TG7120B

ADC Application Note

Version 0.2

Author:

Security: Public

Date: 2020.9

Revision History

Revision	Author	Date	Description
v0.1		2020.2.31	This document is used for TG7120B.

目录

图表目录

1 简介

ADC，全称 Analog-to-Digital Converter，此时 IO 口做模拟脚使用。

QFN32	Default mode	Default IN_OUT	ANA
P11	GPIO	IN	√
P14	GPIO	IN	√
P15	GPIO	IN	√
P18	GPIO	IN	√
P20	GPIO	IN	√

表 1：GPIO ANA 引脚

只有 P11~P20 支持模拟功能。不同引脚用途也不一样，共有如下三种用途：

- ADC 采集：采集引脚电压，P11、P14、P15、P18、P20。单端支持的引脚有 P11、P14、P15、P20，差分支持的引脚有 P20 P15。

2 ADC 典型应用

2.1 32.768K 晶振振荡电路

P16、P17 接 32.768Khz 晶振、电容组成振荡电路。其中 P16 为 XTALI，P17 为 XTALO。
目前，即使系统中采用 RC 32K，P16、P17 也不可作其他用途。

2.1.1 晶振振荡电路硬件注意事项

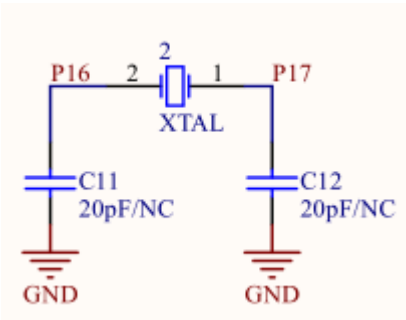

图 1：32K 晶振振荡电路

2.1.2 晶振振荡软件注意事项

软件可以根据应用需要，选择 32K 晶振或 RC32K 作为 32K 时钟源。

```
//system.c->SystemInit
#ifdef CONFIG_USE_XTAL_CLK
    g_clk32K_config = CLK_32K_XTAL; //选择32.768Khz晶振作为32K时钟源
#else
    g_clk32K_config = CLK_32K_RCOSC; //选择RC 32K晶振作为32K时钟源
```

```
#endif
```

```
//main.c->main
g_clk32K_config = CLK_32K_XTAL;//选择32.768Khz晶振作为32K时钟源
g_clk32K_config = CLK_32K_RCOSC; //选择RC 32K晶振作为32K时钟源
```

2.2 ADC 概述

芯片内部提供 ADC 基准电压，ADC 基准电压 0.8V。
芯片内置 12bit SAR ADC，共有 5 个输入端口可用。引脚按用途可分单端输入、差分输入。

- 单端输入：P11、P14、P15、P18、P20，典型应用 ADC 单端采集。
- 差分输入：P20(+)P15(-)，典型应用 ADC 差分采集。

硬件支持单端模式和差分模式：

- 单端模式：测量引脚和 GND 之间的电压。
- 差分模式：测量两个引脚之间的电压。

硬件支持手动模式和自动扫描模式：

- 手动模式：一次只支持一个单端通道或一组差分采集通道，使其能够将某种特殊的输入方式转化为单端或差分输入。
- 自动扫描：自动扫描所有已启用的多个单端通道，并将转换后的数据存储在相应的内存位置。

硬件支持 bypass 模式和 attenuation 模式：

- bypass 模式：引脚输入电压芯片内部直接进入 ADC，此时量程为 0~0.8V。
- attenuation 模式：引脚输入电压经芯片内部分压电阻后直接进入 ADC，分压电阻比约为 4：1，分压电阻阻值为一个 14.15K，一个 4.72K。此时理论量程为 0~3.2V。需要注意，电阻误差率为+-15%。不同通道的寄生电阻不一样，寄生电阻对衰减系数的影响已经包含在驱动中。

ADC 理论精度：

- bypass 模式： 单端理论精度 0.8/4096V，约为 0.2mV，实际小于 1mV。当使用 bpass 模式，外部采用分压电阻，此时再用外部分压电阻计算电压时，也要注意电阻的误差。
- attenuation 模式：芯片内部电阻误差约为+-15%。

ADC 支持采样速率有：80K，160K，320K，默认速率为 320K。
硬件支持芯片电源电压测量，所配置的模拟引脚在芯片内部已经做了处理，所以该引脚需要保持孤立。

注意事项：

- 不能同时使用内部分压电阻和外部分压电阻，即如使用 attenuation 模式，外部不要使用分压电阻。

2.2.1 ADC 硬件注意事项

当要采集的电压较小时，比如小于 0.8V，也就是在 bypass 模式量程内，直接使用 bypass 模式即可。注意采集引脚需要接滤波电容。

当要采集的电压略大时，比如大于 0.8V 但小于 3.2V，可使用 attenuation 模式或外加电阻分压后的 bypass 模式。芯片内部电阻误差+-15%，芯片外部电阻误差可选，一般+-1%。因此，对精度要求高时可使用外部电阻分压后的 bypass 模式。注意采集引脚需要接滤波电容。

当要采集的电压较大时，比如大于 3.2V，即超过 attenuation 模式的量程，此时必须采用外部电阻分压后的 bypass 模式。注意采集引脚需要接滤波电容。

当使用外部电阻分压时时，电阻电容需要满足一定的约束条件，如下图：

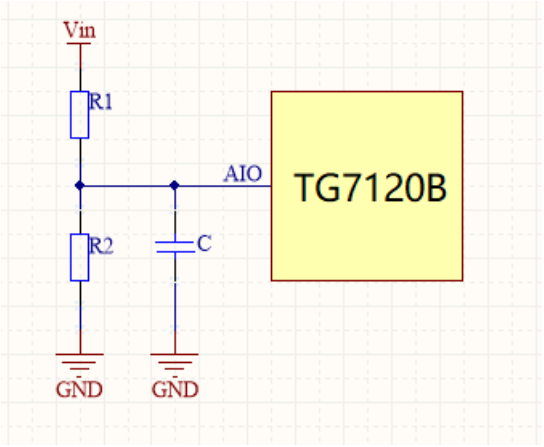

图 2：当使用外部电阻分压时

- 模式选择 bypass，测试量程为【0V，0.8V】。
- 检测电压V_{AIO}需要小于 0.8V 。

计算公式如下：

$$V_{AIO} = \frac{\frac{R2}{R1+R2}}{1+jw\frac{R1R2}{R1+R2}C}Vin$$

1. Vin 检测频率 $f_{in} < \frac{1}{2\pi\frac{R1R2}{R1+R2}C}$

2. 增益Gain $= \frac{R2}{R1+R2}$
3. Vin 驱动使能　R1//R2//C

2.2.2 ADC 软件注意事项

```
/*
支持bypass的adc和芯片电源电压检测(attenuation)，轮询方式读取adc值
比如开启P11和P20，其中P11用于检测芯片电源电压，P20用于检测adc
如果不检测芯片电源电压，用P11和P20检测adc，可以用drv_adc_config替换drv_adc_battery_config
*/
void get_battery_voltage(void)
{
    int32_t ret;
    adc_handle_t hd;
    adc_status_t adc_status;
    volatile uint32_t adc_data[4]={0};//save adc data
    uint32_t ch_adc[] = {ADC_CH1N_P11,ADC_CH3P_P20};//config adc pin
    adc_conf_t adc_config;

    adc_config.mode=ADC_SCAN;
    adc_config.intrp_mode=0;
    adc_config.channel_array=ch_adc;
    adc_config.channel_nbr=sizeof(ch_adc)/sizeof(ch_adc[0]);
    adc_config.conv_cnt = 10;

    if(adc_config.mode == ADC_SCAN)
    {
        hd = drv_adc_initialize(0,NULL);
        if(hd == NULL)
        {
            printf("err:%d",__LINE__);
        }

        ret = drv_adc_battery_config(hd,&adc_config,ADC_CH1N_P11);
        if(ret!=0)
        {
            printf("err:%d",__LINE__);
        }

        ret = drv_adc_start(hd);
        if(ret!=0)
        {
            printf("err:%d",__LINE__);
```

```
            }

            {
                mdelay(10);
                ret = drv_adc_read(hd,&adc_data[0],sizeof(ch_adc)/sizeof(ch_adc[0]));

                LOGI(TAG, "P11:%dmV    P20:%dmV\n",adc_data[0],adc_data[1]);
            }
        }

        ret = drv_adc_stop(hd);
        drv_adc_uninitialize(hd);
        return 0;
}
```

```
/*
结构体adc_Cfg_t可以配置adc的通道、模式等参数。
typedef struct _adc_Cfg_t{
   uint8_t channel;
   bool    is_continue_mode;
   uint8_t    is_differential_mode;
   uint8_t    is_high_resolution;
}adc_Cfg_t;
channel：用于配置通道， 单端模式支持多个通道，差分模式支持一组通道。每个bit对应一个单端通道或一组差分通道。
is_continue_mode：adc工作方式，一直打开还是间隔打开。TRUE一直打着，FALSE采集一次关闭，下次需要重新打开。
is_differential_mode：是否是差分模式，全0单端模式，否则差分模式
is_high_resolution：bypass模式和attenuation模式，每个bit代表一个单端通道或一组差分通道。0代表attenuation模式，1代表bypass模式。


*/
static adc_Cfg_t s_adc_cfg = {
        .channel = 0,
        .is_continue_mode = TRUE,
        .is_differential_mode = 0x00,
        .is_high_resolution = 0xFF, //默认bypass模式
};
```