USER CODE ADAPTATION

to QRNG library version 2.0.0

Outline

REVISION HISTORY

Date	Version	Description
28/07/2022	1.0	Initial version

1. Introduction

In this file is explained the changes that the customer must make in their applications, what are the changes made in the functions and the necessary modification with respect to the previous library, so that the application is compatible with this new version.

2. Initialization

In the new C library has been renamed the header file to include the library within the user application.

	Before	After
Include	`#include "qusideQRNG.h"`	`#include "quside_QRNG_admin.h"`
		`#include "quside_QRNG_user.h"`

Connect	`connectQRNG("192.168.1.12"[1]);`	`if(connectToServer("192.168.1.12"[1]) != 0)`
		`{`
		`  puts("Error connect.");`
		`  return -1;`
		
		`} else {`
		`  puts("Connected");`
		`}`

Disconnect	`disconnectQRNG();`	`disconnectServer();`

The "QusideQRNGLAL" is the new Python library that includes the new features and is explained in the user manual.

	Before	After
Import	`from PythonClient.api_server import APIServer`	`from QusideQRNGLALAdmin.quside_QRNG_LAL_Admin import QusideQRNGLalAdmin`
		`from QusideQRNGLALAUser.quside_QRNG_LAL_User import QusideQRNGLalUser`
Initialization	`api = APIServer(ip=´192.168.1.12´[1])`	`api = QusideQRNGLALAdmin(ip=´192.168.1.12´[1])`
		`api = QusideQRNGLALUser(ip=´192.168.1.12´[1])`

[1] This IP is an example

3. Functions

In this section the functions that have been modified or removed from the new library for both C and Python are listed:

C

- **void connectQRNG(char *serverIP):**
  This function has been renamed and added a returned value:
  - **int connectToServer(char *serverIP):**
    Returns an integer value that if it success returns 0, otherwise -1.

- **void disconnectQRNG():**
  This function has been renamed:
  - **void disconnectServer().**

- **int get_random(uint32_t *mem_slot, const size_t Nuint32):**
  This function remains the same, but a parameter is added, and due to the standardization with PCIe it is necessary to pass as parameter the device ID, but in the case of Ethernet the device ID is 0:
  - **int get_random(uint32_t *mem_slot, const size_t Nuint32, const uint16_t devInd).**

- **int get_raw(uint32_t *mem_slot, const size_t Nuint32):**
  This function remains the same, but a parameter is added, and due to the standardization with PCIe it is necessary to pass as parameter the device ID, but in case of Ethernet the device ID is 0.
  - **int get_raw(uint32_t *mem_slot, const size_t Nuint32, const uint16_t devInd).**

- **double monitor_read_temperature():**
  This function keeps the same name but changes the return value and parameters:
  - **int monitor_read_temperature(const uint16_t devInd, float *temp):**
    This function read the temperature at ºC.

    Parameters:

    - devInd: index of the device to control. Due to the standardization with PCIe it is necessary to pass as a parameter the device ID, but in the case of Ethernet the device ID is 0.

- temp: variable that will contain the temperature value of the QRNG module.

Return:

- If it success 0, otherwise returns -1.

- **double monitor_read_supplyVoltage():**
  This function has been renamed and changes the return value and parameters:
  - **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. Due to the standardization with PCIe it is necessary to pass as a parameter the device ID, but in the case of Ethernet the device ID is 0.
    - vcc: variable that will contain the VCC values.
    - nVCCs: number of VCC values returned.

    Return:

    - If it success 0, otherwise returns -1.

- **double monitor_read_opticalPower():**
  This function has been renamed and changes the return value and parameters:
  - **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. Due to the standardization with PCIe, it is necessary to pass as a parameter the device ID, but in the case of Ethernet the device ID is 0.
    - opPwr: variable that will contain the optical power values.
    - nOpPwrs: number of optical power values returned.

    Return:

    - 0 if success, otherwise -1.

- **double monitor_read_biasMonitor():**
  This function has been renamed and changes the return value and parameters:
  - **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 milliamps.

    Parameters:

    - devInd: index of the device to control. Due to the standardization with PCIe, it is necessary to pass as a parameter the device ID, but in the case of Ethernet, the device ID is 0.
    - bias: variable that will contain the bias values.
    - nBias: number of bias values returned.

    Return:

    - If it success 0, otherwise returns -1.

- **double quality_Qfactor():**
  This function keeps the same name but changes the return value and parameters:
  - **int quality_Qfactor(const uint16_t devInd, float* qFactor):**
    Calculates Q Factor. This value is a 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. Due to the standardization with PCIe it is necessary to pass as a parameter the device ID, but in the case of Ethernet, the device ID is 0.
    - qFactor: this variable will contain the value of the qFactor.

    Returns:

    - If it success 0, otherwise returns -1.

- **double quality_minEntropyBound():**
  This function has been renamed but changes the return value and parameters:
  - **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 to control. Due to the standardization with PCIe it is necessary to pass as a parameter the device ID, but in the case of Ethernet the device ID is 0.
- hMin: will contain the minimum entropy of the system.

Returns:

- If it success 0, otherwise returns -1.

PYTHON

- **def close():**
  This function has been removed because closing the server has no functionality.

- **def captureToFile():**
  This function has been removed. This functionality and the format in which to save the captured random numbers are left up to the customer application to decide.

- **def captureToArray():**
  This function has been replaced by two functions:
  - **def get_random(num_bytes, devInd):**
    Returns a Numpy array with the extracted random numbers of size 'num_bytes' and due to the standardization with PCIe it is necessary to pass as parameter the device ID, but in the case of Ethernet the device ID is 0.
  - **def get_raw(num_bytes, devInd):**
    Returns a Numpy array with the raw random numbers of size 'num_bytes' and due to the standardization with PCIe it is necessary to pass as parameter the device ID, but in the case of Ethernet the device ID is 0.

- **def readTemperature():**
  This function has been replaced by:
  - **def monitor_read_temperature(devInd):**
    Returns the temperature value (ºC) as float and due to the standardization with PCIe it is necessary to pass as a parameter the device ID, but in the case of ethernet the device ID is 0.

- **def readVcc():**
  This function has been replaced by:
  - **def monitor_read_supply_voltage(devInd):**

Returns the voltage value (V) as a float array and due to the standardization with PCIe it is necessary to pass as a parameter the device ID, but in the case of Ethernet the device ID is 0.

- **def readOpticalPower():**
  This function has been replaced by:
  - **def monitor_read_optical_power(devInd):**
    Returns the optical power value (dBm) as float array and due to the standardization with PCIe it is necessary to pass as a parameter the device ID, but in case of Ethernet the device ID is 0.

- **def readBiasMonitor():**
  This function has been replaced by:
  - **def monitor_read_bias_monitor(devInd):**
    Returns the bias value (mA) as float array and due to the standardization with PCIe it is necessary to pass as parameter the device ID, but in the case of Ethernet the device ID is 0.

- **def readQFactor():**
  This function has been replaced by:
  - **def quality_Qfactor (devInd):**
    Returns the Q Factor value as float and due to the standardization with PCIe it is necessary to pass as parameter the device ID, but in the case of Ethernet the device ID is 0.

4. Example of use

C

CAPTURE

Before

```
uint32_t * mem_slot = NULL, *allocated = NULL;
size_t Nuint32 = 1024;
posix_memalign((void **)&allocated, 4096, (unsigned int)Nuint32 + 4096);
if (!allocated) {
    fprintf(stderr, "OOM %u.\n", (unsigned int)Nuint32 + 4096);
}
mem_slot = allocated + 0;
/* Print number of bytes read. */
printf("Random: %d Bytes\n", get_random(mem_slot, Nuint32));
free(mem_slot);
```

After

```
size_t Nuint32 = 1024;
uint32_t *mem_slot = (uint32_t *)malloc(Nuint32);
uint16_t devInd = 0;
int lenData = get_random(mem_slot, Nuint32, devInd);
printf("Len Data: %d\n", lenData);
free(mem_slot);
```

MONITOR

Temperature

Before

```
printf("Temperature: %f ºC\n", monitor_read_temperature());
```

After

```
float temp;
uint16_t devInd = 0;
monitor_read_temperature(devInd, &temp);
printf("Temp: %f\n", temp);
```

Optical power

Before

```
printf("Optical Power: %f dBm\n", monitor_read_opticalPower());
```

After

```
int nOpPwrs;
float *opPwr = NULL;
uint16_t devInd = 0;
monitor_read_optical_power(devInd, &opPwr, &nOpPwrs);
for(int i = 0; i < nOpPwrs; ++i) {
    printf("OPower: %f\n", opPwr[i]);
}
```

PYTHON

	Before	After
Capture	`buffer = captureToArray(1024)`	`devInd = 0` `buffer = get_random(1024, devInd)`
Monitor	`temp = readTemperature()`	`devInd = 0` `temp = monitor_read_temperature(devInd)`