	SSE-320 BSP Pack User Guide
Version:	1.1.0
Date of Issue:	March 28, 2025

Contents

1 Concept

1.1 Introduction

This document is a general guide to use the SSE-320 BSP pack. The CMSIS pack is to be used with the Corstone®-320 platform FVP model (Arm Corstone™ SSE-320 with Cortex®-M85 Example Subsystem). The pack contains necessary source files, a linker script file, and a specification document to kick-start development for the Corstone-320 platform. It provides reference secure side Blinky, Vio and ISP examples, to enable a user to understand csolution project configuration. The pack also provides a System View Description (SVD) file for the platform to be used with the uVision and IAR debugger. This document specifies system prerequisites and explains how to build and run the reference Blinky, Vio and ISP examples on the SSE-320 FVP model.

Terms and Abbreviations:

Terms	Meaning
BSP	Board Support Pack
FVP	Fixed Virtual Platform
VIO	Virtual I/O
ISP	Image Signal Processor

1.2 Arm Mali®-C55 Versatile Image Signal Processor

The Corstone-320 Example Subsystem contains the Arm Mali®-C55 Versatile Image Signal Processor for Computer Vision and Smart Display Systems. The pack contains the necessary source files for HDLCD, to enable displaying the results of Image Signal Processor.
Example application and Driver can be found in Arm™ Mali®-C55 bare-metal ISP driver gitlab repository. An example can also be found as part of this pack, introduced in the ISP minimal example: CMSIS-Toolbox chapter.

1.3 Prerequisites

- **Developement Environments:**
  CMSIS-Toolbox v2.6.0 or Keil Studio for Visual Studio Code.
- **Compiler:**
  Arm Compiler for Embedded (Version 6.20 or newer),
  GNU Arm Embedded Toolchain Arm GCC (Version 13.3 or newer),
  LLVM Embedded Toolchain for Arm (Version 17.0.1 or newer), or
  IAR C/C++ Compiler for Arm (Version 9.50.1 or newer).
- **FVP model:**
  Corstone SSE-320 FVP model
- **Package manager:**
  vcpkg

After installing the FVP, you should source the runtime script, which will provide the necessary exports to run it without any issue.

```
$ <path-to-fvp-root-dir>/scripts/runtime.sh
```

If you encounter issues with running the FVP after sourcing the runtime, check the `LD_LIBRARY_PATH` environment variable, and re-export it without the path containing `fmtplib`.

vcpkg will automatically install all project dependencies ready for compilation, including compilers `armclang, gcc, clang` and `CMSIS-Toolbox`, but it will not install `iararm`.

After installation, make sure to configure the appropriate license settings in your build environment, as some toolchains may require a valid license.

1.4 Documents

There are no available resources at the pack yet.

SSE-320 BSP Pack contains the additional documentations in the "Documents" folder.

2 Installation

2.1 CMSIS-Toolbox

To install the `ARM::SSE_320_BSP` pack in command-line, use:

```
# run once, to initialize the package repository
$ cpackget init https://www.keil.com/pack/index.pidx

# run once, to get access to the latest packages
$ cpackget update-index

# run with ‘-a‘ to accept all the EULA, and if you want to install from script
$ cpackget add ARM.SSE_320_BSP
```

This will install the prerequired `ARM.CMSIS.6.1.0`, `ARM.CMSIS-Compiler.2.0.0`, `ARM.DMA350.1.0.0` and `ARM.CMSIS-FreeRTOS.11.1.0` packages as well.

3 Blinky example: CMSIS-Toolbox

You can see the example's folder structure under Blinky example tree chapter. The following example will assume the usage of a linux machine, therefore some commands might differ on Windows and Mac.
As the example contains a `Makefile`, you can use that as well, but we will go through in every configuration without it as well.

3.1 Import

Copy the containing folder from the CMSIS Pack to your working directory, and provide it with the necessary permissions:

```
$ cp -r ${CMSIS_PACK_ROOT}/ARM/SSE_320_BSP/1.1.0/Examples/Blinky blinky-example
$ chmod -R +w blinky-example
$ cd blinky-example
```

Then install the project dependencies:

```
$ vcpkg-shell activate
```

Read more about vcpkg

3.2 Build

The examples support the `[AC6, GCC, IAR, CLANG]` compilers, which are represented as `[armclang, gcc, iar, clang]` in the `Makefile`. Let's make a build with `AC6` toolchain:

```
$ cbuild Blinky.csolution.yml --packs --update-rte --jobs $(nproc) --context .Debug --toolchain AC6
```

or

```
$ make armclang
```

If you just want to generate the `.cprj` project files, without building the whole project, you can use `csolution`.

```
$ csolution convert Blinky.csolution.yml --toolchain [AC6, GCC, IAR, CLANG] --context .[Debug, Release]
```

Used tags while building:

Tag	Mandatory	Meaning
—packs	No	Downloading packs required for the pack.
—update-rte	No	Update the RTE directory and files.
—jobs [n]	No	Build on `[n]` threads.
—toolchain	Yes	Compiler Toolchain. If missing builds for all 4 compilers.
—context	Yes	Selecting the build type or board. Select `.Debug` or `.Release`

See the help and documentation of `cbuild` and `cmsis-toolbox` for further information.

Binary:

Find the binary files in the following location:

```
build/{platform}/{toolchain}/{type}/{example}/outdir/Blinky.[axf,elf,out]
```

as

```
build/SSE-320-FVP/AC6/Debug/Blinky/outdir/Blinky.axf
```

3.3 Run - Terminal

After building the target, you can launch the FVP, using the command:

```
$ <path_to_fvp>/FVP_Corstone_SSE-320 -a build/SSE-320-FVP/AC6/Debug/Blinky/outdir/Blinky.axf
```

or, after setting the model path `<path_to_fvp>` in the `Makefile`:

```
$ make run-armclang
```

4 Blinky example: Keil Studio

You can see the example's folder structure under Blinky example tree chapter.

To Use Keil Studio, install the Keil Studio Pack for **Visual Studio Code**

4.1 Import

Copy the containing folder from the CMSIS Pack:

```
$ cp -r ${CMSIS_PACK_ROOT}/ARM/SSE_320_BSP/1.1.0/Examples/Blinky blinky-example
$ chmod -R +w blinky-example
$ cd blinky-example
```

Then open your working directory in **Visual Studio Code**.

Or open from Visual Studio Code:

Go to CMSIS, then create a solution with `SSE-320` selected as the following:

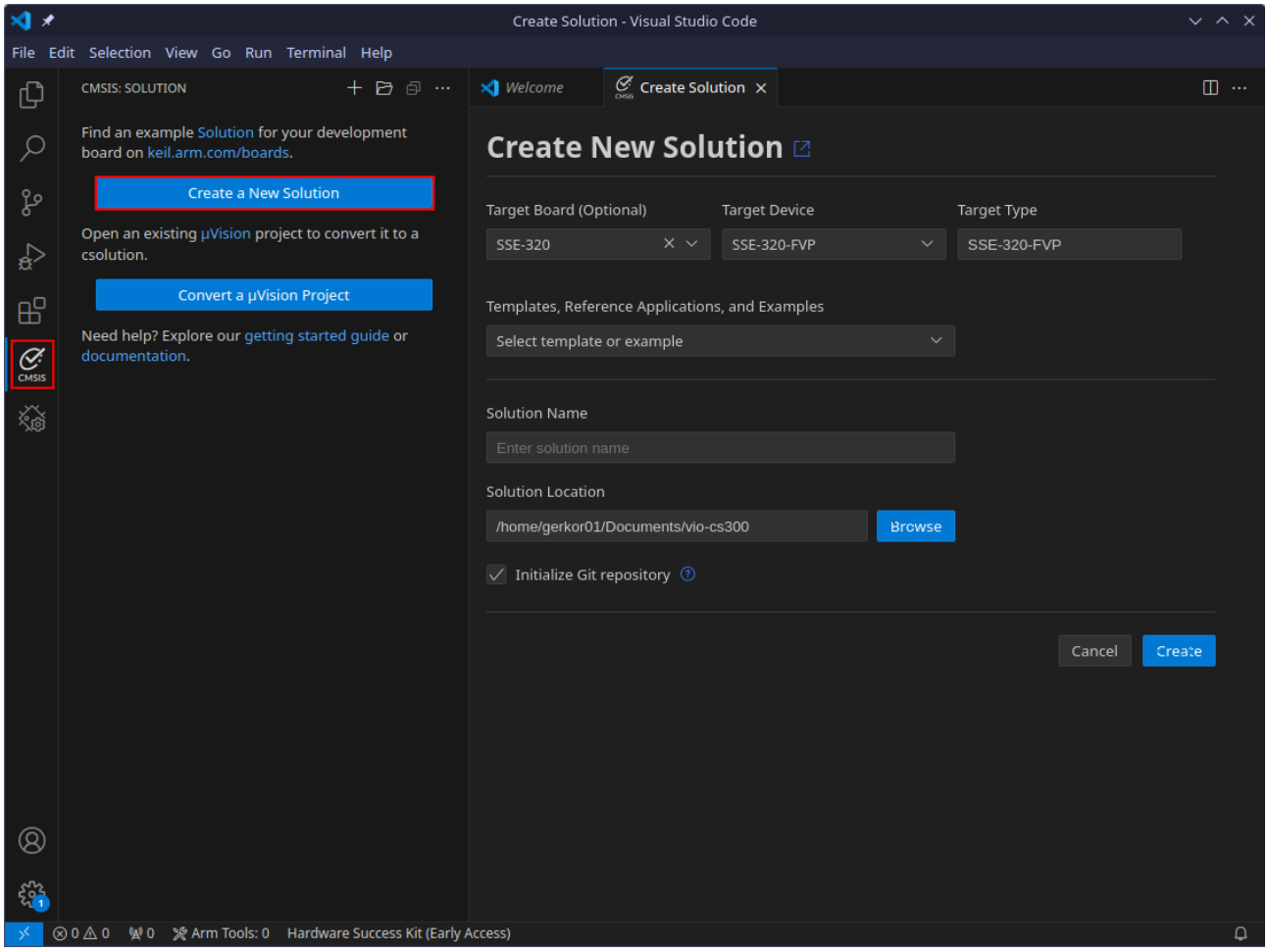

Figure 1: Keil Studio: Importing Blinky Project

4.2 Build

To build the example, open the **CMSIS** tab on the left side of **Visual Studio Code**, then click on **Build**.

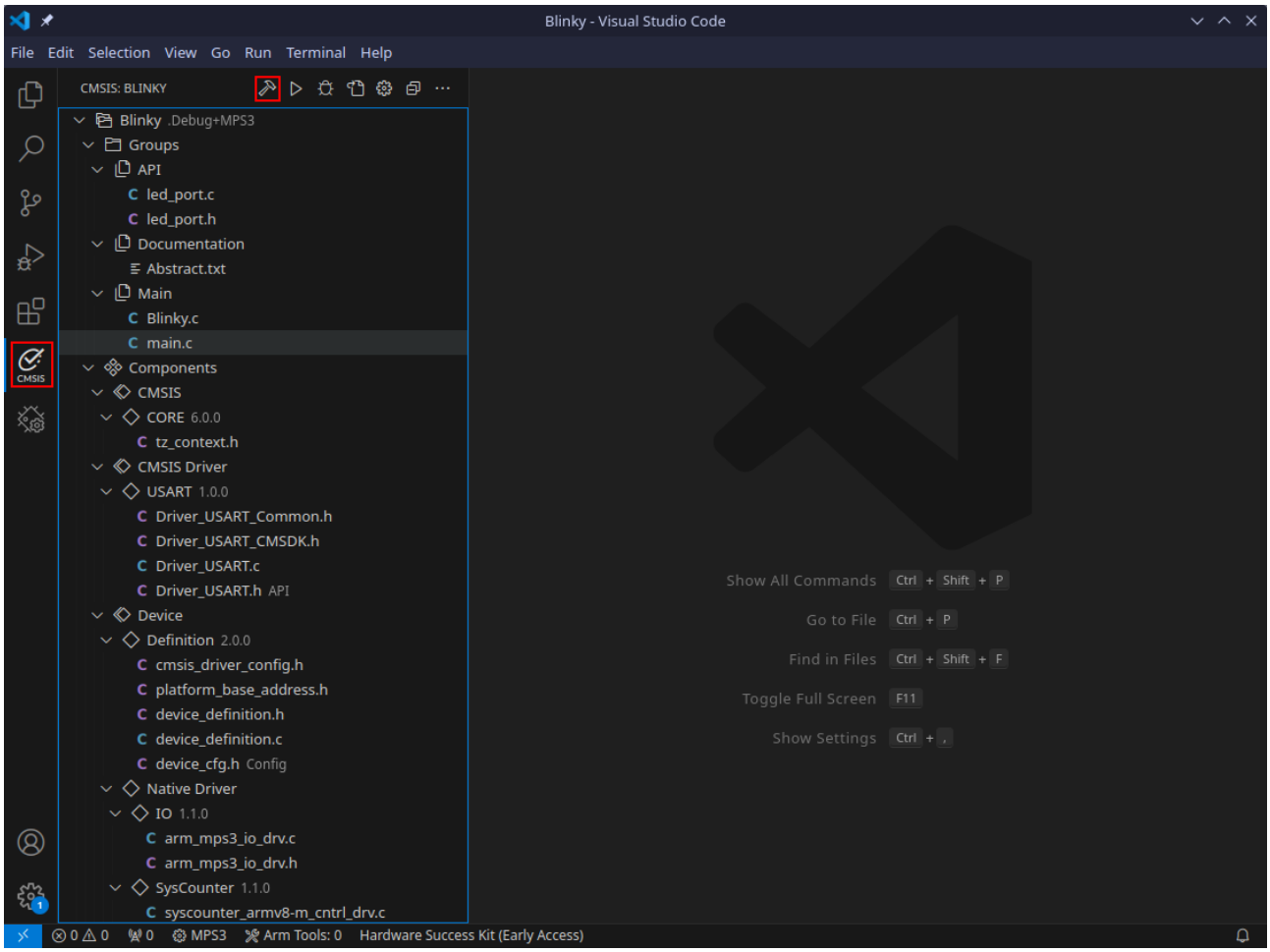

Figure 2: Keil Studio: Building Project

Set compiler:
By default the project is built by `AC6` toolchain.
To build with other toolchain, add `'compiler: [AC6, GCC, IAR, CLANG]'` to `Blinky.csolution.yml` under `solution:` tag, like:

```
12.
13.  compiler:  GCC
14.  cdefault:
15.
```

Binary:
Find the binary files in the following location:

`build/{platform}/{toolchain}/{type}/{example}/outdir/Blinky.[axf,elf,out]`
as
`build/SSE-320-FVP/AC6/Debug/Blinky/outdir/Blinky.axf`

4.3 Run - Terminal

After building the target, you can launch the FVP, using the command:

```
$ <path_to_fvp>/FVP_Corstone_SSE-320 -a build/SSE-320-FVP/AC6/Debug/Blinky/outdir/Blinky.axf
```

5 Vio example: CMSIS-Toolbox

You can see the example's folder structure under Vio example tree chapter. The following example will assume the usage of a linux machine, therefore some commands might differ on Windows and Mac.
As the example contains a `Makefile`, you can use that as well, but we will go through in every configuration without it as well.

5.1 Import

Copy the containing folder from the CMSIS Pack to your working directory, and provide it with the necessary permissions:

```
$ cp -r ${CMSIS_PACK_ROOT}/ARM/SSE_320_BSP/1.1.0/Examples/Vio vio-example
$ chmod -R +w vio-example
$ cd vio-example
```

Then install the project dependencies:

```
$ vcpkg-shell activate
```

Read more about vcpkg

5.2 Build

The examples support the `[AC6, GCC, IAR, CLANG]` compilers, which are represented as `[armclang, gcc, iar, clang]` in the `Makefile`. Let's make a build with `AC6` toolchain:

```
$ cbuild Vio.csolution.yml --packs --update-rte --jobs $(nproc) --context .Debug --toolchain AC6
```

or

```
$ make armclang
```

Used tags while building:

Tag	Mandatory	Meaning
—packs	No	Downloading packs required for the pack.
—update-rte	No	Update the RTE directory and files.
—jobs [n]	No	Build on `[n]` threads.
—toolchain	Yes	Compiler Toolchain. If missing builds for all 4 compilers.
—context	Yes	Selecting the build type or board. Select `.Debug` or `.Release`

See the help and documentation of `cbuild` and `cmsis-toolbox` for further information.

If you just want to generate the `.cprj` project files, without building the whole project, you can use `csolution`.

```
$ csolution convert Blinky.csolution.yml --context .[Debug, Release] --toolchain
[AC6, GCC, IAR, CLANG]
```

Binary:
Find the binary files in the following location:

`build/{platform}/{toolchain}/{type}/{example}/outdir/Vio.[axf,elf,out]`
as
`build/SSE-320-FVP/AC6/Debug/Vio/outdir/Vio.axf`

5.3 Run - Terminal

Download **arm_vio.py**, like the following:
On Windows:
```
PS > wget https://github.com/ARM-software/AVH/raw/main/interface/python/arm_vio.py -OutFile arm_vio.py
```
On Linux:
```
$ wget https://github.com/ARM-software/AVH/raw/main/interface/python/arm_vio.py -O arm_vio.py
```

Then set `verbosity = logging.DEBUG` in it.

```
15
16  ## Set verbosity level
17  verbosity = logging.DEBUG
18  #verbosity = logging.ERROR
19
```

Figure 3: Setting debug mode in `arm_vio.py`

After building the target, you can launch the FVP, using the command:
```
$ <path_to_fvp>/FVP_Corstone_SSE-320 -a build/SSE-320-FVP/AC6/Debug/Vio/outdir/Vio.axf
  -C mps4_board.v_path=<path_to_vio_py_dir>
```

or, after setting the model path `<path_to_fvp>` and `<path_to_vio_py_dir>` in the `Makefile`:
```
$ make run-armclang
```

Remember to use the path of the containing folder of `arm_vio.py` when specifying path `<path_to_vio_py_dir>`.

6 Vio example: Keil Studio

You can see the example's folder structure under Vio example tree chapter.

To Use Keil Studio, install the Keil Studio Pack for **Visual Studio Code**

6.1 Import

Copy the containing folder from the CMSIS Pack:

```
$ cp -r ${CMSIS_PACK_ROOT}/ARM/SSE_320_BSP/1.1.0/Examples/Vio vio-example
$ chmod -R +w vio-example
$ cd vio-example
```

Then open your working directory in **Visual Studio Code**.

Or open from Visual Studio Code:

Go to CMSIS, then create a solution with `SSE-320` selected as the following:

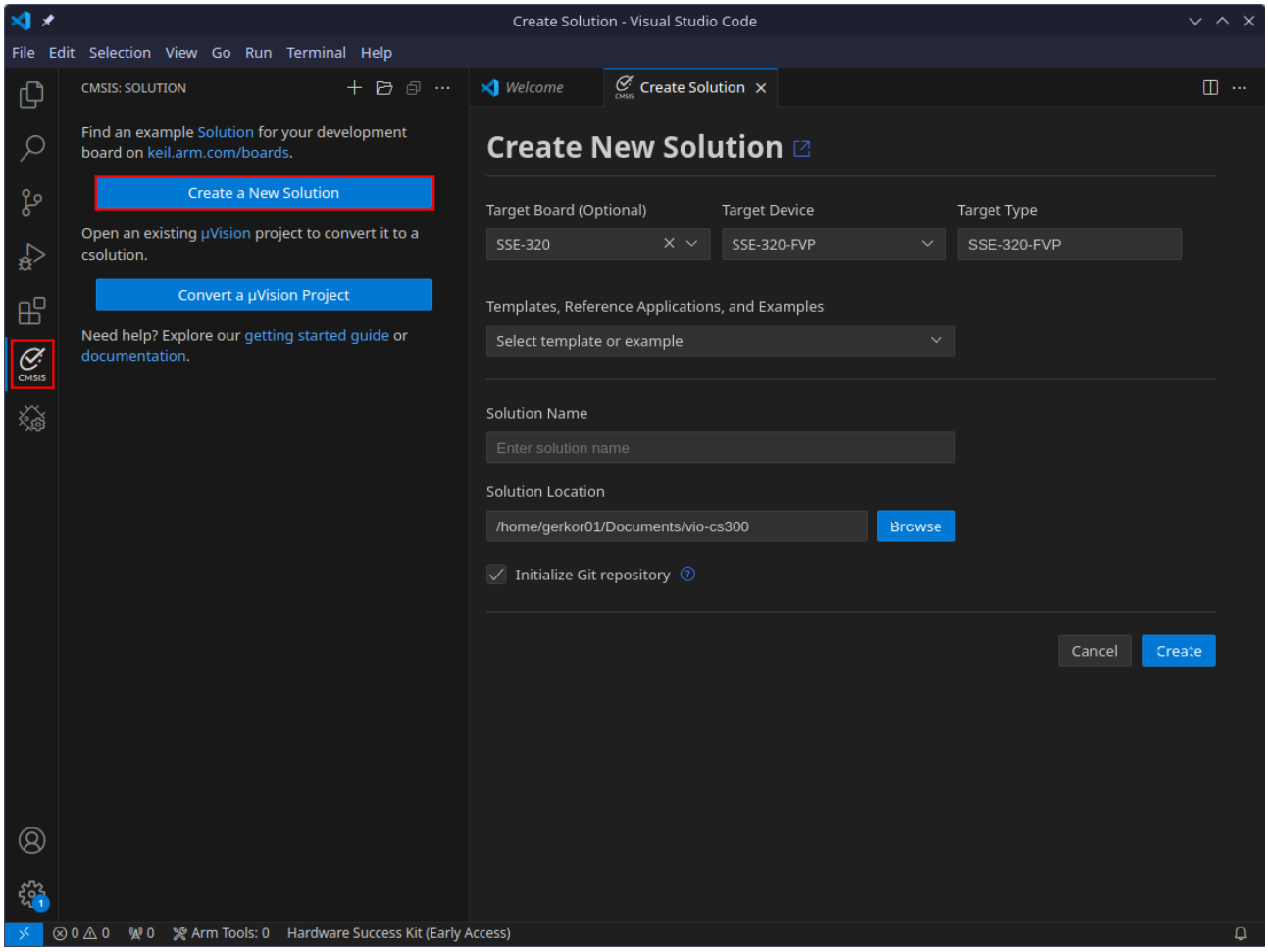

Figure 4: Keil Studio: Importing Vio Project

6.2 Build

To build the example, open the **CMSIS** tab on the left side of **Visual Studio Code**, then click on **Build**.

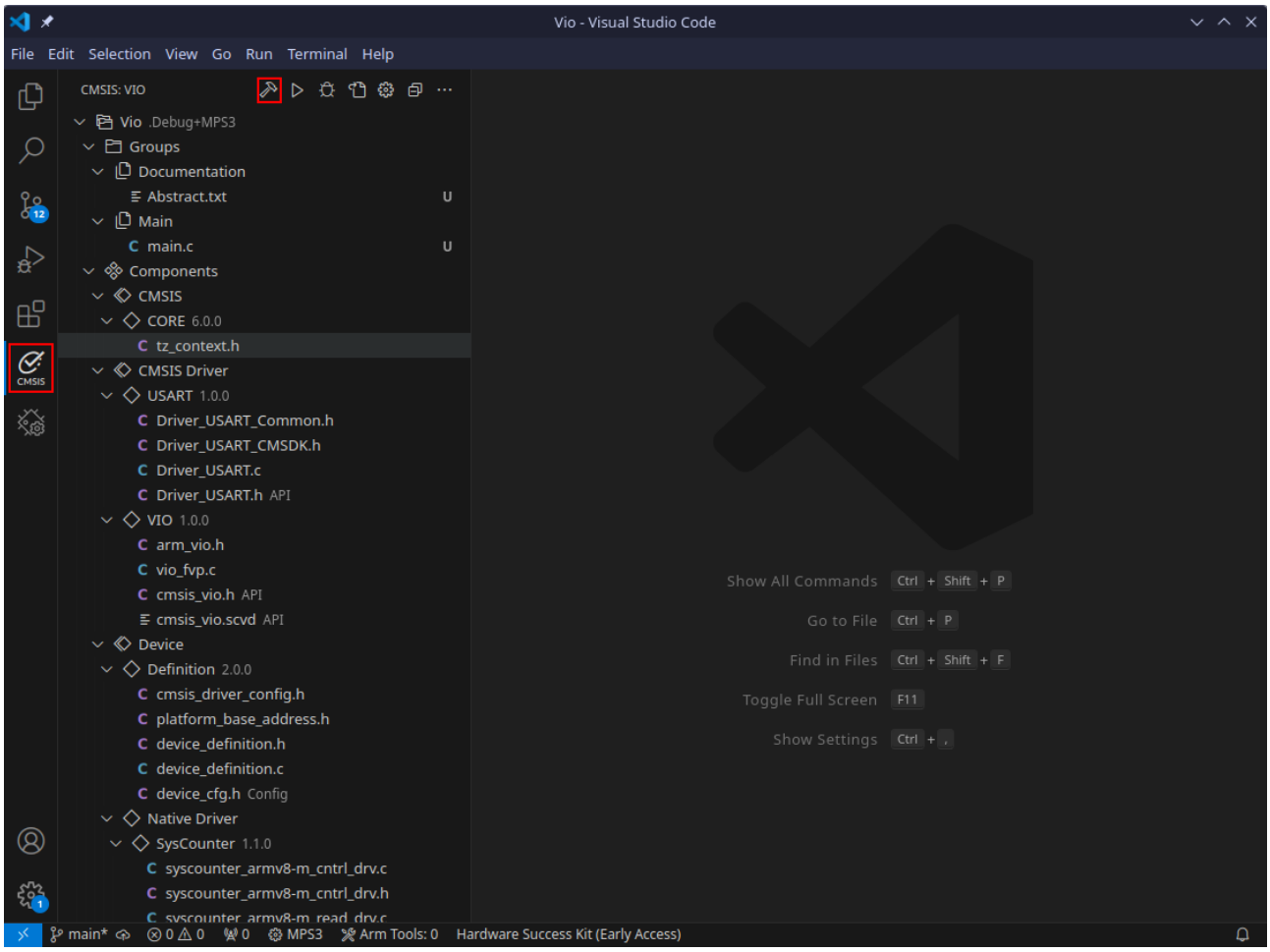

Figure 5: Keil Studio: Building Project

Set compiler:
By default the project is built by `AC6` toolchain.
To build with other toolchain, add `'compiler: [AC6, GCC, IAR, CLANG]'` to `Vio.csolution.yml` under `solution:` tag, like:

```
12.
13.  compiler:  GCC
14.  cdefault:
15.
```

Binary:
Find the binary files in the following location:

`build/{platform}/{toolchain}/{type}/{example}/outdir/Vio.[axf,elf,out]`
as
`build/SSE-320-FVP/AC6/Debug/Blinky/outdir/Vio.axf`

6.3 Run - Terminal

Download **arm_vio.py**, like the following:

On Windows:

```
PS > wget https://github.com/ARM-software/AVH/raw/main/interface/python/arm_vio.py -OutFile arm_vio.py
```

On Linux:

```
$ wget https://github.com/ARM-software/AVH/raw/main/interface/python/arm_vio.py -O arm_vio.py
```

Then set `verbosity = logging.DEBUG` in it.

```
15
16  ## Set verbosity level
17  verbosity = logging.DEBUG
18  #verbosity = logging.ERROR
19
```

Figure 6: Setting debug mode in `arm_vio.py`

After building the target, you can launch the FVP, using the command:

```
$ <path_to_fvp>/FVP_Corstone_SSE-320 -a build/SSE-320-FVP/AC6/Debug/Vio/outdir/Vio.axf
  -C mps4_board.v_path=<path_to_vio_py_dir>
```

or, after setting the model path `<path_to_fvp>` and `<path_to_vio_py_dir>` in the `Makefile`:

```
$ make run-armclang
```

Remember to use the path of the containing folder of `arm_vio.py` when specifying path `<path_to_vio_py_dir>`.

7 ISP minimal example: CMSIS-Toolbox

The ISP minimal example uses the FVP's built-in camera module to feed the ISP. The ISP's output is shown on the HDLCD, either at full resolution or as a downscaled image, depending on the configuration set by the `DISPLAY_FR_NDS` macro. For further information, refer to the ISP documentation and the Arm™ Mali®-C55 bare-metal ISP driver.
You can see the example's folder structure under ISP minimal example tree chapter. The following example will assume the usage of a linux machine, therefore some commands might differ on Windows and Mac.
As the example contains a `Makefile`, you can use that as well, but we will go through in every configuration without it as well.

7.1 Import

Copy the containing folder from the CMSIS Pack to your working directory, and provide it with the necessary permissions:

```
$ cp -r ${CMSIS_PACK_ROOT}/ARM/SSE_320_BSP/1.1.0/Examples/ISP_minimal isp-minimal-example
$ chmod -R +w isp-minimal-example
$ cd isp-minimal-example
```

Then install the project dependencies:
`$ vcpkg-shell activate`
Read more about vcpkg

7.2 Build

The examples support the `[AC6, GCC]` compilers, which are represented as `[armclang, gcc]` in the `Makefile`. Let's make a build with `AC6` toolchain:

```
$ cbuild ISP_minimal.csolution.yml --packs --update-rte --jobs $(nproc) --context .Debug --toolchain AC6
```
or
```
$ make armclang
```

If you just want to generate the `.cprj` project files, without building the whole project, you can use `csolution`.
`$ csolution convert ISP_minimal.csolution.yml --toolchain [AC6, GCC] --context .[Debug, Release]`

Used tags while building:

Tag	Mandatory	Meaning
—packs	No	Downloading packs required for the pack.
—update-rte	No	Update the RTE directory and files.
—jobs [n]	No	Build on `[n]` threads.
—toolchain	Yes	Compiler Toolchain. If missing builds for all 4 compilers.
—context	Yes	Selecting the build type or board. Select `.Debug` or `.Release`

See the help and documentation of `cbuild` and `cmsis-toolbox` for further information.

Binary:
Find the binary files in the following location:

`build/{platform}/{toolchain}/{type}/{example}/outdir/{example}.[axf,elf]`
as
`build/SSE-320-FVP/AC6/Debug/ISP_minimal/outdir/ISP_minimal.axf`

7.3 Run - Terminal

After building the target, you can launch the FVP, using the command:

```
$ <path_to_fvp>/FVP_Corstone_SSE-320 -a build/SSE-320-FVP/AC6/Debug/ISP_minimal/outdir/ISP_minimal.axf
  -C mps4_board.isp_c55_camera.image_file=ISP_minimal/test.frm
  -C mps4_board.isp_c55_capture_ds.do_capture=0
  -C mps4_board.isp_c55_capture_fr.do_capture=0
  -C mps4_board.telnetterminal0.mode=raw
```

or, after setting the model path `<path_to_fvp>` in the `Makefile`:

```
$ make run-armclang
```

7.3.1 Expected Output

When running the application, the HDLCD output will appear in a separate window (Figure 7). By default, it displays the downscaled input image. To display the full-resolution image, set the `DISPLAY_FR_NDS` macro to `1` in the `isp_demo.c` file.

Figure 7: Image displayed on the HDLCD

The application's log output will be shown in a terminal window (Figure 8). In this example, the application prints log messages—for instance, when a new frame starts, when the ISP output is ready, and when the image is displayed on the HDLCD.

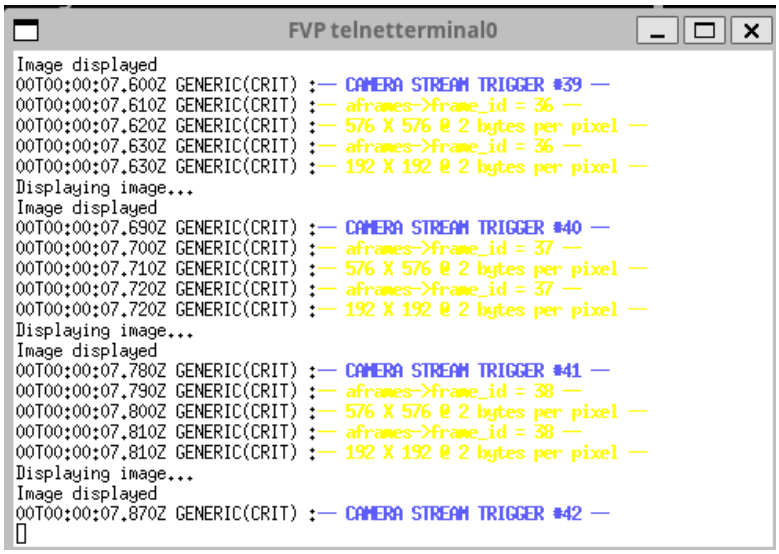

Figure 8: Example log output

8 ISP minimal example: Keil Studio

You can see the example's folder structure under ISP minimal example tree chapter.

To Use Keil Studio, install the Keil Studio Pack for **Visual Studio Code**

8.1 Import

Copy the containing folder from the CMSIS Pack:

```
$ cp -r ${CMSIS_PACK_ROOT}/ARM/SSE_320_BSP/1.1.0/Examples/ISP_minimal isp-minimal-example
$ chmod -R +w isp-minimal-example
$ cd isp-minimal-example
```

Then open your working directory in **Visual Studio Code**.

Or open from Visual Studio Code:

Go to CMSIS, then create a solution with `SSE-320` selected as the following:

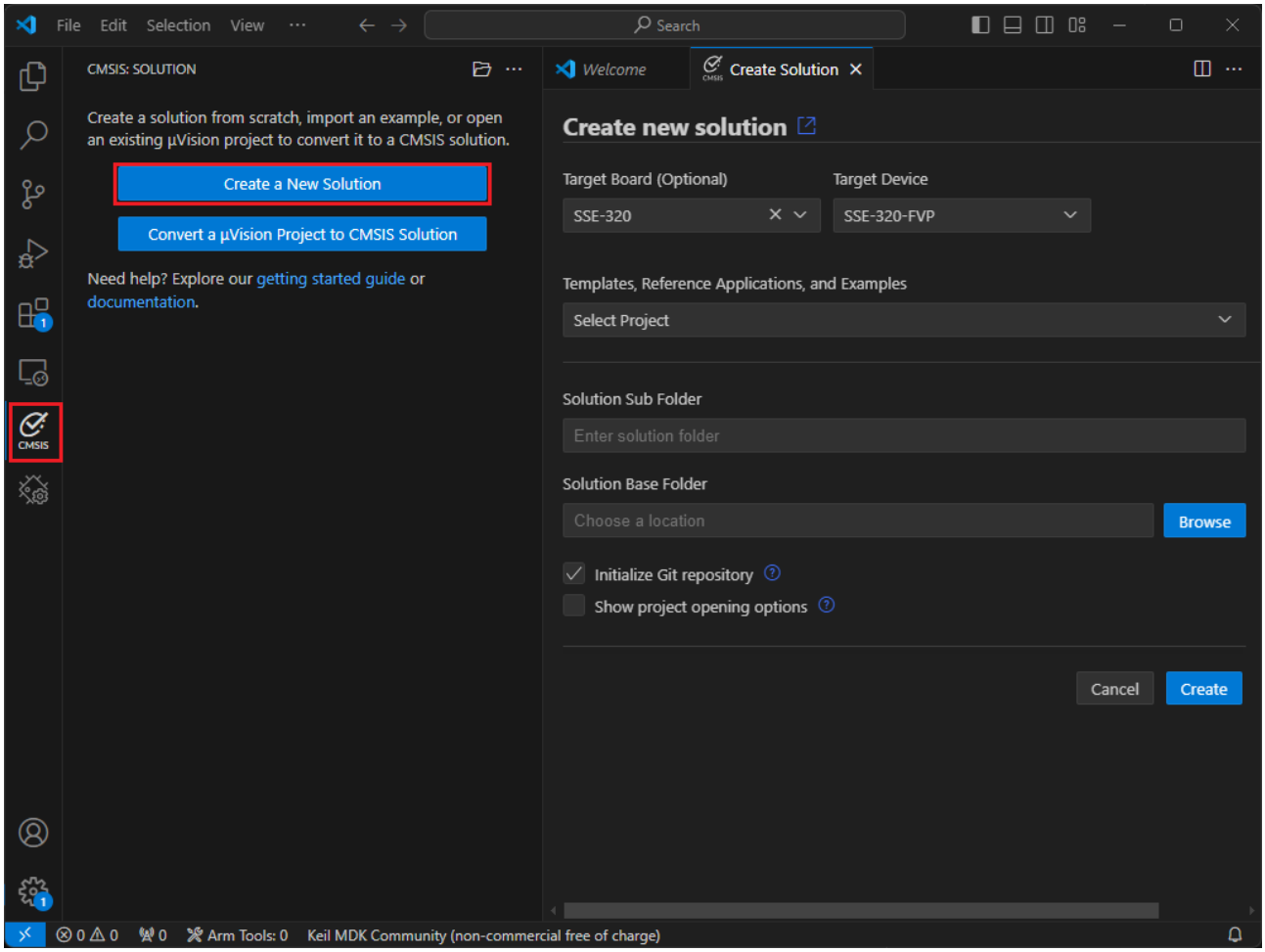

Figure 9: Keil Studio: Importing ISP Minimal Project

8.2 Build

To build the example, open the **CMSIS** tab on the left side of **Visual Studio Code**, then click on **Build**.

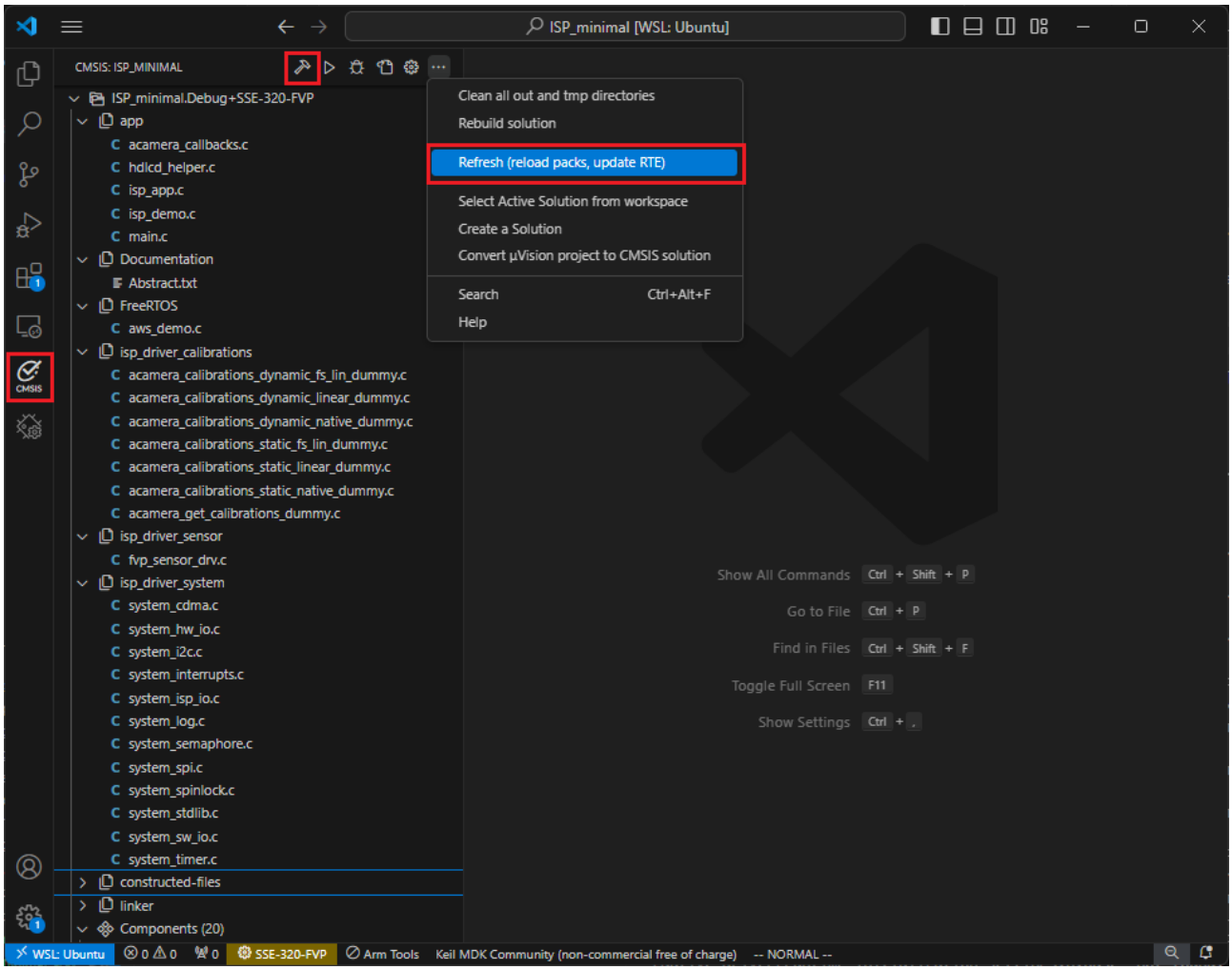

Figure 10: Keil Studio: Building Project

Before building the example, you may need to update the RTE directory by clicking the refresh button.

Set compiler:
By default the project is built by `AC6` toolchain.
To build with other toolchain, add `'compiler: [AC6, GCC]'` to `ISP_minimal.csolution.yml` under `solution:` tag, like:

```
12.
13.  compiler:  GCC
14.  cdefault:
15.
```

Binary:
Find the binary files in the following location:

`build/{platform}/{toolchain}/{type}/{example}/outdir/{example}.[axf,elf]`
as
`build/SSE-320-FVP/AC6/Debug/ISP_minimal/outdir/ISP_minimal.axf`

8.3 Run - Terminal

After building the target, you can launch the FVP, using the command:

```
$ <path_to_fvp>/FVP_Corstone_SSE-320 -a build/SSE-320-FVP/AC6/Debug/ISP_minimal/outdir/ISP_minimal.axf
  -C mps4_board.isp_c55_camera.image_file=ISP_minimal/test.frm
  -C mps4_board.isp_c55_capture_ds.do_capture=0
  -C mps4_board.isp_c55_capture_fr.do_capture=0
  -C mps4_board.telnetterminal0.mode=raw
```

8.3.1 Expected Output

The expected output of the FVP is the same as in Expected Output.

9 Attachments

9.1 Blinky example tree

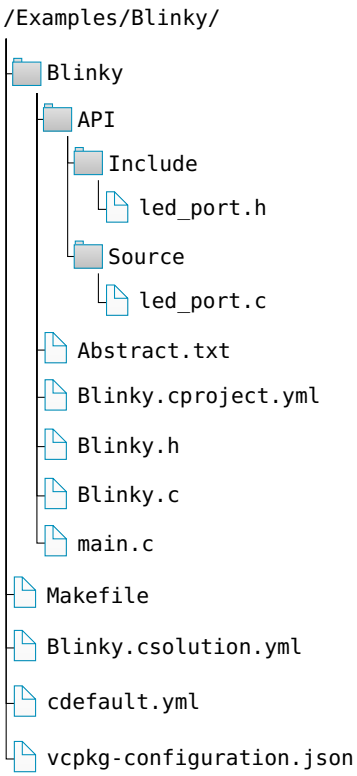

Figure 11: Blinky example's folder structure

9.2 Vio example tree

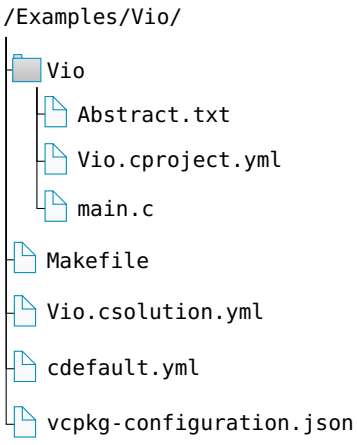

Figure 12: VIO example's folder structure

9.3 ISP minimal example tree

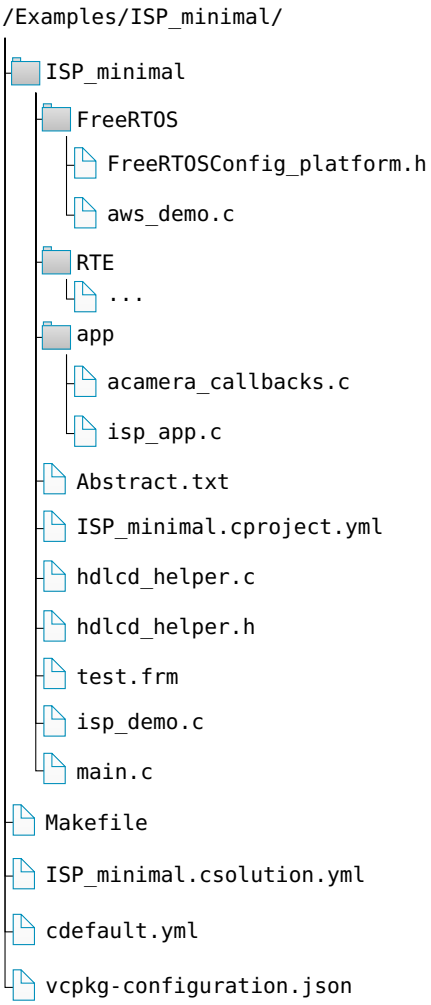

Figure 13: ISP minimal example's folder structure