Visual Studio Code C/C++ Application Development

Version 0.1
(2024-05-29)

© F&S Elektronik Systeme GmbH
Untere Waldplätze 23
D-70569 Stuttgart
Germany

Phone: +49(0)711-123722-0
Fax: +49(0)711-123722-99

About This Document

This document describes how to set up the Tester Board and the Device under Test to perform tests with the Linux Release Tester software.

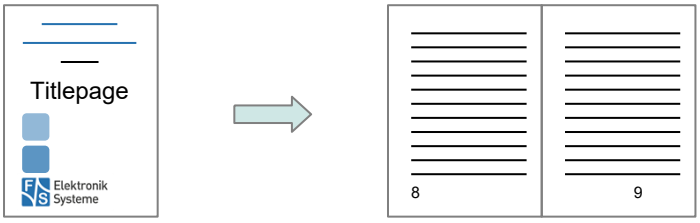
Remark

The version number on the title page of this document is the version of the document. It is not related to the version number of any software release!

How To Print This Document

This document is designed to be printed double-sided (front and back) on A4 paper. If you want to read it with a PDF reader program, you should use a two-page layout where the title page is an extra single page. The settings are correct if the page numbers are at the outside of the pages, even pages on the left and odd pages on the right side. If it is reversed, then the title page is handled wrongly and is part of the first double-page instead of a single page.

Typographical Conventions



We use different fonts and highlighting to emphasize the context of special terms:

`File names`

Menu entries

`Board input/output`

`Program code`

`PC input/output`

`Listings`

`Generic input/output`

`Variables`

History

Date	V	Platform	A,M,R	Chapter	Description	Au
2024-05-27	0.1	ALL	A	ALL	Initial version	BS

V	Version
A,M,R	Added, Modified, Removed
Au	Author

Table of Contents

1 Tested Software and Hardware Versions

The Software used in this document has been tested with the following versions.

Software	Version
Development Machine	F_S_Development_Machine-Fedora-36_V1.5
Visual Studio Code	Version: 1.89.1 Commit: dc96b837cf6bb4af9cd736aa3af08cf8279f7685 Date: 2024-05-07T05:16:23.416Z Electron: 28.2.8 ElectronBuildId: 27744544 Chromium: 120.0.6099.291 Node.js: 18.18.2 V8: 12.0.267.19-electron.0 OS: Linux x64 6.2.15-100.fc36.x86_64
Tested F&S Releases	fsimx93-Y2024.03-pre
Tested Boards	PicoCoreMX93, Rev 1.00

2 Setup of the F&S Board

For remote debugging on your F&S Board, gdbserver is needed. It should be installed in future F&S Yocto releases. If not already installed, add it to your Yocto conf/local.conf and create a new image.

```
EXTRA_IMAGE_FEATURES:append = " tools-debug"
```

3 Create the SDK

Yocto can create the SDK needed for your specific board and software environment. The SDK is needed on your host machine for cross compilation.

```
bitbake [IMAGE NAME] -c populate_sdk
```

This command produces an SDK installer that will be in tmp/deploy/sdk in the Build Directory after bitbake completes. Keep in mind that this command will not create a new image, only the SDK.

For further information consult the Yocto documentation:

https://docs.yoctoproject.org/sdk-manual/appendix-obtain.html#building-an-sdk-installer

4 Setup of the Linux Build Host

Installing the SDK

First you need to install the SDK on your Build Host.

Run the `*.sh` installation script that was created during *3 Create the SDK*. The name of the `*.sh` file may be different, depending on your board and software.

```
./fus-imx-wayland-glibc-x86_64-fus-image-std-armv8a-fsimx93-
toolchain-6.1-mickledore.sh
```

When running the script you are asked for the target directory of the SDK. For this example the default is kept, which in this case is `/opt/fus-imx-wayland/6.1-mickledore`. Remember your directory, as that is needed later on for setting up the VS Code project.

For more information, visit https://docs.yoctoproject.org/sdk-manual/using.html.

Installing gdb

Check your Linux Build Host if `gdb` is installed.

```
which -a gdb
```

If `gdb` is already installed, you will get a response like this:

```
[developer@fs-development-machine ~]$ which -a gdb
/usr/bin/gdb
/bin/gdb
```

In this case, you can skip directly to *Installing Visual Studio Code*.

Else, if `gdb` is not installed, the response looks like this:

```
/usr/bin/which: no gdb in
(/home/developer/.local/bin:/home/developer/bin:/usr/lib64/ccache:
/usr/local/bin:/usr/bin:/bin:/usr/local/sbin:/usr/sbin:/usr/local/
arm/fs-toolchain-11.2-armv8ahf/bin)
```

To install `gdb` on Fedora, first update the yum database with following command:

```
sudo dnf makecache –refresh
```

The output should look similar to this:

```
[developer@fs-development-machine ~]$ sudo dnf makecache --refresh
Fedora 36 - x86_64                                10 kB/s | 6.7 kB     00:00
Fedora 36 openh264 (From Cisco) - x86_64         2.2 kB/s | 989  B     00:00
Fedora Modular 36 - x86_64                        10 kB/s | 6.6 kB     00:00
Fedora 36 - x86_64 - Updates                      13 kB/s | 5.5 kB     00:00
Fedora Modular 36 - x86_64 - Updates              14 kB/s | 5.5 kB     00:00
Metadata cache created.
[developer@fs-development-machine ~]$
```

After updating the database you can install `gdb` using `dnf` with this command:

```
sudo dnf -y install gdb
```

If you use another Linux distribution as Fedora, change **`dnf`** to your distributions package manager (for example **`apt`** on Ubuntu / Debian).

Installing Visual Studio Code

On Fedora, the latest stable 64-bit VS Code is available in a yum repository. Add the repository:

```
sudo rpm --import
https://packages.microsoft.com/keys/microsoft.asc

echo -e "[code]\nname=Visual Studio
Code\nbaseurl=https://packages.microsoft.com/yumrepos/vscode\nenab
led=1\ngpgcheck=1\ngpgkey=https://packages.microsoft.com/keys/micr
osoft.asc" | sudo tee /etc/yum.repos.d/vscode.repo > /dev/null
```

Then update the package cache:

```
dnf check-update
```

And last, install VS Code:

```
sudo dnf install code
```

For more information and installation on other distributions as Fedora, see the VS Code documentation at https://code.visualstudio.com/docs/setup/linux.

After the installation of VS Code finished you should install the needed extension for C and C++ debugging:

```
code --install-extension ms-vscode.cpptools
```

Get the example Code

Clone the example `fs_vscode_c-cpp_app_development` from Github.

```
git clone https://github.com/FSEmbedded/fs_vscode_c-
cpp_app_development
```

Navigate to the example directory and open it in VS Code:

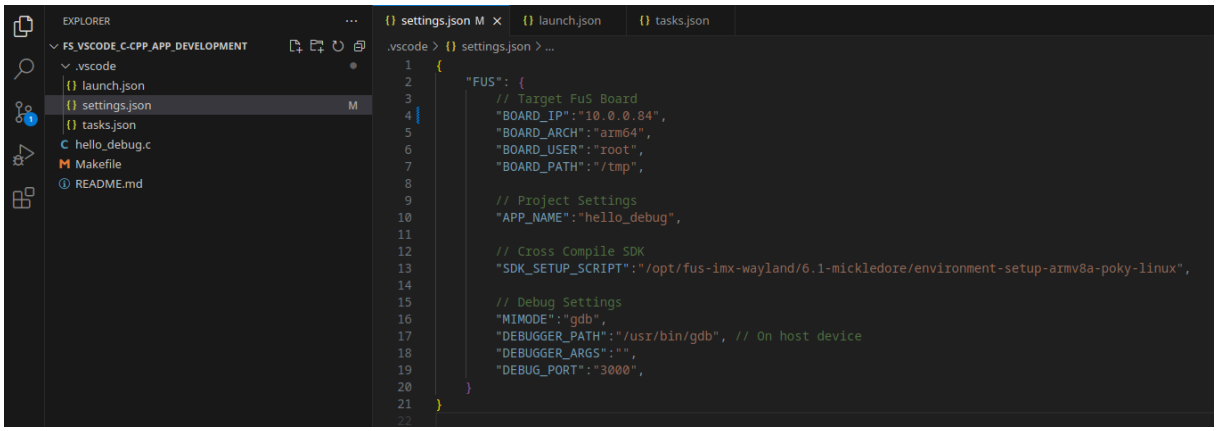
```
cd fs_vscode_c-cpp_app_development

code .
```

Adapt Settings

Inside `fs_vscode_c-cpp_app_development`, have a look at `.vscode`. There are three json-files which contain everything needed for cross compiling and remote debugging.

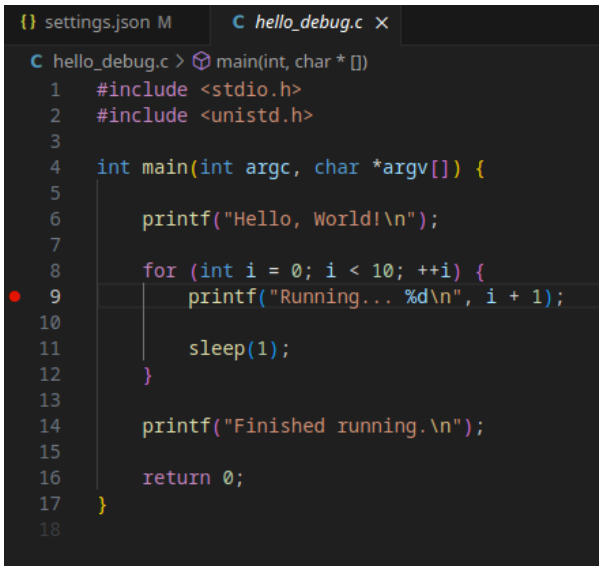
Adapt the settings in `settings.json` to your environment. The most important Settings are the BOARD_IP and SDK_SETUP_SCRIPT. The other values can probably stay the same.

For SDK_SETUP_SCRIPT, follow the path defined as target directory when installing the SDK. Inside this directory is a file named *environment-setup-**. Add the whole path to this file as value for SDK_SETUP_SCRIPT

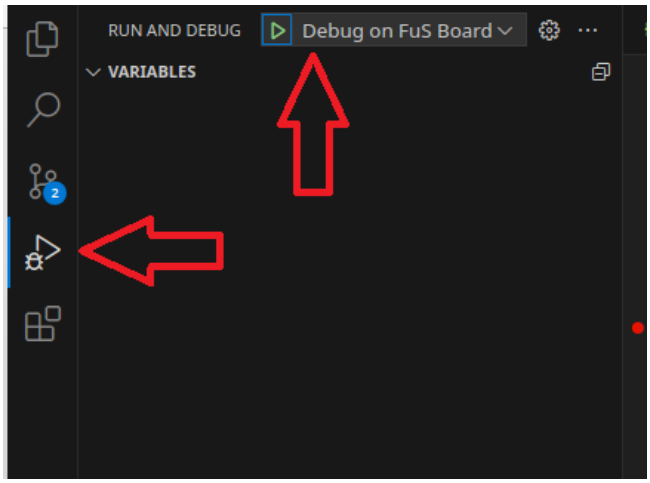


Cross Compilation and Debugging

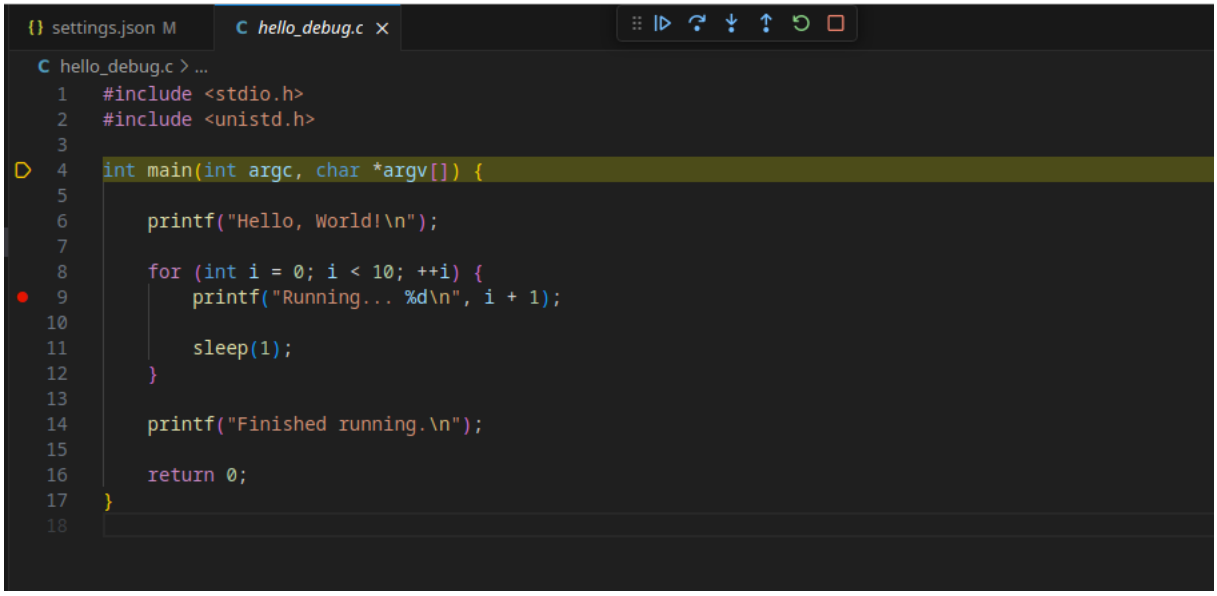
Now you are ready for remote debugging. You can add a breakpoint in `hello_debug.c` to see the debugger working.



Go to the *Run and Debug* tab and select *Debug on FuS Board*.

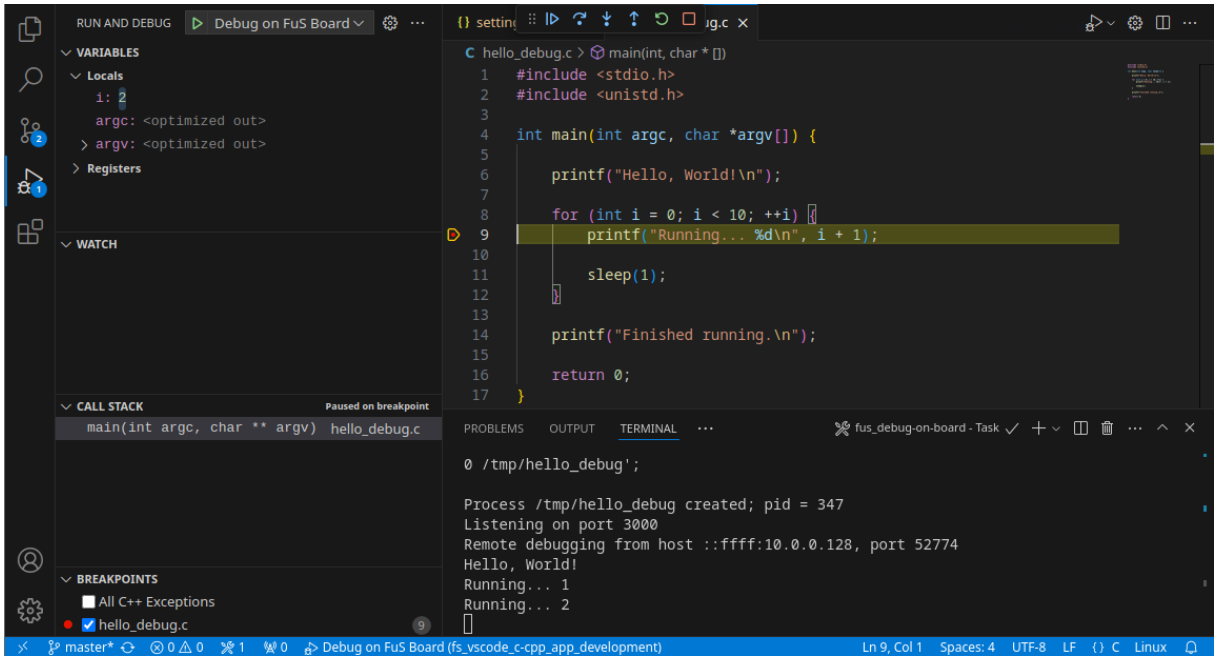


Click *Start Debugging* or use the Shortcut F5 to start the Debug process. The program will stop at the execution of main and at your set breakpoints.



If you start debugging, VS Code will first cross compile using the SDK, then deploy the binary and finally start `gdbserver` on your board, all automated using the tasks in `tasks.json`.

You can see the output of hello_debug in the terminal inside VS Code.



If you only want to execute one of the tasks included in the *Debug on FuS Board* launch configuration, you can start each task in `tasks.json` by opening the *Command Palette* in VS Code (**Ctrl+Shift+P**), type "***Run Task***" and select your choice.

fus_cross-compile will only compile using the SDK.

fus_deploy-to-board will compile and then deploy to your board.

fus_launch-on-board will compile, deploy and launch the application on your board.

fus_debug-on-board is the task executed by *Debug on FuS Board*.

build is a shortcut to ***fus_cross-compile*** and can be called directly: **Ctrl+Shift+B**