USER MANUAL

Quside QRNG Library version 2.0.0

Outline

REVISION HISTORY

Date	Version	Description
20/05/2022	1.0	Initial version
01/08/2022	1.1	Added function ETH and installation guide LAL

1. System Requirements

This is the minimum requirements to operate a Quside QRNGs and it depends on the interface.

- The Operating System supported is Windows 10 and Ubuntu 20.04 OS.
- GCC == 9.4.0
- Make == 4.2.1
- The Server has to be connected to the same network L2 domain to which the QRNG is connected.

2. Installation

2.1. Steps

The library is distributed as zip file that contains the following components:

Item	**Description**
User_manual_v1_0	This file explains how to install the required software to operate Quside QRNGs.
Packages to be installed	Linux • QusideQRNGLibraryAdmin-2.0.0.deb • QusideQRNGLibraryUser-2.0.0.deb Windows • QusideQRNGLibraryAdmin-2.0.0.msi • QusideQRNGLibraryUser-2.0.0.msi
Example	This folder contains an example of how to capture extracted random data and a makefile to compile the code for Linux OS. To use this example, it is mandatory first to install the Quside QRNG Library.

Before installing the Windows or Linux libraries, it is necessary to uninstall the installed version if it exists. To install the library on the host, execute the following command:

```
sudo dpkg -i QusideQRNGLibraryAdmin-2.0.0.deb
```

Instead, to install the library on Windows systems, it is necessary to have to double-click on the installer.

2.2. Notes

After the installation, it is required to reboot the System.

3. User modes

The Library can operate in two different modes: user and admin. The libraries for each user type are distributed in separate .zip files. Both libraries can be installed on the same host Server but they cannot run simultaneously, except on Windows where it is not possible to have both user modes installed.

Operating mode	**Description**
User (libqusideQRNGuser)	The User mode allows for the request of random data from the QRNG, including raw data and extracted data.
Admin (libqusideQRNGadmin)	The Admin mode includes all features of the User mode as well as advanced monitoring features, including Randomness Metrology and hardware status.

4. Entropy Sources

The Quside™ ETH library can operate with different quantum entropy source modules from Quside. Some functions may not be available for different quantum entropy source products. In this document we indicate which functions are available depending on the product you acquired.

5. Function description

This section describes the Quside software to control the entropy source through ethernet. The following functions are available in all user modes

- **int connectToServer (char *serverIP):**
  Tries to initiate the communication between the QRNG server and the host.
  - Parameters:
    - serverIP: IP address of the server.
  - Returns:
    - If it success 0, otherwise returns -1.

- **void disconnectServer (void):**
  Tries to disconnect the communication between the QRNG server and the host.

- **int get_random(uint32_t* mem_slot, const size_t Nuint32, const uint32_t devInd):**
  This function capture N bytes (multiple of 1024 in case of using QRNG PCIe) of extracted random numbers into mem_slot pointer and return total random numbers captured.
  - Parameters:
    - mem_slot: pointer to the memory region where the numbers are saved.
    - Nuint32: count of random numbers to capture in bytes.
    - devInd: index of the device to use.
  - Return:
    - If it success 0, otherwise returns -1.

- **int get_raw(uint32_t* mem_slot, const size_t Nuint32, const uint32_t devInd):**
  This function capture N bytes (multiple of 1024 in case of using QRNG PCIe) of raw random numbers into mem_slot pointer and return the total random numbers captured.
  - Parameters:
    - mem_slot: pointer to the memory region where the numbers are saved.
    - Nuint32: count of random numbers to capture in bytes.
    - devInd: index of the device to control.
  - Return:
    - If it success 0, otherwise returns -1.

The following functions can be used by Admin mode:

- **void reset (void):**
  Reset system.

- **int monitor_read_temperature(const uint16_t devInd, float* temp):**
  This function read the temperature in ºC.
  - Parameters:
    - devInd: index of the device to control.
    - temp: variable that will contain the temperature value of QRNG module.
  - Return:
    - If it success 0, otherwise returns -1.

- **int get_laser_temperatures(const uint16_t devInd, float** temp, int* nTemps):**
  Reads voltage of the ressistors that heat the laser.
  - Parameters:
    - devInd: index of the device ID in the devices list.
    - temp: will contain the temperature values.
    - nTemps: number of temperature values returned.
  - Returns:
    - If it success 0, otherwise returns -1.

- **int monitor_read_supply_voltage(const uint16_t devInd, float** vcc, int* nVCCs):**
  This function read the VCC (Power supply) in Volts. ONLY AVAILABLE FOR THE FMC-400 ENTROPY SOURCE MODULE.
  - Parameters:
    - devInd: index of the device to control.
    - vcc: variable that will contain the VCC values.
    - nVCCs: number of VCC values returned.
  - Return:
    - If it success 0, otherwise returns -1.

- **int monitor_read_bias_monitor(const uint16_t devInd, float** bias, int* nBias):**
  This function reads the current bias (initial operating condition to operate correctly) monitor in Amps.
  - Parameters:
    - devInd: index of the device to control.
    - bias: variable that will contain the bias values.
    - nBias: number of bias values returned.

- Return:
  - If it success 0, otherwise returns -1.

- **int monitor_read_optical_power(const uint16_t devInd, float** opPwr, int* nOpPwrs):**
  This function read the optical power in dBm.
  - Parameters:
    - devInd: index of the device to control.
    - opPwr: variable that will contain the optical power values.
    - nOpPwrs: number of optical power values returned.
  - Return:
    - 0 if success, otherwisde -1.

- **int get_laser_status(const uint16_t devInd, int** laserStatus, int* nLasers):**
  Gets the laser status.
  - Parameters:
    - devInd: index of the device ID in the devices list.
    - laserStatus: will contain the laser status values.
    - nLasers: number of lasers status values returned.
  - Returns:
    - If it success 0, otherwise returns -1.

- **int get_Vcomp(const uint16_t devInd, float* vComp):**
  Gets voltage comparator in volts.
  - Parameters:
    - devInd: index of the device ID in the devices list.
    - vComp: will contain the comparator voltage value.
  - Returns:
    - If it success 0, otherwise returns -1.

- **int quality_Qfactor(const uint16_t devInd, float* qFactor):**
  Calculates Q Factor. This value is an statistical calculation of the quantic quality based on the running average of the output and the correlators.
  - Parameters:
    - devInd: index of the device to control.
    - qFactor: this variable will cotain the value of the qFactor.
  - Returns:
    - If it success 0, otherwise returns -1.

- **int get_hmin(const uint16_t devInd, float* hMin):**

Gets the minimum entropy of the system. This value only changes after calibration.
  - Parameters:
    - devInd: index of the device ID in the devices list.
    - hMin: will contain the minimum entropy of the system.
  - Returns:
    - If it success 0, otherwise returns -1.

- **int get_calibration_status(const uint16_t devInd, calibrationStatus* status):**
  Gets the QRNG calibration status.
  - Parameters:
    - devInd: index of the device ID in the devices list.
    - status: this variable will contain the current status of the calibration:
      - DEFAULT(0)
      - CALIBRATING(1)
      - CALIB_SUCCED(2)
      - CALIB_FAIL(3)
      - I2C_ERROR(4)

      These values can differ when using the FMC-400 entropy source module or the FMC-ONE entropy source module.
  - Returns:
    - If it success 0, otherwise returns -1.

- **int check_thresholds(const uint16_t devInd):**
  Checks if the monitors of the system are between the correct operational limits.
  - Parameters:
    - devInd: index of the device ID in the devices list.
  - Returns:
    - If it success 0, otherwise returns -1.
- **int get_monitor_value(const alarmType at, size_t num, const uint16_t devInd):**
  Gets the value of the specific monitor.
  - Parameters:
    - devInd: index of the device ID in the devices list.
    - at: enumerate tha specifies the monitor to check.
    - num: returns the number of measured values related to the given monitor.
  - Returns:
    - If it success 0, otherwise returns -1.

- **int set_monitor_enable(const alarmType at, bool enable, const uint16_t devInd):**
  Sets the monitor status, true means that the check_thresholds function will evaluate the value of the monitor, otherwisde no.
  - Parameters:
    - devInd: index of the device ID in the devices list.
    - at: enumerate that specifies the monitor to set.
    - enable: value to set to the monitor status.
  - Returns:
    - If it success 0, otherwise returns -1.

- **int update_thresholds(const uint16_t devInd):**
  Updates the thresholds of the monitors. This function must be called after calibration of the system and usually called in a regular period(get_delta_t) to maintain the alarms updated.
  - Parameters:
    - devInd: index of the device ID in the devices list.
  - Returns:
    - If it success 0, otherwise returns -1.

- **int get_num_lasers(void)**
  Returns the number of QRNG lasers in the entropy source module.
  - Returns:
    - Number of lasers.

- **size_t get_Delta_t (void):**
  Gets the time that the library should wait between different calls of updateThresholds function. Default is 10 seconds.
  - Returns:
    - Time in seconds.

- **void setTimeout (int seconds):**
  Set timeout for client connection in seconds. Default is 300 seconds.
  - Parameters:
    - seconds: timeout.

6. Monitors

This is the list of the monitors that the library allows Admin mode to check:

Monitor	**Description**	**Function name**	**Compatibility**	**Units**
Board temperature	Measures the PCB temperature	getTemperature	FMC-400 FMC-ONE*1	ºC
VCC	Reads the power supply voltage of the PCB	getVCC	FMC-400	Volts
VTC	Read/set the temperature of the lasers.	getLaserTemperatures	FMC-ONE	Volts
Bias monitor	Read bias current applied to the laser(s).	getBiasMonitor	FMC-400 FMC-ONE	Amps
Optical power	Read the optical power emitted by the laser through internal monitors.	getOpticalPower	FMC-400 FMC-ONE*2	dBm (decibel , mW)
Comparator voltage	Read/set the comparator threshold used to digitize the quantum signal.	getVcomp	FMC-400 FMC-ONE	Volts
Laser status	Checks the on/off status of the laser.	getLaserStatus	FMC-400 FMC-ONE	-
Qfactor	Quality factor based on statistical metrics (bias and correlations).	getQFactor	FMC-400 FMC-ONE	-
Hmin	Min-entropy estimate derived from the physical model of the QRNG.	getHmin	FMC-400*4 FMC- ONE*3, 4	-
Calibration	Start calibration (set) and obtain calibration (get) status of the QRNG.	getCalibrationStatus	FMC-400*5 FMC-ONE*6	-

Notes:

1. *Only available after calibration.*
2. *Not all FMC-ONE models have this monitor available. If the user tries to get the value when the monitor is not available, a zero is obtained.*
3. *This monitor is not currently available, so if this function is called, a zero will be returned.*
4. *Hmin is calculated right after calibration. If a new Hmin value is sought, calibrate again.*
5. *The calibration time is currently ~2 minutes.*
6. *The calibration time is currently ~8 minutes.*

Also, the monitors provide two additional functions (described previously):

- check_thresholds
- update_thresholds

With these two functions, the quality control of the QRNG can be configured, including setting the values at which the system flags a warning or halts.

It is necessary to differentiate between **absolute thresholds** and **relative thresholds**. Absolute thresholds are the ones that, should the measurements of the monitors be out of such ranges, the system is halted immediately, as the quality of the QRNG cannot be guaranteed. A Server shutdown may be required for the PCIe QRNG Series. In contrast, the relative thresholds are those that track the fluctuations of the signals and track they are within expected statistical ranges.

Examples

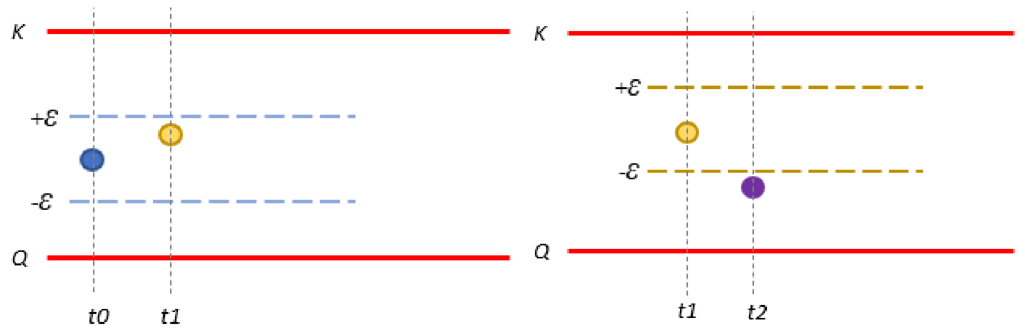


Figure. (Left) Thresholds functionality t1. In red absolute values (set by constants Q and K) and blue dashed represent relative thresholds evaluated at t0. The parameter ε represents the expected statistical range for subsequent samples and defines the relative thresholds. In this example, with the values taken at t0 (blue circle) and t1 (yellow circle), none of the alarms will be triggered. (Right)_Figure 2. In this case, the value taken at t2 is within the absolute range, but outside the relative range, meaning this value changed faster than expected. An alarm will be flagged on relative thresholds in this case.

To update the **relative thresholds** it is necessary to call the **updateThresholds** function. To check if all the monitors are between the **relative and absolute thresholds**, the **checkThresholds** function should be used instead. The alarms are accessible through the library using the following functions and data structures.

Defined data structures

```
alarmType
```

Description:

Enum that contains the alarms that can be accessed and user.

Parameters:

- LASER_STATUS

- LASER_TEMP
- OPTICAL_PW
- BIAS_MON
- TEMP
- VCC
- VCOMP
- QFACTOR
- SYSTEM_CALIBRATED

```
monitorValue
```

Description:

Array containing the possible values of the alarms. When a monitor is within the relative thresholds and the system is calibrated, the return value is OK = 0. When the monitor is outside of the relative thresholds, the return value will be either LOW_VALUE = -1 or HIGH_VALUE = -2. When the system is not calibrated, the return value is OFF = -3. Finally, when a system is outside of the absolute thresholds, the return value is OUT_OF_SECURE_RANGE = -4.

Parameters:

- OK = 0
- LOW_VALUE = -1
- HIGH_VALUE = -2
- OFF = -3
- OUT_OF_SECURE_RANGE = -4

Functions

```
monitorValue* getMonitorValue(const alarmType at, size_t* num, const uint16_t devInd).
```

Description:

Access to the alarm and monitor values of the QRNG system. We define two structures, alarmType and monitorValue, which are defined as:

Parameters:

- at: alarmType
- num: returns the number alarms of this type the system contains.
- devInd: is an input parameter to indicates the ID of the monitored device

Return:

- **monitorValue array**

```
Void setMonitorEnable(const alarmType at, const bool enable, const uint16_t devInd):
```

Description:

Enable and disable the alarm system.

Parameters:

- at: alarmType
- enable: indicates if alarm is enabled (True) or disabled (False)
- devInd: is an input parameter to indicates the ID of the monitored device

7. User Modes

This section shows the mandatory functions required to call for each user to operate the libraries properly.

6.1. USER

```
/* QRNG IP address. This value must be changed to the IP address assigned to the
QRNG. */
char *qrngIP = "192.168.11.5";

/* Connection with the QRNG device. */
if(connectToServer(qrngIP) <= 0) {
    puts("Connection error.");
    return -1;
}

/* In the ethernet connection there is only one device connected and 0 is assigned.
*/
const int devIndex = 0;
/* ... */
/* your code */
/* ... */

/* It is mandatory to execute this function to close the connection with the QRNG
device. */
disconnectServer();
```

6.2. ADMIN

```
/* QRNG IP address. This value must be changed to the IP address assigned to the
QRNG. */
char *qrngIP = "192.168.11.5";

/* Connection with the QRNG device. */
if(connectToServer(qrngIP) <= 0) {
    puts("Connection error.");
    return -1;
}
```

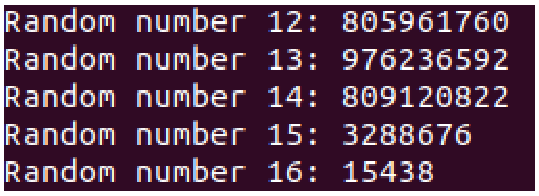
```
/* In the ethernet connection there is only one device connected and 0 is assigned.
*/
const int devIndex = 0;
/* ... */
/* your code */
/* ... */

/* It is mandatory to execute this function to close the connection with the QRNG
device. */
disconnectServer();
```

7. Example

As described in section INSTALLATION, the library is distributed as zip file that contains a folder called Example with an example code to capture extracted data called **quside_QRNG_library_example.c.** To execute this example the following commands, have to be executed in a terminal:

- make all
- ./quside_QRNG_Library_example

With these commands the source file is compiled and executed. If everything went well, the terminal has to show the following although the random numbers captured will be different:



```
Random number 12: 805961760
Random number 13: 976236592
Random number 14: 809120822
Random number 15: 3288676
Random number 16: 15438
```

Figure 1. Result of running the example code.

8. Quside QRNG LAL

There is a software called Quside QRNG LAL (*Library abstract layer*) that implements and abstract interface in Python to manage the library. **This Python library contains the same functions and modes as the C library**, although the Python class initialize mandatory functions as connectToServer and disconnectServer.

8.1. Requirements

The following package must be installed for Quside QRNG LAL to work:

- Numpy >= 1.22.2

8.2. Installation

8.2.1. Steps

The Quside QRNG LAL is distributed as zip file that contains the following components:

- Package to be installed:
  - QusideQRNGLAL-1.0.26.py3-none-any.whl
  - QusideQRNGLALAdmin-1.0.26.py3-none-any.whl
- Example This folder contains an example of how to capture extracted random data. To use this example, it is mandatory first to install the Quside QRNG Library.

As well as C library, each mode is distributed in a separate zip file with different name that can be installed on the same Server but cannot be run at the same time. To install the Quside QRNG LAL on the system it is necessary to execute the following command inside the folder in which the installation package mentioned above is located:

- User mode:

  python3 -m pip install QusideQRNGLAL-1.0.26.py3-none-any.whl

- Admin mode:

  python3 -m pip install QusideQRNGLALAdmin-1.0.26.py3-none-any.whl

8.3. User modes

8.3.1. User

```
# Import Quside QRNG LAL
from QusideQRNGLAL.quside_QRNG_LAL import QusideQRNGLAL

/* QRNG IP address. This value must be changed to the IP address assigned to the QRNG.
*/
qrngIP = “192.168.11.5”;

/* Initialize class and connection with the QRNG device. */
lib = QusideQRNGLAL(ip=qrngIP)

/* In the ethernet connection there is only one device connected and 0 is assigned.
*/
const int devIndex = 0;
/* ... */
/* your code */
/* ... */

/* It is mandatory to execute this function to close the connection with the QRNG
device. */
disconnectServer();
```

8.3.2. Admin

```
# Import Quside QRNG LAL Admin
from QusideQRNGLALAdmin.quside_QRNG_LAL_admin import QusideQRNGLALAdmin

/* QRNG IP address. This value must be changed to the IP address assigned to the QRNG.
*/
qrngIP = “192.168.11.5”;

/* Initialize class and connection with the QRNG device. */
lib = QusideQRNGLALAdmin(ip=qrngIP)

/* In the ethernet connection there is only one device connected and 0 is assigned.
*/
const int devIndex = 0;
/* ... */
/* your code */
/* ... */

/* It is mandatory to execute this function to close the connection with the QRNG
device. */
disconnectServer();
```