

## 第一章 I2C-读取温湿度

### 1. 学习目的及目标

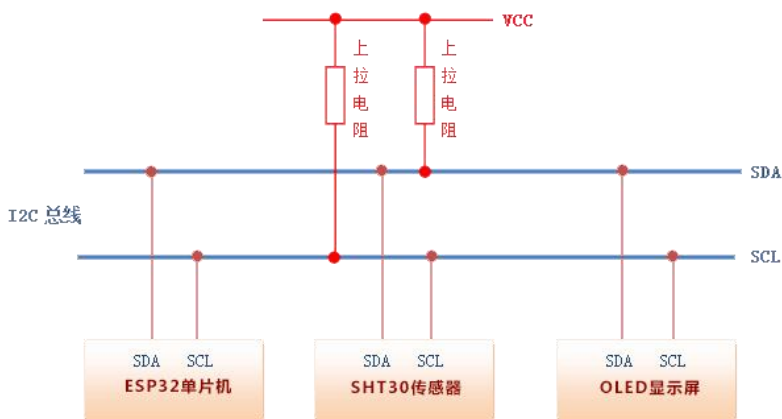
- I2C 通信的原理
- 学习 ESP32 的 I2C 功能的配置
- 掌握 I2C 读取 SHT30 的温湿度程序

### 2. I2C 通讯协议简介

I2C 通讯协议(Inter-Integrated Circuit)是由 Philips 公司开发的, 由于它引脚少, 硬件实现简单, 可扩展性强, 不需要 USART、CAN 等通讯协议的外部收发设备, 现在被广泛地使用在系统内多个集成电路(IC)间的通讯。下面我们分别对 I2C 协议的物理层及协议层进行讲解。

- 物理层

I2C 通讯设备之间的常用连接方式通讯结构如下。



I2C 总线物理拓扑图

它的物理层有以下几个主要特点:

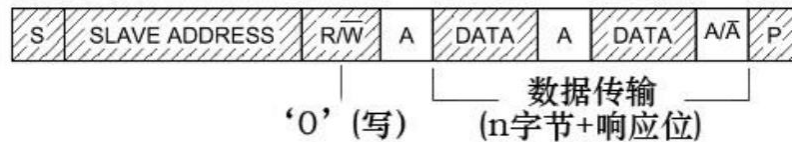
- 支持多设备的总线。“总线”指多个设备共用的信号线。在一个 I2C 通讯总线中, 可连接多个 I2C 通讯设备, 支持多个通讯主机及多个通讯从机。
- 一个 I2C 总线只使用两条总线线路, 一条双向串行数据线(SDA), 一条串行时钟线(SCL)。数据线即用来表示数据, 时钟线用于数据收发同步。
- 每个连接到总线的设备都有一个唯一的地址, 主机利用这个地址在不同设备之间的访问。
- 总线通过上拉电阻接到电源。当 I2C 设备空闲时, 会输出高阻态, 而当所有设备都空闲, 都输出高阻态时, 由上拉电阻把总线拉成高电平。多个主机同时使用总线时, 为了防止数据冲突, 会利用仲裁方式决定由哪个设备占用总线。
- 常用的速率: 标准模式传输速率为 100kbit/s, 快速模式为 400kbit/s。
- 最近 I3C 出来了, 性能提升大大的。

#### 2.1. 协议层

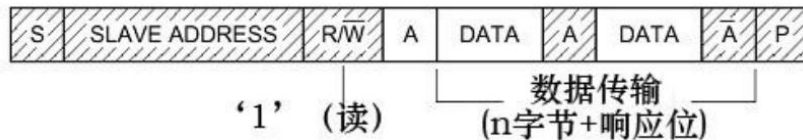
I2C 的协议定义了通讯的起始和停止信号、数据有效性、响应、仲裁、时钟同步和地址广播等环节。

I2C 通讯过程的基本结构如下:

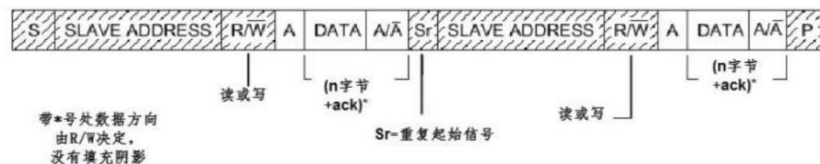
### ➤ I2C 写格式



### ➤ I2C 读格式



### ➤ I2C 读写格式



图例: 数据由主机传输至从机    S: 传输开始信号

SLAVE\_ADDRESS: 从机地址

数据由从机传输至主机    R/W: 传输方向选择位, 1 为读, 0 为写

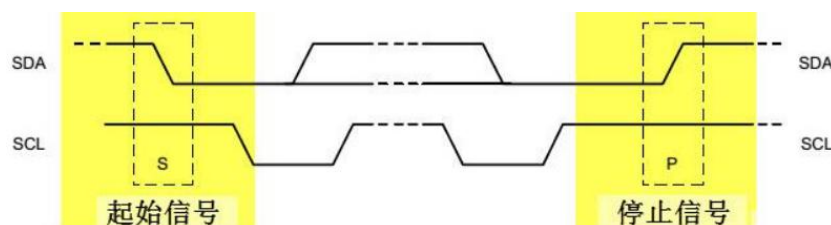
A/Ā: 应答(ACK)或非应答(NACK)信号

P: 停止传输信号

I2C 的几个细节如下:

### ➤ I2C 的起始和停止信号

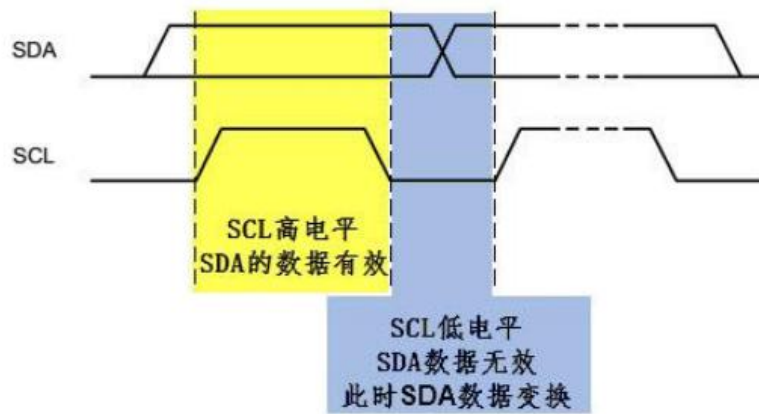
前文中提到的起始(S)和停止(P)信号是两种特殊的状态, 见图 23-5。当 SCL 线是高电平时 SDA 线从高电平向低电平切换, 这个情况表示通讯的起始。当 SCL 是高电平时 SDA 线由低电平向高电平切换, 表示通讯的停止。起始和停止信号一般由主机产生。



### ➤ 数据有效性

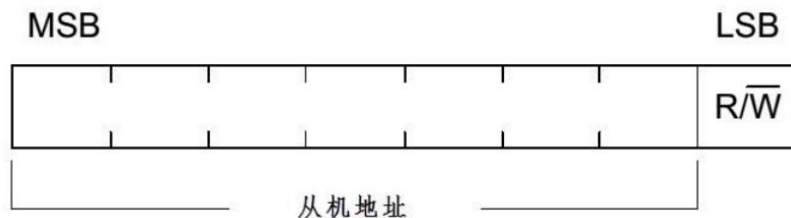
I2C 使用 SDA 信号线来传输数据, 使用 SCL 信号线进行数据同步。SDA 数据线在 SCL 的每个时钟周期传输一位数据。传输时, SCL 为高电平的时候 SDA 表示的数据有效, 即此时的 SDA 为高电平时表示数据“1”, 为低电平时表示数据“0”。当 SCL 为低电平时, SDA 的数据无效, 一般在这个时候 SDA 进行电平切换, 为下一次表示数据做好准备。

每次数据传输都以字节为单位, 每次传输的字节数不受限制。



### ➤ 地址及数据方向

I2C 总线上的每个设备都有自己的独立地址，主机发起通讯时，通过 SDA 信号线发送设备地址(SLAVE\_ADDRESS)来查找从机。I2C 协议规定设备地址可以是 7 位或 10 位，实际中 7 位的地址应用比较广泛。紧跟设备地址的一个数据位用来表示数据传输方向，它是数据方向位(R/W)，第 8 位或第 11 位。数据方向位为“1”时表示主机由从机读数据，该位为“0”时表示主机向从机写数据。

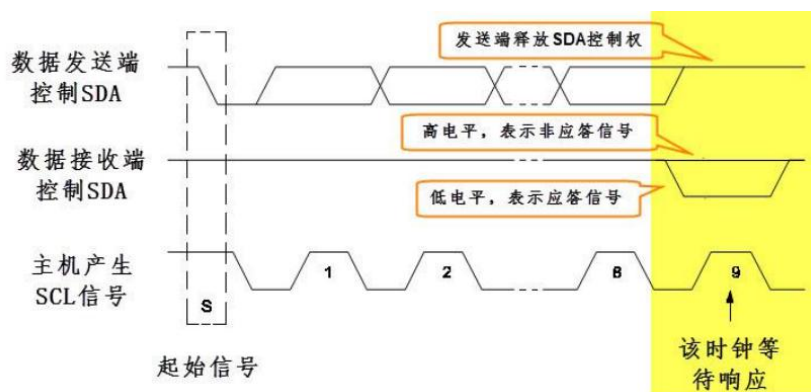


读数据方向时，主机会释放对 SDA 信号线的控制，由从机控制 SDA 信号线，主机接收信号，写数据方向时，SDA 由主机控制，从机接收信号。

### ➤ 响应

I2C 的数据和地址传输都带响应。响应包括“应答(ACK)”和“非应答(NACK)”两种信号。作为数据接收端时，当设备(无论主从机)接收到 I2C 传输的一个字节数据或地址后，若希望对方继续发送数据，则需要向对方发送“应答(ACK)”信号，发送方会继续发送下一个数据；若接收端希望结束数据传输，则向对方发送“非应答(NACK)”信号，发送方接收到该信号后会产生一个停止信号，结束信号传输。

传输时主机产生时钟，在第 9 个时钟时，数据发送端会释放 SDA 的控制权，由数据接收端控制 SDA，若 SDA 为高电平，表示非应答信号(NACK)，低电平表示应答信号(ACK)。



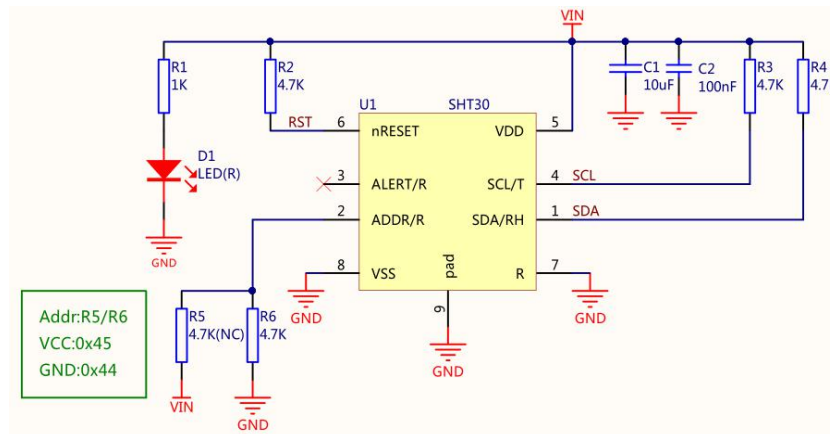
抄了这么多理论其实没什么用，因为 ESP32 带硬件 I2C，只要调用相关 API 即可，用起来非常简单。

### 3. SHT30 温湿度传感器参数介绍

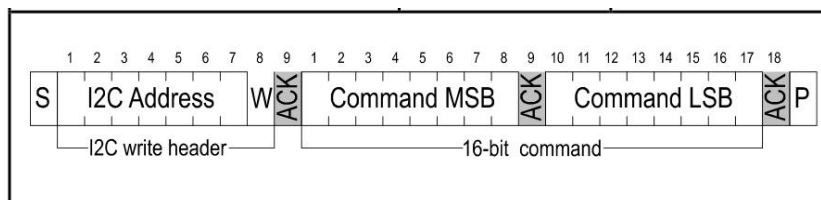
➤ SHT30 温湿度测试范围

| 温度               | 湿度               |
|------------------|------------------|
| -40~125℃ 误差±0.3℃ | 0~100% 误差征服 3%RH |

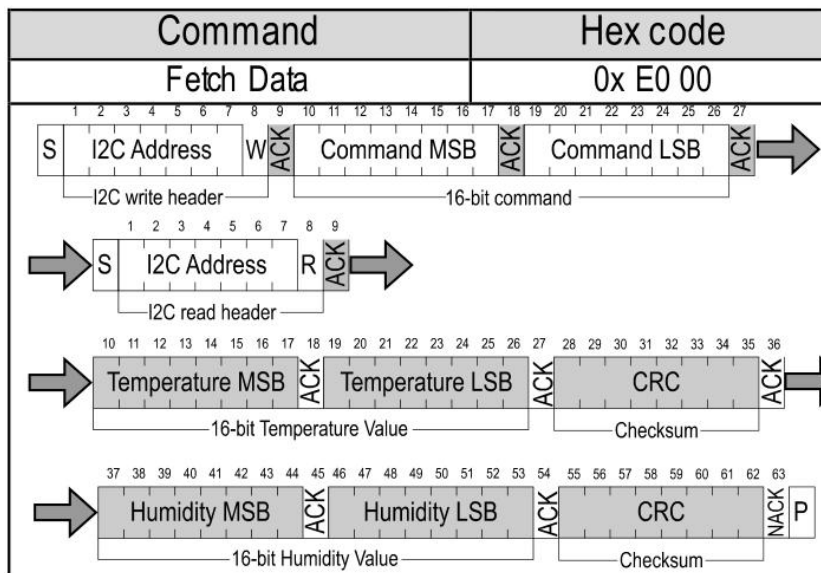
➤ SHT30 有两种通信格式，此处只讲解 I2C 通信，原理图如下：



➤ SHT30 写时序



➤ SHT30 读温度时序



➤ SHT30 其他命令的时序类似，请参考 SHT30 英文手册。

### 4. 硬件设计及原理

本实验板使用了 ESP32 的 I2C\_1，下表是我们的程序 IO 的映射。

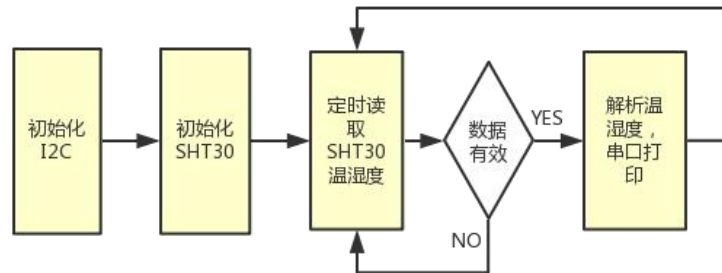
| I2C_1 | 功能 | 映射 ESP32 的引脚 |
|-------|----|--------------|
|-------|----|--------------|

|     |    |      |
|-----|----|------|
| SCL | 时钟 | I033 |
| SDA | 数据 | I032 |

若您使用的实验板 I2C 的连接方式或引脚不一样，只需根据我们的工程修改引脚即可，程序的控制原理相同。

## 5. 软件设计

### 5.1. 代码逻辑



### 5.2. ESP32 的 I2C master 接口介绍

此处的接口可以和 I2C 通信原理里面的名词对应上。

➤ I2C 配置函数: `i2c_param_config()`;

|      |   |
|------|---|
| 函数原型 | <pre> esp_err_t i2c_param_config (     i2c_port_t i2c_num,     const i2c_config_t* i2c_conf ) </pre>  |
| 函数功能 | I2C 配置函数  |
| 参数   | <p>[in] i2c_num: I2C 编号, 取值</p> <ul style="list-style-type: none"> <li>I2C_NUM_0 = 0, /*I2C_0 */</li> <li>I2C_NUM_1 , /*I2C_1*/</li> </ul> <p>[in] i2c_conf: I2C 参数配置</p> <pre> typedef struct{     i2c_mode_t mode; /*I2C 模式 */     gpio_num_t sda_io_num; /*SDA 引脚*/     gpio_pullup_t sda_pullup_en; /*SDA 上拉使能*/     gpio_num_t scl_io_num; /*SCL 引脚*/     gpio_pullup_t scl_pullup_en; /*SCL 上拉使能*/     union {         struct {             uint32_t clk_speed; /*时钟速度*/         } master;         struct {             uint8_t addr_10bit_en; /*10 位地址使能*/             uint16_t slave_addr; /*作为从机地址*/         } slave;     } } i2c_config_t; </pre> |

|     |   |
|-----|---|
|     | };<br>}i2c_config_t;                      |
| 返回值 | ESP_OK : 成功<br>ESP_ERR_INVALID_ARG : 参数错误 |

➤ I2C 功能安装使能函数: `i2c_driver_install()`;

|      |   |
|------|---|
| 函数原型 | <pre>esp_err_t i2c_driver_install (     i2c_port_t i2c_num,     i2c_mode_t mode,     size_t slv_rx_buf_len,     size_t slv_tx_buf_len,     int intr_alloc_flags )</pre> |
| 函数功能 | I2C 功能安装使能函数  |
| 参数   | [in] i2c_num: I2C 编号<br>[in] mode: I2C 模式<br>[in] slv_rx_buf_len: 接收缓存大小<br>[in] slv_tx_buf_len: 发送缓存大小<br>[in] intr_alloc_flags: 分配中断标记                                |
| 返回值  | ESP_OK : 成功<br>ESP_ERR_INVALID_ARG : 参数错误   |

➤ 创建 I2C 连接函数: `i2c_cmd_link_create()`;

|      |   |
|------|---|
| 函数原型 | <code>int i2c_cmd_link_create()</code>    |
| 函数功能 | 创建 I2C 连接函数                               |
| 参数   | [in] 无                                    |
| 返回值  | <code>i2c_cmd_handle_t</code> : I2C 连接的句柄 |

➤ 写启动信号到缓存函数: `i2c_master_start()`;

|      |   |
|------|---|
| 函数原型 | <pre>esp_err_t i2c_master_start (     i2c_cmd_handle_t cmd_handle )</pre> |
| 函数功能 | I2C 写启动信号到缓存函数  |
| 参数   | [in] cmd_handle: I2C 连接的句柄, <code>i2c_cmd_link_create()</code> 函数的返回值     |
| 返回值  | ESP_OK : 成功<br>ESP_ERR_INVALID_ARG : 参数错误                                 |

➤ 写一个字节命令放到缓存函数: `i2c_master_write_byte()`;

|      |   |
|------|---|
| 函数原型 | <pre>esp_err_t i2c_master_write_byte (     i2c_cmd_handle_t cmd_handle,     uint8_t data,     bool ack_en )</pre> |
| 函数功能 | I2C 写一个字节命令放到缓存函数   |
| 参数   | [in] cmd_handle: I2C 连接的句柄, <code>i2c_cmd_link_create()</code> 函数的返回值   |

|     |  |
|-----|--|
|     | [in] data:发送的数据<br>[in] ack_en:是否需要等待 ack 使能 |
| 返回值 | ESP_OK : 成功<br>ESP_ERR_INVALID_ARG : 参数错误    |

➤ 写停止信号到缓存函数: `i2c_master_stop()`;

|      |  |
|------|--|
| 函数原型 | <pre>esp_err_t i2c_master_stop (     i2c_cmd_handle_t cmd_handle )</pre> |
| 函数功能 | I2C 写停止信号到缓存函数   |
| 参数   | [in] cmd_handle:I2C 连接的句柄, <code>i2c_cmd_link_create()</code> 函数的返回值     |
| 返回值  | ESP_OK : 成功<br>ESP_ERR_INVALID_ARG : 参数错误                                |

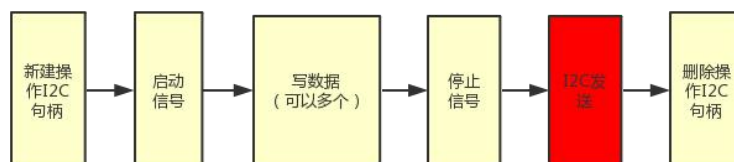
➤ I2C 发送函数: `i2c_master_cmd_begin()`;

|      |   |
|------|---|
| 函数原型 | <pre>esp_err_t i2c_master_cmd_begin (     i2c_port_t i2c_num,     i2c_cmd_handle_t cmd_handle,     TickType_t ticks_to_wait )</pre> |
| 函数功能 | I2C 发送函数  |
| 参数   | [in] i2c_num:I2C 编号<br>[in] cmd_handle:I2C 连接的句柄, <code>i2c_cmd_link_create()</code> 函数的返回值<br>[in] ticks_to_wait:等待时间              |
| 返回值  | ESP_OK : 成功<br>ESP_ERR_INVALID_ARG : 参数错误<br>ESP_FAIL : 发送错误<br>ESP_ERR_INVALID_STATE : I2C 设备未初始化<br>ESP_ERR_TIMEOUT : 超时          |

➤ 删除 I2C 连接函数: `i2c_cmd_link_delete()`;

|      |   |
|------|---|
| 函数原型 | <pre>void i2c_cmd_link_delete (     i2c_cmd_handle_t cmd_handle )</pre> |
| 函数功能 | I2C 发送启动信号函数  |
| 参数   | [in] cmd_handle:I2C 连接的句柄, <code>i2c_cmd_link_create()</code> 函数的返回值    |
| 返回值  | 无   |

以上是 I2C 发送数据的整个流程的 API, 流程如下图

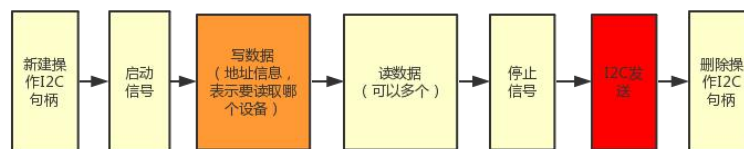




➤ 读一个字节的命令放到缓存函数: `i2c_master_read_byte()`;

|      |  |
|------|--|
| 函数原型 | <pre>esp_err_t i2c_master_read_byte (     i2c_cmd_handle_t cmd_handle,     uint8_t* data,     i2c_ack_type_t ack )</pre> |
| 函数功能 | I2C 读一个字节的命令放到缓存函数   |
| 参数   | [in] cmd_handle: I2C 连接的句柄, <code>i2c_cmd_link_create()</code> 函数的返回值<br>[in] data: 发送的数据<br>[in] ack: 应答的值              |
| 返回值  | ESP_OK : 成功<br>ESP_ERR_INVALID_ARG : 参数错误  |

以上是 I2C 读数据的整个流程的 API, 流程如下图



更多更详细接口请参考[官方指南](#)。

### 5.3. SHT30 温度采集代码编写

加载 I2C 相关的头文件、定义 I2C 的 IO 映射引脚、定义相关变量等。

```

1  #include <stdio.h>
2  #include "string.h"
3  #include "esp_system.h"
4  #include "esp_spi_flash.h"
5  #include "esp_wifi.h"
6  #include "esp_event_loop.h"
7  #include "esp_log.h"
8  #include "esp_err.h"
9  #include "nvs_flash.h"
10 #include "freertos/FreeRTOS.h"
11 #include "freertos/task.h"
12 #include "freertos/FreeRTOS.h"
13 #include "freertos/task.h"
14 #include "driver/gpio.h"
15 #include "driver/i2c.h"
16 //I2C
17 #define I2C_SCL_IO 33 //SCL->IO33
18 #define I2C_SDA_IO 32 //SDA->IO32
19 #define I2C_MASTER_NUM I2C_NUM_1 //I2C_1
20 #define WRITE_BIT I2C_MASTER_WRITE //写:0

```



```
21 #define READ_BIT I2C_MASTER_READ //读:1
22 #define ACK_CHECK_EN 0x1 //主机检查从机的 ACK
23 #define ACK_CHECK_DIS 0x0 //主机不检查从机的 ACK
24 #define ACK_VAL 0x0 //应答
25 #define NACK_VAL 0x1 //不应答
26 //SHT30
27 #define SHT30_WRITE_ADDR 0x44 //地址
28 #define CMD_FETCH_DATA_H 0x22 //循环采样, 参考 sht30 datasheet
29 #define CMD_FETCH_DATA_L 0x36
```

### ➤ I2C 配置函数

```
1 void i2c_init(void)
2 {
3     //i2c 配置结构体
4     i2c_config_t conf;
5     conf.mode = I2C_MODE_MASTER;           //I2C 模式
6     conf.sda_io_num = I2C_SDA_IO;          //SDA IO 映射
7     conf.sda_pullup_en = GPIO_PULLUP_ENABLE; //SDA IO 模式
8     conf.scl_io_num = I2C_SCL_IO;          //SCL IO 映射
9     conf.scl_pullup_en = GPIO_PULLUP_ENABLE; //SCL IO 模式
10    conf.master.clk_speed = 100000;         //I2C CLK 频率
11    i2c_param_config(I2C_MASTER_NUM, &conf); //设置 I2C
12    //注册 I2C 服务即使能
13    i2c_driver_install(I2C_MASTER_NUM, conf.mode, 0, 0, 0);
14 }
```

### ➤ SHT30 配置函数

```
1 int sht30_init(void)
2 {
3     int ret;
4     //配置 SHT30 的寄存器
5     i2c_cmd_handle_t cmd = i2c_cmd_link_create(); //新建操作 I2C 句柄
6     i2c_master_start(cmd);                        //启动 I2C
7     i2c_master_write_byte(cmd, SHT30_WRITE_ADDR << 1 | WRITE_BIT, ACK_CHECK_EN); //发地址+写+检查 ack
8     i2c_master_write_byte(cmd, CMD_FETCH_DATA_H, ACK_CHECK_EN); //发数据高 8 位+检查 ack
9     i2c_master_write_byte(cmd, CMD_FETCH_DATA_L, ACK_CHECK_EN); //发数据低 8 位+检查 ack
10    i2c_master_stop(cmd);                          //停止 I2C
11    ret = i2c_master_cmd_begin(I2C_MASTER_NUM, cmd, 100 / portTICK_RATE_MS); //I2C 发送
12    i2c_cmd_link_delete(cmd);                      //删除 I2C 句柄
13    return ret;
14 }
```

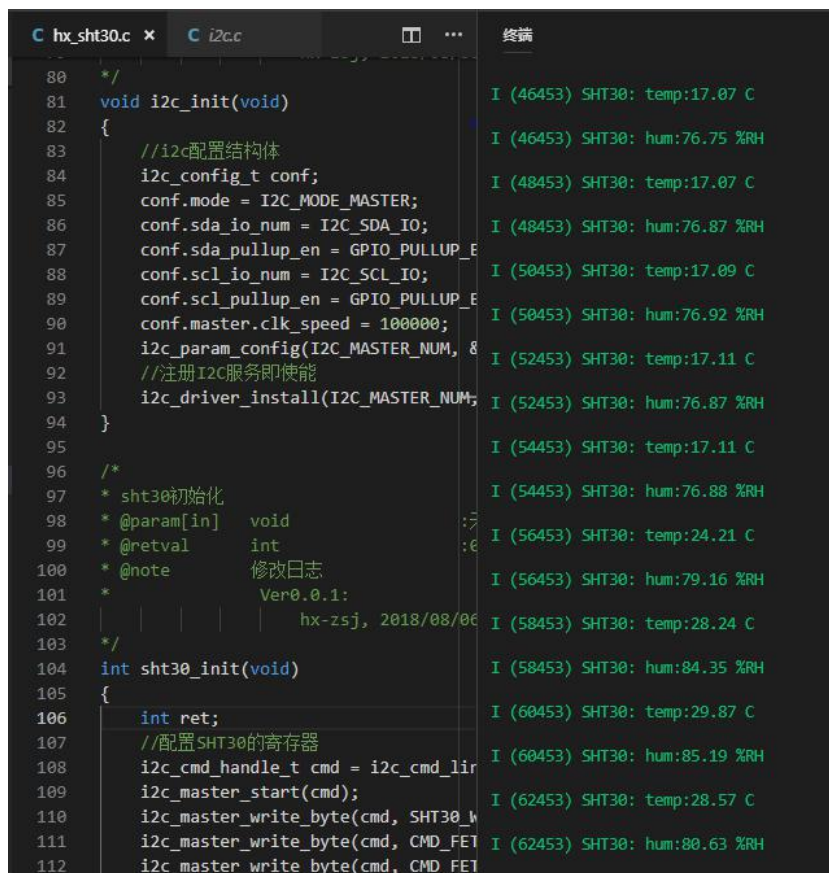
### ➤ 主函数: I2C 初始化、SHT30 初始化、定时读取温湿度值打印等。

```
1 void app_main()
2 {
3     i2c_init();                //I2C 初始化
4     sht30_init();              //sht30 初始化
5     vTaskDelay(100/portTICK_RATE_MS); //延时 100ms
6     while(1)
7     {
8         if(sht30_get_value()==ESP_OK) //获取温湿度
9         {
10             //算法参考 sht30 datasheet
11             g_temp  =( ( ( (sht30_buf[0]*256) +sht30_buf[1]) *175  )/65535.0  -45  );
12             g_rh   = ( ( (sht30_buf[3]*256) + (sht30_buf[4]) ) *100/65535.0) ;
13             ESP_LOGI("SHT30", "temp:%4.2f C \r\n", g_temp); //°C打印出来是乱码,所以用 C
14             ESP_LOGI("SHT30", "hum:%4.2f %%RH \r\n", g_rh);
15
16         }
17         vTaskDelay(2000/portTICK_RATE_MS);
18     }
19 }
```

#### 5.4. 硬件连接

可按照 IO 映射表将 SHT30 模块和 ESP32 开发板接好。

## 5.5. 效果展示



```

80  */
81  void i2c_init(void)
82  {
83      //i2c配置结构体
84      i2c_config_t conf;
85      conf.mode = I2C_MODE_MASTER;
86      conf.sda_io_num = I2C_SDA_IO;
87      conf.sda_pullup_en = GPIO_PULLUP_EN;
88      conf.scl_io_num = I2C_SCL_IO;
89      conf.scl_pullup_en = GPIO_PULLUP_EN;
90      conf.master.clk_speed = 100000;
91      i2c_param_config(I2C_MASTER_NUM, &conf);
92      //注册I2C服务即使能
93      i2c_driver_install(I2C_MASTER_NUM, I2C_MASTER_RX_BUF_DISABLE, I2C_MASTER_TX_BUF_DISABLE, 0, I2C_CMD_WRITE);
94  }
95
96  /*
97  * sht30初始化
98  * @param[in] void
99  * @retval int
100  * @note 修改日志
101  * Ver0.0.1:
102  * | | | | | hx-zsj, 2018/08/06
103  */
104  int sht30_init(void)
105  {
106      int ret;
107      //配置SHT30的寄存器
108      i2c_cmd_handle_t cmd = i2c_cmd_hlapi(I2C_MASTER_NUM, I2C_SHT30_ADDR, I2C_CMD_WRITE, 1);
109      i2c_master_start(cmd);
110      i2c_master_write_byte(cmd, SHT30_SOFT_RESET, 1);
111      i2c_master_write_byte(cmd, CMD_FET, 1);
112      i2c_master_write_byte(cmd, CMD_FET, 1);

```

Terminal Output:

```

I (46453) SHT30: temp:17.07 C
I (46453) SHT30: hum:76.75 %RH
I (48453) SHT30: temp:17.07 C
I (48453) SHT30: hum:76.87 %RH
I (50453) SHT30: temp:17.09 C
I (50453) SHT30: hum:76.92 %RH
I (52453) SHT30: temp:17.11 C
I (52453) SHT30: hum:76.87 %RH
I (54453) SHT30: temp:17.11 C
I (54453) SHT30: hum:76.88 %RH
I (56453) SHT30: temp:24.21 C
I (56453) SHT30: hum:79.16 %RH
I (58453) SHT30: temp:28.24 C
I (58453) SHT30: hum:84.35 %RH
I (60453) SHT30: temp:29.87 C
I (60453) SHT30: hum:85.19 %RH
I (62453) SHT30: temp:28.57 C
I (62453) SHT30: hum:80.63 %RH

```

## 6. 温湿度传感器总结

- 乐鑫已经把 I2C 部分的 API 封装的非常好，所以流程显得非常重要，无论什么 I2C 设备，一定要知道设备的 I2C 读写的流程。
- 温湿度传感器如果在产品中想要保证准确度，传感器必须放置在产品边缘，产品外壳挖孔，传感器周围挖空等措施，在使用软件校正的办法保证最终的准确度。
- 源码地址: <https://github.com/xiaolongba/wireless-tech>