Overview of migrating Tuya's MCU SDK

Device Development > Access Mode MCU > Wi-Fi General Solution >

Software Reference Wi-Fi

Version: 20200428

Online Version

Contents

1 Introduction **1**

2 Precautions **2**

3 ile Structure **3**

4 Roadmap **4**

4.1 Compile the MCU basic program and migrate the SDK file. 4

4.2 Verify the macro definition in **protocol.h**. 5

4.3 Migrating the protocol.c File and Invoking Functions 11

4.4 Processing DP Data Report and Delivery Functions 12

4.5 Optimize the network configuration and indicator functions. 15

4.6 Optimize the product testing function. 18

1 Introduction

The **mcu_sdk** package contains the MCU code that is automatically generated based on product functions defined on the Tuya Smart platform. The communication and protocol resolution architecture is prepared and can be directly added to the original MCU project to quickly develop MCU programs.

2 Precautions

The SDK package has the following requirements on MCU hardware resources:

- Flash memory: 4 KB
- RAM: tens of bytes (depending on the DP data length), or 260 KB or higher if the OTA upgrade function is required
- The number of nested functions is 9.

Users without sufficient resources can implement protocol interworking without using the MCU SDK.

3 ile Structure

Execution File	Header File	Description
mcu_api.c	mcu_api.h	Contain Wi-Fi–related functions. Customers can invoke the functions on demand.
protocol.c	protocol.h	Protocol files that contain data processing functions. Users need to modify the two files based on project requirements.
system.c	system.h	Contain detailed implementation of the serial port communication protocol.
	wifi.h	Contains Wi-Fi–related macro definitions.

4 Roadmap

Step 1: Compile the MCU basic program and migrate the SDK file.

Step 2: Verify the macro definition in **protocol.h**.

Step 3: Migrate the **protocol.c** file and invoke functions.

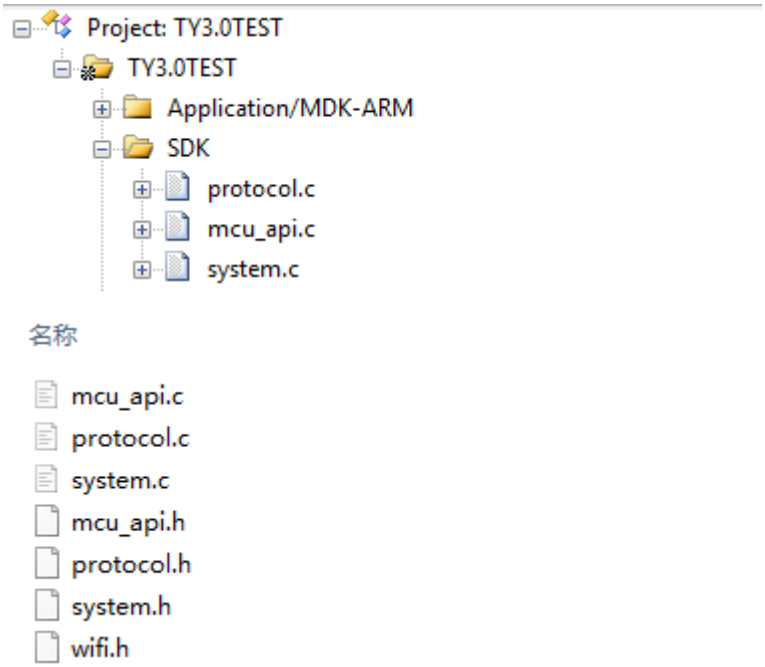
Step 4: Optimize the DP data report and delivery functions.

Step 5: Optimize the network configuration and indicator functions.

Step 6: Optimize the product testing function.

4.1 Compile the MCU basic program and migrate the SDK file.

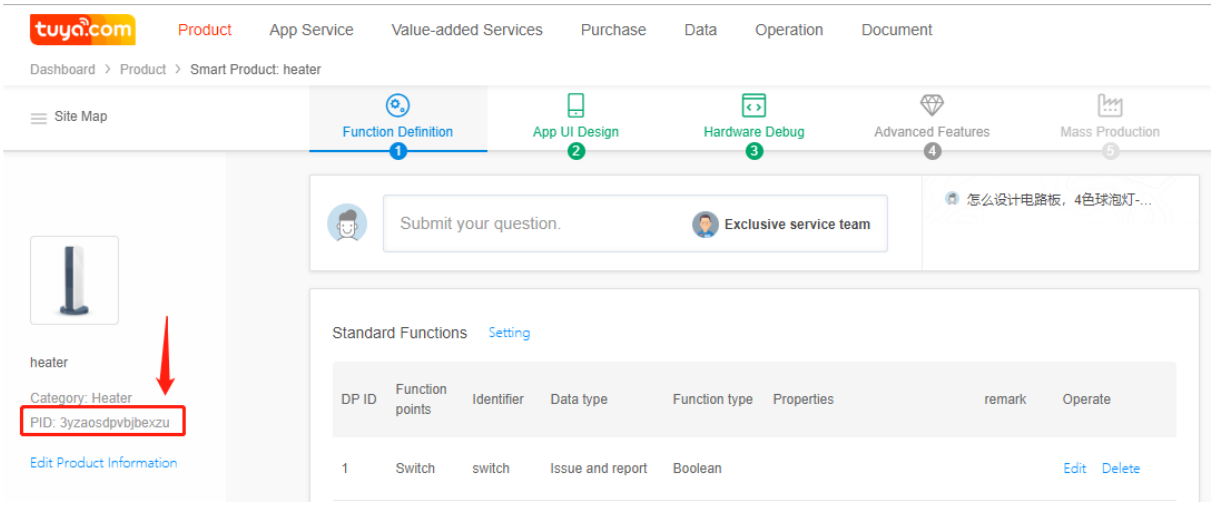
Add the .c and .h files in the mcu_sdk folder and corresponding header file reference path to the original project. Initialize MCU-related peripherals, including the serial port, external interrupt (button), and timer (indicator blinking).



4.2 Verify the macro definition in protocol.h.

4.2.1 Verify the product information

PRODUCT_KEY indicates the macro definition of the product ID (PID), which is the unique identifier of a product. Ensure that the PID is the same as that displayed on the Tuya Smart platform. If the PIDs are different, download the latest SDK package. MCU_VER indicates the software version, which is 1.0.0 by default. If the MCU requires OTA upgrade, you need to update the version number after the OTA upgrade. CONFIG_MODE indicates the network configuration mode, and the typical value is DEFAULT, indicating the default network configuration mode.



```
/*
    ***************************************************************************
                          1:Modify product information
    ***************************************************************************
    */
#define PRODUCT_KEY "pzbmv9a9pbkjtypm"
#define MCU_VER "1.0.0"          //User's software version for MCU
    firmware upgrade, MCU upgrade version needs to be modified



/*  Distribution mode selection, only three choices, Anti-touch mode is
     recommended  */
//#define CONFIG_MODE     CONFIG_MODE_DEFAULT             //Default
    working mode
//#define CONFIG_MODE     CONFIG_MODE_LOWPOWER            //Safe mode (
    low power working mode)
#define CONFIG_MODE     CONFIG_MODE_SPECIAL             // Anti-touch
    mode (special working mode)

/*  Set the opening time of network activity in low power working mode
    and special working mode,
     The define in the comment state will be processed for 3 minutes,Set
         data ranges that can be supported: 3~10 minutes  */
//#define CONFIG_MODE_DELAY_TIME    10          //Network activity time.
       unit: minutes

/*  Special configuration for smart mode and AP mode.
     If you do not use this define, you switch between smart mode and AP
         mode    */
//#define CONFIG_MODE_CHOOSE        0          //Open both AP and smart
    distribution networks without user switching, and the corresponding
    distribution network state 0x06
//#define CONFIG_MODE_CHOOSE        1          //Use only AP network
    configuration mode
```

4.2.2 Check whether the MCU firmware needs to be upgraded

If OTA upgrade of MCU firmware is required, enable the firmware update macro, which is disabled by default.

```
/*
    ******************************************************************************

                          2:Does the MCU require a firmware upgrade?
If you need to support MCU firmware upgrade, please open this macro
The MCU can call the mcu_firm_update_query() function in the mcu_api.c
    file to get the current MCU firmware update.

                        ********WARNING!!!**********
The current receive buffer is the size to turn off the firmware update
    function.
The firmware upgrade package is 256 bytes.
If you need to enable this function, the serial receive buffer will
    become larger.
******************************************************************************
    */
//#define         SUPPORT_MCU_FIRM_UPDATE                 //Enable MCU
    firmware upgrade function (off by default)
/*  Firmware package size selection  */
#ifdef SUPPORT_MCU_FIRM_UPDATE
#define PACKAGE_SIZE                    0       //The package size is
    256 bytes
//#define PACKAGE_SIZE                    1       //The package size is
     512 bytes
//#define PACKAGE_SIZE                    2       //The package size is
     1024 bytes
#endif
```

4.2.3 Define the transmitting and receiving buffers

Modify the buffer size based on the DP definition. The size of the serial port transmitting and receiving buffers must be larger than the maximum DP data length. The default size is 24 bytes. If MCU OTA upgrade is required, a 260-byte buffer is recommended. The receiving buffer size can be reduced if the RAM has insufficient space.

```
/*
    ************************************************************************

                            3:Define the send and receive buffer:
            If the current RAM of the MCU is not enough, it can be
                modified to 24
    ************************************************************************
    */
#ifndef SUPPORT_MCU_FIRM_UPDATE
#define WIFI_UART_RECV_BUF_LMT          16              //UART data
    receiving buffer size, can be reduced if the MCU has insufficient
    RAM
#define WIFI_DATA_PROCESS_LMT           24              //UART data
    processing buffer size, according to the user DP data size, must be
    greater than 24
#else
#define WIFI_UART_RECV_BUF_LMT          128             //UART data
    receiving buffer size, can be reduced if the MCU has insufficient
    RAM

/*  Select the appropriate UART data processing buffer size here
    (select the buffer size based on the size selected by the above MCU
        firmware upgrade package and whether to turn on the weather
       service)  */
#define WIFI_DATA_PROCESS_LMT           1000             //UART data
    processing buffer size. If the MCU firmware upgrade is required, the
     single-packet size is 256, the buffer must be greater than 260, or
    larger if the weather service is enabled
//#define WIFI_DATA_PROCESS_LMT           600             //UART data
    processing buffer size. If the MCU firmware upgrade is required, the
     single-packet size is 512, the buffer must be greater than 520, or
    larger if the weather service is enabled
//#define WIFI_DATA_PROCESS_LMT           1200            //UART data
    processing buffer size. If the MCU firmware upgrade is required, the
     single-packet size is 1024, the buffer must be greater than 1030,
    or larger if the weather service is enabled

#endif

#define WIFIR_UART_SEND_BUF_LMT         48              //According to
    the user's DP data size, it must be greater than 48
```

4.2.4 (Mandatory) Define the working mode of the Wi-Fi module

1) If the MCU controls network configuration triggering and indication, that is, the Wi-Fi reset button and Wi-Fi indicator are on the MCU side, enable cooperative processing by the Wi-Fi module and MCU (common mode) and ensure that #define is commented (the line of code starts with “//”).

```
/*
    ************************************************************************

                          4:Define how the module works
Module self-processing:
          The wifi indicator and wifi reset button are connected to the
              wifi module (turn on the WIFI_CONTROL_SELF_MODE macro)
          And correctly define WF_STATE_KEY and WF_RESET_KEY
MCU self-processing:
          The wifi indicator and wifi reset button are connected to the
              MCU (turn off the WIFI_CONTROL_SELF_MODE macro)
          The MCU calls the mcu_reset_wifi() function in the mcu_api.c
             file where it needs to handle the reset wifi, and can call
              the mcu_get_reset_wifi_flag() function to return the
             reset wifi result
          or call the mcu_set_wifi_mode(WIFI_CONFIG_E mode) function in
              the mcu_api.c file in the wifi mode, and call
             mcu_get_wifi_work_state() to return the setting wifi
             result.
    ************************************************************************
    */
//#define         WIFI_CONTROL_SELF_MODE                          //Wifi
    self-processing button and LED indicator; if the MCU external button
     / LED indicator please turn off the macro
```

2) If the Wi-Fi indicator and Wi-Fi reset button are on the Wi-Fi module, execute the following statement to enable processing by the Wi-Fi module: `#ifdef WIFI_CONTROL_SELF_MODE` Then, add information about the GPIO pins connected to the Wi-Fi indicator and Wi-Fi reset button, as shown in the following figure.

```
#define         WIFI_CONTROL_SELF_MODE                          //Wifi
    self-processing button and LED indicator; if the MCU external button
     / LED indicator please turn off the macro
#ifdef          WIFI_CONTROL_SELF_MODE                        //Module
    self-processing
  #define     WF_STATE_KEY            14                      //Wifi
      module status indication button, please set according to the
      actual GPIO pin
  #define     WF_RESERT_KEY           0                       //Wifi
      module reset button, please set according to the actual GPIO pin
#endif
```

4.2.5 Check whether the MCU needs time verification

If the time verification function is required, enable the RTC check macro.

```
/*
    ************************************************************************

                        5: Does the MCU need to support the time function
                           ?
Open this macro if needed and implement the code in mcu_write_rtctime
    in the Protocol.c file.
Mcu_write_rtctime has #err hint inside, please delete the #err after
    completing the function
Mcu can call the mcu_get_system_time() function to initiate the
    calibration function after the wifi module is properly networked.
************************************************************************
    */
#define          SUPPORT_MCU_RTC_CHECK                //Turn on time
    calibration
```

Write mcu_write_rtctime in the Protocol.c file to implement the code. After the Wi-Fi module successfully connects to the network, the MCU can invoke the `mcu_get_system_time()` function to initiate time verification.

4.2.6 Check whether the Wi-Fi product testing function is enabled

To ensure mass production efficiency and quality, we recommend that you enable the product testing macro. For details about implementation of the product testing function, see section 3.3.6 “Optimizing the Product Testing Function.”

```
/*
    ************************************************************************

                        6:Does the MCU need to support the wifi function
                           test?
Please enable this macro if necessary, and mcu calls mcu_start_wifitest
     in mcu_api.c file when wifi function test is required.
And view the test results in the protocol_c file wifi_test_result
    function.
There is a #err hint inside wifi_test_result. Please delete the #err
    after completing the function.
************************************************************************
    */
#define          WIFI_TEST_ENABLE                //Open WIFI production
    test function (scan designated route)
```

4.3 Migrating the protocol.c File and Invoking Functions

1. Use #include “wifi.h” in the files (for example, the main.c file) that require Wi-Fi–related files.
2. After MCU peripherals are initialized, invoke the wifi_protocol_init() function in the mcu_api.c file.
3. Add the single-byte sending function of the MCU serial port to the uart_transmit_output function in the protocol.c file and delete #error. The following figure shows an example.

```c
/**
 * @brief  Send data processing
 * @param[in] {value} Serial port receives byte data
 * @return Null
 */
void uart_transmit_output(unsigned char value)
{
//  #error "Please fill in the MCU serial port send function and delete the line"
  UART3_SendByte(value);
/*
  //Example:
  extern void Uart_PutChar(unsigned char value);
  Uart_PutChar(value);                                 //Serial port send function
*/
}
```

4. Invoke the uart_receive_input function in the mcu_api.c file in the serial port receiving interrupt service function, and use the received characters as parameter input. The following figure shows an example.

```c
void USART3_IRQHandler(void)
{
  uint8_t ch;

  if(USART_GetITStatus(USART3, USART_IT_RXNE) != RESET)
  {
    ch = USART_ReceiveData(USART3);

#ifndef DEBUG_IN_USART1

    uart_receive_input(ch);
#endif
  }
}
```

5. Invoke the wifi_uart_service() function in the mcu_api.c file after the MCU enters the while cycle. The following shows an example of code structure in main.c.

```
include "wifi.h"
...
void main(void)
{
wifi_protocol_init();
...
while(1)
{
wifi_uart_service();
...
}
}
```

Note: The MCU must directly invoke the wifi_uart_service() function in the mcu_api.c file in while. After the program is successfully initialized, it is recommended that the serial port interrupt not be disabled. If the serial port interrupt must be disabled, ensure that the interrupt is disabled for only a short time to prevent serial port data loss. Do not invoke the report function in the interrupt.

4.4 Processing DP Data Report and Delivery Functions

4.4.1 Reporting data of all DPs

After the Wi-Fi module restarts or the network is reconfigured, the Wi-Fi module proactively delivers a status query command. The MCU needs to report the status of the device's DPs to the Wi-Fi module for synchronization. (1) Open protocol.c and locate the all_data_update(void) function. (2) Enter initial values of all DPs to be reported into corresponding report functions. The values will be displayed on the App control panel. Note: Do not invoke the all_data_update() function manually. This function is automatically invoked at a specific time.

```c
/**
 * @brief  All dp point information of the system is uploaded to realize APP and muc data synchronization
 * @param  Null
 * @return Null
 * @note   This function SDK needs to be called internally;
 *         The MCU must implement the data upload function in the function;
 *         including only reporting and reportable hair style data.
 */
void all_data_update(void)
{
//  #error "Please process the reportable data and report only the data. After the processing is completed, delete the line"

  //This code is automatically generated by the platform.
  //Please modify each reportable and reportable function according to the actual data.
  mcu_dp_bool_update(DPID_SWITCH,0); //Boolean data reporting;
  mcu_dp_value_update(DPID_TEMP_SET,0); //Value data reporting;
  mcu_dp_value_update(DPID_TEMP_CURRENT,0); //Value data reporting;
  mcu_dp_bool_update(DPID_ECO,0); //Boolean data reporting;
  mcu_dp_bool_update(DPID_SHAKE,0); //Boolean data reporting;
  mcu_dp_bool_update(DPID_ANION,0); //Boolean data reporting;

}
```

4.4.2 Reporting data of a single DP

When the status of a DP is changed, the MCU proactively reports the new DP status to the Wi-Fi module, and the DP status displayed on the App will be updated accordingly. The report data format is mcu_dp_xxxx_updata(DPID_X,n). DPID_X indicates the DP whose status has changed. Functions in all_data_update() can be independently invoked. Example: mcu_dp_bool_update(DPID_SWITCH,1); //Boolean data reporting mcu_dp_value_update(DPID_TEMPER_SET,25); //Value data reporting mcu_dp_string_update(DPID_DAY,“1234”,4); //String data reporting

4.4.3 DP data delivery

Each deliverable DP has an independent data delivery processing function in the protocol.c file. The function format is dp_download_xxx_handle(), and xxx indicates a deliverable DP. After the function parses a DP, the MCU performs logical control

in the corresponding position. The following shows an example of receiving switch data.

```c
/*
    ******************************************************************************
Function name : dp_download_switch_handle
Function description : on DPID_SWITCH processing function
Input parameter : value:Source data
        : length:Data length
Return parameter : Successful return:SUCCESS/Failed to return:ERROR
Instructions for use : 可下发可上报type,need to report the result to App after data is dealt with
    ******************************************************************************
    */
static unsigned char dp_download_switch_handle(const unsigned char value[], unsigned short length)
{
  //Example: The current DP type isBOOL
  unsigned char ret;
  //0:关/1:开
  unsigned char switch;

  switch = mcu_get_dp_download_bool(value,length);
  if(switch == 0)
  {
    //开关关
  }
  else
  {
    //开关开
  }

  //处理完DP数据后应有反馈
  ret = mcu_dp_bool_update(DPID_SWITCH,switch);
  if(ret == SUCCESS)
    return SUCCESS;
  else
    return ERROR;
}
```

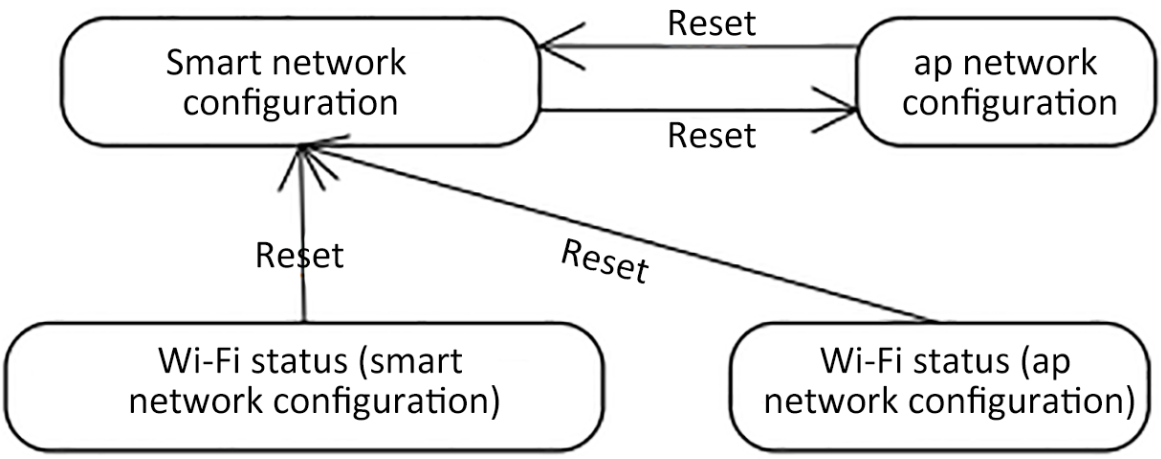
The MCU uses MCU_ON_switch1() and MCU_OFF_switch1() to turn on and off a switch, respectively. When the device status is changed under non-App control, the MCU invokes mcu_dp_bool_update(DPID_SWITCH_1,switch_1) to upload the real status of the switch. Typically, the receiving processing function automatically invokes the function.

4.5 Optimize the network configuration and indicator functions.

Skip this section if processing by the Wi-Fi module is used. When protocol migration is successful, the network configuration command and indicator function need to be optimized for network configuration. In mode of cooperative processing by the Wi-Fi module and MCU, the MCU can select the network configuration triggering and indication modes based on actual requirements. Typically, network configuration is triggered by the Wi-Fi reset button and indicated by quick or slow blinking of the Wi-Fi indicator. We recommend that you enable both network configuration modes for your product. Smart network configuration mode: The operation is simple and convenient, and the Wi-Fi indicator blinks quickly. AP network configuration mode: Network configuration is reliable, and the Wi-Fi indicator blinks slowly.

4.5.1 Network configuration command

The network configuration command can be implemented by the mcu_reset_wifi() and mcu_set_wifi_mode() functions. Typically, these two functions are invoked in the button processing function after the button is pressed for network configuration. After mcu_reset_wifi() is invoked, the Wi-Fi module is reset and the previous network configuration information is cleared. The function invoking also triggers a switchover between the AP and smart network configuration modes.



After mcu_set_wifi_mode() with parameter SMART_CONFIG or AP_CONFIG is invoked, the network configuration information is cleared, and smart or AP network configuration mode is used. This function has the same function as the mcu_reset_wifi() function. You can select one as needed.

4.5.2 Network configuration indication

Typically, the mcu_get_wifi_work_state() function is invoked at while(1) to return the Wi-Fi status. Then, you write the indicator blinking mode in based on the Wi-Fi status.

Device Network Connection Status	Description	Status Value	LED Indicator Status
State 1	Smart network configuration	0x00	The indicator blinks at 250 ms intervals.
State 2	AP network configuration	0x01	The indicator blinks at 1500 ms intervals.
State 3	The Wi-Fi is configured. However, the device fails to connect to the router.	0x02	The indicator is off.
State 4	The Wi-Fi is configured, and the device successfully connects to the router.	0x03	The indicator is steady on.

Device Network Connection Status	Description	Status Value	LED Indicator Status
State 5	The device connects to the router and cloud.	0x04	The indicator is steady on.
State 6	The Wi-Fi device is in low power consumption mode.	0x05	The indicator is off.

Invoke the mcu_get_wifi_work_state() function to obtain the Wi-Fi status. The function architecture is as follows:

```c
void main(void)
{
    ...

    while(1)
    {
        switch(mcu_get_wifi_work_state())
        {
            case SMART_CONFIG_STATE:
            //smart config configuration state: LED flash quickly; the user needs to complete the configuration
            break;
            case AP_STATE:
            //AP configuration state: LED flash slowly
            break;
            case WIFI_NOT_CONNECTED:
            //Wi-Fi configuration is finished; the router is being connected to; LED keeps long dark
            break;
            case WIFI_CONNECTED:
            //The router is successfully connected to; LED keeps long bright
            break;
            default:break;
        }
        ...
    }
}
```

4.6 Optimize the product testing function.

Notes: product test is used only in production, and it is mainly used to test the Wi-Fi function of modules and the communication capability of module and control panel.

The test requires no network connection, and the product test process is triggered by pressing button. The test process takes about 5s.

Note:

1. There shall be as less routers as possible to speed up test of products. 2、Please wait for 2 seconds after the system is connected to the power supply so that the module is started. 3、Product test is not required in daily use.

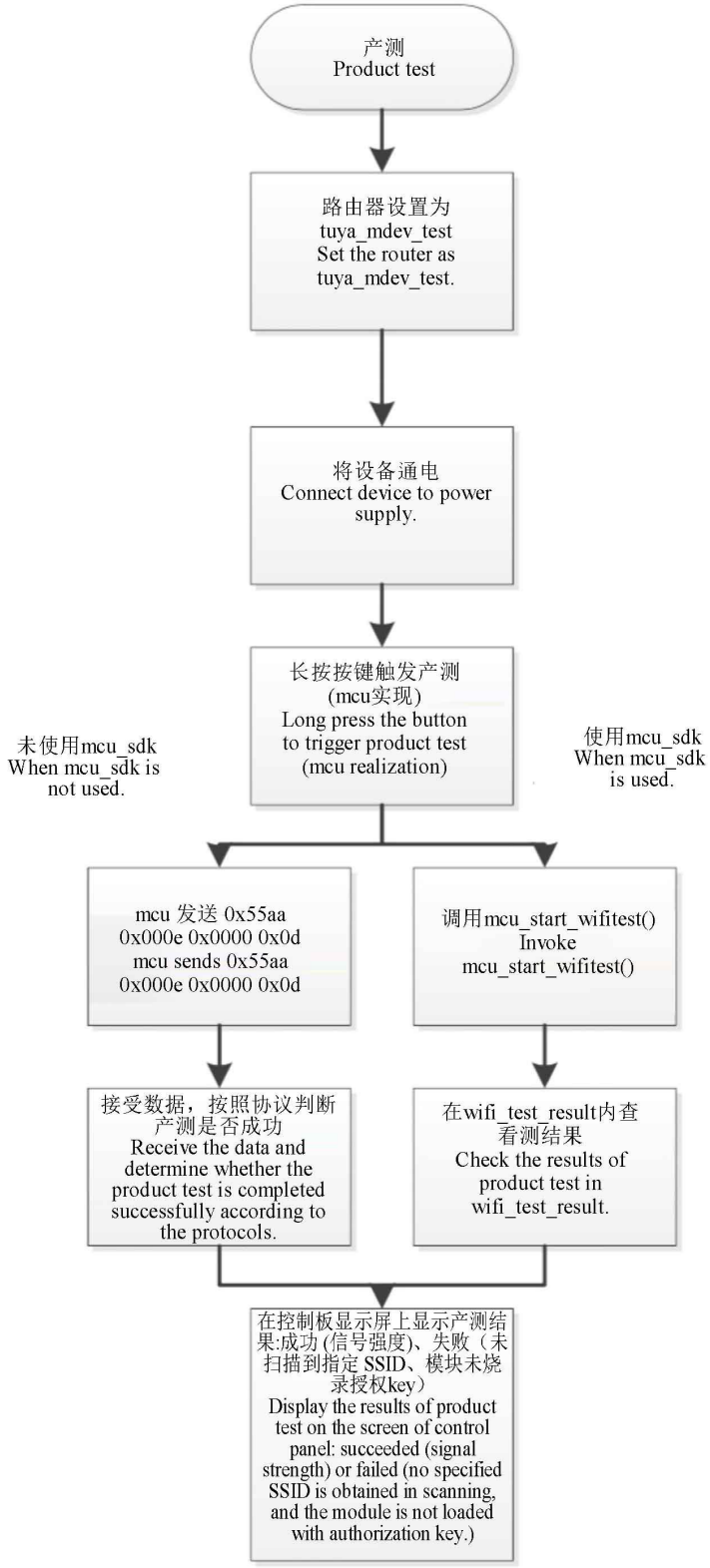


Figure 1: cmd-markdown-logo

4.6.1 Preparation

One 2.4G wireless router and power supply, and network connection is not required. The SSID of router will be set to **tuya_mdev_test**, and it will be placed in the workshop of the production line.

4.6.2 Connect the device to be tested to the power supply.

4.6.3 Trigger the product test.

mcu will press button and hold it to trigger product test (the trigger mode is realized by mcu, it is recommended to use key combination that is uncommon or press some keys for a long period to trigger product test function).

Then the following interfaces will be invoked to trigger product test:

- When mcu_sdk is used, mcu invokes `mcu_start_Wi-Fi test()`
  - When mcu_sdk is not used, mcu sends`0x55 0xaa 0x00 0x0e 0x0000 0x0d`

4.6.4 Check results of test

The following methods can be used to check test results based on whether the mcu_sdk provided by Tuya is used.

mcu_sdk is used. Check results of tests in the `wifi_test_result()`function of the protocol.c file.

```c
void wifi_test_result(unsigned char result,unsigned char rssi)
{

#error "Please add your own codes for successful or failed Wi-Fi function test and delete this line when codes are added."

if(result == 0)
{
//Test failed
if(rssi == 0x00)
{
//The tuya_mdev_test router is not found in the scanning, please check it
}
else if(rssi == 0x01)
{
//The module is not authorized
}
}
else
{
//Test succeeded
//rssi represents signal strength (0-100, 0 represents the weakest signal, and 100 represents the strongest signal)
}
}
```

mcu_sdk is not used Check results of tests according to received data.

WiFi功能 产测（注：扫描 tuya_mde v_test 的 指定SSID）	MCU上报	0x55aa 0x00	0x0e	0x0000		0x0d
	模块发送	0x55aa 0x00	0x0e	0x0002	数据长度为2字节： Data[0]:0x00失败，0x01成功；当Data[0]为0x01，即成功时， Data[1]表示信号强度（0-100，0信号最差，100信号最强）当Data[0]为0x00，即失败时， Data[1]为0x00表示未扫描到指定的ssid， Data[1]为0x01 表示模块未烧录授权key	校验和

Figure 2: cmd-markdown-logo

4.6.5 Display test results

The tests have three kinds of test results, and test results will be displayed on the display screen of the control panel.

1. Signal strength
2. The tuya_mdev_test router is not found in the scanning, please check it
3. he module is not authorized