第 7 课 F2812 的 EV 模块

作者：顾卫钢

由于 2812 主要应用于工业控制场合，例如电机控制、变频器，逆变器等等，所以 2812 的事件管理器模块（EV）可谓是重点部分了，由于 EV 的内容非常丰富，我们在讲义中不能面面俱到，所以希望大家在学习的时候能将手头的书本和我们的讲义结合起来学习，应该效果会更好。我们在今天的课程中将为大家重点介绍 EV 模块中的定时器及其 PWM 电路，希望通过这节课的学习，大家能够熟练使用 EV 来产生自己所需要的 PWM 波形。

1.事件管理器的功能

2812 具有两个事件管理器模块 EVA 和 EVB，这两个 EV 模块就像孪生兄弟一样，具有相同的功能，例如它们的定时器、比较单元、捕捉单元的功能都是完全一样的，只是各个单元的名称因为 EVA 和 EVB 有所区别而已，我们在下面的分析中主要以 EVA 为例。

	EVA		EVB	
事件管理器模块	模块	信号引脚	模块	信号引脚
通用定时器	定时器 1	T1PWM_T1CMP	定时器 3	T3PWM_T3CMP
	定时器 2	T2PWM_T2CMP	定时器 4	T4PWM_T4CMP
比较单元	比较器 1	PWM1/2	比较器 4	PWM7/8
	比较器 2	PWM3/4	比较器 5	PWM9/10
	比较器 3	PWM5/6	比较器 6	PWM11/12
捕获单元	捕获器 1	CAP1_QEP1	捕获器 4	CAP4_QEP3
	捕获器 2	CAP2_QEP2	捕获器 5	CAP5_QEP4
	捕获器 3	CAP3_QEPI1	捕获器 6	CAP6_QEPI2
QEP 电路	QEP1	CAP1_QEP1	QEP3	CAP4_QEP3
	QEP2	CAP2_QEP2	QEP4	CAP5_QEP4
	QEP11	CAP3_QEPI1	QEP12	CAP6_QEPI2
外部定时器输入	计数方向	TDIRA	计数方向	TDIRB
	外部时钟	TCLKINA	外部时钟	TCLKINB
External compare-output trip inputs		C1TRIP		C4TRIP
		C2TRIP		C5TRIP
		C3TRIP		C6TRIP
External timer-compare trip inputs		T1CTRIP_PDPINTA		T3TRIP_PDPINTB
		T2CTRIP/EVASOC		T4CTRIP/EVBSOC
External trip inputs		T1CTRIP_PDPINTA		T3TRIP_PDPINTB
外部 AD 转换启动信号		T2CTRIP/EVASOC		T4CTRIP/EVBSOC

简单来讲的话，每个 EV 模块都具有 2 个通用定时器、3 个比较单元、3 个捕获单元以及 1 个正交编码电路，我们重点来讲定时器和比较单元部分的内容。EVA 和 EVB 的资源详见上面的表格所示，表格中蓝色的字表示该信号是低电平有效。

通用定时器就像秒表一样，是可以用来计时的，而且每个定时器还能产生 1 路独立的 PWM 波形；比较单元主要功能就是用来生成 PWM 波形的，EVA 具有 3 个比较单元，每个单元可以生成一对（两路）互补的 PWM 波形，生成的 6 路 PWM 波形正好可以驱动一个三相桥电路。捕获单元的功能是捕捉外部输入脉冲波形的上升沿或者下降沿，可以统计脉冲的间隔，也可以统计脉冲的个数。正交编码电路的话应该用的比较少，它可以对输入的正交脉冲进行编码和计数，它和光电编码器相连可以获得旋转机械部件的位置和速率等信息。

细心的同学可能会发现，为什么“External compare-output trip inputs”、“External timer-compare trip inputs”、“External trip inputs”用的是英文，其他都翻译成中文了呢？原因是书上翻译的太差了，特别是我们推荐的教材《TMS320C28X 系列 DSP 的 CPU 与外设》中将其分别翻译成了“外部比较-输出行程输入”“外部定时器_比较行程输入”“外部行程输入”，让人看了有些丈二和尚摸不着头脑的感觉，不知道讲的是什么，这几个信号究竟是干嘛用的更是不清楚了，因此，在这里还是向大家介绍原汁原味的英文名称。下面我们讲讲这几个信号的作用：

（1） External compare-output trip inputs—我们可以理解为切断比较输出的外部控制输入，以 C1TRIP 为例，当比较单元 1 工作时，其两个引脚 PWM1 和 PWM2 正在不断的输出 PWM 波形，这时候，如果 C1TRIP 信号变为低电平，则此时 PWM1 和 PWM2 引脚被置成高阻态，不会再有 PWM 波形输出，也就是在这个引脚上输入低电平，则比较输出就会被切断。

（2） External timer-compare trip inputs—我们可以理解为切断定时器比较输出的外部控制输入，以 T1PWM_1CMP 为例，当定时器 1 的比较功能在运行，并且 T1PWM 引脚输出 PWM 波形的时候，这时候如果 T1CTRIP 引脚信号变为低电平，则该引脚状态被置成高电平，也不会再有 PWM 波形输出。

（3） External trip inputs 的 PDPINTx（x=A 或者 B）其实是个功率驱动保护，它为系统的安全提供了保护，例如如果当电路中出现电压、电流或者温度急剧上升的时候，如果 PDPINTx 的中断没有被屏蔽，当 PDPINTx 的引脚变为低电平时，2812 所有的 PWM 输出引脚都会变为高阻态，从而阻止了电路进一步损坏，达到保护系统的目的。当然 PDPINTx 在电路设计时就要考虑到给它配一个监视电路状态的信号。

在上面的描述中，一直提到了高阻态，第一次接触的朋友可能不怎么清楚这个状态，我们在这补充一下：

什么是高阻态？
高阻态这是一个数字电路里常见的术语，指的是电路的一种输出状态，既不是高电平也不是低电平，如果高阻态再输入下一级电路的话，对下级电路无任何影响，和没接一样，如果用万用表测的话有可能是高电平也有可能是低电平，随它后面接的东西来确定。 在 2812 中，悬空的引脚一般就是处在高阻态的，所以如果你的 AD 引脚悬空，那么采样到的 AD 值是一个不确定的数据。

2.通用定时器

下面我们将和大家一起探讨通用定时器相关的内容，主要围绕 EVA 下面的定时器 1，也就是 T1 进行讨论。

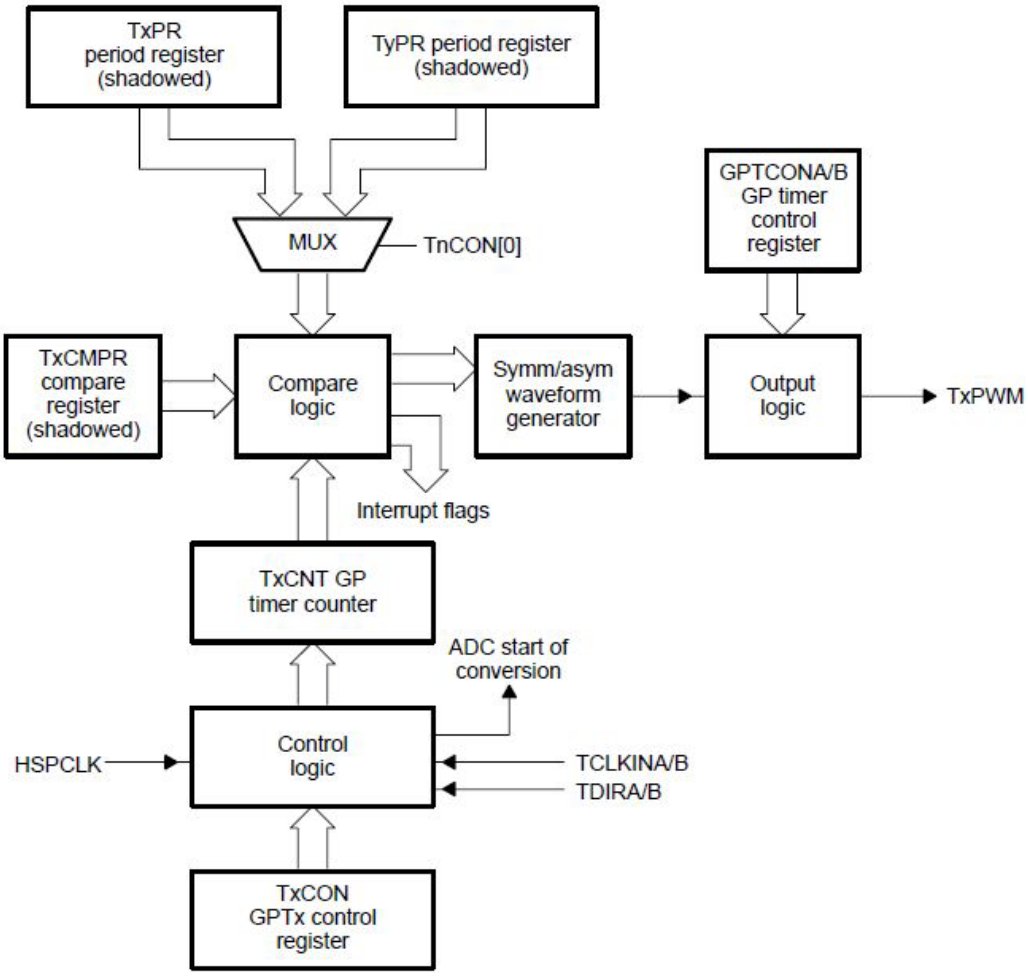


图 1 通用定时器的模块图

上面的图是定时器的模块图，我们通过这个图可以得到下面的一些信息。

和 T1 相关的常用寄存器
1. T1 周期寄存器 T1PR 16 位
2. T1 比较寄存器 T1CMPR 16 位
3. T1 计数寄存器 T1CNT 16 位
4. T1 控制寄存器 T1CON 16 位
5. 全局定时器控制寄存器 A GPTCONA 16 位

T1 的常见输入信号
1. 来自于 CPU 的内部时钟
2. 外部时钟输入 TCLKINA，最大频率为器件自身时钟的 1/4，也就是 1/4*150M
3. TDIRA/B，用于定时器的增/减计数模式

4. 复位信号 RESET

T1 的输出信号
1. 定时器的比较输出 T1PWM_T1CMP
2. 送给 ADC 模块的 AD 转换启动信号
3. 下溢、上溢、比较匹配和周期匹配信号
4. 计数方向指示

T1PR 是定时器 T1 的周期寄存器，用于存放为 T1 设置的周期值。T1CMPR 是定时器 T1 的比较寄存器，用于存放为 T1 设置的比较值。而 T1CNT 为 T1 的计数器寄存器，其内容是随着时钟脉冲不断增加或者减少的，每 1 个 HSPCLK 的脉冲，T1CNT 的值增加 1 或者减少 1。T1PR 和 T1CMPR 在一般情况下是在初始化的时候进行赋值，然后就成为了一个参考标准，CPU 会实时的将 T1CNT 的值和这两个标准进行比较，当 T1CNT 的值和 T1PR 相等时，T1CNT 就会复位成 0 重新开始计数或者逐渐减少直至 0，完成 1 个周期的计数，然后再从 0 开始计数至 T1PR 里面的数值，这样循环下去。当 T1CNT 的值和 T1CMPR 的值相等时，就会产生一些比较事件，例如 PWM 波形就是依靠这个原理来实现的，后面我们会详细介绍。讲了这么多，主要就是让大家对这三个寄存器的功能和特点有所了解。

我们从图中还可以看到 T1PR 和 T1CMPR 都带有 shadowed 这个英文词汇，直接翻译的话就是带有阴影的，其实我们可以将其理解为带有缓冲的意思，以前看到有朋友问这个带有阴影的寄存器是什么意思？为什么要带有阴影？我们以下面的图来说明这个问题。

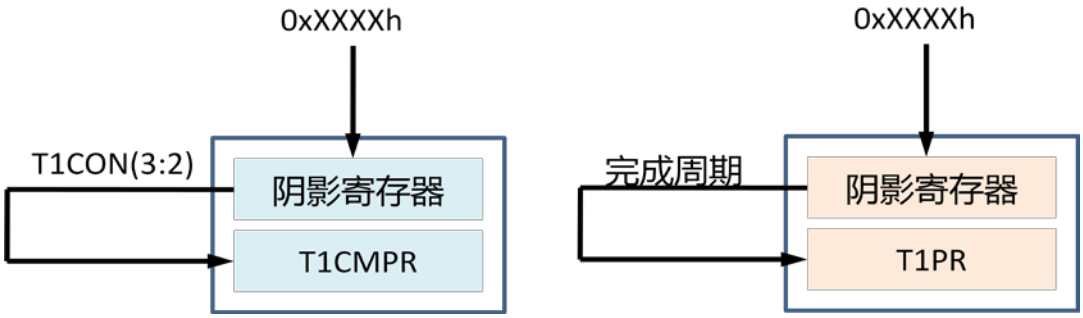


图 2 阴影寄存器的作用

您在学习上面的内容时，可能会考虑到这个问题，在程序执行的过程当中，我们可以改变 T1CMPR 或者 T1PR 的值吗？答案肯定是可以的,我们可以在一个周期的任何时刻向 T1CMPR 或者 T1PR 写入新的数值，其功劳就要归功于这个阴影寄存器。从图 2 所示，假设我们要向 T1CMPR 写入新的数值 0xXXXXh，首先将这个数值写入 T1CMPR 的阴影寄存器，当 T1CON 中第 3 位 TCLD1 和第 2 位 TCLD0 所指定的特定事件发生时，阴影寄存器的数据就会被写入 T1CMPR 的工作寄存器。定时器 1 比较寄存器 T1CMPR 的重载条件如下面的表格所示。如果 TCLD1 和 TCLD0 设置为 10 的话，新的数据就会立即被写入 T1CMPR，从而改变 T1CMPR 的值。

定时器比较寄存器重载条件		
TCLD1	TCLD0	
0	0	当计数器 T1CNT 值为 0

0	1	当计数器 T1CNT 值为 0 或者等于周期寄存器
1	0	立即载入
1	1	保留

如果需要向 T1PR 写入新的数据 0xXXXXh，数据也会被立即写入阴影寄存器，只有当 T1CNT 的完成这个周期的计数，值为 0 的时候，阴影寄存器中的内容才会被载入到工作寄存器中，从而改变 T1PR 的值。大家在实际应用定时器的时候很有可能会需要在程序执行过程中实时的改变这两个寄存器的数值，例如改变 PWM 波形的频率或者脉宽，所以其重载的条件是需要大家关注的。

上面的内容可以归纳为定时器的寄存器重载条件，下面来介绍定时器的计数方式。T1 的计数模式由 T1CON 的第 12 位和第 11 位决定，如下表所示：

T1 计数模式选择		
TMODE1	TMODE0	
0	0	停止/保持
0	1	连续增/减模式
1	0	连续增模式
1	1	定向增/减计数模式（directional up/down count mode）

当 TMODE 值为 0 的时候，就是停止/保持模式，就是定时器计数器 T1CNT 停止计数，保持现有的数值。

当 TMODE 值为 1 的时候，是连续增/减模式，如图 3 所示，T1PR=2，T1CNT 从 0 开始计数至 2，然后再从 2 逐渐减少至 0，周而复始。这时候实际的计数周期为 2*T1PR。

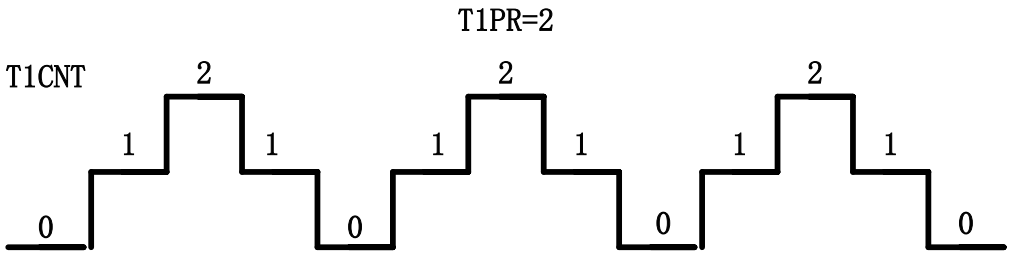


图 3 连续增/减模式

当 TMODE 值为 2 时，就是连续增模式。如图 4 所示，T1PR=2，T1CNT 从 0 开始计数至 2，等于周期寄存器值的值时，直接降为 0，然后再从 0 开始计数至 2，周而复始。这时候实际的计数周期为 T1PR+1。

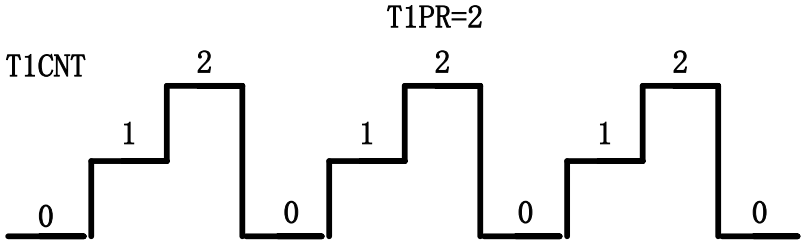


图 4 连续增模式

当 TMODE 值为 3 时，为定向的增或者减计数模式，这时候 T1CNT 进行增计数或者是减计数，得取决于引脚 TDIRA 的电平。如果 TDIRA 为高电平，则 T1CNT 进行增计数；如果 TDIRA 为低电平，则 T1CNT 进行减计数。如图 5 所示，如果是在计数过程中 TDIRA 电平发生了变化，那么必须在完成当前计数周期后的下一个 CPU 时钟周期时，计数方向发生改变。

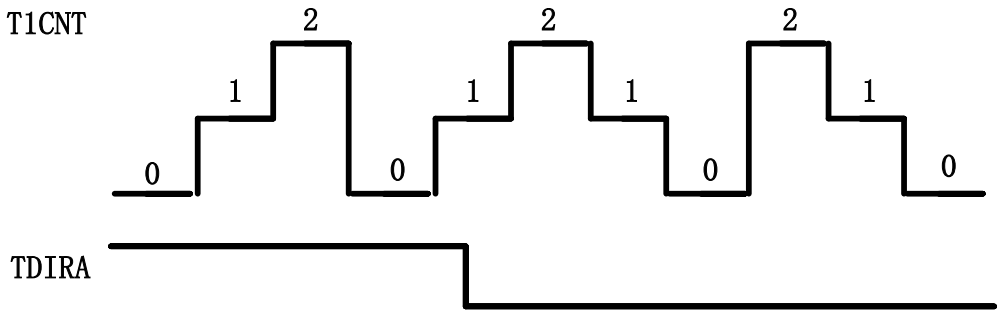


图 5 TDIRA 电平与计数方向的关系

接下来，让我们了解一下与定时器相关的中断，还是以 T1 为例 和 T1 相关的中断有上溢中断 T1OFINT、下溢中断 T1UFINT、比较中断 T1CINT、周期中断 T1PINT。什么情况或者说满足什么条件时会发生上述的这些中断呢？我们一一来说明：

（1）当 T1CNT 的值为 0xFFFFh 的时候，发生定时器 T1 的上溢中断。当上溢事件发生后，再过 1 个 CPU 时钟周期，则上溢中断的标志位被置位。

（2）当 T1CNT 的值为 0x0000h 的时候，发生定时器 T1 的下溢中断。当下溢事件发生后，再过 1 个 CPU 时钟周期，则下溢中断的标志位被置位。

（3）当 T1CNT 的值和 T1 比较寄存器 T1CMPR 的值相等时，发生定时器 T1 的比较中断。当发生比较匹配时，再过 1 个 CPU 时钟周期，则比较中断的标志位被置位。

（4）当 T1CNT 的值和 T1 周期寄存器 T1PR 的值相等时，发生定时器 T1 的周期中断。当发生周期事件时，再过 1 个 CPU 时钟周期，则周期中断的标志位被置位。

根据我们前面所学的，当某个中断的标志位被置位，如果该中断已经使能，则会像 PIE 模块发送中断申请。要记住的是，退出中断的时候，一定要手动清除外设中断标志位。在 EV 中，和上述中断相关的寄存器是 EVAIFRA、EVAIFRB、EVAIMRA、EVAIMRB，大家可自行查阅这些寄存器的相关说明。

上述的一些事件除了能够产生中断以外，还能够在事件发生的时候，产生一个 ADSOC 信号，就是启动 AD 转换的信号，这样可以周期性的去启动 AD 转换。至于具体的是 T1 的哪一个事件发生时产生 AD 转换信号，要看寄存器 GPTCONA 的第 8 和第 7 位，如下表所示。这个功能的优点就在于允许在 CPU 不干涉的情况下使通用定时器的事件和 ADC 启动转换同步进行。

GPTCONA 中 T1 启动 AD 转换的信号的相关位（T1TOADC）		
Bit8	bit7	
0	0	不启动 ADC
0	1	下溢中断启动 ADC
1	0	周期中断启动 ADC
1	1	比较中断启动 ADC

在结束定时器部分的内容前，我们再来讲一下定时器同步的问题。在 EVA 中，通过相关定时器的设置，T2 可以使用 T1 的周期寄存器而忽略自身的周期寄存器，也可以使用 T1 的使能位来启动 T2 计数，这样的功能保证了 T1 和 T2 能够实现同步计数。具体的步骤如下：

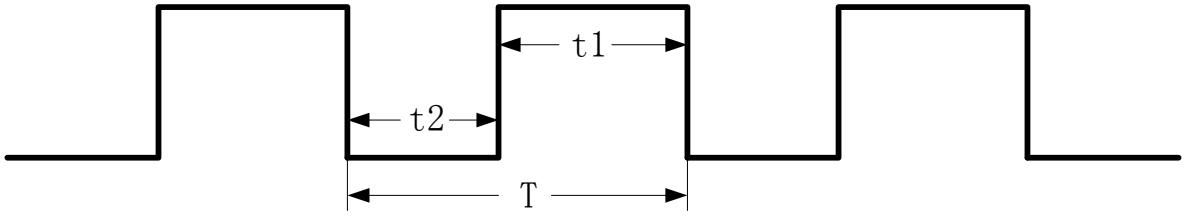
1．将 T2CON 的 T2SWT1 置 1，实现由 T1CON 的 TENABLE 位来启动通用定时器 2 的计数，这样，两个计数器（T1、T2）就能同时启动计数。

2．对 T1CNT 和 T2CNT 进行不同值的初始化。

3．将 T2CON 的 SELT1PR 置 1，指定定时器 2 将定时器 1 的周期寄存器作为自己的周期寄存器。

3.PWM

脉宽调制，简称 PWM(Pulse Width Modulation)是利用微处理器的数字输出来对模拟电路进行控制的一种非常有效的技术，广泛应用在从测量、通信到功率控制与变换的许多领域中，简单的描述就是一些如下图所示的矩形脉冲波形，PWM 波形最重要的三个参数是周期、频率和占空比。



PWM周期：T=t1+t2

PWM频率：F=1/T

PWM占空比：D=t1/(t1+t2)=t1/T

图 6 PWM 波形及其参数

EV 的比较机制能够产生多路 PWM 功能。EVA 的两个通用定时器能够产生 2 路独立的 PWM 波形—T1PWM 和 T2PWM，三个比较单元每一个都能产生一对互补的 PWM 波形，比较单元 1 产生 PWM1 和 PWM2，比较单元 2 产生 PWM3 和 PWM4，比较单元 3 产生 PWM5 和 PWM6。这样，EVA 一共能产生 8 路 PWM 波形。EVB 和 EVA 一样，同样能够产生 8 路 PWM 波形。下面我们将具体的来讲述 EV 产生 PWM 的机制。

首先来介绍通用定时器产生的 PWM 波。我们已经知道 T1 和 T2 分别能够产生 1 路独立的 PWM，我们以 T1 为例进行介绍。当 T1 计数寄存器 T1CNT 的值和 T1CMPR 的值相等时，就会发生比较匹配事件，这时如果 PWM 的功能使能，则 T1PWM 引脚便可以输出 PWM 波形。T1 能够产生两种类型的 PWM，一种是不对称的 PWM 波形，一种是对称的 PWM 波形，究竟产生哪种类型的 PWM 波形取决于 T1CNT 的计数方式。

（1）当 T1CNT 的计数方式为连续增计数时，T1PWM 引脚输出不对称的 PWM 波形。

当定时器 T1 的控制寄存器 T1CON 的 TMODE1 和 TMODE0 为 10 时，定时器 T1 工作于连续增模式。当 T1CNT 的值计数到和 T1CMPR 的值相等时，发生比较匹配事件。如果 T1CON 的第 1 位定时器比较使能为 TECMPR 为 1，即定时器比较操作被使能，且 GPTCONA 的第 6 位比较输出使能位 TCMPOE 为 1，同时 GPTCONA 下的 T1PIN 引脚输出极性为高电平或者低电平的话，T1PWM 就会输出不对称的 PWM 波形，如图 7 所示。

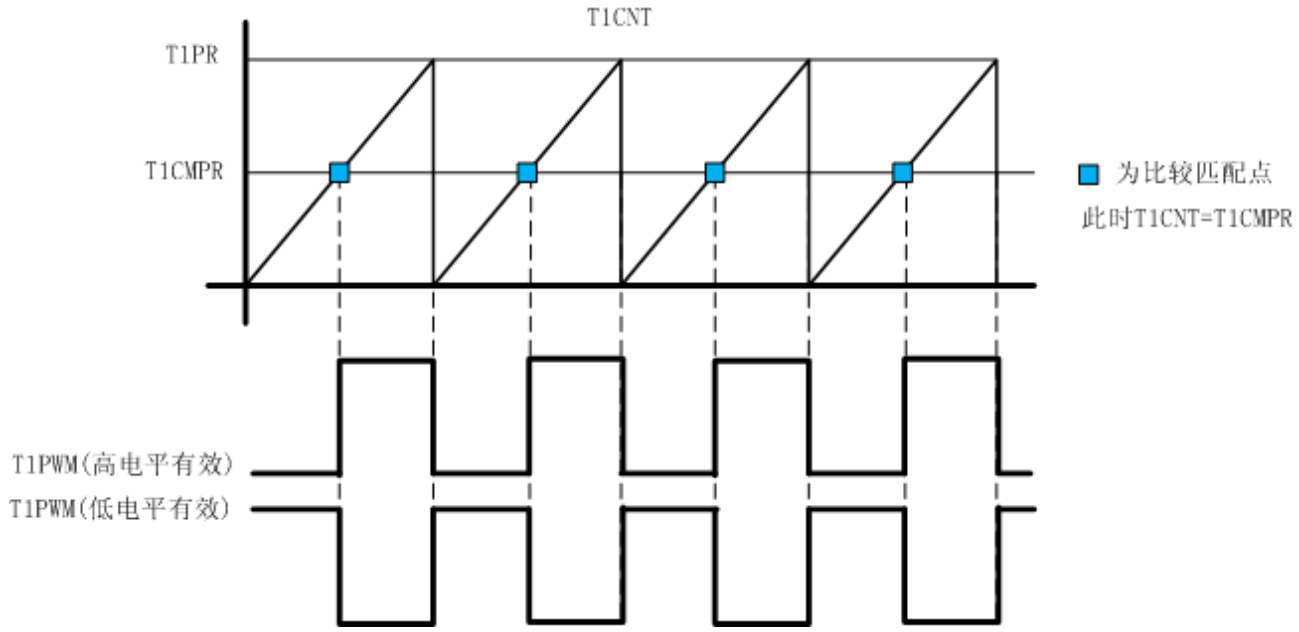


图 7 T1 产生非对称的 PWM 波形

结合前面所学的知识，我们来重点研究一下图 7 中的 PWM 波形的各个参数。我们知道，当 T1 工作于连续增模式时，定时器的周期 T=（T1PR+1）*tc，其中 tc 为 T1CNT 每计数 1 次所需的时间。问题就在于 tc 怎么来确定呢？这就得看提供给 T1 的时钟了。我们从外部晶振开始分析起，看看供给 T1 的时钟是怎么得到的，明白了这个的话 tc 就不难的出来了。

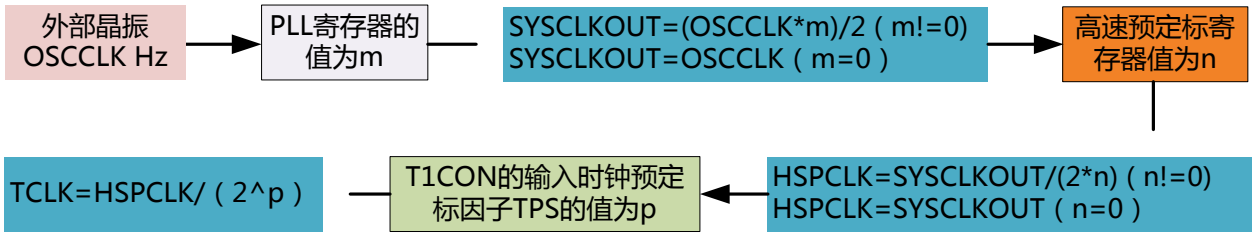


图 8 计算定时器 T1 时钟的原理

从图 8 我们可以看到，假设外部晶振的频率为 OSCCLK Hz，我们常用的是 30M 的晶振，也就是说 OSCCLK=30M，然后经过 2812 内部的 PLL 模块得到系统时钟 SYSCLKOUT，假设 PLL 寄存器的值为 m，则

$$SYSCLKOUT = OSCCLK(m=0)$$
$$SYSCLKOUT = \frac{OSCCLK * m}{2}(m \neq 0) \quad (1)$$

如果 OSCCLK=30M，m=10，则 SYSCLKOUT 就等于 150M 了，达到了 2812 所能支持的最高时钟频率。SYSCLKOUT 信号通过高速预定标因子才能得到提供给例如向 EV、AD 等高速外设的高速时钟 HSPCLK。假设高速预定标寄存器的值为 n，则 HSPCLK 如式 2 所示。

$$HSPCLK = SYSCLKOUT(n=0)$$
$$HSPCLK = \frac{SYSCLKOUT}{2*n}(n\neq 0) \quad (2)$$

HSPCLK 是不是就是提供给 T1 的时钟了呢？还不一定，我们还得看 T1CON 中的定时器输入时钟预定标因子，假设其值为 p，则实际最终提供给 T1 的时钟 TCLK 的值可以通过下面的式子求得：

$$TCLK = \frac{HSPCLK}{2^p} \quad (3)$$

终于得到了输入到定时器 T1 的时钟，现在就不难算出 T1 计数一个周期所需要的时间了：

$$T = (T1PR+1)*tc = \frac{T1PR+1}{TCLK} \quad (4)$$

因此我们从 PWM 产生的原理也不难得出 T1PWM 此时的各个参数：

T1PWM 的周期为：$\frac{T1PR+1}{TCLK}$ ，单位为 s；

T1PWM 的频率为：$\frac{TCLK}{T1PR+1}$，单位为 Hz；

T1PWM 的占空比要分 GPTCONA 中 T1PIN 的输出极性，当 T1PIN 为高电平有效时，则占空比为：

$$D = \frac{T1PR+1-T1CMPR}{T1PR+1} \quad (5)$$

当 T1PIN 为低电平有效时，PWM 波形的占空比为：

$$D = \frac{T1CMPR}{T1PR+1} \quad (6)$$

（2）当 T1CNT 的计数方式为连续增/减计数时，T1PWM 引脚输出对称的 PWM 波形。

当定时器 T1 的控制寄存器 T1CON 的 TMODE1 和 TMODE0 为 01 时，定时器 T1 工作于连续增/减计数模式。当 T1CNT 的值计数到和 T1CMPR 的值相等时，发生比较匹配事件。如果 T1CON 的第 1 位定时器比较使能为 TECMPR 为 1，即定时器比较操作被使能，且 GPTCONA 的第 6 位比较输出使能位 TCMPOE 为 1，同时 GPTCONA 下的 T1PIN 引脚输出极性为高电平或者低电平的话，T1PWM 就会输出对称的 PWM 波形，如图 9 所示。

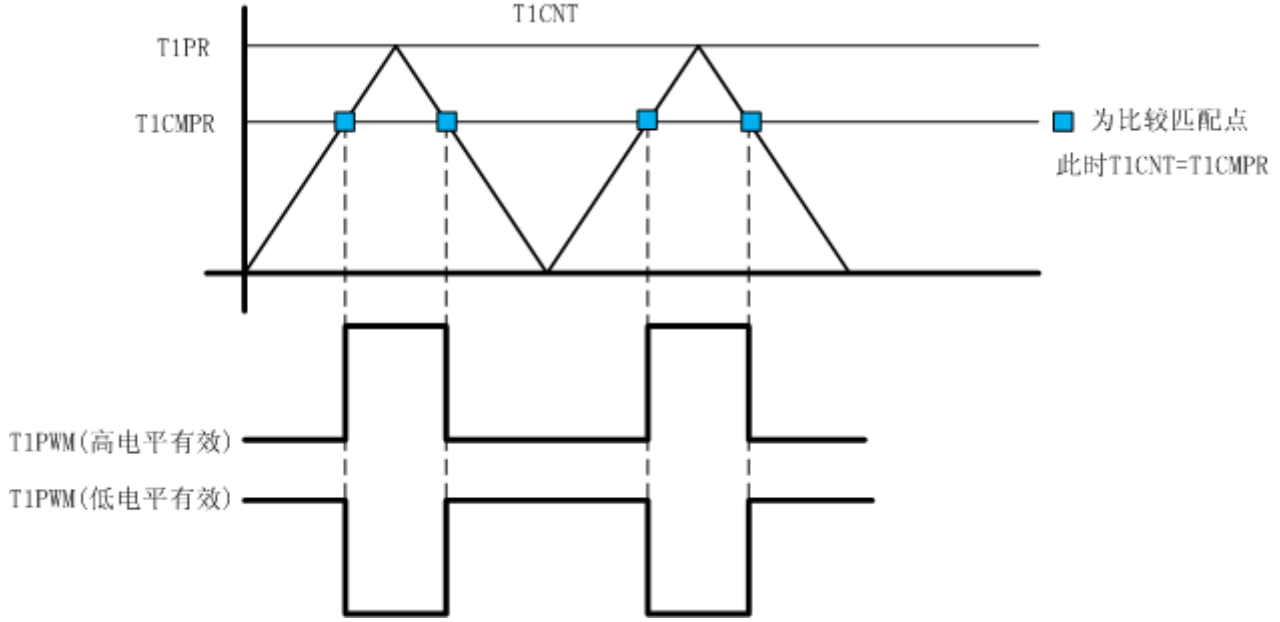


图 9 T1 产生对称的 PWM 波形

从图 9 不难看出，当 T1 工作于连续增/减计数模式时，T1CNT 计数一个周期所需的时间是（2*T1PR）*tc，其中 tc 是 T1CNT 计一次数所花的时间，tc 的计算方法和前面的一样，这里不再重复叙述。结合 PWM 波形的生成原理和图 9，图中 T1PWM 输出的 PWM 波形各个参数计算方法如下：

T1PWM 的周期为：$\frac{2*T1PR}{TCLK}$ ，单位为 s；

T1PWM 的频率为：$\frac{TCLK}{2*T1PR}$，单位为 Hz；

T1PWM 的占空比要分 GPTCONA 中 T1PIN 的输出极性，当 T1PIN 为高电平有效时，则占空比为：

$$D=\frac{T1PR-T1CMPR}{T1PR} \quad (5)$$

当 T1PIN 为低电平有效时，PWM 波形的占空比为：

$$D=\frac{T1CMPR}{T1PR} \quad (6)$$

前面我们介绍了由寄存器 T1 的比较功能产生的 PWM 波形，当然，T2 的比较功能产生 PWM 的原理是和 T1 一样的。下面我们来介绍由比较单元产生的可带有死区的 PWM 波形。

（3）比较单元产生的可带有死区的 PWM 波形

我们在电机控制、开关电源、变频器、逆变器等电力电子电路中，经常会遇到如下图所示的三相全桥控制电路，该电路由 6 个开关管组成，上下两个开关管组成 1 个桥壁。任何一个开关管在输入的 PWM 波形处于高电平时导通，处于低电平时关断。同一桥壁上的上下两个开关管不能同时导通，因为如果同时导通，电源和地就会短接，也就是会发生短路。因此，PHa1 和 PHa2，PHb1 和 PHb2，PHc1 和 PHc2 必须都是互补的，以 PHa1 和 PHa2 为例，理想情况下当 PHa1 为高电平时，PHa2 为低电平；当 PHa1 为低电平时，PHa2 为高电平，如图 11 所示。

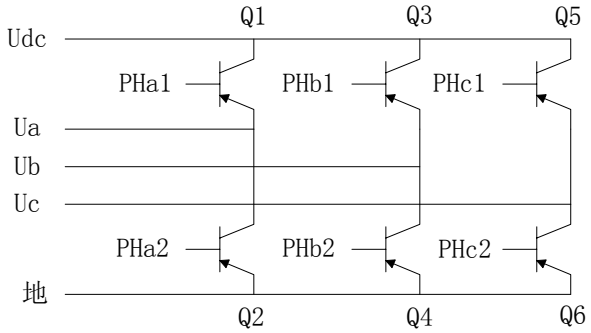


图 10 三相全桥电路

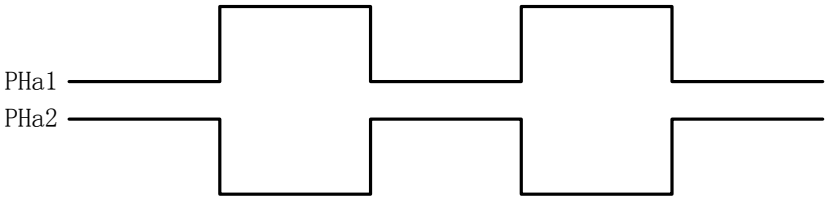


图 11 桥电路理想驱动波形

图 11 为开关管理想的驱动波形，但实际上会有下面的问题。PHa1 为高电平时 Q1 导通，此时 PHa2 为低电平，Q2 关闭，当 PHa1 从高电平转变为低电平时，Q1 由导通变为关断，而此时 Q2 由关断变为导通，实际上开关管从导通转为关断的时候，总会有延时，这样，就会有一小段时间里面其实 Q1 和 Q2 都处于导通状态，这样是非常危险的。为了解决这个问题，我们通常要求上下管输出的驱动波形要具有一定的死区时间，如图 12 所示。这样，上下桥壁中任何一个开关管从关断到导通都要经过 1 个死区时间的延时，也就是等到 Q1 完全关断的时候，Q2 才会导通，反过来也是一样。具体的死区时间如何确定，需要由开关管的参数来决定。

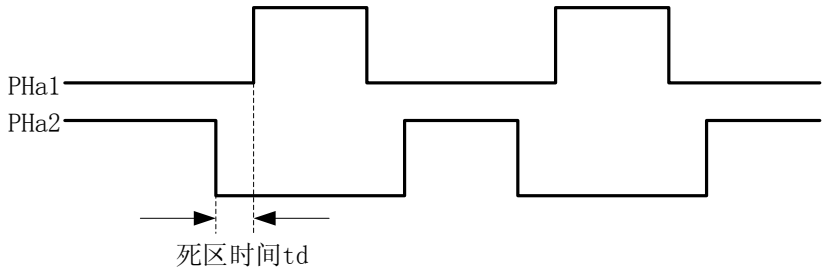


图 12 桥电路实际带死区的驱动波形

听起来是不是有些麻烦？如果要像前面 T1 或者 T2 产生 PWM 的方法，那生成 6 路 PWM 波形，既要互补，还要带有死区，那确实是相当复杂的，而且 T1 和 T2 每个定时器也只能生成 1 路 PWM 波，肯定无法满足要求。不要担心，2812 的 EV 还为我们提供了 3 个全比较单元，分别是比较单元 1，比较单元 2 和比较单元 3。如图 13 所示，这 3 个全比较单元每一个都能产生一对互补的 PWM 波形，也可以通过相应的寄存器设置死区时间。这样，使得 EVA 和 EVB 都有能力去驱动一个三相全桥电路。

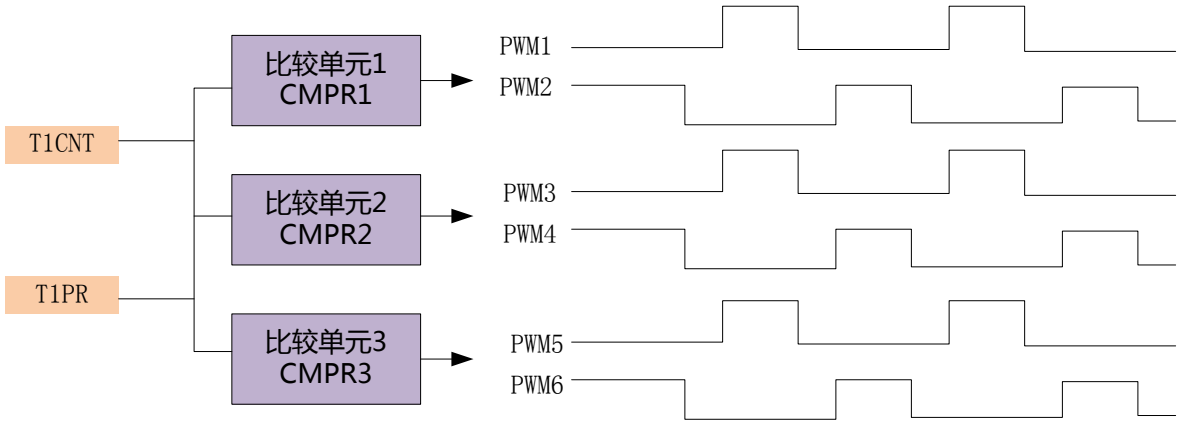


图 13 EV 下的全比较单元

比较单元产生 PWM 的波形和定时器通过比较功能产生 PWM 波形的原理是类似的。只不过定时器中的比较寄存器例如 T1CMPR，变成了比较单元的比较寄存器，例如 CMPR1。由于三个比较单元都是类似的，我们就以比较单元 1 为例来进行介绍。比较单元产生 PWM 时，所相关的寄存器有 T1PR,T1CNT,CMPR1,比较控制寄存器 COMCONA 和比较行为控制寄存器 ACTRA。

比较单元的时基是由 T1 来提供的，因此我们用到的是 T1PR 和 T1CNT，可见和 T2 是没有关系的。当 T1CNT 中的值和 CMPR1 中的值相等时，就发生了比较匹配。这时候，如果 COMCONA 的 CENABLE 为 1，即比较操作被使能，FCMPOE 为 1，比较输出时各路 PWM 波形都由相应的比较逻辑来驱动，同时如果 ACTRA 中 CMP1 和 CMP2 的极性为低电平或者高电平有效的时候，就会产生两路互补的 PWM 波形，PWM1 和 PWM2。和 T1 产生 PWM 一样，当 T1 工作于连续增计数模式时，比较单元 1 输出不对称的 PWM 波形，而当 T1 工作于连续增减计数模式时，比较单元 1 输出对称的 PWM 波形。比较单元输出的 PWM 波形的各种参数和 T1 产生的 PWM 波形的各种参数计算方法基本上都是一样的，所以我们在这里就不重复讲述了，我们重点来研究比较单元的死区电路。

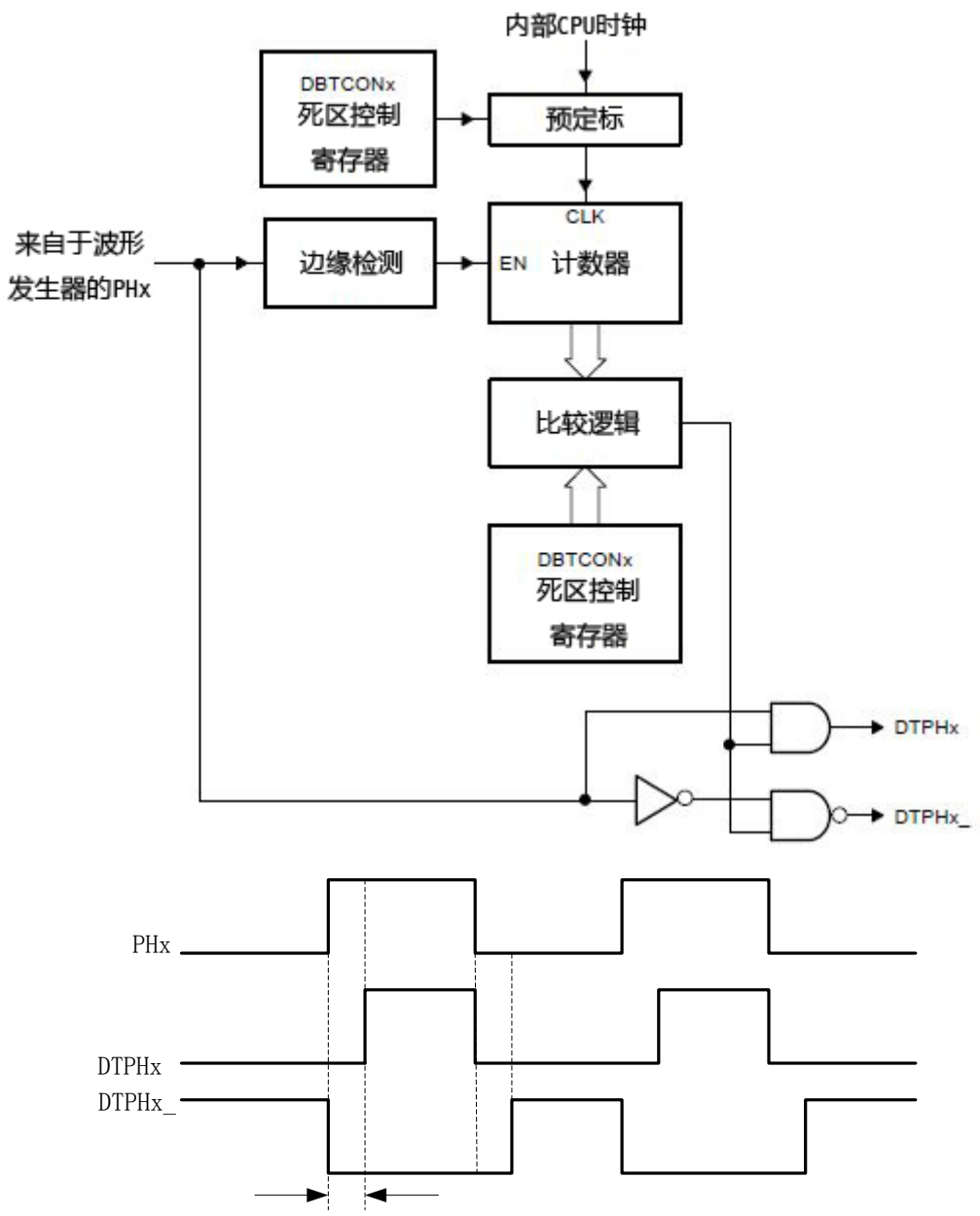


图 14 死区单元的模块图（x=1,2,3）

当比较单元的比较操作被使能，就会产生波形 PHx。PHx 经过死区单元，就会输出两路互补的带有死区的 PWM 波形 DTPHx 和 DTPHx_。我们仔细看看 PHx、DTPHx、DTPHx_之间的关系，如果没有死区，那么 DTPHx 和 DTPHx_应该是完全互补的。DTPHx 的导通时刻是在 PHx 的基础上延时了 1 个死区时间，而关闭时刻未变。DTPHx_是在 PHx 取反的基础上，也将导通时间延迟了 1 个死区时间，而关断的时间没有发生改变。

那具体的波形的死区时间如何控制呢？我们需要涉及到死区控制寄存器 DBTCONx 的[11~8]死区定时器周期和 DBTCON 的[4~2]位死区定时器预定标因子。如果死区定时器周期为 m，死区定时器预定标因子 x/p，则死区的值就为（p*m）个 CPU 时钟周期。

比较单元 1 产生带有死区的不对称的和对称的 PWM 波形分别如图 15 和图 16 所示。

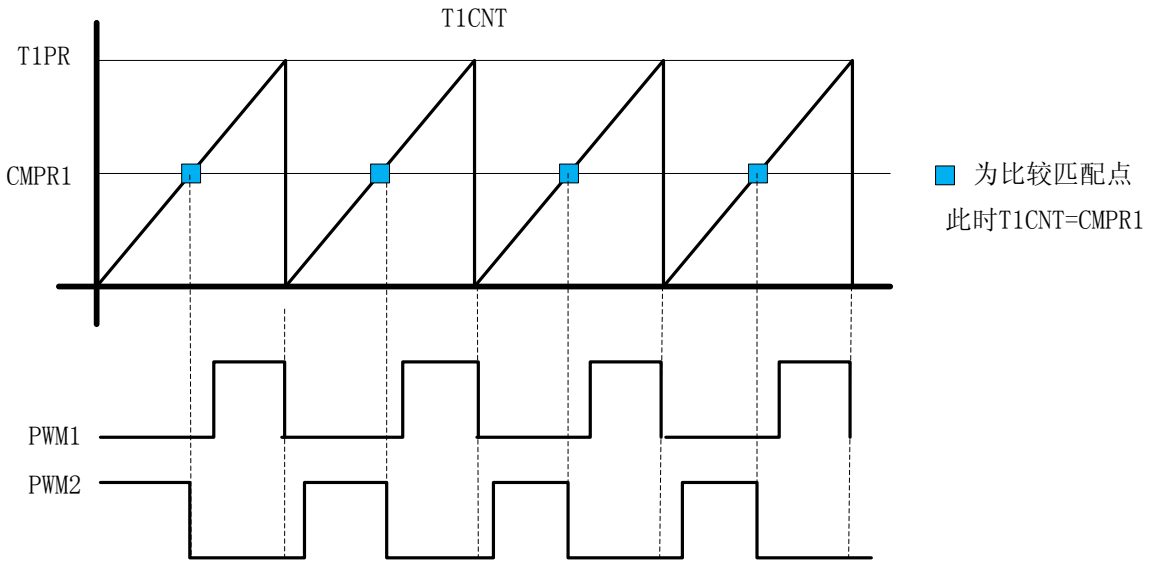


图 15 比较单元 1 产生的不对称 PWM 波形

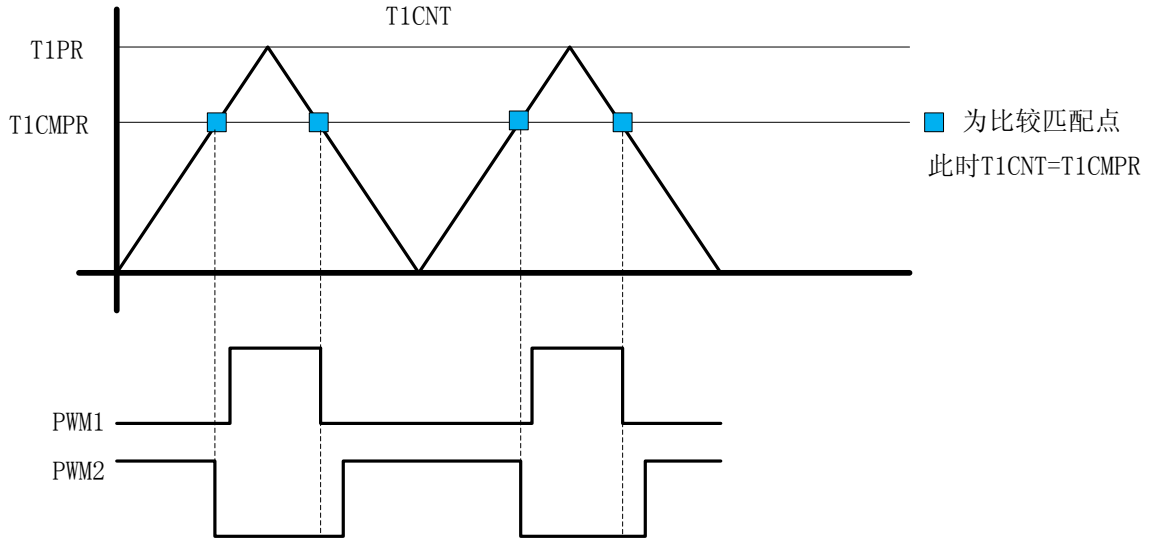


图 16 比较单元 2 产生对称的 PWM 波形

看了上面的介绍，我想大家对于 EV 的定时器和比较单元，以及它们产生 PWM 的机制已经有所了解了，下面我们一起来看一下 EV 的另外一个常用的模块——捕获单元。

4.捕获单元

2812 的 EVA 和 EVB 分别有 3 个捕获单元，EVA 的 CAP1、CAP2、CAP3 和 EVB 的 CAP4、CAP5、CAP6，每一个捕获单元都会有一个捕捉引脚。每一个捕获单元通过相关寄存器的设置能够捕捉输入波形的上升沿，下降沿或者同时捕捉上升沿和下降沿，捕获单元的简单框图如图 17 所示。与 EVA 中的捕获单元相关的寄存器有：捕捉控制寄存器 CAPCONA，捕捉 FIFO 状态寄存器 CAPFIFOA，2 级深度 FIFO 堆栈 CAPFIFO1~3，CAP1FBOT，CAP2FBOT，CAP3FBOT。

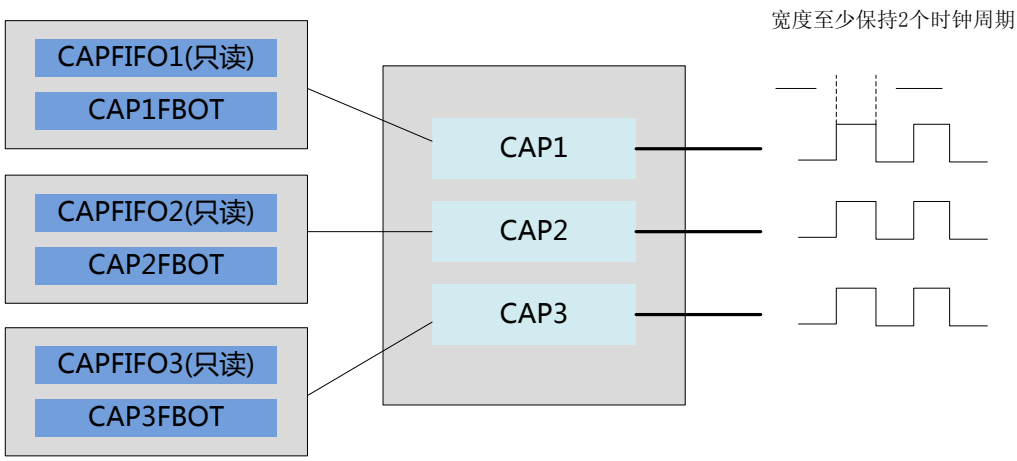


图 17 EVA 捕获单元简单框图

EVA 中的每个捕获单元都能选择定时器 1 或者定时器 2 作为自己的时基，但是 CAP1 和 CAP2 不能选择不同的定时器作为它们自己的时基，也就是说 CAP1 和 CAP2 必须选择相同的定时器来作为自己的时基，而定时器 3 可以根据需求随意进行选择。还值得一提的是，捕捉引脚在捕捉波形的边沿变化情况时，从引脚变化发生到定时器的计数器锁存需要 2 个时钟周期，因此，为了能够捕捉到变化，输入信号应该要至少保持当前状态 2 个时钟周期。捕获单元的捕捉操作不会影响到定时器的任何操作以及与定时器相关的比较/PWM 操作。

从图 17 中我们可以看到，每个捕获单元都有一个专用的 2 级深度的 FIFO 堆栈，顶层堆栈由 CAPFIFO1、CAPFIFO2、CAPFIFO3 组成，底层堆栈由 CAP1FBOT、CAP2FBOT、CAP3FBOT 组成。堆栈的顶层寄存器是只读寄存器，通常存储捕获单元捕捉到的旧值，对 FIFO 堆栈进行读操作的时候，我们读到的是顶层堆栈中的旧值。当 FIFO 堆栈上层寄存器中的旧值被读取后，堆栈底层寄存器中的新值就会被推入顶层寄存器。下面我们以 EV 下 CAP1，并选用的是定时器 T1 作为时基为例来详细的分析捕捉的过程。

（1）第一次捕捉

第一次捕捉，此时堆栈应该为空。当捕捉引脚 CAP1 捕捉到指定的变化时，捕获单元就会捕捉定时器 T1 的计数寄存器 T1CNT 中的值，并将其写入 FIFO 堆栈的上层寄存器 CAPFIFO1 中，这时候，捕捉 FIFO 状态寄存器 CAPFIFOA 中的 CAP1FIFO 状态位变为 01。如果在另一次捕捉发生前读取 FIFO 堆栈的值，则状态位 CAP1FIFO 变为 00。

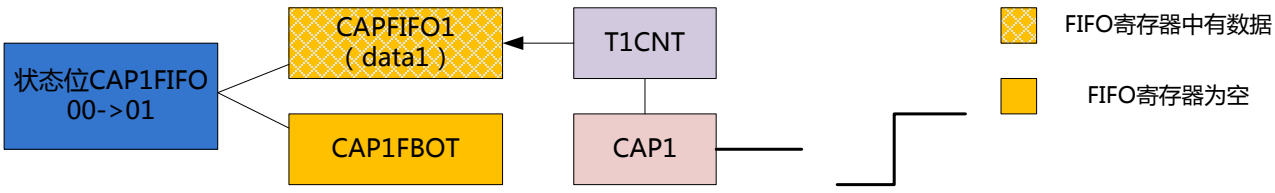


图 18 第一捕捉示意图

（2）第二次捕捉

如果在第一次捕捉的值被读取之前发生第二次捕捉，也就是说在 CAPFOFO1 里面还有数据的时候发生第二次捕捉，则新的捕捉值就会被送入底层寄存去 CAP1FBOT 中，这时候，捕捉 FIFO 状态器 CAPFIFOA 中

的 CAP1FIFO 状态位变为 10，说明现在堆栈中有 2 个数据。如果在另一次捕捉发生前，上层寄存器 CAPFIFO1 中的值被读出，则 CAP1FBOT 中的新值就会被推入 CAPFIFO1，同时状态位 CAP1FIFO 变为 01。

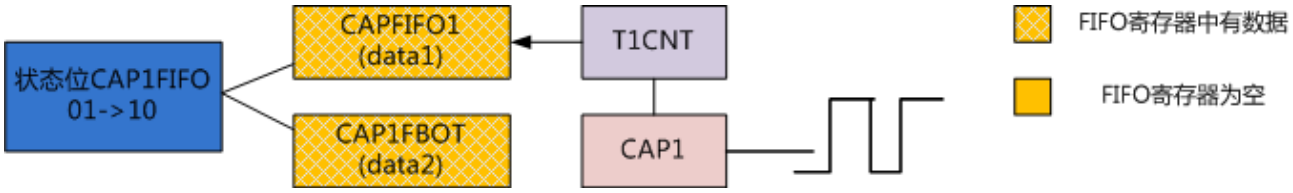


图 19 第二次捕捉示意图

（3）第三次捕捉

如果 FIFO 堆栈中已经存储了两个值，这时又发生了一次捕捉，上层寄存器 CAPFIFO1 的值就会丢失，底层寄存器的值被推入上层寄存器中，而新的捕捉值将写入底层寄存器，同时状态位 CAP1FIFO 会变为 11，说明有 1 个或者多个捕捉旧值已经丢失。当然，如果在第 3 次捕捉之前已经读取了第一次捕捉的旧值，那么底层寄存器的值被送入顶层寄存器，新的值被写入底层寄存器，状态位为 10，和第二次捕捉的情况相同。

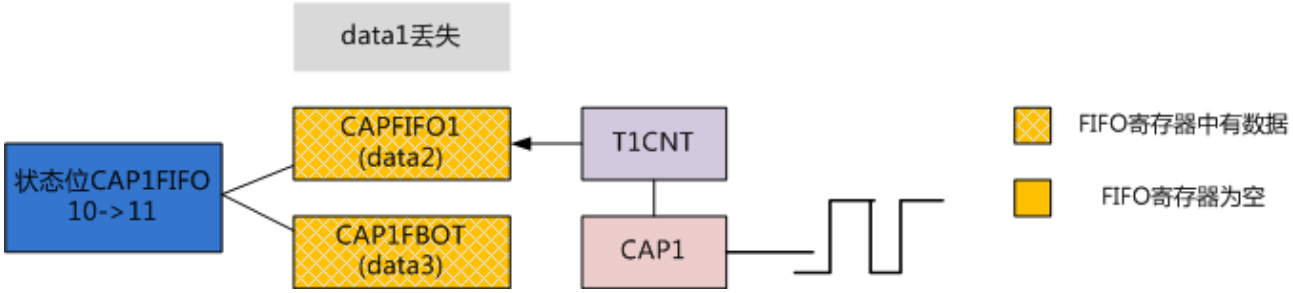


图 20 第 3 次捕捉前面的旧值未读取

如果说捕获单元进行了捕捉，而且 FIFO 堆栈中已经有了 1 个合法的数据，就是说 CAPxFIFO 的值不为 0，则相应的中断标志位置 1，如果中断被使能了，会产生外围设备的中断请求。因此，读取捕捉的数据有两种方法，一种是通过中断，在捕获中断程序中读取捕捉的值，另一种不通过中断程序，而是通过去查询中断标志位和状态位来确定捕捉事件的发生并读取捕捉的数值。

到这里，EV 中的主要内容就介绍的差不多了，还有 1 个正交编码电路，由于用的并不是非常多，所以请大家自行参看各自的书籍进行学习。接下来，我们将和大家一起探讨 EVA 的 2 个例程来学习如何编写产生 PWM 波的程序。请大家自行下载附件 hellodspT1pwm.rar 和 hellodspmotor.rar。

我们首先来看例程 hellodspT1pwm.pjt。在这个例程里，我们需要定时器 T1 产生一个不对称的 PWM 波形，频率为 1KHz，占空比为 40%。我们所使用的程序框架还是和前面介绍的一样，今后不做特殊说明，我们所有的例程，使用的都是 TI 例程的程序框架。

我们来看看应该如何写这个程序。首先第一步是初始化系统时钟，在 DSP28_SysCtrl.c 文件中编写函数 InitSysCtrl（），这个函数的编写我们在上节课中已经做了详细的分析。

系统时钟初始化
void InitSysCtrl(void) { Uint16 i;

```
    EALLOW;

// 禁止开门狗
    SysCtrlRegs.WDCR= 0x0068;

// 初始化 PLL 模块
    SysCtrlRegs.PLLCR = 0xA;   ///系统时钟 30*10/2=150M
// 延时，使得 PLL 初始化成功
    for(i= 0; i< 5000; i++){}

    SysCtrlRegs.HISPCP.all = 0x0001; //HSPCLK=150M/2=75M
    SysCtrlRegs.LOSPCP.all = 0x0002; //LSPCLK=150M/4=37.5M

    SysCtrlRegs.PCLKCR.bit.EVAENCLK=1; //使能 EVA 的时钟

    EDIS;

}
```

然后因为 2812 的 T1PWM 引脚和 IO 是复用的，所以我们得设定一下 T1PWM 这个引脚，使得它的功能是输出 PWM 而不是普通的 IO 口。这就需要在 DSP28_Gpio.c 文件中编写函数 InitGpio()。

初始化 GPIO 口

```
void InitGpio(void)
{

     EALLOW;
     GpioMuxRegs.GPAMUX.bit.T1PWM_GPIOA6=1;
     //将 GPIOA6 脚功能设置为 T1PWM，而不是普通的 IO 口
     EDIS;
}
```

接下来就是得设置我们的重点部分了，初始化 EV。我们需要在 DSP28_Ev.c 文件中编写 InitEv（），声明一下，例程中的程序风格只是我个人喜欢的，大家可以自行选择，呵呵。

初始化 EV

```
void InitEv(void)
{
```

```
/***************************/
/*设置定时器控制寄存器 T1CON*/
/***************************/

EvaRegs.T1CON.bit.TMODE=2;
//因为要产生不对称的 PWM 波形，所以将 T1 的计数方式设定为连续增模式

EvaRegs.T1CON.bit.TPS=1;
//这个位是 T1 输入时钟预定标位，此时，T1CLK=HSPCLK/2=75/2=37.5M

EvaRegs.T1CON.bit.TENABLE=0;
//暂时禁止定时器操作，等到全部设定完毕，再启动定时器

EvaRegs.T1CON.bit.TCLKS10=0;
//定时器使用内部时钟

EvaRegs.T1CON.bit.TCLD10=2;
//如果给定时器比较寄存器赋值，则立即重载

EvaRegs.T1CON.bit.TECMPR=1;
//使能定时器的比较操作

/***************************/
/********设置 GPTCONA********/
/***************************/

EvaRegs.GPTCONA.bit.TCOMPOE=1;
//定时器比较输出 T1PWM 或者 T2PWM 由各自的定时器比较逻辑驱动

EvaRegs.GPTCONA.bit.T1PIN=1;
//定时器 1 的比较输出极性是低电平有效

/***************************/
/*****设置 T1PR 和 T1CMPR******/
/***************************/
```

```
EvaRegs.T1PR=37499;
EvaRegs.T1CMPR=15000;
EvaRegs.T1CNT=0;

EvaRegs.T1CON.bit.TENABLE=1;
//启动定时器操作，开始产生 PWM 波
```

在这里，还是给大家重点分析一下 T1PR 和 T1CMPR 的值的由来。我们要求的 PWM 波是 1KHZ，占空比为 40%的。我们通过系统时间初始化的时候知道 HSPCLK=75M，然后通过 T1 的预定标因子，T1 的时钟就成了 37.5M。也就是说，T1CNT 每计数一次，所需要的时间是（1/37.5）us。由于 T1 现在是连续增模式，那么 1 个 PWM 周期需要的是（1+T1PR）个时钟周期，因此有：

$$(1+T1PR)*\frac{1}{37.5*10^{6}}=10^{-3} \tag{7}$$

解上面的方程便可以得到 T1PR=37499。

然后，由于 T1PWM 的占空比为 40%，根据我们 page9 的式 6 有：

$$0.4=\frac{T1CMPR}{37499+1} \tag{8}$$

解上面的方程便可以得到 T1CMPR=15000；

再说明一下，这里的 37499 或者 15000，都是十进制表示的，如果您想表示成 16 进制的话也是可以的，只是需要转换一下，例如：

```
EvaRegs.T1PR=0x972B;
EvaRegs.T1CMPR=0x3A98;
```

最后我们来写任何一个工程都不能少的 Main 函数。

Main 函数

```
void main(void)
{
    /*初始化系统*/
    InitSysCtrl();

    /*关中断*/
    DINT;
    IER = 0x0000;
    IFR = 0x0000;

    /*初始化 PIE 控制寄存器*/
```

```
    InitPieCtrl();

    /*初始化 PIE 矢量表*/
    InitPieVectTable();

    /*初始化 GPIO*/
    InitGpio();

    /*初始化 EV*/
    InitEv();

    EINT;
    ERTM;

    for(;;)
    {
    }

}
```

这是一个最基本的最简单的程序，但是所有复杂的程序都是由很多简单的模块组合起来的，所以希望大家不要小看这样的小例程。有条件的朋友请上开发板跑一下吧，看看 T1PWM 脚是不是真的输出了我们所期望的 PWM 波形？呵呵，好吧，让我们来看下一个例程。下面的这个例程是专门为购买我们 2812 开发工具套餐的朋友所讲的，如果您购买了我们的学习工具套餐，而且也买了直流电机，那么将这些设备拿出来吧，我们要来做实验啦，呵呵。基础的例程就是 hellodspmotor.pjt。

首先第一步也是初始化系统时钟，在 DSP28_SysCtrl.c 文件中编写函数 InitSysCtrl（），

系统时钟初始化

```
void InitSysCtrl(void)
{
    Uint16 i;
    EALLOW;

    SysCtrlRegs.WDCR= 0x0068;
    //禁止看门狗

    SysCtrlRegs.PLLCR = 0x03;
```

```
    //初始化 PLL 模块，系统时钟 SYSCLKOUT=30M*3/2=45M

    for(i= 0; i< 5000; i++){}
    //延时，以确保 PLL 初始化成功

    SysCtrlRegs.HISPCP.all = 0x0001;
    //高速时钟 HSPCLK=45M/2=22.5M

    SysCtrlRegs.LOSPCP.all = 0x0002;
    //低速时钟 LSPCLK=45M/4=11.25  M

    SysCtrlRegs.PCLKCR.bit.EVAENCLK=1;
    //使能 EV 时钟

    EDIS;

}
```

然后因为 2812 的 PWM1 和 PWM2 引脚和 IO 是复用的，所以我们得设定一下 PWM1 和 PWM2 这两个引脚，使得它的功能是输出 PWM 而不是普通的 IO 口。这就需要在 DSP28_Gpio.c 文件中编写函数 InitGpio()。

初始化 GPIO

```
void InitGpio(void)
{

     EALLOW;

     GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0=1;
     GpioMuxRegs.GPAMUX.bit.PWM2_GPIOA1=2;
     //初始化 GPIO 口，设置 PWM1 和 PWM2 的引脚为 PWM 功能，而不是普通的 IO 口

     EDIS;
}
```

接下来，我们需要在 DSP28_Ev.c 文件中编写 InitEv（）。

初始化 EV

```
void InitEv(void)
{
```

```
/*****************************/
/**********配置 ACTR*********/
/*****************************/

EvaRegs.ACTR.bit.CMP1ACT=2;
//PW1 引脚高电平有效

EvaRegs.ACTR.bit.CMP2ACT=1;
//PWM2 引脚低电平有效

/*****************************/
/*******配置死区寄存器********/
/*****************************/

EvaRegs.DBTCONA.bit.DBT=5;
//死区定时器周期为 5

EvaRegs.DBTCONA.bit.EDBT1=1;
//死区定时器 1 使能

EvaRegs.DBTCONA.bit.DBTPS=3;
//死区定时器预定标因子，死区时钟为 HSPCLK/8

/*****************************/
/*********配置 COMCONA******/
/*****************************/

EvaRegs.COMCONA.bit.CENABLE=1;
//使能比较单元的比较操作

EvaRegs.COMCONA.bit.CLD=2;
//当给 CMPR1 赋新值时，立即装载

EvaRegs.COMCONA.bit.FCOMPOE=1;
//全比较操作，PWM 输出由相应的比较逻辑驱动
```

```
    /*****************************/
    /**********设置 T1CON**********/
    /*****************************/

    EvaRegs.T1CON.bit.TMODE=2;
    //T1 工作于连续增模式

    EvaRegs.T1CON.bit.TPS=3;
    //输入时钟预定标因子为 X/8

    EvaRegs.T1CON.bit.TENABLE=1;
    //禁止定时器操作

    EvaRegs.T1CON.bit.TCLKS10=0;
    //使用内部时钟

    EvaRegs.T1CON.bit.TECMPR=1;
    //使能定时器比较操作



    /*****************************/
    /**********设置 T1PR 等*********/
    /*****************************/
    EvaRegs.T1PR =4999;
    //一个周期时间为(4999+1)*0.365us

    EvaRegs.T1CNT = 0;              ///定时器 1 初值设为 0
    EvaRegs.CMPR1=3500;

    EvaRegs.T1CON.bit.TENABLE=1;
    //启动定时器 T1
}
```

在这里，我们一起来计算一下死区时间。如果死区定时器周期为 m，死区输入时钟预定标因子为 x/p，那么死区时间就是（m*p）*T1CLK。在这里，m=5,p=8,T1CLK=(1/22.5)us，因此，我们得到的死区时间是：

$$T_{dt} = m * p * T_{clk} = \frac{5*8}{22.5} = 1.78us \qquad (9)$$

最后，还是我们的 main 函数了，呵呵。

Main 函数

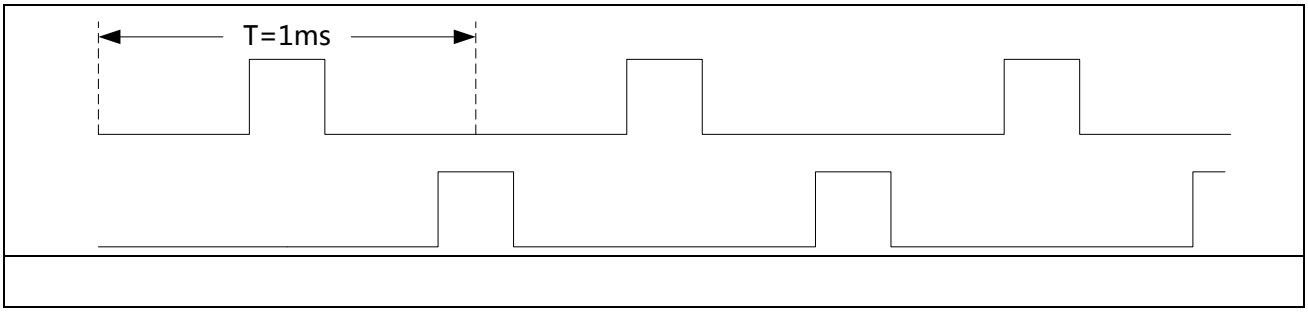
```
void main(void)
{
    /*初始化系统*/
    InitSysCtrl();

    /*关中断*/
    DINT;
    IER = 0x0000;
    IFR = 0x0000;

    /*初始化 PIE 控制寄存器*/
    InitPieCtrl();

    /*初始化 PIE 矢量表*/
    InitPieVectTable();

    /*初始化 GPIO*/
    InitGpio();

    /*初始化 EV*/
    InitEv();

    EINT;
    ERTM;

    for(;;)
    {

    }

}
```

上面的是一个非常简单的例程，能让电动机转起来，因为小直流电机功率小，转动起来可能现象不是那么明显，简单的方法就是在直流电机转轴上帖个小标签，这样转动起来就会带动标签的转动，如果一开始电机没有转，不要紧，因为有可能是启动转矩比较小，你用手拨一下，就开始转动了。改变 PWM1 和

PWM2 的极性，可以改变电动机转动的方向，改变 CMPR1 的值可以改变电动机的转速，值越小，转动的也就越快。

布置个作业吧，呵呵。

第 1 题：请您设计一个程序产生如下的 PWM 波形，这个波形也可以驱动桥电路，周期为 1ms，占空比为 20%。大家看看如何产生这两路 PWM 波形？



第 2 题：是有了我们的开发工具套餐和直流电机的朋友，请在 hellodspmotor.pjt 的基础上通过中断改变电机转速的实验。具体要求是，CMPR1 的值在 35000~500 之间变化，每过 500 个 T1 定时器的周期，CMPR1 的值就减少 100，看看电机是不是转的越来越快了？我们在下周给出例程。

通过这一讲的内容，我向您对 EV 应该有个基本的了解了，本人水平有限，不足之处敬请谅解，欢迎批评指出。下一讲，我们将来一起学习 AD 转换方面的知识，敬请期待！