



Panchip Microelectronics Co., Ltd.

## **PAN2025 Peripheral User Guide**

当前版本: 1.0

发布日期: 2020.02

## **上海磐启微电子有限公司**

地址: 上海张江高科技园区盛夏路 666 号 E 栋 802

联系电话: 021-50802371

网址: <http://www.panchip.com>

## 文档说明

由于版本升级或存在其他原因，本文档内容会不定期进行更新。除非另有约定，本文档内容仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

## 商标

磐启是磐启微电子公司的商标。本文档中提及的其他名称是其各自所有者的商标/注册商标。

## 免责声明

本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，磐启微电子公司对本文档内容不做任何明示或暗示的声明或保证。

## 修订历史

版本	修订时间	描述
V1.0	2020.03.25	初始版本创建

## 目录

1 通用 IO 接口 GPIO.....	1
1.1 GPIO 概述.....	1
1.2 GPIO 寄存器结构.....	1
1.3 GPIO 库函数.....	2
1.3.1 GPIO_SetMode.....	2
1.3.2 GPIO_ClrIntFlag.....	2
1.3.3 GPIO_ClrAllIntFlag.....	3
1.3.4 GPIO_DisableDebounce.....	3
1.3.5 GPIO_EnableDebounce.....	4
1.3.6 GPIO_DisableDigitalPath.....	4
1.3.7 GPIO_EnableDigitalPath.....	5
1.3.8 GPIO_DisablePullupPath.....	5
1.3.9 GPIO_EnablePullupPath.....	6
1.3.10 GPIO_DisableDoutMask.....	6
1.3.11 GPIO_EnableDoutMask.....	7
1.3.12 GPIO_GetIntFlag.....	7
1.3.13 GPIO_SetDebounceTime.....	8
1.3.14 GPIO_GetInData.....	8
1.3.15 GPIO_SetOutData.....	9
1.3.16 GPIO_GetOutData.....	9
1.3.17 GPIO_Toggle.....	10
1.3.18 GPIO_EnableInt.....	10
1.3.19 GPIO_DisableInt.....	11
1.4 GPIO 例程介绍.....	12
1.4.1 gpio_input_output.....	12
1.4.2 gpio_quasi_test.....	13
1.4.3 gpio_quasi_failing_interrupt_test.....	13
1.4.4 gpio_failing_interrupt_test.....	14
1.4.5 gpio_both_edge_interrupt_test.....	15
1.5 注意事项.....	16
2 定时器 TIMER.....	17
2.1 TIMER 概述.....	17
2.2 TIMER 寄存器结构.....	17
2.3 TIMER 库函数.....	18
2.3.1 TIMER_Open.....	18
2.3.2 TIMER_Close.....	18
2.3.3 TIMER_Delay.....	19
2.3.4 TIMER_EnableCapture.....	19
2.3.5 TIMER_DisableCapture.....	20

2.3.	
6	TIMER_EnableEventCounter..... 20
2.3.7	TIMER_DisableEventCounter..... 21
2.3.8	TIMER_GetModuleClock..... 21
2.3.9	TIMER_SelectClockSource..... 22
2.3.10	TIMER_Start..... 22
2.3.11	TIMER_Stop..... 23
2.3.12	TIMER_Reset..... 23
2.3.13	TIMER_EnableWakeup..... 24
2.3.14	TIMER_DisableWakeup..... 24
2.3.15	TIMER_EnableCaptureDebounce..... 25
2.3.16	TIMER_DisableCaptureDebounce..... 25
2.3.17	TIMER_EnableEventCounterDebounce..... 26
2.3.18	TIMER_DisableEventCounterDebounce..... 26
2.3.19	TIMER_EnableInt..... 27
2.3.20	TIMER_DisableInt..... 27
2.3.21	TIMER_EnableCaptureInt..... 28
2.3.22	TIMER_DisableCaptureInt..... 28
2.3.23	TIMER_GetIntFlag..... 29
2.3.24	TIMER_ClearIntFlag..... 29
2.3.25	TIMER_GetTFFlag..... 30
2.3.26	TIMER_ClearTFFlag..... 30
2.3.27	TIMER_GetCaptureIntFlag..... 31
2.3.28	TIMER_ClearCaptureIntFlag..... 31
2.3.29	TIMER_GetCaptureFlag..... 32
2.3.30	TIMER_ClearCaptureFlag..... 32
2.3.31	TIMER_GetWakeupFlag..... 33
2.3.32	TIMER_ClearWakeupFlag..... 33
2.3.33	TIMER_SetCaptureSource..... 34
2.3.34	TIMER_GetCaptureData..... 34
2.3.35	TIMER_GetCounter..... 35
2.4	TIMR 例程介绍..... 36
2.4.1	Timer_Delay..... 36
2.4.2	Timer_Periodic..... 37
2.4.3	Timer_Toggle_Out..... 37
2.4.4	Timer_Event_Counter..... 38
2.4.5	Timer_Free_Counting_Mode..... 38
2.4.6	Timer_TriggerCountingMode..... 38
2.5	注意事项..... 38
3	脉宽调制器 PWM..... 39
3.1	PWM 概述..... 39
3.2	PWM 寄存器结构..... 39
3.3	PWM 库函数..... 40

3.3.	
1 PWM_ConfigOutputChannel.....	40
3.3.2 PWM_Start.....	41
3.3.3 PWM_Stop.....	41
3.3.4 PWM_ForceStop.....	42
3.3.5 PWM_EnableADCTrigger.....	42
3.3.6 PWM_DisableADCTrigger.....	43
3.3.7 PWM_ClearADCTriggerFlag.....	43
3.3.8 PWM_GetADCTriggerFlag.....	44
3.3.9 PWM_EnableOutput.....	44
3.3.10 PWM_DisableOutput.....	45
3.3.11 PWM_EnableDeadZone.....	45
3.3.12 PWM_DisableDeadZone.....	46
3.3.13 PWM_EnableCMPDInt.....	46
3.3.14 PWM_DisableCMPDInt.....	47
3.3.15 PWM_ClearCMPDIntFlag.....	47
3.3.16 PWM_GetCMPDIntFlag.....	48
3.3.17 PWM_EnablePeriodInt.....	48
3.3.18 PWM_DisablePeriodInt.....	49
3.3.19 PWM_ClearPeriodIntFlag.....	49
3.3.20 PWM_GetPeriodIntFlag.....	50
3.3.21 PWM_EnableZeroInt.....	50
3.3.22 PWM_DisableZeroInt.....	51
3.3.23 PWM_ClearZeroIntFlag.....	51
3.3.24 PWM_GetZeroIntFlag.....	52
3.3.25 PWM_EnableCenterInt.....	52
3.3.26 PWM_DisableCenterInt.....	53
3.3.27 PWM_ClearCenterIntFlag.....	53
3.3.28 PWM_GetCenterIntFlag.....	54
3.3.29 PWM_EnableCMPUInt.....	54
3.3.30 PWM_DisableCMPUInt.....	55
3.3.31 PWM_ClearCMPUIntFlag.....	55
3.3.32 PWM_GetCMPUIntFlag.....	56
3.3.33 PWM_PolarityReverse.....	56
3.3.34 PWM_DisablePolarityReverse.....	57
3.4 PWM 例程介绍.....	58
3.4.1 PWM 快速配置.....	58
3.5 注意事项.....	59
4 模数转换器 ADC.....	60
4.1 ADC 概述.....	60
4.2 ADC 寄存器结构.....	60
4.3 ADC 库函数.....	61
4.3.1 ADC_Enable.....	61

4.3.	
2 ADC_Disable.....	61
4.3.3 ADC_Open.....	62
4.3.4 ADC_Close.....	62
4.3.5 ADC_EnableHWTrigger.....	63
4.3.6 ADC_DisableHWTrigger.....	63
4.3.7 ADC_SetExtraSampleTime.....	64
4.3.8 ADC_SelInputRange.....	64
4.3.9 ADC_EnableInt.....	65
4.3.10 ADC_DisableInt.....	65
4.3.11 ADC_SeqModeEnable.....	66
4.3.12 ADC_SeqModeTriggerSrc.....	66
4.3.13 ADC_TRGDLY.....	67
4.4 ADC 例程介绍.....	68
4.4.1 ADC_Compare.....	68
4.4.2 ADC_Convert.....	69
4.4.3 ADC_Calibrate.....	69
4.4.4 ADC_Polling.....	70
4.4.5 ADC_HWTriggerByPWM.....	70
4.4.6 ADC_SequentialMode.....	70
4.5 注意事项.....	70
5 射频模块 RF.....	71
5.1 RF 概述.....	71
5.2 RF 寄存器结构.....	72
5.3 RF 库函数.....	74
5.3.1 RF_Init.....	74
5.3.2 RF_ClockInit.....	75
5.3.3 RF_TxMode.....	75
5.3.4 RF_RxMode.....	76
5.3.5 RF_GetStatus.....	76
5.3.6 RF_ClearStatus.....	77
5.3.7 RF_ClearFIFO.....	77
5.3.8 RF_VccodeRead.....	78
5.3.9 RF_SetChannel.....	78
5.3.10 RF_SetRxLength.....	79
5.3.11 RF_GetRxLength.....	79
5.3.12 RF_SetDataRate.....	80
5.3.13 RF_WhiteningConfig.....	80
5.3.14 RF_ReadBuf.....	81
5.3.15 RF_WriteBuf.....	81
5.3.16 RF_SetAddr.....	82
5.3.17 RF_ReadAddr.....	82
5.3.18 RF_Carrier.....	83

5.4 RF	
例程介绍.....	84
5.4.1 RF_Burst_Interrupt.....	84
5.4.2 RF_Burst_Polling.....	85
5.4.3 RF_Enhance_Interrupt.....	85
5.4.4 RF_Enhance_Polling.....	85
5.4.5 RF_Carrier.....	85
5.5 注意事项.....	85
6 通用异步收发器 UART.....	错误！未定义书签。
6.1 UART 概述.....	错误！未定义书签。
6.2 UART 寄存器结构.....	错误！未定义书签。
6.3 UART 库函数.....	错误！未定义书签。
6.3.1 UART_Open.....	错误！未定义书签。
6.3.2 UART_Close.....	错误！未定义书签。
6.3.3 UART_ClearIntFlag.....	错误！未定义书签。
6.3.4 UART_ReadMultipleData.....	错误！未定义书签。
6.3.5 UART_WriteMultipleData.....	错误！未定义书签。
6.3.6 UART_SetLineConfig.....	错误！未定义书签。
6.3.7 UART_SetTimeoutCnt.....	错误！未定义书签。
6.3.8 UART_ClearRxFifo.....	错误！未定义书签。
6.3.9 UART_Write.....	错误！未定义书签。
6.3.10 UART_Read.....	错误！未定义书签。
6.3.11 UART_GetTxEmpty.....	错误！未定义书签。
6.3.12 UART_GetRxEmpty.....	错误！未定义书签。
6.3.13 UART_IsTxEmpty.....	错误！未定义书签。
6.3.14 UART_WaitTxEmpty.....	错误！未定义书签。
6.3.15 UART_IsRxReady.....	错误！未定义书签。
6.3.16 UART_IsTxFull.....	错误！未定义书签。
6.3.17 UART_IsRxFull.....	错误！未定义书签。
6.3.18 UART_GetTxFull.....	错误！未定义书签。
6.3.19 UART_GetRxFull.....	错误！未定义书签。
6.3.20 UART_EnableInt.....	错误！未定义书签。
6.3.21 UART_DisableInt.....	错误！未定义书签。
6.3.22 UART_SetRxFifoIntTriggerLevel.....	错误！未定义书签。
6.3.23 UART_IsIntOccured.....	错误！未定义书签。
6.3.24 UART_IsFifoStatusMatched.....	错误！未定义书签。
6.3.25 UART_ClearFifoStatus.....	错误！未定义书签。
6.4 UART 例程介绍.....	错误！未定义书签。
6.4.1 UART.....	错误！未定义书签。
6.5 注意事项.....	错误！未定义书签。
7 联系方式.....	86

# 1 通用IO接口 GPIO

## 1.1 GPIO 概述

PAN2025 有多达 24 个通用 I/O 管脚，这些管脚可以通过配置芯片和其他功能管脚共享。这些管脚分配在 P0, P1, P2, P3, P4, P5 六个端口上。每个管脚都是独立的，都有相应的寄存器来控制管脚的功能模式与数据。

I/O 口管脚的状态可由软件配置为输入模式、推挽输出模式、开漏模式和准双向模式。芯片复位后，所有管脚都保持输入模式。

- 四种 IO 模式
  - 准双向模式
  - 输入模式
  - 输出模式
  - 开漏模式
- IO 输入上升沿/下降沿中断
- 上拉模式只在准双向模式使能
- 可配置 IO 唤醒

## 1.2 GPIO 寄存器结构

```
typedef struct {  
    __IO uint32_t MODE;  
    __IO uint32_t DINOFF;  
    __IO uint32_t DOUT;  
    __IO uint32_t DATMSK;  
    __IO uint32_t PIN;  
    __IO uint32_t DBEN;  
    __IO uint32_t INTTYPE;  
    __IO uint32_t INTEN;  
    __IO uint32_t INTSRC;  
}GPIO_T;
```



## 1.3 GPIO 库函数

### 1.3.1 GPIO\_SetMode

#### 句法

```
GPIO_SetMode(GPIO_T *gpio, uint32_t u32PinMask, uint32_t u32Mode)
```

#### 目的

配置 GPIO 口输入输出模式的相关寄存器配置，主要包括高阻抗输入、推挽输出、漏极开路输出和双向输入输出模式

#### 参数

GPIO\_T \*gpio: GPIO 结构体，端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚，有效值 0~7 对应 BIT1~7

uint32\_t u32Mode: GPIO 端口操作模式，有高阻抗输入、推挽输出、漏极开路输出和双向输入输出模式 4 种

#### 返回值

无

### 1.3.2 GPIO\_ClrIntFlag

#### 句法

```
__STATIC_INLINE void GPIO_ClrIntFlag(GPIO_T *gpio, uint32_t u32PinMask)
```

#### 目的

清理指定 GPIO 引脚中断标志

#### 参数

GPIO\_T \*gpio: GPIO 结构体，端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚，有效值 0~7 对应 BIT1~7

#### 返回值

无

### 1.3.3 GPIO\_ClrAllIntFlag

句法

```
__STATIC_INLINE void GPIO_ClrAllIntFlag(GPIO_T *gpio)
```

目的

清理 GPIO 端口所有中断标志

参数

GPIO\_T \*gpio: GPIO 结构体, 端口选择可以是 P0~P5

返回值

无

### 1.3.4 GPIO\_DisableDebounce

句法

```
__STATIC_INLINE void GPIO_DisableDebounce(GPIO_T *gpio, uint32_t u32PinMask)
```

目的

关闭指定 GPIO 引脚的中断去抖功能

参数

GPIO\_T \*gpio: GPIO 结构体, 端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚, 有效值 0~7 对应 BIT1~7

返回值

无

## 1.3.5 GPIO\_EnableDebounce

### 句法

```
__STATIC_INLINE void GPIO_DisableDebounce(GPIO_T *gpio, uint32_t u32PinMask)
```

### 目的

开启指定 GPIO 引脚的中断去抖功能

### 参数

GPIO\_T \*gpio: GPIO 结构体，端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚，有效值 0~7 对应 BIT1~7

### 返回值

无

## 1.3.6 GPIO\_DisableDigitalPath

### 句法

```
__STATIC_INLINE void GPIO_DisableDigitalPath(GPIO_T *gpio, uint32_t u32PinMask)
```

### 目的

关闭指定 GPIO 引脚的 I/O 数字输入通道

### 参数

GPIO\_T \*gpio: GPIO 结构体，端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚，有效值 0~7 对应 BIT1~7

### 返回值

无

## 1.3.7 GPIO\_EnableDigitalPath

### 句法

```
__STATIC_INLINE void GPIO_EnableDigitalPath(GPIO_T *gpio, uint32_t u32PinMask)
```

### 目的

开启指定 GPIO 引脚的 I/O 数字输入通道

### 参数

GPIO\_T \*gpio: GPIO 结构体，端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚，有效值 0~7 对应 BIT1~7

### 返回值

无

## 1.3.8 GPIO\_DisablePullupPath

### 句法

```
__STATIC_INLINE void GPIO_DisablePullupPath(GPIO_T *gpio, uint32_t u32PinMask)
```

### 目的

关闭指定 GPIO 引脚的 I/O 上拉通道

### 参数

GPIO\_T \*gpio: GPIO 结构体，端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚，有效值 0~7 对应 BIT1~7

### 返回值

无

## 1.3.9 GPIO\_EnablePullupPath

句法

```
__STATIC_INLINE void GPIO_ EnablePullupPath(GPIO_T *gpio, uint32_t u32PinMask)
```

目的

开启指定 GPIO 引脚的 I/O 上拉通道

参数

GPIO\_T \*gpio: GPIO 结构体, 端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚, 有效值 0~7 对应 BIT1~7

返回值

无

## 1.3.10 GPIO\_DisableDoutMask

句法

```
__STATIC_INLINE void GPIO_ DisableDoutMask(GPIO_T *gpio, uint32_t u32PinMask)
```

目的

关闭指定 GPIO 引脚的 I/O Dout 上拉

参数

GPIO\_T \*gpio: GPIO 结构体, 端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚, 有效值 0~7 对应 BIT1~7

返回值

无

### 1.3.11 GPIO\_EnableDoutMask

句法

```
__STATIC_INLINE void GPIO_ EnableDoutMask(GPIO_T *gpio, uint32_t u32PinMask)
```

目的

开启指定 GPIO 引脚的 I/O Dout 上拉

参数

GPIO\_T \*gpio: GPIO 结构体, 端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚, 有效值 0~7 对应 BIT1~7

返回值

无

### 1.3.12 GPIO\_GetIntFlag

句法

```
__STATIC_INLINE void GPIO_ GetIntFlag(GPIO_T *gpio, uint32_t u32PinMask)
```

目的

获取指定 GPIO 引脚的中断状态

参数

GPIO\_T \*gpio: GPIO 结构体, 端口选择可以是 P0~P5

uint32\_t u32PinMask: 指定 GPIO 端口的单个或多个引脚, 有效值 0~7 对应 BIT1~7

返回值

无

## 1.3.13 GPIO\_SetDebounceTime

### 句法

```
__STATIC_INLINE void GPIO_ SetDebounceTime(  
    GPIO_ClkSrcDef clksrc,  
    GPIO_ClkSelDef clkssel)
```

### 目的

设置中断去抖采样周期时间

### 参数

GPIO\_ClkSrcDef clksrc: GPIO De-bounce 时钟源。

GPIO\_ClkSelDef clkssel: GPIO De-bounce 采样周期

### 返回值

无

## 1.3.14 GPIO\_GetInData

### 句法

```
__STATIC_INLINE void GPIO_ GetInData(GPIO_T *gpio)
```

### 目的

获取指定 GPIO 端口的寄存器信息

### 参数

GPIO\_T \*gpio: GPIO 结构体，端口选择可以是 P0~P5

### 返回值

无

## 1.3.15 GPIO\_SetOutData

### 句法

```
__STATIC_INLINE void GPIO_ SetOutData(GPIO_T *gpio)
```

### 目的

设置指定的 GPIO 端口

### 参数

GPIO\_T \*gpio: GPIO 结构体, 端口选择可以是 P0~P5

### 返回值

无

## 1.3.16 GPIO\_GetOutData

### 句法

```
__STATIC_INLINE void GPIO_ GetOutData(GPIO_T *gpio)
```

### 目的

获取指定的 GPIO 端口

### 参数

GPIO\_T \*gpio: GPIO 结构体, 端口选择可以是 P0~P5

### 返回值

无



### 1.3.17 GPIO\_Toggle

句法

```
__STATIC_INLINE void GPIO_Toggle(GPIO_T *gpio)
```

目的

翻转指定的 GPIO 引脚

参数

GPIO\_T \*gpio: GPIO 结构体, 端口选择可以是 P0~P5

返回值

无

### 1.3.18 GPIO\_EnableInt

句法

```
__STATIC_INLINE void GPIO_EnableInt(GPIO_T *gpio,  
                                     uint32_t u32Pin,  
                                     uint32_t u32IntAttribs)
```

目的

GPIO 中断触发条件使能

参数

GPIO\_T \*gpio: GPIO 结构体, 端口选择可以是 P0~P5

uint32\_t u32Pin: GPIO 指定的引脚, 可以是 0~7

uint32\_t u32IntAttribs: 指定 GPIO 引脚的中断属性, 可以是高电平、低电平、上升沿、下降沿、上升下降沿

返回值

无

### 1.3.19 GPIO\_DisableInt

#### 句法

```
__STATIC_INLINE void GPIO_DisableInt(GPIO_T *gpio, uint32_t u32Pin)
```

#### 目的

关闭 GPIO 中断触发条件使能

#### 参数

GPIO\_T \*gpio: GPIO 结构体，端口选择可以是 P0~P5

uint32\_t u32Pin: GPIO 指定的引脚，可以是 0~7

#### 返回值

无

## 1.4 GPIO 例程介绍

目前我们提供的 GPIO 例程，如下表所示：

表 1-4. GPIO 例程

例程	说明
gpio_input_output_test()	输入输出测试
gpio_quasi_test()	准双向测试
gpio_quasi_failing_interrupt_test()	准双向/下降沿中断测试
gpio_failing_interrupt_test()	输入/下降沿中断测试
gpio_both_edge_interrupt_test()	输入/上下沿中断测试

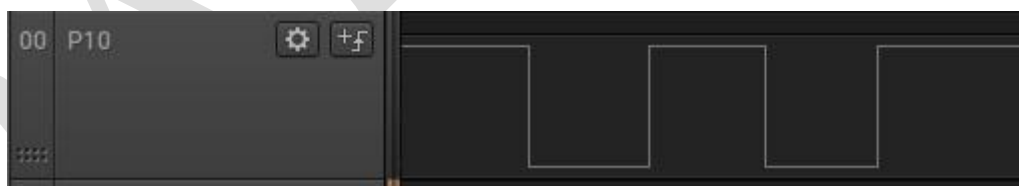
### 1.4.1 gpio\_input\_output

本例程提供 GPIO 的输入/输出测试，可参考 `gpio_input_output_test()` 函数。

#### 1、实验方法：

- 硬件上将 P1.0 与 P3.4 连接。
- 软件上修改宏 `#define TEST_MODE GPIO_INPUT_OUPUT_TEST`。
- 编译下载，根据串口信息进行操作。

#### 2、实验现象



P10 输出：默认内部上拉高->低->高->低->高

P34 输入：检查 P10 电平

串口输出：[OK]/[FAIL] --- Please make sure P1.0 and P3.4 are connected.

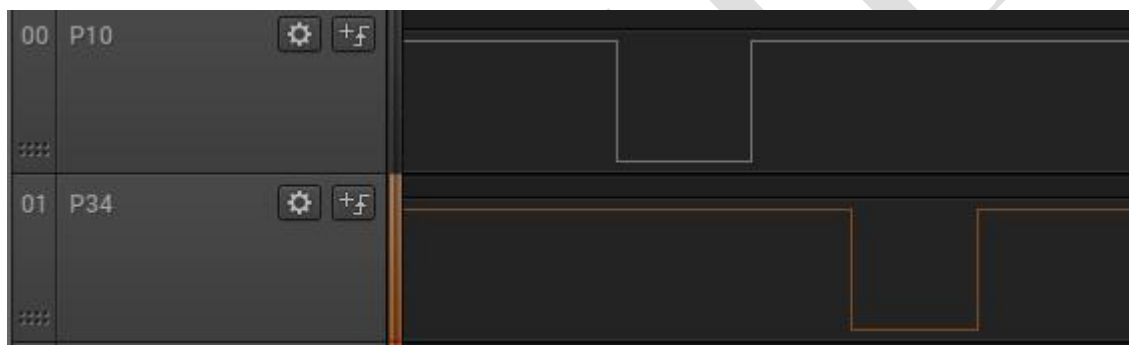
## 1.4.2 gpio\_quasi\_test

本例程提供 GPIO 准双向模式测试，可参考 `gpio_quasi_test()` 函数。

### 1、实验方法：

- A. 硬件上将 P1.0 与 P3.4 连接。
- B. 软件上修改宏 `#define TEST_MODE GPIO_QUASI_TEST`。
- C. 编译下载，根据串口信息进行操作。

### 2、实验现象：



P10 先输出高低电平变换，P34 进行检测，检测完 P34 输出高低电平变化，P10 进行检测。

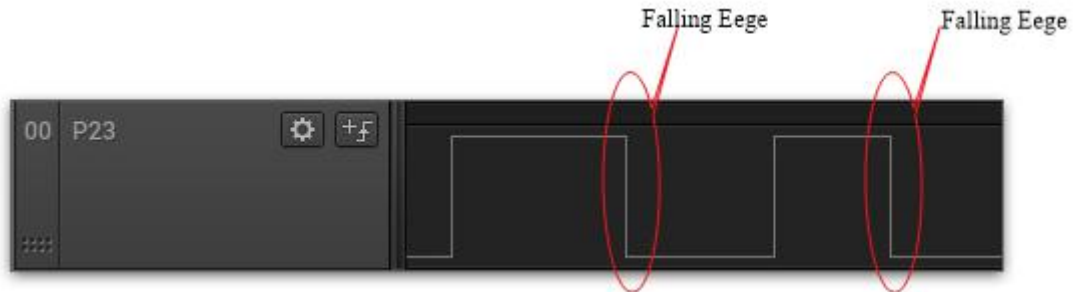
## 1.4.3 gpio\_quasi\_failing\_interrupt\_test

本例程提供 GPIO 下降沿中断(准双向模式)测试

### 1、实验方法：

- A. 硬件上将 P2.2 与 P2.3 连接。
- B. 软件上修改宏 `#define TEST_MODE GPIO_QUASI_FAILING_INTERRUPT_TEST`。
- C. 编译下载，根据串口信息进行操作。

## 2、实验现象：



P23 配置为准双向模式，做为中断源，先输出高电平，延时一段时间，在输出低电平，然后在输出高电平，延时一段时间，在输出低电平，如上图所示，产生 2 个下降沿。

P22 配置为准双向模式，并使能下降沿触发中断。

串口打印如下：

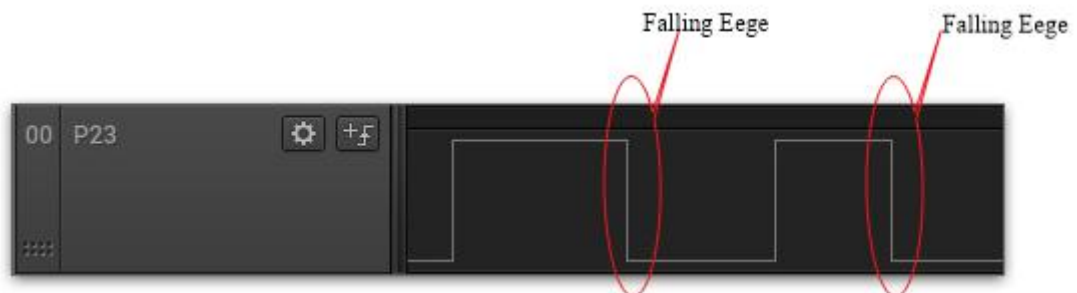
```
>> Please connect P2.2 and P2.3 first <<
Press any key to start test "Quasi-mode and interrupt by falling edge trigger"
P2.2 INT occurred.
P2.2 INT occurred.
```

### 1.4.4 gpio\_failing\_interrupt\_test

#### 1. 实验方法：

- 硬件上将 P3.5 与 P2.3 连接。
- 软件上修改宏 `#define TEST_MODE GPIO_FAILING_INTERRUPT_TEST`。
- 编译下载，根据串口信息进行操作。

#### 2. 实验现象：



P23 配置为输出模式，做为中断源，先输出高电平，延时一段时间，在输出低电平，然后在输出高电平，延时一时间，在输出低电平，如上图所示，产生 2 个下降沿。

P35 配置为输入模式，并使能下降沿触发中断。

串口打印如下：

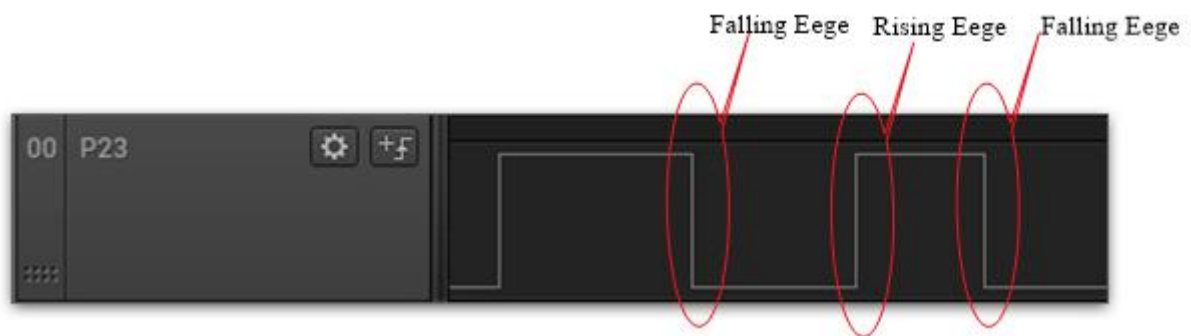
```
>> Please connect P3.5 and P2.3 first <<
Press any key to start test "input-mode and interrupt by falling edge trigger"
P3.5 INT occurred.
P3.5 INT occurred.
```

## 1.4.5 gpio\_both\_edge\_interrupt\_test

### 1. 实验方法：

- A. 硬件上将 P5.0 与 P2.3 连接。
- B. 软件上修改宏#define TEST\_MODE GPIO\_BOTH\_EDGE\_INTERRUPT\_TEST。
- C. 编译下载，根据串口信息进行操作。

### 2. 实验现象：



P23 配置为输出模式，做为中断源，先输出高电平，延时一段时间，在输出低电平，然后在输出高电平，延时一时间，在输出低电平，如上图所示，产生 2 个下降沿，一个上升沿。

P50 配置为输入模式，并使能下降沿和上升沿触发中断。

串口打印如下：

```
>> Please connect P5.0 and P2.3 first <<
Press any key to start test "input-mode and interrupt by falling edge trigger"
P5.0 INT occurred.
P5.0 INT occurred.
P5.0 INT occurred.
```

### 1.5 注意事项

- 1、开漏输出模式，外部需要接上拉电阻；
- 2、准双向模式，要实现数字输入，需要先将 DOUT(Px\_Dout[n])相应位置 1；
- 3、仅在准双向模式下，I/O 管脚内部上拉电阻才使能；
- 4、使用 P56 时，在模式配置完成后，需要进行 3v 区同步，才能够正常使用该 IO 口；

## 2 定时器TIMER

---

### 2.1 TIMER 概述

PAN2025 包含 3 组 32-位定时器，Timer0、Timer1 和 Timer2，提供用户便捷的计数定时功能。

### 2.2 TIMER 寄存器结构

```
typedef struct{  
    __IO uint32_t CTL;  
    __IO uint32_t CMP;  
    __IO uint32_t INTSTS;  
    __I  uint32_t CNT;  
    __I  uint32_t CAP;  
    __IO uint32_t EXTCTL;  
    __IO uint32_t EINTSTS;  
}TIMER_T;
```



## 2.3 TIMER 库函数

### 2.3.1 TIMER\_Open

#### 句法

```
uint32_t TIMER_Open(TIMER_T *timer, uint32_t u32Mode, uint32_t u32Freq)
```

#### 目的

配置定时器指定的工作模式和计数周期

#### 参数

**TIMER\_T \*timer:** TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

**uint32\_t u32Mode:** TIMER 操作模式，可以选择单次计数、周期计数、切换输出、和连续计数四种

**uint32\_t u32Freq:** TIMER 计数期望工作频率

#### 返回值

TIMER 计数实际工作频率

### 2.3.2 TIMER\_Close

#### 句法

```
void TIMER_Close(TIMER_T *timer)
```

#### 目的

停止计数关闭中断

#### 参数

**TIMER\_T \*timer:** TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

#### 返回值

无

### 2.3.3 TIMER\_Delay

#### 句法

```
void TIMER_Delay(TIMER_T *timer, uint32_t u32Usec)
```

#### 目的

启用微秒级别的延时循环

#### 参数

TIMER\_T \*timer: TIMER 结构体, 端口选择可以是 TIMER0、TIMER1 和 TIMER2

uint32\_t u32Usec: 延时周期配置, 范围 10us~1s

#### 返回值

无

### 2.3.4 TIMER\_EnableCapture

#### 句法

```
void TIMER_EnableCapture(TIMER_T *timer, uint32_t u32CapMode, uint32_t u32Edge)
```

#### 目的

启用指定模式和检测边缘的 TIMER 捕获功能

#### 参数

TIMER\_T \*timer: TIMER 结构体, 端口选择可以是 TIMER0、TIMER1 和 TIMER2

uint32\_t u32CapMode: TIMER 捕获模式, 可以选择 FREE\_COUNTING\_MODE、TRIGGER\_COUNTING\_MODE、COUNTER\_RESET\_MODE3 种

uint32\_t u32Edge: TIMER 捕获检测边缘选择, 可以选择 FALLING\_EDGE、RISING\_EDGE、BOTH\_EDGE3 种

#### 返回值

无

## 2.3.5 TIMER\_DisableCapture

### 句法

```
void TIMER_DisableCapture(TIMER_T *timer)
```

### 目的

关闭 TIMER 捕获模式

### 参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

### 返回值

无

## 2.3.6 TIMER\_EnableEventCounter

### 句法

```
void TIMER_EnableEventCounter(TIMER_T *timer, uint32_t u32Edge)
```

### 目的

启用指定检测边缘的 TIMER 事件计数功能

### 参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

uint32\_t u32Edge : 计数引脚边缘检测选择，可选 RISING\_EDGE 和 FALLING\_EDGE 2 种

### 返回值

无

### 2.3.7 TIMER\_DisableEventCounter

#### 句法

```
void TIMER_DisableEventCounter(TIMER_T *timer)
```

#### 目的

关闭 TIMER 事件计数功能

#### 参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

#### 返回值

无

### 2.3.8 TIMER\_GetModuleClock

#### 句法

```
uint32_t TIMER_GetModuleClock(TIMER_T *timer)
```

#### 目的

获取 TIMER 模块时钟频率

#### 参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

#### 返回值

TIMER 模块工作频率

### 2.3.9 TIMER\_SelectClockSource

句法

```
void TIMER_SelectClockSource(TIMER_T *timer, uint32_t u32clkssel)
```

目的

TIMER 时钟源选择

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

uint32\_t u32clkssel: TIMER 时钟源参数

返回值

无

### 2.3.10 TIMER\_Start

句法

```
__STATIC_INLINE void TIMER_Start(TIMER_T *timer)
```

目的

开始 TIMER 计数功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.11 TIMER\_Stop

句法

```
__STATIC_INLINE void TIMER_Stop(TIMER_T *timer)
```

目的

停止 TIMER 计数功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.12 TIMER\_Reset

句法

```
__STATIC_INLINE void TIMER_Reset(TIMER_T *timer)
```

目的

复位 Timer

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.13 TIMER\_EnableWakeup

句法

```
__STATIC_INLINE void TIMER_EnableWakeup(TIMER_T *timer)
```

目的

使能 TIMER 唤醒功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.14 TIMER\_DisableWakeup

句法

```
__STATIC_INLINE void TIMER_DisableWakeup(TIMER_T *timer)
```

目的

关闭 TIMER 唤醒功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.15 TIMER\_EnableCaptureDebounce

句法

```
__STATIC_INLINE void TIMER_EnableCaptureDebounce(TIMER_T *timer)
```

目的

开启捕获检测引脚的去抖功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.16 TIMER\_DisableCaptureDebounce

句法

```
__STATIC_INLINE void TIMER_DisableCaptureDebounce(TIMER_T *timer)
```

目的

关闭捕获检测引脚的去抖功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无



### 2.3.17 TIMER\_EnableEventCounterDebounce

句法

```
__STATIC_INLINE void TIMER_EnableEventCounterDebounce(TIMER_T *timer)
```

目的

开启事件计数引脚的去抖功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.18 TIMER\_DisableEventCounterDebounce

句法

```
__STATIC_INLINE void TIMER_DisableEventCounterDebounce(TIMER_T *timer)
```

目的

关闭事件计数引脚的去抖功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.19 TIMER\_EnableInt

句法

```
__STATIC_INLINE void TIMER_EnableInt(TIMER_T *timer)
```

目的

开启 TIMER time\_out 中断功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.20 TIMER\_DisableInt

句法

```
__STATIC_INLINE void TIMER_DisableInt(TIMER_T *timer)
```

目的

关闭 TIMER time\_out 中断功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.21 TIMER\_EnableCaptureInt

句法

```
__STATIC_INLINE void TIMER_EnableCaptureInt(TIMER_T *timer)
```

目的

开启 TIMER 捕获触发中断功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.22 TIMER\_DisableCaptureInt

句法

```
__STATIC_INLINE void TIMER_DisableCaptureInt(TIMER_T *timer)
```

目的

关闭 TIMER 捕获触发中断功能

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.23 TIMER\_GetIntFlag

句法

```
__STATIC_INLINE uint32_t TIMER_GetIntFlag(TIMER_T *timer)
```

目的

获取 TIMER 的 time\_out 中断是否发生的标志

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.24 TIMER\_ClearIntFlag

句法

```
__STATIC_INLINE void TIMER_ClearIntFlag(TIMER_T *timer)
```

目的

清除 TIMER 的 time\_out 中断标志

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.25 TIMER\_GetTFFlag

句法

```
__STATIC_INLINE uint32_t TIMER_GetTFFlag(TIMER_T *timer)
```

目的

获取 TIMER 的 time\_out 事件是否发生的标志

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.26 TIMER\_ClearTFFlag

句法

```
__STATIC_INLINE void TIMER_ClearTFFlag(TIMER_T *timer)
```

目的

清除 TIMER 的 time\_out 事件标志

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.27 TIMER\_GetCaptureIntFlag

句法

```
__STATIC_INLINE uint32_t TIMER_GetCaptureIntFlag(TIMER_T *timer)
```

目的

获取 TIMER 捕获中断发生的标志

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.28 TIMER\_ClearCaptureIntFlag

句法

```
__STATIC_INLINE void TIMER_ClearCaptureIntFlag(TIMER_T *timer)
```

目的

清除 TIMER 捕获中断标志

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.29 TIMER\_GetCaptureFlag

句法

```
__STATIC_INLINE uint32_t TIMER_GetCaptureFlag(TIMER_T *timer)
```

目的

获取 TIMER 捕获事件发生标志

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.30 TIMER\_ClearCaptureFlag

句法

```
__STATIC_INLINE void TIMER_ClearCaptureFlag(TIMER_T *timer)
```

目的

清除 TIMER 捕获事件标志

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.31 TIMER\_GetWakeupFlag

句法

```
__STATIC_INLINE uint32_t TIMER_GetWakeupFlag(TIMER_T *timer)
```

目的

获取 TIMER 是否唤醒状态

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.32 TIMER\_ClearWakeupFlag

句法

```
__STATIC_INLINE void TIMER_ClearWakeupFlag(TIMER_T *timer)
```

目的

清除 TIMER 唤醒中断

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无



### 2.3.33 TIMER\_SetCaptureSource

句法

```
__STATIC_INLINE void TIMER_SetCaptureSource(TIMER_T *timer)
```

目的

设置 TIMER 捕获源

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.34 TIMER\_GetCaptureData

句法

```
__STATIC_INLINE uint32_t TIMER_GetWakeupFlag(TIMER_T *timer)
```

目的

获取 TIMER 捕获数据

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

### 2.3.35 TIMER\_GetCounter

句法

```
__STATIC_INLINE uint32_t TIMER_GetCounter(TIMER_T *timer)
```

目的

获取当前 TIMER 计数值

参数

TIMER\_T \*timer: TIMER 结构体，端口选择可以是 TIMER0、TIMER1 和 TIMER2

返回值

无

## 2.4 TIMR 例程介绍

目前我们提供的 Timer 例程，如下表所示：

表 5-4.Timer 例程

例程	说明
Timer_Delay	时间延迟功能
Timer_Periodic	时间周期
Timer_ToggleOut	Toggle 模式功能
Timer_EventCouter	
Timer_FreeCountingMode	
Timer_TriggerCountingMode	

### 2.4.1 Timer\_Delay

本例程提供 Timer 的延时功能如何使用。

#### 1、实验方法：

主函数中，Timer 产生 1s 的延时，并翻转 P22 口。

#### 2、实验现象：

通过逻辑分析仪抓取，可以看出，P22 口每 1s 钟翻转一次。



## 2.4.2 Timer\_Periodic

本例程主要验证 Timer 的周期(Periodic)模式；在该模式下，定时器控制器周期性地操作计数和 CMPDAT 的值比较，直到 CNTEN 位由用户软件清 0。

### 1、实验方法：

- A. 配置 Timer0 为周期(Periodic)模式。
- B. 配置 Timer0 工作频率为 1Hz。
- C. 使能 Timer0 中断。
- D. Timer0 中断中翻转 P22 口。

### 2、实验现象

通过逻辑分析仪抓取，可以看出，P22 口每 1s 钟翻转一次。



## 2.4.3 Timer\_Toggle\_Out

本例程提供 Timer 的 Toggle Mode 如何使用。

### 1、实验方法：

- A. 配置 P34 为 Toggle 输出引脚。
- B. 配置 Timer0 为 Toggle 模式。
- C. 配置 Timer0 工作频率为 1000Hz。

### 2、实验现象

通过逻辑分析仪抓取，可以看出，P34 口每 0.5ms 钟翻转一次。



#### **2.4.4 Timer\_Event\_Counter**

TBA

#### **2.4.5 Timer\_Free\_Counting\_Mode**

TBA

#### **2.4.6 Timer\_TriggerCountingMode**

TBA

### **2.5 注意事项**

TBA

## 3 脉宽调制器 PWM

### 3.1 PWM 概述

PAN2025 内置了针对马达驱动应用的 PWM 单元(PWM)。PWM 支持 8 路 PWM 发生器，可配置为相互独立的 8 路 PWM 输出，通道 0 到 7，或配置成 4 对分别带有可编程死区发生器的互补 PWM，通道 0 和 1，2 和 3，4 和 5，6 和 7。

### 3.2 PWM 寄存器结构

```
typedef struct{
    __IO uint32_t CLKPSC;
    __IO uint32_t CLKDIV;
    __IO uint32_t CTL;
    __IO uint32_t PERIOD0;
    __IO uint32_t PERIOD1;
    __IO uint32_t PERIOD2;
    __IO uint32_t PERIOD3;
    __IO uint32_t PERIOD4;
    __IO uint32_t PERIOD5;
    __IO uint32_t PERIOD6;
    __IO uint32_t PERIOD7;
    __IO uint32_t CMPDAT0;
    __IO uint32_t CMPDAT1;
    __IO uint32_t CMPDAT2;
    __IO uint32_t CMPDAT3;
    __IO uint32_t CMPDAT4;
    __IO uint32_t CMPDAT5;
    __IO uint32_t CMPDAT6;
    __IO uint32_t CMPDAT7;
    __IO uint32_t CTL2;
    __IO uint32_t FLAG;
    __IO uint32_t INTEN;
    __IO uint32_t INTSTS;
    __IO uint32_t POEN;
    __IO uint32_t RESERVED0[1];
}
```

```
__IO uint32_t DTCTL;  
__IO uint32_t ADCTL0;  
__IO uint32_t ADCTL1;  
__IO uint32_t ADCTSTS0;  
__IO uint32_t ADCTSTS1;  
__IO uint32_t RESERVED1[4];  
__IO uint32_t PCACTL;  
}PWM_T;
```

## 3.3 PWM 库函数

### 3.3.1 PWM\_ConfigOutputChannel

#### 句法

```
uint32_t PWM_ConfigOutputChannel (PWM_T *pwm,  
                                   uint32_t u32ChannelNum,  
                                   uint32_t u32Frequency,  
                                   uint32_t u32DutyCycle,  
                                   uint32_t u8OperateType)
```

#### 目的

配置 PWM 通道输出方波的相关寄存器配置，主要包含预分频系数、分频值、周期、比较值、计数模式等

#### 参数

**PWM\_T \*pwm:** PWM 结构体，包含全部可能需要配置的信息（PWM 模块基地址）

**uint32\_t u32ChannelNum:** 选择的 PWM 通道，有效值 0~7

**uint32\_t u32Frequency:** 期望输出的方波频率

**uint32\_t u32DutyCycle:** 期望方波的占空比

**uint32\_t u8OperateType:** 计数模式选择，包含边缘、中心、精准中心三种

#### 返回值

返回期望的方波频率

### 3.3.2 PWM\_Start

#### 句法

```
void PWM_Start (PWM_T *pwm, uint32_t u32ChannelMask)
```

#### 目的

PWM 计数器使能开始计数

#### 参数

PWM\_T \*pwm: PWM 结构体, 包含全部可能需要配置的信息 (PWM 模块基地址)

uint32\_t u32ChannelMask: 选择的 PWM 通道, 有效值 0~7

#### 返回值

无

### 3.3.3 PWM\_Stop

#### 句法

```
void PWM_Stop (PWM_T *pwm, uint32_t u32ChannelMask)
```

#### 目的

PWM 输出一直为低, 等同于结束输出方波

#### 参数

PWM\_T \*pwm: PWM 结构体, 包含全部可能需要配置的信息 (PWM 模块基地址)

uint32\_t u32ChannelMask: 选择的 PWM 通道, 有效值 0~7

#### 返回值

无



### 3.3.4 PWM\_ForceStop

#### 句法

```
void PWM_ForceStop (PWM_T *pwm, uint32_t u32ChannelMask)
```

#### 目的

PWM 关闭相应通道的计数器

#### 参数

PWM\_T \*pwm: PWM 结构体, 包含全部可能需要配置的信息 (PWM 模块基地址)

uint32\_t u32ChannelMask: 选择的 PWM 通道, 有效值 0~7

#### 返回值

无

### 3.3.5 PWM\_EnableADCTrigger

#### 句法

```
void PWM_EnableADCTrigger (PWM_T *pwm,  
                             uint32_t u32ChannelNum,  
                             uint32_t u32Condition)
```

#### 目的

触发ADC使能, 包含0点触发ADC使能, 向下比较触发, 周期触发, 向上触发

#### 参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

uint32\_t u32Condition: 选择触发ADC功能的种类, 4种条件均可选择

#### 返回值

无

### 3.3.6 PWM\_DisableADCTrigger

句法

```
void PWM_DisableADCTrigger (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

触发ADC禁止

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.7 PWM\_ClearADCTriggerFlag

句法

```
void PWM_ClearADCTriggerFlag (PWM_T *pwm,  
                               uint32_t u32ChannelNum,  
                               uint32_t u32Condition)
```

目的

清除触发ADC标志

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

uint32\_t u32Condition: 选择触发ADC功能的种类, 4种条件均可选择

返回值

无

### 3.3.8 PWM\_GetADCTriggerFlag

#### 句法

```
uint32_t PWM_GetADCTriggerFlag (PWM_T *pwm,  
                                uint32_t u32ChannelNum,  
                                uint32_t u32Condition)
```

#### 目的

获取触发ADC的标志

#### 参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

uint32\_t u32Condition: 选择触发ADC功能的种类, 4种条件均可选择

#### 返回值

返回触发ADC的标志

### 3.3.9 PWM\_EnableOutput

#### 句法

```
void PWM_EnableOutput (PWM_T *pwm, uint32_t u32ChannelMask)
```

#### 目的

PWM通道输出使能

#### 参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelMask: 选择的PWM通道, 有效值0~7

#### 返回值

无

## 3.3.10 PWM\_DisableOutput

句法

```
void PWM_DisableOutput (PWM_T *pwm, uint32_t u32ChannelMask)
```

目的

PWM通道输出关闭

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelMask: 选择的PWM通道, 有效值0~7

返回值

无

## 3.3.11 PWM\_EnableDeadZone

句法

```
void PWM_EnableDeadZone (PWM_T *pwm,  
                          uint32_t u32ChannelNum,  
                          uint32_t u32Duration)
```

目的

使能插入死区时间, 死区时间 = PWM\_CLK \* (DTInm+1). 这里 nm, 可以是01, 23, 45

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

uint32\_t u32Duration: 死区长度

返回值

无

### 3.3.12 PWM\_DisableDeadZone

句法

```
void PWM_DisableDeadZone (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

禁止插入死区时间

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.13 PWM\_EnableCMPDInt

句法

```
void PWM_EnableCMPDInt (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

比较器向下比较中断使能

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.14 PWM\_DisableCMPDInt

句法

```
void PWM_DisableCMPDInt (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

比较器向下比较中断禁止

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.15 PWM\_ClearCMPDIntFlag

句法

```
void PWM_ClearCMPDIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

清除PWM向下比较中断标志, 当 PWM0\_CHn计数器的值达到CMPn的值后标志被硬件置位, 软件写1清0。

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.16 PWM\_GetCMPDIntFlag

句法

```
uint32_t PWM_GetCMPDIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

获取PWM向下比较中断标志，当 PWM0\_CHn计数器的值达到CMPn的值后标志被硬件置位

参数

PWM\_T \*pwm: PWM结构体，包含全部可能需要配置的信息（PWM模块基地址）

uint32\_t u32ChannelNum: 选择的PWM通道，有效值0~7

返回值

返回PWM向下比较中断标志

### 3.3.17 PWM\_EnablePeriodInt

句法

```
void PWM_EnablePeriodInt (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

PWM Period中断使能

参数

PWM\_T \*pwm: PWM结构体，包含全部可能需要配置的信息（PWM模块基地址）

uint32\_t u32ChannelNum: 选择的PWM通道，有效值0~7

返回值

无

### 3.3.18 PWM\_DisablePeriodInt

句法

```
void PWM_DisablePeriodInt (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

PWM Period中断禁止

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.19 PWM\_ClearPeriodIntFlag

句法

```
void PWM_ClearPeriodIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

清除PWM Period中断标志

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无



### 3.3.20 PWM\_GetPeriodIntFlag

句法

```
uint32_t PWM_GetPeriodIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

获取PWM Period中断标志

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

返回PWM向下比较中断标志

### 3.3.21 PWM\_EnableZeroInt

句法

```
void PWM_EnableZeroInt (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

PWM 0点中断使能, 开启后选择的通道PWM波到达0点会进入中断函数

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.22 PWM\_DisableZeroInt

句法

```
void PWM_DisableZeroInt (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

PWM 0点中断禁止

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.23 PWM\_ClearZeroIntFlag

句法

```
void PWM_ClearZeroIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

清除零点状态标识位, 当标志位在计数器向下计数到达0点时由硬件置1, 软件写1清零

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.24 PWM\_GetZeroIntFlag

句法

```
uint32_t PWM_GetZeroIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

获取PWM 0点中断标志，当 PWMn 计数器向下计数到0点时标志被硬件置1.

参数

PWM\_T \*pwm: PWM结构体，包含全部可能需要配置的信息（PWM模块基地址）

uint32\_t u32ChannelNum: 选择的PWM通道，有效值0~7

返回值

返回零点中断状态

### 3.3.25 PWM\_EnableCenterInt

句法

```
void PWM_EnableCenterInt (PWM_T *pwm,  
                           uint32_t u32ChannelNum,  
                           uint32_t u32IntPeriodType)
```

目的

周期中断使能

参数

PWM\_T \*pwm: PWM结构体，包含全部可能需要配置的信息（PWM模块基地址）

uint32\_t u32ChannelNum: 选择的PWM通道，有效值0~7

uint32\_t u32IntPeriodType: 中断类型选择

返回值

无

### 3.3.26 PWM\_DisableCenterInt

句法

```
void PWM_DisableCenterInt (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

周期中断禁止

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.27 PWM\_ClearCenterIntFlag

句法

```
void PWM_ClearCenterIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

清除PWM周期中断标志, 当 PWM0\_CHn计数器的值达到PERIODn的值后标志被硬件置位, 软件写1清0。

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.28 PWM\_GetCenterIntFlag

句法

```
uint32_t PWM_GetCenterIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

获取PWM周期中断标志，当 PWM0\_CHn计数器的值达到PERIODn的值后标志被硬件置位

参数

PWM\_T \*pwm: PWM结构体，包含全部可能需要配置的信息（PWM模块基地址）

uint32\_t u32ChannelNum: 选择的PWM通道，有效值0~7

返回值

返回PWM周期中断标志

### 3.3.29 PWM\_EnableCMPUInt

句法

```
void PWM_EnableCMPUInt (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

PWM向上比较中断使能

参数

PWM\_T \*pwm: PWM结构体，包含全部可能需要配置的信息（PWM模块基地址）

uint32\_t u32ChannelNum: 选择的PWM通道，有效值0~7

返回值

无

### 3.3.30 PWM\_DisableCMPUInt

句法

```
void PWM_DisableCMPUInt (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

PWM向上比较中断禁止

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.31 PWM\_ClearCMPUIntFlag

句法

```
void PWM_ClearCMPUIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

目的

清除PWM 向上比较中断标志 当 PWM0\_CHn 计数器向上计数达到CMPn的值后标志被硬件置位, 软件写1清零

参数

PWM\_T \*pwm: PWM结构体, 包含全部可能需要配置的信息 (PWM模块基地址)

uint32\_t u32ChannelNum: 选择的PWM通道, 有效值0~7

返回值

无

### 3.3.32 PWM\_GetCMPUIntFlag

#### 句法

```
uint32_t PWM_GetCMPUIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

#### 目的

获取PWM 向上比较中断标志，当 PWM0\_CHn 计数器向上计数达到CMPn的值后标志被硬件置位，软件写1清零

#### 参数

PWM\_T \*pwm: PWM结构体，包含全部可能需要配置的信息（PWM模块基地址）

uint32\_t u32ChannelNum: 选择的PWM通道，有效值0~7

#### 返回值

返回向上比较中断标志

### 3.3.33 PWM\_PolarityReverse

#### 句法

```
void PWM_ClearCMPUIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

#### 目的

清除PWM 向上比较中断标志 当 PWM0\_CHn 计数器向上计数达到CMPn的值后标志被硬件置位，软件写1清零

#### 参数

PWM\_T \*pwm: PWM结构体，包含全部可能需要配置的信息（PWM模块基地址）

uint32\_t u32ChannelNum: 选择的PWM通道，有效值0~7

#### 返回值

无

### 3.3.34 PWM\_DisablePolarityReverse

#### 句法

```
uint32_t PWM_GetCMPUIntFlag (PWM_T *pwm, uint32_t u32ChannelNum)
```

#### 目的

获取PWM 向上比较中断标志，当 PWM0\_CHn 计数器向上计数达到CMPn的值后标志被硬件置位，软件写1清零

#### 参数

PWM\_T \*pwm: PWM结构体，包含全部可能需要配置的信息（PWM模块基地址）

uint32\_t u32ChannelNum: 选择的PWM通道，有效值0~7

#### 返回值

返回向上比较中断标志



## 3.4 PWM 例程介绍

目前我们提供的 PWM 例程，如下表所示：

表 3-4.PWM 例程

例程	说明
PWM	快速配置 PWM 寄存器

### 3.4.1 PWM 快速配置

该例子通过调用 `pwm_calculate()` 函数，根据配置计算出寄存器配置值，用户可以通过串口打印的寄存器值，直接进行快速配置。

#### 1、实验方法：

- 配置 `pwm_calculate` 函数的参数，编译下载进芯片。
- 查看串口打印的寄存器信息，将该信息替换到程序中。
- 查看串口打印的寄存器信息，将该信息替换到程序中。

#### 2、实验现象

##### A. 本实验配置如下：

```
pwm_calculate(PWM,0,200,30,OPERATION_EDGE_ALIGNED);
```

##### B. 串口打印信息如下：

```
please follow these steps to set up the PWM register
PWM->CLKPSC = 3;
PWM->CLKDIV = 4;
PWM_SET_CNTMODE(PWM,0,PWM_CNTMODE_AUTO_RELOAD);
PWM_SET_ALIGNED_TYPE(PWM,0,PWM_EDGE_ALIGNED);
PWM->CMPDAT0= 17999;
PWM->PERIOD0= 59999;
```

##### C. 逻辑分析如下：



### 3.5 注意事项

CNTMODE 不支持 One Shot 功能

## 4 模数转换器 ADC

### 4.1 ADC 概述

PAN2025 包含一个 12 位 8 通道逐次逼近模数转换器(SAR A/D)。A/D 转换器可以由软件、外部引脚(STADC/P3.2)或 PWM 触发。

### 4.2 ADC 寄存器结构

```
typedef struct{
    __I  uint32_t DAT;
    __I  uint32_t RESERVED0[7];
    __IO uint32_t CTL;
    __IO uint32_t CHEN;
    __IO uint32_t CMP0;
    __IO uint32_t CMP1;
    __IO uint32_t STATUS;
    __I  uint32_t RESERVED1[4];
    __IO uint32_t TRGDLY;
    __IO uint32_t EXTSMPT;
    __IO uint32_t SEQCTL;
    __I  uint32_t SEQDAT1;
    __I  uint32_t SEQDAT2;
    __IO uint32_t CTL2;
}ADC_T;
```

## 4.3 ADC 库函数

### 4.3.1 ADC\_Enable

句法

ADC\_Enable(ADC\_T \*adc)

目的

使能ADC转换功能

参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

返回值

无

### 4.3.2 ADC\_Disable

句法

ADC\_Disable(ADC\_T \*adc)

目的

关闭ADC转换功能

参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

返回值

无

### 4.3.3 ADC\_Open

#### 句法

```
ADC_Open(ADC_T *adc,  
          uint32_t u32InputMode,  
          uint32_t u32OpMode,  
          uint32_t u32ChMask)
```

#### 目的

选择开启转换所选通道的输入

#### 参数

ADC\_T \*adc: ADC结构体, ADC模块基地址  
uint32\_t u32InputMode: NULL未用  
uint32\_t u32OpMode: NULL 未用  
uint32\_t u32ChMask: 通道选择位, BIT0对应ch0, BIT1对应ch1...

#### 返回值

无

### 4.3.4 ADC\_Close

#### 句法

```
ADC_Close(ADC_T *adc)
```

#### 目的

关闭ADC模块使能

#### 参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

#### 返回值

无

### 4.3.5 ADC\_EnableHWTrigger

句法

```
ADC_EnableHWTrigger(ADC_T *adc, uint32_t u32Source, uint32_t u32Param)
```

目的

配置选择外部触发条件并启用硬件触发

参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

uint32\_t u32Source: 外部触发源选择, 可以是外部引脚和PWM触发

uint32\_t u32Param: 选择PWM触发时, 此参数用来设置PWM与ADC转换之间的延时, 当通过外部引脚触发时, 改参数用于设置触发条件, 可以为上升沿或下降沿触发

返回值

无

### 4.3.6 ADC\_DisableHWTrigger

句法

```
ADC_DisableHWTrigger(ADC_T *adc)
```

目的

关闭ADC的硬件触发功能

参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

返回值

无

### 4.3.7 ADC\_SetExtraSampleTime

句法

ADC\_SetExtraSampleTime(ADC\_T \*adc, uint32\_t u32ChNum, uint32\_t u32SampleTime)

目的

设置指定通道的ADC采样时间

参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

uint32\_t u32ChNum: NULL 未用

uint32\_t u32SampleTime: ADC采样时间, 可以是0、1、2、4...clock周期

返回值

无

### 4.3.8 ADC\_SelInputRange

句法

ADC\_SelInputRange(uint32\_t u32EnableHigh)

目的

选择输入采样信号的ADC范围

参数

uint32\_t u32EnableHigh: 可以是0和1, 当选择1时, 输入选择0.4~2.4高压档, 当选择位0时, 输入选择0.4~1.4低压档

返回值

无

### 4.3.9 ADC\_EnableInt

句法

```
ADC_EnableInt(ADC_T *adc, uint32_t u32Mask)
```

目的

通过参数配置选择中断类型

参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

uint32\_t u32Mask: 相应BIT对应的中断状态选择, 改参数决定启用哪些中断

返回值

无

### 4.3.10 ADC\_DisableInt

句法

```
ADC_DisableInt(ADC_T *adc, uint32_t u32Mask)
```

目的

通过配置参数关闭对应中断

参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

uint32\_t u32Mask: 相应BIT对应的中断状态选择, 改参数决定启用哪些中断

返回值

无



### 4.3.11 ADC\_SeqModeEnable

句法

```
ADC_SeqModeEnable(ADC_T *adc, uint32_t u32SeqTYPE, uint32_t u32ModeSel)
```

目的

ADC PWM连续模式控制

参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

uint32\_t u32SeqTYPE: 该参数决定PWM中断采样类型, 可选单通道单个中断连续采样两次, 或0~2三个通道任选2个通道, 单中断同时采样

uint32\_t u32ModeSel: 该参数决定任选两通道具体通道选择模式可以为01、12、02三种模式

返回值

无

### 4.3.12 ADC\_SeqModeTriggerSrc

句法

```
ADC_SeqModeTriggerSrc(ADC_T *adc,  
                        uint32_t u32SeqModeTriSrc1,  
                        uint32_t u32SeqModeTriSrc2)
```

目的

ADC PWM连续模式PWM触发源和类型选择

参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

uint32\_t u32SeqModeTriSrc1: 该参数决定第一个PWM触发源和类型选择

uint32\_t u32SeqModeTriSrc2: 该参数决定第二个PWM触发源和类型选择

返回值

无

### 4.3.13 ADC\_TRGDLY

句法

```
ADC_TRGDLY(ADC_T *adc, uint32_t u32Data)
```

目的

PWM触发后延迟ADC启动转换时间

参数

ADC\_T \*adc: ADC结构体, ADC模块基地址

uint32\_t u32Data: 延时时间

返回值

无

## 4.4 ADC 例程介绍

目前我们提供的 ADC 例程，如下表所示：

表 4-4.Timer 例程

例程	说明
ADC_Compare	ADC 转换结果可与指定的值进行比较
ADC_Convert	ADC 转换
ADC_Calibrate	ADC 校准
ADC_Polling	
ADC_HWTriggleByPWM	PWM 触发 ADC 转换
ADC_SequentialMode	PWM 连续转换模式

### 4.4.1 ADC\_Compare

该例子演示 ADC 转换的结果与指定的值进行比较，并触发相应的中断.

#### 1、实验方法：

- 准备好可调稳压源，将电压调到 0V，为防止烧坏芯片，不要将电压调节过大。
- 将稳压源的正极与 P53 相连，稳压源的负极与 GND 相连。
- 使能调节稳压源电压，范围在 0~2v

#### 2、实验现象

- 当调节电压在 0~0.25 时候，串口打印信息如下：  
Channel 0 input < 0x200
- 当调节电压在 0.25~0.375 时候，串口无打印信.
- 当调节电压在 0.375~0.2 时候，串口打印信息如下：  
Channel 0 input >= 0x300

## 4.4.2 ADC\_Convert

该例子演示 ADC 的转换，通过计算并打印出电压值

### 1、实验方法：

- A. 准备好可调稳压源，将电压调到 0V，为防止烧坏芯片，不要将电压调节过大。
- B. 将稳压源的正极与 P53 相连，稳压源的负极与 GND 相连。
- C. 使能调节稳压源电压，范围在 0~2v

### 2、实验现象

串口每 1s 打印 ADC 测量电压值，串口打印数据如下：

```
Convert result is 0.386813
Convert result is 0.387790
Convert result is 0.386813
Convert result is 0.386813
Convert result is 0.391697
Convert result is 0.387790
Convert result is 0.395604
Convert result is 0.419048
Convert result is 0.448352
Convert result is 0.480586
Convert result is 0.499634
Convert result is 0.526496
Convert result is 0.551404
Convert result is 0.552869
Convert result is 0.553846
```

## 4.4.3 ADC\_Calibrate

PAN2025 出厂前会进行 ADC 校准，校准值保存在 info 区，在使用 ADC 前需要先读取 info 的校准值，每个参考电压挡位各有两个校准参数，共四个参数。

如参考电压为 0~2V，info 区存储了 Cali\_a,Cali\_b，那么电压  $v=(code- Cali\_b)/ Cali\_a$ ，其中 code 为寄存器原始值，通过该公式可以直接将 ADC 原始值转换为电压值。

### 1、实验方法：

- A. 准备好可调稳压源，将电压调到 0V，为防止烧坏芯片，不要将电压调节过大。
- B. 将稳压源的正极与 P53 相连，稳压源的负极与 GND 相连。
- C. 使能调节稳压源电压，范围在 0~2v

## 2、实验现象

```
adc value is 1.591479  
adc value is 1.591479  
adc value is 1.591479
```

### 4.4.4 ADC\_Polling

TBA

### 4.4.5 ADC\_HWTriggleByPWM

TBA

### 4.4.6 ADC\_SequentialMode

TBA

## 4.5 注意事项

1. ADC 的分辨率为 12Bit, 但是精度只有 10Bit, 在应用的时候, 可以将低两位去除掉;
2. PAN2025 出厂前会进行 ADC 校准, 在使用 ADC 前需要先读取 info 的校准值参数。

## 5 射频模块 RF

### 5.1 RF 概述

PAN2025 集成了 32 位 MCU 和 2.4G 无线收发电路的单芯片。无线收发电路工作在 2.400--2.483GHz 世界通用 ISM 频段，它集成射频收发机、频率发生器、晶体振荡器、调制解调器等功能模块，并且支持一对多组网和带 ACK 的通信模式。发射输出功率、工作频道以及通信数据率均可配置。它采用 GFSK 通信方式，支持自动应答及自动重传，支持 RSSI 检测功能，自带扰码和 CRC 校验功能在增强模式下，根据是否使能 DYN/NOACK/ACK\_WITH\_PAYLOAD，分成六种常用模式，开发者应根据需要，参考相关的例程。

表 5-1.RF 常用模式

Mode	DYN	ACK_WITH_PAYLOAD	NOACK	说明
Mode1	×	×	×	TX 端在收到 RX 端的 ACK 包后,产生 TX_DS 中断; ACK 接收超时将产生 MAX_RT 中断
Mode2	×	√	×	TX 端发包后收到 RX 端的 ACK 带 payload 产生 TX_DS RX_DR 中断;ACK 接收超时将产生 MAX_RT 中断
Mode3	×	×	√	TX 端发增强模型数据包，产生 TX_DS 中断，RX 端不回 ACK
Mode4	√	×	×	Payload 长度可变，其余跟 Mode1 一样
Mode5	√	√	×	Payload 长度可变，其余跟 Mode2 一样
Mode6	√	×	√	Payload 长度可变，其余跟 Mode3 一样

## 5.2 RF 寄存器结构

```
typedef struct{
    __IO uint32_t CONFIG;
    __IO uint32_t EN_AA;
    __IO uint32_t EN_RXADDR;
    __IO uint32_t SETUP_AW;
    __IO uint32_t SETUP_RETR;
    __IO uint32_t RF_CH;
    __IO uint32_t RF_SETUP;
    __IO uint32_t IRQ_STATUS;
    __IO uint32_t OBSERVE_TX;
    __IO uint32_t DATAOUT_0;
    __IO uint32_t DATAOUT_1;
    __IO uint32_t DATAOUT_2;
    __IO uint32_t DATAOUT_3;
    __IO uint32_t RX_ADDR_P0[5];
    __IO uint32_t RX_ADDR_P1[5];
    __IO uint32_t RX_ADDR_P2;
    __IO uint32_t RX_ADDR_P3;
    __IO uint32_t RX_ADDR_P4;
    __IO uint32_t RX_ADDR_P5;
    __IO uint32_t TX_ADDR[5];
    __IO uint32_t RX_PW_P0;
    __IO uint32_t RX_PW_P1;
    __IO uint32_t RX_PW_P2;
    __IO uint32_t RX_PW_P3;
    __IO uint32_t RX_PW_P4;
    __IO uint32_t RX_PW_P5;
    __IO uint32_t FIFO_STATUS;
    __IO uint32_t PN006_SET_0;
    __IO uint32_t PN006_SET_1;
    __IO uint32_t PN006_SET_2;
    __IO uint32_t PN006_SET_3;
    __IO uint32_t PN006_SET_4;
    __IO uint32_t PN006_SET_5;
    __IO uint32_t PN006_SET_6;
    __IO uint32_t PN006_SET_7;
    __IO uint32_t DEMOD_CAL_0;
    __IO uint32_t DEMOD_CAL_1;
    __IO uint32_t DEMOD_CAL_2;
}
```

```
__IO uint32_t DEMOD_CAL_3;  
__IO uint32_t DEMOD_CAL_4;  
__IO uint32_t DEMOD_CAL_5;  
__IO uint32_t DEMOD_CAL_6;  
__IO uint32_t DEMOD_CAL_7;  
__IO uint32_t DEMOD_CAL_8;  
__IO uint32_t RF_CAL2_0;  
__IO uint32_t RF_CAL2_1;  
__IO uint32_t RF_CAL2_2;  
__IO uint32_t RF_CAL2_3;  
__IO uint32_t RF_CAL2_4;  
__IO uint32_t RF_CAL2_5;  
__IO uint32_t RF_CAL2_6;  
__IO uint32_t RF_CAL2_7;  
__IO uint32_t RF_CAL2_8;  
__IO uint32_t RF_CAL2_9;  
__IO uint32_t DEM_CAL2_0;  
__IO uint32_t DEM_CAL2_1;  
__IO uint32_t DEM_CAL2_2;  
__IO uint32_t DEM_CAL2_3;  
__IO uint32_t DEM_CAL2_4;  
__IO uint32_t DEM_CAL2_5;  
__IO uint32_t DEM_CAL2_6;  
__IO uint32_t DEM_CAL2_7;  
__IO uint32_t DEM_CAL2_8;  
__IO uint32_t DYNPD;  
__IO uint32_t FEATURE;  
__IO uint32_t RF_CAL_0;  
__IO uint32_t RF_CAL_1;  
__IO uint32_t RF_CAL_2;  
__IO uint32_t RF_CAL_3;  
__IO uint32_t RF_CAL_4;  
__IO uint32_t RF_CAL_5;  
__IO uint32_t RF_CAL_6;  
__IO uint32_t RF_CAL_7;  
__IO uint32_t RF_CAL_8;  
__IO uint32_t RF_CAL_9;  
__IO uint32_t BB_CAL_0;  
__IO uint32_t BB_CAL_1;  
__IO uint32_t BB_CAL_2;  
__IO uint32_t BB_CAL_3;  
__IO uint32_t BB_CAL_4;
```



```
__IO uint32_t BB_CAL_5;  
__IO uint32_t BB_CAL_6;  
__IO uint32_t BB_CAL_7;  
__IO uint32_t BB_CAL_8;  
__IO uint32_t BB_CAL_9;  
__IO uint32_t RF_CE;  
__IO uint32_t RF_CMD;  
__IO uint32_t RF_FIFO;  
__IO uint32_t RX_RPL_WID;  
__IO uint32_t CHIP_ID_0;  
__IO uint32_t CHIP_ID_1;  
__IO uint32_t DEVICE_ID_0;  
__IO uint32_t DEVICE_ID_1;  
__IO uint32_t DEVICE_ID_2;  
__IO uint32_t DEVICE_ID_3;  
}RF_T;
```

## 5.3 RF 库函数

### 5.3.1 RF\_Init

句法

RF\_Init(void)

目的

RF初始化

参数

无

返回值

无

### 5.3.2 RF\_ClockInit

句法

RF\_ClockInit(void)

目的

RF时钟初始化

参数

无

返回值

无

### 5.3.3 RF\_TxMode

句法

RF\_TxMode(void)

目的

将RF设置成Tx模式

参数

无

返回值

无

### 5.3.4 RF\_RxMode

句法

```
void RF_RxMode(void)
```

目的

将RF设置成Rx模式

参数

无

返回值

无

### 5.3.5 RF\_GetStatus

句法

```
uint8_t RF_GetStatus(void)
```

目的

获取RF的状态值

参数

无

返回值

返回RF的状态值

### 5.3.6 RF\_ClearStatus

句法

```
void RF_ClearStatus(void)
```

目的

清除RF的IRQ标志

参数

无

返回值

无

### 5.3.7 RF\_ClearFIFO

句法

```
void RF_ClearFIFO(void)
```

目的

清除RF的FIFO

参数

无

返回值

无

### 5.3.8 RF\_VccodeRead

句法

```
void RF_VccodeRead(void)
```

目的

读取RF的VCO的CODE值

参数

无

返回值

无

### 5.3.9 RF\_SetChannel

句法

```
void RF_SetChannel(uint8_t chn)
```

目的

设置RF的通信通道

参数

uint8\_t chn: 通信通道

返回值

无

### 5.3.10 RF\_SetRxLength

句法

```
void RF_SetRxLength(uint8_t length)
```

目的

设置RF的Rx数据长度

参数

uint8\_t length: 数据长度

返回值

无

### 5.3.11 RF\_GetRxLength

句法

```
uint32_t RF_GetRxLength(void)
```

目的

获取RF的Rx数据长度

参数

无

返回值

返回RF的Rx数据长度

### 5.3.12 RF\_SetDataRate

句法

```
void RF_SetDataRate(uint8_t rate,uint8_t dsss)
```

目的

设置RF的通信速率

参数

uint8\_t rate: 通信速率

uint8\_t dsss: dsss 功能是否使能的标志位

返回值

无

### 5.3.13 RF\_WhiteningConfig

句法

```
void RF_WhiteningConfig(FunctionalState NewState)
```

目的

RF的SCR是否使能

参数

**FunctionalState NewState:** SCR是否使能标志状态

返回值

无

### 5.3.14 RF\_ReadBuf

句法

```
void RF_ReadBuf(uint8_t cmd, uint8_t *pBuf, uint8_t length);
```

目的

读取RF的FIFO的值

参数

uint8\_t cmd: RF命令

uint8\_t \*pBuf: 数组指针

uint8\_t length: 长度

返回值

无

### 5.3.15 RF\_WriteBuf

句法

```
void RF_WriteBuf(uint8_t cmd, uint8_t *pBuf, uint8_t length)
```

目的

向RF的FIFO中写入值

参数

uint8\_t cmd: RF命令

uint8\_t \*pBuf: 数组指针

uint8\_t length: 长度

返回值

无



### 5.3.16 RF\_SetAddr

#### 句法

```
void RF_SetAddr(volatile uint32_t* reg, uint8_t* rf_addr_array, uint8_t addr_len)
```

#### 目的

设置RF的通信地址

#### 参数

volatile uint32\_t\* reg: 寄存器指针

uint8\_t\* rf\_addr\_array: 数组指针

uint8\_t addr\_len: 地址长度

#### 返回值

无

### 5.3.17 RF\_ReadAddr

#### 句法

```
void RF_ReadAddr(volatile uint32_t* reg, uint8_t* rf_addr_array, uint8_t addr_len)
```

#### 目的

读取RF的通信地址

#### 参数

volatile uint32\_t\* reg: 寄存器指针

uint8\_t\* rf\_addr\_array: 数组指针

uint8\_t addr\_len: 地址长度

#### 返回值

无

### 5.3.18 RF\_Carrier

句法

```
void RF_Carrier(uint8_t channel)
```

目的

进入RF的载波模式

参数

uint8\_t channel: 通信通道

返回值

无

## 5.4 RF 例程介绍

目前我们提供的 RF 例程，如下表所示：

表 5-4.RF 例程

例程	说明
RF_Burst_Interrupt	Burst 中断模式
RF_Burst_Polling	Burst 轮询模式
RF_Enhance_Interrupt	Enhance 中断模式
RF_Enhance_Polling	Enhance 轮询模式
RF_Carrier	RF 载波模式

### 5.4.1 RF\_Burst\_Interrupt

该例子测试 RF 在 Burst Interrupt 下的通信

1、实验方法：

- A. 配置 RF 为 Burst 模式。
- B. 使能 RF 中断。
- C. 按如下操作发送或接收数据。

```

Description :
The sample code needs two boards. One is Master and
the other is slave. Master will send 1k payload data
to slave. Slave will check if received data is correct
after getting 1k payload.
Please select Master or Slave test
[0] Master   [1] Slave
  
```

### 5.4.2 RF\_Burst\_Polling

该例子演示 ADC 的转换，通过计算并打印出电压值

1、实验方法：

- A. 配置 RF 为 Burst 模式。
- B. 禁止 RF 中断。
- C. 按如下操作发送或接收数据

```
Description :
The sample code needs two boards. One is Master and
the other is slave. Master will send 1k payload data
to slave. Slave will check if received data is correct
after getting 1k payload.
Please select Master or Slave test
[0] Master    [1] Slave
```

### 5.4.3 RF\_Enhance\_Interrupt

TBA

### 5.4.4 RF\_Enhance\_Polling

TBA

### 5.4.5 RF\_Carrier

TBA

## 5.5 注意事项

TBA

## 6 联系方式

---

上海磐启微电子有限公司

电话：021-50802371

传真：021-50802372

地址：中国（上海）自由贸易试验区盛夏路 666 号 E 栋 802

苏州磐启微电子有限公司

电话：0512-68136052

传真：0512-68136051

地址：苏州工业园区崇文路 199 号富华科技大厦 4-F

上海磐启微电子有限公司深圳分公司

电话：0755-26403799

传真：0755-26403799

地址：深圳市南山区科技园科技路 11 号伟杰大厦 106 室