PRBMD02 Application Note

ADC application note

Disclaimer

1. Introduction

ADC, known as Analog-to-Digital converter, and in PRBMD02, ADC pins are multiplexed with GPIO, and listed as following:

GPIO	pin	ADC pin
P14	18	3
P15	19	4
P18	22	7
P20	23	9
P23	2	1
P24	3	2
P25	4	8

ADC:
There are two ADC modes available: Single mode and Differential mode.Pins P14, P15, P20, P23 and P24 can be configured as Single mode, and the following two pairs are also able to be configured as Differential mode:

P18 (22) - P25 (4)
P14 (18) - P24 (3)
P20 (23) - P15 (19)

Voice:
Built-in PGA, collect PCM raw data, P18, P20, P15, P23. DMIC and AMIC are supported, using DMIC, the pins can be configured with flexible FMUX. When using AMIC, only P18(PGA+), P20(PGA-), P15(micphone bias), P23(micphone bias reference voltage), among which P23 is optional.

2. ADC application

2.1. Brief description of ADC

The ADC reference voltage is provided inside the chip, and the ADC reference voltage is 0.8V.

The chip has a built-in 12bit SAR ADC, and a total of 8 input ports are available. Pins can be divided into PGA, single-ended input, differential input according to their purpose

- **PGA**: P18(PGA+), P20(PGA-), the typical application is connected to AMIC to collect VOICE.
- **Single-ended input**: P14, P15, P20, P23, P24, typical application ADC single-ended acquisition.
- **Differential input**: P18(+)P25(-),P14(+)P24(-), P20(+)P15(-), typical application ADC differential acquisition.

Hardware supports single-ended and differential modes:

- **Single-Ended Mode**: Measure the voltage between the pin and GND.
- **Differential Mode**: Measure the voltage between two pins.

The clock of ADC comes from HCLK, the ADC clock is 1Mhz.

Hardware supports manual mode and automatic scan mode:

Manual Mode: Supports only one single-ended channel or a group of differential acquisition channels at a time, enabling it to convert a particular input method to single-ended or differential input.

Auto Mode: Automatically scans all enabled multiple single-ended channels and stores the converted data in the corresponding memory locations. SDK ADC works in automatic mode. One ADC sampling time consists of ADC sampling time and ADC conversion time, both of which can be matched. The former is 2T and 3T, the latter is 3T and 2T, and T is the period of the ADC clock.

Hardware supports bypass mode and attenuation mode:

- **Bypass mode**: The pin input voltage directly enters the ADC inside the chip, and the range is 0~0.8V at this time.
- **Attenuation mode**: The pin input voltage directly enters the ADC after passing through the on-chip voltage divider resistor. The voltage divider resistance ratio is about 4:1, and the resistance values of the voltage divider resistors are one 14.15K and one 4.72K. At this time, the theoretical range is 0~3.2V.

ADC theoretical accuracy:

- **Bypass mode**: single-ended theoretical accuracy 0.8/4096V, about 0.2mV0. When using bypass mode, an external divider resistor is used, and when calculating the voltage with an external divider resistor, pay attention to the error of the resistor.
- **Attenuation mode**: The absolute error of the internal resistance of the chip is about +-15%, and the relative error is about +-1%. Absolute error does not affect ADC accuracy, relative error affects ADC accuracy.

The hardware supports chip power supply voltage measurement, and the configured analog pin has been processed inside the chip, so this pin needs to be kept isolated.

Attention: Do not use internal divider resistors and external divider resistors at the same time, that is, if you use attenuation mode, do not use external divider resistors.

2.1.1.ADC hardware consideration

When the voltage to be collected is small, such as less than 0.8V, that is, within the bypass mode range, you can directly use the bypass mode. Note that the acquisition pin needs to be connected to a filter capacitor.

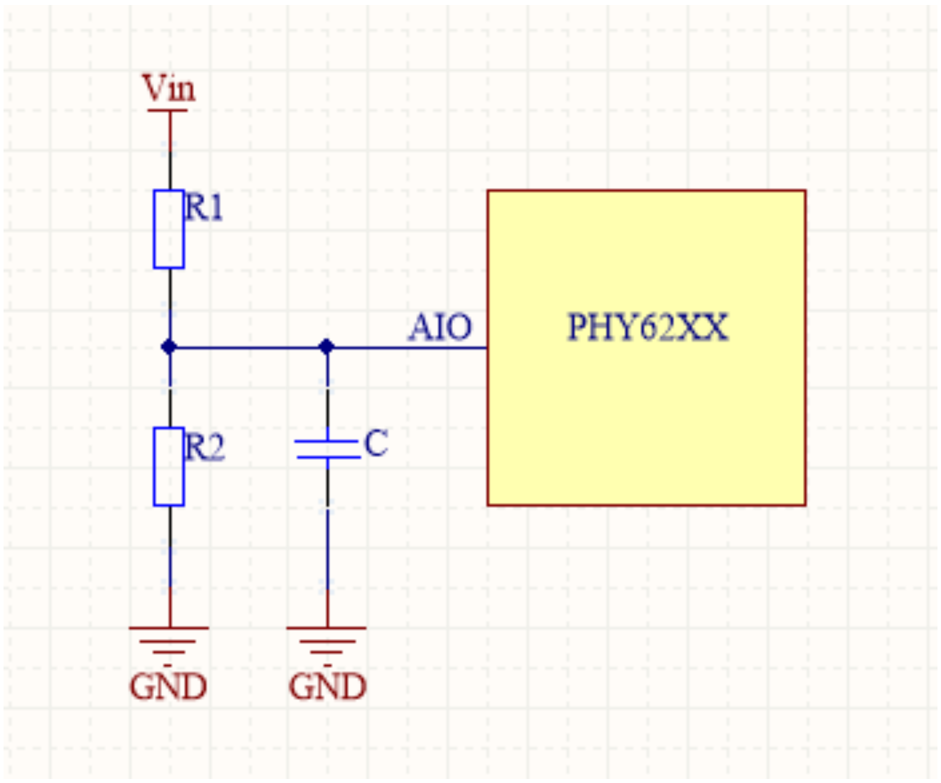
When the voltage to be collected is slightly larger, such as greater than 0.8V but less than 3.2V, you can use the attenuation mode or the bypass mode after adding a resistor to divide the voltage. The ADC accuracy depends on the relative accuracy of the resistor. In the attenuation mode, the internal relative accuracy is +-1%, and the external resistor can also choose +-1% resistor.

When the voltage to be collected is large, such as greater than 3.2V, that is, it exceeds the range of the attenuation mode, and the bypass mode after voltage division by an external resistor must be used. Note that the acquisition pin needs to be connected to a filter capacitor.

When using an external resistor to divide the voltage, the resistor and capacitor need to meet certain constraints, as shown in the figure below:

```
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;
}
```

$$\text{Gain} = \frac{\text{R2}}{\text{R1+R2}}$$



- The mode selects bypass, and the test range is [0V, 0.8V].
- The detection voltage V_{AIO} needs to be less than 0.8V.

The calculation is as following:

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

- Vin Detection frequency:

$$f_{in} < \frac{1}{2\pi\frac{R1R2}{R1+R2}C}$$

- Gain =
- Vin drive enable R1//R2//C

2.1.2.ADC software consideration

2.1.3.ADC calibration

The ADC calibration principle here does not refer to the calibration of the ADC overall acquisition curve with the fixed sampling point calibration value. The calibration of the fixed sampling point calibration value to the ADC overall acquisition curve is to collect the ADC values at different voltage points in advance and store them in the Flash, and use these values to calibrate the acquired values in actual use.

The ADC calibration principle here means that when using the attenuation mode, the parasitic resistance is different between different channels due to different distances from the ADC module. The ADC acquisition voltage will amplify the voltage on the pin according to the designed resistance ratio. At this time, the parasitic resistance on the pin is The resistance will also be involved, which will affect the acquisition accuracy of the attenuation mode. The influence of parasitic resistance on the attenuation coefficient needs to be considered in the drive, and the influence of the parasitic resistance on the precision in the attenuation mode should be eliminated in the drive.

The parasitic resistance between different channels is different, which is related to the internal wiring of the IC. The theoretical deviation of the parasitic resistance of the same channel of the same model is not large. For example, the parasitic resistance ratio theory of each channel of PHY6222 is consistent, and the parasitic resistance ratio theory of each channel of PHY6252 is consistent. The attenuation coefficient only cares about the ratio of the resistance, not the absolute value of the resistance.

Test calculation of attenuation coefficient, bypass mode, input a voltage Vin1, output Vout1. Attenuation mode, input a voltage Vin2, output Vout2, input Vin3, output Vout3.

Then the attenuation coefficient λ=Vin2*Vout1/Vin1/Vout2, λ=Vin3*Vout1/Vin1/Vout3, and Vin2/Vout2 = Vin3/Vout3, Vin3=(Vin2/Vout2)*Vout3.

For example, in PRBMD02 single-ended mode, the bypass input is 400mV, and the attenuation input is 1602mV. The data of each channel are as follows:

You can judge whether the attenuation coefficient has been updated to the driver by checking the driver. If phy_adc.c(6220/6250), adc.c(6222/6252) have included and used adc_Lambda, then the drive uses the attenuation factor considering parasitic resistance. Otherwise, the drive uses an attenuation factor that does not consider the parasitic resistance of each channel.

3. Voice typical application

3.1. Brief description on Voice

Voice supports DMIC(SAR-ADC) and AMIC(L+R). Supported sampling rates are 8K, 16K, 32K, 64K.

- DMIC: Using an external DMIC, the pins are flexible and configurable. Multiplex the corresponding pins as clk_1p28m, adcc_dmic_out. The PDM sampling rate is 1.28Mhz, the L channel rate is on the rising edge, and the R channel data is on the falling edge.

```
/*
6220/6250
The structure adc_Cfg_t can configure parameters such as channel and mode of adc.
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;
```

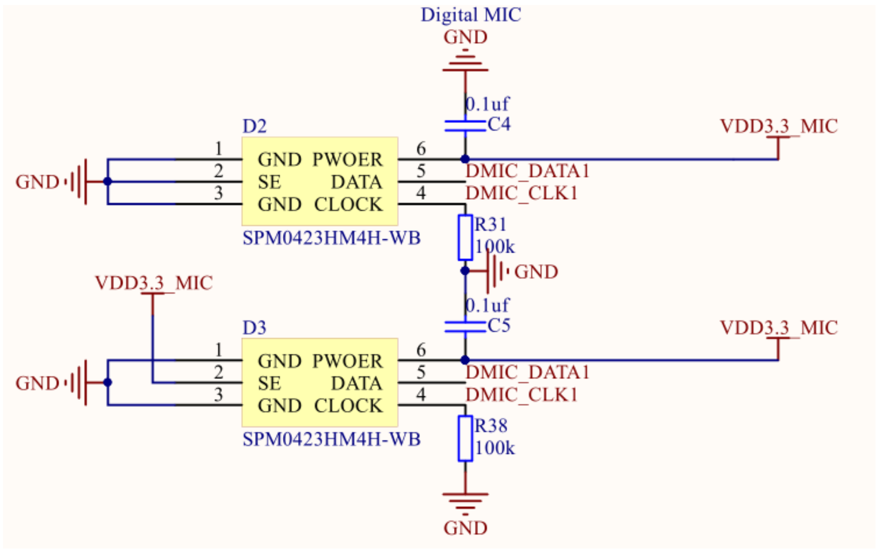
```
channel: used to configure the channel, single-ended mode supports multiple channels,
differential mode supports a group of channels. Each bit corresponds to a single-ended channel
or a group of differential channels.

is_continue_mode: adc working mode, always open or intermittently open. TRUE is always on,
FALSE is closed once for collection, and needs to be reopened next time.

is_differential_mode: Whether it is differential mode, all 0 single-ended mode, otherwise
differential mode

is_high_resolution: bypass mode and attenuation mode, each bit represents a single-ended
channel or a group of differential channels. 0 represents attenuation mode, 1 represents bypass
mode.

Supports power supply voltage detection.

More details on SDK: example\peripheral\adc

*/
```

- AMIC: Enable the internal PGA, ADC, the pin is fixed. P18(PGA+), P20(PGA-), P15(micphonebias), P23(micphone bias reference voltage), among which P23 is optional. When using P23 as the microphone reference voltage, the smaller the ripple, the better the data.

The address space of the voice memory buffer is 0x4005800~ 0x4005BFF, with a total of 1024 Bytes (256 Words). 1 word is a sampling point, of which the upper 16 bits are the original data of the left channel, and the lower 16 bits are temporarily unused. Every 128 samples are collected, a half interrupt or full interrupt is triggered.

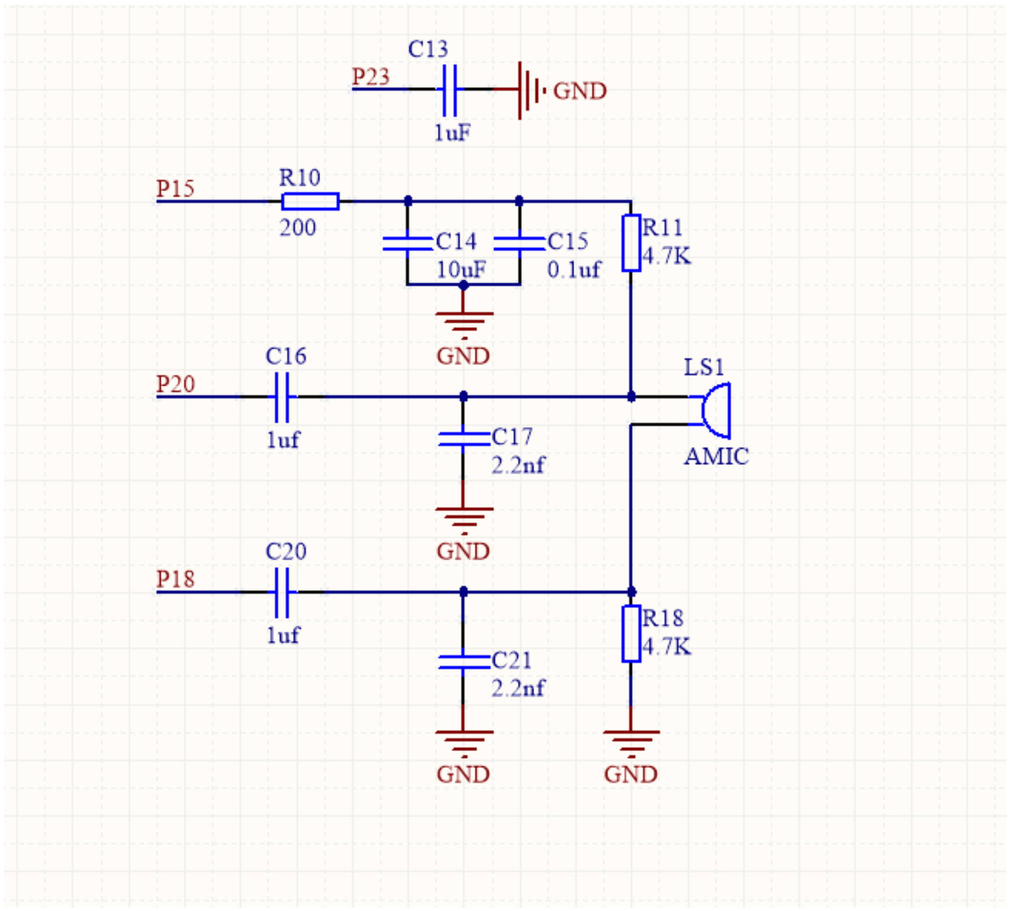
3.1.1. Voice hardware consideration

DMIC reference circuit:

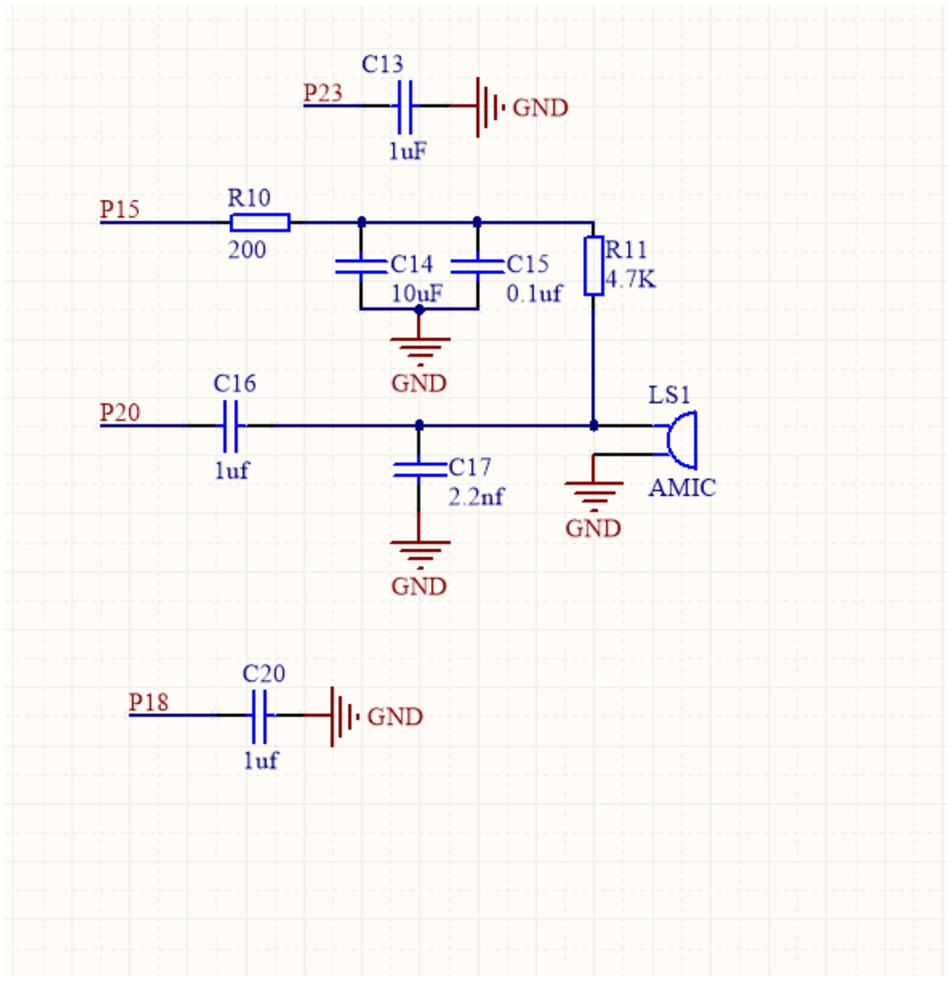


External double-ended AMIC hardware reference circuit:

pin	bypass input 400mV	Attenuation input 1602mV	Attenuation λ	Vin2/Vout2
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



External single-ended AMIC hardware reference circuit, the connection of P18 to the capacitor to GND cannot be omitted:



3.1.2. Voice software consideration

Voice is used in the software, mainly for related parameter configuration and data processing.

There is a structure voice_Cfg_t to configure Voice flexibly. Data processing is achieved by configuring callback functions.

```
typedef struct _voice_Cfg_t{
                bool                        voiceSelAmicDmic;
                gpio_pin_e                  dmicDataPin;
                gpio_pin_e                  dmicClkPin;
                unit8_t                     amicGain;
                uint9_t                     voiceGain;
                VOICE_ENCODE_t              voiceEncodeMode;
                VOICE_RATE_t                voiceRate;
                bool                        voiceAutoMuteOnOff;
                }voice_Cfg_t;
```

voiceSelAmicDmic: Choose AMIC or DMIC. When AMIC is selected, dmicDataPin and dmicClkPin are valid, otherwise they are invalid. amicGain: used for AMIC PGA amplification factor, there are two levels of parameters that can be configured.
voiceGain: used for VOICE amplification factor, [-20dB, +20dB], each step is 0.5dB.
voiceEncodeMode: data encoding mode
voiceRate: Sample rate, supports 8K, 16K, 32K, 64K.
voiceAutoMuteOnOff: Whether to mute automatically, default 0, support.

```
volatile int voiceConfigStatus = hal_voice_config(cfg, voice_evt_handler_adpcm);
Configure Voice parameters and callback functions
```

```
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)(voiceSampleDual & 65535);
             voiceSampleLeft = (int16)((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);

        }

    }
  }
}
```