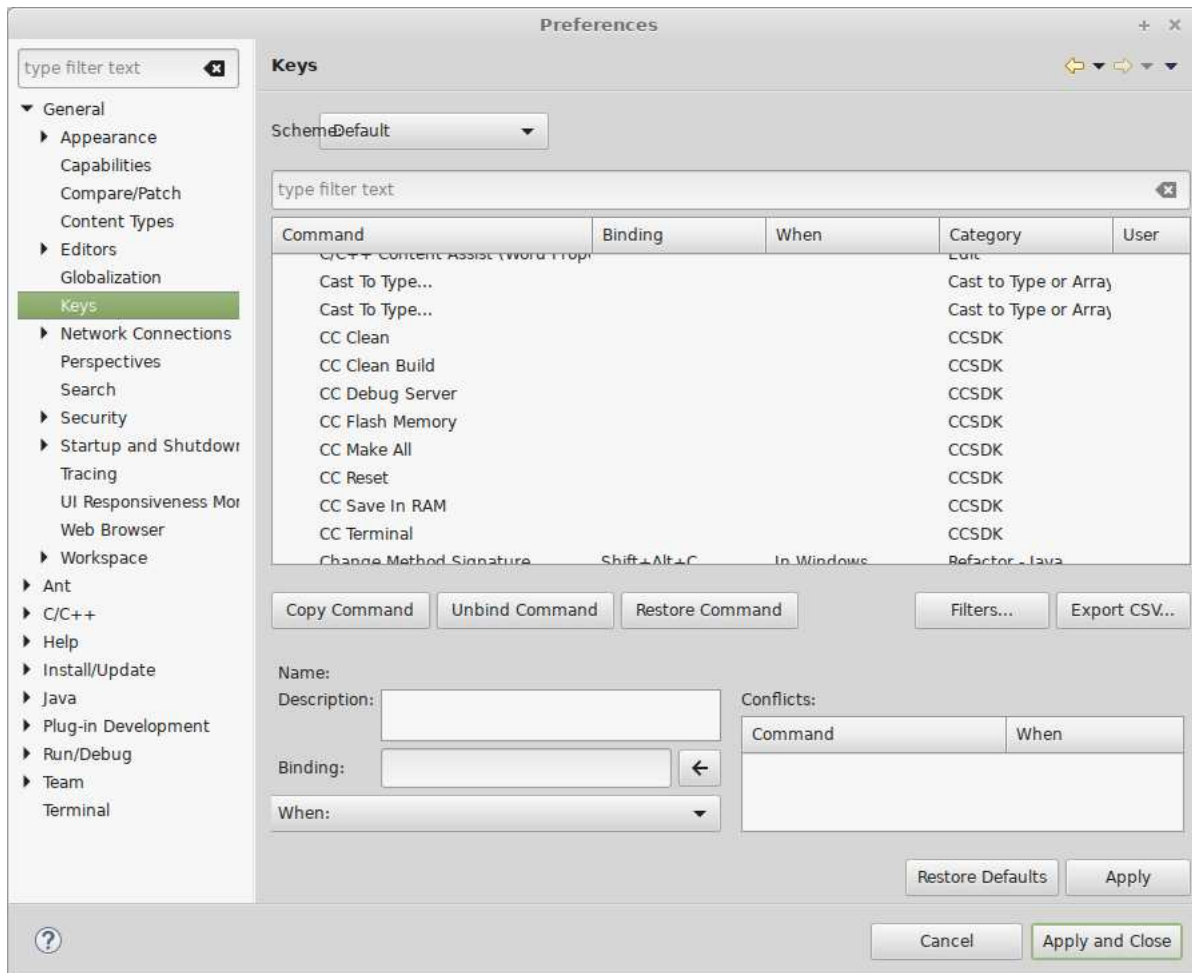


Figure 2.25. Hotkeys preferences.





### 3. Troubleshooting CCIDE

ChipCraft does its best to make sure CCIDE works good on all supported platforms but not all configurations can be tested before release so you can stand upon errors. Please report them to ChipCraft to help make our product better. Find information in this section useful to get more details about possible errors.

In case of every error Eclipse platform log file is worth checking for the cause. The easiest way to read its contents is to open *Help > About Eclipse Platform > Configuration Details*. This prints out a great number of details about the environment and also concatenates the .log file. It is great for including in a bug report.

In case of debugging issues it is useful to enable GDB traces console. To do it go to *Window -> Preferences -> C/C++ -> Debug -> GDB*. Then activate *Show the GDB traces consoles with character limit* and set the limit to a big number e.g. 5000000. After you save the changes there should be console named "gdb traces" available in Console View console selector during debugging session. All commands sent to GDB and all responses from GDB (possibly errors) should appear there.

If you still can't locate an issue happening during debugging you can enable debug mode of `dbgserver.py` script which is translating GDB commands to OCD commands. To do it go to launch configuration properties and add following arguments to the debug server command line:

```
--log DEBUG --log-file dbgserver.log
```



## 4. Revision History

Doc. Rev.	Date	Comments
1.0	03-2019	First Issue.







**ChipCraft Sp. z o.o.**

Dobrzańskiego 3 lok. BS073, 20-262 Lublin, POLAND

[www.chipcraft-ic.com](http://www.chipcraft-ic.com)

©2019 ChipCraft Sp. z o.o.

CC-IDE-UserGuide-Doc\_032019.

ChipCraft®, ChipCraft logo and combination of thereof are registered trademarks or trademarks of ChipCraft Sp. z o.o. All other names are the property of their respective owners.

Disclaimer: ChipCraft makes no representations or warranties with respect to the accuracy or completeness of the contents of this document and reserves the right to make changes to specifications and product descriptions at any time without notice. ChipCraft does not make any commitment to update the information contained herein.