## PI模块

1.PI模块的初始化函数，底层库中已经在函数Driver\_init中调用，Driver\_init函数底层已经在初始化阶段调用。

2.PI通道频率值获取

|  |  |  |
| --- | --- | --- |
| 函数名称 | / | |
| 语法格式 | / | |
| 输入参数 | / | |
| 返回值 | / | / |
| 功能描述 | PI通道频率值，通过TIM外设单元中断计算实现。不需要额外的函数函数。PI通道的频率值，保存在如下数组变量中,16位整型数据   |  |  |  | | --- | --- | --- | | PIN端口 | 变量名称 | 信号值描述 | | 13 | Frequency[0] | PI1通道频率值，单位HZ | | 14 | Frequency[1] | PI2通道频率值，单位HZ | | 15 | Frequency[2] | PI3通道频率值，单位HZ | | 16 | Frequency[3] | PI4通道频率值，单位HZ | | 17 | Frequency[4] | PI5通道频率值，单位HZ | | 18 | Frequency[5] | PI6通道频率值，单位HZ | | |
| 使用示例 | PI6引脚的电平状态保存在变量Frequency[5]中，赋值语句读取即可。 | |

3.PI通道脉冲计数值获取

|  |  |  |
| --- | --- | --- |
| 函数名称 | / | |
| 语法格式 | / | |
| 输入参数 | / | |
| 返回值 | / | / |
| 功能描述 | PI通道脉冲计数值。当通道有上升沿波形时，则计数值会加1，值保存在32位的数组变量中。计数值一直累加，当溢出时会从0开始计数。脉冲上升沿计数保存在如下数组中，32位整型数据：   |  |  |  | | --- | --- | --- | | PIN端口 | 变量名称 | 信号值描述 | | 13 | PI\_Single\_Pulse\_Count[0] | PI1脉冲计数累加值 | | 14 | PI\_Single\_Pulse\_Count[1] | PI2脉冲计数累加值 | | 15 | PI\_Single\_Pulse\_Count[2] | PI3脉冲计数累加值 | | 16 | PI\_Single\_Pulse\_Count[3] | PI4脉冲计数累加值 | | 17 | PI\_Single\_Pulse\_Count[4] | PI5脉冲计数累加值 | | 18 | PI\_Single\_Pulse\_Count[5] | PI6脉冲计数累加值 | | |
| 使用示例 | PI6引脚的脉冲计数累加值保存在变量PI\_Single\_Pulse\_Count[5]中，赋值语句读取即可。 | |

4.PI通道双脉冲计数值获取

|  |  |  |
| --- | --- | --- |
| 函数名称 | / | |
| 语法格式 | / | |
| 输入参数 | / | |
| 返回值 | / | / |
| 功能描述 | 实现双脉冲计数功能，计数在两个通道的上升沿和下降沿都会计数。计数值可能是递增或者递减。  以双脉冲组0(PI1和PI5为例)，下图TI1表示PI1信号，TI2表示PI5信号。函数返回的双脉冲计数值在两个通道信号的上升沿和下降沿都会计数。递增计数还是递减计数，要看编码器哪一路信号的相位超前。值保存在32位的数组变量中。计数值一直累加，当溢出时会从0开始计数。脉冲上升沿计数保存在如下数组中，32位整型数据：  IMG_256   |  |  |  | | --- | --- | --- | | PIN端口 | 变量名称 | 信号值描述 | | 13和17 | PI\_Single\_Pulse\_Count[0] | PI1和PI5脉冲计数累加值 | | 14和19 | PI\_Single\_Pulse\_Count[1] | PI2和PI6脉冲计数累加值 | | |
| 使用示例 | PI1接编码器的A相，PI5接编码器的B相，想要获取AB两相的双脉冲值，保存在变量PI\_Double\_Pulse\_Count[0],，赋值语句读取即可。 | |

5.PI通道双脉冲计数值清0

|  |  |  |
| --- | --- | --- |
| 函数名称 | API\_PI\_Double\_Count\_Reset | |
| 语法格式 | void API\_PI\_Double\_Count\_Reset(uint8\_t group\_u8) | |
| 输入参数 | group\_u8：  0-表示使用PI1和PI5组成的双脉冲通道  1-表示使用PI2和PI6组成的双脉冲通道 | |
| 返回值 | / | / |
| 功能描述 | 函数用于实现双脉冲通道，计数值的清0. | |
| 使用示例 | PI1接编码器的A相，PI5接编码器的B相，想要清除组0通道的双脉冲计数值,则调用函数实现。  API\_PI\_Double\_Count\_Reset(0); | |

6.PI通道双脉冲方向获取

|  |  |  |
| --- | --- | --- |
| 函数名称 | API\_PI\_Double\_Dir\_Get | |
| 语法格式 | uint16\_t API\_PI\_Double\_Dir\_Get(uint8\_t group\_u8) | |
| 输入参数 | group\_u8：  0-表示使用PI1和PI5组成的双脉冲通道  1-表示使用PI2和PI6组成的双脉冲通道 | |
| 返回值 | uint16\_t | 1. 表示双脉冲递增计数 2. 表示双脉冲递减计数 |
| 功能描述 | 函数用于获取双脉冲的方向，和编码器组合使用，可以获取转动方向。至于0/1哪个表示正转或者反转，需要结合编码器特性。信号正确输入双脉冲通道后，编码器正转或者反转，函数返回值为0或者1. | |
| 使用示例 | PI1接编码器的A相，PI5接编码器的B相，想要获取编码器的旋转方向，并保存在变量CW\_Value中,则调用函数实现。  CW\_Value = API\_PI\_Double\_Dir\_Get(0); | |

**1.2 CAN模块**

## CAN通信模块

1.CAN模块的初始化函数，底层库中已经在函数Driver\_init中调用，Driver\_init函数需要用户在初始化阶段调用。

2.CAN波特率配置函数

|  |  |  |
| --- | --- | --- |
| 函数名称 | CAN\_Hardware\_Config | |
| 语法格式 | void CAN\_Hardware\_Config(uint16\_t can1\_baud,uint16\_t can2\_baud) | |
| 输入参数 | can1\_baud：对CAN1通道波特率进行配置。  can2\_baud：对CAN2通道波特率进行配置。  数据范围：  50:配置波特率为50kbps  100:配置波特率为100kbps  125:配置波特率为125kbps  250:配置波特率为250kbps  500:配置波特率为500kbps  1000:配置波特率为1Mbps  CAN1输入其他值会默认配置为500kbps.  CAN2输入其他值会默认配置为250kbps. | |
| 返回值 | / | / |
| 功能描述 | 此函数用于配置CAN1通道和CAN2通道的波特率值。  CAN1通道指的是接插件的53/60引脚通道的通信。默认500kbps。  CAN2通道指的是接插件的55/61引脚通道的通信。默认250kbps。 | |
| 使用示例 | 配置实现CAN1通道波特率250kbps和CAN2通道的波特率500kbps：  调用函数CAN\_Config(250,500);实现配置功能。 | |

3.CAN1通道数据接收函数：

|  |  |  |
| --- | --- | --- |
| 函数名称 | CAN1\_ReceiveObj | |
| 语法格式 | signed short CAN1\_ReceiveObj(uint32\_t ID, uint8\_t\* Len, uint8\_t\* Data) | |
| 输入参数 | ID：需要接收报文的ID值  \*Len：接收到的数据长度  \*data：接收到的数据保存位置的指针 | |
| 返回值 | UINT8 | 0：接收成功  -1：接收失败 |
| 功能描述 | 此函数用于实现CAN1通道的具体报文的接收，需要设置接收CANID值，接收到的数据长度和具体的数值，分别保存在对应的指针变量中。 | |
| 使用示例 | 配置实现CAN1通道接收ID=0x151的标准帧报文，数据长度保存在变量Test\_Can1\_Receive\_Data.canID = 0x151;  CAN1\_ReceiveObj(Test\_Can1\_Receive\_Data.canID, &Test\_Can1\_Receive\_Data.len, Test\_Can1\_Receive\_Data.data); | |

4.CAN1通道发送函数1：

函数Can1\_Tx\_Msg

|  |  |  |
| --- | --- | --- |
| 函数名称 | Can1\_Tx\_Msg | |
| 语法格式 | uint8\_t Can1\_Tx\_Msg(uint32\_t id,uint8\_t ide,uint8\_t rtr,uint8\_t len,uint8\_t \*dat) | |
| 输入参数 | id：发送报文的ID值  ide：标准帧还是扩展帧。0-发送标准帧;1-发送扩展帧  rtr：数据帧还是远程帧。0-发送数据帧;1-发送远程帧  len：发送数据域的长度 范围0-8  \*dat：发送报文的数据指针 | |
| 返回值 | UINT8 | 0/1/2 发送成功,返回邮箱编号  255 发送失败，无空闲邮箱 |
| 功能描述 | 此函数用于实现CAN1通道的具体报文的发送，需要设置发送的帧类型，报文ID，数据信息，发送周期等。函数内部为直接操作CAN相关的寄存器语句。 | |
| 使用示例 | 配置实现CAN1通道发报文ID=0x110，发送Test\_Can1\_Send\_Data.data[8]中的数据，长度8字节，标准帧，周期发送，发送周期200ms：  Test\_Can1\_Send\_Data.canID = 0x110;  Test\_Can1\_Send\_Data.ext = 0;  Test\_Can1\_Send\_Data.len = 8;  Can1\_Tx\_Msg(Test\_Can1\_Send\_Data.canID,Test\_Can1\_Send\_Data.ext,  Test\_Can1\_Send\_Data.len,Test\_Can1\_Send\_Data.data); | |

5.CAN1通道发送函数2：

函数CAN1\_WriteData

|  |  |  |
| --- | --- | --- |
| 函数名称 | CAN1\_WriteData | |
| 语法格式 | void CAN1\_WriteData(uint32\_t msgID, uint8\_t bBytes[], int8\_t iNoBytes,uint8\_t ext,uint8\_t mode, uint16\_t cycle\_ms) | |
| 输入参数 | msgID：发送报文的ID值  bBytes[]：发送报文的数据数组  iNoBytes：发送数据域的长度 范围0-8  ext：标准帧还是扩展帧。0-发送标准帧;1-发送扩展帧  mode：发送模式。0-不发送;1-周期发送;2-改变发送即发送的数据值发生改变时再发送。  cycle\_ms：周期发送时间配置 | |
| 返回值 | / | / |
| 功能描述 | 此函数用于实现CAN通道的具体报文的发送，需要设置发送的帧类型，报文ID，数据信息，发送周期等。进行逻辑判断并调用函数Can1\_Tx\_Msg实现具体的发送动作。 | |
| 使用示例 | 配置实现CAN1通道发报文ID=0x110，发送Test\_Can1\_Send\_Data.data[8]中的数据，长度8字节，标准帧，周期发送，发送周期200ms：  Test\_Can1\_Send\_Data.canID = 0x110;  Test\_Can1\_Send\_Data.tick = 0;  Test\_Can1\_Send\_Data.ext = 0;  Test\_Can1\_Send\_Data.len = 8;    CAN1\_WriteData(Test\_Can1\_Send\_Data.canID,Test\_Can1\_Send\_Data.data, 8,0,TXMODE\_CYCLE,200); | |

6.CAN2通道数据发送接收函数：

函数CAN2\_ReceiveObj：CAN2通道接收函数。具体函数使用参考CAN1\_ReceiveObj函数。

函数Can2\_Tx\_Msg：CAN2通道发送函数1。具体函数使用参考Can1\_Tx\_Msg函数。

函数CAN2\_WriteData：CAN2通道发送函数2。具体函数使用参考CAN1\_WriteData函数。

**待更新CAN3部分**

## 1.3.看门狗模块

**1.外部看门狗功能禁止/使能函数：**API\_WatchDog\_Enable

|  |  |  |
| --- | --- | --- |
| 函数名称 | API\_WatchDog\_Enable | |
| 语法格式 | uint8\_t API\_WatchDog\_Enable(uint8\_t Statue\_u8) | |
| 输入参数 | Statue\_u8：  0：禁止看门狗  1：打开看门狗 | |
| 返回值 | uint8\_t | 无意义 |
| 功能描述 | 系统采用外部看门狗，使用的芯片为SP706SEN-L，函数实现对看门狗功能的禁止和使能。看门狗使能后，需要在合适的时间进行喂狗操作，否则会发生芯片NRST引脚的复位。 | |
| 使用示例 | 打开外部芯片看门狗功能：  调用函数API\_WatchDog\_Enable(1);即可实现 | |

**2.外部看门狗喂狗函数：**API\_WatchDog\_FeedDog

|  |  |  |
| --- | --- | --- |
| 函数名称 | API\_WatchDog\_FeedDog | |
| 语法格式 | uint8\_t API\_WatchDog\_FeedDog(void) | |
| 输入参数 | / | |
| 返回值 | uint8\_t | 无意义 |
| 功能描述 | 系统采用外部看门狗，使用的芯片为SP706SEN-L，函数实现对看门狗的喂狗功能，防止发生看门狗复位。 | |
| 使用示例 | 对看门狗进行喂狗：  调用函数API\_WatchDog\_FeedDog();即可实现。 | |