DC Charging Communication with RISE-V2G and Raspi-EVSE HAT

Application note

Contents

Wiring Diagram 2

OPEN-PLC-UTILS 2

Step 1: Cloning the Github Repository 2

Step 2: Building the open-plc-utils applications 2

Step 3: Running the EVSE side SLAC application: 3

RISE-V2G 3

Step 1: Cloning the Github Repository 3

Step 2: Java & JDK Installation 3

Step 3: Install maven build environment 4

Step 4: Creating the executable package 4

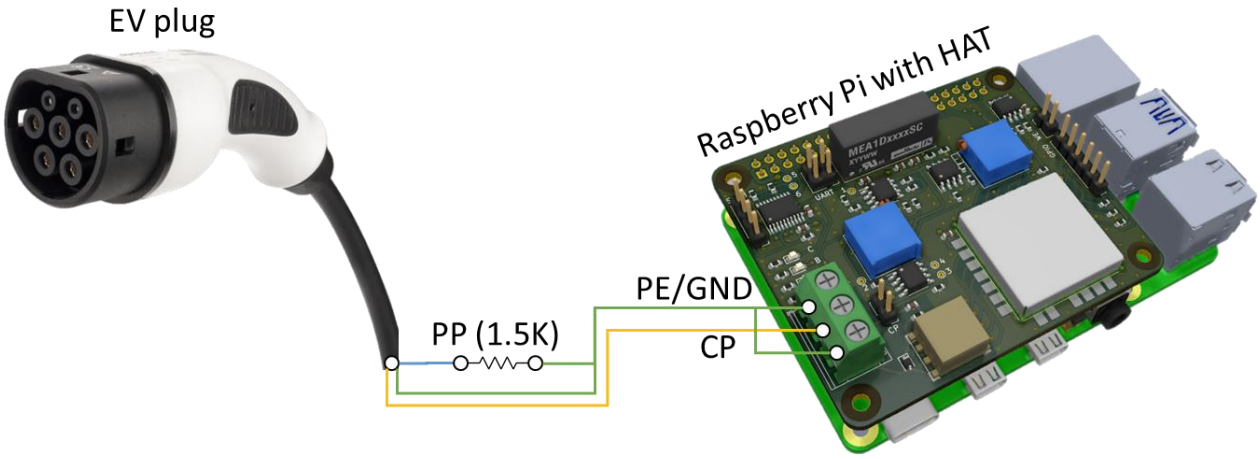
Step 5: Configuring RISE-V2G-SECC 5

Step 6: Running RISE-V2G-SECC 5

Wiring Diagram

See the schematic below on how to wire up the communication lines of the Raspi-EVSE HAT with the European charge plug.

In addition to the CP signal the PP signal must be wired up as seen below with a 1.5k Ohm resistor in order to have the car detect the presence of the charge connector.



OPEN-PLC-UTILS

Here we will get the open-plc-utils ready in order to do SLAC according to ISO-15118/DIN 70121. This establishes the physical link layer between the vehicle and the charging station on the powerline channel.

Step 1: Cloning the Github Repository

```
pi@raspberry:~ $ git clone https://github.com/qca/open-plc-utils.git
Cloning into 'open-plc-utils'...
remote: Enumerating objects: 17765, done.
remote: Counting objects: 100% (52/52), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 17765 (delta 23), reused 20 (delta 11), pack-reused 17713
Receiving objects: 100% (17765/17765), 11.69 MiB | 3.45 MiB/s, done.
Resolving deltas: 100% (13506/13506), done.
```

Step 2: Building the open-plc-utils applications

Because open-plc-utils are mainly based on C, the build instructions follow the usual **make** procedure. So just enter the directory and perform the make command:

```
pi@raspberry:~/open-plc-utils $ make
```

And after this has finished a make install command. If you prefer not to install the binaries the system directories you can also skip this step and execute the open-plc-utils applications directly from the build directories.

```
pi@raspberry:~/open-plc-utils $ sudo make install
```

Step 3: Running the EVSE side SLAC application:

```
pi@raspberry:~/open-plc-utils $ evse -i eth1
evse: evse_cm_set_key: --> CM_SET_KEY.REQ
evse: evse_cm_set_key: <-- CM_SET_KEY.CNF
evse: UnoccupiedState: Listening ...
evse: evse_cm_slac_param: <-- CM_SLAC_PARAM.REQ ?
evse: evse_cm_slac_param: <-- CM_SLAC_PARAM.REQ ?
evse: evse_cm_slac_param: <-- CM_SLAC_PARAM.REQ ?
evse: evse_cm_slac_param: <-- CM_SLAC_PARAM.REQ ?
evse: evse_cm_slac_param: <-- CM_SLAC_PARAM.REQ ?
evse: evse_cm_slac_param: <-- CM_SLAC_PARAM.REQ ?
...
```

Now the EVSE is ready and waiting for incoming SLAC requests of a vehicle. You will see further output detailing the SLAC process once a vehicle is connected.

RISE-V2G

Here we will look at how to get the RISE-V2G stack up and running.

Step 1: Cloning the Github Repository

As usually first we need to get the source files for the application:

```
pi@raspberry:~ $ git clone https://github.com/SwitchEV/RISE-V2G.git
Cloning into 'RISE-V2G'...
remote: Enumerating objects: 4171, done.
remote: Counting objects: 100% (668/668), done.
remote: Compressing objects: 100% (355/355), done.
remote: Total 4171 (delta 358), reused 414 (delta 183), pack-reused 3503
Receiving objects: 100% (4171/4171), 3.42 MiB | 2.40 MiB/s, done.
Resolving deltas: 100% (2413/2413), done.
```

Step 2: Java & JDK Installation

RISE-V2G requires JDK version 8. With the recent JDK version 11 I was not able to compile the RISE-V2G version 1.2.6 without errors. Therefore, it is important to install the JDK as follows:

```
pi@raspberry:~ $ sudo apt install openjdk-8-jdk
```

It might be necessary to do a **`sudo apt update`** followed by **`sudo apt upgrade`** in order to install this package.

After installation check that java is using the JDK 8:

```
pi@raspberry:~ $ java -version
openjdk version "1.8.0_212"
OpenJDK Runtime Environment (build 1.8.0_212-8u212-b01-1+rpi1-b01)
OpenJDK Client VM (build 25.212-b01, mixed mode)
```

In case you see java version 11, it is necessary to switch java to JDK 8 with the following command:

```
pi@raspberry:~ $ sudo update-alternatives --config java
There are 2 choices for the alternative java (providing /usr/bin/java).

  Selection    Path                                            Priority   Status
------------------------------------------------------------
  0            /usr/lib/jvm/java-11-openjdk-armhf/bin/java      1111      auto mode
  1            /usr/lib/jvm/java-11-openjdk-armhf/bin/java      1111      manual mode
* 2            /usr/lib/jvm/java-8-openjdk-armhf/jre/bin/java   1081      manual mode

Press <enter> to keep the current choice[*], or type selection number:
```

And select the java-8-openjdk by entering the corresponding number. After that **`java -version`** should look similar to the above output.

Step 3: Install maven build environment

This is a simple installation from the Raspberry Pi repositories:

```
pi@raspberry:~ $ sudo apt install maven
```

Step 4: Creating the executable package

In order to compile the .JAVA files into an executable .JAR file use the following procedure.

Enter the directory where the maven project file is located:

```
pi@raspberry:~ $ cd RISE-V2G/RISE-V2G-PARENT/
pi@raspberry:~/RISE-V2G/RISE-V2G-PARENT $ ls
pom.xml  src
```

pom.xml is the project file for the complete RISE-V2G package. Now enter:

```
pi@raspberry:~ $ mvn package
Downloaded from central:
https://repo.maven.apache.org/maven2/org/easymock/easymock/3.4/easymock-3.4.jar (474 kB at
35 kB/s)
Downloaded from central:
https://repo.maven.apache.org/maven2/org/osgi/org.osgi.compendium/4.2.0/org.osgi.compendium-
4.2.0.jar (614 kB at 35 kB/s)
...
```

Now maven should start downloading packages and building the various .JAR files. This will take several minutes on the Raspberry Pi.

When it's finished the last lines should see the following output:

```
...
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for rise-v2g-parent 1.2.6:
[INFO]
[INFO] rise-v2g-parent ..................................... SUCCESS [01:27 min]
[INFO] rise-v2g-shared ..................................... SUCCESS [02:40 min]
[INFO] rise-v2g-evcc ....................................... SUCCESS [01:06 min]
[INFO] rise-v2g-secc ....................................... SUCCESS [  5.196 s]
[INFO] ------------------------------------------------------------------------
```

```
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  05:19 min
[INFO] Finished at: 2021-06-21T19:06:01+02:00
[INFO] ------------------------------------------------------------------------
```

Step 5: Configuring RISE-V2G-SECC

In order to let RISE-V2G-SECC project know what settings to use when running the application, the file **`SECCConfig.properties`** should be edited:

```
pi@raspberry:~/RISE-V2G/RISE-V2G-SECC $ nano SECCConfig.properties
```

The most important setting is probably the network interface. This must be changed in order to point to the network interface provided by the PLC modem. Look for the following location in the file and change it to eth1 for the Raspi-EVSE HAT

```
# Network interface
#------------------
#
# The network interface name like en3 or eth1 of the network interface on which to
communicate with the EVCC via a
# link-local IPv6 address
network.interface = eth1
```

Step 6: Running RISE-V2G-SECC

Now finally, you can execute the previously build package for the EVSE side of the RISE-V2G project, also called RISE-V2G-SECC.

It is important that you execute the command from the directory in which the configuration file is located (**`~/RISE-V2G/RISE-V2G-SECC`**), otherwise RISE-V2G will complain that it can't open the configuration file.

```
pi@babspi:~/RISE-V2G/RISE-V2G-SECC $ java -jar target/rise-v2g-secc-1.2.6.jar
2021-06-21T19:14:17,072 INFO [main] UDPServer: UDP server initialized at link-local address
fe80:0:0:0:a34b:1735:7699:342f%eth0 and port 15118
2021-06-21T19:14:17,097 ERROR [main] SecurityUtils: Keystore file location
'./seccKeystore.jks' not found (FileNotFoundException).
2021-06-21T19:14:17,097 ERROR [main] SecurityUtils: Keystore file location
'./seccTruststore.jks' not found (FileNotFoundException).
2021-06-21T19:14:17,406 INFO [main] TLSServer: TLS server initialized at link-local address
fe80:0:0:0:a34b:1735:7699:342f%eth0 and port 64304
2021-06-21T19:14:17,407 INFO [main] TCPServer: TCP server initialized at link-local address
fe80:0:0:0:a34b:1735:7699:342f%eth0 and port 64177
2021-06-21T19:14:18,743 INFO [TLSServerThread] TLSServer: Waiting for new TLS client
connection ...
2021-06-21T19:14:18,743 INFO [TCPServerThread] TCPServer: Waiting for new TCP client
connection ...
```

As you can see the V2G stack is running and a TCP server and a UDP server is ready.

If you get an error regarding the network interface, make sure that the network interface has an IPv6 address already assigned before starting the application.