SPC1168 ECAP 使用指南

概述

增强型捕获（ECAP）模块用于需要准确计时外部事件的情况，以及对外部输入的信号的占空比以及周期信息进行解码。

目录

图片列表

版本历史

版本	日期	作者	状态	变更
A/0	2023-06-15	CanChai	Outdated	首次发布。
C/0	2024-04-22	Jiali Zhou	Released	修改排版格式。

术语或缩写

术语或缩写	描述
MCU	Microcontroller Unit，微控制器单元
ECAP	Enhanced capture，增强型捕获单元

1 ECAP 特性

SPC1168 内建一个 Enhanced capture (ECAP)单元，用于抓取外部事件发生时的精确时间点。

SPC1168 的 ECAP 单元可以用作以下功能：

- 测量旋转机械的转速；
- 测量脉冲信号的周期及占空比；
- 解码加密的占空比信息；

SPC1168 的 ECAP 单元有以下特点：

- 基于 32bit 的计时器，且精度可达 5ns（时钟为 200MHz）；
- 灵活的输入捕获引脚：任意 GPIO 均可配置为捕获引脚；
- 4 个时间标签捕获寄存器；
- 4 个事件均可独立选择边沿极性（上升/下降沿）；
- 4 个事件均可支持中断；
- 持续捕获模式，在最多 4 个捕获事件间持续轮流执行；
- 绝对时间戳捕获；
- 相对时间戳捕获；
- 若不需要捕获功能，可以将 ECAP 配置为简单的 PWM 工作模式；

2 ECAP 实例

如本文档第一章描述，SPC1168 内置的 ECAP 可连续捕获 4 个事件的时间戳信息，两种模式（捕获模式和 APWM 模式）。

APWM 模式：作为 APWM 模式时，ECAP 可以当作 PWM 功能进行输出波形，CAP0 作为 APWM0 的周期寄存器，CAP1 作为 APWM 的比较寄存器，CAP2 和 CAP3 作为 CAP0 和 CAP1 的影子寄存器，在 CNT=PRD 时会更新 CAP0 和 CAP1。

捕获模式：捕获某个事件，并记录事件发生时的时间戳。

在捕获模式下 ECAP 根据是否会循环更新 CAPx 寄存器的数值，其捕获事件的行为又可分为连续捕获和一次性捕获。

连续捕获：计数器从 0 开始连续递增计数，当事件触发时，会将计数器当成时间戳按事件次序依次锁存到 CAPx，且新的对应事件的时间戳会覆盖旧的 CAPx 数值。

一次性捕获：计数器从 0 开始连续递增计数，当事件触发时，会将计数器当成时间戳按事件次序依次锁存到 CAPx，但当事件发生的个数达到预设数值时，将停止计数，并且也不会进一步更新 CAPx 的数值。

比较重要的是，在捕获模式下，ECAP 记录事件发生时机的时间戳计数器，其计数方式又分为绝对时间戳计数和相对时间戳计数。

绝对时间戳计数：计数器从 0 开始连续递增计数，当最后事件触发时，计数器将会复位，从 0 开始计数。

相对时间戳计数：计数器从 0 开始连续递增计数，当有事件触发时，计数器将会复位，从 0 开始计数。

计数器的这两种计数方式，在连续捕获和一次性捕获中均可使用。

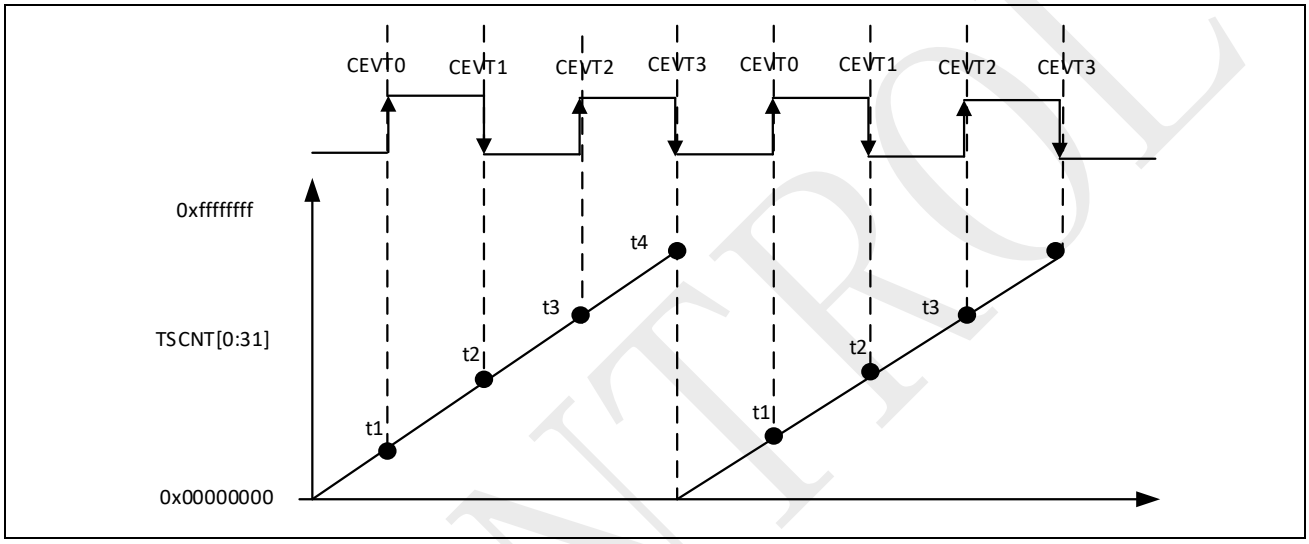
2.1 捕获模式

2.1.1 连续捕获

2.1.1.1 绝对时间戳计数

本示例中演示对 PWM 波进行连续捕获，以绝对时间戳方式计数。对 PWM 信号的捕获信息如图 2-1，根据捕获的信息可以推出波形的周期 T= t3 – t1 或 t4 – t2、波形为高占空 = t2 – t1 或 t4 – t3、波形为低占空 = t3 – t2。

图 2-1：连续捕获（绝对时间戳计数）



如下示例将捕获 PWM5 输出一路波形，并对 PWM5 输入的波形进行解析计算频率。其配置流程如下：

- 初始化系统时钟，UART 调试口；
- 初始化 PWM5 并生成一路波形；
- 调用 ECAP_CaptureModeInit()函数初始化 ECAP 为捕获模式，并捕获 GPIO28 引脚，GPIO28 为 PWM 产生的一路波形进行捕获，ECAP_CaptureModeInit 函数默认捕获事件为 CAP0 上升沿捕获、CAP1 下降沿捕获、CAP2 上升沿捕获、CAP3 下降沿捕获；
- 调用 PIN_EnableInputChannel()函数设置 GPIO28 为输入功能，将 GPIO28 引脚的 PWM 信号送入 ECAP；
- 如果输入信号存在干扰，可以调用使能滤波函数 GPIO_EnableDeglitch()和设置滤波函数窗口 GPIO_SetDeglitchWindow()进行滤波，减少干扰带来的误差；
- 调用 ECAP_EnableEventResetCounter()函数使能 ECAP_CEVT3，当事件 3（CAP3 捕获下降沿）发生，ECAP 的计数器将进行复位，重新开始计数；
- 调用 ECAP_EnableInt()函数使能 EVT3 事件中断，当事件 3 发生将会产生中断，然后在中断服务函数中通过时间戳寄存器 CAPx 进行计算波形的频率；

连续捕获(绝对时间戳计数)

```
/* This macro is used to get the PWM period with a specified frequency */
#define PWMPeriod(u32PWMFreqHz)
((CLOCK_GetModuleClock(PWM_MODULE))/u32PWMFreqHz)/2;

#define PWM_FREQ                                        10000
/* 10kHz PWM */

uint32_t TStamp0, TStamp1, TStamp2, TStamp3;

/*usd the count to calculate the period*/
#define CNTTOFREQ(x) (CLOCK_GetModuleClock(ECAP_MODULE) / (x))

uint32_t u32PWMPeriod;

int main()
{
    FLASH_WALLOW();

    FLASH_SetTiming(200000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

    Delay_Init();

    /*
     * Init the UART
     *
     * 1.Set the GPIO34/35 as UART FUNC
     *
     * 2.Enable the UART CLK
     *
     * 3.Set the rest para
     */
    GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
    GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
    CLOCK_EnableModule(UART_MODULE);
    UART_Init(UART, 38400);

    /* Init PWM5, PWM freq will be set as 10kHz in this function */
    PWM_SingleChannelInit(PWM5, PWM_CHA, PWM_FREQ);

    /* Set PWM5A output 50% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM5, u32PWMPeriod / 2);

    /* Start PWM5 */
    PWM_RunCounter(PWM5);

    /* Select GPIO28 as the channel A output of PWM5 */
    GPIO_SetPinChannel(GPIO_28, GPIO28_PWM5A);

    /* Enable Pin input deglitch filter */
    GPIO_EnableDeglitch(GPIO_28);

    /* The smaller the value, the more accurate the measurement result, but the
more susceptible to interference */
    GPIO_SetDeglitchWindow(DGCLKCTL_ALL_DIV_(0x0));

    /* ECAP Init */
```

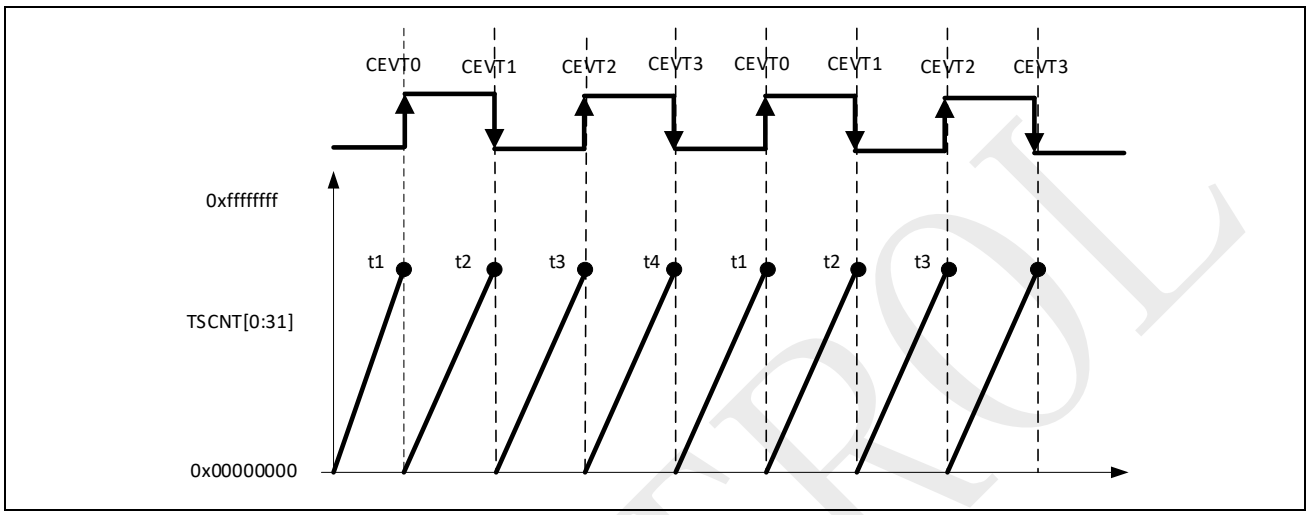
连续捕获(绝对时间戳计数)

```
    ECAP_CaptureModeInit(ECAP, GPIO_28);

    /* Set the count reset back to zero when event3 happened */
    ECAP_DisableEventResetCounter(ECAP, ECAP_CEVT0);
    ECAP_DisableEventResetCounter(ECAP, ECAP_CEVT1);
    ECAP_DisableEventResetCounter(ECAP, ECAP_CEVT2);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT3);

    /* Enable EVT3 Interrupt */
    ECAP_EnableInt(ECAP, ECAP_INT_CEVT3);

    /* Enable global interrupt of EACP */
    NVIC_EnableIRQ(ECAP_IRQn);

    /* Start ECAP */
    ECAP_RunCounter(ECAP);

    while (1)
    {

    }
}

void ECAP_IRQHandler(void)
{
    uint32_t cnt;

    TStamp0 = ECAP->CAP0.all;
    TStamp1 = ECAP->CAP1.all;
    TStamp2 = ECAP->CAP2.all;
    TStamp3 = ECAP->CAP3.all;

    cnt = ((TStamp2 - TStamp0) + (TStamp3 - TStamp1)) / 2;
    printf("Frequency = %dHz\n", CNTTOFREQ(cnt));

    /* Clear CEVT3 */
    ECAP_ClearCEVT3Int(ECAP);

    /* Clear global INT of ECAP */
    ECAP_ClearGlobalInt(ECAP);
}
```

2.1.1.2 相对时间戳计数

本示例中演示对 PWM 波进行连续捕获，以相对时间戳方式计数。对 PWM 信号的捕获信息如图 2-2，根据捕获的信息可以推出波形的周期 T= t2 + t3 或 t3 + t4、波形为高占空 = t2 或 t4、波形为低占空 = t3。

图 2-2：连续捕获（相对时间戳计数）



如下示例将捕获 PWM5 输出一路波形，并对 PWM5 输入的波形进行解析计算频率。其配置流程如下：

- 初始化系统时钟，UART 调试口；
- 初始化 PWM5 并生成一路波形；
- 调用 ECAP_CaptureModeInit()函数初始化 ECAP 为捕获模式，并捕获 GPIO28 引脚，GPIO28 为 PWM 产生的一路波形进行捕获，ECAP_CaptureModeInit 函数默认捕获事件为 CAP0 上升沿捕获、CAP1 下降沿捕获、CAP2 上升沿捕获、CAP3 下降沿捕获；
- 调用 PIN_EnableInputChannel()函数设置 GPIO28 为输入功能，将 GPIO28 引脚的 PWM 信号送入 ECAP；
- 如果输入信号存在干扰，可以调用使能滤波函数 GPIO_EnableDeglitch()和设置滤波函数窗口 GPIO_SetDeglitchWindow()进行滤波，减少干扰带来的误差；
- 调用 ECAP_EnableEventResetCounter()函数使能 ECAP_CEVT0~3，当事件发生，ECAP 的计数器都进行复位，重新开始计数；
- 调用 ECAP_EnableInt()函数使能 EVT3 事件中断，当事件发生将会产生中断，然后在中断服务函数中通过时间戳寄存器 CAPx 进行计算波形的频率；

连续捕获(相对时间戳计数)

```
/* This macro is used to get the PWM period with a specified frequency */
#define PWMPeriod(u32PWMFreqHz)
((CLOCK_GetModuleClock(PWM_MODULE))/u32PWMFreqHz)/2;

#define PWM_FREQ                                        10000
/* 10kHz PWM */

uint32_t TStamp0, TStamp1, TStamp2, TStamp3;

/*usd the count to calculate the period*/
#define CNTTOFREQ(x) (CLOCK_GetModuleClock(ECAP_MODULE) / (x))

uint32_t u32PWMPeriod;

int main()
{
    FLASH_WALLOW();

    FLASH_SetTiming(200000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

    Delay_Init();

    /*
     * Init the UART
     *
     * 1.Set the GPIO34/35 as UART FUNC
     *
     * 2.Enable the UART CLK
     *
     * 3.Set the rest para
     */
    GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
    GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
    CLOCK_EnableModule(UART_MODULE);
    UART_Init(UART, 38400);

    /* Init PWM5, PWM freq will be set as 10kHz in this function */
    PWM_SingleChannelInit(PWM5, PWM_CHA, PWM_FREQ);

    /* Set PWM5A output 50% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM5, u32PWMPeriod / 2);

    /* Start PWM5 */
    PWM_RunCounter(PWM5);

    /* Select GPIO28 as the channel A output of PWM5 */
    GPIO_SetPinChannel(GPIO_28, GPIO28_PWM5A);

    /* Enable Pin input deglitch filter */
    GPIO_EnableDeglitch(GPIO_28);

    /* The smaller the value, the more accurate the measurement result, but the
more susceptible to interference */
    GPIO_SetDeglitchWindow(DGCLKCTL_ALL_DIV_(0x0));

    /* ECAP Init */
```

连续捕获(相对时间戳计数)

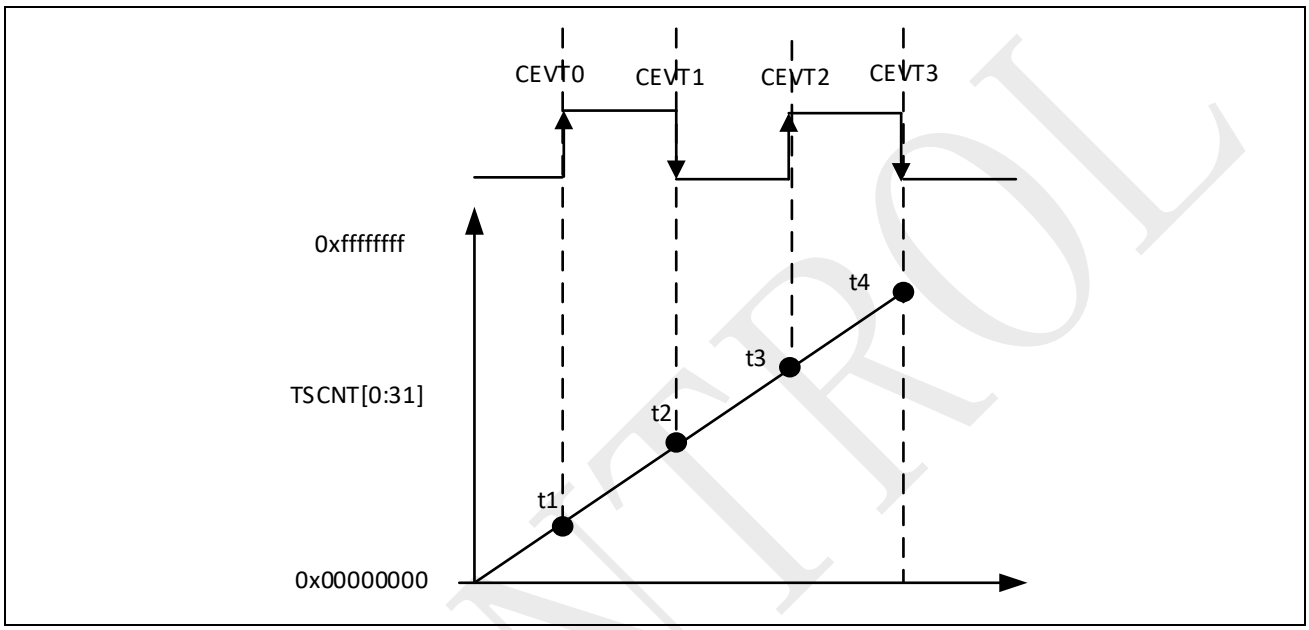
```
    ECAP_CaptureModeInit(ECAP, GPIO_28);

    /* Set the count reset back to zero when event happened */
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT0);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT1);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT2);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT3);

    /* Enable EVT3 Interrupt */
    ECAP_EnableInt(ECAP, ECAP_INT_CEVT3);

    /* Enable global interrupt of EACP */
    NVIC_EnableIRQ(ECAP_IRQn);

    /* Start ECAP */
    ECAP_RunCounter(ECAP);

    while (1)
    {

    }
}

void ECAP_IRQHandler(void)
{
    uint32_t cnt;

    TStamp0 = ECAP->CAP0.all;
    TStamp1 = ECAP->CAP1.all;
    TStamp2 = ECAP->CAP2.all;
    TStamp3 = ECAP->CAP3.all;

    cnt = ((TStamp2 + TStamp1) + (TStamp3 + TStamp2)) / 2;
    printf("Frequency = %dHz\n", CNTTOFREQ(cnt));

    /* Clear CEVT3 */
    ECAP_ClearCEVT3Int(ECAP);

    /* Clear global INT of ECAP */
    ECAP_ClearGlobalInt(ECAP);
}
```

2.1.2 单次模式

2.1.2.1 绝对时间戳计数

本示例中演示对 PWM 波进行单次捕获，以绝对时间戳方式计数。对 PWM 信号的捕获信息如图 2-3，根据捕获的信息可以推出波形的周期 T= t3 – t1 或 t4 - t2、波形为高占空 = t2 – t1 或 t4 - t3、波形为低占空 = t3 – t2。

图 2-3：单次捕获（绝对时间戳计数）



如下示例将捕获 PWM5 输出一路波形，并对 PWM5 输入的波形进行解析计算频率。其配置流程如下：

- 初始化系统时钟，UART 调试口；
- 初始化 PWM5 并生成一路波形；
- 调用 ECAP_CaptureModeInit()函数初始化 ECAP 为捕获模式，并捕获 GPIO28 引脚，GPIO28 为 PWM 产生的一路波形进行捕获，ECAP_CaptureModeInit 函数默认捕获事件为 CAP0 上升沿捕获、CAP1 下降沿捕获、CAP2 上升沿捕获、CAP3 下降沿捕获；
- 调用 ECAP_EnableOneshotMode()函数进行配置 ECAP 为单次模式，否则为连续模式；
- 调用 ECAP_EnableEventResetCounter()函数使能 ECAP_CEVT3，当事件 3（CAP3 捕获下降沿）发生，ECAP 的计数器将进行复位，重新开始计数；
- 调用 PIN_EnableInputChannel()函数设置 GPIO28 为输入功能，将 GPIO28 引脚的 PWM 信号送入 ECAP；
- 如果输入信号存在干扰，可以调用使能滤波函数 GPIO_EnableDeglitch()和设置滤波函数窗口 GPIO_SetDeglitchWindow()进行滤波，减少干扰带来的误差；
- 调用 ECAP_EnableInt()函数使能 EVT3 事件中断，当事件 3 发生将会产生中断，然后在中断服务函数中通过时间戳寄存器 CAPx 进行计算波形的频率；
- 可以通过调用 ECAP_OneshotReArm()使能下一次单次循环。

单次模式(绝对时间戳捕获)

```
/* This macro is used to get the PWM period with a specified frequency */
#define PWMPeriod(u32PWMFreqHz)
((CLOCK_GetModuleClock(PWM_MODULE))/u32PWMFreqHz)/2;

/* Usd the count to calculate the period */
#define CNTTOFREQ(x) (CLOCK_GetModuleClock(ECAP_MODULE) / (x))

#define PWM_FREQ                                10000
/* 10kHz PWM */
#define ECAP_PIN                                GPIO_28
/* ECAP PIN */
#define ECAP_PIN_GPIO_FUNC                       GPIO28_GPIO28
/* ECAP PIN act as GPIO function */
#define ECAP_PIN_PWM_FUNC                        GPIO28_PWM5A
/* ECAP PIN act as PWM function */
#define PWM_Sel                                 PWM5


uint32_t u32PWMPeriod;
uint32_t TStamp0, TStamp1, TStamp2, TStamp3;

int main()
{
    FLASH_WALLOW();

#if defined(SPC1158)
    FLASH_SetTiming(100000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_100MHZ);
#else
    FLASH_SetTiming(200000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
#endif

    Delay_Init();

    /*
     * Init the UART
     *
     * 1.Set the GPIO34/35 as UART FUNC
     *
     * 2.Enable the UART CLK
     *
     * 3.Set the rest para
     */
    GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
    GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
    CLOCK_EnableModule(UART_MODULE);
    UART_Init(UART, 38400);

    /* Init PWM_Sel, PWM freq will be set as 10kHz in this function */
    PWM_SingleChannelInit(PWM_Sel, PWM_CHA, PWM_FREQ);

    /* Set PWM_SelA output 50% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM_Sel, u32PWMPeriod / 2);
```

单次模式(绝对时间戳捕获)

```
    /* ECAP Init */
    ECAP_CaptureModeInit(ECAP, ECAP_PIN);

    /* Set ECAP as onshot mode */
    ECAP_EnableOneshotMode(ECAP);

    /* Set EVT DIV to 1 */
    ECAP_SetEventDiv(ECAP, 1);

    /* ---------------Steps for fixing BUG : BEGIN------------------------ */
    /*
     * As described in technical reference manual, the ECAP event prescaler is
implemented
     * with existing clock divider IP, so there is extra four event cycles
(regarded as clock
     * cycles to the divider) required to initialize the divider after power-up.
For one shot
     * mode, in which the user cares about exactly each incoming event, the
first 4 events may
     * not work correctly. So we need software work-around to purposely generate
4 dummy events
     * via GPIO toggling in Gen 3 products to initialize the event prescaler.
     */

    /*
     * EVT INT are opened in 'ECAP_CaptureModeInit', the 4 dummy events may
cause dummy INT,
     * so disable all EVT INT when sending the 4 dummy event to the ECAP module.
     */
    ECAP_DisableCEVT0Int(ECAP);
    ECAP_DisableCEVT1Int(ECAP);
    ECAP_DisableCEVT2Int(ECAP);
    ECAP_DisableCEVT3Int(ECAP);

    /* Set the ECAP PIN as the INPUT GPIO function in order to sent 4 dummy
event */
    GPIO_SetPinChannel(ECAP_PIN, ECAP_PIN_GPIO_FUNC);
    GPIO_WritePin(ECAP_PIN, GPIO_LEVEL_LOW);
    GPIO_SetPinDir(ECAP_PIN, GPIO_OUTPUT);

    /* Toggle the ECAP PIN 4 times as 4 events */
    GPIO_TogglePin(ECAP_PIN);
    GPIO_TogglePin(ECAP_PIN);
    GPIO_TogglePin(ECAP_PIN);
    GPIO_TogglePin(ECAP_PIN);

    /* Dummy Event may cause accidental INT, So clear all INT there */
    ECAP->CAPIC.all = 0xFFFFFFFF;

    /* RE-arm ECAP event count register(in order to clear the impact of the 4
dummy events) */
    ECAP_OneshotReArm(ECAP);
    /* ---------------Steps for fixing BUG : END------------------------ */

    /* Select ECAP_PIN as the channel A output of PWM_Sel */
    GPIO_SetPinChannel(ECAP_PIN, ECAP_PIN_PWM_FUNC);

    /* Enable Pin input deglitch filter */
    GPIO_EnableDeglitch(ECAP_PIN);
```

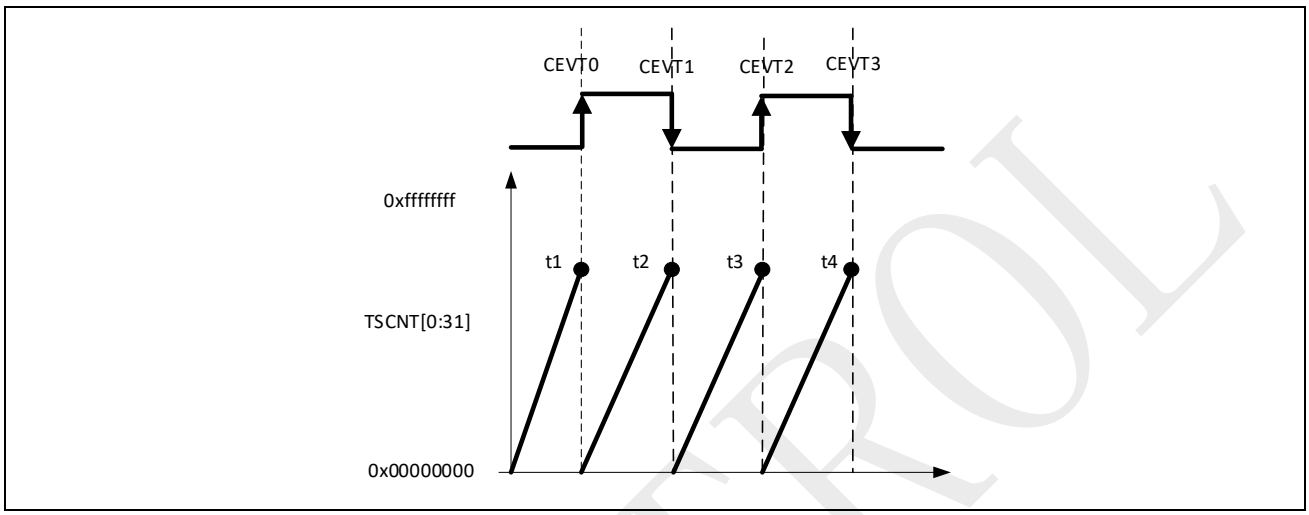
单次模式(绝对时间戳捕获)

```
    /* The smaller the value, the more accurate the measurement result, but the
more susceptible to interference */
    GPIO_SetDeglitchWindow(DGCLKCTL_ALL_DIV_(0x0));

    /* Set the count reset back to zero when event3 happened */
    ECAP_DisableEventResetCounter(ECAP, ECAP_CEVT0);
    ECAP_DisableEventResetCounter(ECAP, ECAP_CEVT1);
    ECAP_DisableEventResetCounter(ECAP, ECAP_CEVT2);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT3);

    /* Enable EVT3 Interrupt */
    ECAP_EnableInt(ECAP, ECAP_INT_CEVT3);

    /* Enable global interrupt of EACP */
    NVIC_EnableIRQ(ECAP_IRQn);

    /* Start ECAP */
    ECAP_RunCounter(ECAP);

    /* Start PWM_Sel */
    PWM_RunCounter(PWM_Sel);

    while (1)
    {

    }
}

void ECAP_IRQHandler(void)
{
    uint32_t cnt;

    TStamp0 = ECAP->CAP0.all;
    TStamp1 = ECAP->CAP1.all;
    TStamp2 = ECAP->CAP2.all;
    TStamp3 = ECAP->CAP3.all;

    cnt = ((TStamp2 - TStamp0) + (TStamp3 - TStamp1)) / 2;
    printf("Frequency = %dHz\n", CNTTOFREQ(cnt));

    /* Clear CEVT3 */
    ECAP_ClearCEVT3Int(ECAP);

    /* Clear global INT of ECAP */
    ECAP_ClearGlobalInt(ECAP);

    /* Rearm the ECAP for next events */
    ECAP_OneshotReArm(ECAP);
}
```

2.1.2.2 相对时间戳计数

本示例中演示对 PWM 波进行单次捕获，以相对时间戳方式计数。对 PWM 信号的捕获信息如图 2-4，根据捕获的信息可以推出波形的周期 T= t2 + t3、波形为高占空 = t2、波形为低占空 = t3。

图 2-4：单次模式（相对时间戳计数）



如下示例将捕获 PWM5 输出一路波形，并对 PWM5 输入的波形进行解析计算频率。其配置流程如下：

- 初始化系统时钟，UART 调试口；
- 初始化 PWM5 并生成一路波形；
- 调用 ECAP_CaptureModeInit()函数初始化 ECAP 为捕获模式，并捕获 GPIO28 引脚，GPIO28 为 PWM 产生的一路波形进行捕获，ECAP_CaptureModeInit 函数默认捕获事件为 CAP0 上升沿捕获、CAP1 下降沿捕获、CAP2 上升沿捕获、CAP3 下降沿捕获；
- 调用 ECAP_EnableOneshotMode()函数进行配置 ECAP 为单次模式，否则为连续模式；
- 调用 ECAP_EnableEventResetCounter()函数使能 ECAP_CEVT0~3，当事件发生，ECAP 的计数器都进行复位，重新开始计数；
- 调用 PIN_EnableInputChannel()函数设置 GPIO28 为输入功能，将 GPIO28 引脚的 PWM 信号送入 ECAP；
- 如果输入信号存在干扰，可以调用使能滤波函数 GPIO_EnableDeglitch()和设置滤波函数窗口 GPIO_SetDeglitchWindow()进行滤波，减少干扰带来的误差；
- 调用 ECAP_EnableInt()函数使能 EVT3 事件中断，当事件 3 发生将会产生中断，然后在中断服务函数中通过时间戳寄存器 CAPx 进行计算波形的频率；
- 可以通过调用 ECAP_OneshotReArm()使能下一次单次循环。

单次模式(相对时间戳计数)

```
/* This macro is used to get the PWM period with a specified frequency */
#define PWMPeriod(u32PWMFreqHz)
((CLOCK_GetModuleClock(PWM_MODULE))/u32PWMFreqHz)/2;

/* Usd the count to calculate the period */
#define CNTTOFREQ(x) (CLOCK_GetModuleClock(ECAP_MODULE) / (x))

#define PWM_FREQ                                10000
/* 10kHz PWM */
#define ECAP_PIN                                GPIO_28
/* ECAP PIN */
#define ECAP_PIN_GPIO_FUNC                       GPIO28_GPIO28
/* ECAP PIN act as GPIO function */
#define ECAP_PIN_PWM_FUNC                        GPIO28_PWM5A
/* ECAP PIN act as PWM function */
#define PWM_Sel                                 PWM5


uint32_t u32PWMPeriod;
uint32_t TStamp0, TStamp1, TStamp2, TStamp3;

int main()
{
    FLASH_WALLOW();

#if defined(SPC1158)
    FLASH_SetTiming(100000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_100MHZ);
#else
    FLASH_SetTiming(200000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);
#endif

    Delay_Init();

    /*
     * Init the UART
     *
     * 1.Set the GPIO34/35 as UART FUNC
     *
     * 2.Enable the UART CLK
     *
     * 3.Set the rest para
     */
    GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
    GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
    CLOCK_EnableModule(UART_MODULE);
    UART_Init(UART, 38400);

    /* Init PWM_Sel, PWM freq will be set as 10kHz in this function */
    PWM_SingleChannelInit(PWM_Sel, PWM_CHA, PWM_FREQ);

    /* Set PWM_SelA output 50% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM_Sel, u32PWMPeriod / 2);
```

单次模式(相对时间戳计数)

```
    /* ECAP Init */
    ECAP_CaptureModeInit(ECAP, ECAP_PIN);

    /* Set ECAP as onshot mode */
    ECAP_EnableOneshotMode(ECAP);

    /* Set EVT DIV to 1 */
    ECAP_SetEventDiv(ECAP, 1);

    /* ---------------Steps for fixing BUG : BEGIN------------------------ */
    /*
     * As described in technical reference manual, the ECAP event prescaler is
implemented
     * with existing clock divider IP, so there is extra four event cycles
(regarded as clock
     * cycles to the divider) required to initialize the divider after power-up.
For one shot
     * mode, in which the user cares about exactly each incoming event, the
first 4 events may
     * not work correctly. So we need software work-around to purposely generate
4 dummy events
     * via GPIO toggling in Gen 3 products to initialize the event prescaler.
     */

    /*
     * EVT INT are opened in 'ECAP_CaptureModeInit', the 4 dummy events may
cause dummy INT,
     * so disable all EVT INT when sending the 4 dummy event to the ECAP module.
     */
    ECAP_DisableCEVT0Int(ECAP);
    ECAP_DisableCEVT1Int(ECAP);
    ECAP_DisableCEVT2Int(ECAP);
    ECAP_DisableCEVT3Int(ECAP);

    /* Set the ECAP PIN as the INPUT GPIO function in order to sent 4 dummy
event */
    GPIO_SetPinChannel(ECAP_PIN, ECAP_PIN_GPIO_FUNC);
    GPIO_WritePin(ECAP_PIN, GPIO_LEVEL_LOW);
    GPIO_SetPinDir(ECAP_PIN, GPIO_OUTPUT);

    /* Toggle the ECAP PIN 4 times as 4 events */
    GPIO_TogglePin(ECAP_PIN);
    GPIO_TogglePin(ECAP_PIN);
    GPIO_TogglePin(ECAP_PIN);
    GPIO_TogglePin(ECAP_PIN);

    /* Dummy Event may cause accidental INT, So clear all INT there */
    ECAP->CAPIC.all = 0xFFFFFFFF;

    /* RE-arm ECAP event count register(in order to clear the impact of the 4
dummy events) */
    ECAP_OneshotReArm(ECAP);
    /* ---------------Steps for fixing BUG : END------------------------ */

    /* Select ECAP_PIN as the channel A output of PWM_Sel */
    GPIO_SetPinChannel(ECAP_PIN, ECAP_PIN_PWM_FUNC);

    /* Enable Pin input deglitch filter */
    GPIO_EnableDeglitch(ECAP_PIN);
```

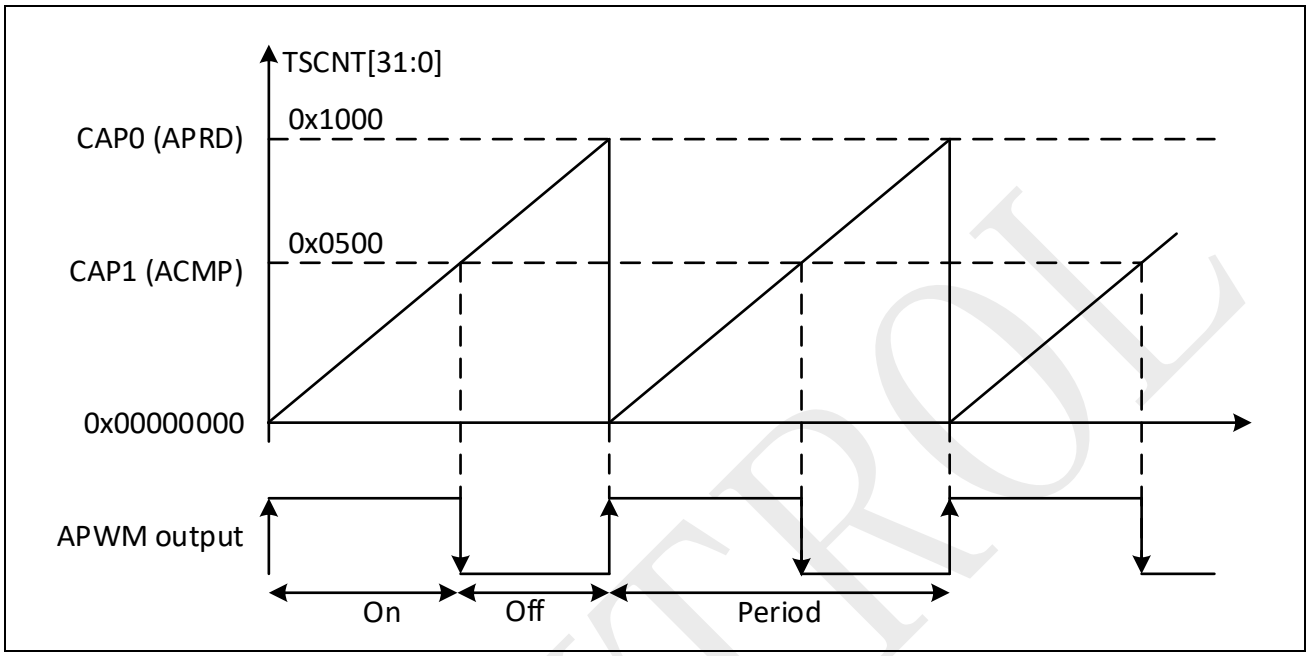
单次模式(相对时间戳计数)

```
    /* The smaller the value, the more accurate the measurement result, but the
more susceptible to interference */
    GPIO_SetDeglitchWindow(DGCLKCTL_ALL_DIV_(0x0));

    /* Set the count reset back to zero when event happened */
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT0);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT1);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT2);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT3);

    /* Enable EVT3 Interrupt */
    ECAP_EnableInt(ECAP, ECAP_INT_CEVT3);

    /* Enable global interrupt of EACP */
    NVIC_EnableIRQ(ECAP_IRQn);

    /* Start ECAP */
    ECAP_RunCounter(ECAP);

    /* Start PWM_Sel */
    PWM_RunCounter(PWM_Sel);

    while (1)
    {

    }
}

void ECAP_IRQHandler(void)
{
    uint32_t cnt;

    TStamp0 = ECAP->CAP0.all;
    TStamp1 = ECAP->CAP1.all;
    TStamp2 = ECAP->CAP2.all;
    TStamp3 = ECAP->CAP3.all;

    cnt = TStamp2 + TStamp3;
    printf("Fre = %dHz\n", CNTTOFREQ(cnt));

    /* Clear CEVT3 */
    ECAP_ClearInt(ECAP, ECAP_INT_CEVT3);
    ECAP_ClearGlobalInt(ECAP);
    ECAP_OneshotReArm(ECAP);
}
```

2.2 APWM 模式

本示例中演示 ECAP 在 APWM 模式下输出 10KHz 的波形，产生的波形如图 2-5。

图 2-5：APWM 模式



- 初始化系统时钟，UART 调试口；
- 调用 ECAP_APWMModeInit()函数设置 APWM 模式；
- 调用 ECAP_APWMSetDuty()函数设置 APWM 的占空比（范围 0~10000）；
- 调用 ECAP_RunCounter()函数使能 PWM 功能，输出波形；

APWM 模式

```
#define ECAP_APWM_GPIO      GPIO_0      /* ECAP_APWM Output gpio */
#define ECAP_APWM_FREQ      10000       /* 10kHz PWM */
#define ECAP_CMP_COUNT      5000        /* 50% duty of APWM waveform */

int main()
{
    FLASH_WALLOW();

    FLASH_SetTiming(200000000);
    /* Disable flash write access after flash operation had done */
    FLASH_WDIS();

    CLOCK_InitWithRCO(CLOCK_HCLK_200MHZ);

    Delay_Init();

    /*
     * Init the UART
     *
     * 1.Set the GPIO34/35 as UART FUNC
     *
     * 2.Enable the UART CLK
```

APWM 模式

```
     *
     * 3.Set the rest para
     */
    GPIO_SetPinChannel(GPIO_34, GPIO34_UART_TXD);
    GPIO_SetPinChannel(GPIO_35, GPIO35_UART_RXD);
    CLOCK_EnableModule(UART_MODULE);
    UART_Init(UART, 38400);

    printf("ECAP APWM test....\n");

    CLOCK_EnableModule(ECAP_MODULE);

    /* ECAP APWM Mode Init */
    ECAP_APWMModeInit(ECAP, ECAP_APWM_GPIO, ECAP_APWM_FREQ);

    /* Set the duty of APWM waveform */
    ECAP_APWMSetDuty(ECAP, ECAP_CMP_COUNT);

    /* Start ECAP */
    ECAP_RunCounter(ECAP);

    while (1)
    {

    }
}
```