NV32F100x GPIO 通用 IO 编程

第一章 所有库函数简介

库函数列表

void GPIO_Init(GPIO_Type *pGPIO, uint32_t u32PinMask, GPIO_PinConfigType sGpioType)
GPIO 初始化函数
void GPIO_Toggle(GPIO_Type *pGPIO, uint32_t u32PinMask)
IO 电平批量翻转函数
uint32_t GPIO_Read(GPIO_Type *pGPIO)
IO 电平读取函数
void GPIO_Write(GPIO_Type *pGPIO, uint32_t u32Value)
IO 电平输出函数
uint8_t GPIO_BitRead(GPIO_PinType GPIO_Pin)
单个 IO 电平读取函数
void GPIO_PinToggle(GPIO_PinType GPIO_Pin)
单个 IO 电平翻转函数
void GPIO_PinSet(GPIO_PinType GPIO_Pin)
单个 IO 输出高电平
void GPIO_PinClear(GPIO_PinType GPIO_Pin)
单个 IO 输出电平

寄存器操作的内联函数，调用内联函数和直接操作寄存器效率一样高
FGPIO_Toggle(FGPIO_Type *pFGPIO, uint32_t u32PinMask)
IO 电平翻转
uint32_t FGPIO_Read(FGPIO_Type *pFGPIO)
读取 IO 电平
void FGPIO_Write(FGPIO_Type *pFGPIO, uint32_t u32Value)
输出 IO 电平

GPIO 特性说明

*端口数据输入寄存器适用于所有数字引脚多路复用模式
*端口数据输入寄存器带对应的置位/清零/切换寄存器
*端口数据方向寄存器
*通过 IOPORT 实现对 GPIO 寄存器的零等待状态访问

GPIO 使用说明

对 IO 进行输入输出方向的配置后，相应的 IO 口就可以使用了，对输出寄存器对应 IO 位赋值就可以输出 1 或者 0，输入只需要读取输入寄存器的值就可以读入相应 IO 口的输入电平值。

1.1 GPIO 模块初始化

函数名	GPIO_Init
函数原形	GPIO_Init(GPIO_Type *pGPIO, uint32_t u32PinMask, GPIO_PinConfigType sGpioType)
功能描述	以配置结构体 sGpioType 和 pinMask 来初始化 pinMask 对应的 GPIO
输入参数	GPIO_Type 结构体，gpio mask，GPIO 配置结构体
输出参数	无
返回值	无
先决条件	无
函数使用实例	GPIO_Init(GPIOB, GPIO_PTE7_MASK, GPIO_PinInput); 设置 PTE 为输入 PIN

对 GPIO_PDDR 和 GPIO_PIDR 进行配置即可完成 GPIO 的输入输出配置

字段	描述
PDD	端口数据方向 将各个端口引脚配置为输入或输出: 0 引脚配置为通用输入，用于 GPIO 功能。若在 GPIOx_PIDR 寄存器中禁用端口输入，则引脚将为高阻态。 1 引脚配置为通用输出，用于 GPIO 功能。

字段	描述
PID	端口输入禁用 0 假设引脚配置为用于任意数字功能，则该引脚配置为通用输入。 1 引脚未配置为通用输入。对应的端口数据输入寄存器位将读取零。

```
/*****************************************************************************//*!
* @简介      初始化对应的 GPIO
*
*
* @无返回
*
*****************************************************************************/
void GPIO_Init(GPIO_Type *pGPIO, uint32_t u32PinMask, GPIO_PinConfigType sGpioType)
{

#if defined(CPU_NV32)
    ASSERT((pGPIO == GPIOA) || (pGPIO == GPIOB));
#endif
    /* 初始化 GPIO 为输入或者输出 */
    if ((sGpioType == GPIO_PinOutput) || (sGpioType == GPIO_PinOutput_HighCurrent))
    {
        pGPIO->PDDR |= u32PinMask;          /* 使能 IO 端口方向寄存器 */
        pGPIO->PIDR |= u32PinMask;          /* 设置输入禁用寄存器 */
```

```
    }
    {
        pGPIO->PDDR &= ~u32PinMask;     /* 禁用 IO 端口方向寄存器 */
        pGPIO->PIDR &= ~u32PinMask;     /* 设置输入禁用寄存器 */
    }
    /* 配置 GPIO 是否默认上拉 */
#if defined(CPU_NV32)
    switch((uint32_t)pGPIO)
    {
        case GPIOA_BASE:
            (sGpioType == GPIO_PinInput_InternalPullup)?(PORT->PUEL |= u32PinMask):(PORT->PUEL &=
~u32PinMask);
            break;
        case GPIOB_BASE:
            (sGpioType == GPIO_PinInput_InternalPullup)?(PORT->PUEH |= u32PinMask):(PORT->PUEH
&= ~u32PinMask);
            break;
        default:
            break;
     }
#endif

    /* 配置 GPIO 端口是否为高电流驱动 */
#if defined(CPU_NV32) | defined(CPU_NV32M4)
    if (pGPIO == GPIOA)
    {
        if (u32PinMask & GPIO_PTB4_MASK)
        {
            PORT->HDRVE |= PORT_HDRVE_PTB4_MASK;
        }
        if (u32PinMask & GPIO_PTB5_MASK)
        {
            PORT->HDRVE |= PORT_HDRVE_PTB5_MASK;
        }
        if (u32PinMask & GPIO_PTD0_MASK)
        {
            PORT->HDRVE |= PORT_HDRVE_PTD0_MASK;
        }
        if (u32PinMask & GPIO_PTD1_MASK)
        {
            PORT->HDRVE |= PORT_HDRVE_PTD1_MASK;
        }
    }
    if (pGPIO == GPIOB)
```

```
    {
        if (u32PinMask & GPIO_PTE0_MASK)
        {
            PORT->HDRVE |= PORT_HDRVE_PTE0_MASK;
        }
        if (u32PinMask & GPIO_PTE1_MASK)
        {
            PORT->HDRVE |= PORT_HDRVE_PTE1_MASK;
        }
        if (u32PinMask & GPIO_PTH0_MASK)
        {
            PORT->HDRVE |= PORT_HDRVE_PTH0_MASK;
        }
        if (u32PinMask & GPIO_PTH1_MASK)
        {
            PORT->HDRVE |= PORT_HDRVE_PTH1_MASK;
        }
    }

#endif

}
```

1.2 IO 电平批量翻转函数

函数名	GPIO_Toggle
函数原形	GPIO_Toggle(GPIO_Type *pGPIO, uint32_t u32PinMask)
功能描述	对 对应 PinMask 的 GPIO 切换电压（可以单个，也可以多个）
输入参数	GPIO_Type 模块结构体，gpio PinMask
输出参数	无
返回值	无
先决条件	对该组 GPIO 设置为输出
函数使用实例	GPIO_PinToggle(GPIO_PTE7)；PTE7 管脚电压切换，原来为 1，则为 0

字段	描述
PTTO	端口切换输出 如下所示对该寄存器进行写操作将更新 PDOR 中对应位的值： 0 PDORn 中的对应位不发生变化。 1 PDORn 中的对应位置位为现有逻辑状态的反相电平。

```
/*****************************************************************************//*!
* @简介  将对应 IO 的电平取反
*
* @无返回
```

```
*****************************************************************************/
void GPIO_Toggle(GPIO_Type *pGPIO, uint32_t u32PinMask)
{

#if defined(CPU_NV32)
        ASSERT((pGPIO == GPIOA) || (pGPIO == GPIOB));
#endif
#if defined(CPU_NV32M3)
      ASSERT(pGPIO == GPIOA);
#endif
#if defined(CPU_NV32M4)
         ASSERT((pGPIO == GPIOA) || (pGPIO == GPIOB) || (pGPIO == GPIOC));
#endif

      pGPIO->PTOR = u32PinMask;     /* 对翻转寄存器配置 */
}
```

1.3 IO 电平读取函数

函数名	GPIO_Read
函数原形	GPIO_Read(GPIO_Type *pGPIO)
功能描述	读取 pGPIO 对应的值
输入参数	GPIO_Type 模块结构体
输出参数	无
返回值	返回读取值
先决条件	对该组 GPIO 设置为输入
函数使用实例	GPIO_Read(GPIOA);

字段	描述
PDI	端口数据输入 在特定器件的未实施引脚读取 0。未配置为用于数字功能的引脚读取 0。如果禁用端口控制和中断模块，那么 PDIR 中的对应位不更新。 0 引脚逻辑电平为逻辑 0，或者未配置为供数字功能使用。 1 引脚逻辑电平为逻辑 1。

```
/*****************************************************************************//*!
* @简介  读取所有 IO 的输入
*
* @返回 32 个 IO 端口的输入电平
*****************************************************************************/
uint32_t GPIO_Read(GPIO_Type *pGPIO)
{

#if defined(CPU_NV32)
```

```
    ASSERT((pGPIO == GPIOA) || (pGPIO == GPIOB));
#endif
#if defined(CPU_NV32M3)
    ASSERT(pGPIO == GPIOA);
#endif
#if defined(CPU_NV32M4)
    ASSERT((pGPIO == GPIOA) || (pGPIO == GPIOB) || (pGPIO == GPIOC));
#endif

    return (pGPIO->PDIR);    /* 返回 32 个 IO 口的输入电平 */

}
```

1.4 IO 电平输出函数

函数名	GPIO_Write
函数原形	GPIO_Write(GPIO_Type *pGPIO, uint32_t u32Value)
功能描述	GPIO 输出所写的值
输入参数	GPIO_Type 模块结构体，要写的值
输出参数	无
返回值	无
先决条件	对该组 GPIO 设置为输出
函数使用实例	GPIO_Read(GPIOA, 0xFFFFFFFF)；GPIOA 组全部输出 1

字段	描述
PDO	端口数据输出 读取未连接引脚的寄存器位将返回未定义值 0 假设引脚配置为通过输出，则在引脚上驱动逻辑电平 0。 1 假设引脚配置为通过输出，则在引脚上驱动逻辑电平 1。

```
/*****************************************************************************//*!
* @简介  对输出寄存器写入想要输出的值输出
*
* @无返回
*****************************************************************************/
void GPIO_Write(GPIO_Type *pGPIO, uint32_t u32Value)
{
#if defined(CPU_NV32)
    ASSERT((pGPIO == GPIOA) || (pGPIO == GPIOB));
#endif
#if defined(CPU_NV32M3)
    ASSERT(pGPIO == GPIOA);
#endif
#if defined(CPU_NV32M4)
```

```
    ASSERT((pGPIO == GPIOA) || (pGPIO == GPIOB) || (pGPIO == GPIOC));
#endif

    pGPIO->PDOR = u32Value;      /*配置数据输出寄存器*/

}
```

1.5 单个 IO 电平读取函数

函数名	GPIO_BitRead
函数原形	GPIO_BitRead(GPIO_PinType GPIO_Pin)
功能描述	读取单个 IO 的电平
输入参数	单个 IO 的管脚名
输出参数	无
返回值	无
先决条件	对该 GPIO 设置为输入
函数使用实例	GPIO_BitRead(PTE7);

```
/*****************************************************************************//
* @简介  读取单个 GPIO 的输入电平
*
* @返回单个 IO 的输入电平
*
*****************************************************************************/
uint8_t GPIO_BitRead(GPIO_PinType GPIO_Pin)
{
    ASSERT(GPIO_Pin <= GPIO_PTI7);

  #if defined(CPU_NV32)
    if (GPIO_Pin < GPIO_PTE0)
    {
        if(((1<<GPIO_Pin) & GPIOA->PDIR) > 0)          /*  判断 IO 电平  */
            return 1;                                  /*  电平 1 返回 1 */
        else
                return 0;
    }

    else if (GPIO_Pin < GPIO_PTI0)
    {
      GPIO_Pin = (GPIO_PinType)(GPIO_Pin - 32);

        if(((1<<GPIO_Pin) & GPIOB->PDIR) > 0)          /*  判断 IO 电平  */
            return 1;                                  /*  电平 1 返回 1 */
        else
```

```
                return 0;
        }
#endif

}
```

1.5 单个 IO 的电平翻转函数

函数名	GPIO_PinToggle
函数原形	GPIO_PinToggle(GPIO_PinType GPIO_Pin)
功能描述	对输入参数的单个 IO 进行翻转
输入参数	单个 IO 的管脚名
输出参数	无
返回值	无
先决条件	对该 GPIO 设置为输出
函数使用实例	GPIO_PinToggle(PTE7);

```
/*****************************************************************************//*!
* @简介     单个 IO 的电平翻转
*
* @无返回
*****************************************************************************/
void GPIO_PinToggle(GPIO_PinType GPIO_Pin)
{
    /* Sanity check */
    ASSERT(GPIO_Pin <= GPIO_PTI7);

    if (GPIO_Pin < GPIO_PTE0)
    {
        /*  对单个 GPIO 配置  */
        GPIOA->PTOR = (1<<GPIO_Pin);
    }

#if (defined(CPU_NV32) | defined(CPU_NV32M4))

    else if (GPIO_Pin < GPIO_PTI0)
    {

        GPIO_Pin = (GPIO_PinType)(GPIO_Pin - GPIO_PTE0);
        GPIOB->PTOR = (1<<GPIO_Pin);
    }
#endif
}
```

1.6 单个 IO 输出高电平

函数名	GPIO_PinSet
函数原形	GPIO_PinSet(GPIO_PinType GPIO_Pin)
功能描述	对单个 IO 置 1
输入参数	单个 IO 的管脚名
输出参数	无
返回值	无
先决条件	对该 GPIO 设置为输出
函数使用实例	GPIO_PinSet(PTE7);

字段	描述
PTSO	数据置位输出 如下所示对该寄存器进行写操作将更新 PDOR 中对应位的值： 0 PDORn 中的对应位不发生变化。 1 PDORn 中的对应位置位为逻辑电平 1。

```
/*****************************************************************************//*!
* @简介   单个 IO 输出高电平函数
*
* @无返回
*****************************************************************************/
void GPIO_PinSet(GPIO_PinType GPIO_Pin)
{

    ASSERT(GPIO_Pin <= GPIO_PTI7);

    if (GPIO_Pin < GPIO_PTE0)
    {
        /* PTA0-7, PTB0-7, PTC0-7, PTD0-7 */
        GPIOA->PSOR = (1<<GPIO_Pin);        //配置对应的 IO 输出为高电平
    }

#if (defined(CPU_NV32) | defined(CPU_NV32M4))

    else if (GPIO_Pin < GPIO_PTI0)
    {

        GPIO_Pin = (GPIO_PinType)(GPIO_Pin - GPIO_PTE0);
        GPIOB->PSOR = (1<<GPIO_Pin);        //配置对应的 IO 输出为高电平
    }
#endif

}
```

1.7 单个 IO 输出低电平

函数名	GPIO_PinClear
函数原形	GPIO_PinClear(GPIO_PinType GPIO_Pin)
功能描述	对单个 IO 进行清 0
输入参数	单个 IO 的管脚名
输出参数	无
返回值	无
先决条件	对该 GPIO 设置为输出
函数使用实例	GPIO_PinClear(PTE7);

字段	描述
PTCO	端口清零输出 如下所示对该寄存器进行写操作将更新 PDOR 中对应位的值： 0 PDORn 中的对应位不发生变化。 1 PDORn 中的对应位清零为逻辑电平 0。

```
/*****************************************************************************//*!
* @简介    单个 IO 输出低电平
*
* @无返回
*****************************************************************************/
void GPIO_PinClear(GPIO_PinType GPIO_Pin)
{

    ASSERT(GPIO_Pin <= GPIO_PTI7);

    if (GPIO_Pin < GPIO_PTE0)
    {
        /* PTA0-7, PTB0-7, PTC0-7, PTD0-7 */
        GPIOA->PCOR = (1<<GPIO_Pin);       //配置对应的 IO 输出为低电平
    }

#if (defined(CPU_NV32) | defined(CPU_NV32M4))

    else if (GPIO_Pin < GPIO_PTI0)
    {
        /* PTE0-7, PTF0-7, PTH0-7, PTI0-7 */
        GPIO_Pin = (GPIO_PinType)(GPIO_Pin - GPIO_PTE0);
        GPIOB->PCOR = (1<<GPIO_Pin);            //配置对应的 IO 输出为低电平
    }
#endif

#if defined(CPU_NV32M4)
```

```
    else if(GPIO_Pin < GPIO_PIN_MAX)
    {
        /* PTI0-7 */
        GPIO_Pin = (GPIO_PinType)(GPIO_Pin - GPIO_PTI0);
        GPIOC->PCOR = (1<<GPIO_Pin);          //配置对应的 IO 输出为低电平
    }
#endif
}
```

第二章 样例程序

2.1 使用 RTC 中断来实现 GPIO 电平翻转

```
/*****************************************************************************************************
*
*
* @简介   此例程配置 RTC 固定时间中断，在中断里对 IO 电平取反
*
*
*****************************************************************************************************/

#include "common.h"
#include "ics.h"
#include "rtc.h"
#include "uart.h"
#include "gpio.h"
#include "sysinit.h"
#include "start.h"

int main (void);
void RTC_Task(void);

/*********************************************************************/
int main (void)
{
    /* 系统初始化 */
    sysinit();
    cpu_identify();
```

```
    RTC_ConfigType   sRTCConfig;
    RTC_ConfigType   *pRTCConfig = &sRTCConfig;

    printf("\nRunning the GPIO_demo project.\n");

    /* 初始化 RTC 中断频率 1Hz  */
    pRTCConfig->u16ModuloValue = 9;
    pRTCConfig->bInterruptEn     = RTC_INTERRUPT_ENABLE;      /* 打开 RTC 中断使能 */
    pRTCConfig->bClockSource     = RTC_CLKSRC_1KHZ;           /*时钟源 1khz*/
    pRTCConfig->bClockPresaler = RTC_CLK_PRESCALER_100;     /*分频系数 100*/

    RTC_SetCallback(RTC_Task);          //指向回执函数
    RTC_Init(pRTCConfig);

    /* 方式 1.GPIO 整体配置 */
    GPIO_Init(GPIOB, GPIO_PTE7_MASK, GPIO_PinInput);
    /* 方式 2. GPIO 单 IO 配置 */
    GPIO_PinInit(GPIO_PTE7, GPIO_PinInput);


    while (1);

}
/*****************************************************************************//*!
*
* @简介 RTC 中断回执函数
*
* @无返回
*****************************************************************************/

void RTC_Task(void)
{
    /* 调用 GPIO 翻转函数 */
    GPIO_PinToggle(GPIO_PTE7);
}
/*****************************************************************************/
```