Thermometer demo reference design

Version: 1.0

Release date: 21 July 2016

Document Revision History

Revision	Date	Description
1.0	21 July 2016	Porting SON1421 to SDK3.3.1.

Table of contents

1. Introduction

1.1. Objective

The purpose of this document is to describe the design of WIFI thermometer.

In this document, you can create your own digital thermometer by using MT7687 and a temperature sensor, and upload the temperature information to the PC or cloud via socket connection. This design is also can be used in hospital or in smart home, and other scene that need a temperature information.

In this document, HW design and SW coding should be introduced for reader. And reader could build the thermometer in a short time

2. Hardware

2.1. Component

MCU: MediaTek MT7687

WIFI: embedded in MediaTek MT7687 (WIFI SOC)

Single-line digital temperature sensor: SOON ELECTRONICS SON1421

2.2. Interface

Table 1. Interface

76x7 Pin Function	Peripheral Pin Name	Pin description	I/O	level shift
GPIO1	SON1421 DIO	Gpio for single-line communication	I/O	no need

2.3. SON1421 Power Scheme

Table 2. SON1421 Power Scheme

Power name	Power voltage	Power supply	Power usage for SON1421
VDD	3.3V	76X7 3V3	Sensor power supply

2.4. Schematic Diagram

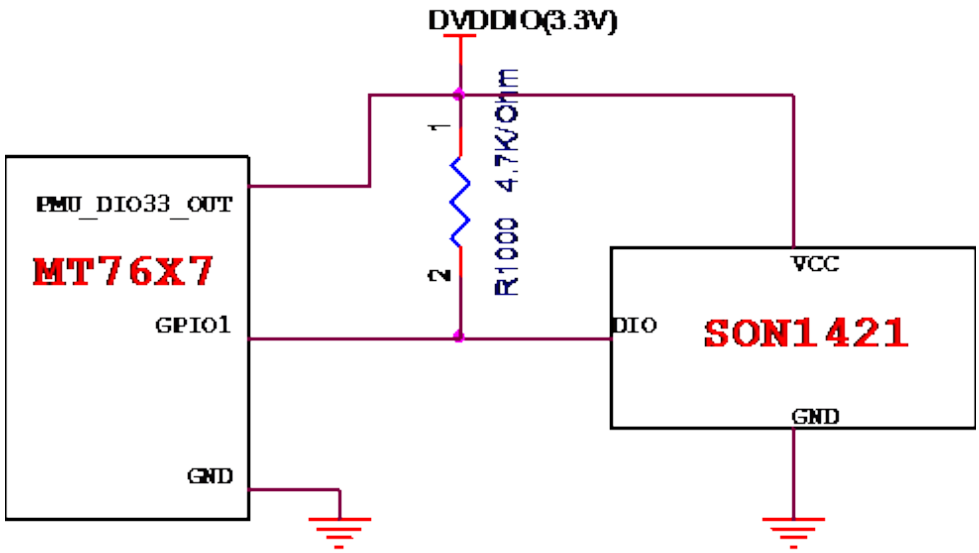


Figure 1. Schematic diagram

3. Software

3.1. SON1421 Driver

SON1421 is a single-line sensor component. So only 1 GPIO is needed in software, but timing is the most important parameter for single-line application.

So ASM delay is use for GPIO delay:

```
static void delay_us(unsigned int us)
{
   us*=10;
#ifdef __GNUC__
   us = 3 * us - 3;
   __asm volatile(
"      mov     r0, %0     \n\t"
"      ldr     r1,=0      \n\t"
"loop:                    \n\t"
"      cmp     r1, r0     \n\t"
"      bcc     add_r1     \n\t"
"      b       done       \n\t"
"add_r1:                  \n\t"
"      adds    r1, #1     \n\t"
"      b       loop       \n\t"
"done:                    \n\t":: "r" (us) : "r0" , "r1");

#elif defined(__arm__)
   uint32_t i = 0;

   if ( us < 5 ) {
      return;
   }
   us = 3 * us - 10;
   do {
      i = i + 3 ;
   }while(i<us);
   #endif
}
```

By the way, one task is crated for SON1421. And the priority is higher than WIFI and TCP/IP stack, just for protecting the driver timing of the SON1421 from interrupt by other task.

```
#define THERMOMETER_STACK_SIZE   (1000/sizeof(portSTACK_TYPE))
#define THERMOMETER_TASK_PRIO    2

   if (pdPASS != xTaskCreate(get_thermometer,
                        "thermometer_demo",
```

```
                    THERMOMETER_STACK_SIZE,
                    NULL,
                    THERMOMETER_TASK_PRIO,
                    NULL)) {
    LOG_E(common, "create user task fail");
    return -1;
  }

void get_thermometer(void *args)
{
        while(1)
        {
                thermometer_value = get_temperature();
                vTaskDelay(2000);
        }
}
```

For SON1421 driver, the HAL GPIO layer is merged to the demo code:

```
#define SingleLinePullLow  temperature_gpio_write(0)
#define SingleLinePullHigh  temperature_gpio_write(1)
#define SingleLineConfigureInputState
temperature_gpio_set_mode(TEMPERATURE_GPIO,HAL_GPIO_DIRECTION_INPUT)
#define SingleLineConfigureOutputState
temperature_gpio_set_mode(TEMPERATURE_GPIO,HAL_GPIO_DIRECTION_OUTPUT)
#define SingleLineGetInput     temperature_gpio_read(TEMPERATURE_GPIO)

void temperature_gpio_set_mode(hal_gpio_pin_t gpio_num,uint8_t mode)
{
  hal_gpio_init(gpio_num);
  /* Set pin as GPIO mode.*/
  //hal_pinmux_set_function(gpio_num, temperature_gpio_mode);
        if(mode == HAL_GPIO_DIRECTION_OUTPUT)
        {
          /* Set GPIO as output.*/
          hal_gpio_set_direction(gpio_num, HAL_GPIO_DIRECTION_OUTPUT);
        }
        else
        {
          /* Set GPIO as output.*/
          hal_gpio_set_direction(gpio_num, HAL_GPIO_DIRECTION_INPUT);
                /* Configure the pull state to pull-up. */
                //hal_gpio_pull_up(gpio_num);
        }

  hal_gpio_deinit(gpio_num);
}
```

```
bool temperature_gpio_write(hal_gpio_data_t value)
{
   /* Set GPIO as output.*/
   hal_gpio_set_direction(TEMPERATURE_GPIO, HAL_GPIO_DIRECTION_OUTPUT);
   hal_gpio_set_output(TEMPERATURE_GPIO, value);
#if GPIO_FOR_DEBUG
   hal_gpio_set_output(HAL_GPIO_26, 1);
   hal_gpio_set_output(HAL_GPIO_26, 0);
#endif
            return 1;
}

hal_gpio_data_t temperature_gpio_read(hal_gpio_pin_t gpio_num)
{
            hal_gpio_data_t data_pull_up;
            //hal_gpio_data_t data_pull_down;

            /* Set GPIO as input.*/
            hal_gpio_set_direction(TEMPERATURE_GPIO, HAL_GPIO_DIRECTION_INPUT);
            /* Configure the pull state to pull-up. */
            //hal_gpio_pull_up(TEMPERATURE_GPIO);
            /* Read the input data of the pin for further validation.*/
            hal_gpio_get_input(TEMPERATURE_GPIO, &data_pull_up);

#if GPIO_FOR_DEBUG
   hal_gpio_set_output(HAL_GPIO_25, 1);
   hal_gpio_set_output(HAL_GPIO_25, 0);
#endif
            return data_pull_up;
}
```

And finally, the thermometer can be get in get_temperature() API:

```
float get_temperature()
{
  unsigned char buf[20] = {0};
  float T = 0;
  StartTemperature_RW1820();       //start temperature conversion
  if( ReadScratchpad(buf) )
  {
    T=(buf[1]*256+buf[0])/16.0;
  }
  else
  {
             log_hal_info("ReadScratchpad fail");
```

```
  }
 return T;
}
```

3.2. TCP client in device

There is a sample project for TCP/UDP in MT7687 SDK, project root as follow:

\project\mt7687_hdk\apps\lwip_socket

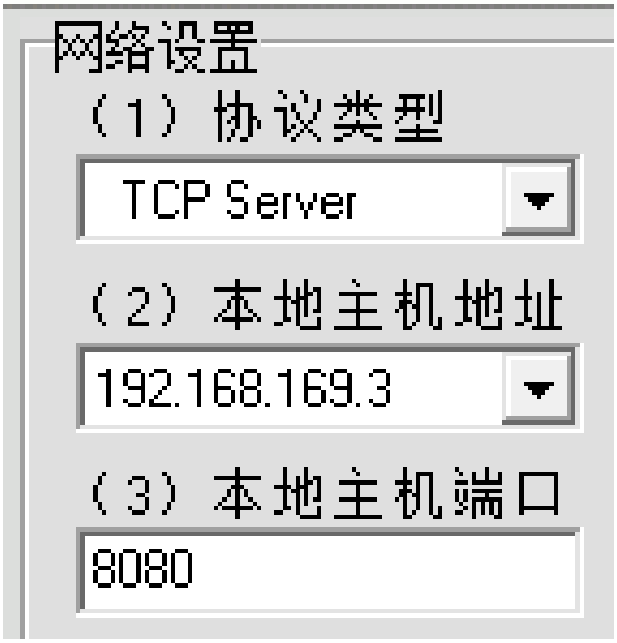
This sample can be used to complete the socket connection, modification as follow:

Cause that MT7687 work as client, so server port should be set to the same as PS server side.

Should be modified in main.c

```
#define SOCK_TCP_SRV_PORT      8080
```

Should be the same as PC server setting



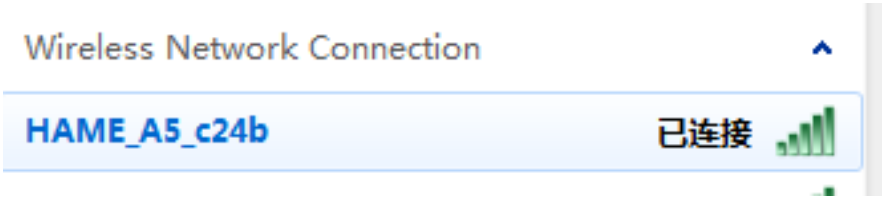
IP address of client is negligible, so DHCP is used for getting the dynamic IP address.

The SSID and password should be modified to the same as which one the PC server is connected:

Should be modified in Sta_network.c

```
#define USE_DHCP              1
#define WIFI_SSID ("HAME_A5_c24b")
#define WIFI_PASSWORD ("7c******b8")
```

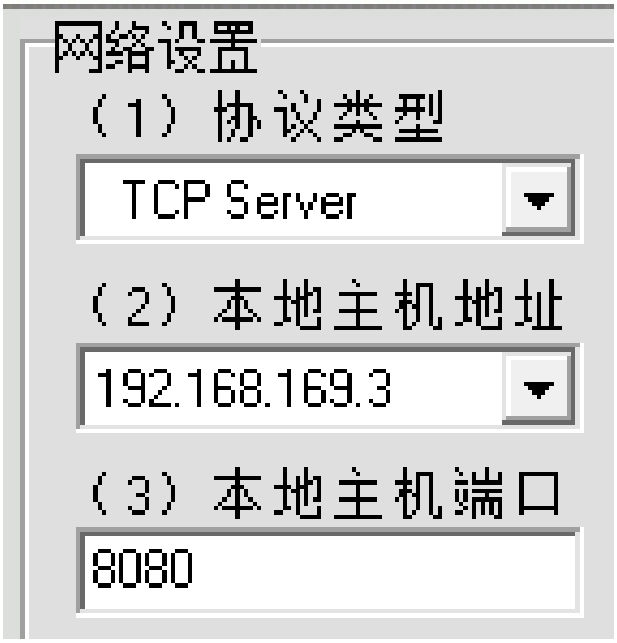
WIFI connection setting shoulb be the same as the PC connected



Set the TCP server IP as what you connect to. The IP address is little-endian.

```
static void tcp_client_test(void)
{
   int s;
   int ret;
   struct sockaddr_in addr;
   int count = 0;
  ……
   os_memset(&addr, 0, sizeof(addr));
   addr.sin_len = sizeof(addr);
   addr.sin_family = AF_INET;
   addr.sin_port = lwip_htons(SOCK_TCP_SRV_PORT);
   inet_addr_from_ipaddr(&addr.sin_addr, netif_ip4_addr(sta_if));
    addr.sin_addr.s_addr= 0x03a9a8c0;  //set to the same as PC server
```

IP address setting in code should be the same as PC server IP address, and IP address in code is little-endian



For thermometer data part, you can get the data periodically:

```
   while (1){//count < TRX_PACKET_COUNT) {
     /* Write something */
                    temperature = thermometer_value;//get_temperature();
```

```
                tem_decade = (char)temperature;
                tem_unit = (char)(temperature*100 - tem_decade*100);
                sprintf(send_data,"temperature = %d.%d\n",tem_decade,tem_unit);
    ret = lwip_write(s, send_data, sizeof(send_data));
    LOG_I(common, "TCP client write:ret = %d", ret);
    vTaskDelay(2000);
    }
```

3.3. PC server in PC side

You can use net assist to test the socket connection. And you can set the net assist refer to 4.Test & Debug

4. Test & debug

4.1. Software merge

Copy the demo project to the following path:

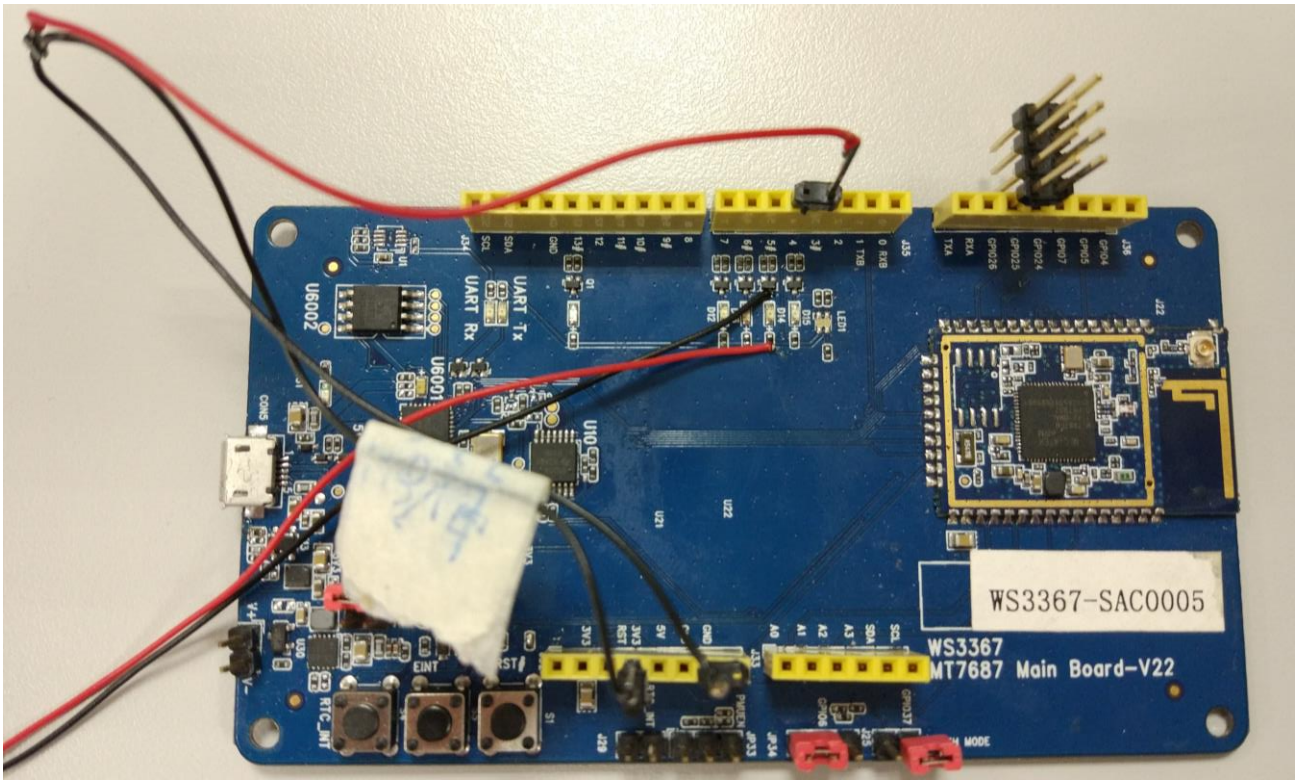
\project\mt7687_hdk

And then run the build command:

./build.sh mt7687_hdk thermometer_demo

4.2. Device

Connect the device like the following picture:



Download the software BIN into MT7687, for how to download, refer to “LinkIt for RTOS get started”

Then power on the MT7687 HDK

NOTE: Before power on, TCP server in PC side must be setup firstly

4.3. PC setting

Run TCP net assist and make it work in TCP sever mode. Net assist can get the IP by itself.

And set the socket port as setting in MT7687. Then click “connect” button.

Then you can get the temperature in PC side

5. Limitation

5.1. Driver

SON1421 can support multi-sensor read. But in this document, only one sensor is supported. If multi-sensor is needed, reader can apply the feature by merging the driver to MT7687

5.2. Cloud

There is only socket sample code in this design. If cloud service is needed, reader can link the cloud service by socket or http service.