RVX CLI Manual for Linux/Windows

Kyuseung Han, Sukho Lee, Jae-Jin Lee

ETRI, Daejeon, South Korea

v2025-10-23 or later

Contents

1 Overview

- This manual provides instructions for using the Command Line Interface (CLI) after RVX is installed on the user's machine.

2 Notice

- All results produced using RVX are subject to the following conditions:
  - They must not be used beyond the predefined purpose and scope specified in advance for a particular class or research project.
  - They must not be distributed to third parties other than the designated users or organizations.
  - They are free for non-commercial research use, provided that the paper is cited. All other uses require prior approval and a technology transfer agreement.

3 Things to Know

- Manuals are available online - `riscvexpress.github.io`
- Any part starting with # should be replaced or modified according to your environment.
- On Linux, use the bash shell for command-line operations.
- On Windows, use the Windows Power Shell for command-line operations.
- Skills for Linux
- Skills for Windows

4 Platform

4.1 Overview

A platform includes both the hardware and software associated with an SoC. It is organized under a directory that shares the same name as the SoC the user intends to design.

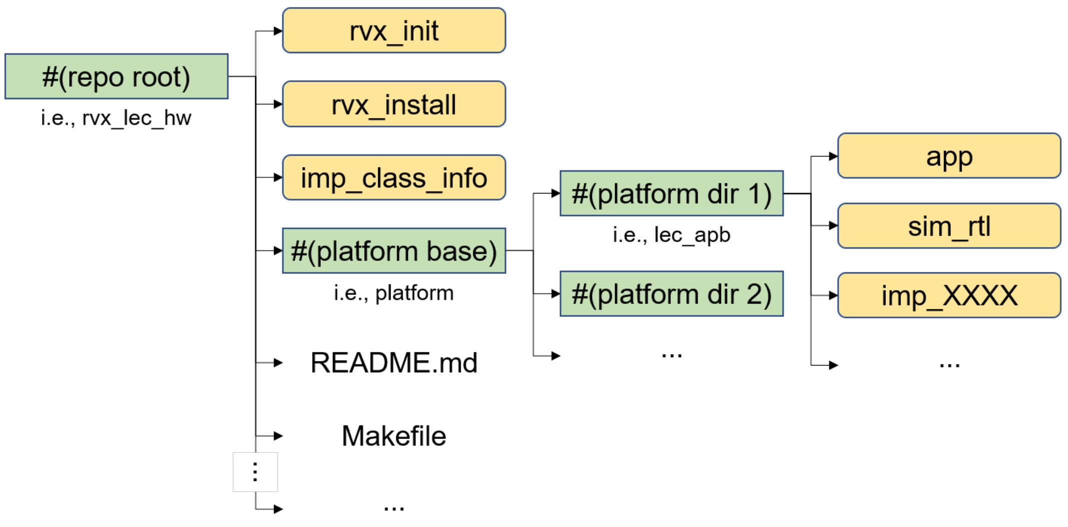


Figure 1: Detailed Structure of the Platform Directory.

In Figure 1, lec_apb is both the name of the SoC and the name of the corresponding directory.

4.2 File Structure

The structure under #(platform dir) is as follows:

Path	Usage	Description
./#(platform name).xml	Editable	SoC description file
./app/	Editable	Application development environment
./user/	Editable	User-managed environment
./util/	Editable	Utility environment
./sim_rtl/	Use Only	RTL simulation environment
./imp_XXXX/	Use Only	FPGA prototyping environment
./arch/	System Reserved	Generated from ./#(platform name).xml
./fpga_component/	System Reserved	Used in imp_XXXX

4.3 Functionality

4.3.1 Creating a New Platform

- Command:

```
cmd) cd #(platform base)
cmd) make new PLATFORM_NAME=#(platform name)
```

- Result:
  #(platform base)/#(platform name) is created,
  which we refer to as #(platform dir).

4.3.2 Designing a Platform

- Prerequisite: Creating a New Platform
- Command:

```
cmd) cd #(platform dir)
inst) Edit the ./#(platform name).xml file with a text editor.
```

4.3.3 Synthesizing a Platform

- Prerequisite: Designing a Platform
- Command:

```
cmd) cd #(platform dir)
cmd) make syn
```

- Result: #(platform dir)/arch is created.
- Note: #(platform dir)/arch is automatically managed by the RVX tool.

4.3.4 Cleaning All Platform Results

The following files and/or directories are maintained:
./#(platform name).xml, ./app, ./user, and ./util

```
cmd) cd #(platform dir)
cmd) make clean
```

5 Application

5.1 Overview

In the application directory, users can develop and compile the application software source code. Execution of the application must be performed using either RTL simulation or FPGA prototyping.

5.2 Build Mode

	debug	release	profile
assert*	O	X	X
assert_must*	O	O	O
printf	O	O	X
printf_must	O	O	O
debug_print*	O	O	O

5.3 Functionality

5.3.1 Creating an Application Directory

- Command:

```
cmd) cd #(platform dir)/app
cmd) make new APP_NAME=#(app name)
```

- Result:
  #(platform dir)/app/#(app name) is created,
  which we refer to as #(app dir).

5.3.2 Developing an Application

The application software must be written in C,
and the source code should be stored in #(app dir)/src.
The location of the source code can be changed by modifying #(app dir).

5.3.3 Building an Application (Including Compiling)

- Prerequisite: Synthesizing a Platform
- Command:

```
cmd) cd #(app dir)
cmd) make rtl
```

- Result: #(app dir)/rtl.release is created.

- Note: #(app dir)/rtl.release is automatically managed by the RVX tool.

5.3.4 Cleaning Build Results

The following files and/or directories are maintained:
./Makefile, ./compile_list, and ./src

```
cmd) cd #(app dir)
cmd) make clean
```

5.4 Auxiliary Functionality

5.4.1 Specifying Compile Options Manually

Specify CFLAGS_RELEASE, CFLAGS_DEBUG, and/or CFLAGS_PROFILE in #(app dir)/rvx_each.mh, depending on your needs.

5.4.2 Specifying Compile Sources Manually

Edit #(app dir)/compile_list.

5.4.3 Updating Makefile after RVX Update

This is not mandatory unless an error occurs.

```
cmd) cd #(app dir)
cmd) make update_makefile
```

6 RTL Simulation

6.1 Overview

This manual is intended for simulating applications on an RTL-based hardware platform. Applications developed in #(app dir) are converted into RISC-V binary files through the build process, and are then loaded into the main memory of the RTL platform at the start of simulation. The simulation automatically terminates when the application's main function returns.
Note that this process requires a license for the mixed-language RTL simulator, which must be obtained separately.

6.2 Simulation Functionality

6.2.1 Creating an RTL Simulation Directory

- Prerequisite: Synthesizing a Platform
- Command:

  ```
  cmd) cd #(platform dir)
  cmd) make sim_rtl
  ```

- Result:
  #(platform dir)/sim_rtl is created,
  which we refer to as #(sim dir).
- Note: #(sim dir) is automatically managed by the RVX tool.

6.2.2 Cleaning All Simulation Results

```
cmd) cd #(sim dir)
cmd) make clean
```

6.2.3 Compiling the Platform RTL Code

```
cmd) cd #(sim dir)
cmd) make compile_test
```

6.2.4 Displaying the Compilation Log of the Platform RTL Code

```
cmd) cd #(sim dir)
cmd) make compile_check
```

6.2.5 Simulating the RTL Platform Using an Application

6.2.6 Common

- Prerequisite: Developing an Application
- Build Mode:

  - The default value is `debug` for all simulation commands.
  - You can explicitly specify `BUILD_MODE` when invoking `make`.
  - Or, can be defined in `#(sim dir)/rvx_each.mh`.
  - Or, can be defined in `#(platform dir)/user/sim/env/set_sim_env.mh`.
- Note:
  - `Build` compiles only the parts that have changed.

6.2.7 Simulating the Platform with an Application

- Included Process:
  - `Building an Application`
  - `Compiling the Platform RTL Code`

```
cmd) cd #(sim dir)
cmd) make #(app name).sim
```

6.2.8 Simulating the Platform Including an Application Rebuild

- Included Process:
  - `Cleaning Build Results`
  - `Building an Application`
  - `Compiling the Platform RTL Code`

```
cmd) cd #(sim dir)
cmd) make #(app name).resim
```

6.2.9 Simulating the Platform from Scratch

- Included Process:
  - `Cleaning All Simulation Results`
  - `Cleaning Build Results`
  - `Building an Application`
  - `Compiling the Platform RTL Code`

```
cmd) cd #(sim dir)
cmd) make #(app name).all // make clean #(app name).resim
```

6.2.10 Simulating with RTL Waveform Recording

- Included Process:
  - Building an Application
  - Compiling the Platform RTL Code

```
cmd) cd #(sim dir)
cmd) make #(app name).debug
```

6.2.11 Simulating with Rebuild and Recording

- Included Process:
  - Cleaning Build Results
  - Building an Application
  - Compiling the Platform RTL Code

```
cmd) cd #(sim dir)
cmd) make #(app name).redebug
```

6.2.12 Simulating with RTL Waveform Recording from Boot

- Included Process:
  - Building an Application
  - Compiling the Platform RTL Code

```
cmd) cd #(sim dir)
cmd) make #(app name).debug_init
```

6.2.13 Opening the Waveform Viewer for Debugging

```
cmd) cd #(sim dir)
cmd) make view
```

6.3 Auxiliary Functionality

6.3.1 Displaying Applications List

```
cmd) cd #(sim dir)
cmd) make app_list
```

6.3.2 Compiling the Platform RTL Code One by One

This command helps identify missing include files.

```
cmd) cd #(sim dir)
cmd) make compile_test_all
```

6.3.3 Identifying RTL Compilation Failures

```
cmd) cd #(sim dir)
cmd) make compile_check
```

7 FPGA Prototyping

7.1 Overview

- This manual is intended for validating applications on an FPGA platform.
- This process requires an FPGA board compatible with RVX (Supported Boards).
- This process also requires a license for AMD Vivado, which must be obtained separately.

7.2 Prototyping Functionality

7.2.1 Creating a Prototyping Directory

- Prerequisite: Synthesizing a Platform
- Command:

```
cmd) cd #(platform dir)
cmd) make #(FPGA name) // i.e., make arty-100t
```

- Result:
  #(platform dir)/imp_#(fpga name)_#(date) is created,
  which we refer to as #(fpga dir).
- Note: #(fpga dir) is automatically managed by the RVX tool.

7.2.2 Creating a Vivado Project

- Command:

```
cmd) cd #(fpga dir)
cmd) make project
```

7.2.3 Generating an FPGA Bitstream

- Included Process: Creating a Vivado Project
- Command:

```
cmd) cd #(fpga dir)
cmd) make imp
```

- Result:
  A bitstream and reports are generated in the #(fpga dir)/imp_result directory.

7.2.4 Cleaning All Vivado Results

```
cmd) cd #(fpga dir)
cmd) make clean
```

7.3 Validating Functionality

For the following functionalities, you must power on the FPGA board and connect it to your computer.

7.3.1 Programming the FPGA with the Generated Bitstream

- Prerequisite:
  - Generating an FPGA Bitstream
  - An FPGA board connected to your computer - Manual
- Command:

```
cmd) cd #(fpga dir)
cmd) make program
```

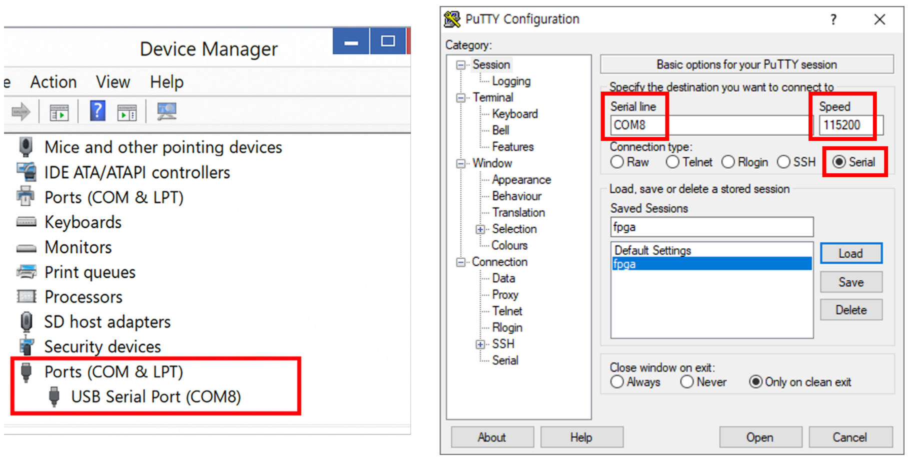
7.3.2 Opening the Terminal to View printf Output

- Prerequisite: Programming the FPGA with the Generated Bitstream

7.3.3 Command @ Linux

```
cmd) cd #(fpga dir)
cmd) make printf
```

7.3.4 Command @ Windows



Serial line: COM8
Speed: 115200
Connection type: ○ Raw ○ Telnet ○ Rlogin ○ SSH [✓ Serial]
Saved Sessions: fpga
Close window on exit: ○ Always ○ Never [✓ Only on clean exit]

Figure 2: Device Manager and PuTTY.

```
cmd) cd #(fpga dir)
cmd) make printf
> Device Manager and PuTTY will be launched.
inst) Check the USB Serial Port number in Device Manager.
> In Figure 2, the port number is COM8.
inst) On PuTTY, configure the settings as highlighted by the three red boxes
in Figure 2.
> If you save the session, you can reuse these settings later.
inst) Open
```

7.3.5 Running an Application on the FPGA Prototype

7.3.6 Common

- Prerequisite:
  - Developing an Application
  - Programming the FPGA with the Generated Bitstream
  - Opening the Terminal to View printf Output
- Note:
  - Build compiles only the parts that have changed.
  - Make sure all the sliding switches of the FPGA board are set to the down position.

7.3.7 Running an Application

- Included Process:
  - Building an Application with BUILD_MODE=debug

```
cmd) cd #(fpga dir)
cmd) make #(app name).run
```

7.3.8 Running an Application with a Rebuild

- Included Process:
  - Cleaning Build Results
  - Building an Application with BUILD_MODE=debug

```
cmd) cd #(fpga dir)
cmd) make #(app name).rerun
```

7.3.9 Running an Application with Optimization

- Included Process:
  - Building an Application with BUILD_MODE=release

```
cmd) cd #(fpga dir)
cmd) make #(app name).opt
```

7.3.10 Running an Optimized Application with a Rebuild

- Included Process:
  - Cleaning Build Results
  - Building an Application with BUILD_MODE=release

```
cmd) cd #(fpga dir)
cmd) make #(app name).reopt
```

7.3.11 Running an Application with Profiling

- Included Process:
  - Cleaning Build Results
  - Building an Application with BUILD_MODE=profile

```
cmd) cd #(fpga dir)
cmd) make #(app name).profile
```

7.4 Auxiliary Functionality

7.4.1 Open a Vivado Project

- Prerequisite: Creating a Vivado Project
- Command:

```
cmd) cd #(fpga dir)
cmd) make open_project
```

7.4.2 Displaying Available FPGA List

```
cmd) cd #(platform dir)
cmd) make fpga_list
```

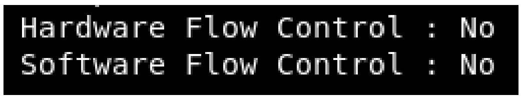
7.4.3 Displaying Applications List

```
cmd) cd #(fpga dir)
cmd) make app_list
```

7.4.4 Deleting All Prototyping Directories

```
cmd) cd #(platform dir)
cmd) make clean_imp
```

7.4.5 Enabling Keyboard Input in Minicom



```
Hardware Flow Control : No
Software Flow Control : No
```

Figure 3: Minicom Setup.

```
cmd) sudo minicom -s
inst) Select ‘‘Serial port setup’’.
inst) Configure the settings as shown in Figure 3.
```

8 Navigate

- Home