PHY62XX

ADC Application Note

Version 0.5

Author: Fu Xiaoliang

Security: Public

Date: 2019.8

PhyPlus

Revision History

Revision	Author	Participant	Date	Description
V0.2	Fu Xiaoliang		06/13/2018	Draft file
V0.3	Fu Xiaoliang		04/08/2019	Revision
V0.4	Fu Xiaoliang		06/28/2019	Revised edition to support multi-channel acquisition
V0.5	Fu Xiaoliang		08/02/2019	Improve the ADC example, add ADC migration notes, refer to 3.6

Table of Contents

1 Introduction 1

1.1 Mode selection 1

1.2 Acquisition accuracy 2

1.3 Buffer buffering mechanism 2

1.4 Acquisition rate 2

1.5 Operating mode 2

1.5.1 Interruption mode 2

1.5.2 Polling mode 3

2 API 4

2.1 Enumeration & Macro 4

2.1.1 adc_CH_t 4

2.1.2 ADC events 4

2.1.3 adc_CLOCK_SEL_t 5

2.2 Data structure 5

2.2.1 adc_Cfg_t 5

2.2.2 adc_Evt_t 5

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) 6

2.3.3 int hal_adc_clock_config(adc_CLOCK_SEL_t clk); 7

2.3.4 int hal_adc_start(void) 7

2.3.5 int hal_adc_stop(void) 7

2.3.6 void __attribute__((weak)) hal_ADC_IRQHandler(void) 8

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) 8

3 Software application 9

3.1 Continuous acquisition and discontinuous acquisition 9

3.2 Bypass mode and Attenuation mode 10

3.3 Detection of supply voltage 10

3.4 Differential mode 11

3.5 ADC acquisition under Bluetooth broadcast and connection 12

3.6 ADC transplantation 13

4 Peripheral circuits 16

4.1 Measured voltage is less than or equal to the chip operating voltage AVDD33 .. 16

4.2 Measured voltage is greater than the chip operating voltage AVDD33 16

4.3 Detection of supply voltage 17

Figures and Tables

Figure 1: Correspondence between ADC Channel and GPIO 1
Figure 2: Acquisition Accuracy in Different Modes of Normal ADC...............................2
Figure 3: When the Voltage of the Measured Point is Greater than AVDD33, the Peripheral Needs a Voltage Divider Circuit 16

1 Introduction

This file introduces the principle and usage of the PHY62XX ADC module.
PHY62XX ADC has 9 ADC channels in total: 1 PGA, 1 Temp Sensor, 6 Normal ADCs, and 1 Voice.
This file mainly introduces the usage of 6 Normal ADCs.
The corresponding relationship between ADC channel and GPIO is as follows:

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

Figure 1: Correspondence between ADC Channel and GPIO

1.1 Mode selection

The 6 Normal ADC channels have two working modes for selection:

- Single-ended mode: Each channel of ADC_CH2~ADC_CH7 can work independently and acquire the single-ended voltage on its pins.
- Differential mode: ADC_CH2~ADC_CH3, ADC_CH4~ADC_CH5, ADC_CH6~ADC_CH7 shall appear in pairs, and the acquired voltage is the differential voltage on ADC_CH2, ADC_CH4, ADC_CH6 relative to ADC_CH3, ADC_CH5, ADC_CH7.

The 6 Normal ADC channels have two test ranges for selection:

- bypass mode: Test voltage range is [0V, 1V].
- attenuation mode: Test voltage range is [0V, AVDD33].

1.2 Acquisition accuracy

In different modes, the acquisition accuracy of Normal ADC is different, see the table below for details:

	Single-ended 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

Figure 2: Acquisition Accuracy in Different Modes of Normal ADC

1.3 Buffer buffering mechanism

Normal ADC is a 12-bit analog-to-digital converter. Hardware provides 32-word continuous buffers for each ADC channel to buffer the ADC acquisition results. Each word saves the ADC conversion results twice.

1.4 Acquisition rate

Normal ADC supports sampling rates: 80K, 160K, 320K, and the default rate is 320K.

1.5 Operating mode

Normal ADC has two working modes: interrupt mode and polling mode.

1.5.1 Interruption mode

Nine ADC channels share one interrupt entry, the interrupt number is: CM0 (29).
Interrupt of each Normal ADC channel can be individually masked and cleared.
Interrupt trigger condition is that the buffer is full, that is, when the data fills the entire memory, the interrupt will be triggered.
Interrupt mode software processing flow is as follows, many of the flows have been encapsulated in the API and can be used directly.

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 Polling mode

Software processing flow in polling mode:

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

The ADC driver provides asynchronous AD acquisition function, and after the acquisition is completed, the ADC acquisition result is returned through the callback function.

2.1 Enumeration & Macro

2.1.1 adc_CH_t

ADC physical channel.

ADC_CH1	This channel is not currently supported.
ADC_CH2 ADC_CH1N_P11	Work independently in single-ended mode. In differential mode, it is used in combination with ADC_CH1P_P12.
ADC_CH3 ADC_CH1P_P12	Work independently in single-ended mode. In differential mode, it is used in combination with ADC_CH1N_P11.
ADC_CH4 ADC_CH2N_P13	Work independently in single-ended mode. In differential mode, it is used in combination with ADC_CH2P_P14.
ADC_CH5 ADC_CH2P_P14	Work independently in single-ended mode. In differential mode, it is used in combination with ADC_CH2N_P13.
ADC_CH6 ADC_CH3N_P15	Work independently in single-ended mode. In differential mode, it is used in combination with ADC_CH3P_P20.
ADC_CH7 ADC_CH3P_P20	Work independently in single-ended mode. In differential mode, it is used in combination with ADC_CH3N_P15.
ADC_CH_VOICE	Voice channel is used for acquiring analog microphones.

2.1.2 ADC events

ADC events will be thrown in the callback function of the ADC driver.

HAL_ADC_EVT_DATA	ADC sampling data, if the sampling data is ready, the registered ADC callback function will be called to send the event.
HAL_ADC_EVT_FAIL	ADC sampling failed.

2.1.3 adc_CLOCK_SEL_t

ADC rate setting.

HAL_ADC_CLOCK_80K	Sampling rate 80K
HAL_ADC_CLOCK_160K	Sampling rate 160K
HAL_ADC_CLOCK_320K	Sampling rate 320K

2.2 Data structure

2.2.1 adc_Cfg_t

ADC Configuration parameters.

uint8_t	channel	Configure ADC channel, bit2~bit7 correspond to P11~P15, P20.
bool	is_continue_mode	Whether continuous acquisition mode. If it is true, ADC will always acquire automatically. If it is false, ADC start and stop are controlled by software.
uint8_t	is_differential_mode	Whether it is a differential mode, the differential mode requires the P end and N end to work in pairs. Support bit7, bit5, bit3, select [P20,P15], [P14,P13], [P12,P11] respectively, and the channel configuration must be consistent with is_differential_mode.
uint8_t	is_high_resolution	True is bypass mode, and the range is 0V~1V. False is the attenuation mode, and the range is 0V~AVDD33. Support bit2~bit7, which needs to be consistent with the channel configuration.

2.2.2 adc_Evt_t

Data structure of the ADC driving event.

int	type	ADC event type. HAL_ADC_EVT_DATA: Sampling is successful and the data is valid. HAL_ADC_EVT_FAIL: Sampling failed and the data is invalid.
adc_CH_t	ch	ADC Channel See adc_CH_t for channel range.
uint16_t*	data	ADC sampling data pointer entry.
uint8_t	size	Number of data sampled by ADC.

2.2.3 adc_Hdl_t

ADC callback function type.
typedef void (*adc_Hdl_t)(adc_Evt_t* pev)

2.2.4 adc_Ctx_t

ADC module configuration.

bool	enable	ADC module enable flag.
uint8_t	all_channel	Channel opened by ADC supports bit2~bit7.
adc_Hdl_t	evt_handler	ADC acquisition completes callback function.

2.3 API

2.3.1 void hal_adc_init(void)

The ADC module is initialized, and other functions of the module need to be configured before use, otherwise the result cannot be predicted or an error is returned.

- Parameter
  None.

- Return value
  None.

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

Configure the ADC acquisition channel.

- Parameter

Type	Parameter name	Description
adc_Cfg_t	cfg	ADC configuration information.
adc_Hdl_t	evt_handler	Event callback function.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

2.3.3 int hal_adc_clock_config(adc_CLOCK_SEL_t clk);

Configure the sampling frequency of the ADC module, which is called before the ADC starts.

- Parameter

Type	Parameter name	Description
adc_CLOCK_SEL_t	clk	ADC sampling frequency selection, you can choose 80K, 160K, 320K. The default value is 320K.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

2.3.4 int hal_adc_start(void)

Start acquiring.

- Parameter

  None.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

2.3.5 int hal_adc_stop(void)

Stop acquiring.

- Parameter

  None.

- Return value

PPlus_SUCCESS	Succeeded.
Other values	Reference<error.h>

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

ADC interrupt processing function.

- Parameter

  None.

- Return value

  None.

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)

Calculate the ADC value, the output is a floating point number, the voltage value of the sampling point, and the arithmetic average of the input buffer.

- Parameter

Type	Parameter name	Description
adc_CH_t	ch	ADC Channel
uint16_t*	buf	ADC sampling data pointer.
uint8_t	size	The amount of ADC sampled data.
uint8_t	high_resol	Whether it is bypass mode, it supports bit2~bit7.
uint8_t	diff_mode	Whether it is a differential channel, it supports bit7, bit5, and bit3.

- Return value

float	Voltage value of the sampling point.

3 Software application

Test reference hardware: PHY6200_32_V1.4.
Test reference software: peripheral\adc, modify the test on this example.

3.1 Continuous acquisition and discontinuous acquisition

Description
Is_continue_mode in adc_Cfg_t is used to configure whether the ADC is continuous or discontinuous acquisition.
Continuous acquisition means that the ADC is initialized once, and the ADC will automatically acquire cyclically.
Discontinuous acquisition means that the ADC will stop after the acquisition is completed. If you need to acquire again, you need to trigger the software again.

```
//P14, P15 working mode: continuous acquisition (non-continuous), single-ended, 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,
};
```

```
//Discontinuous acquisition mode, the software starts ADC acquisition 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);
                }
  ......
}
```

3.2 Bypass mode and Attenuation mode

Description
Is_high_resolution in adc_Cfg_t is used to configure the acquisition range of the ADC channel.
Bit2~bit7 of this variable correspond to P11~P15 and P20 respectively.
If the corresponding position 1 is bypass mode, and the range is 0V~1V, it is recommended.
If the corresponding position 0 is attenuation mode, the range is 0V~VCC.

```
            {
                  osal_start_timerEx( adcDemo_TaskID, adcMeasureTask_EVT,500);
            }
  ......
}
```

```
//P14, P15 working mode: discontinuous acquisition mode, single-ended, 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 Detection of supply voltage

Description
When measuring the chip supply voltage, AVD33 and ADC pin have been connected inside the chip, so it is necessary to ensure that the ADC pin for measuring the chip power supply is suspended outside the chip
Pin to measure the supply voltage of the chip needs to select the attenuation mode.
There will be more configurations on the pin code for measuring the supply voltage of the chip, see the sample code below for details.

```
//Configure P14, P15, P20, where P20 acquires the chip supply voltage
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 corresponds to P20, so you need to select the
        attenuation mode
};
```

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

3.4 Differential mode

Description
ADC differential mode is not used in many occasions, and only a pair of differential modes can be configured when ADC is working.
Channel configurable parameters are ADC_BIT (ADC_CH3DIFF), ADC_BIT (ADC_CH2DIFF), ADC_BIT (ADC_CH1DIFF), which respectively represent the differential voltage of P20 to P15, P14 to P13, and P12 to P11

```
//ADC_CH3DIFF, which is the differential voltage between P20 and 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,
};
```

3.5 ADC acquisition under Bluetooth broadcast and connection

During Bluetooth broadcast and connection, the instantaneous transmit power will affect the ADC reference voltage. In this case, the ADC is acquiring, then the acquired ADC value is a disturbed value, not the expected ADC value.
A workaround is to delay a few milliseconds after Bluetooth broadcasting and Bluetooth connection, and then perform ADC acquisition after the ADC reference voltage is stable.

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 );
}
}
```

3.6 ADC transplantation

Previous ADC driver does not support multi-channel acquisition. Here are the precautions for porting the old driver to the new driver.
Adc.h and adc.c shall be used together. New driver adjusts the channel sequence on the adc.h software, and the use has no effect.
New driver changes the adc configuration parameters in the old driver from local variables to global variables. Some variables correspond to the adc channel by bit, in order to cooperate with multiple channels.

Collect single-channel non-supply voltage (the old driver on the left and the new driver on the right)

```
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)
{
//The last  two parameters need to be consistent
with 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
}
```

Collect single channel supply voltage (the old driver is on the left, the new driver is on the right)

```
static void adcMeasureTask( void )
{
   int ret;
      bool batt_mode = TRUE;
```

```
adc_Cfg_t adc_cfg = {
      .channel = ADC_BIT(ADC_CH3P_P20),
      .is_continue_mode = FALSE,
      .is_differential_mode = 0x00,
```

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)
{
//The last two parameters need to be consistent with cfg
float value = hal_adc_value_cal(pev->ch,pev->data, pev->size, FALSE, FALSE);
LOG("batt_measure_evt %d\n",(int)(value*1000));
}
}
	.is_high_resolution = **0x00**,
	};
	static void adcMeasureTask(void)
	{
	int ret;
	bool batt_mode = **TRUE**;
	uint8_t batt_ch = **ADC_CH3P_P20**;
	//other code
	}

Collect differential voltage (on the left is the old driver, on the right is the new driver)	

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)
{
//The last two parameters need to be consistent with 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,
	};
	static void adcMeasureTask(void)
	{
	int ret;
	bool batt_mode = **FALSE**;
	}

float value = hal_adc_value_cal(pev->ch,pev->data, pev->size, TRUE, TRUE);
        LOG("adc %d\n",(int)(value*1000));
    }
}	

4 Peripheral circuits

In the actual design of the ADC peripheral circuit, it is necessary to pay attention to the range of the measured voltage.

4.1 Measured voltage is less than or equal to the chip operating voltage AVDD33

Peripheral circuit of the measured point does not require voltage division.

4.2 Measured voltage is greater than the chip operating voltage AVDD33

Peripheral circuit of the tested point needs to be divided, and the divided voltage needs to be less than the chip working voltage AVDD33, as shown in the figure below:

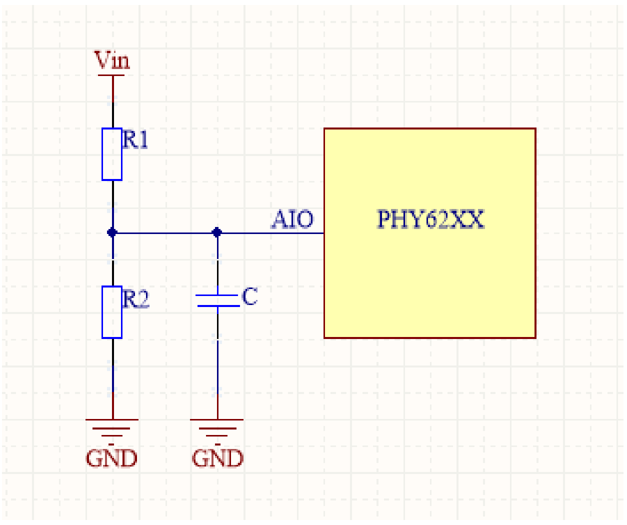

Figure 3: When the Voltage of the Measured Point is Greater than AVDD33, the Peripheral Needs a Voltage Divider Circuit

- Mode selects bypass, and the test range is [0V, 1V].
- Detection voltage needs to be less than 1V.V_{AIO}

The following equation applies:

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

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

2. Gain $\text{Gain} = \frac{R2}{R1+R2}$
3. Vin drive enable R1//R2//C

4.3 Detection of supply voltage

If you turn on the supply voltage detection function, please keep the ADC pin for measuring the supply voltage isolated, because the chip has already been connected.