

NV32F100x ETM 定时器模块编程示例

第一章 特性简介

1.1 ETM 时钟及分频系数的选择

状态及控制寄存器 ETMx_SC 的具体信息见参考手册 6.3.7.3

ETM 的时钟源信号选择主要是根据状态和控制寄存器 ETMx_SC 中 CLKS 的两位进行的。

| | |
|-------------|---|
| 4-3 CLKS | 时钟源选择 从三个 ETM 计数器时钟源选择。 该字段为写保护。仅当 MODE[WPDIS]=1 时，才能写操作。 00 未选定时钟，该设置生效后可禁用 ETM 计数器。 01 系统时钟 10 固定频率时钟 11 外部时钟 |
|-------------|---|

而 ETM 计数器的时钟源是通过分频后再接入，分频控制由状态和控制寄存器 ETMx_SC 的 PS 三位来进行选择。

| | | | | | | | | | | | | | | | | | |
|-----------|--|-----|------|-----|------|-----|------|-----|------|-----|-------|-----|-------|-----|-------|-----|--------|
| 2-0 PS | 预分频系数选择 从 8 个分频系数中选择，用于 CLKS 所选择的时钟源。新的预分频系数将会在新数值更新至寄存器位后，于下一个系统时钟周期影响时钟源。 该字段为写保护。仅当 MODE[WPDIS]=1 时，才能写操作。 | | | | | | | | | | | | | | | | |
| | <table><tr><td>000</td><td>1 分频</td></tr><tr><td>001</td><td>2 分频</td></tr><tr><td>010</td><td>4 分频</td></tr><tr><td>011</td><td>8 分频</td></tr><tr><td>100</td><td>16 分频</td></tr><tr><td>101</td><td>32 分频</td></tr><tr><td>110</td><td>64 分频</td></tr><tr><td>111</td><td>128 分频</td></tr></table> | 000 | 1 分频 | 001 | 2 分频 | 010 | 4 分频 | 011 | 8 分频 | 100 | 16 分频 | 101 | 32 分频 | 110 | 64 分频 | 111 | 128 分频 |
| 000 | 1 分频 | | | | | | | | | | | | | | | | |
| 001 | 2 分频 | | | | | | | | | | | | | | | | |
| 010 | 4 分频 | | | | | | | | | | | | | | | | |
| 011 | 8 分频 | | | | | | | | | | | | | | | | |
| 100 | 16 分频 | | | | | | | | | | | | | | | | |
| 101 | 32 分频 | | | | | | | | | | | | | | | | |
| 110 | 64 分频 | | | | | | | | | | | | | | | | |
| 111 | 128 分频 | | | | | | | | | | | | | | | | |

```
/******//*!  
*  
* @设置时钟资源及分频系数  
* @输入    pETM          指向三个 ETM 定时器其中一个的基址  
* @输入    ClockSource    ETM 时钟资源  
* @输入    ClockPrescale  分频系数  
*  
* @return none.  无返回  
*  
*****//
```

```

void ETM_ClockSet(ETM_Type *pETM, uint8_t u8ClockSource, uint8_t u8ClockPrescale)
{
    uint8_t    u8Temp;
    //pETM 指向的 SC 寄存器低 5 位清 0，即未选择时钟，时钟输入采取 1 分频
    u8Temp = (pETM->SC & 0xE0);
    //时钟选择，及预分频因子选择
    u8Temp |= (ETM_SC_CLKS(u8ClockSource & 0x3) | ETM_SC_PS(u8ClockPrescale & 0x7));
    //配置该 ETM 的状态与控制寄存器 ETMx_SC
    pETM->SC = u8Temp;
}

```

1.2 TOF 频率设置

NUMTOF 是配置寄存器 ETMx_CONF 的后五位标志位，详情见参考手册 6.3.7.21

| | |
|--------|---|
| 4-0 | TOF 频率 |
| NUMTOF | <p>选择计数器溢出数量与 TOF 位置次数之比。</p> <p>NUMTOF=0: TOF 位针对每一位计数器溢出进行置位。</p> <p>NUMTOF=1: TOF 位针对每一位计数器溢出进行置位，但不会对下一个溢出置位。</p> <p>NUMTOF=2: TOF 位针对每一位计数器溢出进行置位，但不对之后的两个溢出置位。</p> <p>NUMTOF=3: TOF 位针对每一位计数器溢出进行置位，但不对之后的三个溢出置位。</p> <p>以此类推，此模式最多可延续到 31 次。</p> |

```

/*****

```

```

* @ETM 中 TOF 频率大小的设置功能函数

```

```

*

```

```

* @输入    pETM          指向三个 ETM 定时器其中一个的基址

```

```

* @输入    u8TOFNUM      TOF 频率数，大小 0 和 31 之间

```

```

*

```

```

* @无返回

```

```

*

```

```

*****/

```

```

void ETM_SetTOFFrequency(ETM_Type *pETM, uint8_t u8TOFNUM)
{
    ASSERT((ETM2 == pETM));
    pETM->CONF &= ~ETM_CONF_NUMTOF_MASK;
    pETM->CONF |= ETM_CONF_NUMTOF(u8TOFNUM);
}

```

1.3 脉冲宽度调制（PWM）

1.3.1 PWM 初始化函数

常用配置为边沿对齐模式的 PWM 即 EPWM，时钟信号为系统时钟，其他为初始化值

| DECAPEN | COMBINE | CPWMS | MSnB: MsnA | ELSnB: ELSnA | 模式 | 配置 |
|---------|---------|-------|------------|--------------|----------------------------------|-------------------------------|
| X | X | X | XX | 0 | 引脚未用于 ETM—通道引脚变回通用 I/O 或其他外设控制用途 | |
| 0 | 0 | 0 | 0 | 1 | 输入捕捉 | 仅在上升沿捕捉 |
| | | | | 10 | | 仅在下降沿捕捉 |
| | | | | 11 | | 在上升沿或下降沿捕捉 |
| | | | 1 | 1 | 输出比较 | 匹配时切换输出 |
| | | | | 10 | | 匹配时清零输出 |
| | | | | 11 | | 匹配时置位输出 |
| | | | 1X | 10 | 边沿对齐 PWM | 高电平真脉冲（匹配时清零输出） |
| | | | | X1 | | 低真脉冲（匹配时置位输出） |
| | | 1 | XX | 10 | 中心对齐 PWM | 高真脉冲（向上匹配时清零输出） |
| | | | | X1 | | 低真脉冲（向上匹配时置位输出） |
| | 1 | 0 | XX | 10 | 组合 PWM | 高真脉冲（通道（n）匹配时置位，通道（n+1）匹配时清零） |
| | | | | X1 | | 低真脉冲（通道（n）匹配时清零，通道（n+1）匹配时置位） |
| 1 | 0 | 0 | X0 | 参见下表 | 双沿捕捉 | 单次捕捉模式 |
| | | | X1 | | | 持续捕捉模式 |

```

/*****
* @ETM 中 PWM 的初始化函数
* @输入      pETM          指向三个 ETM 定时器其中一个的基址
* @输入      PWMModeSelect 居中对齐 CPWM（10）、边沿对齐 EPWM（01）以及级联模式 PWM（11）
* @输入      PWMEdgeSelect 边沿选择
*
* @无返回
*
*****/

```

```
void ETM_PWMInit(ETM_Type *pETM, uint8_t u8PWMModeSelect, uint8_t u8PWMEdgeSelect)
{
    uint8_t channels, i;
    ASSERT((ETM0== pETM) || (ETM1== pETM) || (ETM2== pETM)); //断言来检测 ETM 是否正确
    /* open the clock gate */
    if (ETM0 == pETM)
    {
        channels = 2;
        SIM->SCGC |= SIM_SCGC_ETM0_MASK;
    }
    else if (ETM1 == pETM)
    {
        channels = 2;
    }
    else
    {
        channels = 6;
        SIM->SCGC |= SIM_SCGC_ETM2_MASK;
    }
    pETM->SC = 0x0; /* 关闭计数器 */
    pETM->MOD = ETM_MOD_INIT;
    if (ETM_PWMMODE_CENTERALLIGNED == u8PWMModeSelect) /*打开 CPWM*/
    {
        pETM->SC |= ETM_SC_CPWMS_MASK;
    }
    else if (ETM_PWMMODE_COMBINE == u8PWMModeSelect) /* 打开级联 PWM 模式 */
    {
        ASSERT(ETM2 == pETM);
        pETM->MODE |= ETM_MODE_WPDIS_MASK | ETM_MODE_ETMEN_MASK;
        pETM->COMBINE=ETM_COMBINE_COMBINE0_MASK|ETM_COMBINE_COMPO_MAS|ETM_COMBINE_SYNCENO_MAS|ETM_COMBINE_DTENO_MASK|ETM_COMBINE_COMBINE1_MASK|ETM_COMBINE_COMP1_MASK|ETM_COMBINE_SYNCEN1_MAS|ETM_COMBINE_DTEN1_MASK|ETM_COMBINE_COMBINE2_MASK|ETM_COMBINE_COMP2_MASK|ETM_COMBINE_SYNCEN2_MASK|ETM_COMBINE_DTEN2_MASK ;
        pETM->SC &= ~ETM_SC_CPWMS_MASK;
    }
    if (ETM_PWM_HIGHTRUEPULSE == u8PWMEdgeSelect)
    {
        /* 配置 ETMers 寄存器 */
        for (i=0; i<channels; i++)
        {
            pETM->CONTROLS[i].CnSC = ETM_CnSC_MSB_MASK | ETM_CnSC_ELSB_MASK;
            pETM->CONTROLS[i].CnV = ETM_COV_INIT + i*100;
        }
    }
}
```

```

else if (ETM_PWM_LOWTRUEPULSE == u8PWMEdgeSelect)
{
    for (i=0; i<channels; i++)
    {
        pETM->CONTROLS[i].CnSC = ETM_CnSC_MSB_MASK | ETM_CnSC_ELSA_MASK;
        pETM->CONTROLS[i].CnV  = ETM_COV_INIT + i*100 ;
    }
}
}

```

1.3.2 通道输出极性的设置

ETMx_POL 通道极性寄存器的参数见参考手册 6.3.7.17

```

/*****
*
* @设置通道输出极性的功能函数
*
* @输入      pETM          指向三个 ETM 定时器其中一个的基址
* @输入      Channel      PWM 波的通道选择
* @输入      ActiveValue   极性的选择, 0 为高电平, 1 为低电平
*
* @无返回
*
*****/
void ETM_PolaritySet(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8ActiveValue)
{
    ASSERT((ETM2 == pETM) && (u8ETM_Channel < 6));

    if (ETM_POLARITY_HIGHACTIVE == u8ActiveValue)
    {
        pETM->POL &= ~(1 << u8ETM_Channel);
    }
    else if (ETM_POLARITY_LOWACTIVE == u8ActiveValue)
    {
        pETM->POL |= (1 << u8ETM_Channel);
    }
}

```

1.3.3 配置级联模式下的参数，如占空比

```

/*****
*
*      必须设置奇数通道数，且偶数通道的值不变
*
* @输入      pETM      指向三个 ETM 定时器其中一个的基址
* @输入      ETM_Channel 奇通道数：1、3、5
* @输入      dutyCycle   设置占空比，若 DutyCycle 为 10, 那么占空比就为 10%
*
* @return none.
*
* @ Pass/ Fail criteria: none.
*
*****/
void ETM_SetDutyCycleCombine(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8DutyCycle)
{
    uint16_t cnv = pETM->CONTROLS[u8ETM_Channel-1].CnV;
    uint16_t modulo = pETM->MOD;

    ASSERT((1 == u8ETM_Channel) || (3 == u8ETM_Channel) || (5 == u8ETM_Channel));

    cnv += (u8DutyCycle * (modulo+1)) / 100;
    if(cnv > modulo)
    {
        cnv = modulo - 1;
    }
    pETM->CONTROLS[u8ETM_Channel].CnV = cnv ;

    pETM->PWMLoad |= ETM_PWMLoad_LDOK_MASK | (1<<u8ETM_Channel);
}

```

1.4 ETM-输入捕捉

ETM 工作在输入捕捉模式时，对边沿信号十分的敏感，模式选择由 ETMx_CnSC 寄存器的 ELSnB:ELSnA 两位控制位来进行选择，如下表：

| ELSnB | ELSnA | 通道端口使能 | 已检测边沿 |
|-------|-------|--------|---------|
| 0 | 0 | 禁用 | 无边沿 |
| 0 | 1 | 使能 | 上升沿 |
| 1 | 0 | 使能 | 下降沿 |
| 1 | 1 | 使能 | 上升沿和下降沿 |

1.4.1 输入捕捉初始化

ETM 输入捕捉初始化函数：定义用 ETM 中的哪一个定时器，用对应的哪个通道。以及输入捕捉用上升沿、下降沿、还是双边沿捕捉。

```

/*****
*
* @输入捕捉初始化函数
*
* @输入      pETM          指向三个 ETM 定时器其中一个的基址
* @输入      Channel       配置通道号
* @输入      CaptureMode   选择捕捉方式：上升沿，下降沿或跳变沿。
*
* @无返回
*
*****/
void ETM_InputCaptureInit(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8CaptureMode)
{
    ASSERT(((ETM0 == pETM) && (u8ETM_Channel < 2)) || ((ETM1 == pETM) && (u8ETM_Channel < 2))
    || ((ETM2 == pETM) && (u8ETM_Channel < 6)));
    /* open the clock gate */
    if ((ETM0 == pETM) && (u8ETM_Channel < 2))
    {
        SIM->SCGC |= SIM_SCGC_ETM0_MASK;
        NVIC_EnableIRQ(ETM0_IRQn);
    }
    else if((ETM1 == pETM) && (u8ETM_Channel < 2))
    {
    }
    else
    {
    }
}

```



```

SIM->SCGC |= SIM_SCGC_ETM2_MASK;
NVIC_EnableIRQ(ETM2_IRQn);
}
pETM->SC = 0x0;    //关闭计数器/
pETM->MOD = 0xFFFF;
if(ETM_INPUTCAPTURE_RISINGEDGE == u8CaptureMode) //开启中断，捕获上升沿
{
    pETM->CONTROLS[u8ETM_Channel].CnSC = ETM_CnSC_CHIE_MASK | ETM_CnSC_ELSA_MASK;
}
else if(ETM_INPUTCAPTURE_FALLINGEDGE == u8CaptureMode) //捕获下降沿
{
    pETM->CONTROLS[u8ETM_Channel].CnSC = ETM_CnSC_CHIE_MASK | ETM_CnSC_ELSB_MASK;
}
else if(ETM_INPUTCAPTURE_BOTHEDGE == u8CaptureMode) //接受跳变沿
{
    pETM->CONTROLS[u8ETM_Channel].CnSC = ETM_CnSC_CHIE_MASK | ETM_CnSC_ELSA_MASK |
    ETM_CnSC_ELSB_MASK;
}
}
}

```

1.4.2 ETM 双边沿捕获初始化函数

```

/*****/
*
* @对 ETM_No t 配置双边捕获模式来测量一个脉冲的宽度或周期
*
* @输入      pETM      指向三个 ETM 定时器其中一个的基址
* @输入      ChannelPair  频道配对数的配置为：0, 2, 4.
* @输入      CaptureMode  select capture edge: one shot and continuous mode.
* @输入      Channel_N_Edge  频道 N 边沿检测
* @输入      Channel_Np1_Edge  频道 N+1 边沿检测.
*
* @无返回
*
*****/
void ETM_DualEdgeCaptureInit(ETM_Type *pETM, uint8_t u8ChannelPair, uint8_t u8CaptureMode,
                             uint8_t u8Channel_N_Edge, uint8_t u8Channel_Np1_Edge)
{
    ASSERT((ETM2 == pETM) && (u8ChannelPair < 6) && !(u8ChannelPair & 1));

    SIM->SCGC |= SIM_SCGC_ETM2_MASK;
    if((0 == u8ChannelPair) || (2 == u8ChannelPair))
    {

```

```

/* channel filter is active */
}

pETM->SC      = 0x0;          /* diable counter */
pETM->MOD      = 0xFFFF;
pETM->MODE |= ETM_MODE_ETMEN_MASK; /* ETMEN = 1 */
/* DECAPEN = 1, ChannelPair/2 * 8 */
pETM->COMBINE |= ((ETM_COMBINE_DECAPENO_MASK) << (u8ChannelPair * 4));

pETM->CONTROLS[u8ChannelPair].CnSC &= ~ETM_CnSC_CHF_MASK; /* CH(n)F and CH(n+1)F
bits must be cleared first */
pETM->CONTROLS[u8ChannelPair + 1].CnSC &= ~ETM_CnSC_CHF_MASK;

if(ETM_INPUTCAPTURE_DUALEDGE_ONESHOT == u8CaptureMode) /* oneshot mode */
{
    pETM->CONTROLS[u8ChannelPair].CnSC &= ~ETM_CnSC_MSA_MASK;
    pETM->CONTROLS[u8ChannelPair+1].CnSC &= ~ETM_CnSC_MSA_MASK;
}
else if(ETM_INPUTCAPTURE_DUALEDGE_CONTINUOUS == u8CaptureMode) /* continuouse mode */
{
    pETM->CONTROLS[u8ChannelPair].CnSC |= ETM_CnSC_MSA_MASK;
    pETM->CONTROLS[u8ChannelPair+1].CnSC |= ETM_CnSC_MSA_MASK;
}

pETM->CONTROLS[u8ChannelPair].CnSC |= (u8Channel_N_Edge << 2); /* select detec edge
*/
pETM->CONTROLS[u8ChannelPair + 1].CnSC |= (u8Channel_Np1_Edge << 2);

pETM->COMBINE |= (ETM_COMBINE_DECAPO_MASK << (u8ChannelPair * 4));
}

```

1.5 输出对比初始化函数

在输出比较模式下，ETM 可以通过编程生成特定的定时脉冲，当计数器的值匹配到 CnV 的值时，输出比较通道 n 可以做翻转、置位、清 0 这三个操作。

/******//*/

*

* @用 ETM 完成捕捉

*

* @ 输入 pETM 指向三个 ETM 定时器其中一个的基址

* @输入 Channel 必须完成配置通道即通道号

* @ 输入 CompareMode 选择模式：翻转（01）、置位（11）、清 0（10）

```

*
* @无返回
*
*****/
void ETM_OutputCompareInit(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8CompareMode)
{
    ASSERT(((ETM0 == pETM) && (u8ETM_Channel < 2)) ||
           ((ETM1 == pETM) && (u8ETM_Channel < 2)) ||
           ((ETM2 == pETM) && (u8ETM_Channel < 6))
           );
    /* open the clock gate */
    if(ETM0 == pETM)
    {
        SIM->SCGC |= SIM_SCGC_ETM0_MASK;
    }
    else if(ETM1 == pETM)
    {
        SIM->SCGC |= SIM_SCGC_ETM1_MASK;
#endif
    }
    else
    {
        SIM->SCGC |= SIM_SCGC_ETM2_MASK;
    }
    pETM->SC = 0x0; //关闭计数器
    pETM->MOD = ETM_MOD_INIT;
    pETM->CONTROLS[u8ETM_Channel].CnSC = (ETM_CnSC_MSA_MASK | (u8CompareMode<<2));
    pETM->CONTROLS[u8ETM_Channel].CnV = ETM_COV_INIT;
}

```

1.6 同步设置寄存器 ETMx_SYNCONF 的配置

具体信息见参考手册 6.3.7.11

1.6.1 整体配置

```

/*****
*
* @配置寄存器 ETMx_SYNCONF, 其中里面包含了软件输出的控制是否由硬件触发 HW 或是否有软件出发 SW
* @param[in]    pETM          指向三个 ETM 定时器其中一个的基址
* @param[in]    u32ConfigValue ETMx_SYNCONF 这个寄存器的值

```

```

*
* @无返回
*
*****/
void ETM_SyncConfigDeactivate(ETM_Type *pETM, uint32_t u32ConfigValue)
{
    ASSERT((ETM2 == pETM));
    pETM->SYNCONF &= ~u32ConfigValue;
}

```

1.6.2 保证软件同步

```

/*****/
* @ETM 中配置 ETMx_SYNC 寄存器来保证软件同步
*
* @输入      pETM      指向三个 ETM 定时器其中一个的基址
* @输入      u8TriggerN 选择硬件触发资源
*
* @无返回
*
*****/
void ETM_HardwareSync(ETM_Type *pETM, uint8_t u8TriggerN)
{
    ASSERT(ETM2 == pETM);

    pETM->SYNCONF |= ETM_SYNCONF_SYNCMODE_MASK;

    switch(u8TriggerN)
    {
        case ETM_SYNC_TRIGGER_TRIGGER2:
            pETM->SYNC |= ETM_SYNC_TRIG2_MASK;
            break;
        case ETM_SYNC_TRIGGER_TRIGGER1:
            pETM->SYNC |= ETM_SYNC_TRIG1_MASK;
            break;
        case ETM_SYNC_TRIGGER_TRIGGER0:
            pETM->SYNC |= ETM_SYNC_TRIG0_MASK;
            break;
        default:
            break;
    }
}

```

1. 6. 3 软件输出控制同步触发选择

SWOCTRL 寄存器，在同步设置寄存器 ETMx_SYNCONF 寄存器中。

```

/*****
*
* @配置软件输出控制 SWOCTRL 寄存器的同步是否由软件触发
*
* @输入      pETM          指向三个 ETM 定时器其中一个的基址
* @输入      Channel       PWM 波的通道选择
* @输入      ChannelValue  0 或 1, 0 不触发; 1 触发
*
* @无返回
*
*****/
void ETM_SWOutputControlSet(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8ChannelValue)
{
    ASSERT((ETM2 == pETM) && (u8ETM_Channel < 6));

    if(ETM_SWOCTRL_HIGH == u8ChannelValue)
    {
        pETM->SWOCTRL |= (0x0101 << u8ETM_Channel);
    }
    else if(ETM_SWOCTRL_LOW == u8ChannelValue)
    {
        pETM->SWOCTRL |= (1 << u8ETM_Channel);
        pETM->SWOCTRL &= ~(0x100 << u8ETM_Channel);
    }
    if(pETM->SYNCONF & ETM_SYNCONF_SWOC_MASK)
    {
        pETM->SYNCONF |= ETM_SYNCONF_SYNCMODE_MASK;
        if(pETM->SYNCONF & ETM_SYNCONF_SWSOC_MASK)
        {
            pETM->SYNC |= ETM_SYNC_SWSYNC_MASK;
        }
        else if(pETM->SYNCONF & ETM_SYNCONF_HWSOC_MASK)
        {
            pETM->SYNC |= ETM_SYNC_TRIG2_MASK;
        }
    }

#ifdef CPU_NV32
    SIM->SOPT |= SIM_SOPT_ETMSYNC_MASK;
#endif
}

```

```

else
{
}
}

```

1.6.4 通过配置 ETM 保证硬件同步，产生触发

```

/*****
*
* @通过配置 ETM 保证硬件同步，产生触发
*
* @输入   pETM           指向三个 ETM 定时器其中一个的基址
*@输入   u8TriggerMask   选择硬件触发资源. combine TRIG0~TREG2. (x000xxxx~x111xxx)
*
* @无返回.
*
*****/
void ETM_HardwareSyncCombine(ETM_Type *pETM, uint8_t u8TriggerMask)
{
    ASSERT(ETM2 == pETM);

    pETM->SYNCONF |= ETM_SYNCONF_SYNCMODE_MASK;
    pETM->SYNC    &= 0x8F;
    pETM->SYNC    |= (u8TriggerMask & 0x70);
}

```

1.7 ETM 的基本操作

1.7.1 初始化 ETM

```

/*****
*
* @ETM 初始化函数
*
* @输入   pETM           指向三个 ETM 定时器其中一个的基址
* @输入   pConfig        指向 ETM 的一些基本参数
* @无返回值
*
*****/
void ETM_Init(ETM_Type *pETM, ETM_ConfigType *pConfig)

```

```
{
    ASSERT((ETM0 == pETM) || (ETM1 == pETM) || (ETM2 == pETM));
    if(ETM0 == pETM)
    {
        SIM->SCGC |= SIM_SCGC_ETMO_MASK;
    }
    else
    {
        SIM->SCGC |= SIM_SCGC_ETM2_MASK;
    }

    /* disable counter */
    pETM->SC = 0;
    pETM->MODE = pConfig->mode;
    pETM->MOD = pConfig->modulo;
    pETM->CNT = pConfig->cnt;

    if( pETM->MODE & ETM_MODE_ETMEN_MASK )
    {
        /* when ETMEN = 1, all other registers can be written */
        pETM->COMBINE = pConfig->combine;
        pETM->CNTIN = pConfig->cntin;
        pETM->SYNC = pConfig->sync;
        pETM->OUTINIT = pConfig->outinit;
        pETM->OUTMASK = pConfig->outmask;
        pETM->DEADETME = pConfig->deadETMe;
        pETM->EXTTRIG = pConfig->exttrig;
        pETM->POL = pConfig->pol;
        pETM->FMS = pConfig->fms;
        pETM->FILTER = pConfig->filter;
        pETM->FLTCTRL = pConfig->fltctrl;
        pETM->FLTPOL = pConfig->fltpol;
        pETM->CONF = pConfig->conf;
        pETM->SYNCONF = pConfig->synconf;
        pETM->SWOCTRL = pConfig->swoctrl;
        pETM->PWMLoad = pConfig->pwmload;
    }
    /* write SC to enable clock */
    pETM->SC = pConfig->sc;
}
```

1.7.2 关闭 ETM 组件

```
/******
```

```
关闭相应的 ETM 功能组件函数
```

```
******/
```

```
void ETM_DeInit(ETM_Type *pETM)
{
    ASSERT((ETM0 == pETM) || (ETM1 == pETM) || (ETM2 == pETM));
    pETM->SC = 0;
    pETM->MOD = 0;
    pETM->CNT = 0;
    if(ETM2 == pETM)
    {
        pETM->MODE = 0x4;
        pETM->COMBINE = 0;
        pETM->CNTIN = 0;
        pETM->SYNC = 0;
        pETM->OUTINIT = 0;
        pETM->OUTMASK = 0;
        pETM->DEADTIME = 0;
        pETM->EXTTRIG = 0;
        pETM->POL = 0;
        pETM->FMS = 0;
        pETM->FILTER = 0;
        pETM->FLTCTRL = 0;
        pETM->FLTPOL = 0;
        pETM->CONF = 0;
        pETM->SYNCONF = 0;
        pETM->SWOCTRL = 0;
        pETM->PWMLOAD = 0;
    }
    if (ETM0 == pETM)
    {
        SIM->SCGC &= ~SIM_SCGC_ETM0_MASK;
        NVIC_DisableIRQ(ETM0_IRQn);
    }
    #if !defined(CPU_KE04)
    else if(ETM1 == pETM)
    {
        SIM->SCGC &= ~SIM_SCGC_ETM1_MASK;
        NVIC_DisableIRQ(ETM1_IRQn);
    }
    #endif
    else if (ETM2 == pETM)
```



```

SIM->SCGC &= ~SIM_SCGC_ETM2_MASK;
    NVIC_DisableIRQ(ETM2_IRQn);
}
}

```

1.7.3 由 ETM2 产生的硬件触发

```

/*****//*!
*
* @产生 ETM2 硬件触发, Note: please call ETM_HardwareSyncCombine first.
*
* @输入      pETM      指向三个 ETM 定时器其中一个的基址
*
* @无返回
*
*****/
void ETM_GenerateTrig2(ETM_Type *pETM)
{
    ASSERT(ETM2 == pETM);

    if(pETM->SYNC & ETM_SYNC_TRIG2_MASK)
    {
        #if defined(CPU_NV32)
            SIM->SOPT |= SIM_SOPT_ETMSYNC_MASK;
        #endif
    }
}

```

1.7.4 ETM2 实现同步

```

/*****//*!
*
* @运用 ETM2 来实现软件同步
*
* @输入      pETM      指向三个 ETM 定时器其中一个的基址
*
* @无返回
*
*****/
void ETM_SoftwareSync(ETM_Type *pETM)
{

```

```
ASSERT (ETM2 == pETM);
```

```
pETM->SYNCONF |= ETM_SYNCONF_SYNCMODE_MASK;
```

```
pETM->SYNC |= ETM_SYNC_SWSYNC_MASK;
```

```
}
```

1.7.5 由 ETM2 产生的硬件触发

```
/******//*!
```

```
*
```

```
* @产生 ETM2 硬件触发, Note: please call ETM_HardwareSyncCombine first.
```

```
*
```

```
* @输入 pETM 指向三个 ETM 定时器其中一个的基址
```

```
*
```

```
* @无返回
```

```
*
```

```
*****/
```

```
void ETM_GenerateTrig2 (ETM_Type *pETM)
```

```
{
```

```
    ASSERT (ETM2 == pETM);
```

```
    if (pETM->SYNC & ETM_SYNC_TRIG2_MASK)
```

```
    {
```

```
#if defined (CPU_NV32)
```

```
    SIM->SOPT |= SIM_SOPT_ETMSYNC_MASK;
```

```
#endif
```

```
    }
```

```
}
```

1.7.6 BDM 模式下的 ETM 行为

```
/******
```

```
*
```

```
* @选择 BDM 模式下的 ETM 行为
```

```
*
```

```
* @输入 pETM 指向三个 ETM 定时器其中一个的基址
```

```
* @输入 u8DebugMode debug 的模式从 00-11 之间选择
```

```
*
```

```
* @无返回
```

```
*
```

```
*****/
```

```
void ETM_SetDebugModeBehavior (ETM_Type *pETM, uint8_t u8DebugMode)
```

```
{
    ASSERT((ETM2 == pETM));
    pETM->CONF &= ~ETM_CONF_BDMODE_MASK;
    pETM->CONF |= ETM_CONF_BDMODE(u8DebugMode);
}
```

1.8 对 ETM 通道的操作

1.8.1 通道间交换输出结果

```
/******
* @交换通道 CH (n) 和通道 CH (n+1) 的输出结果
* @输入 pETM 其中一个 ETM 定时器的基址
* @输入 ChannelPair 要被交换的通道数号, 即 n 可为 0, 1, 2,
*****/
void ETM_InvertChannel(ETM_Type *pETM, uint8_t u8ChannelPair)
{
    ASSERT((ETM2 == pETM) && u8ChannelPair <= 2);

    pETM->INVCTRL |= 1<<u8ChannelPair;
    if(pETM->SYNCONF & ETM_SYNCONF_INV_MASK) /* if PWM sync is needed */
    {
        pETM->SYNCONF |= ETM_SYNCONF_SYNCMODE_MASK; /* recommend enhanced sync mode */
        if(pETM->SYNCONF & ETM_SYNCONF_SWINVC_MASK) /* if software sync is needed*/
        {
            pETM->SYNC |= ETM_SYNC_SWSYNC_MASK; /* software sync */
        }
        else if(pETM->SYNCONF & ETM_SYNCONF_HWINVC_MASK) /* if hardware sync is needed*/
        {
            pETM->SYNC |= ETM_SYNC_TRIG2_MASK;
        }
    }

#ifdef CPU_NV32
    SIM->SOPT |= SIM_SOPT_ETMSYNC_MASK; /* hardware sync */
#endif
}
else
{
}
```

1.8.2 ETM 通道初始化

```

/*****
*
* @本函数用来配置 ETM 通道, 包括通道状态及控制寄存器 CnSC 和通道计数值寄存器 CnV
* @输入      pETM          指向三个 ETM 定时器其中一个的基址
* @输入      ETM_Channel    ETM 的通道号
* @输入      pTETMCH_Params 指向 ETM 通道一般参数的指针
*
* @无返回值
*
*****/
void ETM_ChannelInit(ETM_Type *pETM, uint8_t u8ETM_Channel, ETM_ChParamsType
*pTETMCH_Params)
{
    ASSERT((ETM0 == pETM) || (ETM1 == pETM) || (ETM2 == pETM)); //断言检测定时器号是否正
确

    if (ETM0 == pETM)
    {
        ASSERT(u8ETM_Channel < 2);
        SIM->SCGC |= SIM_SCGC_ETM0_MASK;
    }
    #if !defined(CPU_KE04)
    else if (ETM1 == pETM)
    {
        ASSERT(u8ETM_Channel < 2);
        SIM->SCGC |= SIM_SCGC_ETM1_MASK;
    }
    #endif
    else
    {
        ASSERT(u8ETM_Channel < 6);
        SIM->SCGC |= SIM_SCGC_ETM2_MASK;
    }

    pETM->CONTROLS[u8ETM_Channel].CnSC = pTETMCH_Params->u8CnSC;
    pETM->CONTROLS[u8ETM_Channel].CnV = pTETMCH_Params->u16CnV;

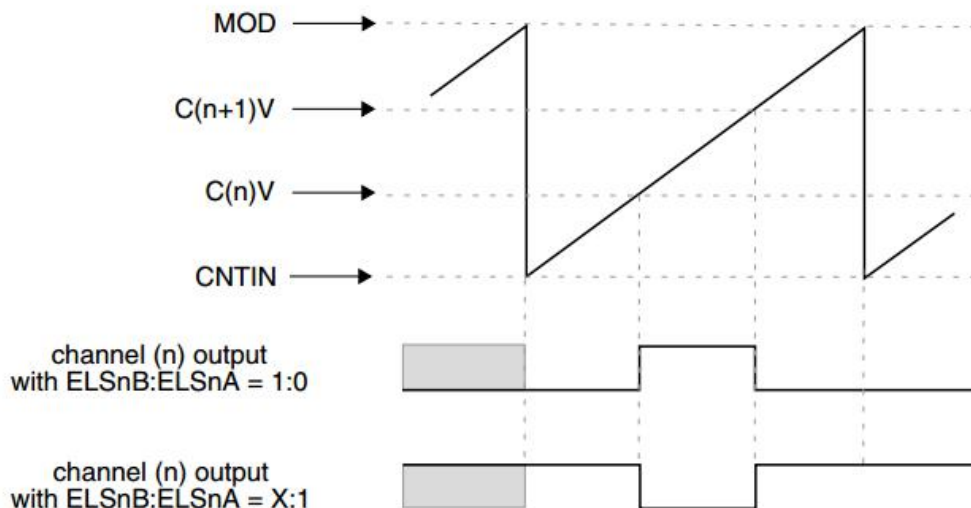
    return;
}

```

第二章 样例程序

2.1 级联模式 PWM 输出

在级联模式下，通道 n （偶数）和通道 $n+1$ （奇数）级联到通道 n 输出一路 PWM，周期由 $(MOD - CNTIN + 1)$ 决定，占空比取决于 $(|C_{n+1}V - C_nV|)$



```

/*****
*
* @ETM2 的通道 0 和通道 1 联合产生一个 PWM 方波, 通道 0 管脚为 PC0, 通道 1 管脚为 PC1
*
*****/
int main (void)
{
    uint8_t u8Ch;
    /* 系统初始化*/
    sysinit();
    printf("\nRunning the ETM_demo project.\n");
    LED0_Init(); //初始化 LED0
    /* s 设置 ETM2 为级联模式*/
    ETM_PWMInit(ETM2, ETM_PWMMODE_COMBINE, ETM_PWM_LOWTRUEPULSE);
    ETM_SetModValue(ETM2, 9999); //设 ETM2 的 MOD 值
    /* 时钟及分频系数的配置*/
    ETM_ClockSet(ETM2, ETM_CLOCK_SYSTEMCLOCK, ETM_CLOCK_PS_DIV1);
    ETM_SetCallback(ETM2, ETM2_Task); //设置 ETM2 回调函数点
    NVIC_EnableIRQ(ETM2_IRQn); //唤醒中断
    ETM_EnableOverflowInt(ETM2); //开启溢出标志位
    u8Ch = UART_GetChar(TERM_PORT); //串口 1 读字符

```

```
/* set the duty cycle, note: only fit for combine mode */
ETM_SetDutyCycleCombine(ETM2, ETM_CHANNEL_CHANNEL1, 50);
while(1)
{
    u8Ch = UART_GetChar(TERM_PORT);           //读取串口字符
    UART_PutChar(TERM_PORT, u8Ch);           //写入串口字符
    LED0_Toggle();                             //打开蓝光 LED
}
}

/*****RTC 中断函数*****/
void RTC_Task(void)
{
    LED0_Toggle();                             //闪烁灯
}

/*****ETM2 中断服务函数** */
void ETM2_Task(void)
{
    static uint32_t count;
    ETM_ClrOverflowFlag(ETM2);                 //清除溢出标志位
    if(count == 2000)
    {
        count = 0;
        UART_PutChar(TERM_PORT, '@ ');        //传输字符@到串口 1
    }
    else
    {
        count++;
    }
}

/*****/
```

2.2 脉冲输入捕获例程

```

/*****
跳变沿脉冲捕获，由 ETM0 的通道 1 产生脉冲信号, 周期为 10000, 脉冲宽度为 5000,
使用 ETM2 的通道 0 作为捕获口
*****/

#include "common.h"
#include "ics.h"
#include "ETM.h"
#include "uart.h"
#include "sysinit.h"
void ETM2_Task(void);
volatile uint16_t u16Ch0Value, u16Ch1Value;
volatile uint8_t u8IntMark;
int main (void)
{
    sysinit(); //初始化
    SIM->PINSEL |= SIM_PINSEL_ETM0PS1_MASK;
    /* ETM0 通道 1（即 PB3 脚）输出方波即 PB3 脚，周期 10000，脉冲宽度为 5000 */
    ETM_OutputCompareInit(ETM0, ETM_CHANNEL_CHANNEL1, ETM_OUTPUT_TOGGL);
    ETM_SetModValue(ETM0, 5000); //装载 ETM0 的 MOD 计数值
    ETM_SetChannelValue(ETM0, ETM_CHANNEL_CHANNEL1, 2000); //设置通道 1 的值
    /* 设置时钟，ETM0 通道 1*/
    ETM_ClockSet(ETM0, ETM_CLOCK_SYSTEMCLOCK, ETM_CLOCK_PS_DIV1);
    /* 配置 ETM2 的通道用来测出脉冲的宽度或周期*/
    ETM_DualEdgeCaptureInit(ETM2,
                            ETM_CHANNELPAIRO,
                            ETM_INPUTCAPTURE_DUALEDGE_ONESHOT,
                            ETM_INPUTCAPTURE_DUALEDGE_RISINGEDGE,
                            ETM_INPUTCAPTURE_DUALEDGE_FALLINGEDGE );
    /*ETM2 的时钟设置, 为系统时钟，1 分频*/
    ETM_ClockSet(ETM2, ETM_CLOCK_SYSTEMCLOCK, ETM_CLOCK_PS_DIV1);
    ETM_SetCallback(ETM2, ETM2_Task);
    NVIC_EnableIRQ(ETM2_IRQn); //唤醒中断
    ETM_EnableChannelInt(ETM2, (ETM_CHANNELPAIRO+1));
    /* echo chars received from terminal */
    while(1)
    {
        if(u8IntMark)
        {
            u16Ch0Value = ETM2->CONTROLS[0].CnV;
            u16Ch1Value = ETM2->CONTROLS[1].CnV;
            u8IntMark= 0; //反转标志位
        }
    }
}

```

```

    printf("\n    Dual    edge    capture    end.    The    input    pulse    width
is %d\n", (uint16_t) (u16Ch1Value - u16Ch0Value));
    /* 重新开启跳变沿捕捉 */
    ETM2->COMBINE |= (ETM_COMBINE_DECAPO_MASK << (ETM_CHANNELPAIR0*4));
}
}
}

/*****
ETM2 回调函数
*****/

void ETM2_Task(void)
{
    ETM_ClrChannelFlag(ETM2, ETM_CHANNELPAIR0);
    ETM_ClrChannelFlag(ETM2, ETM_CHANNELPAIR0+1);
    u8IntMark = 1;
}

```

将 ETM2 Ch0 (PC0) 与 ETM0 Ch1 (PB3) 短接。

串口通信格式：波特率 115200，8 位数据位，1 位停止位，无校验。

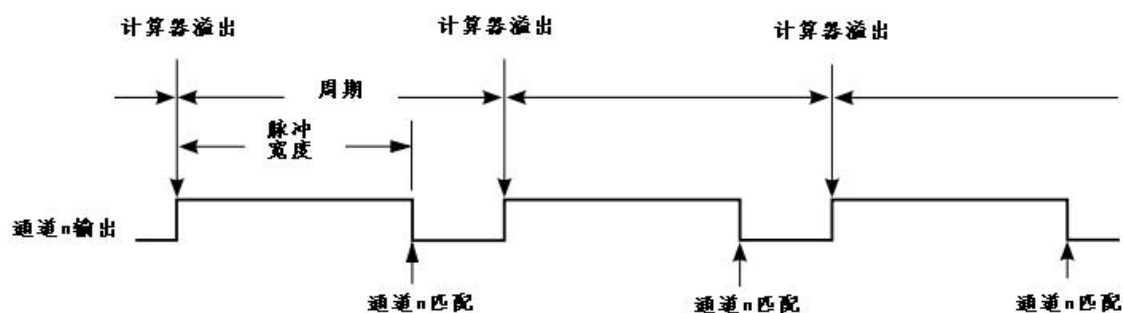
该样例工程在 nv32_pdk\build\keil\NV32\ETM DualEdgeCapture demo 下

串口窗口显示结果如下:



2.3 呼吸灯例程--EPWM 输出

在边沿对齐 EPWM 模式下，周期由 $(MOD-CNTIN+1)$ 决定，占空比由 $CnV-CNTIN$ 决定，脉冲宽度在计数器溢出重新加载 $CNTIN$ 初值时开始，在计数值与 CnV 匹配时结束，此后为低电平时间，直到计数器再次溢出为止。



```

/*****
*
* @ETM2 通道 1 输出边沿对齐的 PWM 波，来控制 NV32F100 板上的 LED 灯的蓝灯闪烁
*
*****/

#include "common.h"
#include "ics.h"
#include "etm.h"
#include "uart.h"
#include "sysinit.h"
void ETM2_Task(void);
int main (void)
{
    /* Perform processor initialization */
    sysinit(); //初始化
    SIM_RemapETM2CH1Pin(); //通道映射
    SIM_RemapETM2CH0Pin();
    /* ETM2 被设置为边沿对齐 PWM 波*/
    ETM_PWMInit(ETM2, ETM_PWMMODE_EDGEALLIGNED, ETM_PWM_HIGHTRUEPULSE); //
    ETM_SetETMEnhanced(ETM2); //
    /* 更新 MOD 的值 */
    ETM_SetModValue(ETM2, 9999); //
    /* set clock source, start counter */
    ETM_ClockSet(ETM2, ETM_CLOCK_SYSTEMCLOCK, ETM_CLOCK_PS_DIV1);
    NVIC_EnableIRQ(ETM2_IRQn); //唤醒中断
    /* 中断回调函数*/

```

```
ETM_SetCallback(ETM2, ETM2_Task);
/* 开启 ETM2 溢出中断 */
ETM_EnableOverflowInt(ETM2);
while(1)
{
}
}

/*****
*****/

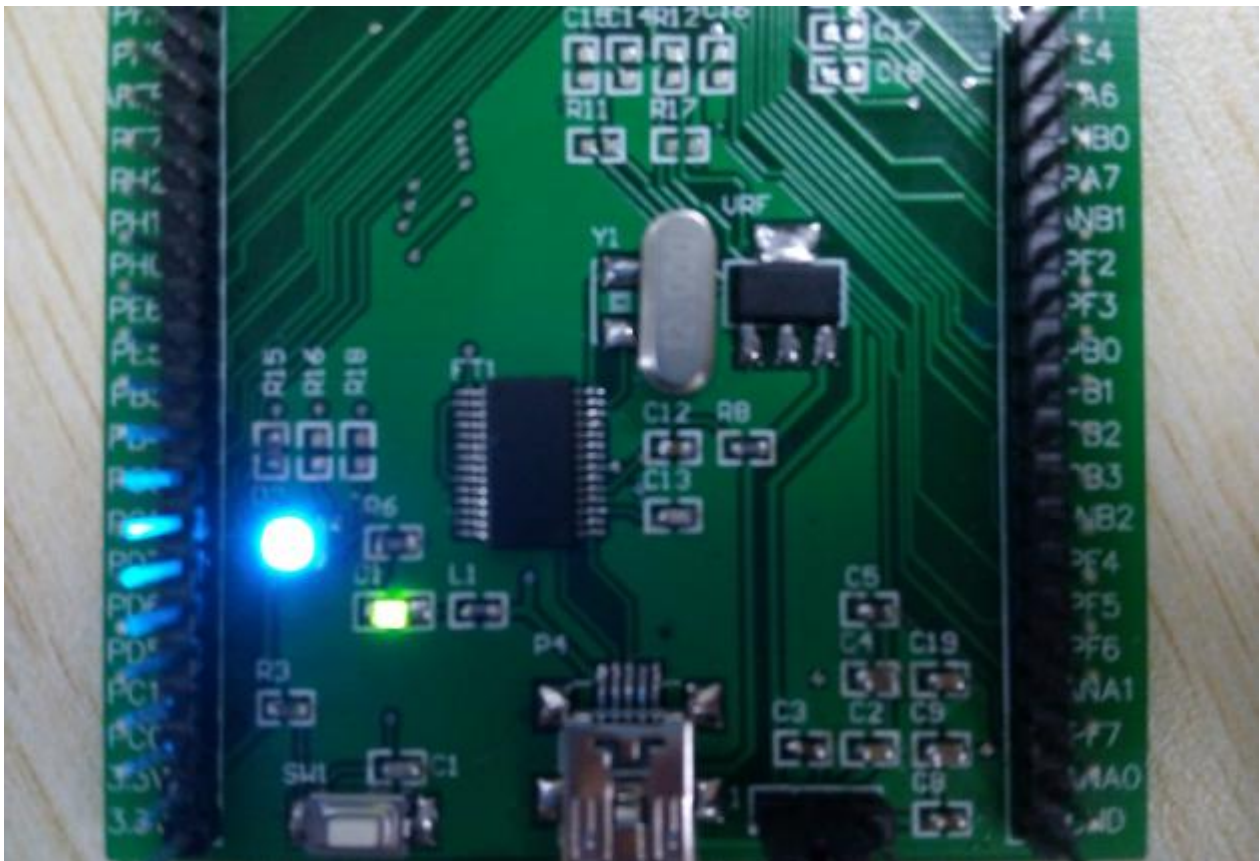
uint16_t u16ChV_old, u16ChV_new;
void ETM2_Task(void)
{
    static uint16_t u16count;
    static uint8_t u8DirMark;
    /* clear the flag */
    ETM_ClearOverflowFlag(ETM2); //清除溢出标志位
    if(100 == u16count)
    {
        u16count = 0;
        u16ChV_old = ETM2->CONTROLS[1].CnV;
        if(!u8DirMark)
        {
            u16ChV_new = u16ChV_old + 200;
            if(u16ChV_new >= ETM2->MOD)
            {
                u16ChV_new = ETM2->MOD - 200;
                u8DirMark = 1;
            }
            else
            {
            }
        }
        else
        {
            u16ChV_new = u16ChV_old - 200;
            if(u16ChV_new < 200)
            {
                u16ChV_new = 200;
                u8DirMark = 0;
            }
            else
            {
            }
        }
    }
}
```

```

/*更新通道值*/
ETM_SetChannelValue(ETM2, ETM_CHANNEL_CHANNEL1, u16ChV_new);
}
else
{
    u16count++;
}
}

```

烧录程序后的效果图如下：



效果：蓝光灯呼吸闪烁，该蓝光灯为板载 LED 灯
该样例工程在 nv32_pdk\build\keil\NV32\ETM_EPWM_demo 下

2.4 输出比较模式

```

/*****
*
* @ 输出比较模式例程
* @ ETM0 的通道 1 产生触发波形，该通道所对应的管脚为 PB3
*
*****/
int main (void)
{
    uint8_t u8Ch;

    /* Perform processor initialization */
    sysinit();

    /* Output ETM0 ch1 to PTB3 which is connected to J2 Pin8 for measure */
    SIM_RemapETM0CH1Pin(); //映射通道到管脚
    /* ETM0 为输出比较模式，通道 1 产生触发波形 */
    ETM_OutputCompareInit(ETM0, ETM_CHANNEL_CHANNEL1, ETM_OUTPUT_TOGGLE); // /* set MOD
value */
    ETM_SetModValue(ETM0, 5000); //加载对应的 MOD 数值
    ETM_SetChannelValue(ETM0, ETM_CHANNEL_CHANNEL1, 2000); //设置 CnV 的值
    ETM_ClockSet(ETM0, ETM_CLOCK_SYSTEMCLOCK, ETM_CLOCK_PS_DIV1); //时钟配置
    while(1)
    {
        u8Ch = UART_GetChar(TERM_PORT);
        UART_PutChar(TERM_PORT, u8Ch);
    }
}

```

该样例工程在 nv32_pdk\build\keil\NV32\ETM_OutputCompare_demo 下