ADC 获取 NTC 温度

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

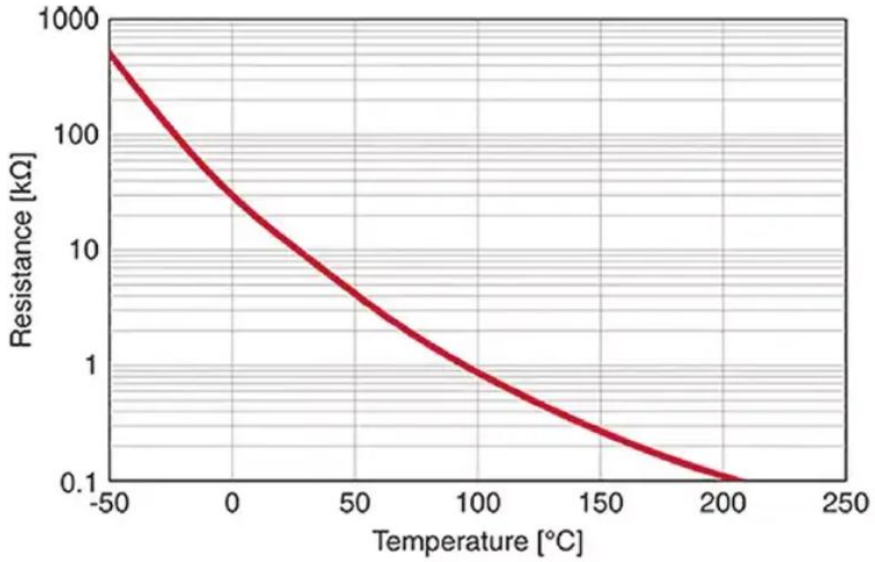
一、NTC 介绍

NTC 是 Negative Temperature Coefficient 的缩写，一般指负温度系数半导体器件，而在我们物联网实验中，称为 NTC 热敏电阻。NTC 热敏电阻阻值计算公式如下：

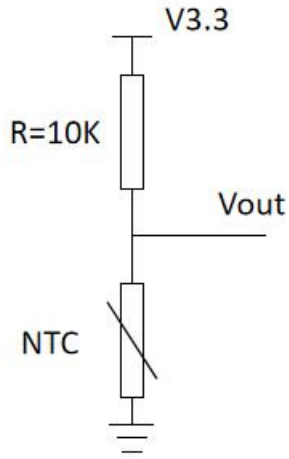
$$R_T = R_{T_0} * e^{B_n*(\frac{1}{T}-\frac{1}{T_0})}$$

式中 RT、 RT0 分别为温度 T 、 T0 时的电阻值，Bn 为材料常数。

对于 T0=25 ℃，R0=10KΩ ，Bn=3950 的电阻-温度曲线如下所示



由此可见，温度越高，阻值越小。对于 NTC 的电路相对简单，如下图



当 NTC 阻值发生变化时，Vout 也随之发生变化，通过采样 Vout 电压，然后根据欧姆定律计算出电阻值，再由电阻值可算出对应的温度值。

二、NTC 例程

由上节可知，我们需要采样 Vout 的值，需要用到 ESP32 的 ADC 功能，esp-idf 库对 ADC 的操作已经封装的相当好，直接引出 API 给我们使用，虽然如此，但依然有一些东西我们要注意，开发板中的 Vout 接到了 GPIO36 上，关于 ESP32 的 ADC，ESP32 上具有两个 ADC 转化模块，分别是 ADC1 和 ADC2，每个 ADC 模块均具有 8 路，由于在启用 ADC2 时，无法使用

WIFI 功能，因此本例程不介绍 ADC2，也不推荐大家使用 ADC2。然后并不是所有的 GPIO 口都具有 ADC 功能，只有如下 GPIO 口具有 ADC 功能

1) GPIO32 ADC1_CH4
2) GPIO33 ADC1_CH5
3) GPIO34 ADC1_CH6
4) GPIO35 ADC1_CH7
5) GPIO36 ADC1_CH0
6) GPIO37 ADC1_CH1
7) GPIO38 ADC1_CH2
8) GPIO39 ADC1_CH3

以下是部分初始化代码

```
/**
 * 温度检测初始化
 * @param 无
 * @return 无
*/
void temp_ntc_init(void)
{
    adc_oneshot_unit_init_cfg_t init_config1 = {
        .unit_id = ADC_UNIT_1,  //WIFI 和 ADC2 无法同时启用，这里选择 ADC1
    };
    //启用单次转换模式
    ESP_ERROR_CHECK(adc_oneshot_new_unit(&init_config1, &s_adc_handle));
    //-------------ADC1 Config---------------//
    adc_oneshot_chan_cfg_t config = {
        .bitwidth = ADC_BITWIDTH_12,       //分辨率
        .atten = ADC_ATTEN_DB_12,
        //衰减倍数，ESP32 设计的 ADC 参考电压为 1100mV,只能测量 0-1100mV 之间的电压，如果要测量更大范围的电压
        //需要设置衰减倍数
        /*以下是对应可测量范围
        ADC_ATTEN_DB_0      100 mV ~ 950 mV
        ADC_ATTEN_DB_2_5    100 mV ~ 1250 mV
        ADC_ATTEN_DB_6      150 mV ~ 1750 mV
        ADC_ATTEN_DB_12     150 mV ~ 2450 mV
        */
    };
    ESP_ERROR_CHECK(adc_oneshot_config_channel(s_adc_handle, TEMP_ADC_CHANNEL, &config));
    //-------------ADC1 Calibration Init---------------//
    do_calibration1 = example_adc_calibration_init(ADC_UNIT_1, ADC_ATTEN_DB_12, &adc1_cali_handle);
    //新建一个任务，不断地进行 ADC 和温度计算
```

```
    xTaskCreatePinnedToCore(temp_adc_task, "adc_task", 2048, NULL,2, NULL,
1);
}
```

这里先说下 adc_oneshot_new_unit 这个函数，这个函数是启用单次转换，ESP32 中有两种转化模式，分别是单次转换和连续转换，单次转换的意思是，我启动 ADC 转换，ADC 模块就只转换一次值然后停止，连续转换是启动 ADC 转换后，ADC 模块会不断地执行 ADC 转换，除非我们手动调用停止。经过我本人亲自的试验，连续转换的精度非常差，而且还受其他通道影响，可能之后乐鑫官方后续推出的系列芯片会修复这些问题，因此本教程只用单次转换。然后需要填充 adc_oneshot_chan_cfg_t 结构体，这个结构体只有两项，分辨率和衰减系数，分辨率的意思是，采样回来的最大值，比如说我们满量程是 2450mV，分辨率设置成 12 位，如果外部输入的电压是 2450mV 则，我们通过 adc_oneshot_read 读取到的值是 2^12-1=4095。本例程中 ADC_BITWIDTH_12 配置成 12 位分辨率。.atten = ADC_ATTEN_DB_12 这个特性可以说是 ESP32 较特殊的特性，ESP32 内部的 ADC 参考电压只有 1100mV，理论上最大只能采样 1100mV，如果我们要采样大于这个值，我们就必须设置衰减，让外部电压到了 ESP32 内部后进行衰减，然后整体来看，我们就可以采用大于 1100mV 电压。以下是衰减对应的测量范围

1) ADC_ATTEN_DB_0　　100 mV ~ 950 mV
2) ADC_ATTEN_DB_2_5　　100 mV ~ 1250 mV
3) ADC_ATTEN_DB_6　　150 mV ~ 1750 mV
4) ADC_ATTEN_DB_12　　150 mV ~ 2450 mV

由此可见，最大的衰减倍数，最高能测量的电压是 2450mV，当输入大于这个值（注意，不能超过 3300mV，否则会损坏芯片），程序中读取到的值都是 4095。

example_adc_calibration_init 函数里面用 ESP32 芯片内部预烧录的参数值对电压采样结果进行校准。

最后新建一个 temp_adc_task 函数不断读取 ADC 值

我们接下来看一下 temp_adc_task 这个函数

```
static void temp_adc_task(void* param)
{
    uint16_t adc_cnt = 0;
    while(1)
    {
        adc_oneshot_read(s_adc_handle, TEMP_ADC_CHANNEL,
&s_adc_raw[adc_cnt]);
        if (do_calibration1) {
            adc_cali_raw_to_voltage(adc1_cali_handle, s_adc_raw[adc_cnt],
&s_voltage_raw[adc_cnt]);
        }
        adc_cnt++;
        if(adc_cnt >= 10)
        {
            int i = 0;
            //用平均值进行滤波
            uint32_t voltage = 0;
```

```
            uint32_t res = 0;
            for(i = 0;i < ADC_VALUE_NUM;i++)
            {
                voltage += s_voltage_raw[i];
            }
            voltage = voltage/ADC_VALUE_NUM;
            if(voltage < ADC_V_MAX)
            {
                //电压转换为相应的电阻值
                res = (voltage*NTC_RES)/(ADC_V_MAX-voltage);
                //根据电阻值查表找出对应的温度
                s_temp_value = get_ntc_temp(res);
            }
            adc_cnt = 0;
        }
        vTaskDelay(pdMS_TO_TICKS(100));
    }
}
```

adc_oneshot_read 函数触发一次转换，并且将转换后的值返回。
adc_cali_raw_to_voltage 函数将读取的 ADC 值转化成相应的电压值
这个任务重每个周期会执行 10 次转换，并且将转化值保存在 s_voltage_raw 中，10 次转换后，通过计算平均值，达到滤波的效果。
由上节的采样电路图以及欧姆定律可知，电阻计算公式如下
R=(V*10K)/(3.3V-V);
其中 V 是采样得到的数值。10K 是 NTC 参考电阻。3.3V 是电路中的参考电压（非 ESP32 内部 ADC 参考电压）
计算得到电阻值后，我们就可以通过电阻值 R 获取到温度值了。
那问题来了

$$R_T = R_{T_0} * e^{B_n*(\frac{1}{T}-\frac{1}{T_0})}$$

，这个公式这么复杂，代入进去计算基本算不出来，一般我们不这样做，我们一般通过查表来得到温度值。如何做呢？
当我们买了 NTC 热敏电阻后，厂家一般都会给我们一份 NTC 电阻温度对应表。如下所示

温度（℃）	欧姆	温度（℃）	阻值(Ω)	温度（℃）	阻值(Ω)	温度（℃）	阻值(Ω)	温度（℃）	阻值(Ω)
-40	336600	-1	34380	38	5776	77	1385	116	433.4
-39	315000	0	32660	39	5546	78	1341	117	421.8
-38	295000	1	31040	40	5326	79	1298	118	410.6
-37	276400	2	29500	41	5118	80	1256	119	399.8
-36	259000	3	28060	42	4918	81	1216	120	389.4
-35	242800	4	26680	43	4726	82	1178	121	379.2
-34	227800	5	25400	44	4544	83	1141	122	369.4
-33	213800	6	24180	45	4368	84	1105	123	359.8

具体的还要参考厂家提供。那我们如何在程序中使用这个表呢？首先我们需要建一个常量数组，把这些值保存起来，在本例程中是这样做的。

```
//NTC温度表
static const temp_res_t s_ntc_table[] =
{
    {-10,51815},
    {-9,49283},
    {-8,46889},
    {-7,44624},
    {-6,42481},
    {-5,40450},
    {-4,38526},
    {-3,36702},
    {-2,34971},
    {-1,33329},
    {0,31770},
    {1,30253},
    {2,28815},
    {3,27453},
    {4,26160},
```

这个数组单元是一个结构体，第一个成员是温度值，第二个成员是电阻值。
现在有两个问题
1）查找效率
2）值问题

先看查找效率，我们计算出电阻值后，如果查找。一种最笨的方法是遍历，但相对耗时，还有一种是二分查找，二分查找的前提是，表是按顺序排列的，我们看这个表，阻值是从大大小的顺序排列的，因此我们可以用二分查找的方法进行查找，代码如下：

```
/** 二分查找，通过电阻值查找出温度值对应的下标
 * @param res 电阻值
 * @param left ntc 表的左边界
 * @param right ntc 表的右边界
 * @return 温度值数组下标
*/
static int find_ntc_index(uint32_t  res,uint16_t left, uint16_t right)
{
    uint16_t middle = (left + right) / 2;
    if (right <= left || left == middle)
    {
        if (res >= s_ntc_table[left].res)
            return left;
        else
            return right;
    }
    if (res > s_ntc_table[middle].res)
    {
        right = middle;
        return find_ntc_index(res,left, right);
```

```
    }
    else if (res < s_ntc_table[middle].res)
    {
        left = middle;
        return find_ntc_index(res,left, right);
    }
    else
        return middle;
}
```

再看问题 2

我们计算出的电阻值，很有可能在这个表上查不到。那我们如何处理呢？上述经过二分查找后，我们会返回一个下标，这个下标对应数组中的电阻值会比计算出来的电阻值稍小，而这个下标的下一个数组值又会比计算出来的电阻值稍大。伪代码就是

R >= s_ntc_table[index].rec;

R< s_ntc_table[index+1].rec;

R 是我们采样电压计算出来的电阻值，index 是二分查找返回的数组下标

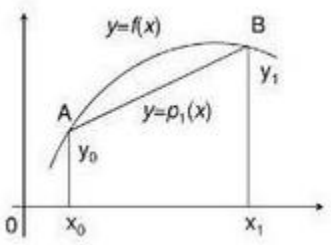
假设我们刚查到这个电阻值在表中，那么很简单，直接用下标对应的温度即可。但很多时候我们不会这么理想，大多数情况都是 R>s_ntc_table[index].rec && R < s_ntc_table[index+1].rec。这种情况我们该如何处理最好？

有两种处理方式，

第一种我们直接用 s_ntc_table[index].temp 或 s_ntc_table[index+1].temp 就行，简单粗暴，这种没有计算出小数点，因为表中的温度值全部是整数。

第二种就是使用线性插值

何为线性插值？请看下图



假设 y=f(x)是一段较复杂的曲线方程，而坐标(X0，Y0)和(X1，Y1)是落在这条曲线上的两点，因为这个方程比较复杂，我们如果有一些点需要计算，直接代入到原方程计算比较困难，如果要计算的点满足 1）落在 X0，X1 之间，2）X0 和 X1 距离比较近，这时后我们就可以把坐标(X0，Y0)和(X1，Y1)之间当成是一条直线。通过两点式计算出这条直线的方程，然后再将要计算的值代入这条直线方程，计算就简单多了。代码如下：

```
 /* @param x 需要计算的 X 坐标
 * @param x1,x2,y1,y2 两点式坐标
 * @return y 值
*/
static float linera_interpolation(int32_t x,int32_t x1, int32_t x2, int32_t
y1, int32_t y2)
{
```

```
    float k = (float)(y2 - y1) /(float)(x2 - x1);
    float b = (float)(x2 * y1 - x1 * y2) / (float)(x2 - x1);
    float y = k * x + b;
    return y;
}
```

本例程中，
X1=s_ntc_table[index].res
X2=s_ntc_table[index+1].res
Y1=s_ntc_table[index].temp
Y2=s_ntc_table[index+1].temp
X = R(采用电压对应计算出来的值)
而返回了的 Y 值就是线性插值计算出来的温度值

app_main()函数比较简单

```
void app_main(void)
{
    temp_ntc_init();
    while(1)
    {
        ESP_LOGI(TAG,"current temp:%.2f",get_temp());
        vTaskDelay(pdMS_TO_TICKS(1000));
    }

}
```

每隔 1000ms 获取温度值并且打印

完成的例程可以参考 esp32-board/ntc