PRBMD0x Application Note

AD convertor application note

Disclaimer

Liability Disclaimer
K-Solution Consulting Co. Ltd reserves the right to make changes without further notice to the product to improve reliability, function or design. K-Solution Consulting Co. Ltd does not assume any liability arising out of the application or use of any product or circuits described herein.

Life Support Applications
K-Solution Consulting Co. Ltd's products are not designed for use in life support appliances, devices, or systems where malfunction of these products can reasonably be expected to result in personal injury. K-Solution Consulting Co. Ltd customers using or selling these products for use in such applications do so at their own risk and agree to fully indemnify K-Solution Consulting Co. Ltd for any damages resulting from such improper use or sale.

1. Introduction

This document describes the ADC of PRBMD0x and its application.

PRBMD0x consists up to 9 ADC channel: one for PGA; one for Temperature sensor; one for voice and six for normal purposes. This document will describe the application of normal ADC only.

ADC channels are multiplexed with GPIO, and the mapping is as following:

Channel no	Channel Feature	GPIO	Comments
ADC_CH0			PGA
ADC_CH1			Temp sensor
ADC_CH2	1P/1DIFF	P12	Normal ADC
ADC_CH3	1N	P11	
ADC_CH4	2P/2DIFF	P14	
ADC_CH5	2N	P13	
ADC_CH6	3P/3DIFF	P20	
ADC_CH7	3N	P15	
ADC_Voice			Voice

Table 1 : ADC and GPIO mapping

i. Mode selection

ADC_CH2 to ADC_7 provides two mode of operations, user can select either of one:

Single end mode: each channel works independently, and each channel sample the voltage at the pin.

Differential mode: the channels are to be used as paired, CH2-CH3; CH4-CH5 and CH6-CH7. Differential voltage on CH2 (vs CH3), CH4 (vs CH5) and CH6 (vs CH7) will be sampled

There are two sampling mode available for these 6 channels:

Bypass mode: the sampling voltage is from 0-1VDC

Attenuation mode: the sampling voltage is from 0-3.3VDC (or VCC) **

ii. Sampling accuracy

Accuracy varies under different sampling mode, as illustrated as below:

	Single end mode(unit:V)	Differential mode(unit:V)
Bypass	ADC_code/4096	ADC_code/2048-1
Attenuation	(ADC_code/4096)*4	(ADC_code/2048-1)*4

Table 2: Normal ADC accuracy under different mode

iii. Buffering

Buffer length of each ADC channel is 32 word, which can hold two ADC results.

iv. Sampling rate

The sampling rate is 80K, 160K, 320K, where the default rate is 320K.

v. Operation

General ADC can can be operated by interrupt or polling.

- **Interrupt**

Each channel share the same interrupt: CM0(29)

Each General ADC channel can be mask and clear independently.

Interrupt is triggered when buffer is full.

The flow of handling ADC interrupt is as following:

1. system initial
2. ADC initial
3. ADC enable
4. irq enable
5. enable ADC interrupt
6. wait interrupt
7. Collect Data
8. calculate ADC value
9. mask interrupt
10. clear interrupt
11. disable ADC

- **Polling**

The flow of polling ADC is as following:

1.system initial

2.ADC initial

3.ADC enable

4.wait a few us

5.Collect Data

6.calculate ADC value

7.disable ADC

2. API

ADC driver provided in API supports async AD sampling feature, user can get the sampling result by calling functions.

i. ADC driver definition

a. ADC_CH_t

ADC channel definition。

ADC_CH0	Not support for this channel
ADC_CH1	Not support for this channel
ADC_CH2 ADC_CH1N_P11	Single end mode, operate independently Differential mode，operate with ADC_CH1P_P12
ADC_CH3 ADC_CH1P_P12	Single end mode, operate independently 差分模式下，和ADC_CH1N_P11组合使用。
ADC_CH4 ADC_CH2N_P13	Single end mode, operate independently Differential mode，operate with ADC_CH2P_P14.
ADC_CH5 ADC_CH2P_P14	Single end mode, operate independently Differential mode，operate with ADC_CH2N_P13.
ADC_CH6 ADC_CH3N_P15	Single end mode, operate independently Differential mode，operate with ADC_CH3P_P20.
ADC_CH7 ADC_CH3P_P20	Single end mode, operate independently Differential mode，operate with ADC_CH3N_P15.
ADC_CH_VOICE	voice channel, for sampling from microphone.

b. ADC event

when there is an ADC event, calling ADC driver function will return:

HAL_ADC_EVT_DATA	ADC sampled data. If the sampling data is ready, it will call the registered ADC callback function to send the event.
HAL_ADC_EVT_FAIL	ADC sampling fail

c. ADC_CLOCK_SEL_t

ADC sampling rate

HAL_ADC_CLOCK_80K	sampling rate = 80K
HAL_ADC_CLOCK_160K	sampling rate = 160K
HAL_ADC_CLOCK_320K	sampling rate = 320K

ii. Data structure

a. ADC_Cfg_t

ADC configuration parameter。

uint8_t	channel	configure ADC channel，bit2~bit7 corresponding to P11~P15、P20.
bool	is_continue_mode	Continuous sampling If true，ADC will continuous sampling automatically。 If false，a single ADC and controlled by software
uint8_t	is_differential_mode	Differential sampling. bit7, bit5, bit3 represents differential pairs [P20,P15], [P14,P13] and [P12,P11]. the setting of "channel" must be consistence with "s_differential_mode".
uint8_t	is_high_resolution	ADC is "bypass mode" if true，the sampling range is 0V~1V。 it is "attenuation mode" if false，sampling range is 0V~3.3V. supports bit2~bit7, and must consistence with "channel"

b. adc_Evt_t

ADC driver event data structure.

int	type	ADC event type. HAL_ADC_EVT_DATA：sampling success, valid data HAL_ADC_EVT_FAIL：sampling fail, invalid data
adc_CH_t	ch	ADC channel。 channel range, adc_CH_t。
uint16_t*	data	entrance pointer for ADC sampling.
uint8_t	size	ADC sampling data size,

c. adc_Hdl_t

ADC return function type。

typedef void (*adc_Hdl_t)(adc_Evt_t* pev)

d. adc_Ctx_t

ADC block configuration。

bool	enable	ADC模块使能标志。???
uint8_t	all_channel	enabled ADC channel, supports bit2~bit7。
adc_Hdl_t	evt_handler	Return function of sampled ADC.

iii. API

a. void hal_adc_init(void)

ADC initialisation, after initialised ，parameter of ADC must be configured properly before using it, otherwise un-predicted data may be obtained.

- parameter: none
- return value: none

b. int hal_adc_config_channel (adc_CH_t channel, adc_Cfg_t cfg, adc_Hdl_t evt_handler)

configure ADC sampling channel。

- parameter

type	parameter	description
adc_Cfg_t	cfg	ADC configuration message。
adc_Hdl_t	evt_handler	event return function

- return value

PPlus_SUCCESS	success
other parameter	refers to <error.h>

c. int hal_adc_clock_config(adc_CLOCK_SEL_t clk);

setting ADC sampling range, configure before use.

- parameter

type	parameter	description
adc_CLOCK_SEL_t	clk	ADC sampling range: 80K, 160K, 320K(default)

- return value

PPlus_SUCCESS	success
other value	refers to <error.h>

4. int hal_adc_start(void)

start sampling

- parameter: none
- return value:

PPlus_SUCCESS	sampling success
other value	refers to <error.h>

5. int hal_adc_stop(void)

stop sampling

- parameter: none
- return value:

PPlus_SUCCESS	success
Other value	refers to <error.h>

6. void __attribute__((weak)) hal_ADC_IRQHandler(void)

ADC halt function

- parameter: none
- return value: none

7. float hal_adc_value_cal(adc_CH_t ch,uint16_t* buf, uint8_t size, uint8_t high_resol, uint8_t diff_mode)

ADC value, return value is average sampled voltage at buffer, it is a floating point format.

- parameter

type	parameter	description
adc_CH_t	ch	ADC channel.
uint16_t*	buf	pointer to ADC sampled data.
uint8_t	size	size of ADC sampled data.
uint8_t	high_resol	If bypass mode, supports bit2~bit7
uint8_t	diff_mode	if differential mode, supports bit7、bit5、bit3

- return value

float	sampled voltage

3. Software application

testing hardware: PRBMDD0x evaluation board

testing software: peripheral\adc

i. sample continuously and single

Description
"is_continue-mode" in "adc_Cfg_t" is for configuring the ADC sampling is continuously or a single.
When continuous sampling is enable, ADC will repetitive sample the voltage after initialisation
ADC stops after a single sampling when in single sample. ADC need to re-activate for another sample.

```
//P14、P15 setting: continuous (single)、single end、bypass mode。
adc_Cfg_t adc_cfg = {
    .channel = ADC_BIT(ADC_CH2P_P14)|ADC_BIT(ADC_CH3N_P15),
    .is_continue_mode = TRUE,//(FALSE)
    .is_differential_mode = 0x00,
    .is_high_resolution = 0xff,
};
```

```
//single sampling, ADC sample every 500ms
static void adc_evt(adc_Evt_t* pev)
{
......
        if(adc_cfg.is_continue_mode == FALSE)
        {
            osal_start_timerEx( adcDemo_TaskID, adcMeasureTask_EVT,500);
        }
 ......
}
```

ii. Bypass mode and Attenuation mode

description
"is_high_resolution" in "adc_Cfg_t" is for configuration of ADC range.
the variable of bit2~bit7 corresponding to P11~P15、P20。
ADC channel is bypass mode if the corresponding bit is 1, the voltage range will be 0-1V
If the corresponding bit is 0, the ADC is at attenuation mode，sampling range is 0V~Vcc。

```
//P14、P15工作模式: 非连续采集模式、单端、bypass mode(attenuation mode)。
adc_Cfg_t adc_cfg = {
    .channel = ADC_BIT(ADC_CH2P_P14)|ADC_BIT(ADC_CH3N_P15),
    .is_continue_mode = FALSE,
    .is_differential_mode = 0x00,
    .is_high_resolution = 0xFF//(0x00)
};
```

iii. battery voltage detection

Description

Detect the battery level to the module. Since the core chip has 测量芯片电源电压时，芯片内部已经将AVD33和ADC脚进行了连接，所以芯片外部需要保证测量芯片电源的ADC引脚悬空。

“attenuation mode” must be selected for detecting battery level。

Below is the configuration for the detection pin:

```
//Configure P14, P15, P20，where P20 sample the battery level
adc_Cfg_t adc_cfg = {
    .channel = ADC_BIT(ADC_CH2P_P14)|ADC_BIT(ADC_CH3N_P15)|ADC_BIT(ADC_CH3P_P20),
    .is_continue_mode = FALSE,
    .is_differential_mode = 0x00,
    .is_high_resolution = 0x7f,//bit7 corresponding to P20，attenuation mode is needed
};
```

```
// Configure ADC pin for battery level
static void adcMeasureTask( void )
{
    int ret;
    bool batt_mode = TRUE;
    uint8_t batt_ch = ADC_CH3P_P20;
    GPIO_Pin_e pin;
……
}
```

iv. Differential mode

Description

Pair of ADC pin is needed for differential mode.

Parameter for each differential channel are ADC_BIT(ADC_CH3DIFF)、ADC_BIT(ADC_CH2DIFF)、ADC_BIT(ADC_CH1DIFF), representing the differential voltage of P20-P15, P14-P13, P12-P11 respectively

```
// ADC_CH3DIFF, detect the voltage on P20 compare with P15
adc_Cfg_t adc_cfg = {
    .channel = ADC_BIT(ADC_CH3DIFF),
    .is_continue_mode = FALSE,
    .is_differential_mode = ADC_BIT(CH3DIFF),
    .is_high_resolution = 0xff,
};
```

v. Bluetooth and ADC sampling

During Bluetooth advertisement and connection, the ，the Instantaneous transmit power may affect the reference voltage of ADC and the sampled data may not be accurate.
◦ It is suggested to implement ADC sample at least few milli-second after advertising or connection

reference code：

```
void SimpleBLEPeripheral_Init( uint8 task_id )
{
HCI_PPLUS_AdvEventDoneNoticeCmd(simpleBLEPeripheral_TaskID, ADC_BROADCAST_EVT);
}
static void peripheralStateNotificationCB( gaprole_States_t newState )
{
      case GAPROLE_CONNECTED:
         HCI_PPLUS_ConnEventDoneNoticeCmd(simpleBLEPeripheral_TaskID, ADC_CONNECT_EVT);
       break;
}

uint16 SimpleBLEPeripheral_ProcessEvent( uint8 task_id, uint16 events )
{
if ( events & ADC_BROADCAST_EVT ){
//start adc sample later,uncontinue mode
osal_start_timerEx( adcDemo_TaskID, 0x0080,5);
return ( events ^ ADC_BROADCAST_EVT );
}
if ( events & ADC_CONNECT_EVT ){
//start adc sample later,uncontinue mode
osal_start_timerEx( adcDemo_TaskID, 0x0080,5);
   return ( events ^ ADC_CONNECT_EVT );
}
}
```

vi. upgrade ADC driver

Previous version ADC driver didn't support multiple channel sample, here is the comparison of using old and latest version ADC driver.

adc.h and adc.c must be used together，new driver version in adc.h should have no affect.。

New driver change the adc configuration parameter from local to global variable, and some variable bit corresponding to ADC channels.

Sampling a voltage (non-battery). Left side is old version, Right side is latest version

```
static void adcMeasureTask( void )
{
 int ret;
   bool batt_mode = FALSE;
   adc_CH_t channel = ADC_CH3P_P20;
   GPIO_Pin_e pin = s_pinmap[channel];
 adc_Cfg_t cfg = {
    .is_continue_mode = FALSE,
    .is_differential_mode = FALSE,
    .is_high_resolution = TRUE,
    .is_auto_mode = FALSE,
};
//other code
}
static void adc_evt(adc_Evt_t* pev)
{
if(pev->type == HAL_ADC_EVT_DATA)
{
// parameter must be consistence with parameter cfg
float value = hal_adc_value_cal(pev->ch,pev->data, pev->size, TRUE ,FALSE);
        LOG("adc %d\n",(int)(value*1000));
 }
}
```

```
adc_Cfg_t adc_cfg = {
        .channel = ADC_BIT(ADC_CH3P_P20),
        .is_continue_mode = FALSE,
        .is_differential_mode = 0x00,
        .is_high_resolution = 0xff,
};
static void adcMeasureTask( void )
{
        int ret;
        bool batt_mode = FALSE;
//other code
}
```

Sampling a battery voltage. Left side is old version, Right side is latest version

```
static void adcMeasureTask( void )
{
 int ret;
   bool batt_mode = TRUE;
   adc_CH_t channel = ADC_CH3P_P20;
   GPIO_Pin_e pin = s_pinmap[channel];
 adc_Cfg_t cfg = {
    .is_continue_mode = FALSE,
    .is_differential_mode = FALSE,
    .is_high_resolution = FALSE,
    .is_auto_mode = FALSE,
};
//other code
}
static void adc_evt(adc_Evt_t* pev)
{
if(pev->type == HAL_ADC_EVT_DATA)
{
//parameter must be consistence with parameter cfg
float value = hal_adc_value_cal(pev->ch,pev->data, pev->size, FALSE, FALSE);   LOG("batt_measure_evt %d\n", (int)(value*1000));
 }
}
```

```
adc_Cfg_t adc_cfg = {
        .channel = ADC_BIT(ADC_CH3P_P20),
        .is_continue_mode = FALSE,
        .is_differential_mode = 0x00,
        .is_high_resolution = 0x00,
};
static void adcMeasureTask( void )
{
        int ret;
        bool batt_mode = TRUE;
        uint8_t batt_ch = ADC_CH3P_P20;
//other code
}
```

Differential voltage sampling. Left side is old version, Right side is latest version

```
static void adcMeasureTask( void )
{
  int ret;
    bool batt_mode = FALSE;
    adc_CH_t channel = ADC_CH3DIFF;
    GPIO_Pin_e pin = s_pinmap[channel];
  adc_Cfg_t cfg = {
      .is_continue_mode = FALSE,
      .is_differential_mode = TRUE,
      .is_high_resolution = TRUE,
      .is_auto_mode = FALSE,
};
//other code
}
static void adc_evt(adc_Evt_t* pev)
{
  if(pev->type == HAL_ADC_EVT_DATA)
{
//parameter must be consistence with parameter cfg
float value = hal_adc_value_cal(pev->ch,pev->data, pev->size, TRUE, TRUE);
        LOG("adc %d\n",(int)(value*1000));
  }
}
```

```
adc_Cfg_t adc_cfg = {
        .channel = ADC_BIT(ADC_CH3DIFF),
        .is_continue_mode = FALSE,
        .is_differential_mode = ADC_BIT(ADC_CH3DIFF),
        .is_high_resolution = 0xff,
};
static void adcMeasureTask( void )
{
  int ret;
    bool batt_mode = FALSE;
}
```

4. External circuit

When designing external circuit for ADC, the voltage range must be in the range.

i. Sample voltage is lower than 3.3VDC or VDD

ADC can direct connected with the point of detection

ii. Sample voltage is higher than 3.3V or VDD

A voltage divider is needed for voltage is higher than 3.3VDC or VDD

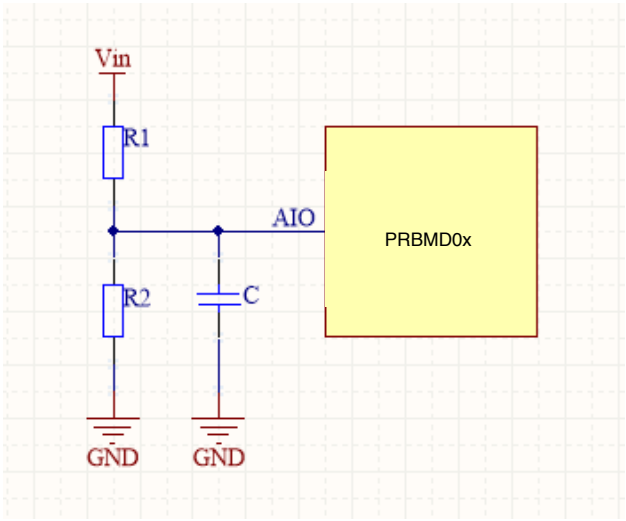

Diagram 3: a suggested circuit for voltage over VDD

- selecte bypass，voltage range is【0V，1V】。
- the detected voltage V_{AIO} must lower than 1V 。

the calculation is as follow：

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

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

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

iii. Battery voltage detection

When battery voltage detection feature is enable, the ADC pin left open as it will be internal connected to VCC.

** PRBMD00 consists a DC/DC, hence the VDD to the core chip is always 3.3V; PRBMD01 has no DC/DC, hence the VDD may vary base on the voltage applied

5. Reference

This document is base on Application note PHY62XX_ADC_Application_Note v0.5 20190806 from PHYPlus semiconductors.