AN12390
ALIOS Porting Guide - Based on LPCXpresso5410x + GT202

Rev. 0 — April 2019

Application Note

Contents

1 Introduction

AliOS Things is Alibaba's IoT version of AliOS family. It was announced in 2017 by Alibaba Cloud, and open sourced in 20th, October, 2017 under APACHE2.0 license at github (https://github.com/alibaba/AliOS-Things). The IoT devices with AliOS Things integrated can connect to Alibaba Cloud (ACloud) and utilize ACloud services.

1.1 AliOS Things architecture

AliOS Things consists of the following components in the layered architecture, as shown in Figure 1. on page 1.

- **Board Support Package (BSP)**: mainly developed and maintained by SoC Vendors
- **Hardware Abstraction Layer (HAL)**: like Flash, UART
- **Kernel**: Rhino RTOS Kernel, Yloop, VFS, KV Storage
- **Protol Stack**: LwIP, uMesh mesh networking stack
- **Security**: TLS, Trusted Framework Service(TFS), Trusted Execution Environment (TEE)

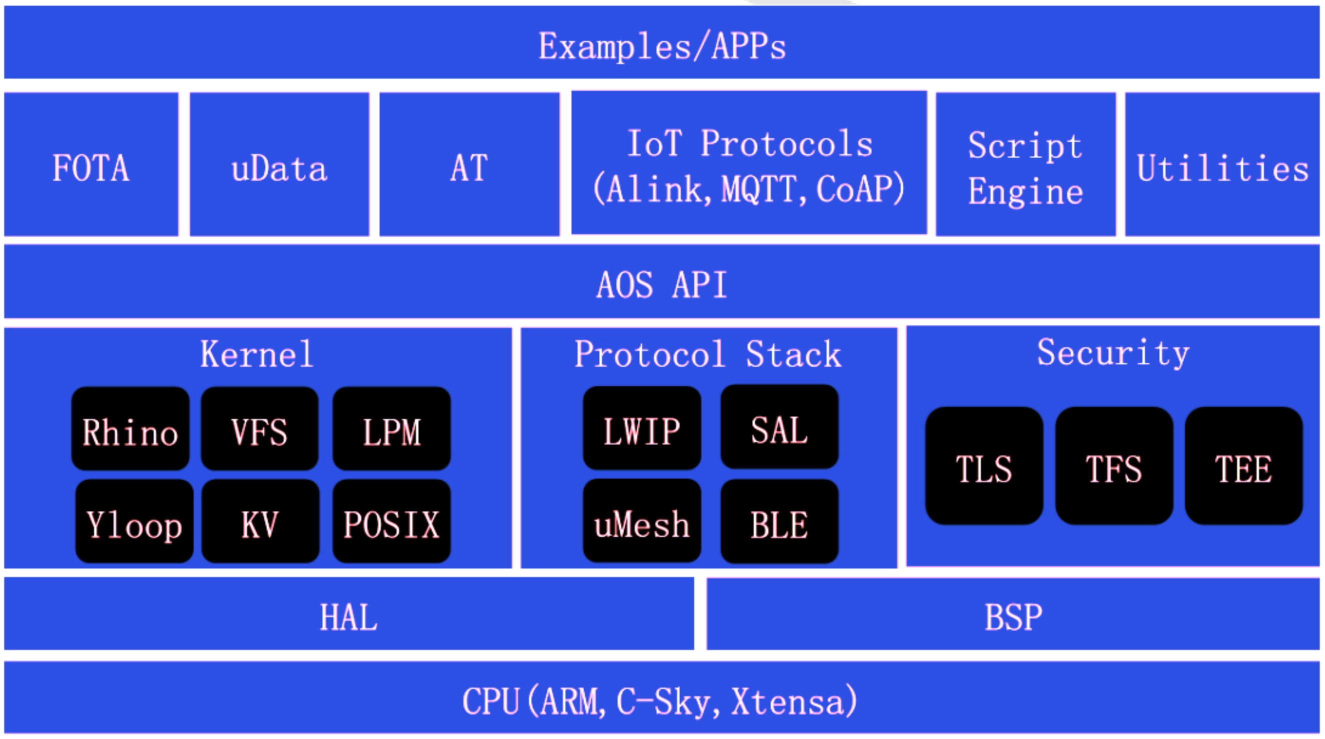
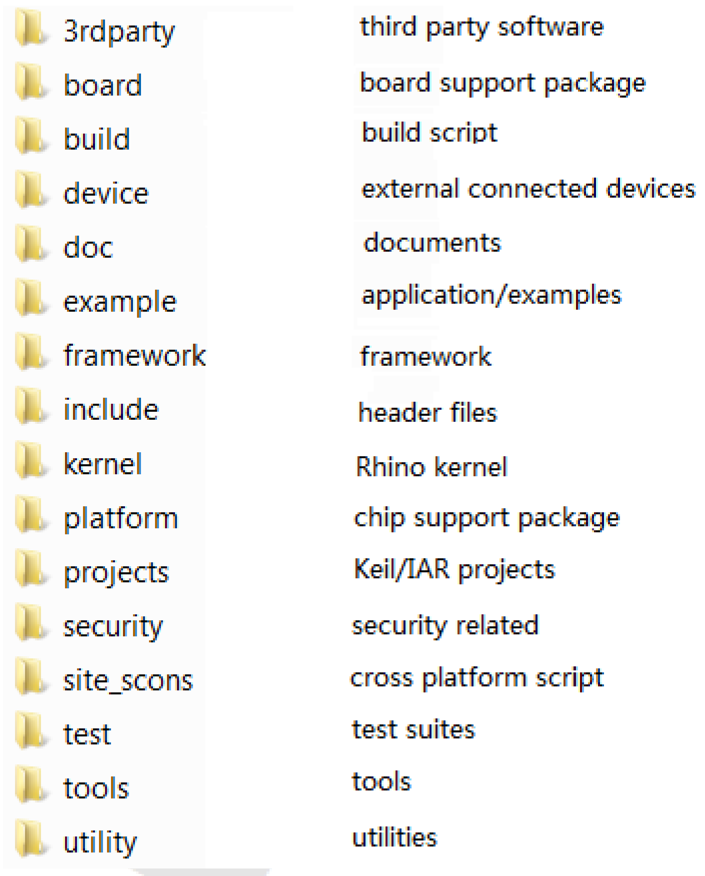


Figure 1. AliOS Things block diagram

1.2 AliOS Things folder structure

Taking AliOS Things version 1.3.2 as example, Figure 2. on page 2 shows the structure of AliOS Things root folder. Notes are added to explain the stuff under the folder.



3rdparty	third party software
board	board support package
build	build script
device	external connected devices
doc	documents
example	application/examples
framework	framework
include	header files
kernel	Rhino kernel
platform	chip support package
projects	Keil/IAR projects
security	security related
site_scons	cross platform script
test	test suites
tools	tools
utility	utilities

Figure 2. AliOS Things folder structure

1.3 AliOS Things development

The tutorial for AliOS Things development is explained on https://github.com/alibaba/AliOS-Things/wiki/Quick-Start, where you can find the information about **environment setup** and **build procedures**.

2 AliOS Things kernel porting

2.1 Rhino

Rhino is RTOS kernel in AliOS Things. Same to other real-time OS, Rhino supplies basic functionalities like task management (task creation and deletion), task communication (queue, condition, semaphore, mutex), and memory management (heap allocation and free). Also, utilities like software timer, lower power management is designed.

2.1.1 CPU architecture

AliOS Things supports various CPU architectures, like Arm®, MIPS, RL78, XTENSA. Specifically, in Arm processors, Armv5, Armv6m, Armv7a and Armv7m are supported by the software release TAG1.2.2. Armv8m support is coming soon.

As to CPU architecture porting, there is Arm Cortex®-M4 processor in LPC54101. Arm Cortex M4 is with Armv7m architecture. As Armv7m is already supported by AliOS Things, there is no extra work for CPU porting.

2.1.2 Tick interrupt

All real-time kernel needs tick interrupt for context switch. In Arm CPU, you can use SYSTICK interrupt.

2.1.2.1 Configuring SYSTICK timer

SYSTICK timer must be configured and enabled before the `aos_start()` function. The `aos_start()` function starts OS scheduler which relies on TICK. The scheduler can check the task state and decide the context switch in TICK ISR. TICK period is decided by the application.

2.1.2.2 TICK interrupt

The `krhino_tick_proc()` function must be called in the TICK interrupt function so that Rhino kernel can perform context switch in period. The reference code for TICK interrupt is as follows (using SYSTICK ISR for TICK).

```
void tick_interrupt(void) {
    krhino_intrpt_enter();
    krhino_tick_proc();
    krhino_intrpt_exit();
}
```

NOTE

`krhino_intrpt_enter()` and `krhino_intrpt_exit()` must be wrapped for all enabled interrupts.

3 Hardware Abstraction Layer (HAL)

HAL is designed in AliOS Things to adapt various boards and platforms. HAL APIs are enclosed in the header files like `adc.h`, `gpio.h`, and `i2c.h (\root\include\hal\*)`. NXP's MCUXpresso SDK is used (https://mcuxpresso.nxp.com) for AliOS Things HAL development.

3.1 UART and LOG

System log is dependent to UART HAL implementation. The standard input and output function, `printf` in C library, routes to the `hal_uart_send` function in AliOS Things. ALI CLI works like a user shell. It uses the `hal_uart_recv_II` function to get user commands for debugging. The UART HAL allows synchronized data transmission with user defined timeout. No callback based asynchronous UART transmission is supported.

For UART HAL implementation, in LPC54102 MCUXpresso SDK, UART can support blocking transmission without considering the timeout mechanism. Thus, peripheral functional APIs and raw registers access may be inevitable in the case.

3.2 Flash

FLASH HAL implementation is crucial to device over-the-air firmware update and Key-Value storage. FLASH HAL is a unified interface for persistent storage like on-chip FLASH, external QSPI FLASH and EEPROM. Logical partition objects represent FLASH usage for application. A `hal_logic_partition_t` object must be defined per usage. Read and write permission is required to be explicitly assigned.

```
hal_logic_partition_t hal_logic_partition[HAL_PARTITION_MAX] =
{
    {HAL_FLASH_EMBEDDED, "Bootloader", 0, 0, PAR_OPT_WRITE_DIS|PAR_OPT_READ_DIS},
    {HAL_FLASH_EMBEDDED, "Application", 0x40000, 0x40000, PAR_OPT_WRITE_EN|PAR_OPT_READ_EN},
    {HAL_FLASH_NONE, NULL, 0, 0, PAR_OPT_WRITE_DIS|PAR_OPT_READ_DIS},
    {HAL_FLASH_NONE, NULL, 0, 0, PAR_OPT_WRITE_DIS|PAR_OPT_READ_DIS},
    {HAL_FLASH_NONE, NULL, 0, 0, PAR_OPT_WRITE_DIS|PAR_OPT_READ_DIS},
    {HAL_FLASH_NONE, NULL, 0, 0, PAR_OPT_WRITE_DIS|PAR_OPT_READ_DIS},
    {HAL_FLASH_NONE, NULL, 0, 0, PAR_OPT_WRITE_DIS|PAR_OPT_READ_DIS},
    {HAL_FLASH_NONE, NULL, 0, 0, PAR_OPT_WRITE_DIS|PAR_OPT_READ_DIS},
    {HAL_FLASH_NONE, NULL, 0, 0, PAR_OPT_WRITE_DIS|PAR_OPT_READ_DIS},
    {HAL_FLASH_NONE, NULL, 0, 0, PAR_OPT_WRITE_DIS|PAR_OPT_READ_DIS},
};
```

The structure of `hal_logic_partition_t` is described as follows.

```
typedef struct {
    hal_flash_t partition_owner; /* Physical media for storage */
    const char *partition_description; /* Text description */
    uint32_t    partition_start_addr; /* Partition start address */
    uint32_t    partition_length; /* Partition length */
    uint32_t    partition_options; /* READ/WRITE Permission */
} hal_logic_partition_t;
```

All FLASH HAL APIs take the `hal_logic_partition_t` object name as the first parameter. The FLASH HAL implementation maps the connection between logical partition and physical media storage and converts the logical partition operation to FLASH read/write operation.

For LPC54102 implementation, FLASH IAP methods, used for FLASH programing in application, have rules on FLASH programming.

- They only allow FLASH erase and write starting from FLASH page boundary per on-chip FLASH characteristic.
- They only allow the write length of 256 bytes, 512 bytes, 1024 bytes or 4096 bytes.

  However, there is no limitation from FLASH HAL caller for the write starting address and size to write. FLASH HAL resolves these IAP limitations. One possible solution is to use an intermediate buffer in FLASH HAL.

  1. A new buffer is allocated.
  2. FLASH data is retrieved in the buffer and new data are concatenated backwards or forwards.
- The concatenated data can be written using FLASH IAP methods.

Two things are important for FLASH HAL implementation.

- `__disable_irq()/__enable_irq()` is wrapped around FLASH IAP methods according to LPC5410x UM.
- Switch off the FLASH controller clock, `kCLOCK_Flash` when FLASH programming operation has completed for power efficiency.

4 Wi-Fi

4.1 Overview

GT202 Kit is used for the Wi-Fi connectivity. GT202 kit can be installed on LPCXpresso54102 board via Arduino interface, as shown in Figure 3. on page 5.

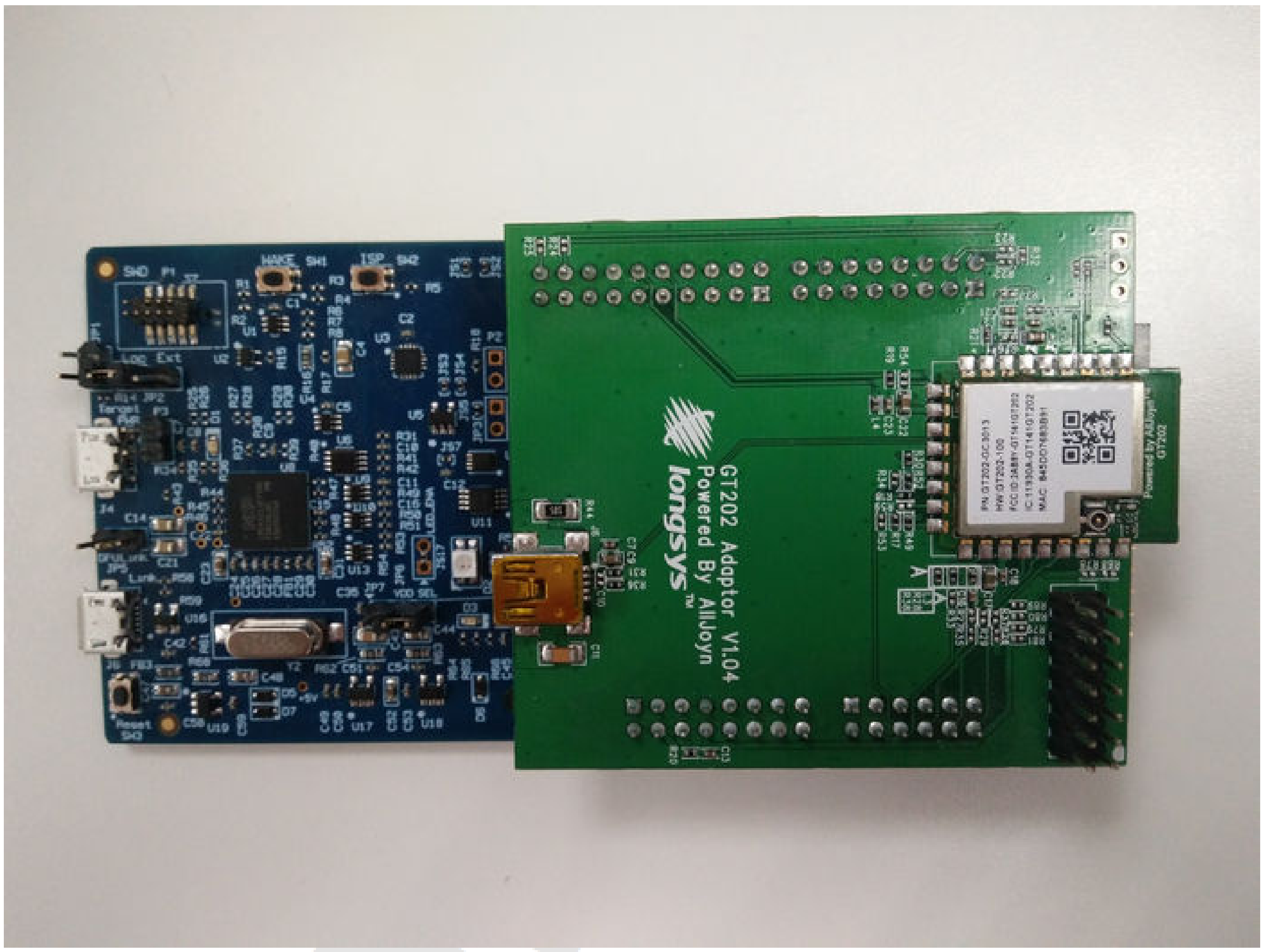


Figure 3. Installing GT202 Wi-Fi KIT on LPCXpresso54102 via Arduino interface

On GT202 Wi-Fi kit, there is a QCA4002 Wi-Fi module from Qualcomm. Figure 4. on page 6 shows the hardware connection between LPC54102 and QCA4002 Wi-Fi module. One set of SPI and two GPIO pins are used. The PWD pin controls the QCA4002 power mode. QCA4002 is active when the pin is pulled high. QCA4002 can notify LPC54102 for incoming events by pin IRQ and SPI is for bi-directional communication between QCA4002 and LPC54102.

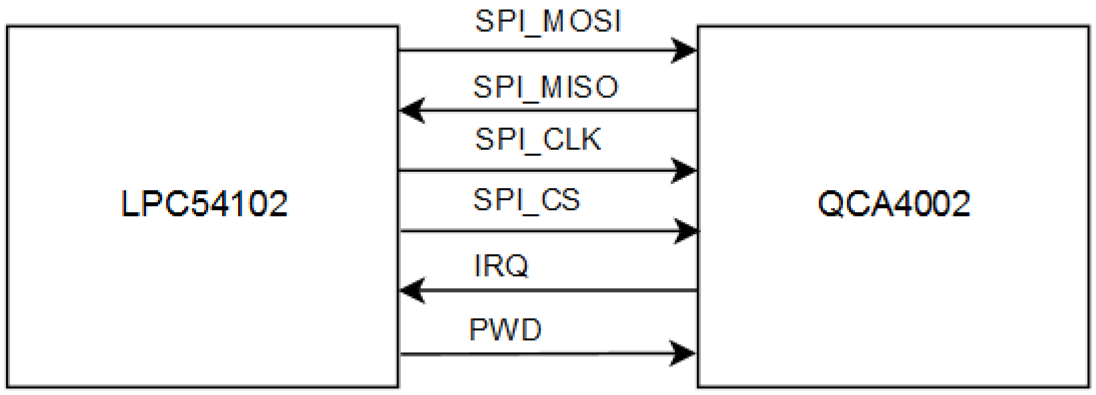


Figure 4. Hardware connection between LPC54102 and QCA4002

4.2 Atheros WMI middleware

Atheros middleware implements QCA4002 device driver. Some MCU parts (e.g. LPC5460X, K64) support Atheros QCA middleware in MCUXpresso SDK.

To deploy Atheros middleware in AliOS Things for a new board (e.g. LPCXpresso54102), platform dependent code in the `wifi_qca\port` folder should be developed as follows.

1. **chip support and board support**

   SPI platform driver (SPI FIFO, SPI DMA), PIN interrupt and GPIO operation may be different among different microcontrollers. The driver might be adapted to new MCU part. As the driver is implemented using MCUXpresso SDK, the chip support porting should be simple. For SPI, peripheral FIFO implementation can be slightly different.

   New PIN MUX and IOCFG should be created according to the board setup.

2. **critical region**

   WMI middleware requires methods for critical region. Entering critical region/Exiting critical region macros are declared in the `wifi_env.h`.

   ```
   #define OSA_EnterCritical(x) { RHINO_CRITICAL_ENTER(); }
   #define OSA_ExitCritical(x) { RHINO_CRITICAL_EXIT();}
   ```

   Rhino kernel supports critical region, `RHINO_CRITICAL_ENTER()` and `RHINO_CRITICAL_EXIT()`) are declared in rhino kernel header file, `k_critical.h`.

   NOTE

   `A_ENTER_CRITICAL/A_EXIT_CRITICAL` may be used for critical region methods in the coming release.

3. **mutex**

   Rhino kernel supports mutex natively, Atheros WMI mutex can be implemented by Rhino kernel functions. Table 1. Mutex in Atheros and WMI and Rhino on page 7 described the relation between Atheros WMI mutex and Rhino mutex functions. Atheros WMI mutex implementation is in the file of `wifi_qca\port\env\AliOS\wifi_env.c`.

Table 1. Mutex in Atheros and WMI and Rhino

Atheros WMI	Rhino
a_mutex_init	krhino_mutex_create
a_mutex_acquire	krhino_mutex_lock
a_mutex_release	krhino_mutex_unlock
a_mutex_delete	krhino_mutex_del

4. **event**

   Rhino kernel supports event natively, Atheros WMI event can be implemented by Rhino kernel functions. Table 2. Event in Atheros and WMI and Rhino on page 7 describes the relation between Atheros WMI event and Rhino event functions. Atheros WMI event implementation is in the file of `wifi_qca\port\env\AliOS\wifi_env.c`.

Table 2. Event in Atheros and WMI and Rhino

Atheros WMI	Rhino
a_event_init	krhino_event_create
a_event_set	krhino_event_set with flag of RHINO_OR
a_event_clear	krhino_event_set with flag of RHINO_AND
a_event_wait	krhino_event_get
a_event_delete	krhino_event_del

5. **time**

   Atheros WMI uses time for debug information. The `a_time_get_msec` function is used for retrieving time in milli-second. The current implementation by Rhino kernel TICK function `krhino_sys_tick_get` which is less accurate but OK for debug propose.

6. **task creation and delay**

   Atheros WMI needs to create a task for executing network transmission requests asynchronously. The driver task, `Atheros_Driver_Task`, is implemented in `driver_main.c`. The task creation function, `Driver_CreateThread`, is done in `wifi_driver_main.c` using Rhino kernel function `krhino_task_create`.

   The recommended stack size for WMI driver task is 256 words. The task priority is higher than user application and any other tasks in ethernet data path to avoid network package lost.

   Task delay function of `a_task_delay` in Atheros WMI is in line with Rhino kernel task sleep function, `krhino_task_sleep`.

4.3 Wi-Fi HAL

Wi-Fi HAL implementation is necessary when Wi-Fi connection is required. In Wi-Fi HAL, all operations and interfaces are capsulated in the structure of `hal_wifi_module_t` in `wifi_hal.h`. A `hal_wifi_module_t` instance must be created in Wi-Fi HAL to adapt GT202 Wi-Fi module. The `hal_wifi_module_t` module consists of Wi-Fi HAL callback and Wi-Fi HAL interface.

4.3.1 Callback

`Netmgr` means that the net manager controls the WiiFi module in AliOS Things framework. The `Netmgr` module can register Wi-Fi HAL module and invoke Wi-Fi HAL interface. It can register the `hal_wifi_event_cb_t` callback function to Wi-Fi HAL. When Wi-Fi is starting and running, Wi-Fi HAL can invoke this registered callback function to report events to the `netmgr` module in a task context.

The Wi-Fi HAL callback function prototype is declared in `wifi_hal.h`.

```
typedef struct {
    /* Notify connection failure reason */
void (*connect_fail)(hal_wifi_module_t *m, int err, void *arg);
/* Notify IP when IP is assigned */
void (*ip_got)(hal_wifi_module_t *m, hal_wifi_ip_stat_t *pnet, void *arg);
/* Notify the WiFi state change */
void (*stat_chg)(hal_wifi_module_t *m, hal_wifi_event_t stat, void *arg);
/* Notify AP information when channel scan is completed */
    void (*scan_compeleted)(hal_wifi_module_t *m, hal_wifi_scan_result_t *result, void *arg);
/* Notify detailed AP information when channel scan is completed */
    void (*scan_adv_compeleted)(hal_wifi_module_t *m,
hal_wifi_scan_result_adv_t *result, void *arg);
   /* Notify AP Information when info changed */
void (*para_chg)(hal_wifi_module_t *m, hal_wifi_ap_info_adv_t *ap_info, char *key, int key_len, void
*arg);
/* Notify the fatal error */
    void (*fatal_err)(hal_wifi_module_t *m, void *arg);
} hal_wifi_event_cb_t;
```

4.3.2 Interface

Perform the following Wi-Fi HAL interface functions to support the device-cloud communication.

- **init**

  The `init` function is designed to initialize Wi-Fi hardware module, resources like memory heap, OS related materials, and peripherals.

  For QCA4004 porting, perform the following initializations:

  — SPI and SPI DMA are initialized for QCA4002 data transmission.

  — PIN interrupt is declared for QCA4002 notification.

  — A new task is created for data transmission. The task priority is higher than the application task so that network data is timely responded.

  — A semaphore is initialized to synchronize the initialization process. Do not start any network activity until Wi-Fi is initialized.

- **get_mac_addr**

  Wi-fi physical address is retrieved when the `get_mac_addr` is invoked.

  For QCA4002, use the `qcom_get_bssid` function. The MAC address is expressed in ASCII.

- **start**

  The `start` function carries the parameter of `mode`, so Wi-Fi module can start either in AP mode or station mode. Typically, LPC54102 acting as IoT device works in station mode and it can connect to router for internet access. IoT device IP can be set in two ways, statically assigned or assigned by router via DHCP service. Once IoT device owns its IP, the `got_ip` callback function is invoked to notify the system that the network connection is available.

- **get_ip_stat**

  The `get_ip_stat` function is used to get device IP, gateway, sub-mask, and MAC information.

- **start_scan**

  The `start_scan` function can get Access Point (AP) information by scanning channels. When AP information is retrieved, the `scan_compeleted` callback function is called. Allocate the memory in scan implementation for storing the AP information, and set free the memory only after the `scan_compeleted` callback function is returned.

4.4 Registration

The device calls the `hal_wifi_register_module` to register Wi-Fi HAL to system. The system only registers one Wi-Fi HAL module. When the system is initialized, the `hal_wifi_init` function is called to initialize the Wi-Fi HAL.

After the registration, any task can get WI-FI HAL instance and manipulate Wi-Fi through the Wi-Fi HAL interface.

4.5 Network socket

AliOS Things application uses the BSD socket interface for network transmission. In IoT world, some devices use TCP/IP stack (like LWIP) at host side with ethernet interface, but more often, IoT devices consist of a Wi-Fi module in the system. The Wi-Fi module has integrated TCP/IP stack which meant no TCP/IP stack is needed as host side.

Socket Adaptation Layer (SAL) is developed in AliOS Things so that network application gets minimized impaction from different TCP/IP implementation. SAL defines an interface for adapting Wi-Fi modules which is various in the IoT market. The SAL path in AliOS Thing is `/device/sal`.

In LPCXpresso54102 + GT202 approach, TCP/IP stack is implemented GT202 Wi-Fi module. Atheros WMI supplies a suite of TCP/UDP APIs for network data transmission. For socket integration, GT202 Wi-Fi module is in the `/device/sal/wifi/`. The corresponding integration code and makefile should be made.

4.5.1 Socket interface

In the `GT202.c` file, the `sal_op_t` typed instance is implemented and registered to SAL. In the `sal_op_t` structure, all lower level dependencies are capsulated. The `sal_op_t` structure and interfaces are defined as below.

```
typedef struct {
    char *version;
    /* Initialize WiFi socket module, socket handle is returned */
    int (*init)(void);
/* Request module to start a connection using
* 1. Socket Handle, 2, Connection type TCP/UDP
* 3. Addr IP or domain name
* 4. Local Port/Remote port
* 5. TCP keep active time */
    int (*start)(sal_conn_t *c);
/* Send data through socket in blocking mode */
/* fd is the handle created by init function */
    int (*send)(int fd, uint8_t *data, uint32_t len,
                char remote_ip[16], int32_t remote_port);
    /* Convert domain string to IP address, only IPV4 is supported */
int (*domain_to_ip)(char *domain, char ip[16]);
/* Close a socket handle locally */
    int (*close)(int fd, int32_t remote_port);
    /* De-initialize WiFi socket module if needed*/
    int (*deinit)(void);
    /* Register callback function for RX data arriving */
int (*register_netconn_data_input_cb)(netconn_data_input_cb_t cb);
} sal_op_t;
```

4.5.2 Socket registration

In `sal_device.c`, GT202 module is initialized in SAL initialization. A new Wi-Fi module is added when other Wi-Fi module is integrated.

```
int sal_device_init()
{
int ret = 0;
#ifdef DEV_SAL_GT202
    ret = GT202_sal_init();
#endif
    if (ret){
        LOGE(TAG, "device init fail ret is %d\n", ret);
    }
    return ret;
}
```

In `GT202_sal_init`, the `sal_op_t` instance is created and registered. In GT202 porting, the `GT202_sal_op` instance is created in `GT202_sal.c` and is registered with the `sal_module_register(&GT202_sal_op)` function.

5 AliOS Things application

There are 48 examples AliOS Things applications under “example” folder in AliOS Things release 1.3.2. To build one specific example, below command should be used.

```
$ aos make example_name@board_name
```

The machine executable binary is generated at output folder in the `bin` and `hex` format.

Tools like MCUXpresso, JLINK or Keil can program executable binary into the flash.

In this chapter, two examples, `nano` and `mqttapp` are elaborated.

5.1 Nano

Nano is the simplest example for AliOS Things. In the example, USART log is printed with task name information in period.

To build nano on LPCXpresso54102, it is necessary to apply **example name – nano** and **board name – LPCXpresso54102**.

```
$ aos make nano@LPCXpresso54102
```

After the binary is programed, print the following UART log after the chip resets.

```
nano app_delayed_action: 10 app
```

5.2 Mqttapp

Mqttapp demonstrates the communication between AliOS Things device and AliCloud via MQTT protocol. MQTT protocol is described as a machine-to-machine/**Internet of Things** connectivity protocol. For detailed information about MQTT, refer to http://mqtt.org/.

To use mqttapp in AliOS Thing, user should follow the instructions in AliOS Things Wiki. Wiki page “AliOS Things MQTT channel test guide” elaborates the details for the preparation work for mqttapp.

A new created device must consist product attributes “product key”, “device name” and “device secret”. These product attributes should be updated in mqtt-example.c, there are three corresponding MACROS in the code.

Users need to build mqttapp with the `aos make mqttapp@LPCXpresso54102` command and place the output binary in the `out/mqttapp@lpc54102xpresso54102` folder.

Mqttapp demonstrates a bi-direction communication between AliOS Things device and AliCloud.

1. AliOS Things device initializes the MQTT client and subscribes the `get` topic.
2. AliOS Things device publishes **dummy** temperature messages to AliCloud in a period of three seconds.
3. AliOS Things device unsubscribes MQTT topic **get** after 200 dummy temperature messages are sent.

During the period between subscribing and unsubscribing **get** topic, user can publish user-defined message to the topic from AliCloud console, and the device can get the message. Figure 5. on page 11 shows the overall server-client communication.

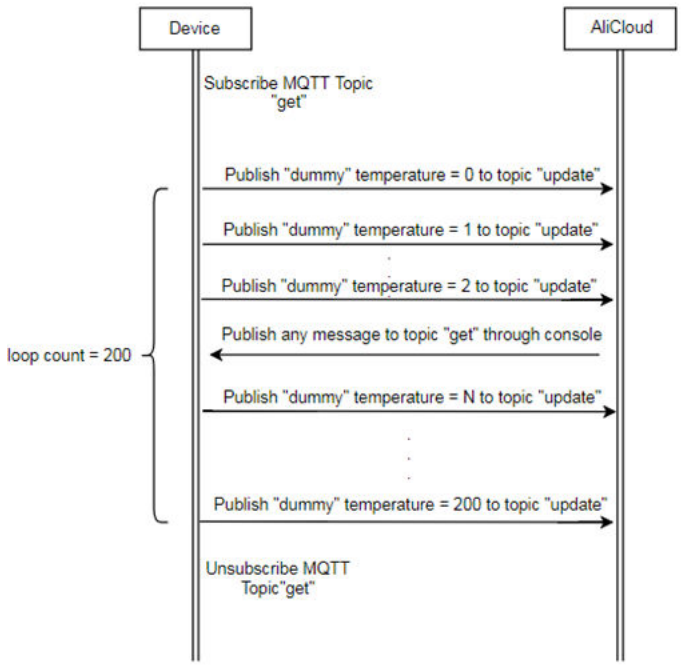

Device
AliCloud
Subscribe MQTT Topic "get"
Publish "dummy" temperature = 0 to topic "update"
Publish "dummy" temperature = 1 to topic "update"
Publish "dummy" temperature = 2 to topic "update"
Publish any message to topic "get" through console
loop count = 200
Publish "dummy" temperature = N to topic "update"
Publish "dummy" temperature = 200 to topic "update"
Unsubscribe MQTT Topic"get"

Figure 5. Messages between AliOS Things and AliCloud in MQTTAPP example

6 AliOS things certification

6.1 Introduction

According to AliOS Things Wiki page, AliOS IoT certification services consists of **AliOS Things kernel test**, **AliOS Things channel test**, **AliOS Things uMesh test**, **AliOS Things OTA test**, and **AliOS Things security test**, etc.

Alibaba can grant AliOS Things devices certification if the devices are verified in AliOS certification test organized by Alibaba. Device or platform vendor should follow the certification procedures in the latest **AliOS Things Certify Guideline** document for certification application.

Figure 6. on page 12 describes the procedure for AliOS Things certification.

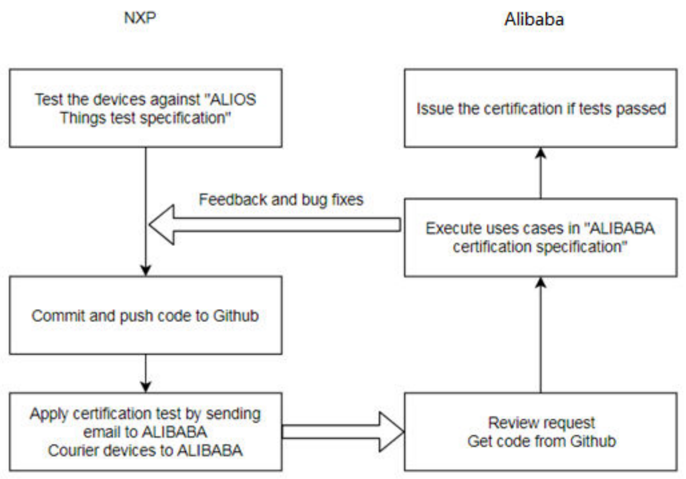

NXP
Alibaba
Test the devices against "ALIOS Things test specification"
Issue the certification if tests passed
Feedback and bug fixes
Execute uses cases in "ALIBABA certification specification"
Commit and push code to Github
Apply certification test by sending email to ALIBABA Courier devices to ALIBABA
Review request Get code from Github

Figure 6. Procedures for AliOS Thing certification

1. Test the device against **AliOS Things test specification**

   There are test specifications released from Alibaba for pre-certification unit tests. For example, **AliOS channel test specification** is for channel test and **YTS test** is for AliOS kernel test. More unit tests can be released from Alibaba with AliOS Things evolution. For more details about AliOS Things kenel test, refer to Kernel test on page 12.

2. Commit and push code to AliOS Things Github

   AliOS Things is open sources software in Github, and there are project maintainers in Alibaba. The project maintainer can help to have the new code reviewed and pushed to Github public repository.

3. Apply certification test by sending emails to Alibaba and courier devices to Alibaba

   A formal request should be sent to Alibaba for certification application. The applicant information is supplied and supporting documents like schematics, kernel test report, datasheet, device photos should be enclosed in the email. The supporting document checklist is explained in the **Alibaba certification application form** document.

4. Alibaba starts to execute certification tests and feedback applicant for bug reports or information request. Certification can be issued once all certification tests are passed.

6.2 Kernel test

Rhino Kernel test should be done by YTS application, when building YTS application, CLI module must be enabled and USART HAL must be implemented.

Before building the YTS test application, fill in device information in the `aos_test.c` file. The test configuration can be changed in the file considering memory constraint.

To build YTS application for LPCXpresso54102 in Linux, type in the following command line in the terminal.

```
$ aos make yts@LPCXpresso54102
```

The output firmware is `yts@LPCXpresso54102.hex` in the output folder.

Before running the test, connect the LPCXpresso54102 to PC for UART communication, which is typically USB VCOM and keep the USART terminal open. Type in the `yts_run` command in CLI shell terminal to start the YTS test. The test will run automatically, and test report will be printed in CLI shell terminal when test is completed.

7 Conclusion

This document introduces AliOS porting on LPC5410x microcontroller and can be regarded as supplementary information to AliOS Things Github Wiki.

AliOS Things porting guide introduces detailed porting guideline about kernel, HAL and Wi-Fi, and information about AliOS Things application and certification.

How To Reach Us

Home Page:

nxp.com

Web Support:

nxp.com/support

For more information, please visit: http://www.nxp.com
For sales office addresses, please send an email to: salesaddresses@nxp.com

Date of release: April 2019
Document identifier: AN12390