SR04 超声波传感器实验

淘宝店铺：

PC 端：

http://n-xytrt8gqu585po94mwj5atokcyd4.taobao.com/index.htm

手机端：

https://shop.m.taobao.com/shop/shop_index.htm?sellerId=755668508&shopId=104493595&inShopPageId=423890608&pathInfo=shop/index2

资料下载地址：

链接：https://pan.baidu.com/s/1kCjD8yktZECSGmHomx_veg?pwd=q8er
提取码：q8er

源码下载地址：

https://gitee.com/vi-iot/esp32-board.git

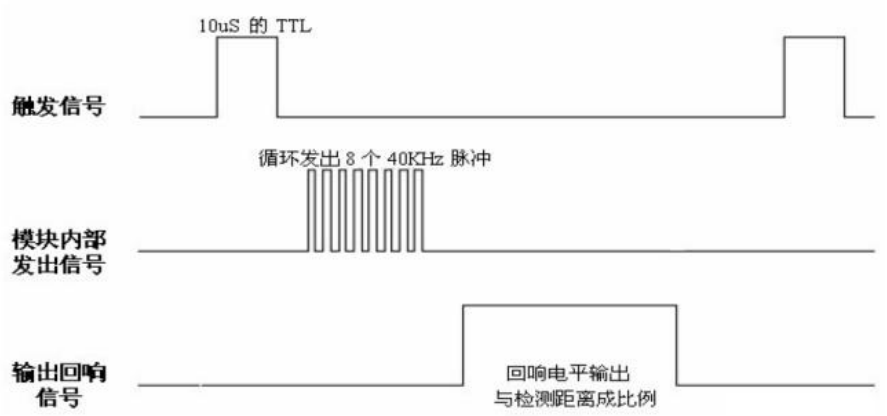
一、HC-SR04 介绍

HC-SR04 超声波测距模块是依据超声波遇到障碍物发生反射的原理进行测距的。HC-SR04 能够产生发射超声波信号，接收并且处理超声波，最后输出一段和发送与接收间隔时间相同的高电平信号，是物联网实验中常用的测距模块之一。

HC-SR04 有 4 个引脚功能如下

管脚	定义	功能
VCC	电源	3.3-5V
Trig	触发	输入触发脉冲
Echo	回响	测距完毕，会产生回响电平
GND	地	地

工作时序如下，通过 Trig 引脚，触发一个 10us 左右电平，然后模块内部自动会发出 8 个 40KHz 的超声波，我们的程序就监控回响引脚 Echo，Echo 会返回一段高电平持续时间，这个就是超声波从发射到返回的时间。



那么我们假设获取到的时间是 T，声波在空气中传播速度大概是 v=340m/s，我们可以计算单向距离 D=T*v/2。

二、控制程序

在了解完 HC-SR04 的工作原理后，回到我们实验程序，程序最关键的地方在于获取 Echo 的电平时间，换句话说，我们要捕获 Echo 引脚上升沿和下降沿对应的时间，然后相减。由于时间是微秒级别，我们用软件的方法误差会很大，因此最好的方法是使用定时器功能中的输入捕获功能。输入捕获可以在捕获到相应的边沿信号时进入中断函数，在中断函数里我们将定时器的计数值保存起来。

在 esp-idf 中，mcpwm 组件模块包含了输入捕获功能，原先这个组件模块是用来驱动无刷直流电机的，在本例程中我们不需要设置 6 路 PWM，只需要用到输入捕获功能，这样程序就比较简单了。

本例程的工程目录在 esp32-board/sr04

先来看下部分初始化程序

```
//新建一个捕获定时器，使用默认的时钟源
    mcpwm_cap_timer_handle_t cap_timer = NULL;
    mcpwm_capture_timer_config_t cap_conf = {
```

```
        .clk_src = MCPWM_CAPTURE_CLK_SRC_DEFAULT,
        .group_id = 0,
    };
    ESP_ERROR_CHECK(mcpwm_new_capture_timer(&cap_conf, &cap_timer));

    //新建一个捕获通道，把捕获定时器与捕获通道绑定起来，采取双边捕获的策略
    ESP_LOGI(TAG, "Install capture channel");
    mcpwm_cap_channel_handle_t cap_chan = NULL;
    mcpwm_capture_channel_config_t cap_ch_conf = {
        .gpio_num = HC_SR04_ECHO_GPIO,
        .prescale = 1,
        // capture on both edge
        .flags.neg_edge = true,
        .flags.pos_edge = true,
        // pull up internally
        .flags.pull_up = true,
    };
    ESP_ERROR_CHECK(mcpwm_new_capture_channel(cap_timer, &cap_ch_conf,
&cap_chan));
    //新建一个捕获事件回调函数，当捕获到相关的边沿，则调用这个回调函数
    ESP_LOGI(TAG, "Register capture callback");
    TaskHandle_t cur_task = xTaskGetCurrentTaskHandle();
    mcpwm_capture_event_callbacks_t cbs = {
        .on_cap = hc_sr04_echo_callback,
    };
    ESP_ERROR_CHECK(mcpwm_capture_channel_register_event_callbacks(cap_c
han, &cbs, cur_task));
    //使能捕获通道
    ESP_LOGI(TAG, "Enable capture channel");
    ESP_ERROR_CHECK(mcpwm_capture_channel_enable(cap_chan));
    //初始化 Trig 引脚，设置为输出
    ESP_LOGI(TAG, "Configure Trig pin");
    gpio_config_t io_conf = {
        .mode = GPIO_MODE_OUTPUT,
        .pin_bit_mask = 1ULL << HC_SR04_TRIG_GPIO,
    };
    ESP_ERROR_CHECK(gpio_config(&io_conf));
    ESP_ERROR_CHECK(gpio_set_level(HC_SR04_TRIG_GPIO, 0));
    //使能捕获定时器
    ESP_LOGI(TAG, "Enable and start capture timer");
    ESP_ERROR_CHECK(mcpwm_capture_timer_enable(cap_timer));
    ESP_ERROR_CHECK(mcpwm_capture_timer_start(cap_timer));
```

初始化中首先要初始化新建一个捕获定时器 mcpwm_cap_timer_handle_t，这个定时器

采用默认的时钟源，那么这个时钟频率是多少？因为这里默认采用的是内部经过分频的 APB 时钟，我们调用 esp_clk_apb_freq()函数就可以获得频率，具体是多少我们不用关心。然后需要设定捕获通道 mcpwm_capture_channel_config_t，将捕获定时器和这个捕获通道关联起来，上升沿和下降沿均捕获，设定捕获的 GPIO 脚。后续我们还要设定一个中断回调函数 mcpwm_capture_event_callbacks_t，当发生捕获事件时会调用这个函数。其余就是初始化 IO 口和启动的代码，大家自行看代码注释即可。

接下来我们看看中断回调函数的内容

```
static bool hc_sr04_echo_callback(mcpwm_cap_channel_handle_t cap_chan,
const mcpwm_capture_event_data_t *edata, void *user_data)
{
    static uint32_t cap_val_begin_of_sample = 0;
    static uint32_t cap_val_end_of_sample = 0;
    TaskHandle_t task_to_notify = (TaskHandle_t)user_data;
    BaseType_t high_task_wakeup = pdFALSE;

    //判断边沿，上升沿记录捕获的计数值
    if (edata->cap_edge == MCPWM_CAP_EDGE_POS) {
        // store the timestamp when pos edge is detected
        cap_val_begin_of_sample = edata->cap_value;
        cap_val_end_of_sample = cap_val_begin_of_sample;
    } else {
        //如果是下降沿，也记录捕获的计数值
        cap_val_end_of_sample = edata->cap_value;
        //两个计数值的差值，就是脉宽长度
        uint32_t tof_ticks = cap_val_end_of_sample -
cap_val_begin_of_sample;
        // 通知任务计数差值
        xTaskNotifyFromISR(task_to_notify, tof_ticks,
eSetValueWithOverwrite, &high_task_wakeup);
    }
    return high_task_wakeup == pdTRUE;
}
```

中断函数里面，我们分别判断捕获的边沿电平，然后分别将值记录下来，当检测到下降沿时，将下降沿记录的时钟值和上升沿记录的时钟值作差，通过任务通知方式发送给到主任务。

主任务代码如下

```
if (xTaskNotifyWait(0x00, ULONG_MAX, &tof_ticks, pdMS_TO_TICKS(1000)) ==
pdTRUE) {
            //计算脉宽时间长度，从任务通知获取到的 tof_ticks 是计数，我们需要计
算出对应的时间
            /*
            tof_ticks:计数
            esp_clk_apb_freq():计数时钟频率
```

```
            1/esp_clk_apb_freq():计数时钟周期，也就是 1 个 tof_ticks 时间是多
少秒
            1000000/esp_clk_apb_freq():一个 tof_ticks 时间是多少 us
            因此 tof_ticks * (1000000.0 / esp_clk_apb_freq())就是总的脉宽时
长
            */
            float pulse_width_us = tof_ticks * (1000000.0 /
esp_clk_apb_freq());
            if (pulse_width_us > 35000) {
                // 脉宽太长，超出了 SR04 的计算范围
                continue;
            }
            /*
            计算公式如下：
            距离 D=音速 V*T/2;(T 是上述的脉宽时间，因为是来回时间，所以要除以 2
才是单程时间)
            D(单位 cm)=340m/s*Tus;
            D=34000cm/s*(10^-6)Ts/2;(1us=(10^-6)s)
            D=17000*(10^-6)*T
            D=0.017*T
            D=T/58
            所以下面这个公式/58 是这样来的
            */
            float distance = (float) pulse_width_us / 58;
            ESP_LOGI(TAG, "Measured distance: %.2fcm", distance);
        }
```

从 xTaskNotifyWait 函数中获取到差值，计算出对应的时间，

时间可以这样算，

esp_clk_apb_freq()是时钟频率，

1/esp_clk_apb_freq()是时钟周期（单位是 S）

d=1000000/esp_clk_apb_freq()是时钟周期（单位是 US）

因此差值 tof_ticks*d 就是脉冲时间(单位是 US)

有了脉冲宽度，就可以计算距离了，公式比较简单，大家不懂的可以看代码的注释，注释有很详细的解读。