应用笔记

AN0023

YTM32B1LD0 eTMR 模块介绍

云途 YUNTU

目录

历史版本

版本号	日期	修改
1.0	2022-12-01	初始版本

1 特性

YTM32B1LD0 芯片 eTMR 模块有如下特性：
1、有三个 eTMR，eTMR0 包含 8 个通道，eTMR1 和 eTMR2 各包含 2 个通道；
2、时钟源可以选择系统时钟和外部时钟；
3、每一个 eTMR 包含一个时钟分频器，可以进行 1，2，4，8，16，32，64 和 128 分频；
4、每一个 eTMR 包含一个 16 位的计数器；
5、每一个通道可以被配置成三种模式：输入捕获、输出比较和 PWM 模式；

- 输入捕获：支持上升沿、下降沿或者双沿捕获；eTMR0 的通道 0-通道 3 支持带预分频的输入滤波器；
- 输出比较：输出信号能够被配置成在匹配点设置、清除、切换；
- PWM 模式：每个通道都支持边沿对齐 PWM 模式和中心对齐 PWM 模式；每一对互补通道模式都支持死区插入；

6、每一个 eTMR 都能配置成在匹配点产生触发信号；
7、PWM 输出能够通过软件控制；
8、每个 eTMR 的所有通道都可以控制极性；
9、每个 eTMR 都可以同步加载影子寄存器；
10、eTMR0 支持 4 路故障输入，用于全局故障控制；
11、只有 eTMR0 支持故障中断；
12、每个 eTMR 都支持通道中断；
13、每个 eTMR 都支持计数器溢出中断。

2 eTMR 功能简介

1、普通的计数器；
2、输入捕获；
3、输出比较；
4、PWM 输出。

3 eTMR 时基计算公式

SDK 中默认配置 system clock 的频率是 48MHz。
eTMR 的时钟频率和 ticks：

$$F_{eTMR\_CLK} = \frac{F_{system\_clock}}{Prescaler}$$

$$T_{eTMR\_ticks} = \frac{1}{F_{eTMR\_clock}}$$

举例说明：system clock 是 48MHz, 预分频配置成 4 分频，那么 eTMR 的 clock 就是 12MHz。计数一次的时间是 1/12M = 0.083us。

4 准备工作

4.1 配置 clock

使用 eTMR 之前需将 clock 打开。

```
peripheral_clock_config_t peripheralClockConfig0[NUM_OF_PERIPHERAL_CLOCKS_0] = {
    {
        .clkName = eTMR0_CLK,
        .clkGate = true,    //打开 eTMR0 clock
    },
};
```

4.2 配置 PINMUX

在 LD0 的 RM 中找到 Pinmux Table，选择所需的 pin 用于 eTMR 功能。下面以 pin PTC2 为例介绍 pin 的结构体设置，我们只需要关心结构体中的 4 个成员，包含：base，pinPortIdx，mux，gpioBase，它们分别对应的是控制端口基址，端口序号，pin mux 的功能选择，GPIO 基址。另外，填写 pin 的使用数量 NUM_OF_CONFIGURED_PINS0。Pin 结构体如下：

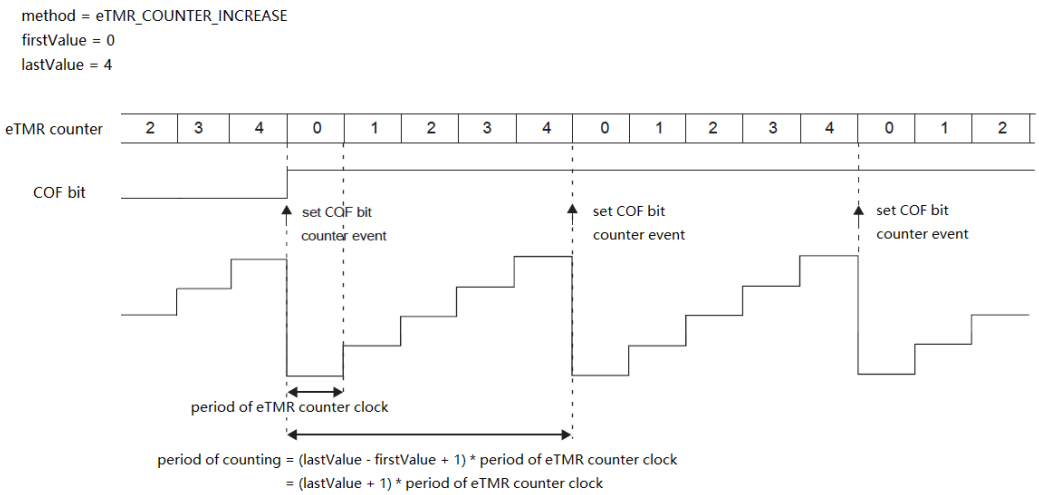
```
pin_settings_config_t g_pin_mux_InitConfigArr0[NUM_OF_CONFIGURED_PINS0] = {
    {
        .base = PCTRLC,
        .pinPortIdx = 2U,
        .pullConfig = PCTRL_INTERNAL_PULL_NOT_ENABLED,
        .passiveFilter = false,
        .mux = PCTRL_MUX_ALT2, //配置成 eTMR0 CH2
        .intConfig = PCTRL_DMA_INT_DISABLED,
        .clearIntFlag = false,
        .gpioBase = GPIOC,
        .digitalFilter = false,
    },
};
```

5 eTMR 作为普通的定时器

5.1 定时器的使用

eTMR 作为普通的定时器是通过计数器溢出中断来实现。它有两种配置：

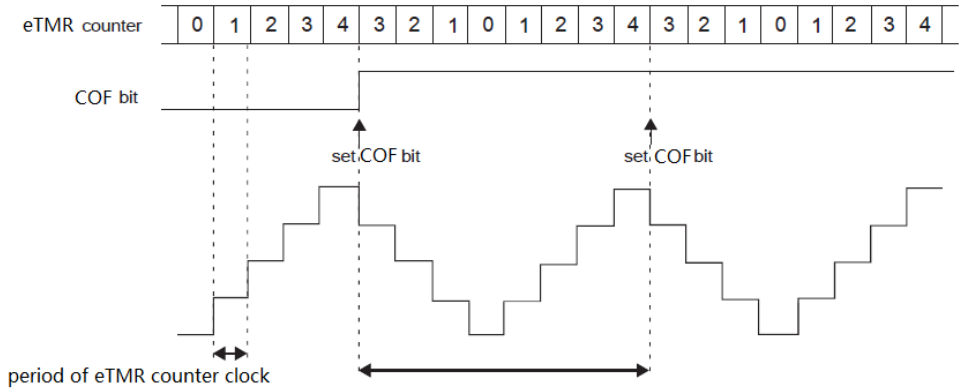
1、method 配置成 eTMR_COUNTER_INCREASE，此时计数器会从 firstValue 逐个递增计数到 lastVaule 值，计数器发生溢出中断；应该保证 firstValue 值小于 lastValue 值，因为如果不满足这个条件可能导致不可预测的行为。



2、method 配置成 eTMR_COUNTER_INCREASE_DECREASE，计数器值到达 lastValue 值时会产生 COF 中断。即首先计数器会从 firstValue 逐个递增计数到 lastVaule 值，计数器发生 COF 中断，然后向下计数到 firstValue 值，又向上计数到 lastValue 值, 计数器又发生 COF 中断, 如此反复；

举例说明：如果时钟源配置成系统时钟，频率为 48MHz，预分频选 4，那么 eTMR 的 clock 就是 12MHz。method 配成 eTMR_COUNTER_INCREASE，firstValue 配成 0，lastValue 配成 12000，那么计数器从 0 开始逐渐递增计数到 12000(即经过 1ms)，计数器发生溢出中断。

method = eTMR_COUNTER_INCREASE_DECREASE
firstValue = 0
lastValue = 4



period of counting = 2 * (lastValue - firstValue) * period of eTMR counter clock
=2 * lastValue * period of eTMR counter clock

```
etmr_config_t etmr_config = {
    /* 预分频 */
    .etmrPrescaler = eTMR_CLOCK_DIVID_BY_4,
    /* 时钟源配成系统时钟 */
    .etmrClockSource = eTMR_CLOCK_SOURCE_SYSTEMCLK,
    /*debug 模式停止计数 */
    .dbg = eTMR_DBG_MODE0,
    /* 计数器溢出中断打开 */
    .isCofIntEnabled = true,
};

etmr_timer_config_t etmr_timer_config = {
    /*method 配置计数器递增 */
    .method = eTMR_COUNTER_INCREASE,
    /*firstValue 值为 0*/
    .firstValue = 0,
    /*lastValue 值为 12000，最大是 0xFFFF*/
    .lastValue = 12000,
};
```

5.2 定时器的相关函数

```
/* 根据配置初始化计数器 */
status_t eTMR_DRV_InitCounter(uint32_t instance, const etmr_timer_config_t * config);
/* 开启计数器 */
status_t eTMR_DRV_CounterStart(uint32_t instance);
/* 停止计数器 */
status_t eTMR_DRV_CounterStop(uint32_t instance);
/* 读取计数器的计数值 */
uint32_t eTMR_DRV_CounterRead(uint32_t instance);
```

6 eTMR 用作输入捕获

6.1 输入捕获的原理:

输入捕获是通过 IP 内部两个 channel 的寄存器分别记录两次边沿变化的 count 值来计算的。在外部接线只需要接一个 channel，通过配置，IP 内部存储 count 时会自动存储在两个镜像的 channel 的 CNT 寄存器当中。

6.2 捕获信号的范围与精度

理论上，对于 16 位 eTMR，系统时钟 48MHz，预分频选 1，那么 eTMR ticks 就是 20.8ns，最小信号的宽度要取 2 倍以上的宽度，取 42ns，最大信号的宽度大约为 2.752ms；预分频选 128，那么 eTMR ticks 就是 2.67us，最小信号的宽度为 5.34us，最大信号的宽度大约为 349.9ms；因此对于 16 位 eTMR，捕获信号的范围是 42ns 到 349.9ms。
实际测量当中，SDK 能捕获的占空比 50% 的 pwm 波形的最大的频率为 20MHz。
在实际应用中，需要考虑到捕获的精度和捕获范围，例如如果想捕获精度为 1% 的 PWM 波，当系统时钟选择 48MHz, 预分频选 1，根据能捕获的占空比 50% 的 pwm 波形的最大的频率 20MHz, 即最小捕获信号宽度为 0.025us, 那么输入波形的频率应小于等于 400KHz。
预分频可以进行如下分频：1，2，4，8，16，32，64，128。值越大，精度越低。

6.3 输入捕获配置

主要配置项:

1. channelNum 输入捕获的 channel 数量;
2. maxCountValue 最大计数器值，最大值为 0xFFFF;
3. inputChannelConfig 输入捕获的 channel 的详细配置，具体如下:

- hwChannelId 通道 ID;
- inputMode 输入模式，有三种输入模式：eTMR_EDGE_DETECT（边沿检测），eTMR_SIGNAL_MEASUREMENT（信号测量）,eTMR_NO_OPERATION（不操作）。当我们需要测量周期和占空比时，将其配置成 eTMR_SIGNAL_MEASUREMENT。
- measurementType 测量类型, 有五种测量类型:

  eTMR_NO_MEASUREMENT：不测量

  eTMR_RISING_EDGE_PERIOD_MEASUREMENT：上升沿周期测量

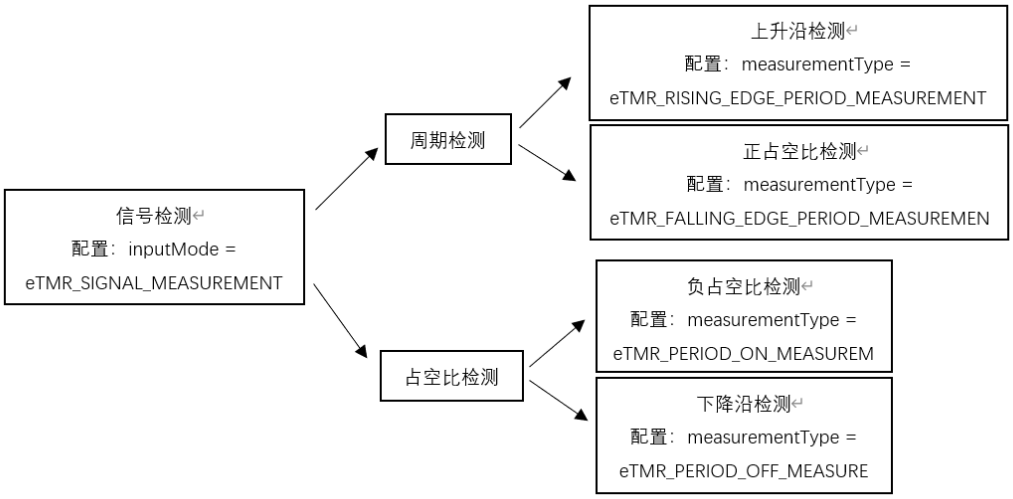
  eTMR_FALLING_EDGE_PERIOD_MEASUREMENT：下降沿周期测量

  eTMR_PERIOD_ON_MEASUREMENT：正占空比

  eTMR_PERIOD_OFF_MEASUREMENT：负占空比

  如果我们只是测量周期的话，那么可以将此项配置成 eTMR_RISING_EDGE_PERIOD_MEASUREMENT 和 eTMR_FALLING_EDGE_PERIOD_MEASUREMENT，如果我们需要测量占空比的话，那么可以将其配置成 eTMR_PERIOD_ON_MEASUREMENT 和 eTMR_PERIOD_OFF_MEASUREMENT。
- filterValue 过滤器值，只有 eTMR0 的 channel0-channel3 支持，当 filterEn 为 true 时，设置值有效。
- filterEn 过滤器使能。
- continuousModeEn 连续模式使能，改配置项控制输入捕获的连续模式，当其为 true 时，输入捕获将连续捕获，当其为 false 时，输入捕获只捕获一次。

- chnInterruptEn 通道中断使能。
- channelsCallbacksParams 通道回调函数参数。
- channelsCallbacks 通道回调函数。




```
etmr_config_t eTMR_Config =
{
    /*eTMR 操作模式 */
    .etmrMode = eTMR_MODE_INPUT_CAPTURE,
    /*eTMR 分频比，设置值 0-7，对应分频比为 1，2，4，8，16，32，64，128。*/
    .etmrPrescaler = eTMR_CLOCK_DIVID_BY_64,
    /*eTMR 时钟源选择时钟 */
    .etmrClockSource = eTMR_CLOCK_SOURCE_SYSTEMCLK,
    /* 禁止计数溢出中断使能 */
    .isCofIntEnabled = false,
    /* 禁止触发使能 */
    .initTriggerEnable = false,
};

etmr_ic_ch_config_t eTMR_ChannelInputCaptureConfig[1] =
{
    {
        .hwChannelId = 0U,
        /* 输入模式 */
        .inputMode = eTMR_SIGNAL_MEASUREMENT,
        /* 测量类型 */
        .measurementType = eTMR_PERIOD_ON_MEASUREMENT,
        .filterValue = 2U,
        .filterEnable = false,
        /* 连续模式 */
        .continuousModeEnable = false,
        .interruptEnable = false,
        .channelsCallbacksParams = NULL,
        .channelsCallbacks = NULL,
    },
};
etmr_ic_config_t eTMR_InputCaptureConfig =
{
    /* 配置的通道数 */
    .channelNum = 1U,
    /* 最大计数器的值，由于计数器是 16 位的，因此最大值为 0xFFFF*/
    .maxCountValue = 0xFFFFU,
    /* 输入通道配置 */
    .inputChannelConfig = eTMR_ChannelInputCaptureConfig,
};
```

6.4 获取捕获结果

从 eTMR 的状态结构体中获取捕获的测量结果，放在成员 measurementResults 当中。eTMR 状态结构体：

```
typedef struct
{
    /*eTMR 的时钟源 */
    etmr_clock_source_t etmrClockSource;
    /*eTMR 的操作模式 */
    etmr_mode_config_t etmrMode;
    /* 输入捕获模式下，最大计数器值 */
    uint16_t etmrMaxValue;
    /*PWM 模式下，eTMR 的 PWM 周期 */
    uint16_t etmrPeriod;
    /*eTMR 的时钟频率 */
    uint32_t etmrSourceClockFrequency;
    /* 测量结果 */
    uint16_t measurementResults [FEATURE_eTMR_CHANNEL_MAX_COUNT];
    /* 测量方式 */
    etmr_signal_measurement_mode_t measurementType;
    /* 标记测量完成的标志 */
    bool measurementComplete;
    /* 回调函数的参数 */
    void *channelsCallbacksParams[FEATURE_eTMR_CHANNEL_MAX_COUNT];
    /* 回调函数 */
    ic_callback_t channelsCallbacks[FEATURE_eTMR_CHANNEL_MAX_COUNT];
    /* 回调通知标志 */
    bool enableNotification[FEATURE_eTMR_CHANNEL_MAX_COUNT];
} etmr_state_t;
```

6.5 SDK 捕获相关函数

```
/* 初始化输入捕获 */
status_t eTMR_DRV_InitInputCapture(uint32_t instance, const etmr_ic_config_t * config);
/* 复位 eTMR 输入捕获相关寄存器 */
status_t eTMR_DRV_DeinitInputCapture(uint32_t instance,const etmr_ic_config_t * config);
/* 开始新的信号测量 */
status_t eTMR_DRV_StartNewSignalMeasurement(uint32_t instance, uint8_t channel);
/* 获取输入捕获测量结果 */
uint16_t eTMR_DRV_GetInputCaptureMeasurement(uint32_t instance, uint8_t channel);
/* 判断捕获是否完成 */
bool eTMR_DRV_IsCaptureComplete(uint32_t instance);
/* 清除捕获完成标志 */
void eTMR_DRV_ClearCaptureComplete(uint32_t instance);
/*eTMR0 中断处理函数 */
void eTMR0_IRQHandler(void);
/*eTMR1 中断处理函数 */
void eTMR1_IRQHandler(void);
/*eTMR2 中断处理函数 */
void eTMR2_IRQHandler(void);
/* 中断处理函数 */
static void eTMR_DRV_IrqHandler(uint32_t instance, uint8_t channelPair);
/* 输入捕获处理函数 */
static void eTMR_DRV_InputCaptureHandler(uint32_t instance, uint8_t channelPair);
/* 获取双沿捕获测量结果 */
status_t eTMR_DRV_GetDualEdgeCaptureMeasurement(uint32_t instance, uint8_t channel);
```

7 eTMR 用作输出比较

7.1 输出比较模式简介

eTMR 的每一个 channel 都可以用作输出比较功能，同一个 eTMR 的不同 channel 共用同一个计数器。输出比较模式下，输出信号可以配置成匹配点设置，清除或者切换三种操作。
输出比较的原理是：计数器从 0 开始计数到匹配点后，按照配置好的操作进行输出操作（设置、清除或者切换），并且计数器继续计数到最大计数值时，计数器归零，重新开始计数，如此循环，周而复始。
输出比较进行操作的时间：

$$T_{Compare\_Operate} = Compare\_Value * T_{eTMR\_ticks}$$

举例说明：如果 system clock 为 48MHz，分频选 48，假如输出最大计数值设置为 1000，输出比较值设置为 200，输出比较的操作配置成切换，那么 eTMR 的 clock 是 1M，1 个 tick 的时间是 1us，寄存器 LAST 的值是 1000-1，也就是说计数器从 0 开始计数，计到 999 后，又重新从 0 开始计数，如此反复。每当计数器计 200 时 toggle 一下，这样从输出波形上看类似是周期为 2000us, 占空比为 50% 的 PWM 波形。

7.2 配置输出比较相关参数

对于输出比较结构体，有 4 个成员变量：channelNum，maxCountValue，mode，outputChannelConfig。
1、channelNum 是输入输出比较总通道数。
2、maxCountValue 输出所用计数器最大值，就是 LAST 寄存器的值。
3、mode eTMR 模式，这里配成 eTMR_MODE_OUTPUT_COMPARE。
4、outputChannelConfig 是每个所用输出比较通道的具体配置，包含：hwChannelId，chMode，comparedValue，channelTriggerEnable。
1> hwChannelId 是通道 ID，对于不同 eTMR, 其值有所不同。eTMR0 可以配 0-7，eTMR1 和 eTMR2 可以配 0-1。
edgeAlignment 是边沿选择，有上升沿、下降沿和双沿。
2> chMode 通道输出的模式，包含设置、清除、切换和不操作输出 pin 四种模式。
3> comparedValue 比较值，当计数器的值到达比较值时，eTMR 通道会执行 chMode 设置的操作。
4> channelTriggerEnable 产生 trigger 使能配置。
举例如下：使用 eTMR0 的通道 2 和通道 3 做输出比较，两个通道的输出 pin 电平都做切换（toggle）操作。eTMR0 的最大计数值为 120，通道 2 设置比较值为 100，通道 3 设置比较值为 20。则结构体设置如下：输出比较通用配置结构体：

```
etmr_config_t eTMR_Config =
{
    .etmrPrescaler = eTMR_CLOCK_DIVID_BY_4,
    .etmrClockSource = eTMR_CLOCK_SOURCE_SYSTEMCLK,
    .dbg = eTMR_DBG_MODE0,
    .isCofIntEnabled = false,
    .initTriggerEnable = false,
};

etmr_oc_ch_config_t eTMR_OutputCompareChannelConfig[2] = {
    {
        .hwChannelId = 2,
        .chMode = eTMR_OUTPUT_TOGGLE,
        .comparedValue = 100,
        .channelTriggerEnable = false,
    },
    {
        .hwChannelId = 3,
        .chMode = eTMR_OUTPUT_TOGGLE,
        .comparedValue = 20,
        .channelTriggerEnable = false,
```

```
    },
};

etmr_oc_config_t eTMR_OutputCompareConfig = {
    .channelNum = 2,
    .mode = eTMR_MODE_OUTPUT_COMPARE,
    .maxCountValue = 120,
    .outputChannelConfig = eTMR_OutputCompareChannelConfig,
};
```

7.3 更新输出比较值

有两种方式更新输出比较值：绝对值和相对值。绝对值：当计数器的值计数到新的 compareValue 值后，输出 pin 电平按照设定好的变化。相对值：从当前计数值开始算，经过新的 compareValue 值后，输出 pin 电平按照设定好的变化。

7.4 SDK 输出比较相关函数

```
/* 初始化输出比较 */
status_t eTMR_DRV_InitOutputCompare(uint32_t instance, const etmr_oc_config_t * config);
/* 复位 eTMR 输出比较相关寄存器 */
status_t eTMR_DRV_DeinitOutputCompare(uint32_t instance, const etmr_oc_config_t * config);
/* 更新输出比较值 */
status_t eTMR_DRV_UpdateOutputCompareChannel(uint32_t instance,
                                             uint8_t channel,
                                             uint16_t nextCompareValue,
                                             etmr_oc_update_t update,
                                             bool softwareTrigger);
```

8 eTMR PWM 输出

8.1 PWM 简介

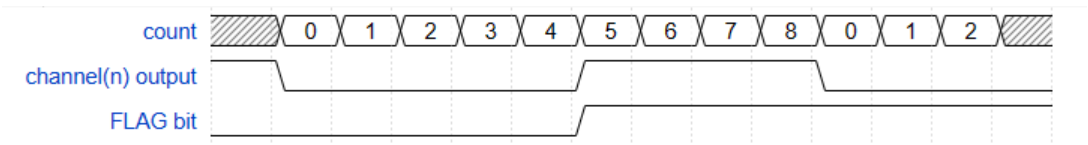
eTMR 每个通道都支持边沿对齐 PWM 模式和中心对齐 PWM 模式；每一对互补通道模式都支持死区插入；由于同一个 eTMR 的各个通道共用一个 LAST 寄存器，所以他们输出的 PWM 频率都相同。
PWM 可以设置边沿对齐和中心对齐。其中，边沿对齐又包含左对齐和右对齐；中心对齐则包含低中心对齐和高中心对齐。

1. 边沿对齐：计数器从 FIRST 寄存器值开始计时，当计数器值达到 CH_CNT 寄存器值时，CH_SCR[FLAG] 标志位置位，并产生中断，同时电平发生反转；继续计数，当计数器的值达到 LAST 寄存器值时，电平再次反转，一个周期结束；如此重复输出 PWM 波形。因此，周期取决于 LAST 和 FIRST 寄存器，脉冲宽度取决于 CH_CNT 寄存器。

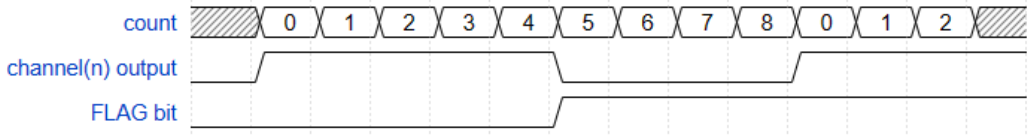
$$T_{PWM\_edgeAligned} = (V_{LAST} - V_{FIRST} + 1) * T_{eTMR\_ticks}$$

$$T_{PULSE} = (V_{CH\_CNT} - V_{FIRST} + 1) * T_{eTMR\_ticks}$$

例如：LAST = 8，CH_CNT= 5 ，右边沿对齐波形图如下：



LAST = 8，CH_CNT= 5 ，左边沿对齐波形图如下：



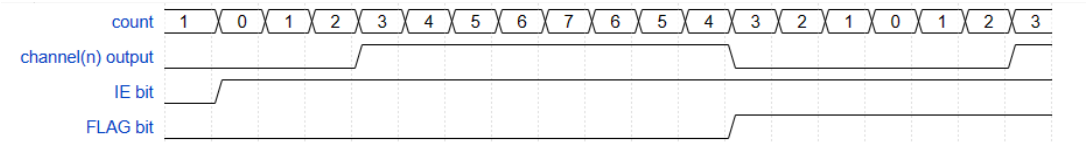
2. 中心对齐：计数器从 FIRST 寄存器值开始递增计数，计到 LAST 寄存器后，递减计数到 FIRST 寄存器值，然后又开始递增计数，如此反复重复。电平变化是在计数器达到 CH_CNT 寄存器的值时发生。CH_SCR[FLAG] 标志位在每个周期中第二次计数器达到 CH_CNT 寄存器值时置位，同时通道产生中断。

- 高中心对齐：当计数器递增到 LAST 值时，强制通道输出高电平。因此这种模式下，初始电平为低电平，当计数器递增到 CH_CNT 寄存器值时，输出电平转换为高电平，然后计数器递增到 LAST 寄存器值，再递减到 CH_CNT 寄存器值时，输出电平转换为低电平，然后计数器继续递减到 FIRST 寄 t 存器值，如此重复。
- 低中心对齐：当计数器递增到 LAST 值时，强制通道输出低电平。因此这种模式下，初始电平为高电平，当计数器递增到 CH_CNT 寄存器值时，输出电平转换为低电平，然后计数器递增到 LAST 寄存器值，再递减到 CH_CNT 寄存器值时，输出电平转换为高电平，然后计数器继续递减到 FIRST 寄存器值，如此重复。

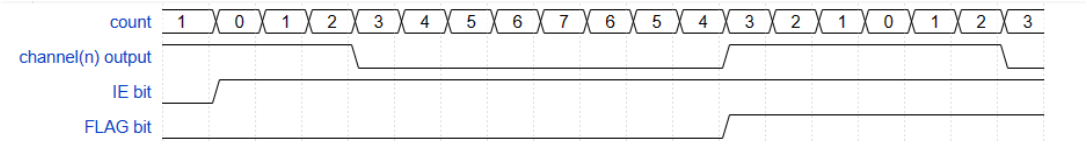
$$T_{PWM\_centerAligned} = (V_{LAST} - V_{FIRST}) * 2 * T_{eTMR\_ticks}$$

$$T_{PULSE} = (V_{CH\_CNT} - V_{FIRST}) * 2 * T_{eTMR\_ticks}$$

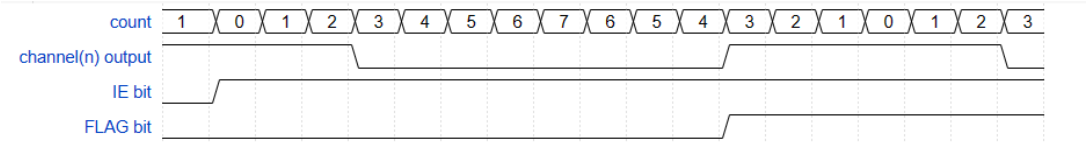
例如： LAST = 7， CH_CNT = 3，高中心对齐波形如下图：



LAST = 7， CH_CNT = 3，低中心对齐波形如下图：



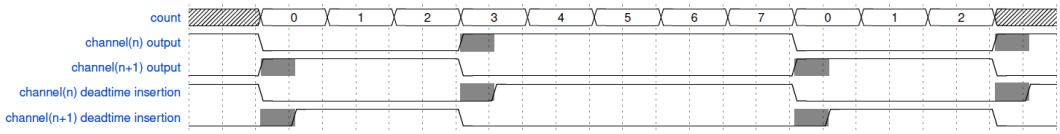
3. 占空比为 0%：当 CH_CNT 寄存器值为 0 时，通道输出 0% 占空比 PWM 波形。
4. 占空比为 100%：当 CH_CNT 寄存器值 >LAST 寄存器值时，通道输出 100% 占空比 PWM 波形。
5. 镜像模式：这种模式下，通道 n+1 和通道 n 输出完全相反的波形。在输出比较模式下，镜像模式无效。波形如下图：



6. 镜像模式死区插入：死区插入的时钟是输入时钟经过死区分频后的时钟。死区分频只能分 1，4，16 三个频率。比如，SDK 中默认配置 system clock 的频率是 48MHz，死区分频为 4，那么死区插入的时钟频率 12MHz，假设死区插入的 value 值是 12，那么死区的时间是 1us。

$$F_{PWM\_deadtimeInsertion} = \frac{F_{system\_clock}}{deadtimeDiv}$$

波形如下图：



7. PWM 更新：FIRST, LAST, CH_CNT, FORCE 和 SFO 寄存器具有影子寄存器，当配置他们的时候，实际上是将配置值写入到影子寄存器当中。我们可以通过配置来约定更新点，当到达更新点时，PWM 更新将更新这些寄存器，并强制 eTMR 计数器到 FIRST 寄存器值。

8. 更新点：当计数器方法选择为递增的方式时，更新点就是计数器从 LAST 变到 FIRST 的时候；当计数器方法选择先递增后递减时，更新点可以配置三种：

   - 如果 UCTRL[RUP] = 0x1，更新点是计数器从 FIRST 变到 FIRST+1；
   - 如果 UCTRL[RUP] = 0x2，更新点是计数器从 LAST 变到 LAST-1；
   - 如果 UCTRL[RUP] = 0x3，上面两种都是更新点。

9. 寄存器更新配置：

配置项	寄存器	位	功能
regUpdatePoint	UCTRL	RUP	**Register Update Point** 00b - Not update. 01b - Register update when counter matches FIRST value. 10b - Register update when counter matches LAST value. 11b - Register update when counter matches FIRST value and LAST value.
sfoRegUpdate	UCFG	SFOUP	**SFO Register Update Point.** 0b - SFO register(working register) is updated from shadow register immediately. 1b - SFO register(working register) is updated with shadow register update.
forceRegUpdate	UCTRL	FUP	FORCE register update point. 0b - Force update immediately. 1b - Force update follows shadow registers update.
firstRegUpdate	UCFG	FIRSTUP	**FIRST Register Update Point** 0b - FIRST register(working register) is updated from shadow register immediately. 1b - FIRST register(working register) is updated with shadow register update.
cntUpdatePoint	UCFG	SWTCNT	**Software Trigger Counter** 0b - eTMR counter update will not be enabled by a software trigger. 1b - eTMR counter update will be enabled by a software trigger.

8.2 配置 eTMR 通用时基

eTMR 通用时基配置的是 eTMR 的时钟源，预分频，模式，更新点等信息。当 eTMR 输出 PWM 波形时，必须配置更新点，而要想配置更新点，etmrMode 必须配置成 eTMR_MODE_OUTPUT_COMPARE，eTMR_MODE_PWM_CEN_ALIGN 和 eTMR_MODE_PWM_EDGE_ALIGN，这三个中任意一个都可以，在 SDK 中，只有当 eTMR 配置成这三个，才会配置更新点。

```
/* Check if the mode operation in PWM mode */
if ((eTMR_MODE_PWM_EDGE_ALIGN == config->etmrMode) || (eTMR_MODE_PWM_CEN_ALIGN == config->etmrMode) || (eTMR_MODE_OUTPUT_COMPARE == config->etmrMode))
{
    /* Configure etmr registers update mode */
    status = eTMR_DRV_ConfigUpdateMode(instance, &(config->updateConfig));
}
```

```
etmr_config_t eTMR_Config = {
    /* 更新配置 */
    {
        .softwareUpdate = true,
        .hardwareUpdate0 = false,
        .hardwareUpdate1 = false,
        .hardwareUpdate2 = false,
        /* 寄存器更新点 */
        .regUpdatePoint = eTMR_REG_LAST_UPDATE,
        /* 软件强制输出寄存器更新点配置，跟随寄存器更新点 */
        .sfoRegUpdate = eTMR_PWM_UPDATE_POINT,
        /* 强制寄存器跟随寄存器更新点更新 */
        .forceRegUpdate = eTMR_PWM_UPDATE_POINT,
        /*First 寄存器跟随寄存器更新点更新 */
        .firstRegUpdate = eTMR_PWM_UPDATE_POINT,
        .hardwareTriggerPattern = true,
        /* 软件触发计数器等待更新点更新，更新后计数器会重新置为 FIRST 寄存器值。*/
        .cntUpdatePoint = eTMR_WAIT_UPDATE_POINT,
    },
    /*eTMR 模式 */
    .etmrMode = eTMR_MODE_PWM_EDGE_ALIGN,
    /*eTMR 预分频 */
    .etmrPrescaler = eTMR_CLOCK_DIVID_BY_4,
    /* 选择系统时钟 */
    .etmrClockSource = eTMR_CLOCK_SOURCE_SYSTEMCLK,
    .dbg = eTMR_DBG_MODE0,
    .isCofIntEnabled = false,
    .initTriggerEnable = false,
};
```

8.3 配置 PWM 相关参数

```
/*PWM 通道配置 */
etmr_pwm_ch_config_t eTMR_PwmChannelConfig[2] = {
    {
        /* 通道号 */
        .hwChannelId = 2,
        /* 占空比，dutyCycle/0x8000，就是占空比。如果 dutyCycle 是 0x8000，那么就是 100%，如果是 0x00，那么占空比是 0%。*/
        .dutyCycle = 0x7000,
        /* 高中心对齐 */
        .align = eTMR_PWM_CENTER_HIGH_ALIGN,
        /* 初始输出状态，可以配置成 0 或 1*/
        .initialOutput = 0,
        /* 反转状态 */
        .invertState = eTMR_NOT_INVERT,
        /* 镜像模式使能 */
        .mirrorModeEnable = false,
        /* 镜像通道反转状态 */
        .mirrorChannelInvertState = eTMR_MIRROR_NOT_INVERTED,
        /* 插入死区使能 */
        .deadtimeInsertEnable = false,
        /* 通道触发使能 */
        .channelTriggerEnable = false,
    },
    {
        .hwChannelId = 3, // CH3
        .dutyCycle = 0x2000,
        .align = eTMR_PWM_CENTER_HIGH_ALIGN,
        .initialOutput = 0,
        .invertState = eTMR_NOT_INVERT,
        .mirrorModeEnable = false,
```

```
        .mirrorChannelInvertState = eTMR_MIRROR_NOT_INVERTED,
        .deadtimeInsertEnable = false,
        .channelTriggerEnable = false,
    }
};

/* 故障配置 */
etmr_fault_config_t eTMR_FaultConfig = {
    /* 故障中断使能 */
    .faultIntEnable = false,
    /* 故障滤波值 */
    .faultFilterValue = 0,
    /* 故障模式 */
    .faultMode = eTMR_FAULT_CONTROL_DISABLED,
};

etmr_pwm_config_t eTMR_PwmConfig = {
    /*PWM 频率 */
    .frequency = 32000U,
    /*PWM 输出模式，可配成 eTMR_MODE_PWM_EDGE_ALIGN 和 eTMR_MODE_PWM_CEN_ALIGN*/
    .mode = eTMR_MODE_PWM_CEN_ALIGN,
    /*PWM 通道数 */
    .pwmChannelNum = 2U,
    /*PWM 通道配置 */
    .pwmChannelConfig = eTMR_PwmChannelConfig,
    /* 插入死区的计数值 */
    .deadtimeValue = 0U,
    /* 插入死区分频 */
    .deadtimeDiv = eTMR_DEADTIME_DIVID_BY_1,
    /* 故障配置 */
    .faultConfig = &eTMR_FaultConfig,
};
```

8.4 SDK 中 PWM 输出相关函数

```
/* 初始化 PWM 独立通道 */
static void eTMR_DRV_InitPwmIndependentChannel(uint32_t instance,
                                               const etmr_pwm_config_t *config);
/* 初始化 PWM 通道占空比 */
static status_t eTMR_DRV_InitPwmDutyCycleChannel(uint32_t instance,
                                                 const etmr_pwm_config_t *config);
/* 初始化 PWM*/
status_t eTMR_DRV_InitPwm(uint32_t instance, const etmr_pwm_config_t *config);
/* 复位 PWM 相关寄存器 */
status_t eTMR_DRV_DeinitPwm(uint32_t instance);
/* 更新 PWM 通道占空比 */
status_t eTMR_DRV_UpdatePwmChannel(uint32_t instance,
                                   uint8_t channel,
                                   etmr_pwm_align_t align,
                                   etmr_pwm_update_option_t typeOfUpdate,
                                   uint16_t edge,
                                   bool softwareTrigger);
/* 更新 PWM 周期 */
status_t eTMR_DRV_UpdatePwmPeriod(uint32_t instance,
                                  etmr_pwm_update_option_t typeOfUpdate,
                                  uint32_t newValue,
                                  bool softwareTrigger);
/* 快速更新多通道 PWM 占空比，它与函数 eTMR_DRV_UpdatePwmChannel 相比的主要区别是执行速度比较快，缺点是，不太灵活，只接收 tick 为单位的参数。*/
status_t eTMR_DRV_FastUpdatePwmChannels(uint32_t instance,
                                        uint8_t numberOfChannels,
                                        const uint8_t * channels,
                                        const uint16_t * duty,
                                        bool softwareTrigger);
```

9 代码应用实例

关于输出比较的完整代码，请参考 LD0 SDK 中的相关 demo。

版权与联系方式

联系我们:
主页: www.ytmicro.com