SPC1169 ECAP 使用指南

版本 A/0 – 2023 年 4 月

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

表格列表

版本历史

版本	日期	作者	状态	变更
A/0	2023 年 4 月 14 日	Xuqinghe	Released	首次发布。

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

SPC1169 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。

SPC1169 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；

SPC1169 的 ECAP 单元有以下特点：

- 基于 32bit 的计时器，且精度可达 10ns（时钟为 100MHz）；
- 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）；
- 与外部事件同步的 4 级序列器；
- 4 个事件均可支持中断；

2 ECAP 实例

如本文档第一章描述，SPC1169 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。连续模式又可根据是否在每个事件发生时复位计数值。

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。

捕获模式下，可捕获单通道信号也可捕获双通道信号。捕获双通道信号时，CAP0 和 CAP2 作为捕获一个输入信号边沿事件的一对寄存器，CAP1 和 CAP3 作为另一个输入信号边沿事件的一对寄存器，捕获单通道信号时 CAP0、CAP1、CAP2、CAP3 相互之间没有成对的关系。

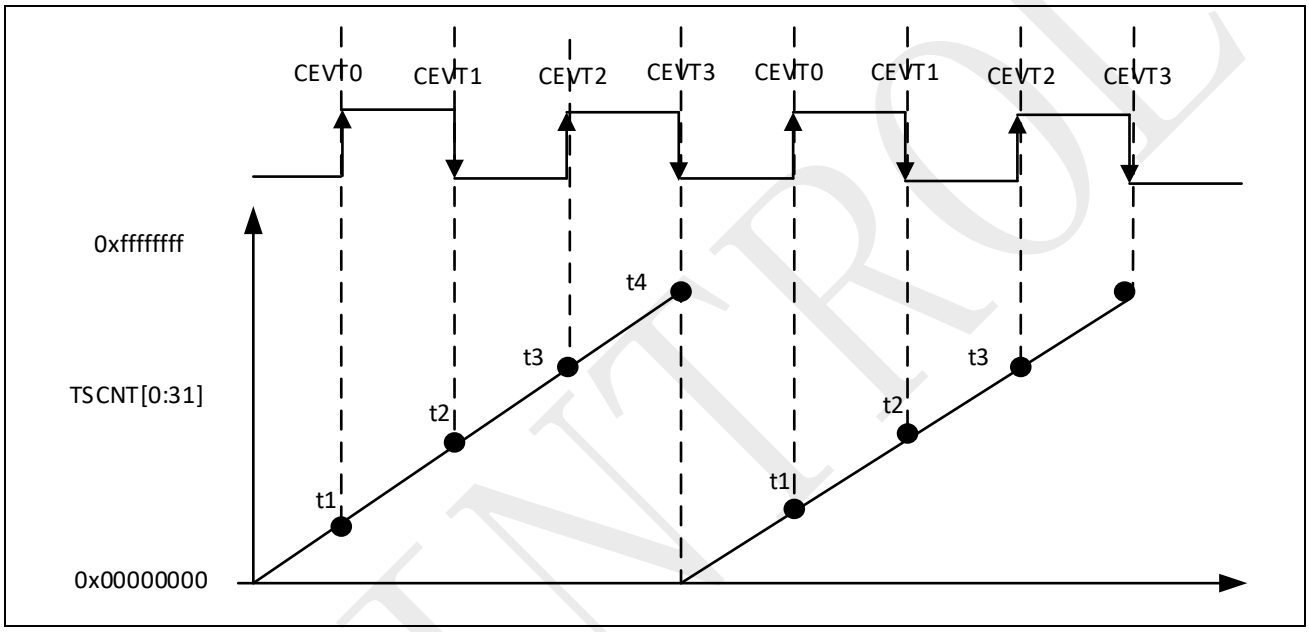
2.1 单通道信号捕获

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：连续捕获（绝对时间戳计数，根据捕获的信息可以推出波形的周期 T= t3 – t1、波形为高占空 = t2 – t1、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 PWM2 输出一路波形，并对 PWM2 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 PWM2 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 GPIO14 引脚，GPIO14 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 GPIO14 为输入功能，将 GPIO14 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 PIN_EnableDeglitch()和设置滤波函数窗口 PIN_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)

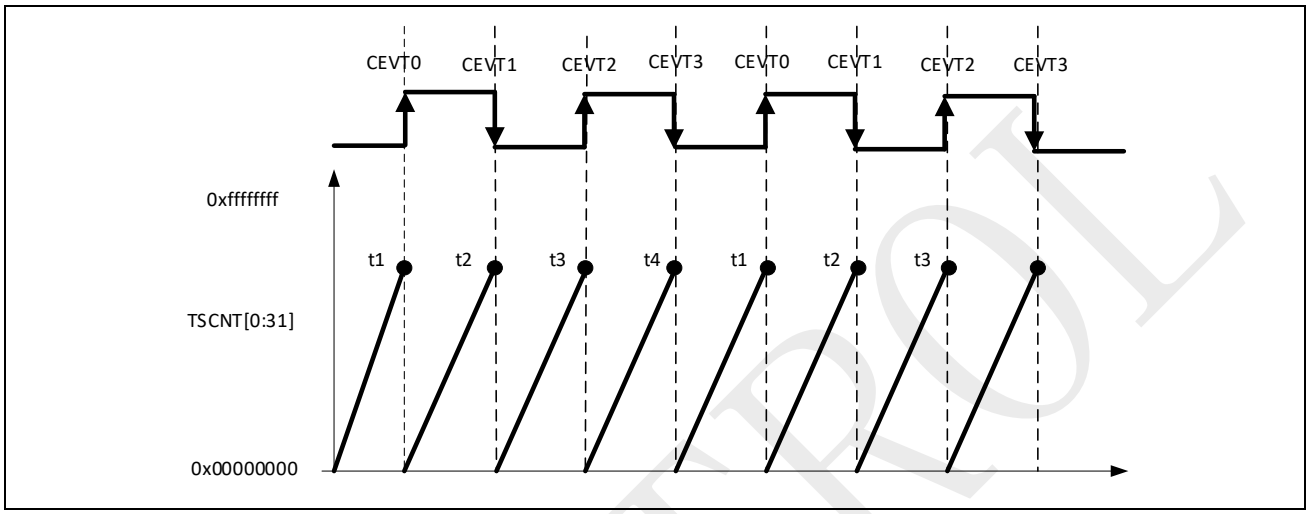
```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


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


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

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Init PWM2, PWM freq will be set as 10kHz in this function */
    PWM_InitSingleChannel(PWM2, PWM_CHA, PWM_FREQ);

    /* Set PWM2A output 50% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM2, u32PWMPeriod / 2);

    /* Start PWM2 */
    PWM_RunCounter(PWM2);

    /* ECAP Init */
    ECAP_CaptureModeInit(ECAP, PIN_GPIO14);

    /* Select PIN_GPIO14 as the channel A output of PWM2 */
    PIN_SetChannel(PIN_GPIO14, PIN_GPIO14_PWM2A);
```

```
    PIN_EnableInputChannel(PIN_GPIO14, PIN_GPIO14_GPIO14);

    /* Enable Pin input deglitch filter */
    PIN_EnableDeglitch(PIN_GPIO14);

    /* The smaller the value, the more accurate the measurement
result, but the more susceptible to interference */
    PIN_SetDeglitchWindow(DGCLKCTL_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 overflow 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;
    TStamp1 = ECAP->CAP1;
    TStamp2 = ECAP->CAP2;
    TStamp3 = ECAP->CAP3;

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

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

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、波形为高占空 = t2、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 PWM2 输出一路波形，并对 PWM2 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 PWM2 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 GPIO14 引脚，GPIO14 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 GPIO14 为输入功能，将 GPIO14 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 PIN_EnableDeglitch()和设置滤波函数窗口 PIN_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)

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


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


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


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Init PWM2, PWM freq will be set as 10kHz in this function */
    PWM_InitSingleChannel(PWM2, PWM_CHA, PWM_FREQ);

    /* Set PWM2A output 50% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM2, u32PWMPeriod / 2);

    /* Start PWM2 */
    PWM_RunCounter(PWM2);

    /* ECAP Init */
    ECAP_CaptureModeInit(ECAP, PIN_GPIO14);

    /* Select PIN_GPIO14 as the channel A output of PWM2 */
    PIN_SetChannel(PIN_GPIO14, PIN_GPIO14_PWM2A);
    PIN_EnableInputChannel(PIN_GPIO14, PIN_GPIO14_GPIO14);


    /* Enable Pin input deglitch filter */
    PIN_EnableDeglitch(PIN_GPIO14);

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

    /* Set the count reset back to zero when event happened */
```

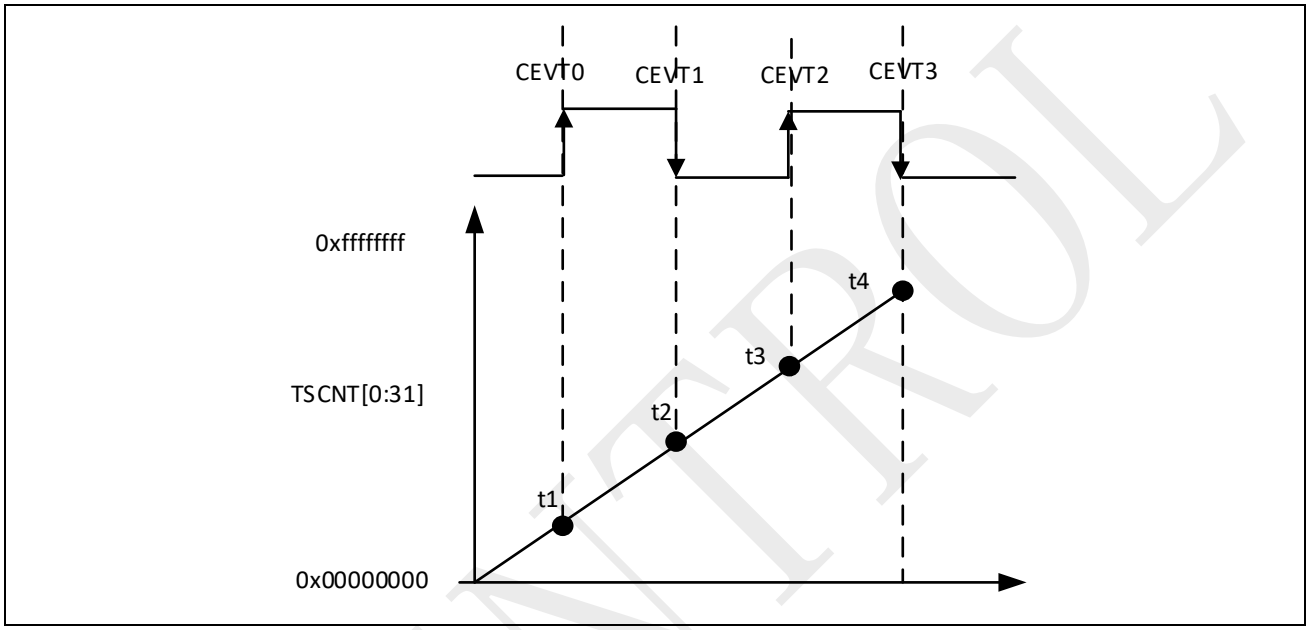
```
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT0);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT1);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT2);
    ECAP_EnableEventResetCounter(ECAP, ECAP_CEVT3);

    /* Enable EVT3 overflow 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;
    TStamp1 = ECAP->CAP1;
    TStamp2 = ECAP->CAP2;
    TStamp3 = ECAP->CAP3;

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

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

2.1.2 一次性模式

2.1.2.1 绝对时间戳计数

本示例中演示对 PWM 波进行一次性捕获，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、波形为高占空 = t2 – t1、波形为低占空 = t3 – t1。

图 2-3：一次性捕获（绝对时间戳计数）



如下示例将捕获 PWM2 输出一路波形，并对 PWM2 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 PWM2 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 GPIO14 引脚，GPIO14 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 为一次性模式，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 GPIO14 为输入功能，将 GPIO14 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 PIN_EnableDeglitch()和设置滤波函数窗口 PIN_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；

一次性模式(绝对时间戳捕获)

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


/* 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 */
#define ECAP_PIN                                                PIN_GPIO14
/* ECAP PIN */
#define ECAP_PIN_GPIO_FUNC                                PIN_GPIO14_GPIO14
/* ECAP PIN act as GPIO function */
#define ECAP_PIN_PWM_FUNC                                 PIN_GPIO14_PWM2A
/* ECAP PIN act as PWM function */
/*usd the count to calculate the period*/
#define                                                   CNTTOFREQ(x)
(CLOCK_GetModuleClock(ECAP_MODULE) / (x))


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

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* Init PWM2, PWM freq will be set as 10kHz in this function */
    PWM_InitSingleChannel(PWM2, PWM_CHA, PWM_FREQ);

    /* Set PWM2A output 50% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM2, u32PWMPeriod / 2);
```

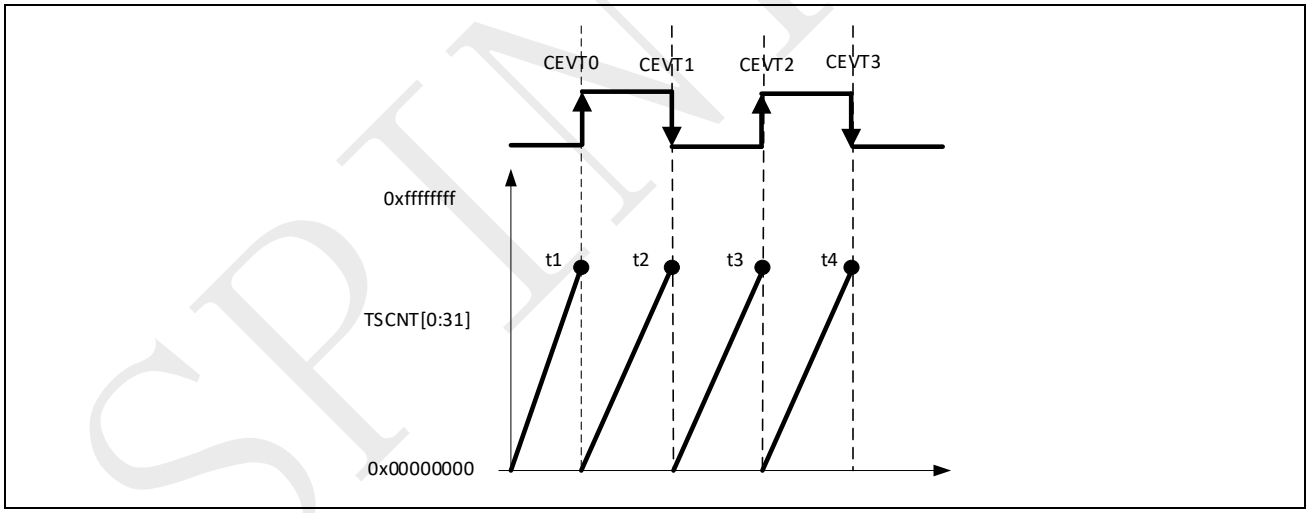
```
    /* Start PWM2 */
    PWM_RunCounter(PWM2);

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

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

    /* Can divide the input signal to observe higher frequency
signals */
    ECAP_SetEventDiv(ECAP, 1);

    /* RE-arm ECAP event count register */
    ECAP_OneshotReArm(ECAP);

    /* 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 overflow Interrupt */
    ECAP_EnableInt(ECAP, ECAP_INT_CEVT3);

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

    /* Select ECAP_PIN as the channel A output of PWM2 */
    PIN_SetChannel(ECAP_PIN, ECAP_PIN_PWM_FUNC);
    PIN_EnableInputChannel(ECAP_PIN, ECAP_PIN_GPIO_FUNC);

    /* Enable Pin input deglitch filter */
    PIN_EnableDeglitch(ECAP_PIN);

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

    /* Start ECAP */
    ECAP_RunCounter(ECAP);

    while (1)
    {

    }
}


void ECAP_IRQHandler(void)
{
    uint32_t cnt;
```

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

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

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

2.1.2.2 相对时间戳计数

本示例中演示对 PWM 波进行一次性捕获，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 2-4：一次性模式（相对时间戳计数）



如下示例将捕获 PWM2 输出一路波形，并对 PWM2 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 PWM2 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 GPIO14 引脚，GPIO14 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 为一次性模式，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 GPIO14 为输入功能，将 GPIO14 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 PIN_EnableDeglitch()和设置滤波函数窗口 PIN_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；

一次性模式(相对时间戳计数)

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


/* 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 */
#define ECAP_PIN                                    PIN_GPIO14
/* ECAP PIN */
#define ECAP_PIN_GPIO_FUNC                           PIN_GPIO14_GPIO14
/* ECAP PIN act as GPIO function */
#define ECAP_PIN_PWM_FUNC                            PIN_GPIO14_PWM2A
/* ECAP PIN act as PWM function */
/*usd the count to calculate the period*/
#define CNTTOFREQ(x)
(CLOCK_GetModuleClock(ECAP_MODULE) / (x))


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


int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    /* Init PWM2, PWM freq will be set as 10kHz in this function */
```

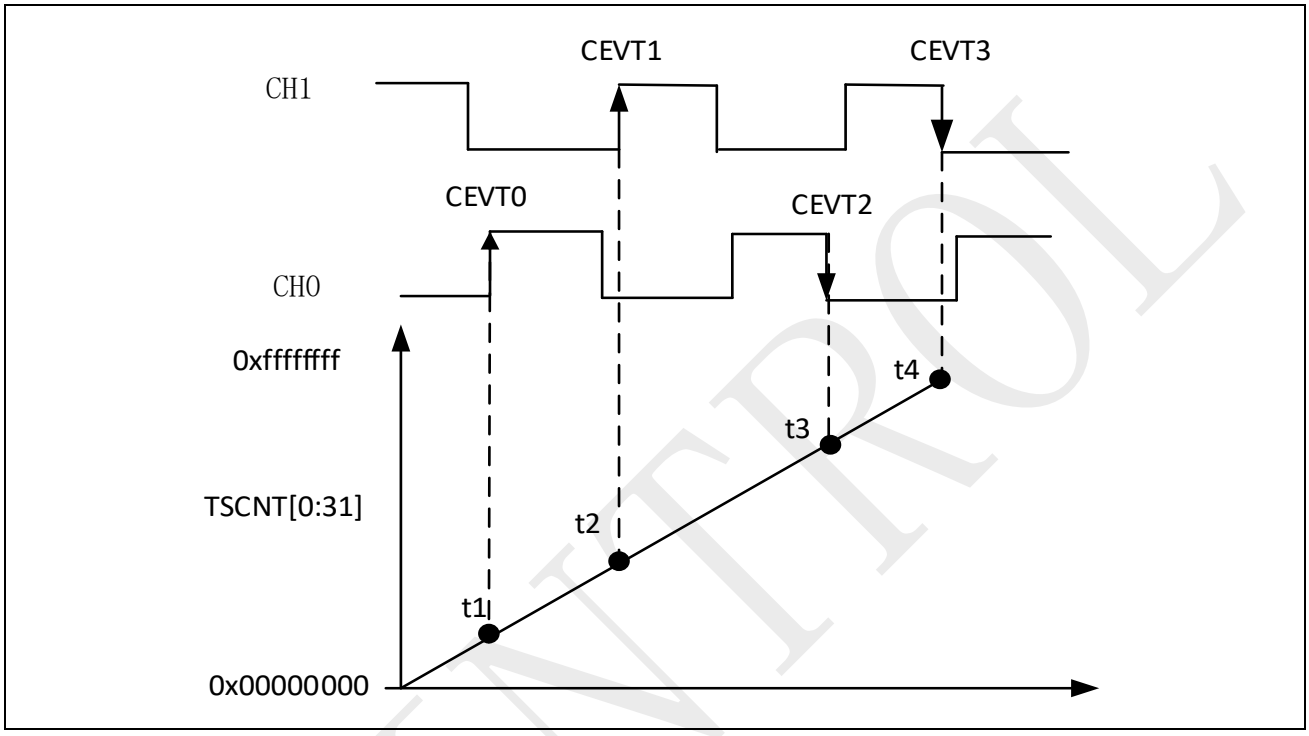
```
    PWM_InitSingleChannel(PWM2, PWM_CHA, PWM_FREQ);

    /* Set PWM2A output 50% duty waveform */
    u32PWMPeriod = PWMPeriod(PWM_FREQ);
    PWM_SetCMPA(PWM2, u32PWMPeriod / 2);

    /* Start PWM2 */
    PWM_RunCounter(PWM2);

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

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

    /* Can divide the input signal to observe higher frequency signals */
    ECAP_SetEventDiv(ECAP, 1);

    /* RE-arm ECAP event count register */
    ECAP_OneshotReArm(ECAP);

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

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

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

    /* Select ECAP_PIN as the channel A output of PWM2 */
    PIN_SetChannel(ECAP_PIN, ECAP_PIN_PWM_FUNC);
    PIN_EnableInputChannel(ECAP_PIN, ECAP_PIN_GPIO_FUNC);

    /* Enable Pin input deglitch filter */
    PIN_EnableDeglitch(ECAP_PIN);

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

    /* Start ECAP */
    ECAP_RunCounter(ECAP);

    while (1)
    {

    }
}


void ECAP_IRQHandler(void)
{
    uint32_t cnt;

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

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

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

2.2 双通道捕获

本示例中演示对 PWM 两个通道波进行连续捕获，以相对时间戳方式计数。对 PWM 信号的捕获信息如图 2-5，根据捕获的信息可以推出波形的周期 T= (t1 – t0) + (t4 – t3)、相位差 = t2 – t1。

图 2-5：双通道捕获（相对时间戳计数）



- 初始化系统时钟，UART 调试口；
- 初始化 PWM1 并生成两路波形；
- 调用 ECAP_CaptureModeDualPinInit()函数初始化 ECAP 为双通道模式，并将 PWM 产生的 GPIO12 和 GPIO13 两路信号送进 ECAP 进行捕获。ECAP_CaptureModeDualPinInit 函数默认设置捕获事件为 CAP0 上升沿捕获、CAP1 上升沿捕获、CAP2 下降沿捕获、CAP3 下降沿捕获；
- 调用 PIN_EnableInputChannel()函数设置 GPIO12 和 GPIO13 为输入功能，将信号送入 ECAP；
- 如果输入信号存在干扰，可以调用使能滤波函数 PIN_EnableDeglitch()和设置滤波函数窗口 PIN_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和占空比；

双通道捕获

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


/* 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 */
#define PWM_DeadTimeNs                     400   /* 400 ns */

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

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

void ECAP_CaptureModeDualPinInit(ECAP_REGS *ECAPx, PIN_NameEnum
ePinName1,  PIN_NameEnum ePinName2)
{
    /* Open ECAP clock */
    CLOCK_EnableModule(ECAP_MODULE);

    /* ECAP Init */
    ECAP->CAPSRCCTL |= CAPSRCCTL_POL1_(GPIO_LEVEL_HIGH);
    ECAP->CAPSRCCTL |= CAPSRCCTL_IOSEL1_(ePinName2);

    /* Configure IO Pin for capture */
    ECAP_SetInput(ECAPx, ePinName1, GPIO_LEVEL_HIGH);

    ECAPx->CAPCTL =   CAPCTL_APWMMODE_CAPTURE_MODE       /*
Configure ECAP as Capture mode           */
                    | CAPCTL_CAPLDEN_ENABLE             /* Enable
Cap Register loading on a capture event */
                    | CAPCTL_STOPWRAP_ON_CAPTURE_EVENT3   /*
Stop/Wrap after Capture Event 3          */
                    | CAPCTL_ONESHOT_DISABLE            /* Operate
in Continous mode                */
                    | CAPCTL_CAP0POL_TRIG_ON_RISING_EDGE  /* Event 0
occurs on pulse rising edge       */
                    | CAPCTL_CAP1POL_TRIG_ON_RISING_EDGE  /* Event 1
occurs on pulse rising edge      */
                    | CAPCTL_CAP2POL_TRIG_ON_FALLING_EDGE /* Event
2 occurs on pulse falling edge       */
```

```
                    | CAPCTL_CAP3POL_TRIG_ON_FALLING_EDGE /* Event
3 occurs on pulse falling edge      */
                    | CAPCTL_CNTRST0_ENABLE             /* Reset
capture counter when Event 0 occurs  */
                    | CAPCTL_CNTRST1_ENABLE             /* Reset
capture counter when Event 1 occurs  */
                    | CAPCTL_CNTRST2_ENABLE             /* Reset
capture counter when Event 2 occurs  */
                    | CAPCTL_CNTRST3_ENABLE             /* Reset
capture counter when Event 3 occurs  */
                    | CAPCTL_TSCNTRUN_STOP_COUNTER;      /* Disable
ECAP counter                        */

    /* Enable Dual pin capture mode */
    ECAP->CAPCTL |= CAPCTL_DUALPIN_ENABLE;

    /* Disable Capture event interrupt */
    ECAPx->CAPIE = 0U;
}

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);

    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    /* Init PWM1, PWM freq will be set as 10kHz in this function
*/
    PWM_InitComplementaryPairChannel(PWM1, PWM_FREQ,
PWM_DeadTimeNs);

    /* Select PIN as the channel A/B output of PWM1 */
    PIN_SetChannel(PIN_GPIO12, PIN_GPIO12_PWM1A);
    PIN_SetChannel(PIN_GPIO13, PIN_GPIO13_PWM1B);

    /* Start PWM1 */
    PWM_RunCounter(PWM1);

    /* ECAP Dual Pin Init */
    ECAP_CaptureModeDualPinInit(ECAP, PIN_GPIO12, PIN_GPIO13);

    PIN_EnableInputChannel(PIN_GPIO12, PIN_GPIO12_GPIO12);
    PIN_EnableInputChannel(PIN_GPIO13, PIN_GPIO13_GPIO13);
```

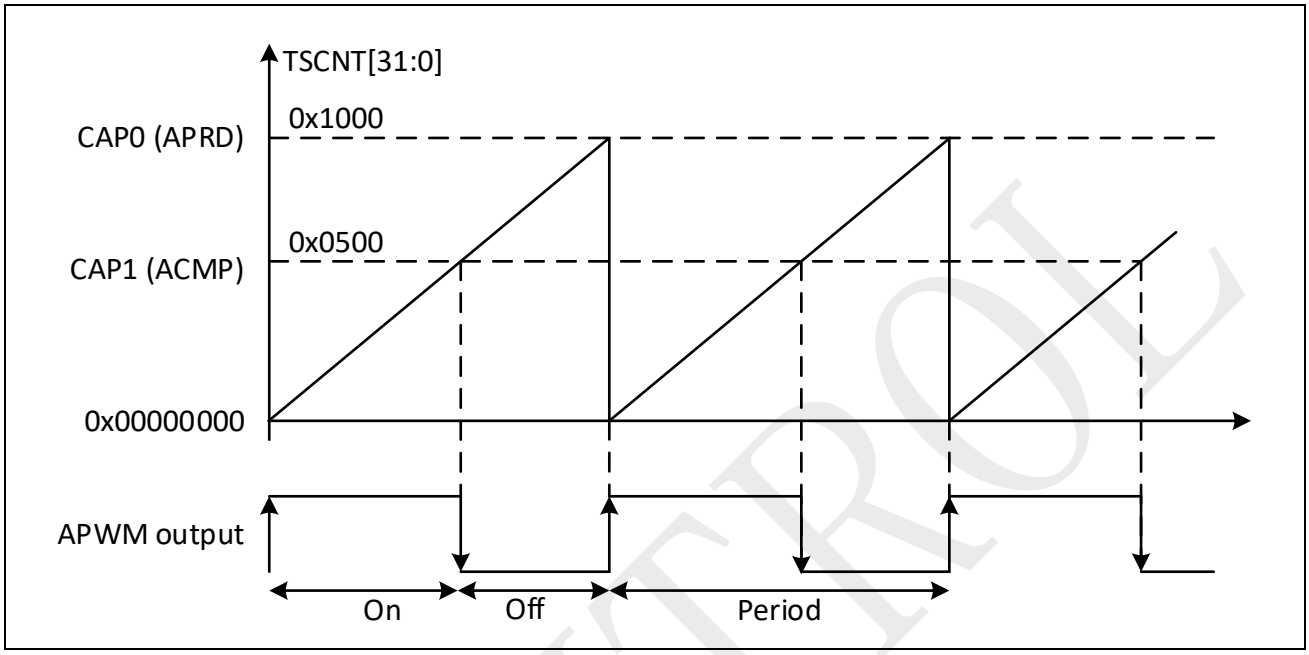
```
    /* The smaller the value, the more accurate the measurement
result, but the more susceptible to interference */
    PIN_SetDeglitchWindow(DGCLKCTL_DIV_(0x0));

    /* 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);

    /* Can divide the input signal to observe higher frequency
signals */
    ECAP_SetEventDiv(ECAP, 1);

    /* Enable EVT3 overflow Interrupt */
    ECAP_DisableInt(ECAP, ECAP_INT_CEVT0);
    ECAP_DisableInt(ECAP, ECAP_INT_CEVT1);
    ECAP_DisableInt(ECAP, ECAP_INT_CEVT2);
    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 cnt1, cnt2;

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

    printf("ECAP->CAP0 = %d\n", TStamp0);
    printf("ECAP->CAP1 = %d\n", TStamp1);
    printf("ECAP->CAP2 = %d\n", TStamp2);
    printf("ECAP->CAP3 = %d\n", TStamp3);

    cnt1 = TStamp1 - TStamp0;
    cnt2 = TStamp3 - TStamp2;

    printf("Fre = %dHz\n", CNTTOFREQ(cnt1 + cnt2));
    printf("Duty = %d%%\n",  cnt1 * 100 / ((cnt1 + cnt2)));
```

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

2.3 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 2-6。

图 2-6：APWM 模式



- 初始化系统时钟，UART 调试口；
- 调用 ECAP_SetMode()函数设置 APWM 模式；
- 调用 ECAP_SetPRD()函数设置 APWM 的周期 count 值；
- 调用 ECAP_APWMSetDuty()函数设置 APWM 的占空比（范围 0~10000）；
- 调用 PIN_SetChannel()函数设置 GPIO1 作为 APWM 功能；

APWM 模式

```
#include <stdio.h>

#if defined(SPD1179)
    #include "spd1179.h"
#else
    #include "spc1169.h"
#endif


#define PWM_FREQ                                    10000
/* 10000Hz PWM */

int main(void)
{
    CLOCK_InitWithRCO(CLOCK_CPU_100MHZ);
```

```
    Delay_Init();

    /*
     * Init the UART
     */
    PIN_SetChannel(PIN_GPIO10, PIN_GPIO10_UART0_TXD);
    PIN_SetChannel(PIN_GPIO11, PIN_GPIO11_UART0_RXD);
    UART_Init(UART0, 38400);

    printf("Enter the test\n");

    CLOCK_EnableModule(ECAP_MODULE);

    /* Config ECAP operating mode */
    ECAP_SetMode(ECAP, ECAP_APWM_MODE);

    /* Calculate Period Value based on system clock and ECAP clock
*/
    ECAP_SetPRD(ECAP,      CLOCK_GetModuleClock(ECAP_MODULE)      /
PWM_FREQ);

    printf("ECAP->CAP0 is %x", ECAP->CAP0);

    /* Set polarity */
    ECAP_SetAPWMOutputPolarity(ECAP, GPIO_LEVEL_HIGH);

    /* Set duty */
    ECAP_APWMSetDuty(ECAP, 5000);

    /* Set Output */
    PIN_SetChannel(PIN_GPIO1, PIN_GPIO1_ECAP_APWMO);

    /* Run Counter */
    ECAP_RunCounter(ECAP);

    while (1)
    {

    }
}
```