NV32F100x 库模型简介

目录结构

```
├──common
├──cpu
|    └──headers
├──drivers
|    ├──acmp
|    ├──adc
|    ├──bitband
|    ├──bos
|    ├──crc
|    ├──eeprom
|    ├──etm
|    ├──gpio
|    ├──ics
|    ├──iic
|    ├──kbi
|    ├──LemcUSB
|    ├──nvm
|    ├──pit
|    ├──PMC
|    ├──rtc
|    ├──sim
|    ├──spi
|    ├──uart
|    └──wdog
├──lib
├──platforms
├──ports
├──projects
|    └──NV32
```

所有的库文件都在 drives 目录，并且按照模块分成不同的子目录，考虑到很多 STM 的用户，我们把所有的库文件放在 lib 目录，如果需要可以直接调用

cpu/header 包含了 NV32.h，是对整个 MCU 所有模块结构体（对象实例化）的定义。

Common 包含了 M0+的常用的宏定义。

注意

**********static inline **********

内联函数有些类似于宏。内联函数的代码会被直接嵌入在它被调用的地方，调用几次就嵌入几次，没有使用 call 指令。这样省去了函数调用时的一些额外开销，比如保存和恢复函数返回地址等，可以加快速度。不过调用次数多的话，会使可执行文件变大，这样会降低速度。相比起宏来说，内核开发者一般更喜欢使用内联函数。因为内联函数没有长度限制，格式限制。编译器还可以检查函数调用方式，以防止其被误用。

static inline 的内联函数，一般情况下不会产生函数本身的代码，而是全部被嵌入在被调用的地方。如果不加 static，则表示该函数有可能会被其他编译单元所调用，所以一定会产生函数本身的代码。所以加了 static，一般可令可执行文件变小。内核里一般见不到只用 inline 的情况，而都是使用 static inline。

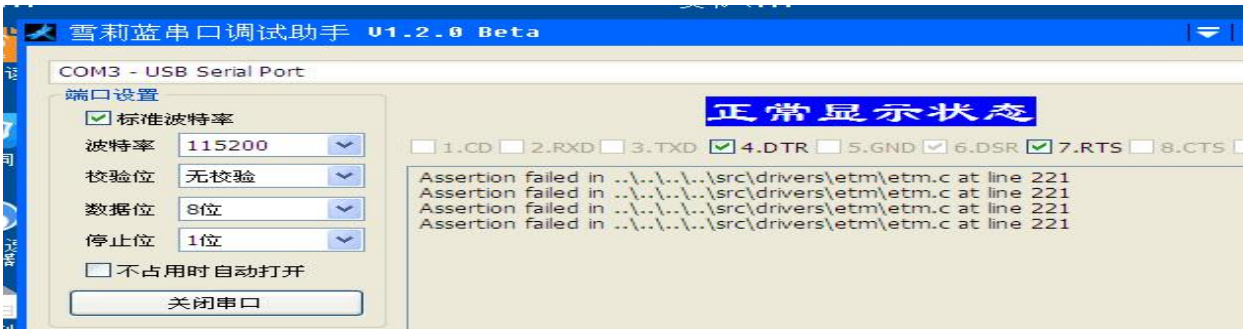
**********static inline **********

**********ASSERT **********

关于 ASSERT 的使用。比如在例程包里的双边沿检测，只可 ETM2 使用

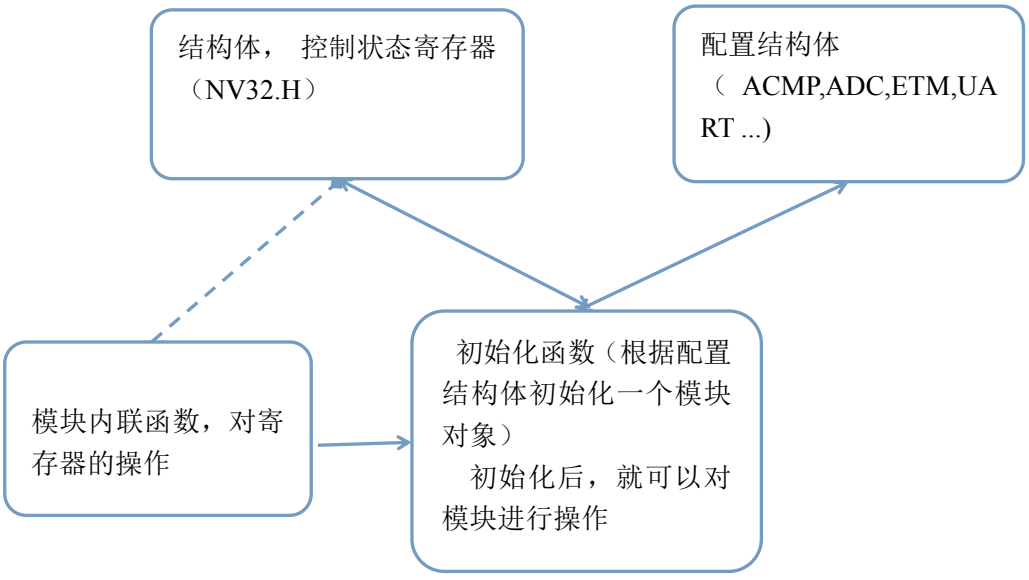
```
ASSERT((ETM2 == pETM)    && (u8ChannelPair < 6) && !(u8ChannelPair & 1) );
```

如果输入 ETM1，程序执行到此死等待，并打印错误行



**********ASSERT **********

基本架构



NV32.h 定义模块结构体，实例化出来模块对象，模块驱动定义了配置结构体和基本的操作方法，在对每个模块操作之前，首先初始化模块，然后才能使能，操作，发送接收数据。

模块结构体定义(NV32.H)

NV32.h 定义了所有模块的结构体，便于大家对模块对象的直接操作，并且以所对应的寄存器实例化了对应模块

以 ACMP 为例

```
/** ACMP - Register Layout Typedef */
typedef struct {
  __IO uint8_t CS;                                 /**状态控制寄存器,offset: 0x1 */
  __IO uint8_t C0;                                 /**控制寄存器 0, offset: 0x1 */
  __IO uint8_t C1;                                 /**控制寄存器 1, offset: 0x2 */
  __IO uint8_t C2;                                 /**控制寄存器 2, offset: 0x3 */
} ACMP_Type;

/* ----------------------------------------------------------------------------
   -- ACMP Register Masks
   --寄存器 mask，便于对寄存器的某个控制位(bit 运算)做运算，置 1，清 0
   --
   ---------------------------------------------------------------------------- */


/* CS Bit Fields */
#define ACMP_CS_ACMOD_MASK                       0x3u
#define ACMP_CS_ACMOD_SHIFT                      0
#define                                                  ACMP_CS_ACMOD(x)
(((uint8_t)(((uint8_t)(x))<<ACMP_CS_ACMOD_SHIFT))&ACMP_CS_ACMOD_MASK)
#define ACMP_CS_ACOPE_MASK                       0x4u
#define ACMP_CS_ACOPE_SHIFT                      2
#define ACMP_CS_ACO_MASK                         0x8u
#define ACMP_CS_ACO_SHIFT                        3
#define ACMP_CS_ACIE_MASK                        0x10u
#define ACMP_CS_ACIE_SHIFT                       4
#define ACMP_CS_ACF_MASK                         0x20u
#define ACMP_CS_ACF_SHIFT                        5
#define ACMP_CS_HYST_MASK                        0x40u
#define ACMP_CS_HYST_SHIFT                       6
#define ACMP_CS_ACE_MASK                         0x80u
#define ACMP_CS_ACE_SHIFT                        7
/* C0 Bit Fields */
#define ACMP_C0_ACNSEL_MASK                      0x3u
#define ACMP_C0_ACNSEL_SHIFT                     0
#define                                                  ACMP_C0_ACNSEL(x)
```

```
(((uint8_t)(((uint8_t)(x))<<ACMP_C0_ACNSEL_SHIFT))&ACMP_C0_ACNSEL_MASK)
    #define ACMP_C0_ACPSEL_MASK                      0x30u
    #define ACMP_C0_ACPSEL_SHIFT                     4
    #define                                          ACMP_C0_ACPSEL(x)
(((uint8_t)(((uint8_t)(x))<<ACMP_C0_ACPSEL_SHIFT))&ACMP_C0_ACPSEL_MASK)
    /* C1 Bit Fields */
    #define ACMP_C1_DACVAL_MASK                      0x3Fu
    #define ACMP_C1_DACVAL_SHIFT                     0
    #define                                          ACMP_C1_DACVAL(x)
(((uint8_t)(((uint8_t)(x))<<ACMP_C1_DACVAL_SHIFT))&ACMP_C1_DACVAL_MASK)
    #define ACMP_C1_DACREF_MASK                      0x40u
    #define ACMP_C1_DACREF_SHIFT                     6
    #define ACMP_C1_DACEN_MASK                       0x80u
    #define ACMP_C1_DACEN_SHIFT                      7
    /* C2 Bit Fields */
    #define ACMP_C2_ACIPE_MASK                       0x7u
    #define ACMP_C2_ACIPE_SHIFT                      0
    #define                                          ACMP_C2_ACIPE(x)
(((uint8_t)(((uint8_t)(x))<<ACMP_C2_ACIPE_SHIFT))&ACMP_C2_ACIPE_MASK)
```

```
    /*以 ACMP0，1 的地址实例化 2 个 ACMP：ACMP0,ACMP1 */
    /** Peripheral ACMP0 base address */
    #define ACMP0_BASE                               (0x40073000u)
    /** Peripheral ACMP0 base pointer */
    #define ACMP0                                    ((ACMP_Type *)ACMP0_BASE)
    /** Peripheral ACMP1 base address */
    #define ACMP1_BASE                               (0x40074000u)
    /** Peripheral ACMP1 base pointer */
    #define ACMP1                                    ((ACMP_Type *)ACMP1_BASE)
    /** Array initializer of ACMP peripheral base pointers */
    #define ACMP_BASES                               { ACMP0, ACMP1 }
```

模块配置结构体以及库函数

以 ACMP 为例，acmp.h 包含了对模块配置结构体，这个结构体和模块结构体不同，它主要对应的是控制寄存器

```
    /*  比较器状态控制寄存器  */
    typedef union
    {
        uint8_t byte;                /*!< byte field of union type */
        struct
```

```
    {
        uint8_t bMod          : 2;      /*!< Sensitivity modes of the interrupt trigger */
        uint8_t bOutEn        : 1;      /*!< Output can be placed onto an external pin */
        uint8_t bOutState     : 1;      /*!< The current value of the analog comparator output */
        uint8_t bIntEn        : 1;      /*!< ACMP interrupt enable */
        uint8_t bIntFlag      : 1;      /*!< ACMP Interrupt Flag Bit */
        uint8_t bHyst         : 1;      /*!< Selects ACMP hystersis */
        uint8_t bEn           : 1;      /*!< Enables the ACMP module */
    }bits;                              /*!< bitfield of union type */
}ACMP_CtrlStatusType, *ACMP_CtrlStatusPtr;

                                        */

/*******************************************************************************
*
* ACMP 比较器 pin 输入选择联合体
*
*******************************************************************************/
typedef union
{
    uint8_t byte;                       /*!< byte field of union type */
    struct
    {
        uint8_t bNegPin   : 2;          /*!< Negative pin select */
        uint8_t           : 2;
        uint8_t bPosPin   : 2;          /*!< Positive pin select */
        uint8_t           : 2;
    }bits;                              /*!< bitfield of union type */
}ACMP_PinSelType, *ACMP_PinSelPtr;



/*******************************************************************************
* DAC 控制联合体
* ACMP DAC control struct
*
*******************************************************************************/
typedef union
{
    uint8_t byte;                       /*!< byte field of union type */
    struct
    {
        uint8_t bVal   : 6;             /*!< 6 bit DAC value */
        uint8_t bRef   : 1;             /*!< 6 bit DAC reference select */
```

```
        uint8_t bEn     : 1;                /*!< 6 bit DAC enable bit */
    }bits;                                  /*!< bitfield of union type */
}ACMP_DACType, *ACMP_DACPtr;          /*!< ACMP DAC control structure */
                                            */

/*******************************************************************************
*
* ACMP 外部管脚使能联合体
*
*******************************************************************************/
typedef union
{
    uint8_t byte;                           /*!< byte field of union type */
    struct
    {
        uint8_t bEn     : 3;                /*!< ACMP external input pin enable */
        uint8_t bRsvd : 5;
    }bits;                                  /*!< bitfield of union type */
}ACMP_PinEnType, *ACMP_PinEnPtr;       /*!< ACMP Pin enable structure */


/*******************************************************************************
*
* ACMP 配置结构体
*
*******************************************************************************/

typedef struct
{
    ACMP_CtrlStatusType   sCtrlStatus;      /*!< ACMP control and status */
    ACMP_PinSelType       sPinSelect;       /*!< ACMP pin select */
    ACMP_DACType          sDacSet;          /*!< ACMP internal dac set */
    ACMP_PinEnType        sPinEnable;       /*!< ACMP external pin control */
}ACMP_ConfigType, *ACMP_ConfigPtr;
```

库函数以及内联函数

NV32 对每个模块都提供了丰富的库函数，能够满足很多场合应用，模块初始化，发送数据，获取数据，使能中断，关闭中断。

首先是初始化，初始化一般是设置工作模式，工作时钟，中断使能，这个我们可以通过查阅配置结构体来配置模块。用户需要实现的是获取或者发送数据，中断服务处理函数。

对寄存器操作的内联函数，库中定义了，几乎所有对寄存器位操作的内联函数，调用内联函数不会有跳转，相当于直接对寄存器的操作,以 ACMP 为例:

```
void ACMP_InputPinDisable(ACMP_Type *pACMPx, uint8_t u8InputPin)
//比较器输入使能
void ACMP_DacOutputSet(ACMP_Type *pACMPx, uint8_t u8DacValue)
//比较器内部 DAC 输出电压配置
Void ACMP_DacReferenceSelect(ACMP_Type *pACMPx, uint8_t u8RefSelect)
//内部 DAC 参考电压配置
void ACMP_DacDisable(ACMP_Type *pACMPx)
//禁止内部 DAC
void ACMP_DacEnable(ACMP_Type *pACMPx)
//使能内部 DAC
void ACMP_NegativeInputSelect(ACMP_Type *pACMPx, uint8_t u8NegPinSel)
//比较器负极输入配置
void ACMP_PositiveInputSelect(ACMP_Type *pACMPx, uint8_t u8PosPinSel)
//比较器正极输入配置
void ACMP_ClrFlag(ACMP_Type *pACMPx)
//清比较器中断标识位
uint8_t ACMP_GetFlag(ACMP_Type *pACMPx)
//获取比较器中断标识位
void ACMP_DisableInterrupt(ACMP_Type *pACMPx)
//禁止比较器中断
void ACMP_EnableInterrupt(ACMP_Type *pACMPx)
//使能比较器中断
void ACMP_SelectHyst(ACMP_Type *pACMPx, uint8_t u8HystSelect)
//设置比较器迟滞
void ACMP_DisablePinOut(ACMP_Type *pACMPx)
//禁止比较器结果输出
void ACMP_EnablePinOut(ACMP_Type *pACMPx)
//使能比较器结果输出
void ACMP_SelectIntMode(ACMP_Type *pACMPx, uint8_t u8EdgeSelect)
//设置触发比较器中断模式
void ACMP_Disable(ACMP_Type *pACMPx)
//禁止比较器
void ACMP_Enable(ACMP_Type *pACMPx)
//使能比较器
```

中断机制

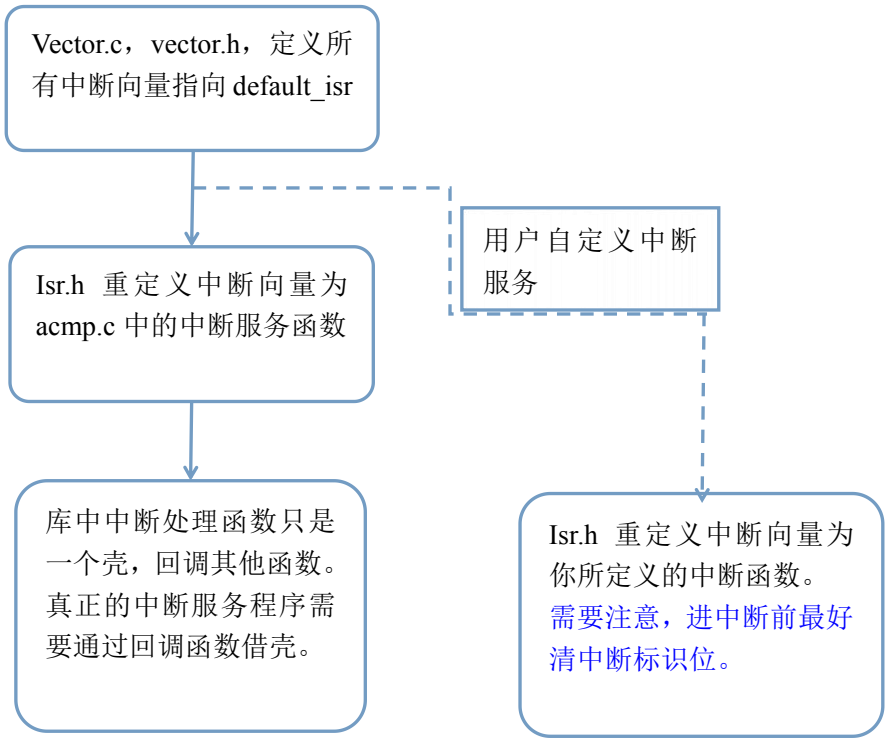
我们所有的 demo 中断采用回调函数，这样的好处是中断函数完全由用户定义，只需要把中断处理函数

交给回调就可以，中断函数我们在库中统一定义，我们的目的是方便初级用户。当然用户也完全可以自己定义中断处理。

```
/*ISR.H  首先重定义中断向量所指的函数*/
#undef    VECTOR_032
#define VECTOR_032 ACMP0_Isr

#undef    VECTOR_037
#define VECTOR_037 ACMP1_Isr

extern void ACMP0_Isr(void);
extern void ACMP1_Isr(void);
```




```
/*ISR.H  用户也可以自定义中断服务函数,而不使用回调*/
#undef    VECTOR_032
#define VECTOR_032 ACMP0_UserDefine_Isr

#undef    VECTOR_037
#define VECTOR_037 ACMP1_UserDefine_Isr

extern void ACMP0_UserDefine_Isr(void);
extern void ACMP1_UserDefine_Isr(void);
```