OpenCPU

Quectel Cellular Engine

OpenCPU Development Guide

OPEN_CPU_DGD_V1.1

Document Title:	OpenCPU Development Guide
Revision:	1.1
Date:	2010-10-27
Status:	Release
Document Control ID:	OPEN_CPU_DGD_V1.1

General Notes

Quectel offers this information as a service to its customers, to support application and engineering efforts that use the products designed by Quectel. The information provided is based upon requirements specifically provided to Quectel by the customers. Quectel has not undertaken any independent search for additional relevant information, including any information that may be in the customer's possession. Furthermore, system validation of this product designed by Quectel within a larger electronic system remains the responsibility of the customer or the customer's system integrator. All specifications supplied herein are subject to change.

Copyright

Contents

0. Revision History

Revision	Date	Author	Description of change
1.00	2009-7-27	Willis YANG	Initial
1.01	2009-9-7	Willis YANG	1.Add API function Ql_Reset. 2.Support controlling 3 UART ports.
	2009-9-7	Jay XIN	1.Remove Pin QL_PINNAME_TXD3 and QL_PINNAME_RXD3. 2.Modify the parameters description of lowpulesnumber and highpulesnumber.
1.02	2009-10-10	Jay XIN	Add file system function interface
	2009-10-23	Jay XIN	Add audio file function interface
	2009-11-6	Jay XIN	Add Tcpip function interface
	2009-11-14	Jay XIN	Add Task manage interface
1.03	2009-11-24	Jay XIN	Add audio stream function interface
	2009-11-27	Willis YANG	1. Add API Ql_SetUartBaudRate. 2.Add API Ql_SetPortOwner 3.Add API Ql_SetUartDCBConfig
1.1	2010-09-17	Stanley YONG	Add: 1. Ql_SendToUart_2() 2. event: EVENT_UARTREADY
	2010-09-28	Stanley YONG	Delete API: Ql_SetUart1dataToQL
	2010-10-8	Stanley YONG	Delete APIs about Flash
	2010-10-13	Stanley.YONG	Add: 1. Ql_SetUartFlowCtrl(); 2. Ql_GetRelativeTime_Counter(); 3. Ql_memTotalLeftSize(); 4. Ql_memMaxCanAllocSize(); 5. Section: Customer Configuration
	2010-10-18	Stanley.YONG	Add FOTA APIs: Ql_Fota_Core_Init,Ql_Fota_Core_Write_Data, Ql_Fota_Core_Finish,Ql_Fota_App_Init, Ql_Fota_App_Write_Data,Ql_Fota_App_Finish, Ql_Fota_Update
	2010-10-25	Stanley.YONG	Add tcpip APIs: 1. Ql_SocketBind 2. Ql_SocketCheckIp 3. Ql_SocketSetSendBufferSize 4. Ql_SocketSetRecvBufferSize 5. Ql_SocketHtonl 6. Ql_SocketHtons
	2010-10-26	Stanley YONG	Consummate and optimize document linguistically and syntactically;

1. Introduction

OpenCPU is a software mechanism which provides a powerful support environment designed to facilitate the development of cost-effective wireless machine-to-machine applications for Quectel M10 and M30 modules. Using standard C language, OpenCPU enables you to develop innovative new applications and embed them directly into Quectel GSM/GPRS modules. It is also easy to port the application software being run on different MCU platform.

1.1 Reference Documents

SN	Document name
[1]	M10 AT Commands Set
[2]	Quectel_OpenCPU_Specification
[3]	Open_CPU_Tutorial

1.2 Glossary

Glossary	Description
Application Task	OpenCPU provides an absolute application task to simulate the MCU environment. Like a normal MCU task, the Application Task has a main entrance and runs forever.
Embedded Application API	Software interfaces developed by Quectel is open to licensed Embedded Application developers. The APIs include audio API, FCM API, system API, periphery API, timer API, and debug API, standard C library API, and so on.
Embedded Application	User created application that utilizes Embedded API functions to interact with Quectel core software. The application can only run on a Quectel product.
Core System	The Core system released by Quectel, which includes the core binary file and Quectel library.
EVENT	Capitalized EVENT notion used in this document is to represent specified system EVENT in Embedded Application.

1.3 Abbreviations

Abbreviation	Description
API	Application Programming Interface
CPU	Central Processing Unit
FCM	Flow Control Manager
FOTA	Firmware Over The Air
KB	Kilobyte
OS	Operating System
PDU	Protocol Data Unit
RAM	Random-Access Memory
ROM	Read-Only Memory
TCP/IP	Transfer Control Protocol / Internet Protocol
UART	Universal Asynchronous Receiver and Transmitter

2. OpenCPU SDK

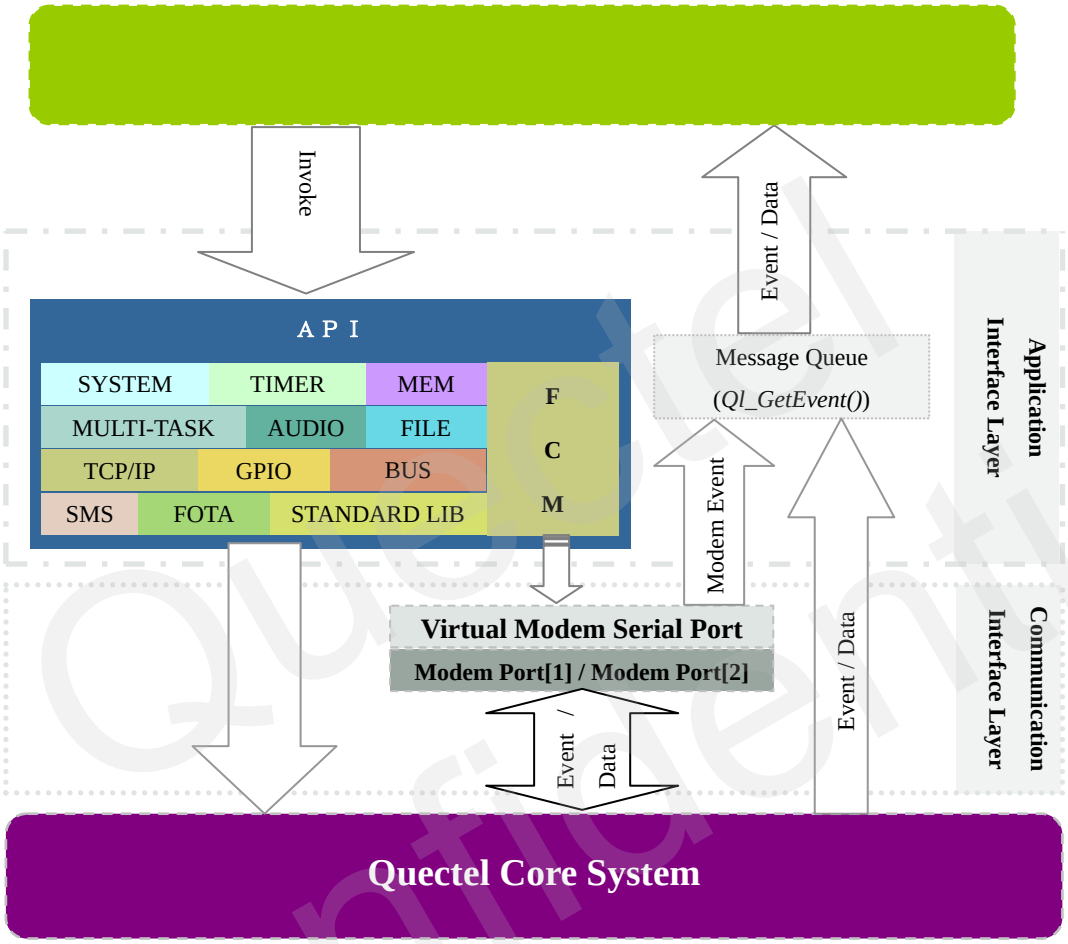
OpenCPU SDK provides some resources as follows for developers:

- Build environment.
- A set of .h header files defines all interface functions and OpenCPU parameters.
- Development guide and other related documents
- Source code for examples.
- Quectel Core System firmware, binary file.
- Download tool for image bin.

3. OpenCPU Software Architecture

3.1. Software architecture

The following figure shows the fundamental principle of OpenCPU software architecture.



When passing information from the Core System to Embedded Application, the *Ql_GetEvent* function catches EVENT then notifies Embedded Application with a parameter of event type.

When need to use the resources of Core System, developers may simply call API functions for the appropriate purposes. Some operations have to dispose the feedback data by *Ql_GetEvent*.

OpenCPU software mechanism appropriately prevents the developers from directly accessing core variables and stack spaces, which insures the developers can safely access to the resources of the Core System, and at the same time, reduces the workload of developing application.

3.2. Application Code Frame

3.2.1. Least Application Code

The following code is the least codes that an Embedded Application requires.

```
/* main function */
QlEventBuffer qlEventBuffer;
bool keepGoing = TRUE;
void ql_entry(void)
{
   while (keepGoing)
   {
      Ql_GetEvent(&qlEventBuffer); //get event from Quectel Core system
      switch(qlEventBuffer.eventTyp)
      {
        //To do: add your code for parsing EVENTs.
        default:
        break;
      }
   }
}
```

ql_entry is the main entrance of Embedded Application.

Ql_GetEvent is the important system interface that the Embedded Application receives events (or messages) from Core System.

3.2.2. Application Entry

The *ql_entry()* is the entry of Embedded Application, just like the main() in C application.

3.2.3. Application Termination

In the entry function of the Embedded Application, the *while* statement keeps the application running. If developers need to terminate the application (actually a task, the main task or a subtask) programmatically, a simple setting, as below, will make it.

***keepGoing* = FALSE.**

3.3. Memory Resources

OpenCPU runs within Real-Time Kernel Task. Developers have to pre-define the size of customer

application call-stack. The stack size of the main task is defined using *QL_TASK_STACK_SIZE* in *ql_customer_config.c*. And please refer to the *QlMutitask* structure to get know of the definition of the stack size for subtask.

Quectel Core System and Embedded Application manage their own RAM regions. Access from one of these programs to the other's RAM region is prohibited, because the operation may cause fatal error.

4. BASIC DATA TYPES

File *ql_typ.h* declares all the basic data types used in OpenCPU.

```
typedef unsigned   char      bool;   //TURE or FALSE
typedef unsigned   char      u8;
typedef signed     char      s8;
typedef            char      ascii;
typedef unsigned   short     u16;
typedef            short     s16;
typedef unsigned   int       u32;
typedef            int       s32;
typedef unsigned   int       ticks;
```

5. Event

OpenCPU provides an event-driven communication mechanism for the communication from Core System to Embedded Application. Embedded Application can accept the EVENTs from Core System by calling *Ql_GetEvent(&qlEventBuffer)*.

EVENT data structure is wrapped in the structure *QlEventBuffer*, and EVENT data consists of two parts:

- EVENT type
  The type of the received EVENT. It can also be used to work out the associated structure of the EVENT data part.

- EVENT data
  Specific EVENT body.

```
typedef struct QlEventBufferTag
{
   QlEventType  eventType;
   QlEventData  eventData;
}QlEventBuffer;
```

5.1. EVENT TYPE

All EVENTs are defined in *QlEventType*.

```
typedef enum QlEventTypeTag
{
    EVENT_NULL = 0,
    EVENT_INTR,
    EVENT_KEY,
    EVENT_UARTREADY,
    EVENT_UARTDATA,
    EVENT_MODEMDATA,
    EVENT_TIMER,
    EVENT_SERIALSTATUS,
    EVENT_MSG,
    EVENT_POWERKEY,
    EVENT_HEADSET,
    EVENT_UARTESCAPE,
    EVENT_UARTFE,          /* only support Generic uart, uart2 */
    EVENT_MAX = 0Xff
}QlEventType;
```

5.1.1. EVENT_INTR

The event is triggered when Embedded Application receives an interrupt signal from the Core System. Interrupt signals are generated by the interrupt pins which were subscribed by calling *Ql_pinSubscribe* function. To get extended information about the interrupt pins, please refer to the PERIPHERY API chapter.

5.1.2. EVENT_KEY

The event is triggered when a key' status has changed, and the key EVENT data contains the information of a key press or release. Under the reset condition of OpenCPU, there is a predefined keypad of five columns and five rows. When the status of any one of these keys (suppose that above mentioned pins have not been configured as other uses) is changed, Embedded Application will receive the EVENT_KEY event.

5.1.3. EVENT_UARTDATA

The event is triggered when Core System has received data from UART port. For example, when GPS data is fed in through UART port, this event will be trigger.

5.1.4. EVENT_UARTREADY

This event is designed for the function *Ql_SendToUart_2*. When the number of bytes sent actually is less than the number of bytes to send, Embedded Application should stop sending data, till receive the EVENT_UARTREADY event. The data not sent out last time should be resent.

5.1.5. EVENT_MODMEDATA

The event is triggered when data from Core System is sent to Embedded Application. For instance when Embedded Application receives an AT command's responds, or PPP/CSD data from protocol stack.

5.1.6. EVENT_TIMER

The event is triggered when a timer expires. Timer can be stopped before it expires, thus stopped timer will no longer trigger *EVENT_TIMER*. For more details about timer, please refer to TIMER API section

5.1.7. EVENT_SERIALSTATUS

The event is triggered when the status of CTS or DCD of serial port changes.

5.1.8. EVENT_MSG

The event passes message between tasks.

5.1.9. EVENT_ POWERKEY

The event is trigged when power key is pressed..

Note:
The EVENT_POWERKEY event can only be received by main task.

5.1.10. EVENT_HEADSET

When earphone is plugged in device or plugged out from device, Embedded Application will receive this event.

Note:
The EVENT_HEADSET event can only be received by main task.

5.1.11. EVENT_UARTESCAPE

UART port has the Escape function. Embedded Application can configure this function by calling *Ql_UartConfigEscape()*. Under the condition that the Escape function is enabled, if external device inputs three consecutive characters '+' through UART port, Embedded Application will receive the EVENT_UARTESCAPE event.

Note:
The EVENT_UARTESCAPE event can be received only by the owner task. Please see *Ql_SetPortOwner()* to get extended information about owner task.

The default control character for Escape function is '+'. However, developer can set it to other character by calling *Ql_UartConfigEscape()*.

5.1.12. EVENT_UARTFE

When error occurs during transferring data through UART port, such as frame error, parity error, Embedded Application will receive the EVENT_UARTFE event.

Note:
- The EVENT_ UARTFE event just works for UART2..
- Embedded Application should call *Ql_UartGenericClearFEFlag()* to clear some flags after received the EVENT_UARTFE event. Or, no EVENT_UARTFE event will be broadcast when next error occurs.

- The EVENT_ UARTFE event can only be received by the owner task. Please see *Ql_SetPortOwner()* to get extended information about owner task.

5.2. EVENT DATA STRUCTUR

5.2.1. Event Data Union

Each of the EVENT types is one to one mapping with the corresponding EVENT data.

```
typedef union QlEventDataTag
{
   Timer_Event        timer_evt;
   Key_Event          key_evt;
   PortData_Event     uartdata_evt;
   PortData_Event     modemdata_evt;
   Intr_Event         intr_evt;
   PortStatus_Event   portstatus_evt;
   Msg_Event          msg_evt;
   Powerkey_Event     powerkey_evt;
   Headset_Event      headset_evt;
   UartEscape_Event   uartescape_evt;
   UartFE_Event       uartfe_evt;  /*only support Generic uart, uart2*/
}QlEventData;
```

QlEventData is a union type, and each of the data types has its own structure. Below sections will describe these structures in detail respectively.

5.2.2. Timer_Event

```
typedef struct Timer_EventTag
{
   u32  timer_id;
   u32  interval; // it is measured in tick.
}Timer_Event;
```

timer_id: Timer ID.
interval: The time, in milliseconds, between raisings of the elapsed event.

5.2.3. Key_Event

```
typedef struct Key_EventTag
{
   u16    key_val;
   bool   isPressed;
}Key_Event;
```

key_val: The key value.
isPressed: Whether the key is pressed.

5.2.4. PortData_Event

OpenCPU provides two virtual modem serial ports and three physical UART ports. Embedded Application can use virtual modem serial ports to send AT commands to Core System or receive AT response from Core System. Embedded Application can communicate with external device through physical UART ports. All virtual modem serial ports and physical UART ports are enumerated in the structure of *QlPort*.

```
typedef enum QlPortTag
{
    ql_md_port1,
    ql_md_port2,
    ql_uart_port1,
    ql_uart_port2,
    ql_uart_port3
}QlPort;

typedef struct PortData_EventTag
{
   u8                    length;
   u8                    data[EVENT_MAX_DATA];
   QlPort                port;
   QlDataType            type;
} PortData_Event;
```

length: The length of the data being transported.

data: The data buffer, which maximum size is 1024 bytes.

port: Serial port that data comes from.

type: The type of the data, one of the *QlDataType* types. The QlDataType types are defined as below:

```
typedef enum QlDataTypeTAG
{
   DATA_AT = 0,
   DATA_PPP,
   DATA_CSD,
   DATA_MAX
}QlDataType;
```

DATA_AT: AT command data type.

DATA_PPP: Point-to-Point Protocol data type.

DATA_CSD: Circuit-Switched Data type.

5.2.5. Intr_Event

```
typedef struct Intr_EventTag
{
   QlPinName      pinName;
   bool           pinState;
}Intr_Event;
```

pinName: Name of the pin.
pinState: State of the pin:

5.2.6. PortStatus_Event

```
typedef enum QlLineTypeTag
{
    DCD,
    CTS
}QlLineType;
typedef struct  PortStatus_EventTag{
    u8 val;
    QlLineType type;
   QlPort  port;
}PortStatus_Event;
```

val: serial port data
type: line type of serial port.
port: serial port.

5.2.7. Msg_Event

```
typedef struct Msg_EventTag
{
   s32       src_taskid;
   u32       data1;
   u32       data2;
}Msg_Event;
```

src_taskid: Id of the task, which sent message event.
data1: user data.
data2: user data

5.2.8. Powerkey_Event

This event takes effect only when the function of Controlling Switch (i.e. *Customer_user_qlconfig.powerautoon* and *Customer_user_qlconfig.powerautooff* are set to TRUE)

is turned on.

```
typedef enum QlPowerKeyTypeTag
{
    POWERKEY_ON,
    POWERKEY_OFF
}QlPowerKeyType;
typedef struct Powerkey_EventTag
{
   QlPowerKeyType powerkey;
   bool           isPressed;
}Powerkey_Event;
```

powerkey: if POWERKEY_ON, this event indicates switching on device;
if POWERKEY_OFF, this event indicates switching off device.
isPressed: when powerkey is POWERKEY_OFF, TRUE indicates that the PWRKEY key is pressed, and FALSE indicates that the PWRKEY key is released.

5.2.9. Headset_Event

This event takes effect only after the earphone detecting function is configured to ON.

```
typedef enum QlHeadsetTypeTag
{
    HEADSET_PLUGOUT,     // Plug out earphone
    HEADSET_PLUGIN,       // Plug in earphone
    HEADSET_KEYPRESS,     // Press down the button on earphone
    HEADSET_KEYRELEASE,   // Release the button on earphone
    HEADSET_ADC           // ADC
}QlHeadsetType;

typedef struct Headset_EventTag
{
   QlHeadsetType       headsettype;
   u32                 data1;
}Headset_Event;
```

data1: ADC value, and available only when headsettype is set to HEADSET_ADC.

5.2.10. UartEscape_Event

UART port has the Escape function. Embedded Application can configure this function by calling *Ql_UartConfigEscape()*. Under the condition that the Escape function is enabled, if external device consecutively inputs three characters of '+' through UART port, Embedded Application will receive the EVENT_UARTESCAPE event.

```
typedef struct UartEscape_EventTag
{
    QlPort      port;
}UartEscape_Event;
```

port: UART port, in which Escape event occurs.

5.2.11. UartFE_Event

```
typedef struct UartFE_EventTag
{
    QlPort      port;
    u32         data1;
    u32         data2;
}UartFE_Event;
```

port: UART port, in which error occurs.
data1: reserved.
data2: reserved.

5.3. Example for Event Handling

The following code skeleton demonstrates how events are captured and dealt with in Embedded Applications.

```
QlEventBuffer  qlEventBuffer;   // Please keep this variable a global variable
                                // because of its big size.
void ql_entry() //task entrance
{
  // …
   while(TRUE)
   {
      Ql_GetEvent(&qlEventBuffer);
      switch(qlEventBuffer.eventType)
      {
        case EVENT_UARTDATA:
          break;
        case EVENT_MODEMDATA:
          break;
        case EVENT_KEY:
          break;
        case EVENT_TIMER:
         if(qlEventBuffer.sig_p.timer_evt.timer_id == timerDemo.timerId)
         {
            //deal with the timerDemo's event
            Ql_SendToUart(PortNo, "the timer expired!", Len);
         }
         break;
        //…
        default:
          break;
      }
    }
}
```

Here, if the expired timer's ID equals to *timerDemo.timerId*, Embedded Application sends "the timer expired!" to the specified uart port.

6. API Functions

6.1. SYSTEM API

The *ql_interface.h* file declares system related APIs. These functions are essential to any custom applications. Make sure to include the header file.

6.1.1. Ql_GetEvent

This function gets system EVENTs from the Core System. When there is no event in customer task's event queue, the task is in the waiting state.

- **Prototype**

```
void Ql_GetEvent(QlEventBuffer  *event_p);
```

- **Parameters**

event_p: A pointer to a particular *QlEventBuffer*, refer to chapter EVENT for *QlEventBuffer* structure.

The following code is an example about how to create a signal buffer, and listen to incoming signals using *Ql_GetEvent* function.

```
QlEventBuffer  qlEventBuffer;    // Please keep this variable a global variable
                                 // because of its big size.
void Ql_entry(void)
{
  while (TRUE)
  {
     Ql_GetEvent(&qlEventBuffer); //get EVENT from Quectel Core system
     switch(qlEventBuffer.eventType)
     {
         …
      }
   }
}
```

6.1.2. Ql_Reset

This function resets the system. And this function can be the alias of different functions with different parameters.

- **Prototype**

```
void Ql_Reset(u8 resettype);
```

Parameters:

resettype: Types of this function.
0 = WDT_RESET()
1 = DRV_ABN_RESET()
2 = DRV_RESET()
3 = ASSERT(0)

6.1.3. Ql_Sleep

This function suspends the execution of the current task until the time-out interval elapses.

- **Prototype**

```
void Ql_Sleep(u32 msec);
```

Parameters:

msec: The time interval for which execution is to be suspended, in milliseconds.

6.1.4. Ql_PowerDown

This function powers off the module.

- **Prototype**

```
void Ql_PowerDown(u8 powertype);
```

Parameters:

powertype: Action types of this function.
0 = Urgently power off
1 = Normal power off

6.1.5. Ql_PowerOnAck

This function is designed to power device on. Developer may call this function at a proper time to decide the time to switch on device. The *Customer_user_qlconfig.powerautoon* should be set to FALSE before calling this function.

- **Prototype**

```
void Ql_PowerOnAck(void);
```

6.1.6. Ql_StartWatchdog

This function starts watch-dog service. If not call Ql_FeedWatchdog within 90*200*10s=18000s=3 minutes, the module will automatically power down.

- **Prototype**

```
bool Ql_StartWatchdog(u16 tick10ms, u32 overfeedcount, u16 resettype);
```

tick10ms:

Counter which counts by step 1 every 10ms.

overfeedcount:

Counter threshold.
The module will restart when the counter's value
is over this threshold.

resettype:

0 = WDT_RESET, 1 = powerdown, 2 = ASSERT

Return Value:

TRUE indicates success in starting watch dog service.
FALSE indicates failure.

6.1.7. Ql_FeedWatchdog

Calling this function will reset watch-dog. If not call Ql_FeedWatchdog within 90*200*10s=18000s=3 minutes, the module will automatically power down.

- **Prototype**

```
void Ql_FeedWatchdog(void);
```

6.1.8. Ql_StopWatchdog

Stop the watch-dog, which was started previously.

- **Prototype**

```
void Ql_StopWatchdog(void);
```

6.2. MEMORY API

Head file *ql_memory.h* declares memory related APIs. These functions are essential to any custom

applications. Make sure to include the header file.

The example in the *example_multimemory.c* of OpenCPU SDK shows the proper usages of these methods

6.2.1. Define Memory Size

The memory size is defined use *QL_MEMORY_HEAP_SIZE* in ql_customer_config.c

For Example
Defines the memory size to 10K bytes:

```
#define QL_MEMORY_HEAP_SIZE (10*1024)
```

6.2.2. Ql_GetMemory

This function allocates a block of memory large enough from memory pool.

- **Prototype**

```
void *Ql_GetMemory(u16 Size);
```

Parameter
Size: The size of memory will be allocated, unit is the byte.

- **Return value**

The address of the allocated memory. NULL will be returned if the allocatioin .fails

6.2.3. Ql_FreeMemory

Deallocates or frees a memory block.

- **Prototype**

```
s32 Ql_FreeMemory (void *Ptr);
```

- **Parameters**

Ptr: The address of allocated memory

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.2.4. Ql_memMaxCanAllocSize

This function gets the max memory free block that is allocable in SRAM.

- **Prototype**

```
u32 Ql_memMaxCanAllocSize(void);
```

- **Return value**

Number of bytes.

6.2.5. Ql_memTotalLeftSize

This function gets the total number of bytes of the free space in SRAM.

- **Prototype**

```
U32 Ql_memTotalLeftSize(void);
```

- **Return value**

Number of bytes.

6.3. FILE SYSTEM API

These interfaces are used to operate file system. In order to use these interfaces the header file *ql_filesystem.h* must be included.

Note:
This stack size of the task, in which file operations will be acted, cannot be less than 4KB.

The example in the *example_file.c* of OpenCPU SDK shows the proper usages of these methods

6.3.1. Ql_FileGetFreeSize

This function obtains the total amount of free space of file system.

- **Prototype**

```
s32 Ql_FileGetFreeSize(void);
```

- **Return value:**

The total number of bytes of free space of file system.

6.3.2. Ql_FileOpenEx

This function opens or creates a file with a specified name.

- **Prototype**

```
s32 Ql_FileOpenEx(u8* asciifilename, u32 Flag);
```

Parameters:

asciifilename:

The name of the file. The name is limited to 252 characters. You must use a relative path, such as “filename.ext” or “dirname\filename.ext”.

Flag:

A u32 that defines the file's opening and access mode. The possible values are listed as follow:

QL_FS_READ_WRITE, can read and write

QL_FS_READ_ONLY, can only read

QL_FS_CREATE, opens the file, if it exists. If the file does not exist, the function creates the file

QL_FS_CREATE_ALWAYS, creates a new file. If the file exists, the function overwrites the file and clears the existing attributes.

- **Return value:**

If the function succeeds, the return value specifies a file handle. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.3. Ql_FileRead

Reads data from the specified file, starting at the position indicated by the file pointer. After the read operation has been completed, the file pointer is adjusted by the number of bytes actually read.

- **Prototype**

```
s32 Ql_FileRead(s32 filehandle, u8 *readbuffer, u32 readlength, u32 *readedlen);
```

Parameters:

filehandle: [in] A handle to the file to be read, which is the return value of the function *Ql_FileOpen*.

readbuffer: [out] Pointer to the buffer that receives the data read from the file.

readlength: [in] Number of bytes to be read.

readedlen: [out] Pointer to the number of bytes read.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.4. Ql_FileWrite

Writes data to the specified file, starting at the position indicated by the file pointer. After the writing operation has been completed, the file pointer is adjusted by the number of bytes actually written.

- **Prototype**

```
s32 Ql_FileWrite(s32 filehandle, u8 *writebuffer, u32 writelength, u32 *written);
```

Parameters:

filehandle: [in] A handle to the file to be written, which is the return value of the function *Ql_FileOpen*.

writebuffer: [in] Pointer to the buffer containing the data to be written to the file.

writelength: [in] Numberof bytes to write to the file.

written: [out] Pointer to the number of bytes written by the function call.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.5. Ql_FileSeek

This function repositions the pointer in the previously opened file.

- **Prototype**

```
s32 Ql_FileSeek(s32 filehandle, s32 Offset, s32 Whence);
```

Parameters:

filehandle: [in] File handle, which is the return value of the function *Ql_FileOpen*.

Offset: [in] Number of bytes to move the file pointer.

Whence: [in] Pointer movement mode. Must be one of the following values.

```
typedef enum QlFsSeekPosTag
{
  QL_FS_FILE_BEGIN,
  QL_FS_FILE_CURRENT,
  QL_FS_FILE_END
}QlFsSeekPos;
```

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.6. Ql_FileGetFilePosition

This function gets the current value of the file pointer.

- **Prototype**

```
s32 Ql_FileGetFilePosition(s32 filehandle, u32 * Position);
```

Parameters:

filehandle: [in] File handle, which is the return value of the function *Ql_FileOpen*.

Position: [out] Address of an u32 that will receive the current offset from the beginning of the file.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.7. Ql_FileTruncate

This function truncates the specified file to zero length.

- **Prototype**

```
s32 Ql_FileTruncate(s32 filehandle);
```

Parameters:

filehandle: The file handle, it is the return value of the function *Ql_FileOpen*.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.8. Ql_FileFlush

Forces any data remaining in the file buffer to be written to the file.

- **Prototype**

```
void Ql_FileFlush(s32 filehandle);
```

Parameters:

filehandle: The file handle, which is the return value of the function *Ql_FileOpen*.

6.3.9. Ql_FileClose

Closes the file associated with the file handle and makes the file unavailable for reading or

writing.

- **Prototype**

```
void Ql_FileClose(s32 filehandle);
```

Parameters:

filehandle: The file handle, which is the return value of the function *Ql_FileOpen*.

6.3.10. Ql_FileGetSize

This function retrieves the size, in bytes, of the specified file.

- **Prototype**

```
s32 Ql_FileGetSize(u8 *asciifilename, u32 *filesize);
```

Parameters:

asciifilename: The name of the file. The name is limited to 252 characters. You must use a relative path, such as “filename.ext” or “dirname\filename.ext”.

filesize: A pointer to the variable where the file size, in bytes, is stored.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.11. Ql_FileDelete

This function deletes an existing file.

- **Prototype**

```
s32 Ql_FileDelete(u8 *asciifilename);
```

Parameters:

asciifilename: The name of the file to be deleted. The name is limited to 252 characters. You must use a relative path, such as “filename.ext” or “dirname\filename.ext”.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.12. Ql_FileCheck

This function checks whether the file exists or not.

- **Prototype**

```
s32 Ql_FileCheck(u8 *asciifilename);
```

Parameters:

asciifilename: The name of the file. The name is limited to 252 characters. You must use a relative path, such as “filename.ext” or “dirname\filename.ext”.

- **Return value:**

QL_RET_OK indicates that the file exists.
If *QL_RET_ERR_FILENOTFOUND*, indicates the file is not exists.
Other negative, please see ERROR CODES.

6.3.13. Ql_FileRename

This function renames an existing file.

- **Prototype**

```
s32 Ql_FileRename(u8 *asciifilename, u8 *newasciifilename);
```

Parameters:

asciifilename: The current name of the file. The name is limited to 252 characters. You must use a relative path, such as “filename.ext” or “dirname\filename.ext”.

newasciifilename: The new name for the file. The new name must not already exist. The name is limited to 252 characters. You must use a relative path, such as “filename.ext” or “dirname\filename.ext”.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.14. Ql_FileCreateDir

This function creates a directory.

- **Prototype**

```
s32 Ql_FileCreateDir(u8 *asciidirname);
```

Parameters:

asciidirname: The name of the directory to create. The name is limited to 252 characters. You

must use a relative path, such as “dirname1” or “dirname1\dirname2”.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.15. Ql_FileRemoveDir

This function removes an existing directory.

- **Prototype**

```
s32 Ql_FileRemoveDir(u8 *asciidirname);
```

Parameters:

asciidirname: The name of the directory to be removed. The name is limited to 252 characters. You must use a relative path, such as “dirname1” or “dirname1\dirname2”.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.3.16. Ql_FileCheckDir

This function checks whether the directory exists or not.

- **Prototype**

```
s32 Ql_FileCheckDir(u8 *asciidirname);
```

Parameters:

asciidirname: The name of the directory. The name is limited to 252 characters. You must use a relative path, such as “dirname1” or “dirname1\dirname2”.

- **Return value:**

QL_RET_OK indicates the directory exists.
If *QL_RET_ERR_FILENOTFOUND*, indicate the directory is not exists.
Other negative, please see ERROR CODES.

6.3.17. Ql_FileFindFirst

Searches a directory for a file or subdirectory which name matches the specified file name.

- **Prototype**

```
s32 Ql_FileFindFirst(u8 *asciipath, u8 *asciifilename, u32 filenamelength,
u32 *filesize, bool *isdir);
```

Parameters:

asciipath: [in] The directory or path. The *asciipath* is limited to 252 characters, which can include wildcard characters. You must use a relative path, such as "dirname*" or "dirname*.txt".

asciifilename: [in] A pointer to the buffer that receives the name of the file.

filenamelength: [in] The maximum number of bytes to be received of the name.

filesize: [in] A pointer to the variable that the size specified by the file.

isdir: [in] A pointer to the variable that the type specified by the file.

- **Return value:**

If the function succeeds, the return value is a search handle that can be used in a subsequent call to the *Ql_FindNextFile* or *Ql_FindClose* functions.

QL_RET_ERR_FILENOMORE indicates that there's no file and subdirectory found.
Other negative indicates failure. To get extended error information, please see ERROR CODES.

6.3.18. Ql_FileFindNext

This function continues a file search from a previous call to the *Ql_FileFindFirst* function.

- **Prototype**

```
s32 Ql_FileFindNext(s32 handle, u8 *asciifilename, u32 filenamelength, u32
*filesize, bool *isdir);
```

Parameters:

handle: The search handle returned by a previous call to the *Ql_FileFindFirst* function.
asciifilename: A pointer to the buffer that receives the name of the file.
filenamelength: The maximum number of bytes to be received of the name.
filesize: A pointer to the variable that the size specified by the file.
isdir: A pointer to the variable that the type specified by the file.

- **Return value:**

If positive or 0, that is the find handle, find successfully.
If *QL_RET_ERR_FILENOMORE*, indicate no files and subdirectories.
Other negative, please refer to chapter for ERROR CODES.

6.3.19. Ql_FileFindClose

This function closes the specified search handle.

- **Prototype**

```
void Ql_FileFindClose(s32 handle);
```

Parameters:

handle: Find handle, returned by a previous call of the *Ql_FileFindFirst* function.

6.4. PERIPHERY API

The interface functions in this section are declared in header file *ql_bus.h*, *ql_pin.h*.

6.4.1. Multifunction Pins

OpenCPU provides abundant multifunction pins for Embedded Application. Application can configure diverse function for each pin programmatically. Each multifunctional pin may work in one of four modes.

6.4.1.1. Pin Names

- Multifunction pins definition for M10.

```
typedef enum QlPinNameTag
{
   QL_PINNAME_DISP_DATA = 0,
   QL_PINNAME_DISP_CLK,
   QL_PINNAME_DISP_CS,
   QL_PINNAME_DISP_DC,
   QL_PINNAME_DISP_RST,
   QL_PINNAME_NETLIGHT,
   QL_PINNAME_SIM_PRESENCE,
   QL_PINNAME_KBR0,
   QL_PINNAME_KBR1,
   QL_PINNAME_KBR2,
   QL_PINNAME_KBR3,
   QL_PINNAME_KBR4,
   QL_PINNAME_KBC0,
   QL_PINNAME_KBC1,
   QL_PINNAME_KBC2,
   QL_PINNAME_KBC3,
   QL_PINNAME_KBC4,
   QL_PINNAME_GPIO1,
```

```
    QL_PINNAME_BUZZER,
    QL_PINNAME_DSR,
    QL_PINNAME_RI,
    QL_PINNAME_DCD,
    QL_PINNAME_CTS,
    QL_PINNAME_RTS,
    QL_PINNAME_DTR,
    QL_PINNAME_GPIO0,
    QL_PINNAME_MAX
} QlPinName;
```

6.4.1.2. Working Modes

Each multifunctional pin may work in one of four Pin Modes once. The four modes are listed in the following enumeration.

```
typedef enum QlPinModeTag
{
    QL_PINMODE_1 = 0,
    QL_PINMODE_2 = 1,
    QL_PINMODE_3 = 2,
    QL_PINMODE_4 = 3,
    QL_PINMODE_UNSET = 255
} QlPinMode;
```

The following table shows the relationship between multifunctional pins and working modes.

Table 1 Multifunction pins and working modes for M10

PIN No	QlPinName	RESET	MODE1	MODE2	MODE3	MODE4
1	QL_PINNAME_DISP_DATA	O,L,PD	DISP_DATA	GPIO		
2	QL_PINNAME_DISP_CLK	O,L,PD	DISP_CLK	GPIO		
3	QL_PINNAME_DISP_CS	O,H,PU	DISP_CS	GPIO		
4	QL_PINNAME_DISP_D/C	O,L,PD	DISP_D/C	GPIO		
5	QL_PINNAME_DISP_RST	O,H,PU	DISP_RST	GPIO		
6	QL_PINNAME_NETLIGHT	I,PD	NETLIGHT	GPIO	PWM	
11	QL_PINNAME_SIM_PRESENCE	I,PU	SIM_PRES	GPIO	EINT	
28	QL_PINNAME_KBR0	O	KBR0	GPIO		
29	QL_PINNAME_KBR1		KBR1	GPIO		
30	QL_PINNAME_KBR2		KBR2	GPIO		
31	QL_PINNAME_KBR3		KBR3	GPIO		
32	QL_PINNAME_KBR4		KBR4	GPIO		
33	QL_PINNAME_KBC0	I,PU	KBC0	GPIO		
34	QL_PINNAME_KBC1		KBC1	GPIO		
35	QL_PINNAME_KBC2		KBC2	GPIO		
36	QL_PINNAME_KBC3		KBC3	GPIO		

37	QL_PINNAME_KBC4		KBC4	GPIO		
38	QL_PINNAME_GPIO1	I,PU	GPIO	KBC5	CLK_OUT	
39	QL_PINNAME_BUZZER	I,PD	BUZZER	GPIO	ALERTER	
54	QL_PINNAME_DSR	I,PD	DSR	GPIO	EINT	
55	QL_PINNAME_RI	I,PU	RI	GPIO	EINT	
56	QL_PINNAME_DCD	I,PU	DCD	GPIO		
57	QL_PINNAME_CTS	I,PU	CTS	GPIO		
58	QL_PINNAME_RTS	I,PU	RTS	GPIO		
59	QL_PINNAME_DTR	I,PU	DTR	GPIO	EINT	
64	QL_PINNAME_GPIO0	I,PU	GPIO	EINT		

The ‘MODE1’ defines the original status of pin in standard module.

“RESET” column defines the default status of every pin after system power on.
“I” means input.
“O” means output.
“PU” means internal pull-up circuit.
“PD” means internal pull-down circuit.
“L” means low level voltage.
“H” means high level voltage.

Customer can reference the file “*ql_customer_gpio.c*” to initialize these pins. In the following initialization sample code of these pins, when pin is set to *QL_PINMODE_UNSET*, this pin will be configured with default value as “RESET” item in the above table. If pin is set to *QL_PINMODE1*, the pin will be configured with the value of “MODE1” item in above table. Configuration of the other modes is similar.

6.4.1.3. Configure Multifunction Pins

Without configuration, these multifunction pins have the same function status with the corresponding pins in standard module. Developer may configure these multifunction pins to some initial working status.

```
const QlPinConfigTable Customer_QlPinConfigTable =
{
    QL_PIN_VERSION,/*now this version is 0x100*/
    QL_OPENCPU_FLAG,
    {//pinconfigitem[END_OF_QL_PINNAME+1]
    {QL_PINNAME_DISP_DATA, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_DISP_CLK, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_DISP_CS, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_DISP_DC, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_DISP_RST, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_NETLIGHT, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1,
QL_PINPULLENABLE_ENABLE, 0,0,0},
```

```
    //{QL_PINNAME_NETLIGHT, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_2,
    QL_PINPULLENABLE_ENABLE,QL_PINDIRECTION_OUT,QL_PINLEVEL_HIGH,0},/*config GPIO function*/
    //{QL_PINNAME_NETLIGHT, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_3,
    QL_PINPULLENABLE_ENABLE,0,0,0}, /*config PWM other parameter, please use Ql_pinSubscribe(...) function to config*/
    {QL_PINNAME_SIM_PRESENCE, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_2,
    QL_PINPULLENABLE_ENABLE,QL_PINDIRECTION_IN,0,0}, /*config GPIO function*/
    {QL_PINNAME_KBR0, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_KBR1, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_KBR2, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_KBR3, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET , 0,0,0,0},
    {QL_PINNAME_KBR4, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_KBC0, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_KBC1, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_KBC2, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_KBC3, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_KBC4, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0},
    {QL_PINNAME_GPIO1, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULLENABLE_ENABLE,
    QL_PINDIRECTION_OUT, QL_PINLEVEL_LOW,0}, //{QL_PINNAME_GPIO1, QL_PINSUBSCRIBE_UNSUB,
    QL_PINMODE_3, QL_PINPULLENABLE_ENABLE,0,0,0},/*config CLOCK other parameter, please use
    Ql_pinSubscribe(...) function to config*/
    {QL_PINNAME_BUZZER, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULLENABLE_ENABLE,
0, 0, 0 },
    {QL_PINNAME_DSR, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULLENABLE_ENABLE, 0, 0,
0 },
    {QL_PINNAME_RI, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULLENABLE_ENABLE,0,0,0},
    {QL_PINNAME_DCD, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULLENABLE_ENABLE, 0, 0,
0 },
    {QL_PINNAME_CTS, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULLENABLE_ENABLE, 0, 0,
0 },
    {QL_PINNAME_RTS, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULLENABLE_DISABLE, 0, 0,
0 }, /*config rts, must set QL_PINPULLENABLE_DISABLE*/
    {QL_PINNAME_DTR, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULLENABLE_ENABLE, 0, 0,
0 },
    {QL_PINNAME_GPIO0, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULLENABLE_ENABLE,
QL_PINDIRECTION_OUT, QL_PINLEVEL_LOW, 0},
    //{QL_PINNAME_GPIO0, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_2,
QL_PINPULLENABLE_ENABLE, 0, 0, 0}, /*config EINT other parameter, please use Ql_pinSubscribe(...) function to config*/
    {QL_PINNAME_MAX, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_UNSET, 0,0,0,0} /*must end item is QL_PINNAME_MAX*/
    }
};
```

The *QlPinConfigTable* structure **is** defined as below:

```
typedef struct QlPinConfigTableTag
{
   s16 pinconfigversion;   /* now this version is QL_PIN_VERSION */
   s8  quectelflg[QL_OPENCPU_FLAG_MAXLEN];
   QlPinConfigItem pinconfigitem[QL_PINNAME_MAX+1];   /* must end item is
QL_PINNAME_MAX */
}QlPinConfigTable;
```

The *QlPinConfigItem* structure **is** defined as below:

```
typedef struct QlPinConfigItemTag
{
   QlPinName       pinname;
   QlPinSubscribe  subscribe;
   QlPinMode       pinmode;
   u32             parameter1;
   u32             parameter2;
   u32             parameter3;
   u32             parameter4;
}QlPinConfigItem;
```

subscribe:

Indicates whether a pin is registered. If this parameter is set to *QL_PINSUBSCRIBE_SUB*, developers can directly do some operations later, such as *Ql_pinWrite(),Ql_pinRead()*. Or developers have to call *Ql_pinSubscribe()* to register the pin before calling *Ql_pinWrite()*or *Ql_pinRead()*.

pinmode:

Pin mode, a value of QlPinMode.

parameter:

Parameter of the pin. If *pinmode* is *QL_PINMODE_1* which is the default mode as standard module, this parameter has no effect and should be set as NULL. When other *pinmode* is chosen, the definition of this parameter will be related to the *pinmode*.

6.4.2. Pin Functions

This section includes pin API functions that are applicable to request and control the functions of the pins.

The examples in the *example_gpio.c, example_clk.c and example_pwm.c* of OpenCPU SDK show the proper usages of these methods

6.4.2.1. Ql_pinSubscribe

This function subscribes a working mode for pin.

- **Prototype**

```
      s32 Ql_pinSubscribe(QlPinName pinname, QlPinMode pinmode, QlPinParameter
*pinparameter);
```

- **Parameter**

pinName: The name of the pin

pinmode: Mode of the pin.

pinparameter: Parameter of the pin. If *pinmode* is *QL_PINMODE_1* which is the default mode as standard module, this parameter has no effect and should be set as NULL. When other *pinmode* is chosen, the definition of this parameter will be related to the *pinmode*.

Below is the definition about data structure of this parameter.

```
typedef struct QlPinParameterTag
{
   s16                   pinconfigversion;
   QlPinParameterUnion    pinparameterunion;
}QlPinParameter;

typedef union QlPinParameterUnionTag
{
   QlGpioParameter     gpioparameter;
   QlEintParameter     eintparameter;
   QlClockParameter    clockparameter;
   QlPwmParameter      pwmparameter;
   QlAlertParameter    alertparameter;
}QlPinParameterUnion;
```

pinconfigversion: must be set to *QL_PIN_VERSION*.

◆ For GPIO function:

QlGpioParameter is the configuration parameter of GPIO.

```
typedef struct QlGpioParameterTag
{
   QlPinPullEnable   pinpullenable;
   QlPinDirection    pindirection;
   QlPinLevel        pinlevel;
}QlGpioParameter;
```

pinpullenable: enable pin pull-up or pull-down.

QL_PINPULLENABLE_DISABLE

QL_PINPULLENABLE_ENABLE

pindirection: set pin direction.

QL_PINDIRECTION_IN

QL_PINDIRECTION_OUT

pinlevel: set pin voltage level.

QL_PINLEVEL_LOW

QL_PINLEVEL_HIGH

- For EINT function:

QlEintParameter is the configuration parameter of interrupt.

```
typedef struct  QlEintParameterTag
{
   QlEintSensitiveType  eintsensitivetype;
   s32                  hardware_de_bounce;
}QlEintParameter;
```

eintsensitivetype: set EINT sensitive type.

QL_EINTSENSITIVETYPE_EDGE

QL_EINTSENSITIVETYPE_LEVEL

hardware_de_bounce: set EINT debounce time. Unit is ms; the maximum value is 63 ms.

- For clock-out function:

QlClockParameter is the configuration parameter of clock-out.

```
typedef struct   QlClockParameterTag
{
   QlClockSource        clocksource;
}QlClockParameter;
```

clocksource: set clock source.

QL_CLOCKSOURCE_26M

QL_CLOCKSOURCE_13M

QL_CLOCKSOURCE_6DOT5M

QL_CLOCKSOURCE_32K

- For PWM function:

QlPwmParameter is the configuration parameter of PWM.

The output PWM frequency is determined by: (pwmsource/pwmclkdiv) / (lowpulesnumber+highpulesnumber).

```
typedef struct  QlPwmParameterTag
{
   QlPwmSource     pwmsource;
   QlPwmSourceDiv pwmclkdiv;
   u32               lowpulesnumber;
   u32               highpulesnumber;
}QlPwmParameter;
```

pwmsource: set PWM clock source.

QL_PWMSOURCE_13M

QL_PWMSOURCE_32K

pwmclkdiv: set clock divide.

QL_PWMSOURCE_DIV1

QL_PWMSOURCE_DIV2

QL_PWMSOURCE_DIV4

QL_PWMSOURCE_DIV8

lowpulesnumber: set the number of clock cycles to stay at low level. The result of l*owpulesnumber* plus *highpulesnumber* is less than 8193.

highpulesnumber: set the number of clock cycles to stay at high level. The result of l*owpulesnumber* plus *highpulesnumber* is less than 8193.

✧ For Alert function:

QlAlertParameter is the configuration parameter of Alert.

```
typedef struct  QlAlertParameterTag
{
   QlAlertClock  alertclock;
   QlAlertMode   alertmode;
   u16           alertcounter1;
   u16          alertcounter2;
   u16          alertthreshold;
}QlAlertParameter;
```

alertclock: set Alert clock.

QL_ALERTCLOCK_13M

QL_ALERTCLOCK_13DIV2M

QL_ALERTCLOCK_13DIV4M

QL_ALERTCLOCK_13DIV8M

alertmode: set Alert mode.

QL_ALERTMODE_1

QL_ALERTMODE_2

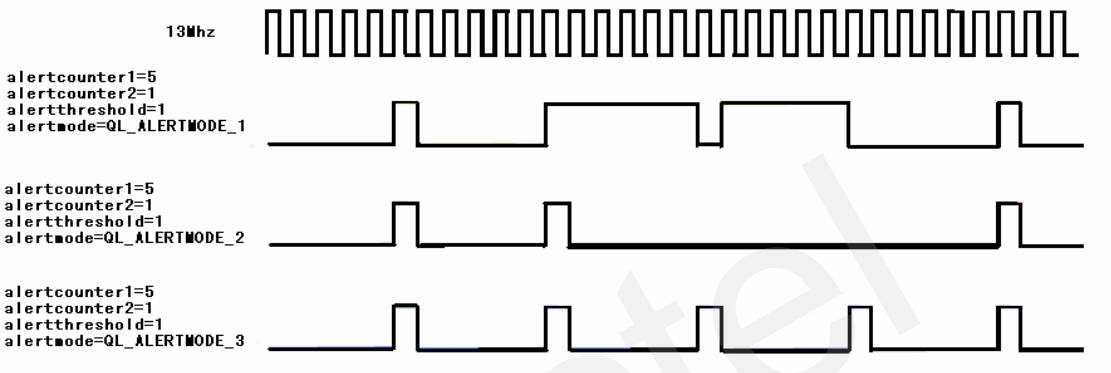
QL_ALERTMODE_3

alertcounter1: set alert counter1. The maximum value is 65535.

alertcounter2: set alert counter2. The maximum value is 65535.

alertthreshold: set alert threshold. The maximum value is 65535.

The following figure shows the Alert waveform with the parameter values.



- **Return value**

  QL_RET_OK on success,

  If negative, please refer to ERROR CODES

Note:

1. *QL_PINMODE_UNSET* can't be used in this function.
2. When a pin is configured to EINT enabled mode by function *Ql_pinSubscribe*, Embedded Application could receive interrupt event (INTR EVENT).
3. In order to subscribe those display related pins such as *QL_PINNAME_DISP_DATA* to LCD function mode, Embedded Application must call *Ql_busSubscribe* function to request LCD function, instead of calling function *Ql_pinSubscribe*.

6.4.2.2. Ql_pinUnSubscribe

This function unsubscribes pin.

- **Prototype**

```
s32 Ql_pinUnSubscribe(QlPinName pinname);
```

- **Parameter**

pinName: The name of the pin.

- **Return value**

  QL_RET_OK on success,

  If negative, please refer to chapter for ERROR CODES

6.4.2.3. Ql_pinQueryMode

This function queries the named pin's mode.

- **Prototype**

```
s32 Ql_pinQueryMode(QlPinName pinname, QlPinSubscribe *subscribe,
QlPinMode *pinmode, QlPinParameter *pinparameter);
```

- **Parameters:**

pinName: Pin name.
subscribe: The pointer to the pin's subscribe state.
pinmode: The pointer to the pin's mode.
pinparameter: Please reference function *Ql_pinSubscribe*.

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.4.2.4. Ql_pinRead

This function inquires the specified pin's level.

- **Prototype**

```
s32 Ql_pinRead(QlPinName pinname, QlPinLevel_e *pinlevel);
```

- **Parameters:**

pinName: The name of the pin.
pinlevel: The pointer to the pin's level.

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.4.2.5. Ql_pinWrite

This function sets the specified pin's level. And this function is available only when the pin is configured as GPIO,

- **Prototype**

```
s32 Ql_pinWrite(QlPinName pinname, QlPinLevel_e pinlevel);
```

- **Parameters:**

pinName: The name of the pin.

pinlevel: The pointer to the pin's level

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.4.2.6. Ql_pinControl

This function controls the specified pin. This function takes effect only when the pin is set to clock-out, PWM, or ALERT function mode.

- **Prototype**

```
s32 Ql_pinControl(QlPinName pinname, QlPinControl_e pincontrol);
```

- **Parameters:**

pinName: The name of the pin.

pincontrol: The operation of pin. For example, when the pin is set to clock-out function mode, *QL_PINCONTROL_START* indicates starting clock output, and *QL_PINCONTROL_STOP* indicates stopping clock output.

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.4.2.7. Ql_eintRead

This function reads level state of a specified interruption pin.

- **Prototype**

```
void Ql_eintRead(u8 eintno, QlPinLevel *pinlevel);
```

- **Parameters:**

eintno:

[in] External interruption number

pinlevell:

[out] A pointer to 'QlPinLevel'

6.4.2.8. Ql_eintMask

The function masks or unmasks a specified external interruption.

- **Prototype**

```
void Ql_eintMask(u8 eintno, bool mask);
```

- **Parameters:**

eintno:

External interruption number

mask:

TRUE or FALSE, mask or unmask

6.4.2.9. Ql_eintSetPolarity

This function sets the polarity of the level of external interruption source.

- **Prototype**

```
void Ql_eintSetPolarity(u8 eintno, QlPinLevel Polaritylevel);
```

- **Parameters:**

eintno:

Pin name

Polaritylevel:

Level polarity, see the definition of 'QlPinLevel'.

6.4.3. BUS Functions

This section describes pin API functions that are applicable to request and control the function of the bus.

The example in the *example_i2c.c* of OpenCPU SDK shows the proper usages of these methods

6.4.3.1. Ql_busSubscribe

This function subscribes BUS function.

- **Prototype**

```
QL_BUS_HANDLE Ql_busSubscribe(QlBusType bustype, QlBusParameter
*busparameter);
```

- **Parameter**

bustype: *QL_BUSTYPE_LCD* or *QL_BUSTYPE_I2C*.

busparameter: an union structure. It has different definition of data structure according to the *bustype*. Below is the definition of data structure.

```
typedef struct QlBusParameterTag
{
    s16                     busconfigversion;
    QlBusParameterUnion     busparameterunion;
}QlBusParameter;
```

```
typedef union QlBusParameterUnionTag
{
    QlLcdParameter     lcdparameter;
    QlI2cParameter     i2cparameter;
}QlBusParameterUnion;
```

Busconfigversion: must be set to QL_BUS_VERSION.

✧ For LCD bus:

QlLcdParameter is the configuration parameter of LCD.

```
typedef struct QlLcdParameterTag
{
   u32     lcd_serial_config; /*QlLcdSerialConfig*/
   u8       brequire_cs;
   u8       brequire_resetpin;
}QlLcdParameter;
```

lcd_serial_config: Configuration parameters of LCD, multi following types can be combined with a logical OR.

Selection of clock source

QL_LCD_SERIAL_CONFIG_13MHZ_CLK

QL_LCD_SERIAL_CONFIG_26MHZ_CLK

Serial Interface Chip Select Polarity Control

QL_LCD_SERIAL_CONFIG_CS_POLARITY

8-bit or 9-bit Interface Selection

QL_LCD_SERIAL_CONFIG_9_BIT_MODE

QL_LCD_SERIAL_CONFIG_8_BIT_MODE

Serial Clock Divide Select

QL_LCD_SERIAL_CONFIG_CLOCK_DIVIDE_1

QL_LCD_SERIAL_CONFIG_CLOCK_DIVIDE_2

QL_LCD_SERIAL_CONFIG_CLOCK_DIVIDE_3

QL_LCD_SERIAL_CONFIG_CLOCK_DIVIDE_4

Clock Phase Control

QL_LCD_SERIAL_CONFIG_CLOCK_PHASE

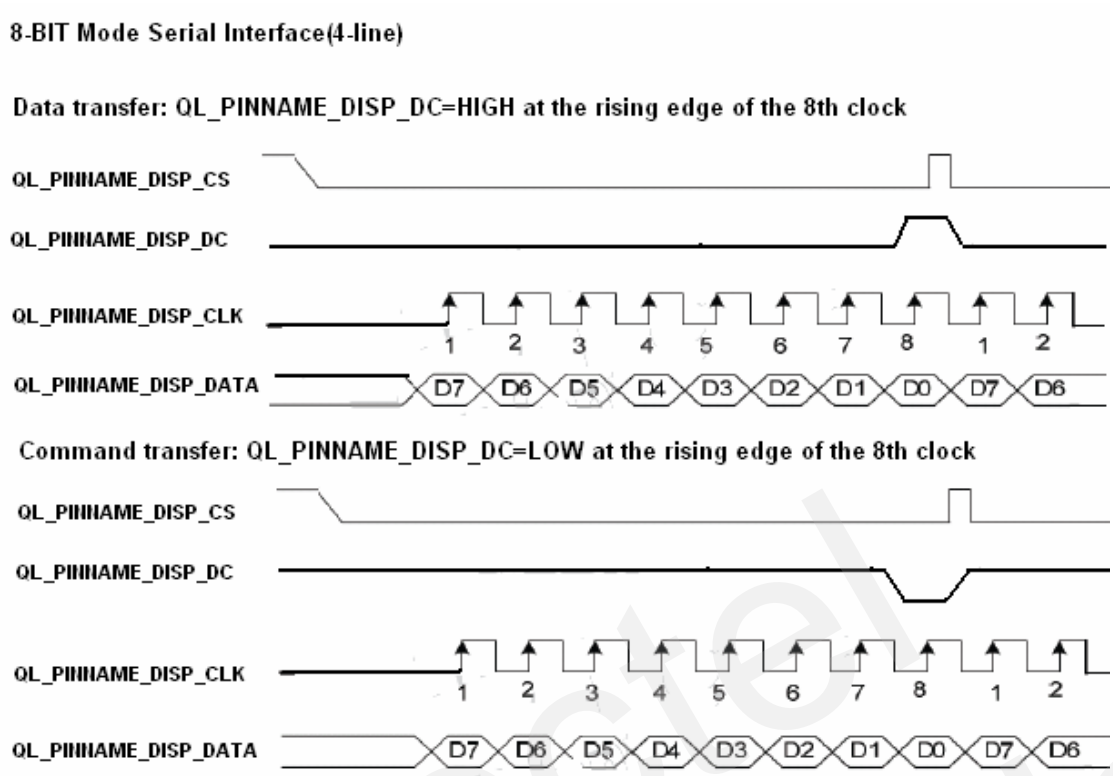
Clock Polarity Control

QL_LCD_SERIAL_CONFIG_CLOCK_POLARITY

brequire_cs: indicates whether *QL_PINNAME_DISP_CS* pin is used to control bus.

brequire_resetpin: indicates whether *QL_PINNAME_DISP_RST* pin is used to control bus.

The following figure shows the timing diagram of the serial LCD interface when *QL_LCD_SERIAL_CONFIG_8_BIT_MODE* is set.



✧ For I2C BUS:

QlI2cParameter is the configuration parameter of I2C.

```
typedef struct QlI2cParameterTag
{
   QlPinName          pin_i2cdata;
   QlPinName          pin_i2cclk;
}QlI2cParameter;
```

pin_i2cdata: the pin that used as I2C data line.

pin_i2cclk: the pin that used as I2C colock line.

- **Return value**

The return value is a positive number, which indicates the handle of bus, if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

Notes:

1. If *bustype* is *QL_BUSTYPE_LCD*, all LCD related pins, *QL_PINNAME_DISP_DATA*, *QL_PINNAME_DISP_CLK*, *QL_PINNAME_DISP_CS*, *QL_PINNAME_DISP_DC*, and *QL_PINNAME_DISP_RST* will be used. These pins must be in unsubscribed state before the BUS is subscribed.
2. If *bustype* is *QL_BUSTYPE_I2C*, two I2C BUS related pins must also be in unsubscribed state.

6.4.3.2. Ql_busUnSubscribe

This function cancels the subscription for BUS.

- **Prototype**

```
s32 Ql_busUnSubscribe(QL_BUS_HANDLE bushandle);
```

- **Parameter**

bushandle: BUS handle returned by *Ql_busSubscribe*

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.4.3.3. Ql_busWrite

This function writes data to the BUS.

- **Prototype**

```
s32 Ql_busWrite(QL_BUS_HANDLE bushandle, QlBusAccess *busAccess, u8
*data_p, u8 datalen)
```

- **Parameters:**

bushandle: BUS handle returned by *Ql_busSubscribe*
busAccess: for different BUS type, this parameter has different meanings.

```
typedef struct QlBusAccessTag
{
   u32 address;
   u32 opcode;
}QlBusAccess;
```

Regarding to the function of parameter *address*, it indicates a 7-bit address of I2C slave device when the BUS is an I2C BUS. It has no effect when the BUS is an LCD BUS.

✧ For I2C bus:

The parameter *opcod*" is set to *QL_BUSACCESSOPCODE_I2C_NOTSTOPBIT*, no stop bit will be sent.
When *opcode* is set as *QL_BUSACCESSOPCODE_I2C_NOTHING*, the stop bit will be sent.

✧ For LCD bus:

When the parameter *opcode* is set to *QL_BUSACCESSOPCODE_LCD_CLEAR_RESET*, the pin *QL_PINNAME_DISP_RST*, which should be configured to LCD function in advance, will pull down to reset LCD. In this case, parameter *data_p* and *datalen* have no effect and should be set to NULL.

When the parameter *opcode* is set to *QL_BUSACCESSOPCODE_LCD_SET_RESET*, the pin QL_PINNAME_DISP_RST, which should be configured to LCD function in advance, will pull

up to resume reset pin. In this case, parameter *data_p* and *datalen* have no effect and should be set to NULL.

When the parameter *opcode* is set to *QL_BUSACCESSOPCODE_LCD_DATAWRITE*, the parameter *data_p* pointing to data buffer will be sent to LCD.

When the parameter *opcode* is set to *QL_BUSACCESSOPCODE_LCD_CMDWRITE*, the parameter *data_p* point to the data buffer of LCD command will be sent to LCD.

data_p: address of the data that will be written to bus.
Datalen: length of the data written.

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.4.3.4. Ql_busRead

This function reads data from BUS. And this function is available only when the BUS is I2C bus.

- **Prototype**

```
s32 Ql_busRead(QL_BUS_HANDLE bushandle, QlBusAccess *busAccess, u8 *data_p, u8
datalen);
```

- **Parameters:**

bushandle: BUS handle returned by *Ql_busSubscribe*
busAccess: for different BUS type, this parameter has different meanings.

```
typedef struct QlBusAccessTag
{
   u32 Address;
   u32 opcode;
}QlBusAccess;
```

Regarding to the function of parameter *address*, it indicates a 7-bit address of I2C slave device when the bus is an I2C bus.

When the parameter *opcode* is set to QL_BUSACCESSOPCODE_I2C_NOTSTOPBIT for I2C BUS, no stop bit will be sent. When *opcode* is set to other value, the stop bit will be sent.

data_p: address of the buffer to save data read.
Datalen: length of the buffer to read data.

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.4.3.5. Ql_busQuery

This function inquires the configuration of the BUS.

- **Prototype**

```
s32 Ql_busQuery(QL_BUS_HANDLE bushandle, QlBusType *bustype,
QlBusParameter *busparameter);
```

- **Parameter**

bushandle: [in] bus handle returned by *Ql_busSubscribe()*

bustype: [out] pointer to the `QlBusType` buffer that stores the current BUS.

busparameter: [out] pointer to the `QlBusParameter` buffer that stores the parameter configuration of the bus.

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.5. AUDIO API

File *ql_audio.h* needs to be included before calling audio functions.

The example in the *example_audio.c* of OpenCPU SDK shows the proper usages of these methods

6.5.1. Audio name

```
typedef enum QlAudioNameTag
{
   QL_AUDIO_EMS_CHIMES_HI = 1,
   QL_AUDIO_EMS_CHIMES_LO,
   QL_AUDIO_EMS_DING,
   QL_AUDIO_EMS_TADA,
   QL_AUDIO_EMS_NOTIFY,
   QL_AUDIO_EMS_DRUM,
   QL_AUDIO_EMS_CLAPS,
   QL_AUDIO_EMS_FANFARE,
   QL_AUDIO_EMS_CHORD_HI,
   QL_AUDIO_EMS_CHORD_LO,
   QL_AUDIO_1,
   QL_AUDIO_2,
   QL_AUDIO_3,
```

```
    QL_AUDIO_4,
    QL_AUDIO_5,
    QL_AUDIO_6,
    QL_AUDIO_7,
    QL_AUDIO_8,
    QL_AUDIO_9,
    QL_AUDIO_10,
    QL_AUDIO_11,
    QL_AUDIO_12,
    QL_AUDIO_13,
    QL_AUDIO_14,
    QL_AUDIO_15,
    QL_AUDIO_16,
    QL_AUDIO_17,
    QL_AUDIO_18,
    QL_AUDIO_19,
    QL_AUDIO_END
}QlAudioName;
```

6.5.2. Ql_PlayAudio

This function plays the predefined music in system.

- **Prototype**

```
s32 Ql_PlayAudio(QlAudioName name, bool repeat);
```

- **Parameters**

name: The audio name

repeat: If TRUE, The audio will be played repeatedly until *Ql_StopAudio* is called to stop playing audio. Otherwise, the audio will be played one time.

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.5.3. Ql_StopAudio

This function stops playing music

- **Prototype**

```
s32 Ql_StopAudio(QlAudioName name);
```

- **Parameter**

name: The audio name

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.5.4. Ql_StartPlayAudioFile

This function plays the audio file in system. It supports wav, mp3, amr audio format;

- **Prototype**

```
s32 Ql_StartPlayAudioFile(u8 *asciifilename, bool repeat, u8 volumelevel,
    u8 audiopath);
```

- **Parameters**

asciifilename: The name of the audio file. The name is limited to 252 characters. You must use a relative path, such as “filename.wav” or “dirname\filename.mp3”

repeat: If TRUE, The audio will be played repeatedly until *Ql_StopPlayAudioFile* is called to stop playing audio. Otherwise, the audio will be played one time.

volumelevel: set audio volume

```
typedef enum QlAudioVolumeLeveltag
{
   QL_AUDIO_VOLUME_LEVEL1 = 0,
   QL_AUDIO_VOLUME_LEVEL2,
   QL_AUDIO_VOLUME_LEVEL3,
   QL_AUDIO_VOLUME_LEVEL4,
   QL_AUDIO_VOLUME_LEVEL5,
   QL_AUDIO_VOLUME_LEVEL6,
   QL_AUDIO_VOLUME_LEVEL7,
   QL_AUDIO_VOLUME_LEVEL_END
}QlAudioVolumeLevel;
```

audiopath: set audio paly path

```
typedef enum QlAudioPlayPathTag
{
   QL_AUDIO_PATH_HEADSET = 1,  /* earphone */
   QL_AUDIO_PATH_LOUDSPEAKER = 2, /* loudspeaker for free sound */
   QL_AUDIO_PATH_END
}QlAudioPlayPath;
```

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.5.5. Ql_StopPlayAudioFile

This function stops playing audio file.

- **Prototype**

```
s32 Ql_StopPlayAudioFile(void);
```

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.5.6. Ql_StartPlayAudioStream

This function plays the audio stream in system. It supports wav, mp3, amr audio format. Customer uses *QlAudioResGen.exe* to build the audio stream data.

- **Prototype**

```
s32 Ql_StartPlayAudioStream(u8 *stream, u32 streamsize, s32 streamformat,
    bool repeat, u8 volume, u8 audiopath);
```

- **Parameters**

stream: The audio stream data.
streamsize: size of the audio stream data.
streamformat: formatsize of the audio stream data.

```
typedef enum QlAudioStreamFormattag
{
    QL_AUDIO_STREAMFORMAT_MP3 = 1,
    QL_AUDIO_STREAMFORMAT_AMR = 2,
    QL_AUDIO_STREAMFORMAT_WAV = 3,
    QL_AUDIO_STREAMFORMAT_END
}QlAudioStreamFormat;
```

repeat: If TRUE, The audio will be played repeatedly until *Ql_StopPlayAudioStream* is called to stop playing audio. Otherwise, the audio will be played one time.
volumelevel: set audio volume, please refer *`Ql_StartPlayAudioFile.`*
audiopath: set audio paly path, please refer *`Ql_StartPlayAudioFile.`*

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.5.7. Ql_StopPlayAudioStream

This function stops playing audio stream.

- **Prototype**

```
s32 Ql_StopPlayAudioStream(void);
```

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.5.8. Ql_VoiceCallChangePath

This function switches voice output source.

- **Prototype**

```
bool Ql_VoiceCallChangePath(QlAudioPlayPath path);
```

- **Parameters**

path:

Voice output source.

- **Return value**

TRUE if the function succeeds.

FALSE, indicates failure.

6.5.9. Ql_VoiceCallGetCurrentPath

This function retrieves the current voice output source.

- **Prototype**

```
QlAudioPlayPath Ql_VoiceCallGetCurrentPath(void);
```

- **Return value**

The current voice output source.

6.6. TIMER API

Head file *ql_timer.h* needs to be included for following APIs to work.

6.6.1. TIMER STRUCTURE

```
typedef struct QlTimerTag
{
   ticks     timeoutPeriod; //the time elapse before the timer times out
   u32       timerId; // the ID of the timer
}
QlTimer;
```

6.6.2. Ql_StartTimer

This function starts a timer.

- **Prototype**

```
u32 Ql_StartTimer(QlTimer *timer_p);
```

- **Parameter**

timer_p: timer to be started

- **Return value**

The ID of this timer

Example

```
QLtimer timerDemo;
// set timeout period to 2 seconds
timerDemo.timeoutPeriod = Ql_SecondToTicks(2);
//start the timer
Ql_StartTimer(&timerDemo);
```

6.6.3. Ql_StopTimer

This function stops the previously raised timer.

- **Prototype**

```
s16 Ql_StopTimer(QlTimer *timer_p);
```

- **Parameter**

timer_p: The timer to be stopped

- **Return value**

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.6.4. Ql_SecondToTicks

This function converts time from seconds to ticks.

- **Prototype**

```
ticks Ql_SecondToTicks(u32 seconds);
```

- **Return value:**

Ticks equivalent to the seconds.

6.6.5. Ql_MillisecondToTicks

This function converts time from milliseconds to ticks.

- **Prototype**

```
ticks Ql_MillisecondToTicks(u32 milliseconds);
```

- **Return value:**

Ticks equivalent to the milliseconds.

6.6.6. Ql_GetRelativeTime

This function returns the number of milliseconds since the device booted.

- **Prototype**

```
u32   Ql_GetRelativeTime(void);
```

- **Return value:**

Number of milliseconds.

6.6.7. Ql_GetRelativeTime_Counter

This function returns the number of MCU counters since the device booted.

- **Prototype**

```
u32   Ql_GetRelativeTime_Counter(void);
```

- **Return value:**

Number of MCU counters.

6.6.8. Ql_GetLocalTime

This function gets the local time.

- **Prototype**

```
bool Ql_GetLocalTime (QlSysTimer * datetime);
```

- **Parameter**

datetime: A *QlSysTimer* struct to store current local time.

- **Return value**

TRUE, if the function succeeds. Otherwise, returns *FALSE*.

QLSysTimer are defined as below:

```
typedef struct QlSysTimerTag
{
   unsigned short year;
   unsigned char month;
   unsigned char day;
   unsigned char hour;
   unsigned char minute;
   unsigned char second;
}QlSysTimer;
```

6.6.9. Ql_SetLocalTime

This function sets the current local date and time.

- **Prototype**

```
bool  Ql_SetLocalTime(QlSysTimer * datetime);
```

- **Parameter**

datetime:

A pointer to *QlSysTimer*, which carries the information of date and time.

- **Return value**

TRUE if this function succeeds in retrieving the local date and time, otherwise FALSE.

6.6.10. Ql_Mktime

This function get total seconds elapsed since 1970.01.01 00:00:00.

- **Prototype**

```
u32  Ql_Mktime(QlSysTimer *psystime);
```

- **Parameter**

psystime:

A pointer to *QlSysTimer*, which will store the information of date and time.

- **Return value**

The total seconds.

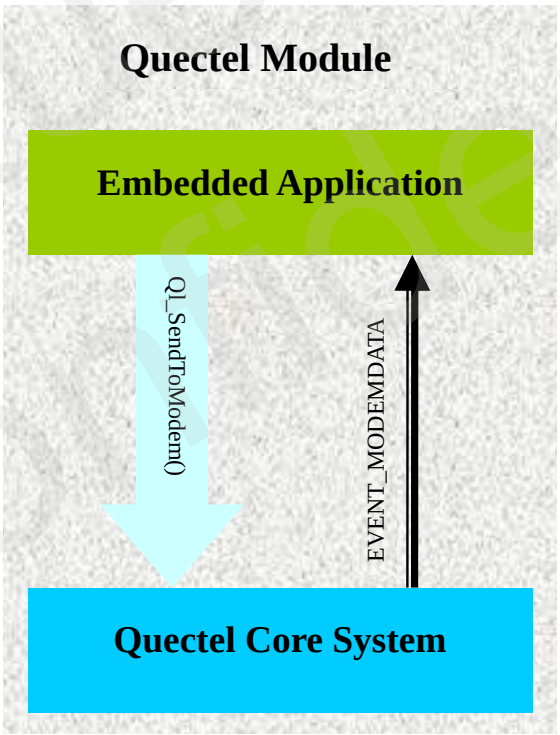
6.7. FCM API

The required head file is *ql_fcm.h* for these APIs in this section.

The examples in OpenCPU SDK show the proper usages of these methods.

6.7.1. Virtual Modem Port

The following diagram illustrates how Embedded Application communicates with Core System via the virtual modem port.



6.7.1.1. Ql_OpenModemPort

This function will open virtual modem serial port. Before Embedded Application sends data to Core System or receive data from Core System by the virtual modem serial port, the port must be opened successfully in advance.

- **Prototype**

```
bool Ql_OpenModemPort(QlPort port);
```

- **Parameters**

port: the virtual serial port that data will be sent to.

- **Return value**

TRUE, if this function opens the virtual serial port successfully.
FALSE, fails to open modem port.

6.7.1.2. Ql_SendToModem

This function sends data to Core System buffer. Such data can be AT commands, CSD data or GPRS data. For AT commands, result codes Ok or ERROR is retrieved by *Ql_GetEvent* function. Refer to the EVENT section for more details. A special character “\x0d” or “\r” (carriage return) should be appended after a string of AT command to indicate the ending. For example: *Ql_SendToModem("ati\0xd",4)* is the same as you type "ATI" command and press ENTER key in Hyper Terminal.

- **Prototype**

```
s32 Ql_SendToModem(QlPort port, u8 *data,u16 data_len);
```

- **Parameters**

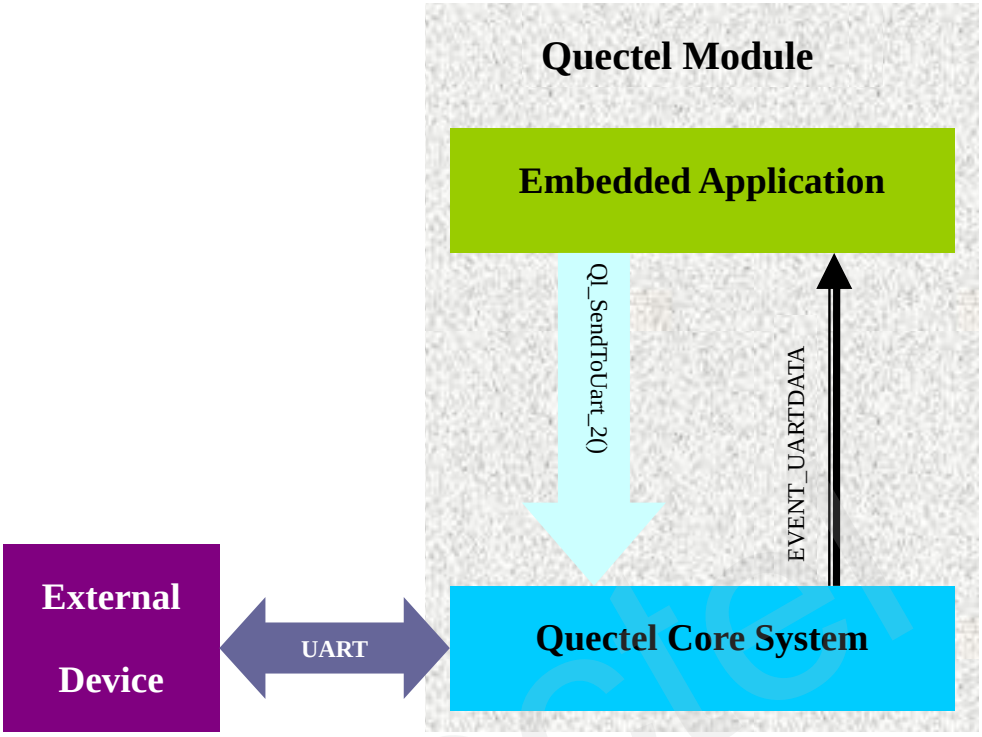
port: the virtual serial port that data will send to.
data: The data sent to Core System
len: The length of the data, cannot exceed 1024

- **Return value**

Number of bytes actually sent. If this function fails to send data, a negative number will be returned. To get extended information, please see 'Error Code Definition'.

6.7.2. UART Port

The following diagram illustrates how Embedded Application communicates with external device.



6.7.2.1. Ql_SendToUart_2

This function is used to send data to the specified UART port. When the number of bytes actually sent is less than that to send, Application should stop sending data, and application will receive an event -- EVENT_UARTREADY later. After receiving this Event Application can continue to send data, and previously unsent data should be resend.

For UART2, only after debug mode is set to basic mode, this function is available.

- **Prototype**

```
s32 Ql_SendToUart_2(QlPort port, u8*src, u16 len);
```

- **Parameters**

port: UART port.
src: Pointer to data to send.
len: The length of the data, cannot exceed 1024.

- **Return value**

Number of bytes actually sent. If this function fails to send data, a negative number will be returned. To get extended information please see 'ERROR CODES'.

6.7.2.2. Ql_SetPortRts

This function sets the enable of the port RTS.

- **Prototype**

```
void Ql_SetPortRts(QlPort port,bool rts);
```

- **Parameter**

port: port name.
rts: whether clear or set the RTS line status of the port.

Note: If *rts* is set to 0, the virtual modem port or UART port will stop sending data to Embedded Application until *rts* is set to 1 again.

6.7.2.3. Ql_SetUartBaudRate

This function sets baud rate of the specified UART port.

- **Prototype**

```
s32 Ql_SetUartBaudRate(QlPort port,s32 rate)
```

- **Parameter**

port: port name.
rate: baud rate.

Supported baud rates:
1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200

Return value
The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.7.2.4. Ql_SetPortOwner

This function sets the id of the task which uses the specified virtual modem port or UART port.

- **Prototype**

```
s32 Ql_SetPortOwner(QlPort port,QlTaskId id);
```

- **Parameter**

port: port name.
id: id of the task which uses the port.

```
typedef enum QlTaskIDtag{
  ql_main_task,
  ql_sub_task1,
  ql_sub_task2,
  ql_sub_task3,
  ql_sub_task4,
  ql_sub_task5,
  ql_sub_task6,
  ql_sub_task7,
  ql_sub_task8,
  ql_sub_task9,
  ql_sub_task10,
  ql_max_task
} QlTaskId;
```

Return value

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

Note:

- This interface function is designed for multitask programming.
- The default owner of the ports is main task.
- Only the owner task can receive the EVENT_MODEMDATA event from Core System.
- Any tasks can *Ql_SendToModem* after Ql_*OpenModemPort*, and not must *Ql_SetPortOwner*.
-

6.7.2.5. Ql_SetUartDCBConfig

This function sets the configuration parameters of the specified UART port.

- **Prototype**

```
s32 Ql_SetUartDCBConfig(QlPort port, s32 rate, s32 dataBits, s32 stopBits, s32 parity);
```

- **Parameter**

port: port name.

rate: baud rate.

dataBits: Data bit. Possible value: 5, 6, 7, 8.

stopBits:

Value	meaning
1	1 stop bit
2	2 stop bits
3	1.5 stop bits

parity:

Value	meaning
0	No parity
1	Odd parity
2	Even parity
3	Space

Supported baud rates:

1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400, 57600, 115200

Return value

The return value is *QL_RET_OK* if this function succeeds. Otherwise, the return value is an error code. To get extended error information, please see ERROR CODES.

6.7.2.6. Ql_SetUartFlowCtrl

This function sets the flow-contrl mode of the specified UART port. And this setting just takes effect during run-time.

- **Prototype**

```
void Ql_SetUartFlowCtrl(QlPort port, Ql_FlowCtrlMode wrtFlowCtrl, Ql_FlowCtrlMode rdFlowCtrl);
```

- **Parameter**

port: port name.

wrtFlowCtrl: Write flow control.

rdFlowCtrl: Read flow control.

6.7.2.7. Ql_UartGetBytesAvail

This function gets the number of bytes of data in the receive buffer.

- **Prototype**

```
u16 Ql_UartGetBytesAvail(QlPort port);
```

- **Parameter**

port: port name.

Return value

The number of bytes of data in the receive buffer.

6.7.2.8. Ql_UartGetTxRoomLeft

This function gets the number of bytes of free space in the send buffer.

- **Prototype**

```
u16 Ql_UartGetTxRoomLeft(QlPort port);
```

- **Parameter**

port: UART port

Return value

The number of bytes of free space in the send buffer.

6.7.2.9. Ql_UartGetTxRestBytes

This function gets the number of bytes not sent out in the send buffer.

- **Prototype**

```
u32 Ql_UartGetTxRestBytes(QlPort port, u32* totalbufffer_size);
```

- **Parameter**

port: UART port

totalbufffer_size:

[out] Pointer to the total number of bytes of the send buffer. This parameter may be set to NULL if don't be cared.

Return value

The number of bytes not sent out in the send buffer.

6.7.2.10. Ql_UartConfigEscape

This function configures Escape function of specified UART port.

- **Prototype**

```
bool Ql_UartConfigEscape(QlPort port, u8 escapechar, u16 escguardtime);
```

- **Parameter**

port: UART port

escapechar: Escape character

escguardtime: Interval between Escape characters

Return value

The number of bytes of free space in the send buffer.

6.7.2.11. Ql_UartClrTxBuffer

This function clears send-buffer of specified UART port.

- **Prototype**

```
void Ql_UartClrTxBuffer(QlPort port);
```

- **Parameter**

port: port name.

6.7.2.12. Ql_UartClrRxBuffer

This function clears receive-buffer of specified UART port.

- **Prototype**

```
void Ql_UartClrRxBuffer(QlPort port);
```

- **Parameter**

port: port name.

6.7.2.13. Ql_UartSetGenericThreshold

This function sets the threshold of receive buffer of specified UART port. Now, this function just works for UART2 (Generic uart, not vfifo uart).

- **Prototype**

```
bool Ql_UartSetGenericThreshold(QlPort port, bool benable, u32 buffer_threshold, u32 waittimeout);
```

- **Parameter**

port:	UART port, only support UART2.
benable:	Enable flag
buffer_threshold:	The number of bytes in the internal input buffer before a Data Received event occurs; The maximum value is 2048.
waittimeout:	If the number of bytes of arrived data has been less than the value of 'buffer_threshold' within the time of 'waittimeout', a Data Received event will occur as a result of expiring 'waittimeout'.

Return value

TRUE indicates success; otherwise failure.

6.7.2.14. Ql_UartGenericClearFEFlag

This function clears FE flag of specified UART port. Now, this function just works for UART2.

- **Prototype**

```
void Ql_UartGenericClearFEFlag(QlPort port);
```

- **Parameter**

port:

UART port, only support UART2.

6.7.2.15. Ql_UartSetVfifoThreshold

This function ssets the buffer size and the threshold in input buffer and output buffer of specified VFIFO UART port.

Now, this function just works for UART1 and UART3.

- **Prototype**

```
bool Ql_UartSetVfifoThreshold(
   QlPort port,
   u32 rx_len,
   u32 tx_len,
   u32 rx_alert_length,
   u32 tx_alert_length
);
```

- **Parameter**

port:

UART port, only support UART1 and UART3.

rx_len:

The size (number of bytes) of the input buffer of specified UART port

tx_len:

The size (number of bytes) of the output buffer of specified UART port

rx_alert_length:

The size (number of bytes) of the internal input buffer before a Data Received event occurs; The maximum value is 2048.

tx_alert_length:

The size (number of bytes) of the internal output buffer before a Data Received event occurs; The maximum value is 3584.

Return value

TRUE indicates success; otherwise failure.

6.7.2.16. Ql_UartMaxGetVfifoThresholdInfo

This function gets the buffer size and the threshold in input buffer and output buffer of specified VFIFO UART port.

Now, this function just works for UART1 and UART3.

- **Prototype**

```
bool Ql_UartGetVfifoThresholdInfo(
        QlPort port,
        u32 *arg_rx_len,
        u32 *arg_tx_len,
        u32 *arg_rx_alert_length,
        u32 *arg_tx_alert_length
    );
```

- **Parameter**

port:

UART port, only support UART1 and UART3.

arg_rx_len:

[out] Pointer to the size (number of bytes) of the input buffer of specified UART port.

arg_tx_len:

[out] Pointer to the size (number of bytes) of the output buffer of specified UART port.

arg_rx_alert_length:

[out] Pointer to the size (number of bytes) of the internal input buffer before a Data Received event occurs.

arg_tx_alert_length:

[out] Pointer to the size (number of bytes) of the internal output buffer before a Data Received event occurs.

Return value

TRUE indicates success; otherwise failure.

6.7.2.17. Ql_UartMaxGetVfifoThresholdInfo

This function gets the possible maximum value of the buffer size and the threshold in input buffer and output buffer of specified VFIFO UART port.

Now, this function just works for UART1 and UART3.

- **Prototype**

```
bool Ql_UartMaxGetVfifoThresholdInfo(
        QlPort port,
        u32 *arg_max_rx_len,
        u32 *arg_max_tx_len,
        u32 *arg_max_rx_alert_length,
        u32 *arg_max_tx_alert_length
);
```

- **Parameter**

port:

UART port, only support UART1 and UART3.

Arg_max_rx_len:

[out] Pointer to the possible maximum size (number of bytes) of the input buffer of specified UART port.

arg_max _tx_len:

[out] Pointer to the possible maximum size (number of bytes) of the output buffer of specified UART port.

arg_max _rx_alert_length:

[out] Pointer to the possible maximum size (number of bytes) of the internal input buffer before a Data Received event occurs.

arg_max _tx_alert_length:

[out] Pointer to the possible maximum size (number of bytes) of the internal output buffer before a Data Received event occurs.

6.7.2.18. Ql_UartDirectnessReadData

This function reads data from specified physical serial port.

If the calling origin is in the EVENT_UARTDATA event, caller should not call this function before handling the data about the EVENT_UARTDATA event.

- **Prototype**

```
s32   Ql_UartDirectnessReadData(QlPort port, u8* buffer_read, u16 buffer_size);
```

- **Parameter**

port:

Port number.

buffer_read:

[out] Pointer to the read data, and the buffer size must great than or equal to 1024 bytes.

buffer_size:

The data length (number of bytes) to read

Return value

The length of read data will be returned if the function succeeds. Otherwise, a Error Code will be returned.

6.7.2.19. Ql_UartQueryDataMode

This function queries the working mode (Command Mode or Data Mode) of specified virtual serial port.

- **Prototype**

```
int   Ql_UartQueryDataMode(QlPort port);
```

- **Parameter**

port:

Port number.

Return value

1 data mode
0 command mode
-1 invalid port

6.7.2.20. Ql_UartForceSendEscape

This function notifies the virtual serial port to quit from Data Mode, and return back to Command Mode.

- **Prototype**

```
bool Ql_UartForceSendEscape(QlPort port);
```

- **Parameter**

port:

Port number.

Return value

TRUE indicates success in calling this function.
FALSE is returned in the case of:

- Invalid port
- The input port is debug port

Remarks:

Success in calling this function doesn't mean the port has quit Data Mode and returned back to Command Mode until the application receives the response ‘OK’ from the virtual serial port.

6.8. TCP/IP API

The interface functions in this section are declared in *ql_tcpio.h*.

Before transferring any data packet between the mobile and the network, a PDP context (OpenCPU uses only IP context) must be defined (refer to the *Ql_GprsAPNSet* function) and activated (refer to the *Ql_GprsNetworkActive* function).

Developers can define up to **10 PDP-context-profiles** (refer to the *Ql_GprsAPNSet* function), but only **two** can be activated at the same time.

The examples in the *example_tcpip.c* and *example_tcplong.c* of OpenCPU SDK show the proper usages of these methods.

Note:

The maximum number of socket is **6**.

6.8.1. Possible Error Codes

The error codes are enumerated in the *ql_soc_error_enum* as below.

```
typedef enum
{
   QL_SOC_SUCCESS           = 0,
   QL_SOC_ERROR             = -1,
   QL_SOC_WOULDBLOCK        = -2,
   QL_SOC_LIMIT_RESOURCE    = -3, /* limited resource */
   QL_SOC_INVALID_SOCKET    = -4, /* invalid socket */
   QL_SOC_INVALID_ACCOUNT   = -5, /* invalid account id */
   QL_SOC_NAMETOOLONG      = -6, /* address too long */
   QL_SOC_ALREADY          = -7, /* operation already in progress */
   QL_SOC_OPNOTSUPP         = -8, /* operation not support */
   QL_SOC_CONNABORTED       = -9, /* Software caused connection abort */
   QL_SOC_INVAL             = -10, /* invalid argument */
   QL_SOC_PIPE              = -11, /* broken pipe */
   QL_SOC_NOTCONN          = -12, /* socket is not connected */
   QL_SOC_MSGSIZE          = -13, /* msg is too long */
   QL_SOC_BEARER_FAIL       = -14, /* bearer is broken */
   QL_SOC_CONNRESET         = -15, /* TCP half-write close, i.e., FINED */
   QL_SOC_DHCP_ERROR        = -16,
   QL_SOC_IP_CHANGED        = -17,
   QL_SOC_ADDRINUSE         = -18,
   QL_SOC_CANCEL_ACT_BEARER = -19   /* cancel the activation of bearer */
} ql_soc_error_enum;
```

6.8.2. Ql_GprsAPNSet

This function sets the authentication parameters apn/login/password to use with a profile id during PDP activation.

- **Prototype**

```
s32 Ql_GprsAPNSet(u8 profileid, u8 *apn, u8 *userId, u8 *password, Ql_CallBack_GprsAPNSet gprsapnset);
```

Parameters:

Profileid:

PDP context profile, which ranges from 0 to 9.

apn:

NULL-terminated APN characters.

userId:

User Id, NULL-terminated characters.

password:

Password, NULL-terminated characters.

gprsapnset:

Callback function. The Core System will invoke this callback function to notify Embedded Application whether this function succeeds or not.

- **Return value:**

The possible returned values:

QL_SOC_WOULDBLOCK	The app should wait, till the callback function is called. And the app can get the information of success or failure in callback function.
QL_SOC_INVAL	invalid argument
QL_SOC_ALREADY	The function is running.

6.8.3. Ql_GprsAPNGet

This function gets the authentication parameters apn/login/password with a profile id.

- **Prototype**

```
s32 Ql_GprsAPNGet(u8 profileid, Ql_CallBack_GprsAPNGet gprsapnget);
```

Parameters:

Profileid:

PDP context profile, which ranges from 0 to 9.

gprsapnget:

Callback function. The Core System will invoke this callback function to notify Embedded Application whether this function succeeds or not.

- **Return value:**

  The possible returned values:

QL_SOC_WOULDBLOCK	The app should wait, till the callback function is called. The app can get the information of success or failure in callback function.
QL_SOC_INVAL	invalid argument
QL_SOC_ALREADY	The function is running.

6.8.4. Ql_GprsNetworkInitialize

This function initializes the PDP context.

- **Prototype**

```
s32 Ql_GprsNetworkInitialize(u8 contextid, u8 profileid,
OpenCpuTcpIp_Callback_t *callback_func);
```

Parameters:

contextid:

OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter can be 0 or 1.

profiledid:

PDP context profile, which ranges from 0 to 9.

callback_func:

This callback is called by OpenCPU to inform Embedded Application whether this function succeeds or not. And this callback function should be implemented by Embedded Application.

This callback function is defined as below:

```
typedef struct
{
   void(*callback_network_actived)(u8 contexid);
   void(*callback_network_deactived)(u8 contexid, s32 error_cause, s32 error);
   void(*callback_socket_connect)(u8 contexid, s8 sock, bool result, s32
error_code);
   void(*callback_socket_close)(u8 contexid, s8 sock, bool result, s32
error_code);
   void(*callback_socket_accept)(u8 contexid, s8 sock, bool result, s32
error_code);
   void(*callback_socket_read)(u8 contexid, s8 sock, bool result, s32
error_code);
   void(*callback_socket_write)(u8 contexid, s8 sock, bool result, s32
error_code);
}OpenCpuTcpIp_Callback_t;
```

callback_network_actived t: refer to *Ql_GprsNetworkActive*
callback_network_deactived: refer to *Ql_GprsNetworkDeactive*
callback_socket_connect: refer to *Ql_SocketConnect*
callback_socket_close: This callback is called by OpenCPU when peer closes the socket TCP connection.
callback_socket_accept: refer to *Ql_SocketAccept*
callback_socket_read: refer to *Ql_SocketRecv*
callback_socket_write: refer to *Ql_SocketSend*

- **Return value:**

QL_SOC_SUCCESS This function succeeds
QL_SOC_INVAL Invalid argument
QL_SOC_ALREADY The GPRS network is already initialized.

6.8.5. Ql_GprsNetworkUnInitialize

This function restores the PDP context.

- **Prototype**

```
s32  Ql_GprsNetworkUnInitialize(u8 contextid);
```

Parameters:
contextid:
OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter can be 0 or 1.

- **Return value:**

QL_SOC_SUCCESS if this function succeeds. Otherwise, other Error Code will be returned. To get extended information, please see Possible Error Codes.

6.8.6. Ql_GprsNetworkActive

This function actives the PDP context.

- **Prototype**

```
s32 Ql_GprsNetworkActive(u8 contxtid);
```

Parameters:
contextid:
OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter can be 0 or 1.

- **Return value:**

The return value is 0 if; or other Error Code is returned. To get extended information, please see *soc_error_enum*.

The possible returned values:

QL_SOC_SUCCESS	This function succeeds
QL_SOC_WOULDBLOCK	The app should wait, till the callback function is called. The app can get the information of success or failure in callback function.
QL_SOC_INVAL	invalid argument
QL_SOC_ALREADY	The function is running.
QL_SOC_BEARER_FAIL	Bearer is broken

6.8.7. Ql_GprsNetworkDeactive

This function deactivates the PDP context.

- **Prototype**

```
s32  Ql_GprsNetworkDeactive(u8 contxtid);
```

Parameters:

contextid:

OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter can be 0 or 1.

- **Return value:**

The return value is 0 if this function succeeds; Otherwise, other Error Code is returned. To get extended information, please see Possible Error Codes.

The possible values:

QL_SOC_SUCCESS	This function succeeds
QL_SOC_WOULDBLOCK	The app should wait, till the *callback_network_deactived* function is called. The app can get the information of success or failure in callback function.
QL_SOC_INVAL	invalid argument
QL_SOC_ERROR	

6.8.8. Ql_GprsNetworkGetState

This function gets the state of GPRS network and PDP context.

- **Prototype**

```
s32 Ql_GprsNetworkGetState(u8 contxtid, OpenCpuNetWorkState_e
*networkstate, u8 *ps_status);
```

Parameters:

contextid:

[in] OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter can be 0 or 1.

networkstate:

[out] Pointer to ***OpenCpuNetWorkState_e***, in which the state of PDP context is stored.

ps_status:

[out] The GPRS network state. If the GPRS network is registered, this parameter will be output with the number 1. Any other value indicates that the GPRS network is not registered.

- **Return value:**

This return value will be *QL_SOC_SUCCESS* (0) if the function succeeds. Otherwise a negative number (Error Code) will be returned.

6.8.9. Ql_SocketCreate

This function creates a socket. The maximum number of socket is **6**.

- **Prototype**

```
s8 Ql_SocketCreate(u8 contxtid, u8 socket_type);
```

Parameters:

contextid:

[in] OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter can be 0 or 1.

socket_type:

[in] This parameter is one of:

```
typedef enum
{
   SOC_SOCK_STREAM = 0,  /* stream socket, TCP */
   SOC_SOCK_DGRAM,      /* datagram socket, UDP */
   SOC_SOCK_SMS,        /* SMS bearer */
   SOC_SOCK_RAW         /* raw socket */
} socket_type_enum;
```

- **Return value:**

The socket ID, or other Error Codes. To get extended information, please see *soc_error_enum*.

The possible returned values:

QL_SOC_INVAL	Invalid argument
QL_SOC_BEARER_FAIL	Bearer is broken
QL_SOC_LIMIT_RESOURCE	Exceed the maximum socket number

6.8.10. Ql_SocketClose

This function closes a socket.

- **Prototype**

```
s32 Ql_SocketClose(s8 socketId);
```

Parameters:

socketId: The socket id.

- **Return value:**

This return value will be *QL_SOC_SUCCESS* (0) if the function succeeds. Otherwise a negative number (Error Code) will be returned.

6.8.11. Ql_SocketConnect

This function establishes a connection to the socket. The host is specified by an IP address and a port number.

- **Prototype**

```
s32 Ql_SocketConnect(s8 socketId, u8 address[4], u16 port);
```

Parameters:

socketId: The socket id.

address: Peer IPv4 address.

port: Peer IPv4 port.

- **Return value:**

This return value will be *QL_SOC_SUCCESS* (0) if the function succeeds. Otherwise a negative number (Error Code) will be returned.

Possible values:

QL_SOC_SUCCESS	This function succeeds
QL_SOC_WOULDBLOCK	The app should wait, till the *callback_socket_connect* function is called. The app can get the information of success or failure in callback function.
QL_SOC_INVALID_SOCKET	invalid socket

6.8.12. Ql_SocketSend

The function sends data to a connected socket.

- **Prototype**

```
s32 Ql_SocketSend(s8 socket, u8 *data_p, s32 dataLen);
```

Parameters:

socket: The socket id.

data_p: Pointer to the data to send.

dataLen: Number of bytes to send.

- **Return value:**

If no error occurs, *Ql_SocketSend* returns the total number of bytes sent, which can be less than the number requested to be sent in the *dataLen* parameter. Otherwise, a value of *ql_soc_error_enum* is returned.

Remarks

The application should call *Ql_SocketSend()* circularly to send data till all the data in *data_p* is sent out. If the number of bytes actually sent is less than the number requested to be sent in the *dataLen* parameter, the application should keep sending out the left data.

If the *Ql_SocketSend* returns a negative number, but not *SOC_WOULDBLOCK*, which indicates some error happened to the socket, the application has to close the socket by calling *Ql_SocketClose()* and reestablish a connection to the socket.

6.8.13. Ql_SocketRecv

This function receives data from a bound socket.

- **Prototype**

```
s32 Ql_SocketRecv(s8 socket, u8 *data_p, s32 dataLen);
```

Parameters:

socket: The socket id.

data_p: Pointer to a buffer that is the storage space for the received data.

dataLen: Length of *data_p*, in bytes.

- **Return value:**

If no error occurs, *Ql_SocketRecv* returns the total number of bytes received. Otherwise, a value of *ql_soc_error_enum* is returned.

Remarks

The application should call *Ql_SocketRecv()* circularly in the *callback_socket_read* function to receive data and do data processing work till the *SOC_WOULDBLOCK* is returned.

If this function returns 0, which indicates the server closed the socket, the application has to close the socket by calling *Ql_SocketClose()* and reestablish a connection to the socket.

If the *Ql_SocketRecv()* returns a negative number, but not *SOC_WOULDBLOCK*, which indicates some error happened to the socket, the application has to close the socket by calling *Ql_SocketClose()* and reestablish a connection to the socket.

6.8.14. Ql_SocketTcpAckNumber

This function gets the TCP socket ACK number.

- **Prototype**

```
s32 Ql_SocketTcpAckNumber(s8 socket, u64 *ackedNum);
```

Parameters:

socket: [in] The socket id.

ackedNum: [out] Pointer to an u64 type that is the storage space for the TCP ACK number.

- **Return value:**

If no error occurs, this return value will be *QL_SOC_SUCCESS* (0). Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.15. Ql_SocketSendTo

This function sends data to a specific destination through UDP socket.

- **Prototype**

```
s32 Ql_SocketSendTo(s8 socket, u8 *data_p, s32 dataLen, u8 address[4], u16
port);
```

Parameters:

socket: The socket id.

data_p: Buffer containing the data to be transmitted.

dataLen: Length of the data in *data_p*, in bytes.

address: Pointer to the address of the target socket.

port: The target port number.

- **Return value:**

If no error occurs, this function returns the number of bytes actually sent. Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.16. Ql_SocketRecvFrom

This function receives a datagram data through TCP socket.

- **Prototype**

```
s32 Ql_SocketRecvFrom(s8 socket, u8 *data_p, s32 dataLen, u8 address[4], u16 *port);
```

Parameters:

socket: [in] The socket id.
data_p: [out] Buffer to store the received data.
dataLen: [in] Length of the data in *data_p*.
address: [out] An optional pointer to a buffer that receives the address of the connecting entity.
port: [out] An optional pointer to an integer that contains the port number of the connecting entity.

- **Return value:**

If no error occurs, this function returns the number of bytes received. Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.17. Ql_SocketBind

This function associates a local address with a socket.

- **Prototype**

```
s32 Ql_SocketBind(s8 socketId, u8 socktype, u16 port);
```

Parameters:

socketId:

Descriptor identifying an unbound socket.

sockettype:

Socket type, which can be 0 (socket stream), 1 (socket datagram).

port:

Socket port number.

- **Return value:**

If no error occurs, this function returns *QL_SOC_SUCCESS (0)*. Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.18. Ql_SocketListen

This function places a socket in a state in which it is listening for an incoming connection..

- **Prototype**

```
s32 Ql_SocketListen(s8 listensocket, u8 address[4], u16 port, u8
maxconnections);
```

Parameters:

listensocket: The listened socket id.
address: Local IPv4 address.
port: Local IPv4 listen port.
maxconnections: Maximum connection number.

- **Return value:**

If no error occurs, this function returns *QL_SOC_SUCCESS (0)*. Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.19. Ql_SocketAccept

This function permits a connection attempt on a socket.

- **Prototype**

```
s8 Ql_SocketAccept(s8 listensocket, u8 address[4], u16 *port);
```

Parameters:

listensocket: [in] The listen socket id.
address: [out] An optional pointer to a buffer that receives the address of the connecting entity.
port: [out] An optional pointer to an integer that contains the port number of the connecting entity.

- **Return value:**

If no error occurs, this function returns a socket Id, which is greater than or equal to zero. Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.20. Ql_GetHostIpbyName

This function retrieves host IP corresponding to a host name.

- **Prototype**

```
s32 Ql_GetHostIpbyName(u8 contextid, u8 *hostname, u8 *addr, u8 *addr_len,
u8 in_entry_num, u8 *out_entry_num, Ql_CallBack_GetIpByName
callback_getipbyname);
```

Parameters:

contextid:

[in] OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter can be 0 or 1.

hostname: [in] The host name.
addr: [out] The buffer of the host IPv4 address.
dddr_len: [out] The length of *addr*.
in_entry_num: [in] address number
out_entry_num: [out] get address number

callback_getipbyname:

[in] This callback is called by Core System to notify whether this function retrieves host IP successfully or not.

Syntax of Callback Function:

```
typedef void (*Ql_CallBack_GetIpByName)(u8 contexid, bool result, s32 error_code, u8 num_entry, u8 *entry_address) ;
```

contextid: OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter can be 0 or 1.
result: TRUE on success, or FALSE on fail
*error*_code: error code if fail
num_entry: get address number.
entry_address: the host IPv4 address.

- **Return value:**

If no error occurs, this return value will be *QL_SOC_SUCCESS* (0). Otherwise, a value of *ql_soc_error_enum* is returned.

However, if the *QL_SOC_WOULDBLOCK* is returned, the application will have to wait till the *callback_getipbyname* is called to know whether this function retrieves host IP successfully or not.

6.8.21. Ql_GetLocalIpAddress

This function retrieves local IP corresponding to the specified PDP context.

- **Prototype**

```
s32 Ql_GetLocalIpAddress(u8 contxtid, u8 ip_addr[4]);
```

Parameters:

contextid:

[in] OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter

can be 0 or 1.

ip_addr:

[out] Pointer to the buffer that is the storage space for the local IPv4 address.

- **Return value:**

If no error occurs, this return value will be *QL_SOC_SUCCESS* (0). Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.22. Ql_GetDnsServerAddress

This function retrieves the DNS server's IP address.

- **Prototype**

```
s32 Ql_GetDnsServerAddress(u8 contxtid, u8 primary_address[4], u8 secondary_address[4]);
```

Parameters:

contextid:

[in] OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter can be 0 or 1.

primary_addr:

[out] Pointer to the buffer that is the storage space for the primary DNS server's IP address.

secondary_addr:

[out] Pointer to the buffer that is the storage space for the secondary DNS server's IP address.

- **Return value:**

If no error occurs, this return value will be *QL_SOC_SUCCESS* (0). Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.23. Ql_SetDnsServerAddress

This function sets the DNS server's IP address.

- **Prototype**

```
s32 Ql_SetDnsServerAddress(kal_uint8 contextId, bool primary_set, u8 primary_address[4], bool secondary_set, u8 secondary_address[4]);
```

Parameters:

contextid:

[in] OpenCPU supports two **PDP-contexts** to the destination host at a time. This parameter can be 0 or 1.

primary_set:

[in] TRUE or FALSE. Enable to set the primary DNS server's IPv4 address.

primary_addr:

[in] Pointer to the buffer that is the storage space for the primary DNS server's IP address

secondary_set:

[in] TRUE or FALSE. Enable to set the secondary DNS server's IPv4 address.

secondary_addr:

[in] Pointer to the buffer that is the storage space for the secondary DNS server's IPv4 address.

- **Return value:**

If no error occurs, this return value will be *QL_SOC_SUCCESS* (0). Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.24. Ql_SocketCheckIp

This function checks whether an IP address is valid IP address or not. If yes, each segment of the IP address string will be converted into integer to store in *ipaddr* parameter.

- **Prototype**

```
s32 Ql_SocketCheckIp(u8 *addressstring, u32* ipaddr)
```

Parameters:

Addressstring:

[in] IP address string.

ipaddr:

[out] Pointer to u32, each byte stores the IP digit converted from the corresponding IP string .

- **Return value:**

QL_SOC_SUCCESS	The IP address string is a valid IP address.
QL_SOC_ERROR	The IP address string is invalid.
QL_SOC_INVAL	Invalid argument

6.8.25. Ql_SocketSetSendBufferSize

This function sets the length of send buffer.

- **Prototype**

```
s32 Ql_SocketSetSendBufferSize(s8 socket, u32 bufferSize);
```

Parameters:

socket: Socket Id.

bufferSize: Length of send buffer, in bytes.

Note:

The send buffer length would better be less than (**8 * 1360**) bytes.

- **Return value:**

If no error occurs, this return value will be *QL_SOC_SUCCESS* (0). Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.26. Ql_SocketSetRecvBufferSize

This function sets the length of receive buffer.

- **Prototype**

```
s32 Ql_SocketSetRecvBufferSize(s8 socket, u32 bufferSize);
```

Parameters:

socket: Socket Id.

bufferSize: Length of receive buffer, in bytes.

Note:

The receive buffer length would better be less than (**8 * 1360**) bytes.

- **Return value:**

If no error occurs, this return value will be *QL_SOC_SUCCESS* (0). Otherwise, a value of *ql_soc_error_enum* is returned.

6.8.27. Ql_GetDeviceCurrentRunState

This function retrieves the current run-state, including SIM card state, network registration state, GPRS network registration state, signal strength and bit error rate.

- **Prototype**

```
void Ql_GetDeviceCurrentRunState(s32 *simcard, s32 *creg, s32 *cgreg, u8
*rssi, u8 *ber);
```

Parameters:

simcard: [out] SIM card state, a value of *Ql_SIM_State*.

creg: [out] Network registion state, a value of *Ql_Reg_State*.

cgreg: [out] GPRS Network registion state, a value of *Ql_Reg_State*.

rssi: [out] Signal strength, unit in dBm.

ber: [out] Bit error rate.

6.8.28. Ql_SocketHtonl

This function reverses the byte order in u32 integer.

- **Prototype**

```
u32 Ql_SocketHtonl(u32 a);
```

Parameters:

a: An u32 integer.

- **Return value:**

An u32 integer reversed.

6.8.29. Ql_SocketHtons

This function reverses the byte order in u16 integer.

- **Prototype**

```
u16 Ql_SocketHtons(u16 a);
```

Parameters:

a: An u16 integer.

- **Return value:**

An u16 integer reversed.

6.8.30. Ql_CallBack_GprsAPNSet

This callback function is invoked by *Ql_GprsAPNSet()*.

- **Prototype**

```
typedef void (*Ql_CallBack_GprsAPNSet)(bool result, s32 error_code);
```

Parameters:

result:

TRUE if the *Ql_GprsAPNSet()* succeeds; Or FALSE

error_code:

Error code. If *result* is FALSE, an error code will be returned in *error_code*.

6.8.31. Ql_CallBack_GprsAPNGet

This callback function is invoked by *Ql_GprsAPNGet()*.

- **Prototype**

```
typedef void (*Ql_CallBack_GprsAPNGet)(u8 profileid, bool result, s32
error_code, u8 *apn, u8 *userId, u8 *password);
```

Parameters:

Profileid:

PDP context profile, which ranges from 0 to 9.

result:

TRUE if the *Ql_GprsAPNGet()* succeeds; Or FALSE.

error_code:

Error code. If *result* is FALSE, an error code will be returned in *error_code*.

apn:

NULL-terminated APN characters.

userId:

User Id, NULL-terminated characters.

password:

Password, NULL-terminated characters.

6.9. MULTITASK API

These interfaces are used for multitask programming. The required header file is *Ql_multitask.h*.

The example in the *example_multitask.c* of OpenCPU SDK shows the proper usages of these methods.

6.9.1. Main Task

The entrance of the main task is *ql_entry()*, please refer to chapter "Least Embedded Application Code".

The stack of main task is defined use *QL_TASK_STACK_SIZE* in ql_customer_config.c.

For example, the following defines the stack of main task is 4K bytes.

```
#define QL_TASK_STACK_SIZE (4*1024)  /* 1K-4K for 32M Flash,
                                        1K-10K for 128M Flash */
// ……
const u32 qlMainTaskPriority = 200; /* 200-255;
                                      the smaller, the greater priority */
const u32 qlMainTaskExtqsize = 10;  // 10-30, length of message queue
```

Note:

- Stack Size

  By default, the stack size is set to 4KB. If some operations with file will be done, the stack size must be set to at least 4KB.

- Message Queue

  If messages are sent to task incessantly, developer should avoid calling these functions:

Ql_Sleep(), *Ql_osTakeSemaphore()* and *Ql_osTakeMutex()*. These functions will block the task, so that the task cannot fetch message from the message queue. If the message queue is filled up, the system will automatically reboot unexpectedly.

6.9.2. Subtasks

The subtask is defined in the array of *QlMutitask SubMutitaskArray[]* in *ql_customer_config.c*. And the maximal number of subtasks is 10.

For example, the following defines the four subtasks

```
QlMutitask SubMutitaskArray[] =  /* max 10 subtasks */
{
    {emaple_subtask1_entry,   1024/*TaskStackSize*/,   201/*TaskPriority*/,
11/*TaskExtqsize*/},
    {emaple_subtask2_entry, 1024, 200, 12},
    {emaple_subtask3_entry, 1024, 200, 13},
    {emaple_subtask4_entry, 1024, 200, 14},
    {NULL, 0},  //  Must be "NULL" at the end
};
```

```
typedef struct QlMutitaskTag
{
    void (*MultiTaskEntry)(s32 TaskId); // The subtask entry point,  TaskId is
between 1 and 10.
    u32  TaskStackSize;     // the subtask stack size
    u32  TaskPriority;      // the task priority
    u32  TaskExtqsize;      // the length of the message queue
}QlMutitask;
```

WARNING:

Strongly recommended, developer should set the priorities of subtasks to the same as that of the main task usually. An improper set of priorities of tasks probably brings on some unexpected problems, such as subtasks' cannot be booted, jobs cannot be executed, and so on.

6.9.3. Possible Error Code

The frequent error-codes, which APIs in multitask programming could return, are enumerated in the *ql_os_error_enum* as below.

```
typedef enum {
   OS_SUCCESS,
   OS_ERROR,
   OS_Q_FULL,
   OS_Q_EMPTY,
   OS_SEM_NOT_AVAILABLE,
   OS_WOULD_BLOCK,
   OS_MESSAGE_TOO_BIG,
   OS_INVALID_ID,
   OS_NOT_INITIALIZED,
   OS_INVALID_LENGHT,
   OS_NULL_ADDRESS,
   OS_NOT_RECEIVE,
   OS_NOT_SEND,
   OS_MEMORY_NOT_VALID,
   OS_NOT_PRESENT,
   OS_MEMORY_NOT_RELEASE
} ql_os_error_enum;
```

6.9.4. Ql_osSendEvent

This function passes messages between tasks. The destination task will receive *EVENT_MSG* through *Ql_GetEvent*.

- **Prototype**

```
s32 Ql_osSendEvent(s32 desttaskid, u32 data1, u32 data2);
```

Parameters:

desttaskid:

The maximum value is 10. The destination task is main task if the value is 0. The destination task is subtask if the value is between 1 and 10.

data1: user data.

data2: user data.

- **Return value:**

The return value is *OS_SUCCESS* if this function succeeds. Otherwise an error code is returned.

Possible values are:

OS_SUCCESS

OS_INVALID_ID

OS_MEMORY_NOT_VALID

OS_Q_FULL

6.9.5. Ql_osCreateMutex

This function creates a MUTEX.

- **Prototype**

```
u32 Ql_osCreateMutex(char *mutexname);
```

Parameters:
mutexname: name of the MUTEX to be created.

- **Return value:**

The created MUTEX id.

6.9.6. Ql_osTakeMutex

This function obtains an instance of the specified MUTEX.

- **Prototype**

```
void Ql_osTakeMutex(u32 mutexid);
```

Parameters:
mutexid: destination MUTEX to be taken.

6.9.7. Ql_osGiveMutex

This function releases an instance of the specified MUTEX.

- **Prototype**

```
void Ql_osGiveMutex(u32 mutexid);
```

Parameters:
mutexid: destination MUTEX to be given.

6.9.8. Ql_osCreateSemaphore

This function creates a counting semaphore.

- **Prototype**

```
u32 Ql_osCreateSemaphore(char *semname, u32 initial_count);
```

Parameters:

semname: name of the semaphore to be created.

initial_count: the semaphore initial value.

- **Return value:**

The created semaphore.

6.9.9. Ql_osTakeSemaphore

This function obtains an instance of the specified semaphore.

- **Prototype**

```
u32 Ql_osTakeSemaphore(u32 semid, bool wait);
```

Parameters:

semid: the destination semaphore to be taken.

wait: the waiting style determines if a task waits infinitely or returns immediately.

- **Return value:**

The return value is *OS_SUCCESS* if this function succeeds. Otherwise an error code is returned.

Possible values are:

OS_SUCCESS

OS_SEM_NOT_AVAILABLE

6.9.10. Ql_osGiveSemaphore

This function releases an instance of the specified semaphore.

- **Prototype**

```
void Ql_osGiveSemaphore(u32 semid);
```

Parameters:

semid: the destination semaphore to be given.

6.10. FOTA API

The interface functions enumerated in the section are designed for FOTA (Firmware Over The Air) function. The required head file is *ql_fota.h*.

The example in the *example_fota.c* of OpenCPU SDK shows the proper usages of these methods.

Note:
The FOTA function is available only in the module with 128 + 32 flash.

6.10.1. Ql_Fota_Core_Init

This function initializes the FOTA-Core related functions. Applications must initialize the FOTA function before they call other FOTA-Core related functions.

- **Prototype**

```
extern void Ql_Fota_Core_Init(void);
```

6.10.2. Ql_Fota_Core_Write_Data

This function writes the delta data of Cores to the special space in the module.

- **Prototype**

```
extern s32  Ql_Fota_Core_Write_Data(s32 length, s8* buffer);
```

length:
The length of writing (Unit: Bytes)

buffer:
Pointer to the start address of data buffer to write.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise an error code is returned.

6.10.3. Ql_Fota_Core_Finish

This function tells Core System that the application completed writing the delta data of Cores.

- **Prototype**

```
extern void Ql_Fota_Core_Finish(void);
```

6.10.4. Ql_Fota_App_Init

This function initializes the FOTA-Application related functions:

1. Initialize data structure and progress initial step
2. Register customized authentication function or encryption function

- **Prototype**

```
extern void Ql_Fota_App_Init(void);
```

6.10.5. Ql_Fota_App_Write_Data

This function writes the delta data of applications to the special space in the module.

- **Prototype**

```
extern s32  Ql_Fota_App_Write_Data(s32 length, s8* buffer);
```

length:

The length of writing (Unit: Bytes)

buffer:

Pointer to the start address of data buffer to write.

- **Return value:**

The return value is *QL_RET_OK* if this function succeeds. Otherwise an error code is returned.

6.10.6. Ql_Fota_App_Finish

This function tells Core System that the application completed writing the delta data of applications

- **Prototype**

```
extern void Ql_Fota_App_Finish(void);
```

6.10.7. Ql_Fota_Update

This function starts FOTA updating process.

- **Prototype**

```
extern s32  Ql_Fota_Update(u32 flag);
```

flag:

A flag, which specifies the action to take when starting FOTA updating. Developer can combine options listed below by using the bitwise-OR (|) operator.

FOTA_UPDATE_FLAG_CORE	Only update Core System software
FOTA_UPDATE_FLAG_APP	Only update Application software

6.11. DEBUG API

The head file *ql_trace.h* must be included so that the debug functions can be called.

The examples in OpenCPU SDK show the proper usages of these methods

6.11.1. Debug Mode

There are two working mode for UART2: BASE MODE and ADVANCE MODE.

```
typedef enum QlDebugModeTag
{
   BASIC_MODE,
   ADVANCE_MODE
} QlDebugMode;
```

Under basic mode, application debug messages will be output as text through UART2 port. And the UART2 port can work like UART3.

Under advance mode, both application debug messages and system debug messages will be output through UART2 port with special format. The Catcher Tool provided by Quectel can be used to capture and analyze these messages.

Developers can set the working mode of UART2 by calling *Ql_SetDebugMode()* with a parameter of *QlDebugMode*.

6.11.2. Ql_SetDebugMode

This function sets debug mode.

- **Prototype**

```
void Ql_SetDebugMode(QlDebugMode mode);
```

- **Parameter**

mode: the debug mode type.

6.11.3. Ql_DebugTrace

This function formats and prints a series of characters and values through the debug serial port (UART2).

- **Prototype**

```
s32 Ql_DebugTrace (char *fmt, ... );
```

- **Parameter**

format:

Format-control string.

A format specification has the following form:

%type

type, a character that determines whether the associated argument is interpreted as a character, a string, or a number.

Character	Type	Output Format
c	**int**	Specifies a single-byte character.
d	**int**	Signed decimal integer.
o	**int**	Unsigned octal integer.
x	**int**	Unsigned hexadecimal integer, using "abcdef."
f	**double**	Float point digit.
p	Pointer to **void**	Prints the address of the argument in hexadecimal digits.

- **Return value**

Number of characters printed.

Remarks

The string to be printed must not be larger than the maximum number of bytes allowed in buffer; otherwise, a buffer overrun can occur.

The maximum allowed number of characters to output is 512.

To print a 64-bit integer, please first convert it to characters using Ql_ *Ql_sprintf()*, and then print the characters of the 64-bit integer.

6.12. STANDARD LIBRARY API

Standard library API functions are defined in the file *ql_stdlib.h*.

```
#define    Ql_sprintf                sprintf

void    Ql_itoa(char **buf, s32 i, s32 base);
char* Ql_ui64toa(u64 val, char* str, s32 radix);
void    Ql_itof(char **buf, s32 i);
char* Ql_strcpy(char* dest, const char* src);
char* Ql_strncpy(char* dest, const char* src, u16 size);
char* Ql_strcat(char* s1, const char* s2);
char* Ql_strncat(char* s1, const char* s2, u16 size);
u16    Ql_strlen(const char* str);
u32    Ql_strcmp(const char*s1, const char*s2);
s32    Ql_strncmp(const char* s1, const char* s2, u16 size);
void* Ql_memset(void* dest, u8 value, u32 size);
void* Ql_memcpy(void* dest, const void* src, u16 size);
s32    Ql_memcmp(const void* dest, const void*src, u16 size);
void* Ql_memmove(void* dest, const void* src, u16 size);
```

Note:

Above standard I/O functions are almost identical to the corresponding standard C functions, and the only difference is that these functions use Quectel defined types instead of standard C types.

7. SYSTEM CONFIGURATION

In the file *ql_customer_gpio.c*, developer can set the initial status of the Embedded Application by configuring the '*Customer_user_qlconfig*' structure with appropriate parameters.

7.1.1. Configure 'Customer_user_qlconfig'

The *Customer_user_qlconfig* structure is defined as below:

```
typedef struct QlCustomerConfigTag
{
    QlPinName handfreeamplifierpin;
    QlMicName handfreeinputmic;
    QlPinName headsetdetectpin;
    bool headsetadccapture;
    u8 headsetdetectdebounce;
    QlAdcName headsetdetectadc;
    u32 headsetadchigh;
    u32 headsetadclow;
    u32 headsetadcsendkey;
    bool powerautoon;
    bool powerautooff;
     bool uart3supportvfifo;
}QlCustomerConfig;
```

handfreeamplifierpin:

Designate a GPIO pin to control PA.

If this field is set to *QL_PINNAME_MAX*, no pin will control PA. Or, you need to set this field to a GPIO pin. Beside, you need to configure the variable: *Customer_QlPinConfigTable*.

For example:

- Step 1:

Set '*handfreeamplifierpin'* to *QL_PINNAME_GPIO1*.

- Sep 2:

Configure the variable: *Customer_QlPinConfigTable* as below.

{QL_PINNAME_GPIO1,QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULLENABLE_ENABLE, QL_PINDIRECTION_OUT,QL_PINLEVEL_LOW,0}.

handfreeinputmic:

Specify the audio source, *QL_MIC_MIC1* or *QL_MIC_MIC2*.

headsetdetectpin:

Designate some GPIO pin to detect earphone's plug-in and plug-out.

If this field is set to *QL_PINNAME_MAX*, no pin will detect earphone's action. Or, you need to set this field to a GPIO pin. Besides, you need to configure the variable: *Customer_QlPinConfigTable*. For example:

- Step 1:

Set '*headsetdetectpin*' to *QL_PINNAME_GPIO0*.

- Sep 2:

Configure the variable: *Customer_QlPinConfigTable* as below.

{QL_PINNAME_GPIO0, QL_PINSUBSCRIBE_UNSUB, QL_PINMODE_1, QL_PINPULL, ENABLE_ENABLE, QL_PINDIRECTION_IN, 0, 0}.

- Sep 3:

Enable GPIO pin to bear the attributes of Interruption and Input Direction.

If the earphone detection function is enabled, Embedded Application will receive the EVENT_HEADSET event when a plug-in or plug-out action of earphone is taken.

headsetadccapture:

FALSE indicates: not capture headset ADC value. This is the default setting.

If you need to test MIC channel, you can set this field to TRUE. When the earphone is plug in or dialing, the EVENT_HEADSET (HEADSET_ADC) with the sampling value of ADC will be trigged.

headsetdetectdebounce:

Set the bounce time of plugging in or out earphone, unit in 10ms, the maximum possible value is 255.

headsetdetectadc:

ADC type, *QL_ADC_ADC0* or *QL_ADC_TEMP_BAT*.

Note:

If the field 'headsetdetectpin' is set to *QL_PINNAME_MAX*, here setting will be useless.

headsetadchigh:

headsetadclow:

The upper two fields define the upper limit and lower limit of a valid ADC sampling value respectively. When a valid ADC value is sampled, the EVENT_HEADSET (HEADSET_ADC) with the sampling value of ADC will be trigged.

headsetadcsendkey:

This field defines the maximum level value during a coming call or the device being in calling. When the sampled ADC value is lower than the value that this field defines, the EVENT_HEADSET (HEADSET_ADC) with the sampling value of ADC will be trigged to indicate that the button on earphone is pressed down.

powerautoon:

When set to TRUE, the system will boot up automatically after a press is given to the power key, and Embedded Application doesn't need to do anything.

When set to FALSE, the EVENT_POWERKEY (POWERKEY_ON) event will be trigged. After receive this event, Embedded Application must call *Ql_PowerOnAck()* to enable the system to boot up before pressing the button on earphone is released. Or, the system will power down after pressing the button on earphone is released.

powerautooff:

When set to TRUE, the system will switch off automatically after a press is given to the PWRKEY key.

When set to FALSE, the EVENT_POWERKEY (POWERKEY_ON) event will be trigged. After receive this event, Embedded Application may do post processing before the system switches off.

uart2supportvfifo:

Must be TRUE.

7.1.2. Example for Headset

Configure 'Customer_user_qlcomfig'
The following configuration is an example:

```
const QlCustomerConfig Customer_user_qlconfig =
{
   QL_PINNAME_MAX,     /* handfreeamplifierpin */
   QL_MIC_MIC1,        /* handfreeinputmic */
   QL_PINNAME_MAX,     /* headsetdetectpin */
   FALSE,              /* headsetadccapture */
   100,                /* headsetdetectdebounce */
   QL_ADC_MAX,         /* headsetdetectadc */
   2800000,            /* headsetadchigh */
   500000,             /* headsetadclow */
   300000,             /* headsetadcsendkey */
   ...
   TRUE,               /* uart3supportvfifo */
};
```

When earphone is plug in or plug out

```
...
Ql_GetEvent(&qlEventBuffer);
switch(qlEventBuffer.eventType)
{
  case EVENT_HEADSET:
     Headset_Event* pHeadsetEvt = & qlEventBuffer.eventType. headset_evt;
     if ( HEADSET_PLUGIN ==  pHeadsetEvt->headsettype)
     {
       // Set voice channel to headset
       Ql_VoiceCallChangePath(QL_AUDIO_PATH_HEADSET);
       // ...
     }
     else  if ( HEADSET_PLUGOUT ==  pHeadsetEvt->headsettype)
     {
       // Set voice channel to loaderspeaker
        Ql_VoiceCallChangePath(QL_AUDIO_PATH_LOUDSPEAKER);
       // ...
     }
    break;
  default:
    break;
}
...
```

When press the button on earphone during a coming call or being in calling

```
...
Ql_GetEvent(&qlEventBuffer);
switch(qlEventBuffer.eventType)
{
  case EVENT_HEADSET:
     Headset_Event* pHeadsetEvt = &qlEventBuffer.eventType.headset_evt;
     if (HEADSET_KEYPRESS ==  pHeadsetEvt->headsettype)
     {
       // Answer the coming call
       // ...
     }
     else  if (HEADSET_KEYRELEASE ==  pHeadsetEvt->headsettype)
     {
       // End the active call
       // ...
     }
    break;
  default:
    break;
}
...
```

8. Appendix - ERROR CODES

The *ql_error.h* defines all the error codes that API functions probably return.

Error code	Error value	Description
QL_RET_OK	0	Normally return.
QL_RET_ERR_PARAM	-1	Parameter error
QL_RET_ERR_PORT_NOT_OPEN	-2	Port not opened
QL_RET_ERR_TIMER_FULL	-3	All timers are used up
QL_RET_ERR_INVALID_TIMER	-4	Invalid timer
QL_RET_ERR_FATAL	-5	Unknown fatal error
QL_RET_ERR_INVALID_OP	-6	Invalid operation
QL_RET_ERR_UART_BUSY	-7	UART is busy
QL_RET_ERR_INVALID_PORT	-8	Invalid serial port
QL_RET_ERR_NOMATCHVERSION	-9	No match pin version
QL_RET_ERR_NOSUPPORTPIN	-10	Not supported pin operation
QL_RET_ERR_NOSUPPORTMODE	-11	Not supported pin mode
QL_RET_ERR_NOSUPPORTEINT	-12	Not supported EINT
QL_RET_ERR_NOSUPPORTSET	-13	Not supported pin setting
QL_RET_ERR_NOSUPPORTCONTROL	-15	Not supported pin control
QL_RET_ERR_PINALREADYSUBCRIBE	-16	Pin already subscribed
QL_RET_ERR_BUSSUBBUSY	-17	BUS already subscribed
QL_RET_ERR_NOGPIOMODE	-18	Not GPIO mode
QL_RET_ERR_NORIGHTOPERATE	-19	Wrong PIN operation
QL_RET_ERR_ALREADYUNSUBCRIBE	-20	Already unsubscribed
QL_RET_ERR_FULLI2CBUS	-21	I2C BUS is full
QL_RET_ERR_NOTSUPPORTBYHANDLE	-22	Not supported handle
QL_RET_ERR_INVALIDBUSHANDLE	-23	Invalid BUS handle
QL_RET_ERR_NOEXISTOBJEXT	-24	Flash object not exist
QL_RET_ERR_OPERATEOBJEXTFAILED	-25	Fail to operate flash
QL_RET_ERR_OPENOBJEXTFAILED	-26	Fail to open flash object
QL_RET_ERR_WRITEOBJEXTFAILED	-27	Fail to write flash object
QL_RET_ERR_READOBJEXTFAILED	-28	Fail to read flash object
QL_RET_ERR_FLASHFULLOVER	-29	User space not enough
QL_RET_ERR_FLASHSPACE	-30	Flash space error
QL_RET_ERR_DRIVE	-31	Flash operation fatal
QL_RET_ERR_DRIVEFULLOVER	-32	Insufficient Flash space
QL_RET_ERR_INVALIDFLASHID	-33	Invalid flash ID
QL_RET_ERR_I2CHWFAILED	-34	Fail to operate I2C
QL_RET_ERR_FILEFAILED	-35	Fail to operate file
QL_RET_ERR_FILEOPENFAILED	-36	Fail to open file
QL_RET_ERR_FILENAMETOOLENGTH	-37	Too long file name

QL_RET_ERR_FILEREADFAILED	-38	Fail to read file
QL_RET_ERR_FILEWRITEFAILED	-39	File write failed
QL_RET_ERR_FILESEEKFAILED	-40	File seek failed
QL_RET_ERR_FILENOTFOUND	-41	File not found
QL_RET_ERR_FILENOMORE	-42	File no more
QL_RET_ERR_FILEDISKFULL	-43	Disk is full
QL_RET_ERR_INVALID_BAUDRATE	-44	Invalid baud rate

Shanghai Quectel Wireless Solutions Ltd.

Room 801, Building E, No.1618, Yishan Road, Shanghai, China 201103
Tel: +86 21 5108 2965
Mail: info@quectel.com