



■ PHY622X

ADC Application Note

Version 1.0

Author: phyplusinc

Security: Public

Date: 2021.01

PhyPlus

Copyright © 2020 Phyplus Microelectronics Limited All rights reserved.
Reproduction in whole or in part is prohibited without the prior written permission of the copyright holder.



Revision History

Revision	Author	Date	Description
V1.0		2021.01.25	This document is used for 6220/6250/6222/6252.

目录

1	简介.....	1
2	ADC 典型应用.....	1
2.1	ADC 概述.....	1
2.1.1	ADC 硬件注意事项.....	2
2.1.2	ADC 软件注意事项.....	4
2.1.3	ADC 的校准原理.....	5
3	Voice 典型应用.....	6
3.1	Voice 概述.....	6
3.1.1	Voice 硬件注意事项.....	7
3.1.2	Voice 软件注意事项.....	9

图表目录

表 1: GPIO ANA 引脚.....	1
图 2: 当使用外部电阻分压时	3
表 2: ADC 衰减系数参考	6
图 3: DMIC 参考电路	7
图 4: 外部双端 AMIC 参考电路	8
图 5: 外部单端 AMIC 参考电路	9

1 简介

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

以 QFN32 为例，ADC 引脚如下表：

QFN32	Default mode	Default IN_OUT	ANA
P11	GPIO	IN	✓
P14	GPIO	IN	✓
P15	GPIO	IN	✓
P16	XTALI(ANA)	ANA	✓
P17	XTALO(ANA)	ANA	✓
P18	GPIO	IN	✓
P20	GPIO	IN	✓
P23	GPIO	IN	✓
P24	GPIO	IN	✓
P25	GPIO	IN	✓

表 1: GPIO ANA 引脚

只有 P11~P15、P20~P25 支持模拟功能。用途如下：

- ADC 采集：采集引脚电压，P11、P14、P15、P18、P20、P23、P24、P25。单端支持的引脚有 P11、P14、P15、P20、P23、P24，差分支支持的引脚有 P18P25、P23P11、P14P24、P20P15。
- VOICE 采集：内置 PGA，采集 PCM 原始数据，P18、P20、P15、P23。支持 DMIC 和 AMIC，使用 DMIC，引脚可灵活 FMUX 配置。使用 AMIC 时，只能 P18(PGA+)、P20(PGA-)、P15(micphone bias)、P23(micphone bias reference voltage)，其中 P23 是可选的。

2 ADC 典型应用

2.1 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 8 个输入端口可用。引脚按用途可分 PGA、单端输入、差分输入。

- PGA：P18(PGA+)、P20(PGA-)，典型应用接 AMIC 采集 VOICE。
- 单端输入：P11、P14、P15、P20、P23、P24，典型应用 ADC 单端采集。

- 差分输入：P18(+)/P25(-)、P23(+)/P11(-)、P14(+)/P24(-)、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。

ADC 的时钟来源于 HCLK，当 HCLK 为 32M、64M 时，ADC 时钟为 1.28Mhz；否则 ADC 时钟为 1Mhz。

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。
- 自动模式：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。SDK ADC 工作在自动模式下。一次 ADC 采样耗时由 ADC 采样时间和 ADC 转换时间组成，两者均可配，前者是 2T 和 3T，后者 3T 和 2T，T 为 ADC 时钟的周期。

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。

ADC 理论精度：

- bypass 模式：单端理论精度 0.8/4096V，约为 0.2mV0。当使用 bypass 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：芯片内部电阻绝对误差约为+/-15%，相对误差约为+/-1%。绝对误差不影响 ADC 精度，相对误差会影响 ADC 精度。

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。

2.1.1 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。ADC 精度取决于电阻的相对精度，attenuation 模式内部相对精度为+/-1%，外部电阻也可以选用+/-1%的电阻。

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：

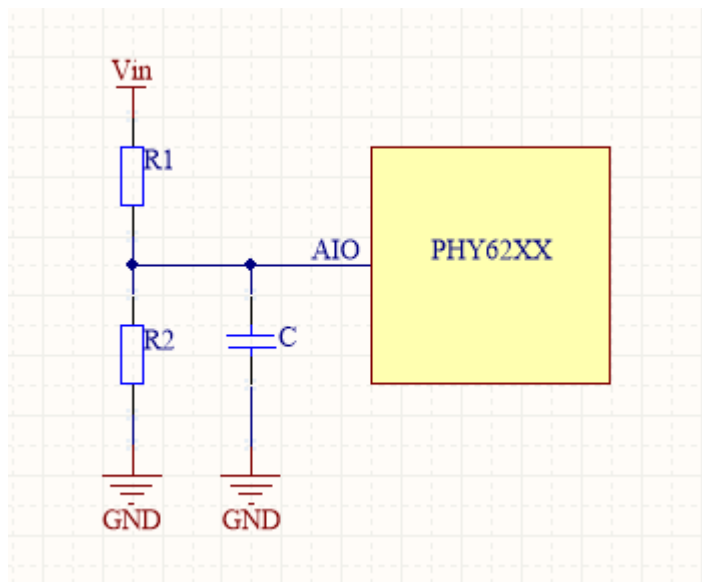


图 1：当使用外部电阻分压时

- 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：

$$V_{AIO} = \frac{R2}{R1 + R2} V_{in}$$

$$1 + j\omega \frac{R1R2}{R1 + R2} C$$

1. V_{in} 检测频率 $f_{in} < \frac{1}{2\pi \frac{R1R2}{R1 + R2} C}$
2. 增益 $Gain = \frac{R2}{R1 + R2}$
3. V_{in} 驱动使能 $R1 // R2 // C$

2.1.2 ADC 软件注意事项

```

/*
6220/6250
6220/6250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;
}

```

```

/*
6222/6252
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。

支持电源电压检测。
详见example\peripheral\adc
*/

```

2.1.3 ADC 的校准原理

这里的 ADC 校准原理并非指用固定采样点校准值对 ADC 整体采集曲线的校准。固定采样点校准值对 ADC 整体采集曲线的校准是事先采集不同电压点 ADC 值存储在 Flash 中，实际使用时用这些值对采集值进行校准。

这里的 ADC 校准原理是指使用 **attenuation** 模式时，不同通道间因到 ADC 模块距离不同所以寄生电阻大小不同，ADC 采集电压会按照设计的电阻比例放大引脚上的电压，此时引脚上的寄生电阻也会参与进来，进而影响 **attenuation** 模式采集精度。驱动中需要考虑到寄生电阻对衰减系数的影响，在驱动中消除寄生电阻对 **attenuation** 模式下精度的影响。

不同通道间的寄生电阻不同，和 ic 内部走线相关，同一型号相同通道的寄生电阻理论偏差不大。比如 PHY6222 各通道寄生电阻比例理论是一致的，PHY6252 各通道寄生电阻比例理论是一致的。衰减系数只关心电阻的比例，不关心电阻的绝对值。

衰减系数的测试计算，bypass 模式，输入一电压 V_{in1} ，输出 V_{out1} 。Attenuation 模式，输入一电压 V_{in2} ，输出 V_{out2} ，输入 V_{in3} ，输出 V_{out3} 。

那么衰减系数 $\lambda = V_{in2} * V_{out1} / V_{in1} / V_{out2}$ ， $\lambda = V_{in3} * V_{out1} / V_{in1} / V_{out3}$ ，另外 $V_{in2} / V_{out2} = V_{in3} / V_{out3}$ ， $V_{in3} = (V_{in2} / V_{out2}) * V_{out3}$ 。

比如 6222 单端模式下，bypass 输入 400mV，attenuation 输入 1602mV 各通道数据如下：

	bypass 输入 400mV	attenuation 输入 1602mv	衰减系数 λ	V_{in2}/V_{out2}
P11	2081.5	1844.5	4.519603	0.868
P23	2128.5	1978.5	4.308629	0.809
P24	2071.5	1946	4.263288	0.823
P14	2133.6	1906.2	4.482718	0.840
P15	2096.9	2008.9	4.180402	0.797
P20	2125	2090	4.072069	0.766

表 2: ADC 衰减系数参考

可以通过查看驱动判断衰减系数是否已经更新到驱动中。如果 `phy_adc.c(6220/6250)`，`adc.c(6222/6252)`中已经包含并且使用了 `adc_Lambda`，那么驱动使用的是考虑寄生电阻的衰减系数。否则，驱动使用的是未考虑各通道寄生电阻的衰减系数。

3 Voice 典型应用

3.1 Voice 概述

Voice 支持 DMIC(SAR-ADC)和 AMIC(L+R)。支持采样率有 8K、16K、32K、64K。主频 HCLK 需要大于等于 16Mhz。

- DMIC: 使用外部 DMIC，引脚灵活可配。将相应引脚复用为 `clk_1p28m`、`adcc_dmic_out`。PDM 采样率 1.28Mhz，L 通道率在上升沿，R 通道数据在下降沿。

- AMIC: 使能内部 PGA、ADC，管脚固定。P18(PGA+)、P20(PGA-)、P15(micphone bias)、P23(micphone bias reference voltage)，其中 P23 是可选。当使用 P23 作为麦克风参考电压时，纹波更小数据更好。

Voice memory buffer 地址空间为 0x4005800~ 0x4005BFF，共 1024 个 Byte(256 个 Word)。1 个 word 为一个采样点，其中高 16bit 为左通道原始数据，低 16bit 暂时未用。每采集 128 个采样点，触发一次 half 中断或 full 中断。

3.1.1 Voice 硬件注意事项

DMIC 硬件参考电路：

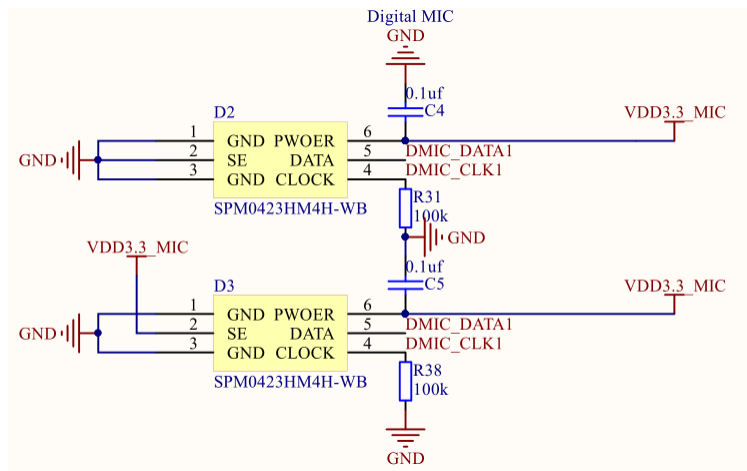


图 2：DMIC 参考电路

外部双端 AMIC 硬件参考电路：

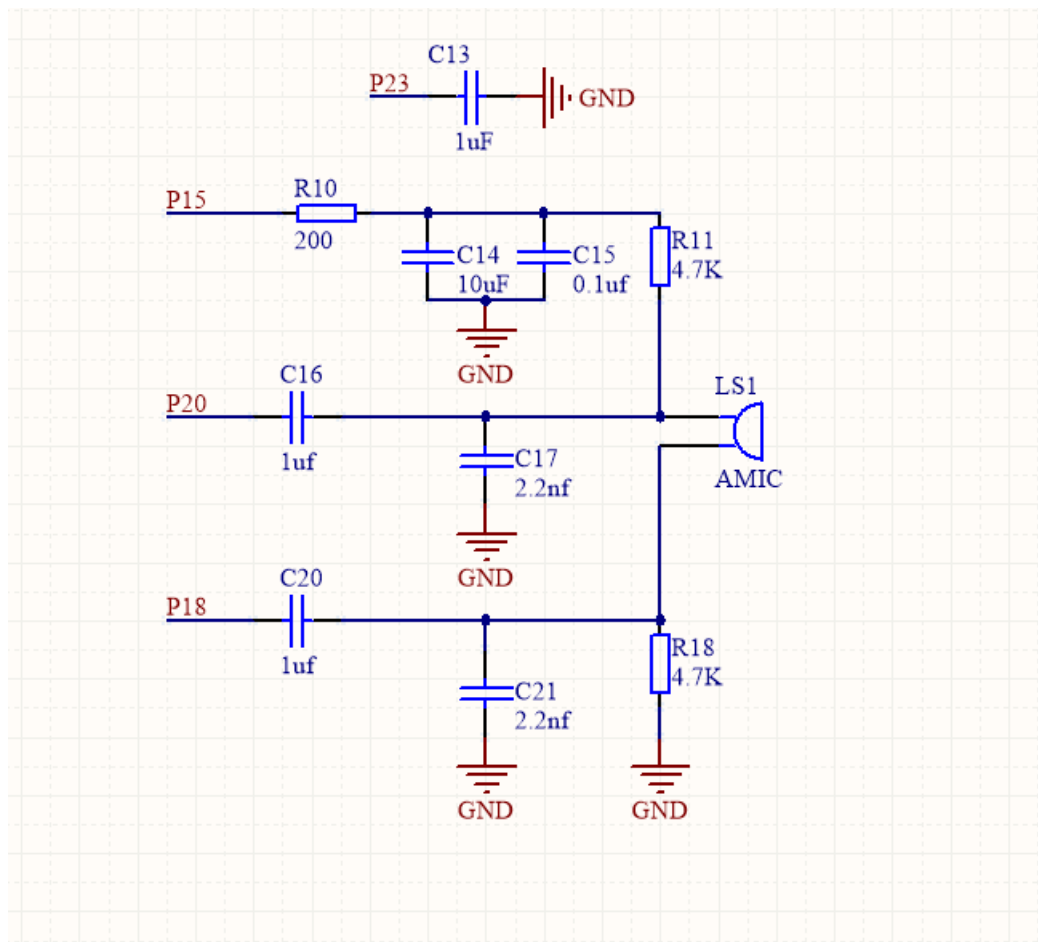


图 3：外部双端 AMIC 参考电路

外部单端 AMIC 硬件参考电路，P18 接电容到 GND 不能省略。

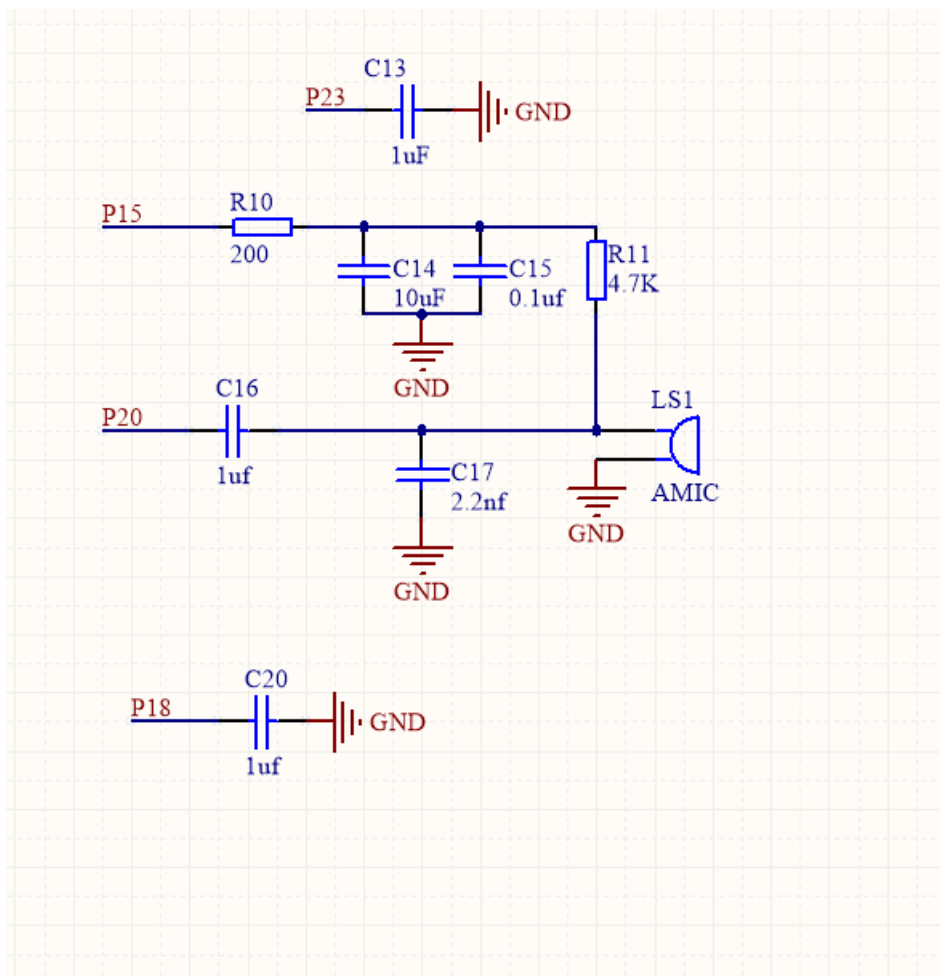


图 4：外部单端 AMIC 参考电路

3.1.2 Voice 软件注意事项

软件上使用 Voice，主要是相关参数配置和数据处理。

有结构体 voice_Cfg_t 可以灵活配置 Voice。数据处理则通过配置回调函数实现。

```
typedef struct _voice_Cfg_t{
    bool                voiceSelAmicDmic;
    gpio_pin_e          dmicDataPin;
    gpio_pin_e          dmicClkPin;
    uint8_t             amicGain;
    uint8_t             voiceGain;
    VOICE_ENCODE_t      voiceEncodeMode;
    VOICE_RATE_t        voiceRate;
    bool                voiceAutoMuteOnOff;
}voice_Cfg_t;
```

voiceSelAmicDmic: 选择AMIC还是DMIC，选择AMIC时dmicDataPin和dmicClkPin有效，否则无效。

amicGain: 用于AMIC PGA放大系数，共有两级参数可配。

voiceGain: 用于VOICE放大系数, [-20dB,+20dB], 每个step为0.5dB。
voiceEncodeMode: 数据编码方式
voiceRate: 采样率, 支持8K、16K、32K、64K。
voiceAutoMuteOnOff: 是否自动静音, 默认0, 支持。

volatile int voiceConfigStatus = hal_voice_config(cfg, voice_evt_handler_adpcm);
配置Voice的参数和回调函数

```
static void voice_evt_handler_adpcm(voice_Evt_t *pev)
{
    uint8_t leftbuf[2];
    uint8_t rightbuf[2];
    uint8_t left_right_chanle;
    uint32_t voiceSampleDual;
    int voiceSampleRight;
    int voiceSampleLeft;
    uint32_t i=0;

    left_right_chanle=SET_LEFT_VOICE;

    if(pev->type == HAL_VOICE_EVT_DATA)
    {
        for(i=0;i < pev->size;i++)
        {
            voiceSampleDual = pev->data[i];

            voiceSampleRight = (int16_t)(voiceSampleDual & 65535);
            voiceSampleLeft = (int16_t)((voiceSampleDual >> 16) & 65535);

            if(left_right_chanle==1)
            {
                leftbuf[0]= voiceSampleLeft;
                leftbuf[1]= voiceSampleLeft>>8;
                //FillBuffer(VoiceRaw_FiFO, leftbuf, 2);
                //data process
            }
            else
            {
                //rightbuf[0]= voiceSampleRight;
                //rightbuf[1]= voiceSampleRight>>8;
                //FillBuffer(VoiceRaw_FiFO, rightbuf, 2);
            }
        }
    }
}
```

```
}  
}
```