ARIA Radar Development Kit
User Manual

1 Document Version

Version	Date	Description
1.0	2024/09/04	Issued

2 Introduction

ARIA RDK (*Radar Development Kit*) is a complete suite to evaluate ARIA Radar Devices and/or to develop custom application algorithms to radar data. ARIA RDK allows for the simultaneous data acquisition from one or more ARIA radar devices. Consequently, users may exploit ARIA RDK to develop different applications, ranging from single-device evaluation, to customer's application development, and to advanced applications such as radar tomography.

3 Software Architecture

The system architecture is shown in Fig.1.

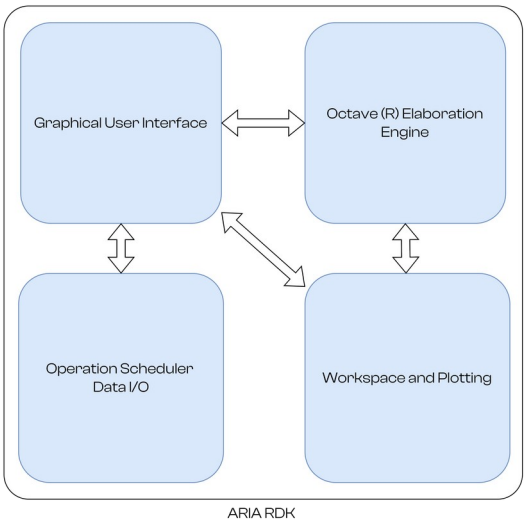


Fig. 1: ARIA RDK Main Architecture

ARIA RDK includes four main components:

1. A Graphical User Interface (Windows and Linux) to create custom projects

2. An interface component to the Scientific Programming Language Octave to run custom algorithms on radar data

3. A Scheduler & Data I/O component to coordinate radar acquisitions and execute user scripts

4. A data workspace management and plotting component

4 Prerequisites

ARIA RDK requires the following components to be compiled and installed in your system:

- Qt6 (>= 6.7.2)
- VTK: The Visualization Toolkit (https://vtk.org/)
- JKQTPlotter: an extensive Qt5 & Qt6 Plotter Framework (https://jkriege2.github.io/JKQtPlotter/index.html)
- Eigen: C++ template library for linear algebra (>= 3.4.0) (https://eigen.tuxfamily.org/)
- Octave (>=9.2.0): a scientific programming language environment

Please refer to respective build/install instruction.

ARIA-RDK is designed to be compiled with CMake®. Currently it has been compiled and tested under Clear Linux SO.

4.1 Building Instructions

The following instructions have been tested in Clear Linux SO, but it is expected full compatibility with other operating systems.

4.1.1 Qt

Install Qt through Qt-Creator self-installer program.

4.1.2 JKQTPlotter

- Download source code:

```
> git clone https://github.com/jkriege2/JKQtPlotter.git
> cd [folder where JKQTPlotter is downloaded]
> mkdir build
> cd build
```

- Start cmake-gui :

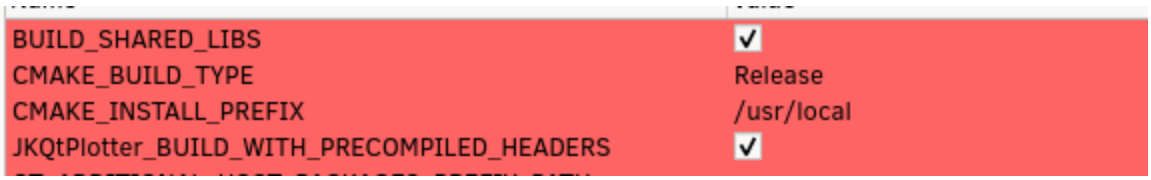
```
>cmake-gui ..
```

- Select the target compiler.

Start Configure and check that Qt6 is found:

Name	Value
QT_DIR	/usr/lib64/cmake/Qt6
Qt6CoreTools_DIR	/usr/lib64/cmake/Qt6CoreTools
Qt6Core_DIR	/usr/lib64/cmake/Qt6Core
Qt6DBusTools_DIR	/usr/lib64/cmake/Qt6DBusTools
Qt6DBus_DIR	/usr/lib64/cmake/Qt6DBus
Qt6GuiTools_DIR	/usr/lib64/cmake/Qt6GuiTools
Qt6Gui_DIR	/usr/lib64/cmake/Qt6Gui
Qt6OpenGLWidgets_DIR	/usr/lib64/cmake/Qt6OpenGLWidgets
Qt6OpenGL_DIR	/usr/lib64/cmake/Qt6OpenGL
Qt6PrintSupport_DIR	/usr/lib64/cmake/Qt6PrintSupport
Qt6Svg_DIR	/usr/lib64/cmake/Qt6Svg
Qt6Test_DIR	/usr/lib64/cmake/Qt6Test
Qt6WidgetsTools_DIR	/usr/lib64/cmake/Qt6WidgetsTools
Qt6Widgets_DIR	/usr/lib64/cmake/Qt6Widgets
Qt6Xml_DIR	/usr/lib64/cmake/Qt6Xml
Qt6_DIR	/usr/lib64/cmake/Qt6

- Verify that "CMAKE_BUILD_TYPE" is set to "Release"
- Set a proper installation folder, the installation folder has to be part of, or included into, OS search path.



- Run *Configure* and *Generate* and build & install:

```
> cmake  - - build .
> sudo cmake - - install .
```

NOTE: during compilation of ARIA-SDK, PrintSupport may not be found. In this case, the compilation is not successful.

To overcome this, check the following "JKQtPlotter_BUILD_FORCE_NO_PRINTER_SUPPORT"

4.1.3 VTK

- Download source archive from https://vtk.org/download/.
- (Linux) Install the following packages

```
build-essential \
cmake-curses-gui \
mesa-common-dev \
mesa-utils \
freeglut3-dev \
```

- Create a folder (e.g. ~/VTK) and unarchive the source code.
- Enter the new folder and mdkir build folder. Enter build folder

```
> cd [folder where VTK source code folder is extracted]
> mkdir build
> cd build
```

- Run the configuration ccmake

```
> ccmake  -S path_to_vtk_source -B path_to_vtk_source/VTK-build
```

- Verify / set the following parameters

```
Qt6CoreTools_DIR                /usr/lib64/cmake/Qt6CoreTools
Qt6Core_DIR                     /usr/lib64/cmake/Qt6Core
Qt6DBusTools_DIR                /usr/lib64/cmake/Qt6DBusTools
Qt6DBus_DIR                     /usr/lib64/cmake/Qt6DBus
Qt6GuiTools_DIR                 /usr/lib64/cmake/Qt6GuiTools
Qt6Gui_DIR                      /usr/lib64/cmake/Qt6Gui
Qt6OpenGLWidgets_DIR            /usr/lib64/cmake/Qt6OpenGLWidgets
Qt6OpenGL_DIR                   /usr/lib64/cmake/Qt6OpenGL
Qt6Sql_DIR                      /usr/lib64/cmake/Qt6Sql
Qt6WidgetsTools_DIR             /usr/lib64/cmake/Qt6WidgetsTools
Qt6Widgets_DIR                  /usr/lib64/cmake/Qt6Widgets
Qt6_DIR                         /usr/lib64/cmake/Qt6
VTK_BUILD_DOCUMENTATION         OFF
VTK_BUILD_EXAMPLES              OFF
VTK_BUILD_SCALED_SOA_ARRAYS     OFF
VTK_BUILD_SPHINX_DOCUMENTATION  OFF
VTK_BUILD_TESTING               OFF
VTK_EXTRA_COMPILER_WARNINGS     OFF
VTK_GROUP_ENABLE_Imaging        WANT
VTK_GROUP_ENABLE_MPI            DONT_WANT
VTK_GROUP_ENABLE_Qt             YES
VTK_GROUP_ENABLE_Rendering      WANT
VTK_GROUP_ENABLE_StandAlone     DONT_WANT
VTK_GROUP_ENABLE_Views          WANT
VTK_GROUP_ENABLE_Web            DONT_WANT
VTK_QT_VERSION                  6
VTK_SMP_IMPLEMENTATION_TYPE     OpenMP
VTK_USE_CUDA                    OFF
VTK_USE_LARGE_DATA              OFF
```

(NB Actual Qt paths may be different)

- Verify that the installation prefix is set to a folder included in the search path, e.g.

```
BUILD_SHARED_LIBS               ON
CMAKE_BUILD_TYPE                Release
CMAKE_INSTALL_PREFIX            /usr/local
```

- Press **c** to configure and **g** to generate
- Go into VTK source folder

```
> cmake --build path_to_vtk_source/VTK-build
> sudo cmake - - install . path_to_vtk_source/VTK-build
```

5 ARIA-RDK Main Concepts

5.1 Workspace

ARIA RDK introduces the concept of workspace, i.e. a folder structure where all the building blocks of a customer application project are defined.

A workspace contains the following elements:

- A set of radar modules definitions: a radar module is a description of one specific UWB module, with the firmware it runs, and its interface with the PC (Fig. 2);
- A set of elaboration scripts: a script is a set of instructions to run customer applications and algorithms. Currently ARIA-RDK support the Octave scientific-programming language;
- A set of radar devices: a device is a physical board attached to the PC. Each device must have its corresponding module definition properly loaded.
- One or more schedulers (handlers): one scheduler is responsible to synchronize more devices to run at the same time, to enable multi-radar acquisition.

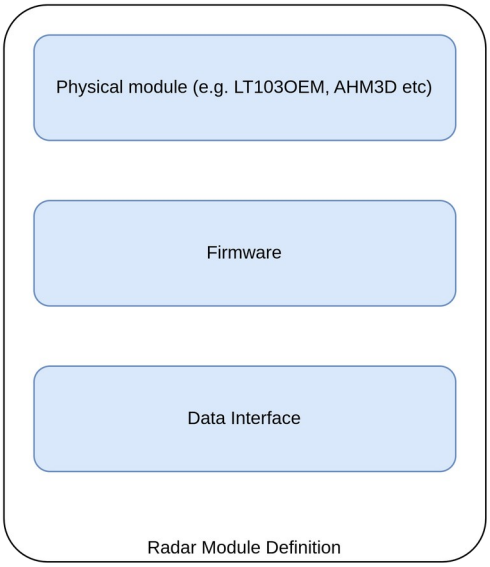


Fig. 2: Radar Module Definition

These components are described in the following sections.

5.2 Radar Modules

One "radar module" is the definition of a peculiar family of radar devices. Each module is uniquely identified by the following data:

1. Its name;
2. A set of parameters which are the radar variables exposed to the PC. They vary according to the radar module and the installed FW. For instance, a module with "basic" FW would provide only the radar's raw-data while the same module with a more advanced FW (e.g. to detect and locate people) would provide the results of the elaboration (e.g. the presence detection outcome and the position of the detected person); see chapter 5.4 for further details.
3. Optional: the definition of the antenna array, with the antennas models and their position to evaluate the beamforming capability;
4. The scripts that are to be executed at start-up, before and after each data acquisition;
5. The configuration of the serial port.

5.3 Radar Devices

A radar device is the physical device connected to the PC. It is the logic representation of a physical device (Fig. 3).

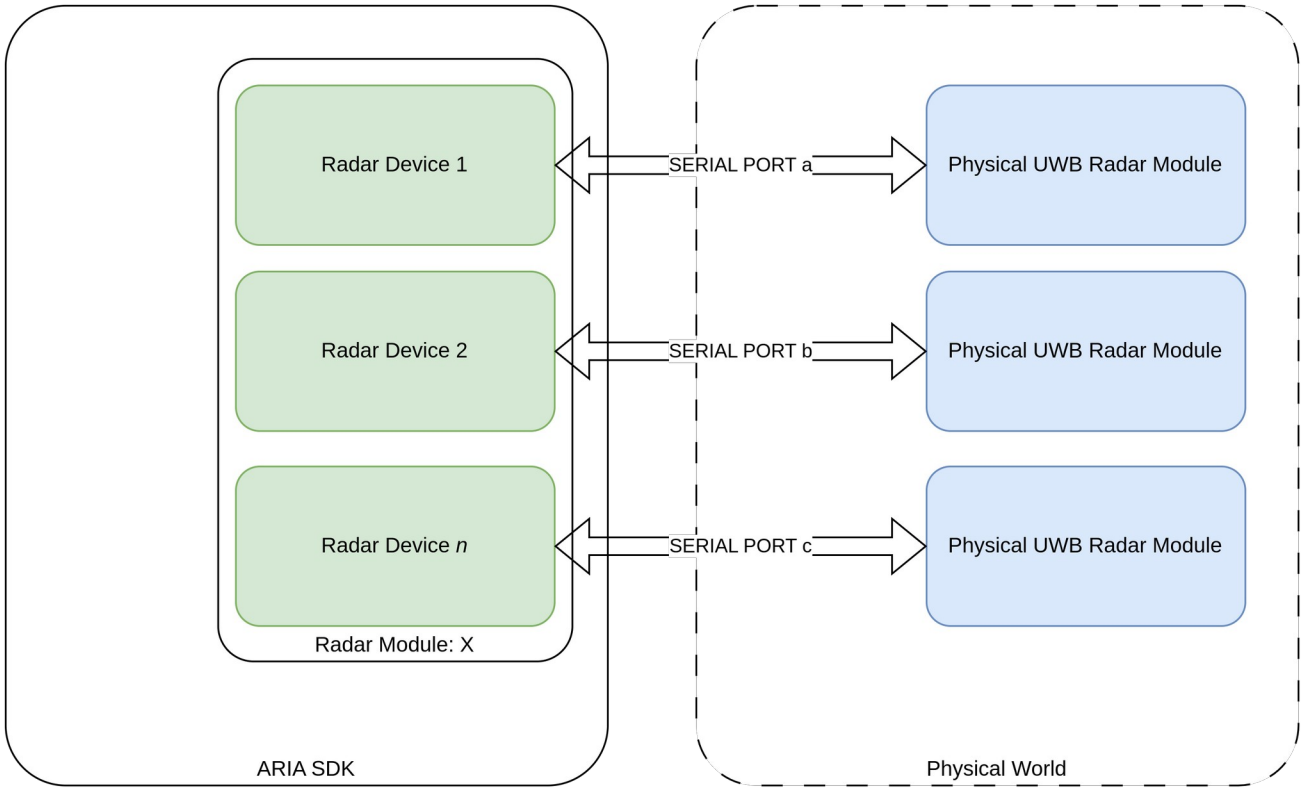


Fig. 3: Logical and Physical Representation of Radar Devices

A radar device is defined by its radar module (from which the parameters are inherited) and a set of scripts that are executed

- at start-up (i.e. before the 1st acquisition cycle)
- before any acquisition cycle
- after any acquisition cycle

See below for details on radar acquisition cycles.

5.3.1 Radar Cycle

A radar cycle is made of 4 steps:

1. init phase: all initialization scripts and variables are set

2.pre-acquisition: all parameters that affect next acquisition are set

3. post-acquisition: this is where customer post-acquisition algorithms are expected

4. frame-rate synchronization

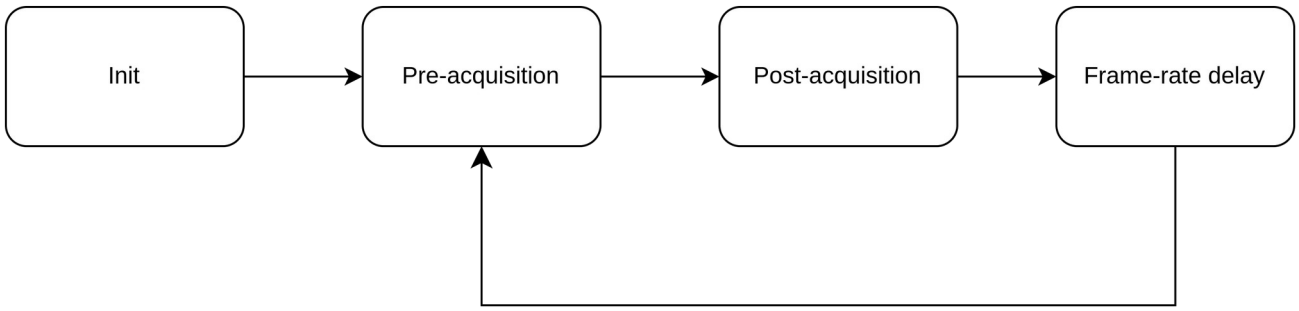


Fig. 4: Basic Radar Cycle

In Fig. 4, the basic radar cycle is shown: after a setup phase ("init"), each acquisition cycle is made of a pre-acquisition phase where the selected scripts are executed and, then, the corresponding parameters are fed into the physical device. When the radar device is able to provide data to the PC and the transfer of the selected parameters is complete, another optional set of scripts is executed (post-acquisition). Please note that the proper acquisition is made by the firmware running into the physical device, so from the RDK perspective we have just two different "moments": before starting an acquisition and after the acquisition has been completed.

If this process (pre-acquisition, acquisition, post-acquisition) is faster than the required refresh rate, a delay is added to maintain the required refresh-rate.

If, instead, the process is longer than the refresh-rate, the behavior of the system is defined into the *scheduler* options. See *Schedulers* for further details.

5.3.2 Policy during init, pre_acquisition and post_acquisition

All phases are further split in 3 different steps:

1. Inquiry of output params and

2. Execution of the scripts related to

3. Update of the params that are Input, I/O and that are modified by Octave

5.3.3 Radar Flow of Operations

Parameter Update

When a parameter is updated, the new value is sent through the serial port to the physical device. It may happen that the actual value stored inside the physical radar differs slightly from

5.4 Parameters

A parameter is a variable that is exposed from the physical device / firmware. ARIA RDK can therefore inquiry for actual value(s) of a parameter, or set its desired value(s).

Basically, each device parameter in ARIA-RDK corresponds to an internal variable in the corresponding radar boards.

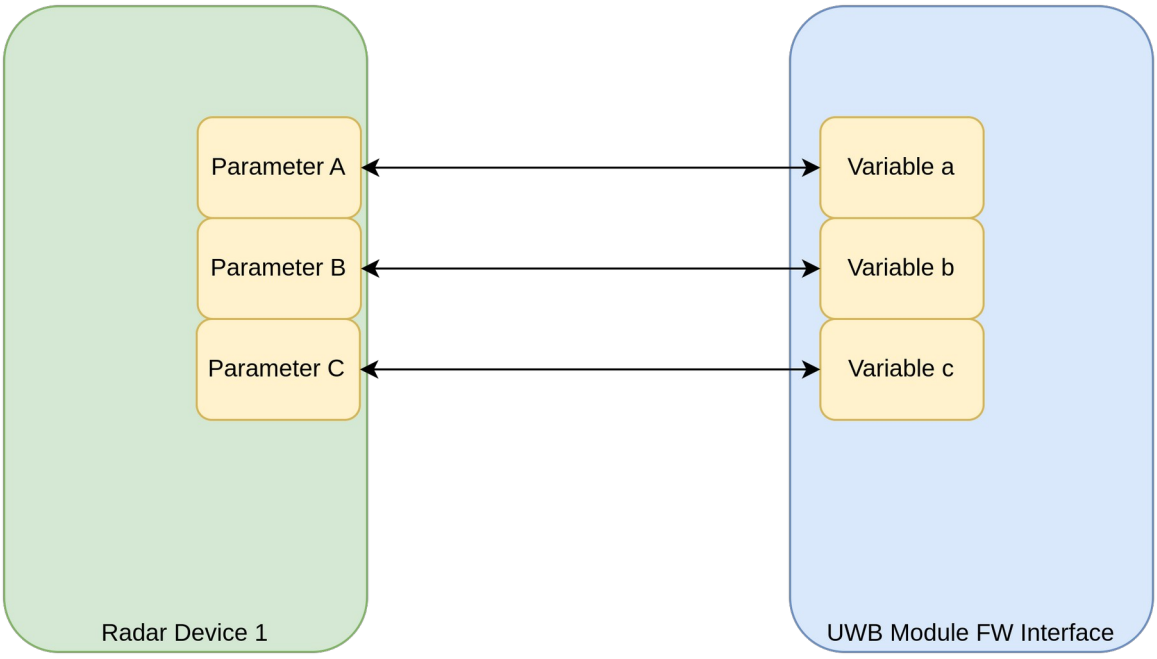


Fig. 5: Parameters and link to physical module's FW variables

5.4.1 Name

Each parameter is defined by a unique, non empty name.

5.4.2 Value

The value of the parameter is the copy of the value of the corresponding variable in the FW of a radar device; clearly, a parameter (and its value) is meaningful only if it is exposed from the radar firmware. Different variables may have different value types. Consequently, the associated parameter defined in ARIA-SDK must contain this information to guarantee proper data transfer/interpretation.

Parameters' values may be of different types, these types are enlisted as follows:

	Description	Data size (bytes)
void	Parameter without data. This kind of parameters are, in fact, pure commands	0
int8,uint8	8-bit integer (unsigned) value or vector	1 * number of elements
int16,uint16	16-bit integer (unsigned) value or vector	2 * number of elements
int32,uint32	32-bit integer (unsigned) value or vector	4 * number of elements
float	32-bit float value or vector	4 * number of elements
char	String or single char	1 * number of elements
enum	Set of available uint8 values with a string descriptor	1 * number of elements

Any parameter (except the "void" case) can be a single value or an array. In case of an *enum* parameter, a parameter value belongs to a set of valid values. Each valid value is described by a unique string. Each element of the valid set is a pair <int8_t, string> where the first part (int8_t field) is the data effectively passed to/from the physical device. The second part (string field) is just a descriptor of the corresponding value.

As an example, the TX POWER of the LT102V2 can have one of the following values:

Value	Descriptor
0x00	TX POWER OFF
0x01	TX LOW POWER
0x02	TX MID POWER
0x03	TX HIGH POWER

ARIA UWB Radar Devices share a common protocol where the radar physical device returns the actual value of a parameter as part of the "command acknowledge" return packet. Basically, when setting the value of a parameter, the radar physical device returns the same command with the value of the corresponding variable after the modification has taken place.

This has a dual objective:

1. It may be used as a confirmation message that the transfer has been successfully interpreted by the radar device

2. Some variables are modified in the radar device according to the desired value but the final value may be different from the desired one for different reasons: for instance, when setting the desired maximum range the LT102V2 rounds the actual value to the closest round-trip distance, sampled according to the sampling speed. With this return value one can check how a device responds to a variable modification request.

5.4.3 Transfer Direction

A parameter may be defined as:

- Input parameter: the value is set only by the ARIA-SDK.
- Input/Output parameters: values can be defined either by the SDK or by the radar device
- Output parameter: values can be defined only by the physical device. The straightforward example is the current radar raw-data. These parameters do not have a value to be attached to the command sent from the ARIA-RDK to the device. Moreover, they don't have an inquiry value.

5.4.4 Valid values

Each parameter (aside from VOID, which has no value associated, and from *enum* type) value can optionally be limited by:

1. a valid min-max interval (a value is valid if and only if it is greater than or equal to the min value and lower than or equal to the max value)

2. a set of valid values (actual value is valid if and only if it belong to said set)

5.4.5 Command String

This is the string that is sent to the physical device through the serial port for parameters inquiry / update.

5.4.6 Inquiry Value

Ideally, when a variable's value is modified by the FW running in a physical module, the corresponding parameter in ARIA-SDK should reflect the actual value.

To reduce the number of transfers over the serial port, ARIA-SDK is responsible for asking the actual value of a parameter, when needed.

To interrogate the actual value of a variable, the corresponding parameter is provided with a "special value". This peculiar value is called the "Inquiry Value". When sending the "inquiry-value" to the radar physical device, the latter does not change the variable corresponding to the parameter. Instead, it returns the actual value of the internal variable.

5.4.7 Command String

This is the string sent along with the payload (either the set-value or the inquiry-value) to allow the radar physical device to identify which parameter/variable is involved in the transfer.

5.4.8 Grouped Values

Some parameters are grouped in several sub-parameters and transferred at once at every single command.

For instance the radar version for LT102V2 is defined by 3 fields which are handled as 3 different parameters

- Version (reserved) [uint16]
- Version_hw_id [uint16] which is a word describing the HW board
- Version_fw_id [uint32] which is a word describing the FW identifier

6 Data Workspace

7 Scripts

A script is a set of instructions to be performed on radar or application data. The underlying elaboration tool and interface is Octave.

ARIA-RDK provide an active link to/from the Octave interpreter; this allows for great flexibility in algorithms description.

8 License

ARIA-RDK is distributed with a LGPL-3.0 License.

ARIA-RDK can be is distributed for the evaluation of algorithms based on ARIA UWB Radars SoCs and Modules. It can be used freely, but WITHOUT ANY WARRANTY; without even the implied warranty of FITNESS FOR A PARTICULAR PURPOSE.