NV32F100x ETM 时钟模块编程

第一章 库函数简介

库函数列表

void ETM_ClockSet(ETM_Type *pETM, uint8_t u8ClockSource, uint8_tu8ClockPrescale);
ETM 时钟及分频系数的选择
void ETM_PWMInit(ETM_Type *pETM, uint8_t u8PWMModeSelect, uint8_t u8PWMEdgeSelect);
PWM 初始化函数
void ETM_SetModValue(ETM_Type *pETM, uint16_t u16ModValue);
设置 ETM 的 MOD 值
void ETM_InputCaptureInit(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8CaptureMode);
ETM 单边输入捕获初始化
void ETM_DualEdgeCaptureInit(ETM_Type *pETM, uint8_t u8ChannelPair, uint8_t u8CaptureMode,
uint8_t u8Channel_N_Edge, uint8_t u8Channel_Np1_Edge);
ETM 双边沿捕获初始化（ETM2）
void ETM_OutputCompareInit(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8CompareMode);
ETM 输出比较初始化
void ETM_EnableOverflowInt(ETM_Type *pETM);
ETM 溢出中断使能
void ETM_EnableOverflowInt(ETM_Type *pETM);
ETM 溢出中断禁止
void ETM_SetETMEnhanced(ETM_Type *pETM);
ETM 专用特性设置（ETM2）
void ETM_SoftwareSync(ETM_Type *pETM);
ETM 设置为 PWM 同步软件触发（ETM2）
void ETM_HardwareSync(ETM_Type *pETM, uint8_t u8TriggerN);
ETM 设置为 PWM 同步硬件触发（ETM2）
void ETM_PolaritySet(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8ActiveValue);
ETM 通道极性的设置（ETM2)
uint8_t ETM_GetOverFlowFlag(ETM_Type *pETM);
获取 ETM 溢出位 TOF
void ETM_ClrOverFlowFlag(ETM_Type *pETM);
清除 ETM 溢出位 TOF

在对 ETM 模块部分的操作之前，首先要了解，NV32F100x 的 ETM 模块，分为 ETM0，ETM1 和 ETM2。其中 ETM0 和 ETM1 分别有 2 个通道，ETM2 为增强型，有 6 个通道。在对该模块操作的时候，一定要按照对应通道数，以及相应的寄存器来进行配置。详细的寄存器相关信息见 NV32F100x 参考手册-ETM 模块。

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

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（注：外接其他频率的时钟，如 ETM0 为 ETM0_CLK 引脚）

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。	
	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

函数名	ETM_ClockSet
函数原形	ETM_ClockSet(ETM_Type*pETM,uint8_tu8ClockSource, uint8_tu8ClockPrescale)
功能描述	设置时钟资源及分频系数
输入参数	ETM 基址 pETM，时钟源 u8ClockSource，时钟分频系数 u8ClockPrescale
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_ClockSet(ETM0,ETM_CLOCK_SYSTEMCLOCK , ETM_CLOCK_PS_DIV1)

```
/*****************************************************************************//*!
*
* @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 选择的时钟源 ： 禁用（00）、系统时钟（01）
*                                固定频率时钟（10）、外部时钟（11）
* @输入      ClockPrescale       分频系数
*
* @无返回
*
*****************************************************************************/
void ETM_ClockSet(ETM_Type *pETM, uint8_t u8ClockSource, uint8_tu8ClockPrescale)
{
    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 ETM 的计数方式

注：ETM0，ETM1 无计数初始值寄存器 CNTIN，通过写入计数器寄存器 CNT 来清 0 的方式加载初始值 0

1.2.1 向上计数

- CPWMS = 0

CNTIN 定义计数的起始值，MOD 定义计数的最终值，参见下图：CNTIN 的值加载到 ETM 计数器中，计数器的值递增，直至达到 MOD 的值，此时计数器将重新加载 CNTIN 的值。

采用向上计数时的 ETM 周期为(MOD - CNTIN + 0x0001) × ETM 计数器时钟的周期。

ETM 计数器从 MOD 变为 CNTIN 时，TOF 位将置位。

ETM 计数为向上计数。

CNTIN=0xFFFC(在二的补码中等于-4) MOD=0x0004

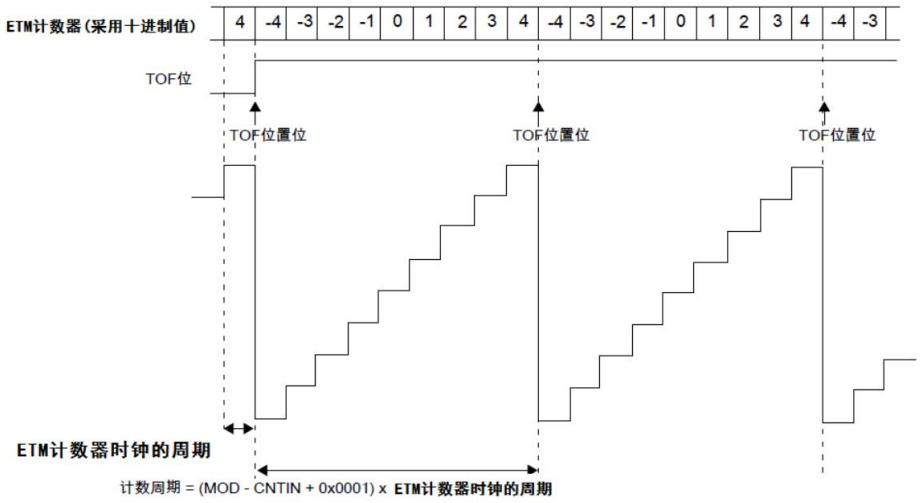


图 1-1 带符号的 ETM 计数器累加模式

1.2.2 向上-向下计数

• CPWMS = 1

CNTIN 定义计数的起始值，MOD 定义计数的最终值。CNTIN 的值加载到 ETM 计数器中，计数器的值一直增加，直至达到 MOD 的值，紧接着计数器的值一直减少，直至回到 CNTIN 的值，然后计数器将重新开始自上而下计数。

采用向上-向下计数时的 ETM 周期为 2 × (MOD - CNTIN)×ETM 计数器时钟的周期。

ETM 计数器从 MOD 更改为(MOD - 1)时，TOF 位将被置位。如果(CNTIN = 0x0000)，ETM 计数为无符号向上-向下计数。参见下图：

ETM 计数为向上-向下计数。

CNTIN=0x0000, MOD=0x0004

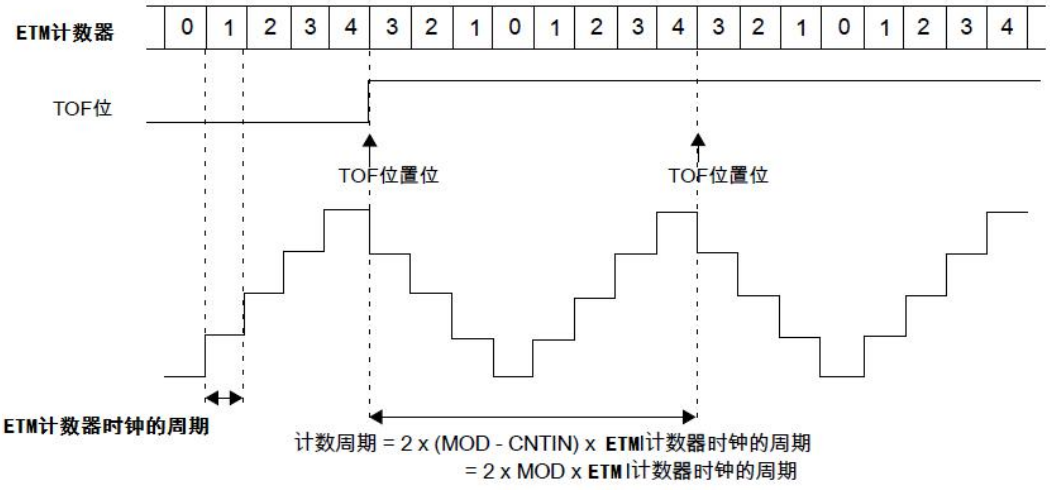


图 1-2 向上-向下计数模式

1.3 脉冲宽度调制（PWM）

1.3.1 PWM 初始化函数

下图为 ETM 模块的通道模式选择，输出 PWM 波常用配置为边沿对齐模式的 PWM 即 EPWM

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

函数名	ETM_PWMInit
函数原形	ETM_PWMInit(ETM_Type *pETM,uint8_t u8PWMModeSelect, uint8_t u8PWMEdgeSelect)
功能描述	初始化 PWM 功能
输入参数	ETM 基址 pETM，PWM 模式 u8PWMModeSelect，边沿触发方式 u8PWMEdgeSelect
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_PWMInit(ETM2,ETM_PWMMODE_EDGEALLIGNED,ETM_PWM_LOWTRUEPULSE)

```
/*****************************************************************************************
* @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       高真脉冲（01）、低真脉冲（10）
*
* @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

    if (ETM0 == pETM)    //ETM0 共有两个通道
    {
        channels = 2;
        SIM->SCGC |= SIM_SCGC_ETM0_MASK;
    }
    else if(ETM1 == pETM) //ETM1 共有两个通道
    {
        channels = 2;
    }
    Else                        //ETM2 共有六个通道
    {
        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_COMP0_MAS|ETM_COMBINE
_SYNCEN0_MAS|ETM_COMBINE_DTEN0_MASK|ETM_COMBINE_COMBINE1_MASK|ETM_COMBINE
_COMP1_MASK|ETM_COMBINE_SYNCEN1_MAS|ETM_COMBINE_DTEN1_MASK|ETM_COMBINE_CO
MBINE2_MASK|ETM_COMBINE_COMP2_MASK|ETM_COMBINE_SYNCEN2_MASK|ETM_COMBINE_D
TEN2_MASK ;
```

```
        pETM->SC &= ~ETM_SC_CPWMS_MASK;
    }
    if(ETM_PWM_HIGHTRUEPULSE == u8PWMEdgeSelect)
    {
        /* 配置通道寄存器，设置通道状态及通道计数值 */
        for(i=0; i<channels; i++)
        {
            pETM->CONTROLS[i].CnSC = ETM_CnSC_MSB_MASK | ETM_CnSC_ELSB_MASK;
            pETM->CONTROLS[i].CnV   = ETM_C0V_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_C0V_INIT + i*100 ;
        }
    }
}
```

1.3.2 通道输出极性的设置（ETM2）

ETMx_POL 通道极性寄存器的参数见参考手册 6.3.7.17

函数名	ETM_PolaritySet
函数原形	ETM_PolaritySet(ETM_Type *pETM,uint8_t u8ETM_Channel, uint8_t u8ActiveValue)
功能描述	设置通道极性
输入参数	ETM 基址 pETM，通道号 u8ETM_Channel，极性选择 u8ActiveValue
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_PolaritySet(ETM2,uint8_t u8ETM_Channel,uint8_t u8ActiveValue)

```
/*****************************************************************************
*
* @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

函数名	ETM_SetDutyCycleCombine
函数原形	ETM_SetDutyCycleCombine(ETM_Type *pETM,uint8_t u8ETM_Channel, uint8_t u8DutyCycle)
功能描述	配置级联模式及占空比
输入参数	ETM 基址 pETM，奇数通道号 u8ETM_Channel，占空比 u8DutyCycle
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_SetDutyCycleCombine(ETM2,1,10)

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

函数名	ETM_SetDutyCycleCombine
函数原形	ETM_SetDutyCycleCombine(ETM_Type *pETM,uint8_t u8ETM_Channel, uint8_t u8DutyCycle)
功能描述	配置级联模式及占空比
输入参数	ETM 基址 pETM，奇数通道号 u8ETM_Channel，占空比 u8DutyCycle
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_SetDutyCycleCombine(ETM2,1,10)

```
/*****************************************************************************
*
* @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)));

    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
    {
     ETM->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（ETM2）

函数名	ETM_DualEdgeCaptureInit
函数原形	void ETM_DualEdgeCaptureInit(ETM_Type *pETM,uint8_t u8ChannelPair, uint8_t u8CaptureMode,uint8_t u8Channel_N_Edge, uint8_t u8Channel_Np1_Edge)
功能描述	配置 ETM 双边沿捕捉模式
输入参数	ETM 基址 pETM ETM2，通道号 u8ChannelPair，捕捉方式 u8CaptureMode 频道 N 边沿检测 u8Channel_N_Edge 频道 N+1 边沿检测 u8Channel_Np1_Edge
输出参数	无
返回值	无
先决条件	无
函数使用实例	void ETM_DualEdgeCaptureInit(ETM2,0,4,1,2)

```
/*****************************************************************************//*!
*
* @对 ETM 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     选择单周期捕捉（4），和连续捕捉方式（5）
* @输入        Channel_N_Edge  频道 N 边沿检测：无（0），上升沿（1）下降沿（2）双沿（3）
* @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))
    {

    }

    pETM->SC      = 0x0;
    pETM->MOD     = 0xFFFF;
    pETM->MODE |= ETM_MODE_ETMEN_MASK;        //ETMEN = 1
```

```
    pETM->COMBINE |=    ((ETM_COMBINE_DECAPEN0_MASK) << (u8ChannelPair * 4));

    pETM->CONTROLS[u8ChannelPair].CnSC&=~ETM_CnSC_CHF_MASK;
pETM->CONTROLS[u8ChannelPair + 1].CnSC &= ~ETM_CnSC_CHF_MASK;

    if(ETM_INPUTCAPTURE_DUALEDGE_ONESHOT == u8CaptureMode)
    {
        pETM->CONTROLS[u8ChannelPair].CnSC &= ~ETM_CnSC_MSA_MASK;
        pETM->CONTROLS[u8ChannelPair+1].CnSC &= ~ETM_CnSC_MSA_MASK;
    }
    else if(ETM_INPUTCAPTURE_DUALEDGE_CONTINUOUS == u8CaptureMode)
    {
        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);
    pETM->CONTROLS[u8ChannelPair + 1].CnSC |= (u8Channel_Np1_Edge << 2);

    pETM->COMBINE |=    (ETM_COMBINE_DECAP0_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。

函数名	ETM_OutputCompareInit
函数原形	void ETM_OutputCompareInit(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8CompareMode)
功能描述	配置输出对比
输入参数	ETM 基址 pETM，通道号 u8ETM_Channel，选择模式 u8CompareMode
输出参数	无
返回值	无
先决条件	无
函数使用实例	void ETM_OutputCompareInit(ETM0,1,1)

```
/*****************************************************************************//*!
*
* @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))
           );

    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_C0V_INIT;
}
```

1.6 同步设置寄存器 ETMx_SYNCONF 的配置(ETM2)

具体信息见参考手册 6.3.7.11

1.6.1 整体配置(ETM2)

函数名	ETM_SyncConfigDeactivate
函数原形	Void ETM_SyncConfigDeactivate(ETM_Type*pETM,uint32_t u32ConfigValue)
功能描述	寄存器 ETMx_SYNCONF 寄存器配置
输入参数	ETM 基址 pETM ETM2，寄存器 ETMx_SYNCONF 的值 u32ConfigValue
输出参数	无
返回值	无
先决条件	无

```
/*****************************************************************************
*
* @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
* @输入          pETM              指向三个 ETM 定时器其中一个的基址
* @输入          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 选择硬件触发资源（ETM2）

函数名	ETM_HardwareSync
函数原形	ETM_HardwareSync(ETM_Type *pETM, uint8_t u8TriggerN)
功能描述	选择硬件触发号
输入参数	ETM 基址 pETM ETM2，硬件触发号 u8TriggerN
输出参数	无
返回值	无
先决条件	无
函数使用实例	void ETM_HardwareSync(ETM2,0)

```
/*****************************************************************************/
* @ETM 中配置 ETMx_SYNC 寄存器来选择硬件触发
*
* @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（ETM2）

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

函数名	ETM_SWOutputControlSet
函数原形	ETM_SWOutputControlSet(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8ChannelValue)
功能描述	ETM2 的软件输出控制同步触发选择
输入参数	ETM 基址 pETM ETM2，通道号 u8ETM_Channel 是否触发 u8ChannelValue
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_SWOutputControlSet(ETM2,0,1)

```
/*******************************************************************************//*!
*
* @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            ETM2
* @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;

#if defined(CPU_NV32)
                SIM->SOPT    |= SIM_SOPT_ETMSYNC_MASK;
#endif
            }
        }
        else
        {
        }
}
```

1.6.4 通过配置 ETM 保证硬件同步，产生联合触发（ETM2）

函数名	ETM_SWOutputControlSet
函数原形	ETM_SWOutputControlSet(ETM_Type *pETM, uint8_t u8ETM_Channel, uint8_t u8ChannelValue)
功能描述	ETM2 的硬件联合同步触发
输入参数	ETM 基址 pETM ETM2，触发标志号 u8ChannelValue
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_SWOutputControlSet(ETM2,0,1)

```
/*****************************************************************************//*!
*
* @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       硬件触发资源标志号.
*
* @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

函数名	ETM_Init
函数原形	ETM_Init(ETM_Type *pETM, ETM_ConfigType *pConfig)
功能描述	以配置结构体 pConfig 来初始化 ETM
输入参数	ETM 基址 pETM，配置结构体 ETM_ConfigType
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_Init(ETM0, &ETM_Config)

```
/*****************************************************************************//*!
*
* @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_ETM0_MASK;
    }
    else
    {
        SIM->SCGC |= SIM_SCGC_ETM2_MASK;
    }

    /*关闭计数器*/
    pETM->SC = 0;
    pETM->MODE = pConfig->mode;
    pETM->MOD = pConfig->modulo;
    pETM->CNT = pConfig->cnt;

    if( pETM->MODE & ETM_MODE_ETMEN_MASK    )
    {

        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;
    }
    pETM->SC = pConfig->sc;
}
```

1.7.2 恢复 ETM 组件

函数名	ETM_DeInit
函数原形	ETM_DeInit(ETM_Type *pETM)
功能描述	恢复 ETM 组件至初始化前
输入参数	ETM 基址 pETM
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_DeInit(ETM0)

```
/*******************************************************************************
 恢复相应的 ETM 功能组件函数至初始化前
*******************************************************************************/
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->DEADETME = 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);
    }
```

```
    else if (ETM2 == pETM)
      SIM->SCGC &= ~SIM_SCGC_ETM2_MASK;
        NVIC_DisableIRQ(ETM2_IRQn);
    }
}
```

1.7.3 由硬件触发 2 产生 PWM 同步（ETM2）

函数名	ETM_GenerateTrig2
函数原形	ETM_GenerateTrig2(ETM_Type *pETM)
功能描述	硬件触发 2 产生 PWM 同步触发
输入参数	ETM 基址 pETM
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_GenerateTrig2(ETM2)

```
/*******************************************************************************//*!
*
* @硬件触发 2 产生 ETM2 的 PWM 同步触发
*
* @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 实现软件同步触发（ETM2）

函数名	ETM_SoftwareSync
函数原形	ETM_SoftwareSync(ETM_Type *pETM)
功能描述	ETM2 的软件同步触发
输入参数	ETM 基址 pETM
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_GenerateTrig2(ETM2)

```
/*******************************************************************************//*!
*
* @实现软件同步触发
*
* @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 BDM 模式下的 ETM 行为（ETM2）

函数名	ETM_SetDebugModeBehavior
函数原形	ETM_SetDebugModeBehavior(ETM_Type *pETM, uint8_t u8DebugMode)
功能描述	ETM2 的软件同步触发
输入参数	ETM 基址 pETM
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_SetDebugModeBehavior(ETM2, 0)

```
/*******************************************************************************
*
* @选择 BDM 模式下的 ETM 行为
*
```

```
* @输入    pETM                ETM2
* @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_BDMMODE_MASK;
    pETM->CONF |= ETM_CONF_BDMMODE(u8DebugMode);
}
```

1.8 对 ETM 通道的操作

1.8.1 通道间交换输出结果（ETM2）

函数名	ETM_InvertChannel
函数原形	ETM_InvertChannel(ETM_Type *pETM, uint8_t u8ChannelPair)
功能描述	ETM2 的通道输出结果交换
输入参数	ETM 基址 pETM ETM2 通道对号 u8ChannelPair
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_InvertChannel(ETM2,0)

```
/*******************************************************************************
* @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_INVC_MASK)
    {
        pETM->SYNCONF |= ETM_SYNCONF_SYNCMODE_MASK;
        if(pETM->SYNCONF & ETM_SYNCONF_SWINVC_MASK)
      {
            pETM->SYNC |= ETM_SYNC_SWSYNC_MASK;
        }
```

```
            else if(pETM->SYNCONF & ETM_SYNCONF_HWINVC_MASK)
            {
                pETM->SYNC |= ETM_SYNC_TRIG2_MASK;

#if defined(CPU_NV32)
                SIM->SOPT    |= SIM_SOPT_ETMSYNC_MASK;
#endif
            }
        }
        else
        {
        }
}
```

1.8.2 ETM 通道初始化

函数名	ETM_ChannelInit
函数原形	Void ETM_ChannelInit(ETM_Type *pETM,uint8_t u8ETM_Channel, ETM_ChParamsType *pETM_ChParams)
功能描述	ETM 的通道初始化
输入参数	ETM 基址 pETM ETM 的通道号 u8ETM_Channel 通道配置 ETM_ChParamsType
输出参数	无
返回值	无
先决条件	无
函数使用实例	Void ETM_ChannelInit(ETM0,0, &ETMCN_Config)

```
/*****************************************************************************//*!
*
* @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 *pETM_ChParams)
{
    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;
    }
    else if(ETM1 == pETM)
    {
        ASSERT(u8ETM_Channel < 2);
        SIM->SCGC |= SIM_SCGC_ETM1_MASK;
    }
    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;
}
```

1.9 TOF 频率设置（ETM2）

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 NUMTOF	TOF 频率 选择计数器溢出数量与 TOF 位置次数之比。 NUMTOF=0：TOF 位针对每一位计数器溢出进行置位。 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。 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。 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。 以此类推，此模式最多可延续到 31 次。

依次类推，该模式进行最多的次数为 31 次。

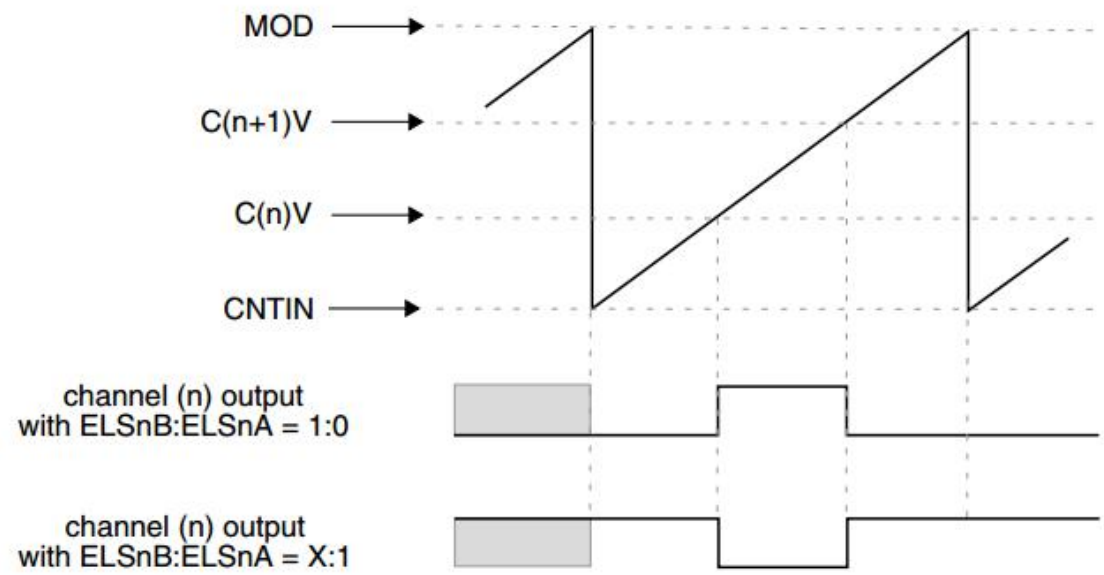
函数名	ETM_SetTOFFrequency
函数原形	ETM_SetTOFFrequency(ETM_Type *pETM, uint8_t u8TOFNUM)
功能描述	TOF 频率设置
输入参数	ETM 基址 pETM TOF 频率数 u8TOFNUM
输出参数	无
返回值	无
先决条件	无
函数使用实例	ETM_SetTOFFrequency(ETM2, 31)

```
/*******************************************************************************//*!
* @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);
}
```

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
    /* 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 回调任务函数** **********************/
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_CHANNELPAIR0,
                            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_CHANNELPAIR0+1));

    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_DECAP0_MASK<<(ETM_CHANNELPAIR0*4));
                }
        }
}
/*****************************************************************************
ETM2 回调任务函数
*****************************************************************************/
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：

```
input pulse width is 5000 Dual edge capture end. The input pulse width is 5000 Dual
edge capture end. The input pulse width is 5000 Dual edge capture end. The input
pulse width is 5000 Dual edge capture end. The input pulse width is 5000 Dual edge
capture end. The input pulse width is 5000 Dual edge capture end. The input pulse
width is 5000 Dual edge capture end. The input pulse width is 5000 Dual edge capture
end. The input pulse width is 5000 Dual edge capture end. The input pulse width is
5000 Dual edge capture end. The input pulse width is 5000 Dual edge capture end. The
input pulse width is 5000 Dual edge capture end. The input pulse width is 5000 Dual
edge capture end. The input pulse width is 5000 Dual edge capture end. The input
pulse width is 5000 Dual edge capture end. The input pulse width is 5000 Dual edge
capture end. The input pulse width is 5000 Dual edge capture end. The input pulse
width is 5000 Dual edge capture end. The input pulse width is 5000 Dual edge capture
end. The input pulse width is 5000 Dual edge capture end. The input pulse width is
5000 Dual edge capture end. The input pulse width is 5000 Dual edge capture end. The
input pulse width is 5000 Dual edge capture end. The input pulse width is 5000 Dual
edge capture end. The input pulse width is 5000 Dual edge capture end. The input
pulse width is 5000 Dual edge capture end. The input pulse width is 5000 Dual edge
capture end. The
```

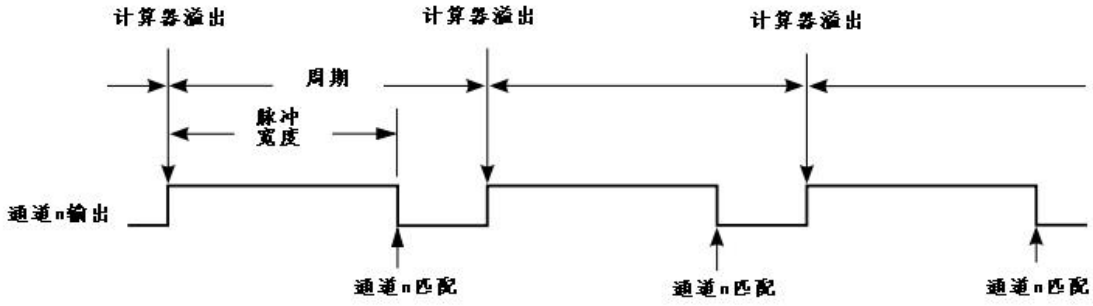
串口号 COM3
波特率 115200
数据位 8
停止位 1
校验位 None
流控制 None
☐ DTR ☐ RTS
☐ 定时发送 1000 ms/次
☐ HEX发送 ☐ 发送新行
abcdefg

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.
<HTML><HEAD><TITLE>Not Found</TITLE>
HTML><HEAD><TITLE>Not Found</TITLE>
TML><HEAD><TITLE>Not Found</TITLE>
ML><HEAD><TITLE>Not Found</TITLE>
```

www.mcu51.com S:0 R:360703 COM3 已关闭 115200bps 8 CTS=0 DSR=0 RLSD=0

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。

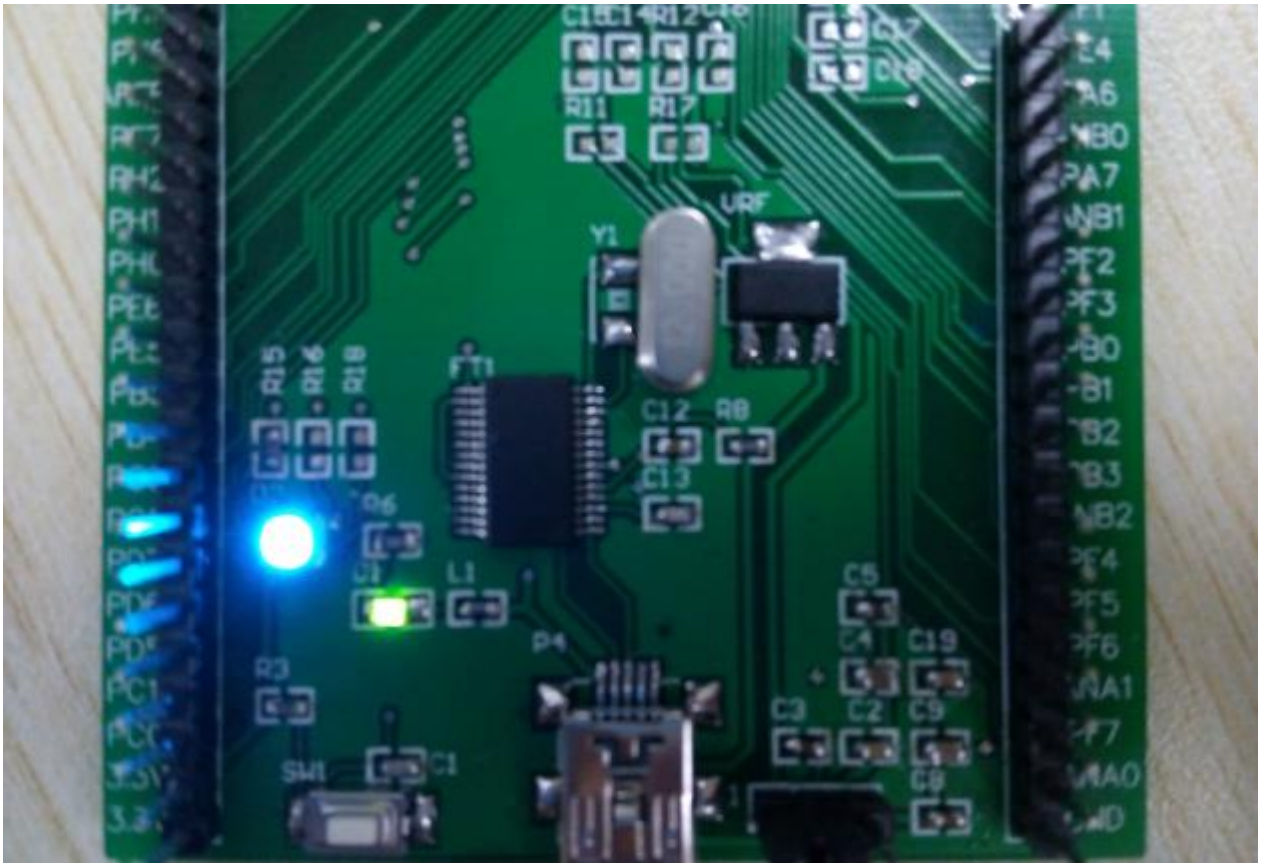



```
/*****************************************************************************
*
* @ETM2 通道 1 输出边沿对齐的 PWM 波，来控制 NV32F100 板上的 LED 灯的蓝灯闪烁
*
*****************************************************************************/
#include "common.h"
#include "ics.h"
#include "etm.h"
#include "uart.h"
#include "sysinit.h"
void ETM2_Task(void);
int main (void)
{

    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);

    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_ClrOverFlowFlag(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;

    sysinit();

     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);

    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

2.5 PWM 互补输出

```
/*********************NV32F100x--PWM 互补输出*******************!>
**********************************************************************
选用 FEE 外部晶振，总线时钟 30MHZ,联合 ETM2 的通道 0（PH0）通道 1（PH1）互补输出*****/
#include "common.h"
#include "ics.h"
#include "etm.h"
#include "uart.h"
#include "sysinit.h"

int main (void)
{

    sysinit();//系统初始化

    SIM_RemapETM2CH0Pin();//映射对应管脚
    SIM_RemapETM2CH1Pin();


    SIM->SCGC    |= SIM_SCGC_ETM2_MASK;//使能 ETM2 时钟
    ETM2->COMBINE &= ~ETM_COMBINE_COMBINE0_MASK;//通道 0 和通道 1 独立
      ETM2->SC |= ETM_SC_CPWMS_MASK;//选择先增后减的计数方式
      ETM2->COMBINE |= ETM_COMBINE_COMP0_MASK; //通道 0 和通道 1 输出互补
      ETM2->CONTROLS[0].CnSC = ETM_CnSC_ELSA_MASK;//低真脉冲
      ETM2->CONTROLS[1].CnSC = ETM_CnSC_ELSA_MASK;


      ETM_SetModValue(ETM2, 2999);//设置频率为 10KHZ

      ETM_SetChannelValue(ETM2, ETM_CHANNEL_CHANNEL0, 1500);//占空比为 50%
      ETM_SetChannelValue(ETM2, ETM_CHANNEL_CHANNEL1, 1500);

      ETM_ClockSet(ETM2, ETM_CLOCK_SYSTEMCLOCK, ETM_CLOCK_PS_DIV1); //ETM2 时钟设置

    while(1)
    {
    }

  }
```

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

2.6 十路 PWM 输出+串口 UART1 调节占空比

/*10 路 PWM 波输出，ETM0-2 路、ETM1-2 路、ETM2-6 路。通过串口 UART1 接收中断，改变占空比
通过串口输入大写字母 A 选择百分 0 的占空比，B 为 20%占空比，C 为 50%占空比，其他任意输入都为 80%占空比
ETM0CH0--PB2　ETM0CH1--PB3（注：需要先在 sysinit.c 中定义#define DISABLE_NMI 来禁用 NMI 引脚功能）
ETM1CH0--PH2 ETM1CH1--PE7
ETM2CH0--PH0 ETM2CH1--PH1　ETM2CH2--PD0　ETM2CH3--PD1　ETM2CH4--PB4　ETM2CH5--PB5
注（1.需要先在 sysinit.c 中定义#define DISABLE_NMI 来禁用 NMI 引脚功能,得到 PB4 脚的 ETM2CH4 输出）
（2.开发板上 PB3 管脚上有滤波电容，在使用 ETM0CH1 时可以去掉）*/
//选择 FEE 外部晶振时钟，总线时钟为 30MHZ

```
#include "common.h"
#include "ics.h"
#include "etm.h"
#include "uart.h"
#include "sysinit.h"
void UART_Recive(UART_Type *pUART);
volatile uint16_t sflag = 1500 ;

int main (void)
{
    UART_ConfigType sConfig;
    sysinit();

    sConfig.u32SysClkHz = BUS_CLK_HZ;
    sConfig.u32Baudrate = UART_PRINT_BITRATE;

    UART_Init(UART1,&sConfig);
    UART_SetCallback(UART_Recive);//设置中断回调入口
    /*映射对应的通道管脚，具体信息查看 SIM_PINSEL 寄存器*/
    SIM_RemapETM2CH0Pin();
    SIM_RemapETM2CH1Pin();
    SIM_RemapETM2CH2Pin();
    SIM_RemapETM2CH3Pin();
    SIM_RemapETM0CH0Pin();
    SIM_RemapETM0CH1Pin();
    SIM_RemapETM1CH0Pin();
    SIM_RemapETM1CH1Pin();
      /*PWM 初始化 */
    ETM_PWMInit(ETM0, ETM_PWMMODE_EDGEALLIGNED, ETM_PWM_HIGHTRUEPULSE);
    ETM_PWMInit(ETM1, ETM_PWMMODE_EDGEALLIGNED, ETM_PWM_HIGHTRUEPULSE);
    ETM_PWMInit(ETM2, ETM_PWMMODE_EDGEALLIGNED, ETM_PWM_HIGHTRUEPULSE);
```

```
    ETM_SetModValue(ETM0, 2999);//设置 ETM 的 MOD 值
    ETM_SetModValue(ETM1, 2999);
    ETM_SetModValue(ETM2, 2999);


    ETM_ClockSet(ETM0, ETM_CLOCK_SYSTEMCLOCK, ETM_CLOCK_PS_DIV1);//初始化 ETM 时钟
    ETM_ClockSet(ETM1, ETM_CLOCK_SYSTEMCLOCK, ETM_CLOCK_PS_DIV1);
    ETM_ClockSet(ETM2, ETM_CLOCK_SYSTEMCLOCK, ETM_CLOCK_PS_DIV1);
    NVIC_EnableIRQ(UART1_IRQn);
    UART_EnableRxBuffFullInt(UART1);
printf("\n\n\n 10 路 PWM 输出，串口改变占空比，输入 A 占空比为 0，输入 B 为 20，输入 C 为 50，输入其他任意都为 80\n\n\n");

    while(1)
    {
        ETM_SetChannelValue(ETM0, ETM_CHANNEL_CHANNEL0, sflag);//改变通道值 CNV
        ETM_SetChannelValue(ETM0, ETM_CHANNEL_CHANNEL1, sflag);
        ETM_SetChannelValue(ETM1, ETM_CHANNEL_CHANNEL0, sflag);
        ETM_SetChannelValue(ETM1, ETM_CHANNEL_CHANNEL1, sflag);

        ETM_SetChannelValue(ETM2, ETM_CHANNEL_CHANNEL0, sflag);
        ETM_SetChannelValue(ETM2, ETM_CHANNEL_CHANNEL1, sflag);
        ETM_SetChannelValue(ETM2, ETM_CHANNEL_CHANNEL2, sflag);
        ETM_SetChannelValue(ETM2, ETM_CHANNEL_CHANNEL3, sflag);
        ETM_SetChannelValue(ETM2, ETM_CHANNEL_CHANNEL4, sflag);
        ETM_SetChannelValue(ETM2, ETM_CHANNEL_CHANNEL5, sflag);

    }
}

/********ISR*********/

void UART_Recive(UART_Type *pUART)
{   uint8_t fip;
    if(UART_IsRxBuffFull(pUART))   //等待接收字符
    {
        fip = UART_GetChar(pUART);
        UART_PutChar(pUART,fip);//回显
    }

    if(fip== 'A')
      {
            sflag=0;  //如果输入 A，则占空比为 0
```

```
    }
      else if(fip=='B')//输入 B 占空比为 20%
    {
            sflag=600;
        }
        else if(fip=='C')//输入 C 占空比为 50%
        {
            sflag=1500;
        }
        else        //输出其他任意则占空比为 80%
        {
          sflag=2400;
        }
}
```

该样例工程在 nv32_pdk\build\keil\NV32\ETM_10PWM_Uart_demo 下