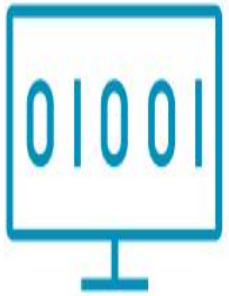
ARM FIXED VIRTUAL PLATFORM

ANURAG SINGH

Last Semester Trainees – 2022

WHAT IS ARM FIXED VIRTUAL PLATFORM ?

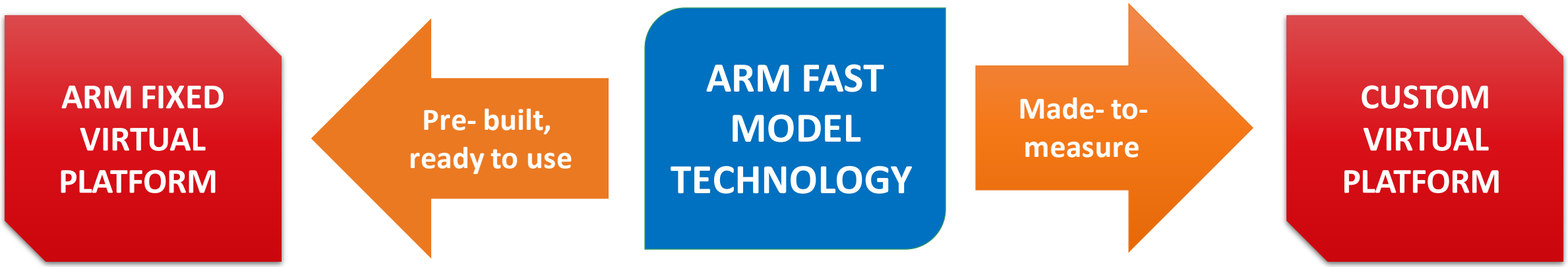


X

✓

- **Fixed Virtual Platforms (FVP)** are simulation models used by engineers for software development before the equipment is delivered.
- FVP are designed to emulate the work of a complete system, just as if it was physically connected to the programming environment.
- The simulation models are developed and tested along with Arm IP, providing **very accurate** and proven virtual prototypes for software development.

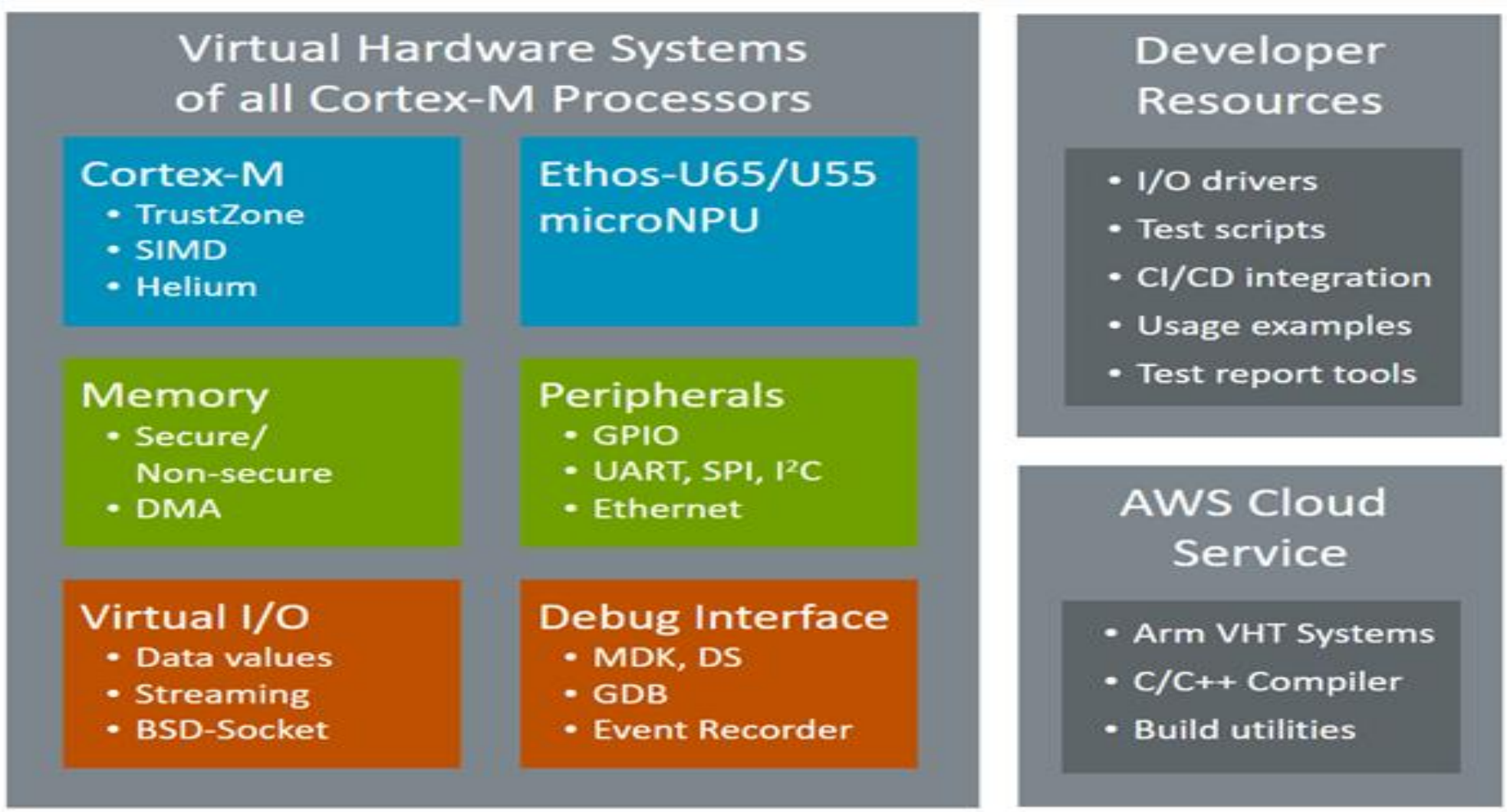
TYPE OF ARM ENABLE VIRTUAL PLATEFORM



- Low cost software development platforms for Arm technologies
- Comprehensive debug and trace
- Available for Cortex-A, Cortex-R and Cortex-M

- Flexible Virtual Platforms tailored to any system, Extensible, scalable.
- Integration with Arm Cycle Models
- Integrates into open source and proprietary System, simulators and EDA partner tools

ARM FIXED VIRTUAL PLATEFORM COMPONENTS



ARM FIXED VIRTUAL PLATEFORM COMPONENTS

ARM VIRTUAL HARDWARE TARGET

ARM VIRTUAL HARDWARE SERVICES

- **Arm Virtual Hardware Targets (VHT)**

1. Simulation models of Cortex-M
2. Complex software verification and testing.
3. Simulation-based test automation

- **Arm Virtual Hardware Services**

1. Cloud-native infra-structure for software test and validation.
2. Integrated into CI/CD and MLOps
3. AWS Marketplace

ARM FIXED VIRTUAL PLATEFORM COMPONENTS

ARM VIRTUAL HARDWARE DEVELOPER

SOFTWARE DEVELOPMENT ENVIRONMENTS

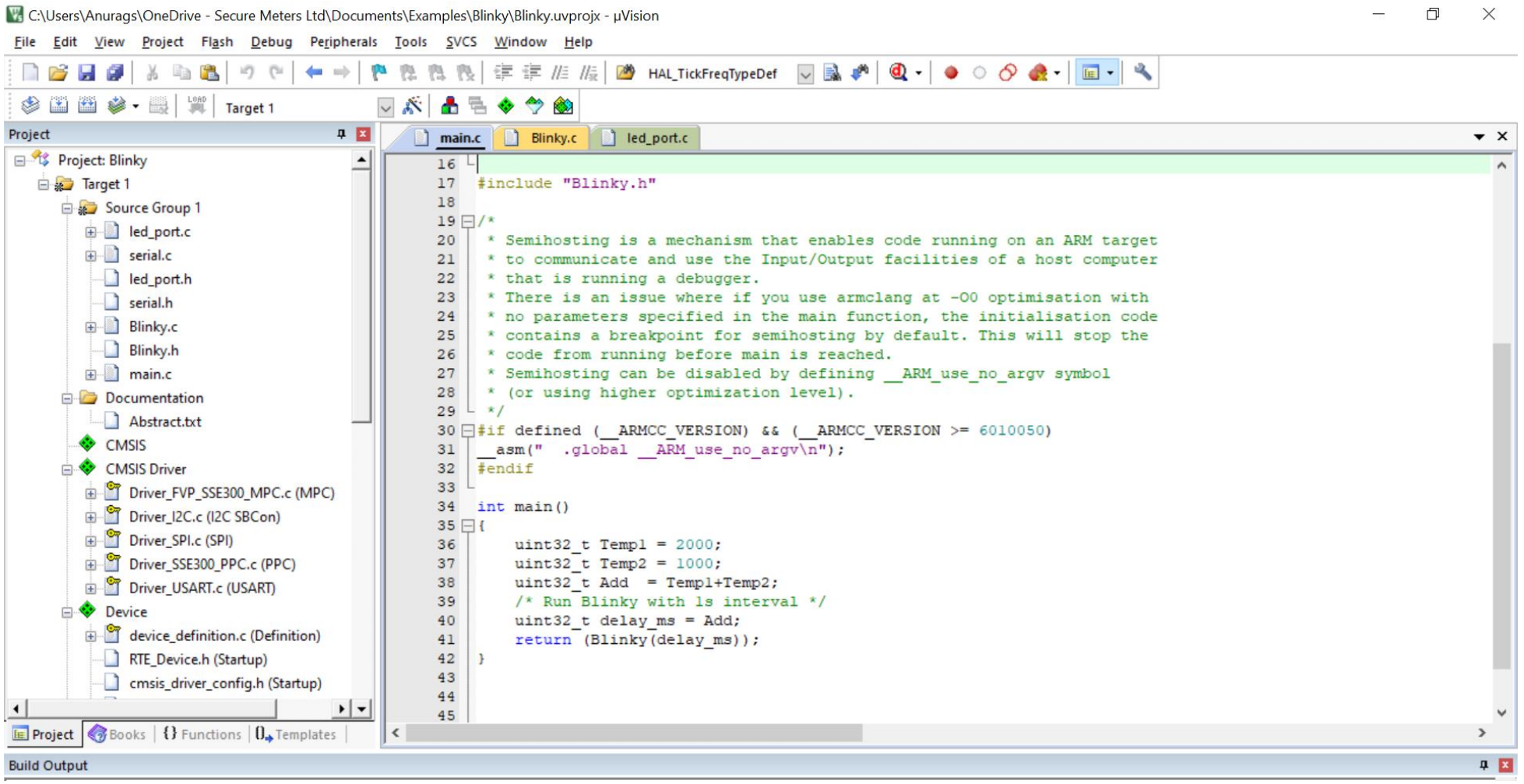
- **Arm Virtual Hardware Developer Resources**

1. Access to interface drivers map to virtual targets and physical hardware.
2. Audio processing, ML algorithm testing,

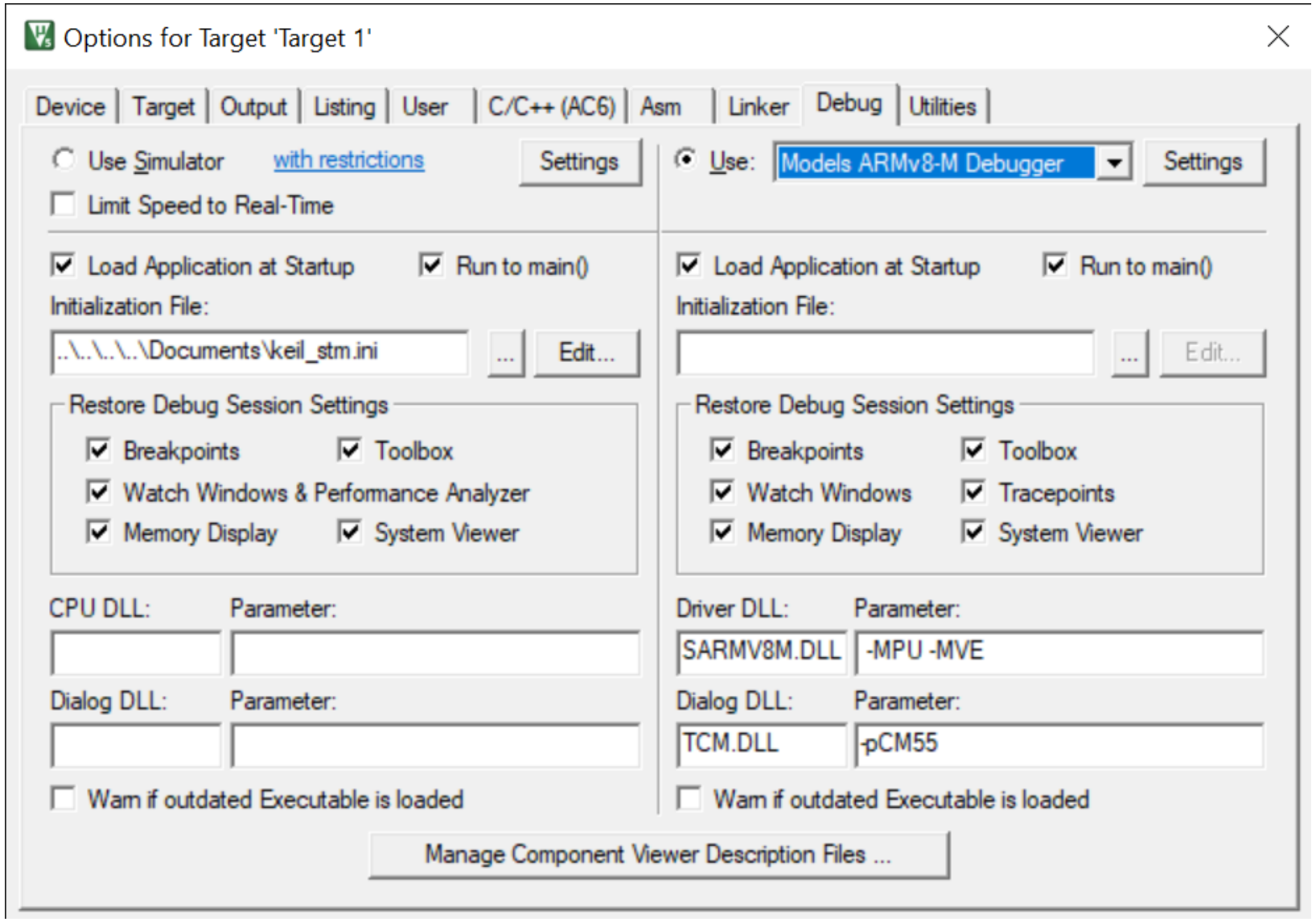
- **Software Development Environments**

1. Hardware is an integral part of the Keil MDK Professional Edition.
2. In future, the next-generation Keil Studio will also integrate Arm Virtual Hardware.

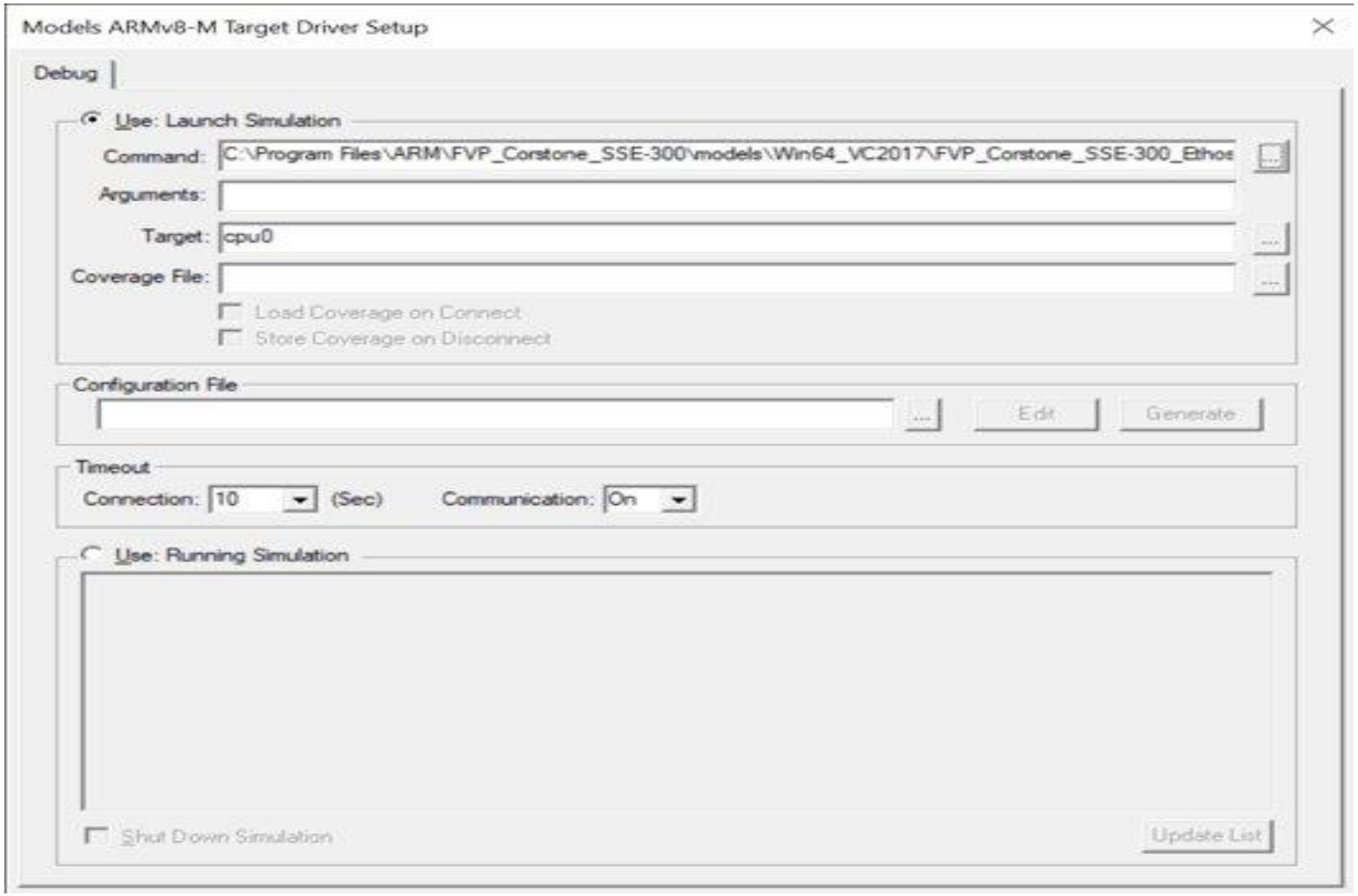
RUNNING SIMPLE BLINKY CODE IN FVP



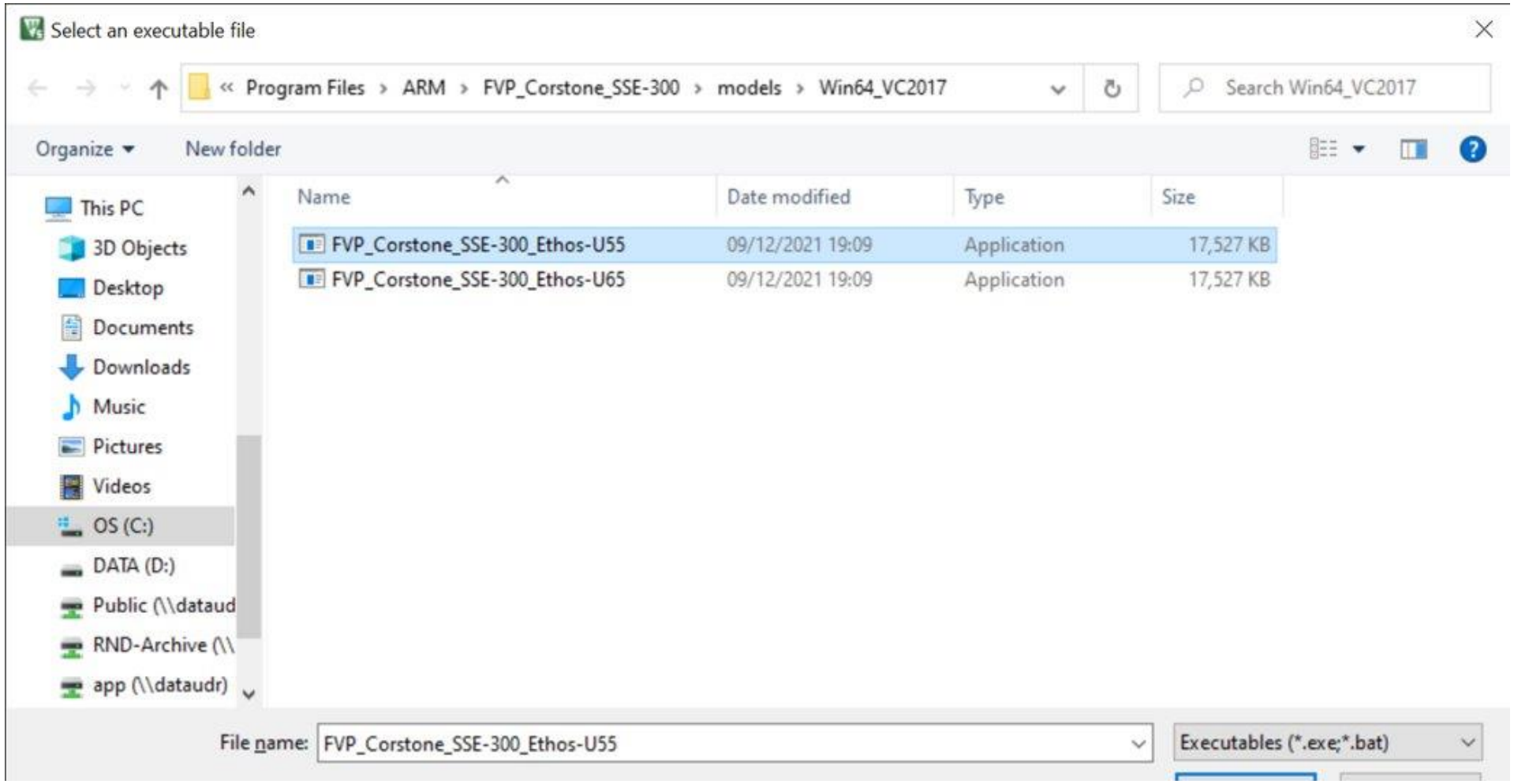
RUNNING SIMPLE BLINKY CODE IN FVP



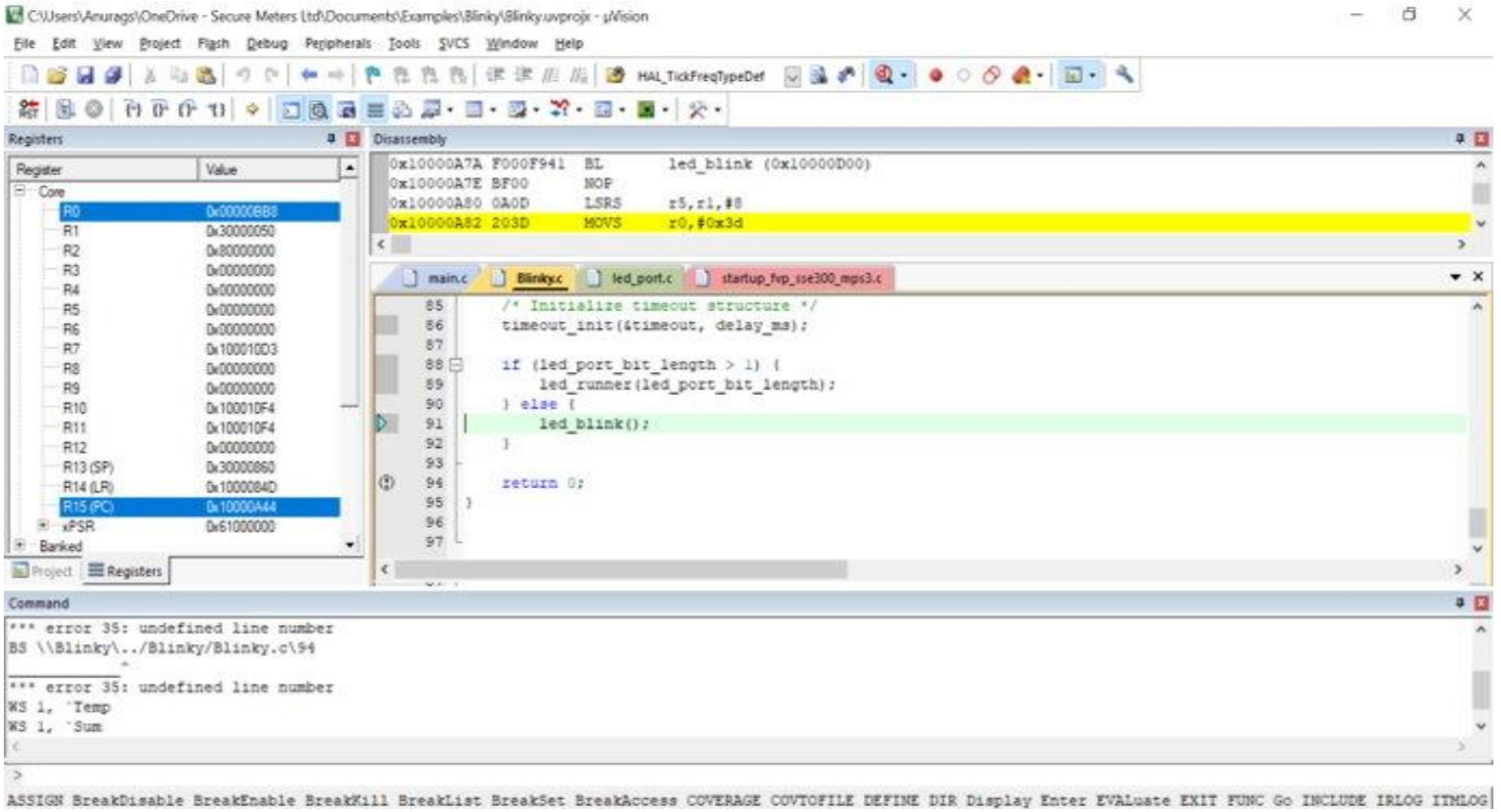
RUNNING SIMPLE BLINKY CODE IN FVP



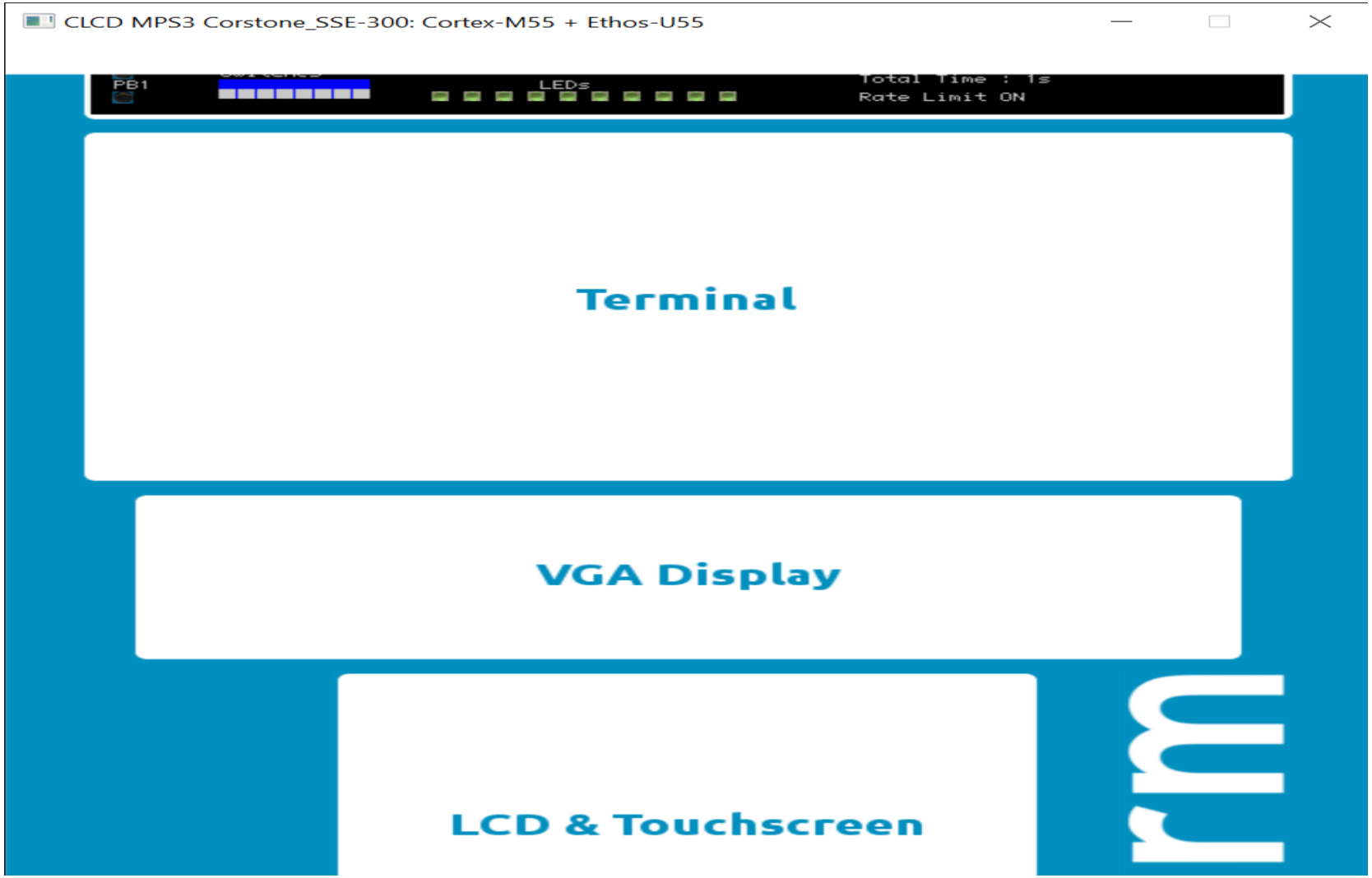
RUNNING SIMPLE BLINKY CODE IN FVP



RUNNING SIMPLE BLINKY CODE IN FVP



RUNNING SIMPLE BLINKY CODE IN FVP



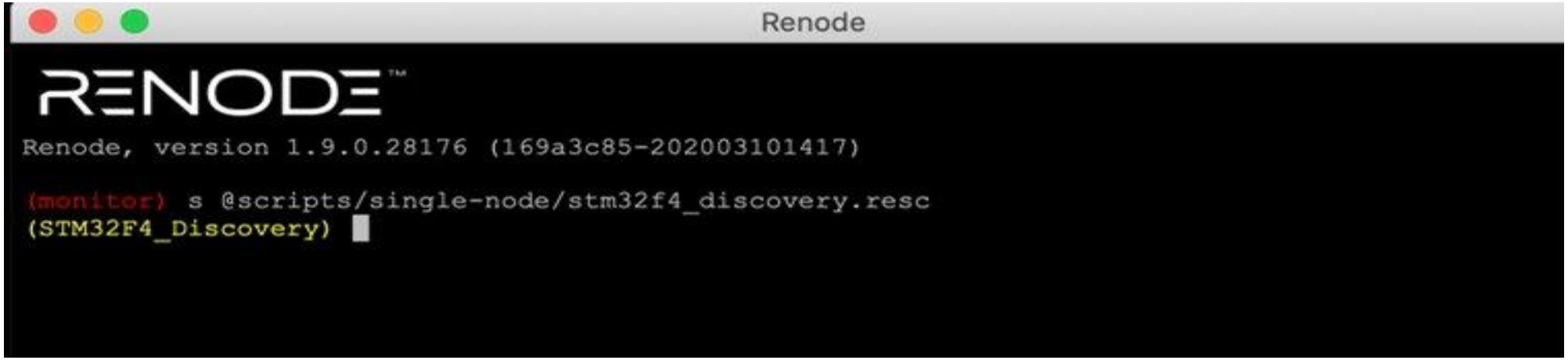
Drawbacks of ARM FVP

- The open source license scope is limited and we were not able to test many features of the Corstone-300.
- In Corstone-300 we were not able to excess the terminals, VGA display, LCD and touch screen components provided in FVP.
- In Corstone-300 we were not able to excess the push button on the FVP display layout.
- The corstone-300 have also not provided definition on external peripherals support. with the FVP.

WHAT IS RENODE ?

- Renode is an open-source Emulator for embedded platforms. Today, it supports x86 (Intel Quark), Cortex-A (NVIDIA Tegra), Cortex-M, SPARC (Leon), and RISC-V based platforms.
- Renode can take the same firmware we are running in production, and run it against emulated cores, peripherals, and even sensors and actuators. Better yet, its extensive networking support and multi-system emulation make it a shoe in for testing systems made up of multiple devices talking together.

.



RUNNING SIMPLE FIRMWARE IN RENODE

Create a machine

A machine represents a device, which can have a number of cores. Renode can run multiple machines in a single run.

Load our firmware

Before we start the machine, we want to do one last thing: open a terminal to display UART data. Some peripherals come with what Renode calls “Analyzers”, which are ways to display their state and data. We enabled UART2 in our firmware, so we will show the analyzer for UART2.

Configure our machine

we configure our machine. We could specify each bus and peripheral by hand, but instead we will load a prebuilt configuration.


Renode
RENODE™
Renode, version 1.12.0.25160 (44d6786a-202104021357)
(monitor) mach create
(machine-0) machine LoadPlatformDescription @platforms/boards/stm32f4_discovery-kit.repl
(machine-0) machine LoadPlatformDescription @example/add-ccm.repl
(machine-0) showAnalyzer sysbus.uart2
(machine-0) sysbus LoadELF @out.elf
(machine-0) start
Starting emulation...
(machine-0)

DEBUGGING IN RENODE

- Inevitably things will go wrong, and you will need to debug them. This is one area where Renode really shines.
- One of the advantages of emulators is that they make it much easier to introspect and trace the device state. One of the more useful hooks exposed by Renode is execution tracing. Provided as to fed the emulator an ELF file with debug symbols, Renode will print out log out every function being executed.

```
(machine-0) sysbus.cpu LogFunctionNames True
(machine-0)
```

```
(machine-0) logFile @/tmp/function-trace.log
(machine-0)
```

```
[...]
23:12:16 [INFO] cpu: Entering function _write at 0x80001D8¬
23:12:16 [INFO] cpu: Entering function usart_send_blocking (entry) at 0x80003B4¬
23:12:16 [INFO] cpu: Entering function usart_wait_send_ready (entry) at 0x80003D2¬
23:12:16 [INFO] cpu: Entering function usart_wait_send_ready at 0x80003D8¬
23:12:16 [INFO] cpu: Entering function usart_send_blocking at 0x80003BE¬
23:12:16 [INFO] cpu: Entering function usart_send (entry) at 0x80003CA¬
23:12:16 [INFO] cpu: Entering function _write at 0x80001E2¬
23:12:16 [INFO] cpu: Entering function _write at 0x80001BE¬
23:12:16 [INFO] cpu: Entering function _write at 0x80001CA¬
23:12:16 [INFO] cpu: Entering function _write at 0x80001D0¬
23:12:16 [INFO] cpu: Entering function usart_send_blocking (entry) at 0x80003B4¬
23:12:16 [INFO] cpu: Entering function usart_wait_send_ready (entry) at 0x80003D2¬
23:12:16 [INFO] cpu: Entering function usart_wait_send_ready at 0x80003D8¬
23:12:16 [INFO] cpu: Entering function usart_send_blocking at 0x80003BE¬
23:12:16 [INFO] cpu: Entering function usart_send (entry) at 0x80003CA¬
23:12:16 [INFO] cpu: Entering function _write at 0x80001D8¬
23:12:16 [INFO] cpu: Entering function usart_send_blocking (entry) at 0x80003B4¬
[...]
```

GDB INTEGRATION IN RENODE

- A debugger is a program that runs other programs, allowing the user to exercise control over these programs, and to examine variables when problems arise. GNU Debugger, which is also called **gdb,** is the most popular debugger for UNIX systems to debug C and C++ programs.
- GNU Debugger helps as in getting information about the following:

1. If a core dump happened, then what statement or expression did the program crash on?
2. If an error occurs while executing a function, what line of the program contains the call to that function, and what are the parameters?
3. What are the values of program variables at a particular point during execution of the program?
4. What is the result of a particular expression in a program?

```
(machine-0) machine StartGdbServer 3333
(machine-0)
```

GDB INTEGRATION IN RENODE

```
$ arm-none-eabi-gdb renode-example.elf
GNU gdb (GNU Tools for Arm Embedded Processors 8-2018-q4-major)
8.2.50.20181213-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.
Type "show copying" and "show warranty" for details.
This GDB was configured as "--host=x86_64-apple-darwin10
--target=arm-none-eabi".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
    <http://www.gnu.org/software/gdb/documentation/>.

For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from renode-example.elf...

(gdb) target remote :3333
Remote debugging using :3333
0x00000000 in ?? ()
```

Drawbacks of Renode

- Renode is fully scripting mode of operation which make it less effective in case of user interface and running the peripherals.
- It required the board ELF file for the initiation of the project. This board file is not available open source yet so that we can develop for our specific boards.
- For running the project, we must provide virtual address for the memory else the code will not work as per the need .
- It lack transparency during execution in debugging mode .

COMPARISON BETWEEN RENODE AND ARM FVP

S.NO	FACTORS	ARM FVP	RENODE
1.	Architecture	ARMv8-A and ARMv8-M	Data framework
2.	CPU Supported	Cortex-A, Cortex-M and Cortex-R	Cortex-M, Cortex-A,RISC-V
3.	Machine required	Single machine at a time	Multiple machine .
3.	Integration	Keil ide	No integration required
4.	Output Display	Visual Graphic mode	Script mode
5.	Debugging	Using Keil only	Online Gdb server
6.	Commerical Support	ARM	Antmicro