

PHY62XX

ADC 应用指南

Version 1.0

Phyplus Inc.

2019/08/02

版本控制信息

版本/状态	作者	参与者	起止日期	备注
V0.2	付晓亮		06/13/2018	文档初稿
V0.3	付晓亮		04/08/2019	修订版
V0.4	付晓亮		06/28/2019	修订版，支持多通道采集
V1.0	付晓亮		08/02/2019	完善 ADC 示例，添加 ADC 移植 注意事项，参考 3.6

目录

1	简介	1
1.1	模式选择	1
1.2	采集精度	2
1.3	Buffer 缓冲机制	2
1.4	采集速率	2
1.5	工作方式	2
1.5.1	中断方式	2
1.5.2	轮询方式	3
2	API	3
2.1	枚举&宏	3
2.1.1	adc_CH_t	3
2.1.2	ADC 事件	4
2.1.3	adc_CLOCK_SEL_t	4
2.2	数据结构	5
2.2.1	adc_Cfg_t	5
2.2.2	adc_Evt_t	6
2.2.3	adc_Hdl_t	6
2.2.4	adc_Ctx_t	6
2.3	API	6
2.3.1	void hal_adc_init(void)	6
2.3.2	int hal_adc_config_channel (adc_CH_t channel, adc_Cfg_t cfg, adc_Hdl_t evt_handler)	7
2.3.3	int hal_adc_clock_config(adc_CLOCK_SEL_t clk);	7
2.3.4	int hal_adc_start(void)	8
2.3.5	int hal_adc_stop(void)	8
2.3.6	void __attribute__((weak)) hal_ADC_IRQHandler(void)	9
2.3.7	float hal_adc_value_cal(adc_CH_t ch,uint16_t* buf, uint8_t size, uint8_t high_resol, uint8_t diff_mode)	9
3	软件应用	10
3.1	连续采集与非连续采集	10
3.2	Bypass mode 和 Attenuation mode	11
3.3	电源电压检测	11
3.4	差分模式	12
3.5	蓝牙广播和连接下的 ADC 采集	13
3.6	ADC 移植	14
4	外围电路	17
4.1	被测电压小于等于芯片工作电压 AVDD33	17
4.2	被测电压大于芯片工作电压 AVDD33	17
4.3	电源电压检测	18

图表

图表 1: ADC 通道和 GPIO 对应关系	1
图表 2: NORMAL ADC 不同模式下采集精度	2
图表 3: 当被测点电压大于 AVDD33 时外围需要分压电路	17

1 简介

本文档介绍了 PHY62XX ADC 模块的原理和使用方法。

PHY62XX ADC 一共有 9 个 ADC 通道：1 个 PGA，1 个 Temp Sensor，6 个 Normal ADC，1 个 Voice。

本文档主要介绍 6 个 Normal ADC 的使用方法。

ADC 通道和 GPIO 对应关系如下图：

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

图表 1：ADC 通道和 GPIO 对应关系

1.1 模式选择

6 个 Normal ADC 通道有两个工作模式供选择：

- 单端模式：ADC_CH2~ADC_CH7 每个通道都可以独立工作，采集其引脚上的单端电压。
- 差分模式：ADC_CH2~ADC_CH3、ADC_CH4~ADC_CH5、ADC_CH6~ADC_CH7 要成对出现，采集的电压是 ADC_CH2、ADC_CH4、ADC_CH6 上相对于 ADC_CH3、ADC_CH5、ADC_CH7 上的差分电压。

6 个 Normal ADC 通道有两个测试量程供选择：

- bypass mode：测试电压范围为【0V，1V】。
- attenuation mode：测试电压范围为【0V，AVDD33】。

1.2 采集精度

不同模式下，Normal ADC 的采集精度也不相同，详见下表：

	单端模式(unit:V)	差分模式(unit:V)
Bypass	ADC_code/4096	ADC_code/2048-1
Attenuation	(ADC_code/4096)*4	(ADC_code/2048-1)*4

图表 2: Normal ADC 不同模式下采集精度

1.3 Buffer 缓冲机制

Normal ADC 是 12bit 长度的模数转换器，硬件为每个 ADC 通路提供了 32 个 word 长度的连续 buffer，用来缓存 ADC 采集结果，每一个 word 保存 2 次 ADC 转换结果。

1.4 采集速率

Normal ADC 支持采样速率有：80K，160K，320K，默认速率为 320K。

1.5 工作方式

Normal ADC 有两种工作方式：中断方式和轮询方式。

1.5.1 中断方式

9 个 ADC 通道共用一个中断入口，中断号为：CM0（29）。

每个 Normal ADC 通道的中断都能单独 mask 和 clear。

中断触发条件为 buffer 满，即数据填满整块 memory 会触发中断。

中断方式软件处理流程如下，很多流程已经被封装到了 API 中，直接调用使用即可。

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

1.5.2 轮询方式

轮询方式下软件处理流程：

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 驱动提供异步 AD 采集功能，采集完成之后通过回调函数回传 ADC 采集结果。

2.1 枚举&宏

2.1.1 adc_CH_t

ADC 物理通道。

ADC_CH0	暂不支持此通道。
ADC_CH1	暂不支持此通道。
ADC_CH2	单端模式下，独立工作。
ADC_CH1N_P11	差分模式下，和 ADC_CH1P_P12 组合使用。
ADC_CH3	单端模式下，独立工作。
ADC_CH1P_P12	差分模式下，和 ADC_CH1N_P11 组合使用。

ADC_CH4	单端模式下，独立工作。
ADC_CH2N_P13	差分模式下，和 ADC_CH2P_P14 组合使用。
ADC_CH5	单端模式下，独立工作。
ADC_CH2P_P14	差分模式下，和 ADC_CH2N_P13 组合使用。
ADC_CH6	单端模式下，独立工作。
ADC_CH3N_P15	差分模式下，和 ADC_CH3P_P20 组合使用。
ADC_CH7	单端模式下，独立工作。
ADC_CH3P_P20	差分模式下，和 ADC_CH3N_P15 组合使用。
ADC_CH_VOICE	语音通道，采集模拟麦克风使用。

2.1.2 ADC 事件

ADC 事件，会在 ADC 驱动的回调函数抛出。

HAL_ADC_EVT_DATA	ADC 采样数据，如果采样数据就绪，会调用已注册的 ADC 回调函数，送出该事件。
HAL_ADC_EVT_FAIL	ADC 采样失败。

2.1.3 adc_CLOCK_SEL_t

ADC 速率设置。

HAL_ADC_CLOCK_80K	采样速率 80K
-------------------	----------

HAL_ADC_CLOCK_160K

采样速率 160K

HAL_ADC_CLOCK_320K

采样速率 320K

2.2 数据结构

2.2.1 adc_Cfg_t

ADC 配置参数。

uint8_t	channel	配置 ADC 通道, bit2~bit7 对应 P11~P15、P20。
		是否连续采集模式。
bool	is_continue_mode	如果是 true, ADC 会一直自动采集。
		如果是 false, ADC 启动停止由软件控制。
uint8_t	is_differential_mode	是否为差分模式, 差分模式需 P 端 N 端成对工作。 支持 bit7、bit5、bit3, 分别选择[P20,P15]、[P14,P13]、[P12,P11], channel 的配置需和 is_differential_mode 保持一致。
		true 为 bypass mode, 量程为 0V~1V。
uint8_t	is_high_resolution	false 为 attenuation mode, 量程为 0V~ AVDD33。 支持 bit2~bit7, 需要和 channel 配置一致。

2.2.2 adc_Evt_t

ADC 驱动事件的数据结构。

int	type	ADC 事件类型。 HAL_ADC_EVT_DATA: 采样成功, 数据有效。 HAL_ADC_EVT_FAIL: 采样失败, 数据无效。
adc_CH_t	ch	ADC 通道。 通道范围见 adc_CH_t。
uint16_t*	data	ADC 采样数据指针入口。
uint8_t	size	ADC 采样数据个数。

2.2.3 adc_Hdl_t

ADC 回调函数类型。

```
typedef void (*adc_Hdl_t)(adc_Evt_t* pev)
```

2.2.4 adc_Ctx_t

ADC 模块配置。

bool	enable	ADC 模块使能标志。
uint8_t	all_channel	ADC 开启的通道, 支持 bit2~bit7。
adc_Hdl_t	evt_handler	ADC 采集完成回调函数。

2.3 API

2.3.1 void hal_adc_init(void)

ADC 模块初始化, 模块其他函数需要配置之后方可使用, 否则无法预测结果, 或者返回错误。

● 参数

无。

- 返回值

无。

2.3.2 int hal_adc_config_channel (adc_CH_t channel, adc_Cfg_t cfg, adc_Hdl_t evt_handler)

配置 ADC 采集通道。

- 参数

类型	参数名	说明
adc_Cfg_t	cfg	ADC 配置信息。
adc_Hdl_t	evt_handler	事件回调函数。

- 返回值

PPlus_SUCCESS	成功。
其他数值	参考<error.h>

2.3.3 int hal_adc_clock_config(adc_CLOCK_SEL_t clk);

配置 ADC 模块的采样频率，在 ADC 启动之前调用。

- 参数

类型	参数名	说明
adc_CLOCK_SEL_t	clk	ADC 采样频率选择，可以选择为 80K、160K、320K。 默认值为 320K。

- 返回值

PPlus_SUCCESS	成功。
其他数值	参考<error.h>

2.3.4 int hal_adc_start(void)

开始采集。

- 参数

无。

- 返回值

PPlus_SUCCESS	成功。
其他数值	参考<error.h>

2.3.5 int hal_adc_stop(void)

停止采集。

- 参数

无。

- 返回值

PPlus_SUCCESS	成功。
其他数值	参考<error.h>

2.3.6 void __attribute__((weak)) hal_ADC_IRQHandler(void)

ADC 中断处理函数。

- 参数

无。

- 返回值

无。

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

计算 ADC 数值，输出为浮点数，为采样点的电压值，并且是输入 buffer 的算术平均值。

- 参数

类型	参数名	说明
adc_CH_t	ch	ADC 通道。
uint16_t*	buf	ADC 采样数据指针。
uint8_t	size	ADC 采样数据数量。
uint8_t	high_resol	是否为 bypass mode，支持 bit2~bit7。
uint8_t	diff_mode	是否为差分通道，支持 bit7、bit5、bit3。

- 返回值

float	采样点的电压值。
-------	----------

3 软件应用

测试参考硬件：PHY6200_32_V1.4。

测试参考软件：peripheral\adc，在该示例上修改测试。

3.1 连续采集与非连续采集

说明

adc_Cfg_t中的is_continue_mode用来配置ADC是连续采集还是非连续采集。

连续采集指初始化启动ADC一次，ADC将循环自动采集。

非连续采集指ADC采集完毕后将停止，如需再次采集需软件再次触发。

//P14、P15工作模式：连续采集(非连续)、单端、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,  
};
```

//非连续采集模式，软件每500ms启动ADC采集一次

```
static void adc_evt(adc_Evt_t* pev)  
{  
    .....  
    if(adc_cfg.is_continue_mode == FALSE)  
    {  
        osal_start_timerEx( adcDemo_TaskID, adcMeasureTask_EVT,500);  
    }  
    .....  
}
```

3.2 Bypass mode 和 Attenuation mode

说明

adc_cfg_t中的is_high_resolution用于配置ADC通道的采集量程。

该变量的bit2~bit7分别对应P11~P15、P20。

如果相应位置1为bypass mode，量程为0V~1V，推荐使用。

如果相应位置0为attenuation mode，量程为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)  
};
```

3.3 电源电压检测

说明

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。

测量芯片电源电压的引脚需要选择attenuation mode。

测量芯片电源电压的引脚代码上会多一些配置，详见下面示例代码。

//配置P14、P15、P20，其中P20采集芯片电源电压

```
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对应P20, 需要选择attenuation mode
};
```

```
//配置电源电压检测的ADC引脚
static void adcMeasureTask( void )
{
    int ret;

    bool batt_mode = TRUE;
    uint8_t batt_ch = ADC_CH3P_P20;
    GPIO_Pin_e pin;

    .....
}
```

3.4 差分模式

说明

ADC差分模式使用场合不多，ADC工作时只能配置一对差分模式。

通道可配置参数为ADC_BIT(ADC_CH3DIFF)、ADC_BIT(ADC_CH2DIFF)、ADC_BIT(ADC_CH1DIFF)，分别表示P20对P15、P14对P13、P12对P11的差分电压

```
// ADC_CH3DIFF,, 即P20相对于P15上的差分电压
adc_Cfg_t adc_cfg = {
    .channel = ADC_BIT(ADC_CH3DIFF),
    .is_continue_mode = FALSE,
    .is_differential_mode = ADC_BIT(CH3DIFF),
    .is_high_resolution = 0xff,
};
```

3.5 蓝牙广播和连接下的 ADC 采集

蓝牙广播和连接时，瞬间发射功率会影响到ADC基准电压，如此时ADC正在采集，那么采集到的ADC值是一个受干扰的值，不是预期的ADC值。

一种规避方法，在蓝牙广播和蓝牙连接后，延时几毫秒，在ADC基准电压稳定后再进行ADC采集。

参考代码：

```
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 );
    }
}
```


3.6 ADC 移植

之前的ADC驱动不支持多通道采集，这里列出旧驱动向新驱动移植的注意事项。

adc.h和adc.c要配套使用，新的驱动对adc.h软件上的通道顺序做了调整，使用无影响。

新驱动将旧驱动中adc配置参数，由局部变量改为全局变量，部分变量按bit对应adc通道，为了配合多通道使用。

采集单通道非电源电压(左边为旧驱动，右边为新驱动)

```
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)
    {
        //后两个参数需要同cfg保持一致

        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
}
```

采集单通道电源电压(左边为旧驱动，右边为新驱动)

```
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)
    {
        //后两个参数需要同cfg保持一致

        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
}
```

采集差分电压(左边为旧驱动，右边为新驱动)

```
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 = {
```

```
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,
};
```

<pre> .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) { //后两个参数需要同cfg保持一致 float value = hal_adc_value_cal(pev->ch,pev->data, pev->size, TRUE, TRUE); LOG("adc %d\n",(int)(value*1000)); } } </pre>	<pre> static void adcMeasureTask(void) { int ret; bool batt_mode = FALSE; } </pre>
---	--

4 外围电路

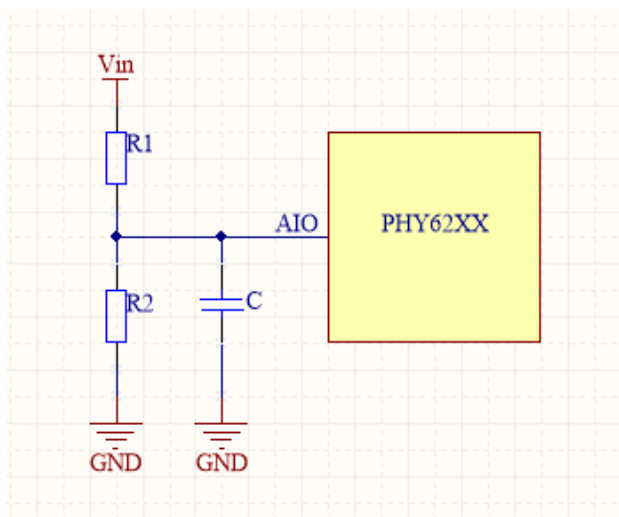
在实际设计 ADC 外围电路中，需要注意被测电压的范围。

4.1 被测电压小于等于芯片工作电压 AVDD33

被测点外围电路不需要分压。

4.2 被测电压大于芯片工作电压 AVDD33

被测点外围电路需要分压，需要保证分压后的电压小于芯片工作电压 AVDD33，如下图：



图表 3: 当被测点电压大于 AVDD33 时外围需要分压电路

- 模式选择 bypass，测试量程为【0V，1V】。
- 检测电压 V_{AIO} 需要小于 1V 。

计算公式如下：

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

$$1、\quad V_{in} \text{ 检测频率 } f_{in} < \frac{1}{2\pi \frac{R1R2}{R1+R2} C}$$

$$2、\quad \text{增益 Gain} = \frac{R2}{R1 + R2}$$

3、 Vin 驱动使能 R1//R2//C

4.3 电源电压检测

如果开启电源电压检测功能，请保持测量电源电压的 ADC 引脚孤立，因为芯片内部已经做了连接处理。