参考网页：

[ESP32学习笔记（22）——ADC接口使用\_Leung的博客-CSDN博客\_adc esp32](https://blog.csdn.net/qq_36347513/article/details/117567677)

[Analog to Digital Converter (ADC) - ESP32 - — ESP-IDF Programming Guide latest documentation (espressif.com)](https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/adc.html)

1. 概述

ESP32 集成了 2 个 12 位逐次逼近模数转换器 (SARADC)，支持 18 个测量通道(模拟使能引脚)。

支持以下通道：

ADC1：

8通道：GPIO32 - GPIO39

ADC2：

10个通道：GPIO0、GPIO2、GPIO4、GPIO12-GPIO15、GOIO25-GPIO27

1. ADC限制

Wi-Fi 驱动程序使用了 ADC2。因此，应用程序只能在未启动 Wi-Fi 驱动程序时使用 ADC2。

某些 ADC2 引脚用作捆扎引脚(GPIO 0,2,15)，因此无法自由使用。

ESP32 DevKitC : 由于外部自动编程电路，无法使用 GPIO 0。

ESP-WROVER-KIT : GPIO 0, 2, 4 和 15 由于不同用途的外部连接而无法使用。

1. ADC采样模式

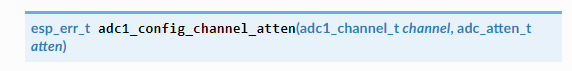
每个 ADC 单元支持两种工作模式，ADC 单次采样模式和ADC连续采样（DMA）模式。

* ADC 单次采样模式：适用于低频采样操作。
* ADC 连续采样(DMA)模式：适用于高频连续采样动作

注：在使用ADC时，最好加一个旁路电容，去噪。

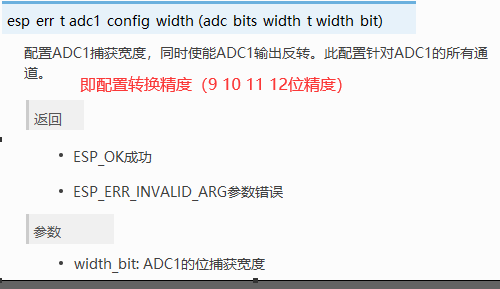
1. API参考

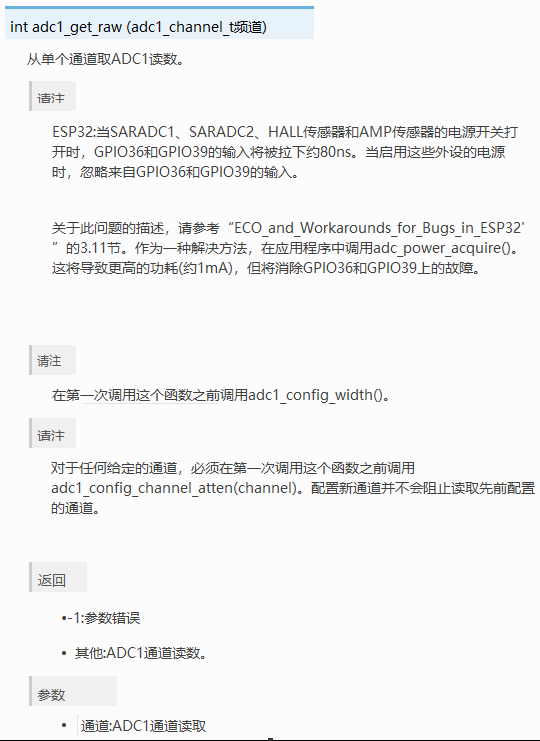




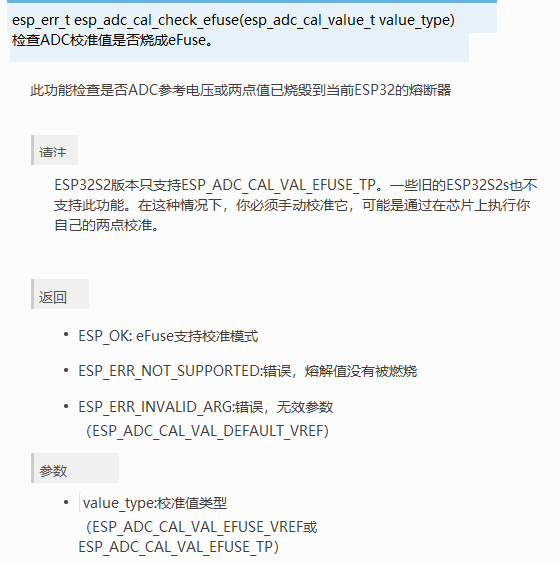
设置 ADC\_1的特定通道的衰减模式，以改变测量电压范围。

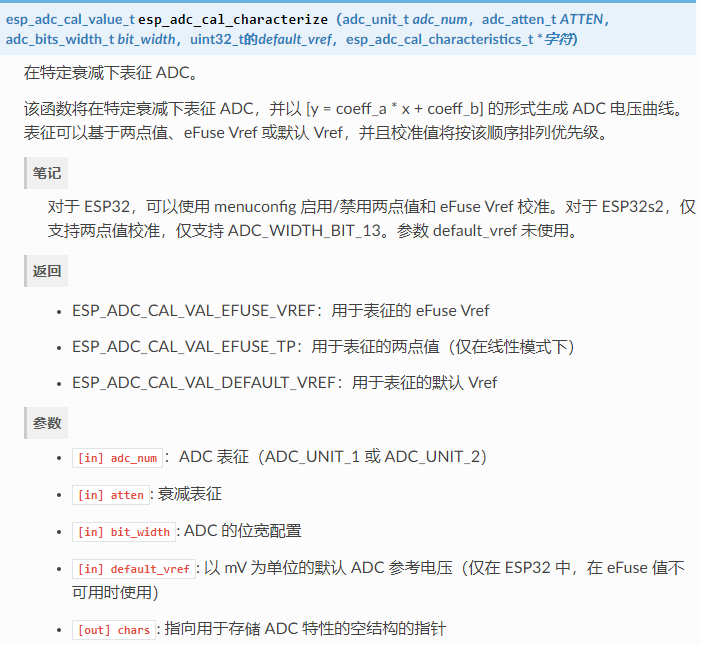
















1. 例程分析

/\* ADC1 Example

   This example code is in the Public Domain (or CC0 licensed, at your option.)

   Unless required by applicable law or agreed to in writing, this

   software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR

   CONDITIONS OF ANY KIND, either express or implied.

\*/

#include <stdio.h>

#include <stdlib.h>

#include "freertos/FreeRTOS.h"

#include "freertos/task.h"

#include "driver/gpio.h"

#include "driver/adc.h"

#include "esp\_adc\_cal.h"

#define **DEFAULT\_VREF**  1100  //参考电压

#define **NO\_OF\_SAMPLES**  64   //Multisampling

//定义一个ADC的参数结构体

static esp\_adc\_cal\_characteristics\_t \*adc\_chars;

#if **CONFIG\_IDF\_TARGET\_ESP32**

//选择采样通道：ADC\_6

static const adc\_channel\_t channel = ADC\_CHANNEL\_6;

//选择采样精度：12位

static const adc\_bits\_width\_t width = ADC\_WIDTH\_BIT\_12;

#elif **CONFIG\_IDF\_TARGET\_ESP32S2**

static const adc\_channel\_t channel = ADC\_CHANNEL\_6;

static const adc\_bits\_width\_t width = ADC\_WIDTH\_BIT\_13;

#endif

//选择ADC的通道的衰减模式，此参数影响测量电压的范围

//ADC\_ATTEN\_DB\_(0,2\_5,6,11)分别对应了 0dB 2.5dB 6dB 11dB衰减

static const adc\_atten\_t atten = ADC\_ATTEN\_DB\_11;

//选择ADC\_1

static const adc\_unit\_t unit = ADC\_UNIT\_1;

//检查eFuse是否准备就绪

static void **check\_efuse**(void)

{

#if **CONFIG\_IDF\_TARGET\_ESP32**

    //Check if TP is burned into eFuse

    //检查两点值(Two Point)是否烧录进 eFuse

    if (**esp\_adc\_cal\_check\_efuse**(ESP\_ADC\_CAL\_VAL\_EFUSE\_TP) == ESP\_OK) {

**printf**("eFuse Two Point: Supported\n");

    } else {

**printf**("eFuse Two Point: NOT supported\n");

    }

    //Check Vref is burned into eFuse

    //检查参考电压模式是否烧录进 eFuse

    if (**esp\_adc\_cal\_check\_efuse**(ESP\_ADC\_CAL\_VAL\_EFUSE\_VREF) == ESP\_OK) {

**printf**("eFuse Vref: Supported\n");

    } else {

**printf**("eFuse Vref: NOT supported\n");

    }

#elif **CONFIG\_IDF\_TARGET\_ESP32S2**

    if (**esp\_adc\_cal\_check\_efuse**(ESP\_ADC\_CAL\_VAL\_EFUSE\_TP) == ESP\_OK) {

**printf**("eFuse Two Point: Supported\n");

    } else {

**printf**("Cannot retrieve eFuse Two Point calibration values. Default calibration values will be used.\n");

    }

#else

#error "This example is configured for ESP32/ESP32S2."

#endif

}

//打印使用的特征值类别信息，此部分函数为 使用ADC时 非必须代码

static void **print\_char\_val\_type**(esp\_adc\_cal\_value\_t val\_type)

{

    if (val\_type == ESP\_ADC\_CAL\_VAL\_EFUSE\_TP) {

**printf**("Characterized using Two Point Value\n");

    } else if (val\_type == ESP\_ADC\_CAL\_VAL\_EFUSE\_VREF) {

**printf**("Characterized using eFuse Vref\n");

    } else {

**printf**("Characterized using Default Vref\n");

    }

}

void **app\_main**(void)

{

    //Check if Two Point or Vref are burned into eFuse

**check\_efuse**();

    //Configure ADC

    if (unit == ADC\_UNIT\_1) {

        //配置ADC\_1的转换精度

**adc1\_config\_width**(width);

        //配置ADC\_1的通道衰减模式

**adc1\_config\_channel\_atten**(channel, atten);

    } else {

**adc2\_config\_channel\_atten**((adc2\_channel\_t)channel, atten);

}

//Characterize ADC

    /\*  在内存中分配 1 个长度为 sizeof(esp\_adc\_cal\_characteristics\_t)

        的连续空间，并将起始地址指针返回给adc\_chars

        \*/

    adc\_chars = **calloc**(1, sizeof(esp\_adc\_cal\_characteristics\_t));

    //设置ADC的通道、衰减模式、精度、在eFuse值不可用时的参考电压、存储ADC参数特性的结构体指针

    esp\_adc\_cal\_value\_t val\_type = **esp\_adc\_cal\_characterize**(unit, atten, width, DEFAULT\_VREF, adc\_chars);

**print\_char\_val\_type**(val\_type);

    //Continuously sample ADC1

    while (1) {

        //定义一个接收获取到的ADC数字量值的变量

        uint32\_t adc\_reading = 0;

        //Multisampling

        //采样NO\_OF\_SAMPLES(64)次，然后取平均值

        for (int i = 0; i < NO\_OF\_SAMPLES; i++) {

            if (unit == ADC\_UNIT\_1) {

                adc\_reading += **adc1\_get\_raw**((adc1\_channel\_t)channel);

            } else {

                int raw;

**adc2\_get\_raw**((adc2\_channel\_t)channel, width, &raw);

                adc\_reading += raw;

            }

        }

        adc\_reading /= NO\_OF\_SAMPLES;

        //Convert adc\_reading to voltage in mV

        //将ADC\_1的取平均读数：adc\_reading 转换为以mV为单位的电压

        uint32\_t voltage = **esp\_adc\_cal\_raw\_to\_voltage**(adc\_reading, adc\_chars);

**printf**("Raw: %d\tVoltage: %dmV\n", adc\_reading, voltage);

**vTaskDelay**(**pdMS\_TO\_TICKS**(1000));//

    }

}