定时器/计数器 E 型（TCE）和波形扩展（WEX）外设入门指南

TB3339

简介

作者：Teodor Lina，Microchip Technology Inc.

AVR® EB 系列单片机配备了功能强大的定时器，适用于从信号测量到事件同步和波形生成的多种应用。E 型定时器/计数器（Timer/Counter type E，TCE）为 16 位定时器/计数器外设，最多可生成四个脉宽调制（Pulse-Width Modulation，PWM）信号，并且具有多种波形生成模式。波形扩展（WEX）是一种可增强 TCE 功能的波形扩展外设。搭配使用 TCE 与 WEX 外设最多可以生成八个 PWM 信号。

16 位 PWM TCE 的功能包括提供精确的程序执行时序，或者执行命令以及生成频率和波形。TCE 由一个基本计数器和一组比较通道组成。基本计数器用于对时钟周期和事件进行计数，或者通过使能相应的事件对时钟周期进行计数。将比较通道与基本计数器配合使用可以进行比较匹配控制、频率生成和脉宽调制。

WEX 可增强 16 位 PWM TCE 的功能。该外设可以生成互补波形输出信号，在这些信号之间生成死区，生成和管理故障事件，以及改写端口输出。WEX 的主要特性是克服了在电机控制等应用中使用定时器时常见的问题。通过添加死区可确保信号不会发生重叠，从而避免晶体管开关时引起的直通问题。此外，WEX 还可以生成故障条件，以将所有信号驱动为用户定义的状态，这一功能在注重安全性的应用中非常重要。WEX 的所有功能均在硬件内部实现，因此在任何情况下生成的波形都是可预测的。

本技术简介旨在帮助读者熟悉 TCE 和 WEX 的多种工作模式，重点说明定时器的特殊性。若要更好地了解功能，请参见数据手册。

本文档涵盖以下三个具体用例：

- **使用 TCE（独立于 WEX）生成四个 PWM 信号：**
  将 TCE 初始化为双斜率模式以生成四个具有不同占空比和缩放选项的 PWM 信号。
- **使用 WEX 为八个信号生成模式：**
  通过配置 WEX 为每个周期性翻转的输出引脚生成一系列模式。
- **搭配使用 TCE 和 WEX 生成八个带故障检测的互补波形输出信号：**
  将 TCE 初始化为单斜率模式并生成四个 PWM 信号。通过配置 WEX 将四个 PWM 信号拆分成八个带死区的互补信号。每个 PWM 信号都具有不同的占空比，介于 0%与 100%之间。在每个比较寄存器中断期间，占空比都会在中断服务程序（Interrupt Service Routine，ISR）中递增。每 250 µs 触发一次故障事件，以将所有输出驱动为逻辑低电平 0。随后会解决故障事件并恢复正常操作，然后重复上述过程。

注：对于本文档中所述的每个用例，均提供两个代码示例：一个是基于 AVR16EB32 开发的裸机代码，另一个是使用 MPLAB®代码配置器（MPLAB Code Configurator，MCC）基于 AVR16EB32 开发的代码。

如需本技术简介中所述功能的 AVR16EB32 裸机代码示例，可单击此处：

如需本技术简介中所述功能的使用 MCC Melody 生成的 AVR16EB32 代码示例，可单击此处：

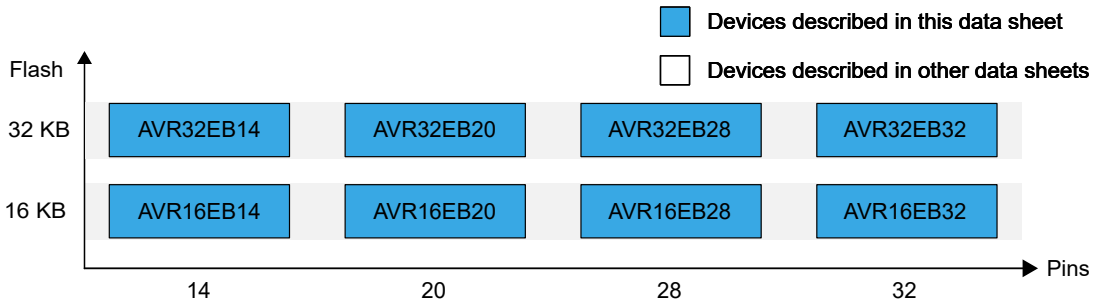
目录

1. 相关器件

本章列出了本文档的相关器件。下图给出了不同系列的器件之间的关系，并注明了不同的引脚数与存储器大小：

- 垂直向上移植无需修改代码，因为这些器件的引脚彼此兼容，可提供相同甚至更多的功能。向下移植到 AVR EB 系列可能需要修改代码，因为某些外设的可用引脚数较少。
- 水平向左移植会减少引脚数和可用的功能
- 具有不同闪存大小的器件通常具有不同的 SRAM 和 EEPROM

图 1-1. AVR® EB 系列概览



2. 概述

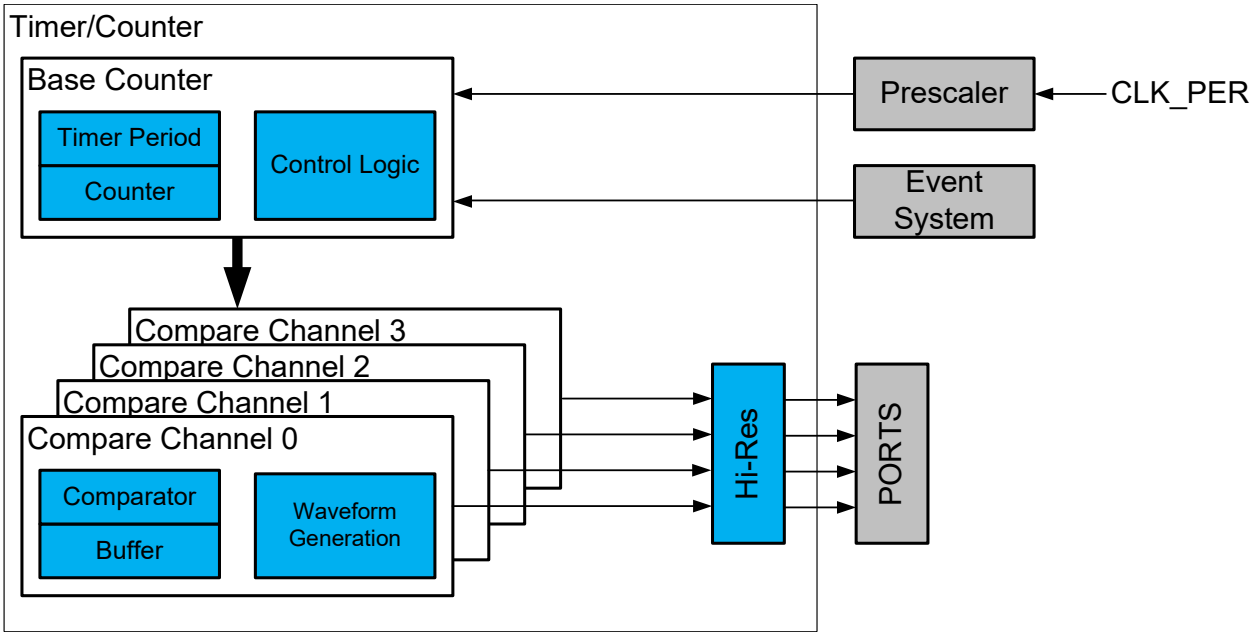
定时器/计数器概述

16 位 TCE 非常灵活，可提供精确的程序执行时序，或者用于生成频率和波形以及执行命令。

TCE 实例由一个基本计数器和四个比较通道组成。用户可根据时钟节拍（定时器）或不同的事件（计数器）设置基本计数器向上或向下计数。定时器/计数器可通过外设时钟（可选预分频）或 EVSYS 进行定时和计数。EVSYS 还可用于控制方向或同步操作。

计数器包括一个高分辨率选项，可将占空比分辨率提升至输入时钟的八倍。此外，还可使用预分频的外设时钟以及 EVSYS 生成的事件来控制计数器。

图 2-1. 定时器/计数器框图



计数器值会不断与零和周期（PER）值进行比较，分别用于确定计数器是否已达到底部或顶部。如果满足其中一个条件，则会更新计数器并产生中断。此外，计数器还与比较寄存器进行比较。如果选择波形生成模式，计数器将基于这些比较结果来产生中断并设置波形和脉宽周期。

计数器寄存器、周期寄存器和比较寄存器及其所有缓冲区的宽度均为 16 位。缓冲区是确保仅在计数器寄存器更新时才对相应寄存器进行更新的机制的一部分。每个缓冲区都有一个缓冲区有效（BV）位，用于判断对应的寄存器是否需要更新。

TCE 可配置为通过 EVSYS 对事件信号的上升沿和/或下降沿进行计数，或者用于使能时钟节拍计数。

如果使能 SCALEMODE 选项，TCE 支持向比较寄存器中写入一个介于 0 与 2 之间的小数值。比较值基于周期、幅值和偏移量值计算得出。

波形扩展概述

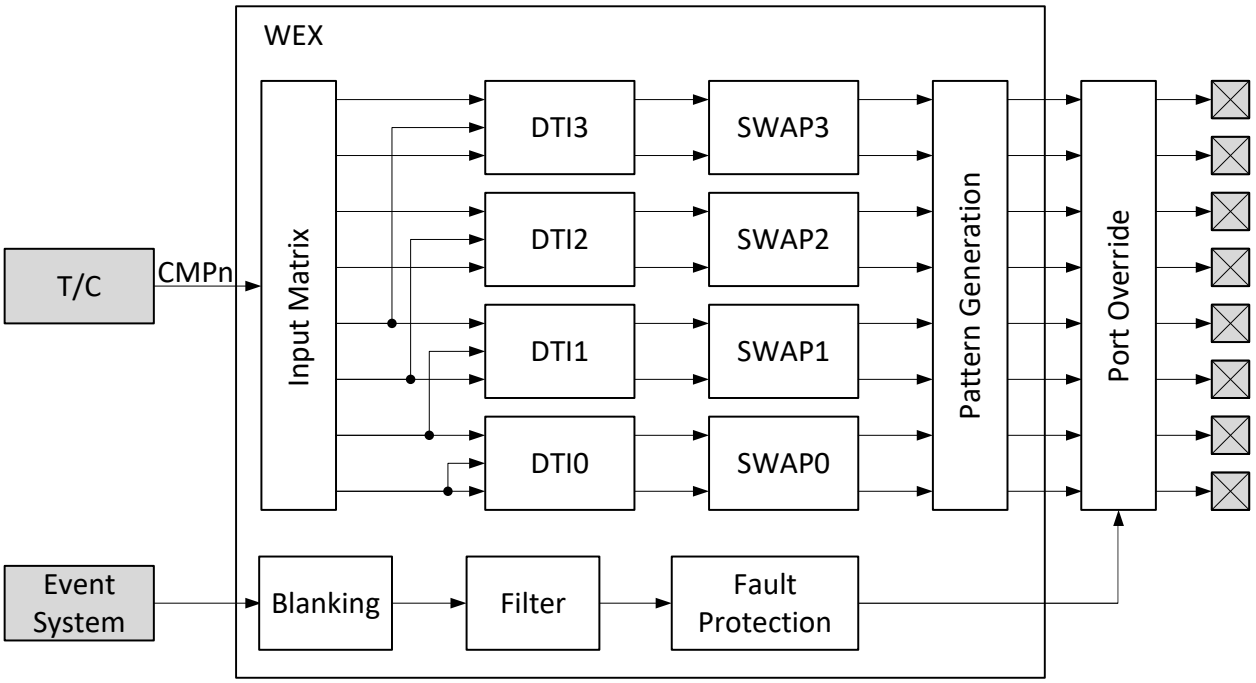
WEX 为波形生成（Waveform Generation，WG）模式下的定时器/计数器提供额外功能，例如带死区的互补信号生成功能。这些功能对于电机控制和电源控制应用很有帮助。

WEX 实例由五个独立单元组成：

- 输入矩阵单元
- 死区插入单元
- 交换单元
- 模式生成单元
- 故障保护单元

用户可以使用输入矩阵单元将定时器/计数器的波形输出以不同的配置连接到端口引脚。死区插入单元可以将定时器的四个输出拆分成两个不重叠的信号，即下桥臂信号和上桥臂信号。WEX 可基于定时器的四个 PWM 输出信号生成八个 PWM 输出信号。交换单元可用于交换下桥臂和上桥臂引脚位置，这一功能对于实现快速电机衰减至关重要。模式生成单元可以将端口引脚改写为恒定的逻辑电平，并且可以独立于定时器/计数器使用。用户最多可以配置八个引脚的模式。故障保护单元可在检测到故障事件时将 WEX 输出设置为定义的状态。故障保护单元与 EVSYS 相连，因此可通过事件触发故障。

图 2-2. 波形扩展框图



3. 使用 TCE 生成 PWM 信号

与 TCB 或 TCA 等定时器相比，TCE 的一个关键特性是 PWM 生成的多样性和精确性。用户可以根据应用的复杂度选择不同的配置。TCE 可配置为单斜率和双斜率 PWM 生成模式，从而在恒定相位（正确相位 PWM）与更高的最大工作频率（快速 PWM）之间进行权衡。此外，TCE 还采用了缓冲方案，可确保 PWM 信号无毛刺。TCE 的另一个特性是可以使用幅值和偏移量来缩放 PWM 信号的占空比。

TCE 和 TCB 均可用于生成具有较高的最大工作频率的 PWM 信号。但是，只有 TCE 可用于关键型应用，这得益于其基于可选方向的双斜率 PWM 功能。双斜率 PWM 在占空比发生变化时不会修改脉冲中心位置。因此，相位始终保持恒定。该特性对于电机控制应用至关重要，因为它可以避免由多个 PWM 信号同时换向引起的开关噪声。该噪声来自晶体管的开关操作。使用具有不同占空比的双斜率 PWM 信号可以防止此类噪声。

缓冲方案为每个比较寄存器和周期寄存器提供独立缓冲区。在关键型应用中，出现意外的长脉冲可能导致短路，此时缓冲方案至关重要。此外，缓冲区的存在还可以防止使用相同定时器但不同比较通道的两个外设之间失去同步。但是，考虑到可以直接更新周期寄存器和比较寄存器，用户可选择绕过缓冲方案。以下波形说明了缓冲操作与非缓冲操作之间的区别。

图 3-1. 非缓冲的双斜率操作

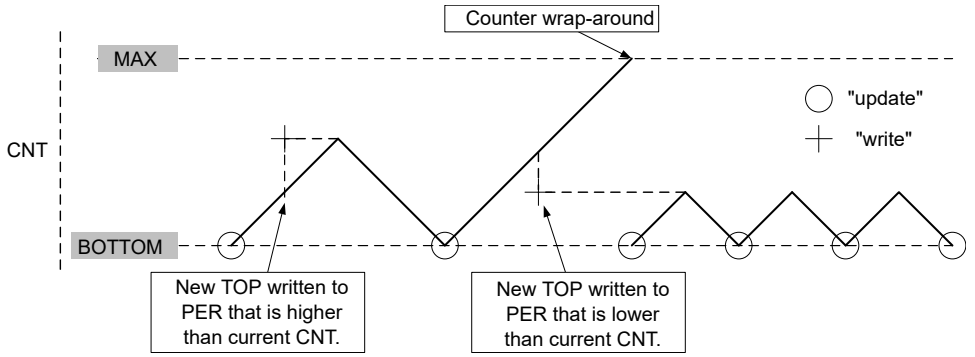
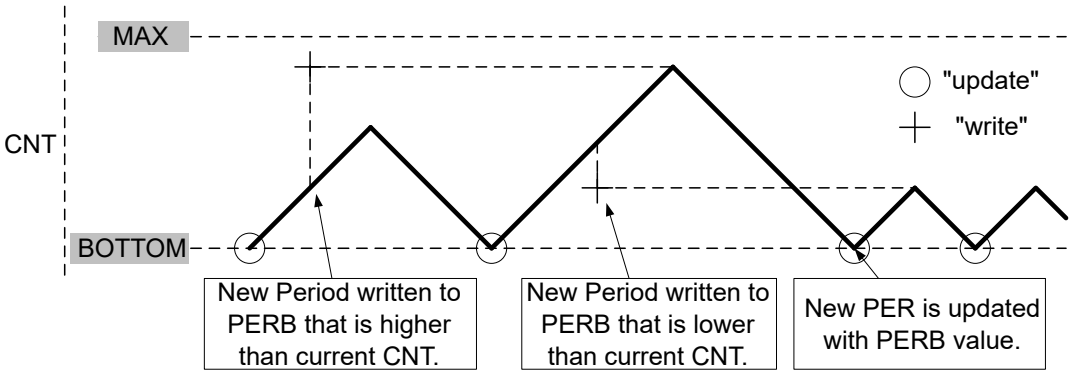


图 3-2. 使用缓冲方案更改周期



如果用户直接更改周期寄存器（非缓冲操作），定时器可能已超过新的阈值，因此将继续计数到最大值。这会导致出现异常长的脉冲，可能引发进一步的问题。此外，如果同时使用两个或多个比较通道，更新其中一个可能导致触发同步丢失。为了防止所有上述可能存在的问题，需要使用缓冲方案。当定时器更新时，缓冲区会保存新值并将其传输到比较寄存器或周期寄存器。当所有的值同时更改时，上述问题就会消失。

下面举例说明了如何设置 TCE 实例，以使用上述缓冲方案生成四个占空比分别为 20%、40%、60%和 80%的 10 kHz PWM 信号。本例重点展示了比较寄存器中设置的值缩放功能以及高分辨率功能。用户可将 PWM 信号分辨率提升至三位。

在本例中，用户可将占空比的最大范围从 0-100%更改为 0-75%、0-50%或 0-150%。占空比基于周期值进行计算。周期将始终为 100%，即占空比的最大值。所有超过 100%的占空比将饱和至 100%，即周期值。当用户将占空比范围设置为周期的 0-50%时，初始生成的四个 PWM 信号的占空比值 20%、40%、60%和 80%将缩放为 10%、20%、30%和 40%。

本例在运行时每 10 ms 修改一次 TCE 的比较寄存器的缩放值。占空比的范围将基于缩放值进行修改，下面前两个小节详细介绍如何配置 TCE 外设以实现所需的行为。第一小节介绍如何使用裸机代码配置 TCE。第二小节详细介绍如何通过 MCC Melody 配置 TCE。最后一个小节提供结果。

3.1. 裸机实现

1. 必须配置 CLKCTRL 外设才能使用 TCE0 的高分辨率功能。需要进行以下寄存器设置：

图 3-3. MCLKCTRLA 寄存器

Bit	7	6	5	4	3	2	1	0
					CLKSEL[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bits 3:0 – CLKSEL[3:0] Clock Select

This bit field controls the source for the Main Clock (CLK_MAIN).

Value	Name	Description
0x0	OSCHF	Internal high-frequency oscillator
0x1	OSC32K	32.768 kHz internal oscillator
0x2	XOSC32K	32.768 kHz external clock or 32.768 kHz external crystal oscillator, depending on the SEL bit in XOSC32KCTRLA
0x3	EXTCLK	External clock or external crystal, depending on the SELHF bit in XOSCHFCTRLA
0x4	PLL	PLL Oscillator
Other	Reserved	Reserved

图 3-4. MCLKCTRLB 寄存器

Bit	7	6	5	4	3	2	1	0
			PBDIV	PDIV[3:0]				PEN
Access			R/W	R/W	R/W	R/W	R/W	R/W
Reset			0	1	0	0	0	1

Bit 5 – PBDIV Prescaler B Division

If this bit is written to '1' a clock running at 4x the main clock is available for peripherals that can take use of this. When prescaler B division is enabled, only prescaler settings matching 2^n are available for PDIV.

Value	Description
0x0	(NONE) = No Division
0x1	(DIV4) = Divide by 4

Bit 0 – PEN Prescaler Enable

This bit controls whether the Main Clock (CLK_MAIN) prescaler is enabled or not.

Value	Description
0	The CLK_MAIN prescaler is disabled
1	The CLK_MAIN prescaler is enabled and the division ratio is controlled by the Prescaler Division (PDIV) bit field

图 3-5. PLLCTRLA 寄存器

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY	SOURCE[1:0]		SOURCEDIV[1:0]			MULFAC[1:0]	
Access	R/W	R/W	R/W	R/W	R/W		R/W	R/W
Reset	0	0	0	0	0		0	0

Bits 4:3 – SOURCEDIV[1:0] Select Source Division for PLL
This bit field divides the source frequency before being used as input to the PLL.

Value	Name	Description
0x0	NONE	No division. Nominal source frequency 2.5 to 5.5 MHz
0x1	DIV2	Divide by 2. Nominal source frequency 5 to 11 MHz
0x2	DIV4	Divide by 4. Nominal source frequency 10 to 22 MHz
0x3	DIV6	Divide by 6. Nominal source frequency 15 to 33 MHz

Bits 1:0 – MULFAC[1:0] Multiplication Factor
This bit field controls the multiplication factor for the Phased-Locked Loop (PLL).

Value	Name	Description
0x0	DISABLE	PLL is disabled
0x1	-	Reserved
0x2	8X	8x multiplication factor
0x3	16X	16x multiplication factor

2. 可在端口多路开关中设置 TCE 对应的寄存器，以将模块输出路由至不同端口。本例中选择端口 A，即默认端口。

```
PORTMUX.TCEROUTEA = 0x0;
```

图 3-6. TCE PORTMUX 寄存器

Bit	7	6	5	4	3	2	1	0
					TCE0[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bits 3:0 – TCE0[3:0] TCE0 Signals
This bit field controls the pin positions for TCE0 signals.

Value	Name	Description							
		WO0	WO1	WO2	WO3	WO4	WO5	WO6	WO7
0x0	PORTA	PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7
0x1	-	Reserved							
0x2	PORTC	PC0	PC1	PC2	PC3	-	-	-	-
0x3	PORTD	PD0	PD1	PD2	PD3	PD4	PD5	PD6	PD7
0x4	-	Reserved							
0x5	PORTF	PF0	PF1	PF2	PF3	PF4	PF5	-	-
0x6 - 0x7	-	Reserved							
0x8	PORTC2	PA0	PA1	PC0	PC1	PC2	PC3	-	-
0x9	PORTA2	PA2	PA3	PA4	PA5	PA6	PA7	-	-
Others	-	Reserved							

3. CTRLB 寄存器包含比较通道的使能位以及决定波形生成模式的位域。在本例中，使用通道 0、1、2 和 3 以及双斜率 PWM 模式。

```
TCE0.CTRLB |= (TCE_CMP0EN_bm | TCE_CMP1EN_bm | TCE_CMP2EN_bm | TCE_CMP3EN_bm);
```

```
TCE0.CTRLB |= TCE_WGMODE_DSBOTH_gc;
```

图 3-7. CTRLB 寄存器

Bit	7	6	5	4	3	2	1	0
	CMP3EN	CMP2EN	CMP1EN	CMP0EN	ALUPD	WGMODE[2:0]		
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 2:0 – WGMODE[2:0] Waveform Generation Mode

This bit field selects the Waveform Generation mode and controls the counting sequence of the counter, TOP value, UPDATE condition, interrupt condition, and the type of waveform generated.

No waveform generation is performed in the Normal mode of operation. For all other modes, the waveform generator output will only be directed to the port pins if the corresponding CMPnEN bit has been set. The port pin direction must be set as output.

Value	Name	Description
0x0	NORMAL	Normal operation mode
0x1	FRQ	Frequency mode
0x2	-	Reserved
0x3	SINGLESLOPE	Single-slope PWM mode
0x4	-	Reserved
0x5	DSTOP	Dual-slope PWM mode with overflow on TOP
0x6	DSBOTH	Dual-slope PWM mode with overflow on TOP and BOTTOM
0x7	DSBOTTOM	Dual-slope PWM mode with overflow on BOTTOM

4. 将 CTRLECLR 寄存器的 DIR 位设置为 1，以将定时器设置为对时钟节拍进行向下计数（递减）。DIR 位的默认值为 0。

```
TCE0.CTRLECLR = TCE_DIR_bm;
```

图 3-8. CTRLECLR 寄存器

Bit	7	6	5	4	3	2	1	0
					CMD[1:0]		LUPD	DIR
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

5. 使能在缩放模式下使用幅值寄存器和偏移量寄存器，并将缩放方法设置为底部缩放。占空比值从0%缩放至100%。此时，可向比较寄存器中写入一个介于 0 与 2 之间的小数值，而非向比较寄存器中写入绝对值。

   将小数值写入比较寄存器或比较缓冲寄存器之前，首先将该值乘以幅值，然后再加上偏移量。使用高分辨率功能，可通过两个额外的位将 PWM 的分辨率提升至四倍，或通过三个额外的位将 PWM 的分辨率提升至八倍。该功能在极低频率下非常有帮助。高分辨率功能可将时钟周期步长缩短至四分之一或八分之一。例如，对于频率为 20 MHz 且关闭高分辨率的时钟，单个时钟周期为 50 ns（T[s] = 1/f[Hz] = 1/20,000,000）。开启 4 倍高分辨率时，单个时钟周期为 12.5 ns。开启 8 倍高分辨率时，单个时钟周期为 6.25 ns。

```
TCE0.CTRLD = TCE_HREN_4X_gc | TCE_SCALE_bm | TCE_AMPEN_bm | TCE_SCALEMODE_BOTTOM_gc;
```

   下面说明了写入比较寄存器的值的缩放公式：

   EQ3.1：

$$Scaled_{CMP} = CMP_{frac} \times AMP + OFFSET$$

   使用中心缩放模式缩放后的值。

   EQ3.2：

$Scaled_{CMP} = CMP_{frac} \times AMP$

使用底部缩放模式缩放后的值。

EQ3.3：

$Scaled_{CMP} = CMP_{frac} \times AMP + OFFSET$

使用顶部缩放模式缩放后的值。

EQ3.4

$Scaled_{CMP} = CMP_{frac} \times AMP$

使用顶部-底部缩放模式缩放后的值。

CMP_{frac} 使用四种不同的值，如 AVR16EB16/20/28/32 AVR® EB Family Data Sheet 中的*表 23-6.Effective Compare Value in Scaled Mode* 所述。

图 3-9. CTRLD 寄存器

Bit	7	6	5	4	3	2	1	0
	HREN[1:0]		SCALEMODE[1:0]		AMPEN	SCALE		
Access	R/W	R/W	R/W	R/W	R/W	R/W		
Reset	0	0	0	0	0	0		

Bits 5:4 – SCALEMODE[1:0] Scale mode

This bit field defines how OFFSET is generated and used when writing to compare registers when scaling is enabled.

Value	Name	Description
0x0	CENTER	Offset generated so written compare values are scaled from center, 50% duty cycle
0x1	BOTTOM	Offset generated so written compare values are scaled from BOTTOM, 0% duty cycle
0x2	TOP	Offset generated so written compare values are scaled from TOP, 100% duty cycle
0x3	TOPBOTTOM	Written compare values <50% are scaled from TOP and written compare values>50% are scaled from BOTTOM. 0% values give compare value 0 and 100% values give compare value equal to TOP.

6. PER 为周期寄存器的缓冲区， EQ3.5 用于设置 PWM 信号的频率：

$$f_{DS\ PWM}(Hz) = \frac{N \times f_{CLK}(Hz)}{2 \times TCE_{prescaler} \times TCE_{period}}$$

N can have the values 1， 4 or 8， depending on the resolution feature

If N = 1， the high resolution feature is turned off

If N = 4， the resolution is increased by 4

If N = 8， the resolution is increased by 8

在本例中，分辨率提升至四倍。因此，使用 EQ3.6 计算写入 PER 寄存器的值：

$$TCE_{period} = \frac{4 \times f_{CLK}(Hz)}{2 \times TCE_{prescaler} \times f_{DS\ PWM}(Hz)} = \frac{4 \times 20000000}{2 \times 1 \times 10000} \cong 4000 = 0xFA0$$

```
TCE0.PER  = 0xFA0;
```

7. 使用比较寄存器设置占空比。由于需要 20%、40%、60%和 80%的占空比，因此比较寄存器中的值分别为周期寄存器中的值的 20%、40%、60%和 80%。当使用缓冲寄存器向比较寄存器中写入小数值时，这些值会在运行时被改写。为了区分在 CMP 寄存器中写入 0 到 PER 范围内的值与写入小数值的差异，可以在初始化时将这些值设置为 0。例如，在未使能缩放模式的情况下，为了实现 20%占空比的

PWM，如果给定 PER 为 0xFA0（相当于 10 kHz 频率或 100 µs 周期）并且开启 4 倍高分辨率，则写入 CMP0 中的值为 0x320。在使能缩放模式的情况下，为了实现 20%占空比的 PWM，写入 CMP0 中的值为 0x1999——通过硬件使用 PER、AMPLITUDE 和 OFFSET 寄存器的值计算占空比。

```
TCE0.CMP0 = 0x320;
```

```
TCE0.CMP1 = 0x640;
```

```
TCE0.CMP2 = 0x960;
```

```
TCE0.CMP3 = 0xC80;
```

8. 通过将所需值写入 AMP 寄存器来设置幅值。

```
TCE0.AMP = 0x8000;
```

9. 通过将所需值写入 OFFSET 寄存器来设置偏移量。

```
TCE0.OFFSET = 0x00;
```

10. 通过更改 CTRLA 寄存器中的 CLKSEL 位域将预分频值设置为 1。要启动计数器，用户必须将同一寄存器中的使能位置 1。

```
TCE0.CTRLA = TCE_CLKSEL_DIV1_gc;
```

```
TCE0.CTRLA |= TCE_ENABLE_bm;
```

图 3-10. CTRLA 寄存器

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY				CLKSEL[2:0]			ENABLE
Access	R/W				R/W	R/W	R/W	R/W
Reset	0				0	0	0	0

Bits 3:1 – CLKSEL[2:0] Clock Select

These bits select the clock frequency for the timer/counter.

Value	Name	Description
0x0	DIV1	$f_{TCE} = f_{CLK\_PER}$(no prescaling)
0x1	DIV2	$f_{TCE} = f_{CLK\_PER}/2$
0x2	DIV4	$f_{TCE} = f_{CLK\_PER}/4$
0x3	DIV8	$f_{TCE} = f_{CLK\_PER}/8$
0x4	DIV16	$f_{TCE} = f_{CLK\_PER}/16$
0x5	DIV64	$f_{TCE} = f_{CLK\_PER}/64$
0x6	DIV256	$f_{TCE} = f_{CLK\_PER}/256$
0x7	DIV1024	$f_{TCE} = f_{CLK\_PER}/1024$

11. 然后，通过向端口的方向寄存器中的相应位写入 1 将端口 A 的引脚 0-3（PA0-3）设置为输出。

```
PORTA.DIRSET = PIN0_bm | PIN1_bm | PIN2_bm | PIN3_bm;
```

12. 添加另一个引脚作为输出，以便在幅值和比较值发生变化时进行翻转。这样，用户就会观察到变化。引脚在运行时发生翻转。

```
PORTD.DIRSET = PIN5_bm;
```

13. 要查看占空比缩放比例，用户必须在运行时更改幅值寄存器的值。向幅值寄存器中写入新值后，用户必须将旧值写入比较寄存器中。这是因为只有向比较寄存器中写入新值后，才会更新占空比缩放比例。

3.2. MCC Melody 实现

要使用 MPLAB 代码配置器 Melody（即 MCC Melody，不支持 MCC Classic）生成该项目，请按照以下步骤操作：

1. 为 AVR16EB32 新建一个 MPLAB X IDE 项目。
2. 从工具栏中打开 MCC（有关安装 MCC 插件的更多信息，请单击此处）。
3. 在 MCC 内容管理器向导中选择 **MCC Melody**，然后单击 **Finish**（完成）。
4. 转至 *Project Resources > System > CLKCTRL*（项目资源 > 系统 > CLKCTRL）
   - 禁用预分频器使能按钮
   - 时钟选择：PLL 振荡器
   - PLL 倍频因子：16 倍频
   - PLL 源分频：DIV4
   - 预分频器 B 分频 PDIVB：DIV4
5. 从 *Device Resources > Drivers > Timer*（器件资源 > 驱动程序 > 定时器）中添加 TCE 模块，然后进行以下配置：
   - 模块使能：默认必须使能。如果未使能，请切换按钮（使能后会变成蓝色）。
   - 时钟选择：系统时钟（默认情况下，分频值必须为 1——系统时钟）
   - 波形生成模式：顶部和底部溢出的双斜率 PWM 模式（DSBOTH）
   - 请求的周期[s]： 0.0001
   - 占空比 0 [%]：20
   - 占空比 1 [%]：40
   - 占空比 2 [%]：60
   - 占空比 3 [%]：80
   - 波形输出 n：勾选波形输出 0、1、2、3 的使能列中的框
   - 占空比高分辨率：分辨率提升至 4 倍
   - 缩放模式：CMP 值从底部（0%占空比）开始缩放
   - 缩放写入寄存器：小数
   - 幅值控制使能：切换按钮（使能后会变成蓝色）
   - 幅值：1
6. 在 **Pin Grid View**（引脚网格视图）选项卡中检查端口 A 上的 TCE_WO[0-3]引脚是否锁定为输出。当勾选波形输出 n 的使能列中的框时，也会锁定引脚。要更改端口，请单击 **Pin Grid View** 中另一个端口的引脚。选择端口 D 的引脚 5 作为输出，然后翻转该引脚以查看幅值和比较值的变化。
7. 在 **Project Resources**（项目资源）选项卡中单击 **Generate**（生成）按钮，以便 MCC 生成所有指定的驱动程序和配置。
8. 编辑 `main.c` 文件，如下所示。
   在包含文件部分添加以下内容：

```
#include "mcc_generated_files/system/system.h"
#include <util/delay.h>
```

添加宏定义：

```
/* 计算周期寄存器、比较寄存器、幅值寄存器和偏移量寄存器的值 */
#define DUTY_CYCLE_20_PERCENT              (0x1999)
#define DUTY_CYCLE_40_PERCENT              (0x3333)
#define DUTY_CYCLE_60_PERCENT              (0x4CCC)
#define DUTY_CYCLE_80_PERCENT              (0x660C)
```

```
#define AMPLITUDE_MAX_DCY_50_PERCENT      (0x4000)
#define AMPLITUDE_MAX_DCY_75_PERCENT      (0x6000)
#define AMPLITUDE_MAX_DCY_100_PERCENT     (0x8000)
#define AMPLITUDE_MAX_DCY_150_PERCENT     (0xC000)
```

添加函数：

```
void Amplitude_Value_Set(uint16_t value)
{
    /* 设置幅值 */
    TCE0_AmplitudeSet(value);

    /* 重写 CMPBUF 寄存器中的值，以使占空比基于新的幅值进行缩放 */
    TCE0_CompareAllChannelsBufferedSet(DUTY_CYCLE_20_PERCENT, DUTY_CYCLE_40_PERCENT,
                                       DUTY_CYCLE_60_PERCENT, DUTY_CYCLE_80_PERCENT);

    /* 翻转 PD5 引脚以获悉幅值何时发生变化 */
    IO_PD5_Toggle();
}
```

编辑主函数：

```
int main(void)
{
    SYSTEM_Initialize();

    while(1)
    {
        Amplitude_Value_Set(AMPLITUDE_MAX_DCY_50_PERCENT);
        _delay_ms(10);
        Amplitude_Value_Set(AMPLITUDE_MAX_DCY_75_PERCENT);
        _delay_ms(10);
        Amplitude_Value_Set(AMPLITUDE_MAX_DCY_100_PERCENT);
        _delay_ms(10);
        Amplitude_Value_Set(AMPLITUDE_MAX_DCY_150_PERCENT);
        _delay_ms(10);
    }
}
```

9. 现在即可从 MPLAB X IDE 中编译并运行该项目。运行时，每 10 ms 修改一次比较寄存器的缩放值，同时也将相应地修改占空比的范围。

3.3. 结果

下图给出了逻辑分析器捕捉到的一些波形，以展示四个 PWM 信号的外观以及如何使用幅值和偏移量创建占空比缩放比例。

图 3-11. 占空比范围缩放至周期的 0-100%

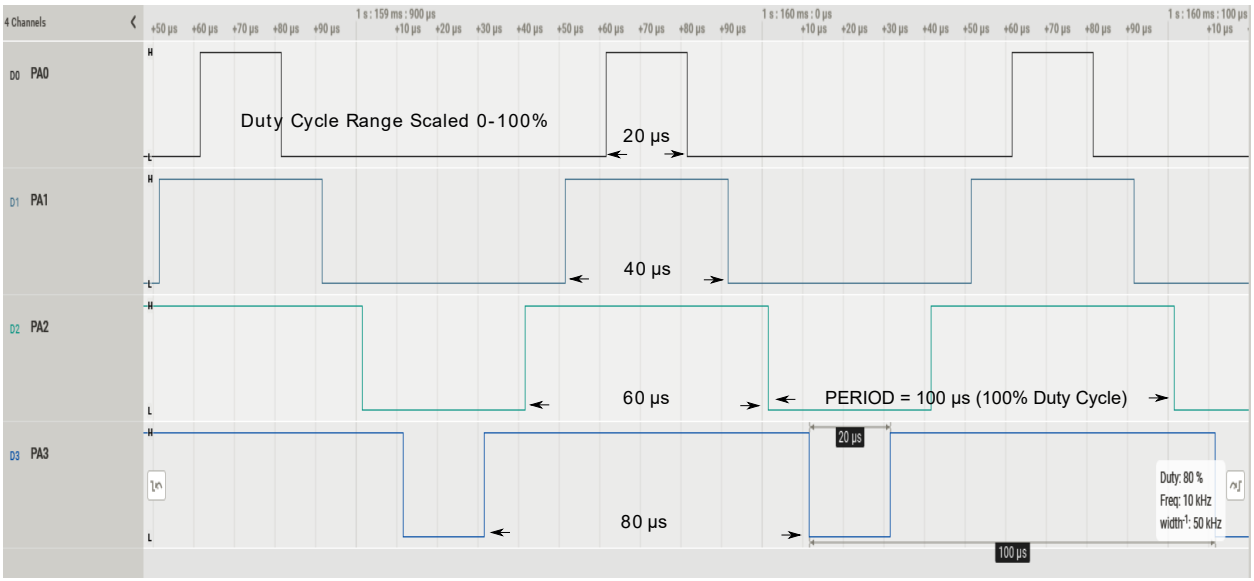


图 3-12. 占空比范围缩放至周期的0-50%

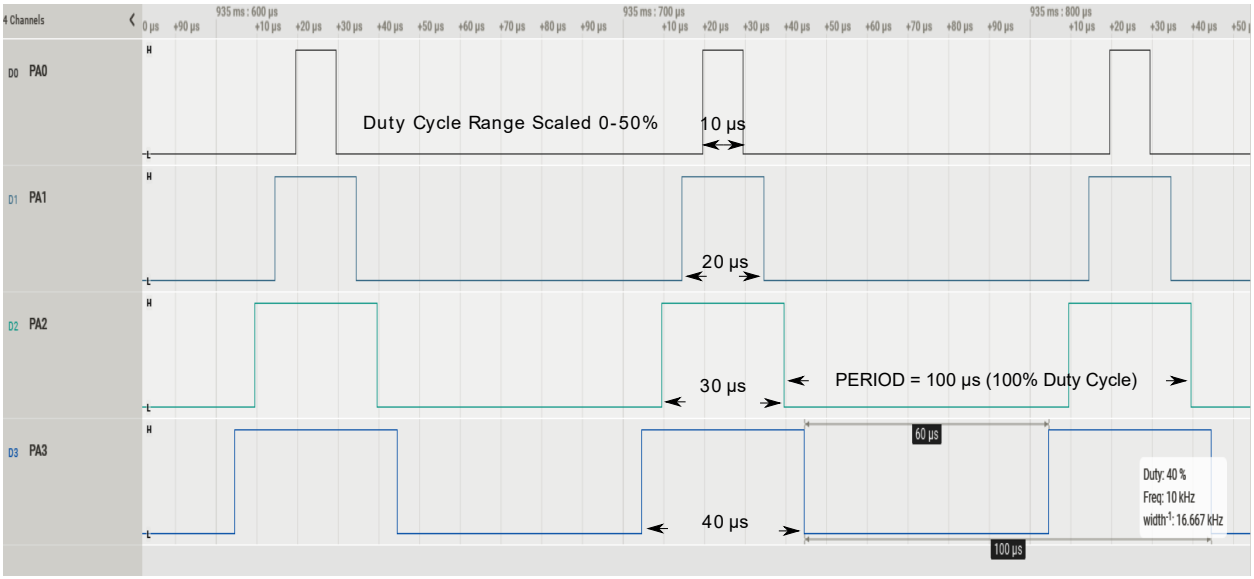
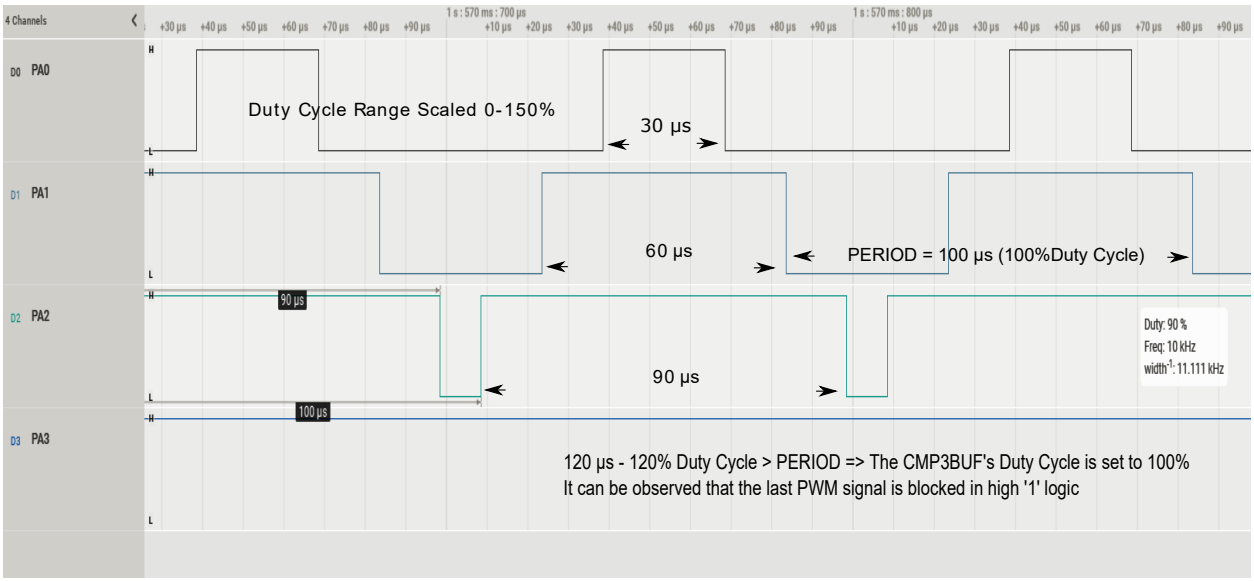


图 3-13. 占空比范围缩放至周期的 0-150%



如需本部分所述功能的 AVR16EB32 裸机代码示例，可单击此处：

如需本部分所述功能的使用 MCC Melody 生成的 AVR16EB32 代码示例，可单击此处：

4. 使用 WEX 实现模式生成

波形扩展的基本用例是为周期性变化的输出信号设置模式。如果用户希望使用 WEX 改写端口引脚，该模式生成模式将非常有帮助。使用 WEX 的模式生成模式时，将控制 TCE 的输出引脚。用户必须使能模式生成模式，并设置包含适当代码的所需模式。

下面提供了一个包含初始化的基本示例。程序将使用软件延时每 25 µs 翻转一次引脚（互补模式），共翻转十次。再经过 250 µs 后将生成阶梯模式，每级阶梯间隔 5 µs 延时。再经过 250 µs 延时后，重复上述过程。

在初始化波形扩展之前，必须通过设置方向寄存器的相应位将引脚配置为输出，具体如下文所述。本例中选择端口 A 的引脚 0-7（PA0-7）。最好使用逻辑分析器或八个 LED 观察模式信号输出。

下面两个小节介绍如何配置 WEX 外设以实现所需的行为。第一小节介绍如何使用裸机代码配置 WEX。第二小节介绍如何使用 MCC Melody 配置 WEX。最后一个小节提供结果。

4.1. 裸机实现

1. 在 CTRLA 寄存器中使能模式生成模式将改写 TCE 生成的波形。

```
WEX0.CTRLA = WEX_PGM_bm;
```

图 4-1. CTRLA 寄存器

Bit	7	6	5	4	3	2	1	0
	PGM	INMX[2:0]			DTI3EN	DTI2EN	DTI1EN	DTI0EN
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – PGM Pattern Generation Mode
Writing this bit to '1' enables the Pattern Generation mode. In Pattern Generation mode, the dead-time buffer registers are used for storing the pattern so these buffers are not available in this mode.

2. 在模式生成模式下，每个信号输出必须先使能才能改写。每个信号与 PGMOVR 寄存器中的位存在一一对应关系。

```
WEX0.PGMOVR  = WEX_PGMOVR0_bm | WEX_PGMOVR1_bm | WEX_PGMOVR2_bm | WEX_PGMOVR3_bm |
                        WEX_PGMOVR4_bm | WEX_PGMOVR5_bm | WEX_PGMOVR6_bm | WEX_PGMOVR7_bm;
```

图 4-2. 模式生成改写寄存器

Bit	7	6	5	4	3	2	1	0
	PGMOVR7	PGMOVR6	PGMOVR5	PGMOVR4	PGMOVR3	PGMOVR2	PGMOVR1	PGMOVR0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 0, 1, 2, 3, 4, 5, 6, 7 – PGMOVR Pattern Generation Override
This register holds override enable PGM mode. If a bit is set to '1' the corresponding bit in PGMOUT specifies the value that will override the output from the SWAP unit.
The available configuration for each bit n in this bit field is shown in the table below:

Value	Description
0	The pin n (Pxn) output is not overwritten
1	The pin n (Pxn) output is overwritten with the value in PGMOUT register

3. PGMOUT 寄存器保存每个信号的模式。值得一提的是，每个信号与 PGMOUT 寄存器中的位存在一一对应关系。例如，要实现 0xAA 模式，需在 PGMOUT 寄存器中将 bit 1、3、5 和 7 设置为逻辑高电平（1）：

```
WEX0.PGMOUT = WEX_PGMOUT1_bm | WEX_PGMOUT3_bm | WEX_PGMOUT5_bm | WEX_PGMOUT7_bm;
```

图 4-3. 模式生成输出寄存器

Bit	7	6	5	4	3	2	1	0
	PGMOUT7	PGMOUT6	PGMOUT5	PGMOUT4	PGMOUT3	PGMOUT2	PGMOUT1	PGMOUT0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bits 0, 1, 2, 3, 4, 5, 6, 7 – PGMOUT Pattern Generation Output

This register holds the Override Value to take effect when Pattern Generation.

The available configuration for each bit n in this bit field is shown in the table below:

Value	Description
0	Waveform output value for Pin n (Pxn) is driven low
1	Waveform output value for Pin n (Pxn) is driven high

4. 将 WEX 配置为模式生成模式后，用户可使用 PGMOUT 寄存器翻转每个信号的状态。

```
WEX0.PGMOUT = ~WEX0.PGMOUT;
```

5. 添加延时以观察何时发生翻转。

```
_delay_us(25);
```

6. 通过向端口的方向寄存器中的相应位写入 1 将端口 A 的引脚 0-7（PA0-7）设置为输出。这些 GPIO 仅配置用于获取可见输出。

```
PORTA.DIRSET = PIN0_bm | PIN1_bm | PIN2_bm | PIN3_bm |
                        PIN4_bm | PIN5_bm | PIN6_bm | PIN7_bm;
```

7. 在 PORTMUX 中进行设置以选择默认的端口 A 引脚。

```
PORTMUX.TCEROUTEA = 0x0;
```

图 4-4. WEX PORTMUX 寄存器

Value	Name	Description							
		WO0	WO1	WO2	WO3	WO4	WO5	WO6	WO7
0x0	PORTA	PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7
0x1	-	Reserved							
0x2	PORTC	PC0	PC1	PC2	PC3	-	-	-	-
0x3	PORTD	PD0	PD1	PD2	PD3	PD4	PD5	PD6	PD7

8. 通过将输出改写使能寄存器中的相应位置 1 使能 WEX 的输出，以便使能 PORTA 引脚改写。

```
WEX0.OUTOVEN = WEX_OUTOVEN0_bm | WEX_OUTOVEN1_bm | WEX_OUTOVEN2_bm | WEX_OUTOVEN3_bm |
                        WEX_OUTOVEN4_bm | WEX_OUTOVEN5_bm | WEX_OUTOVEN6_bm |
WEX_OUTOVEN7_bm;
```

图 4-5. 输出改写使能寄存器

Bit	7	6	5	4	3	2	1	0
	OUTOVEN7	OUTOVEN6	OUTOVEN5	OUTOVEN4	OUTOVEN3	OUTOVEN2	OUTOVEN1	OUTOVEN0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

9. 本例的最后一步是使能 TCE 模块，因为需要通过从 TCE 传递至 WEX 的源时钟来更新定义的模式。

```
TCE0.CTRLA = TCE_ENABLE_bm;
```

4.2. MCC Melody 实现

要使用 MPLAB 代码配置器 Melody（即 MCC Melody，不支持 MCC Classic）生成该项目，请按照以下步骤操作：

1. 为 AVR16EB32 新建一个 MPLAB X IDE 项目。
2. 从工具栏中打开 MCC（有关安装 MCC 插件的更多信息，请单击此处）。
3. 在 MCC 内容管理器向导中选择 **MCC Melody**，然后单击 **Finish**。
4. 转至 *Device Resources > Drivers > Timer*。添加 TCE 模块，然后进行以下配置：
   - 模块使能：切换按钮（使能后会变成蓝色）
5. 转至 *Device Resources > Drivers*（器件资源 > 驱动程序）。添加 WEX 模块，然后进行以下配置：
   - 输入矩阵：直接
   - 更新源：TCE（输出信号的更新条件将基于 TCE 模块）
   - 改写设置：勾选波形输出[0-7]的输出使能列中的所有框
   - 模式生成模式使能：切换按钮（使能后会变成蓝色）
   - 模式生成操作：勾选改写使能列中的所有框，并为每个输出设置所需状态（低电平或高电平）以设置模式
6. 在 **Pin Grid View** 选项卡中检查端口 A 上的 WEX_WO[0-7]引脚是否锁定为输出。当勾选改写设置的输出使能列中的框后，引脚将受到保护。要更改端口，请单击 **Pin Grid View** 中另一个端口的引脚。
7. 在 **Project Resources** 选项卡中单击 **Generate** 按钮，以便 MCC 生成所有指定的驱动程序和配置。
8. 编辑 `main.c` 文件，如下所示。
   在包含文件部分添加以下内容：

```
#include "mcc_generated_files/system/system.h"
#include <util/delay.h>
```

添加宏定义：

```
/* 写入 PGMOUT 寄存器的模式 */
#define COMPLEMENTARY_PATTERN           (WEX_PGMOUT6_bm | WEX_PGMOUT4_bm |
                                         WEX_PGMOUT2_bm | WEX_PGMOUT0_bm)
#define STAIRCASE0_PATTERN              (WEX_PGMOUT0_bm)
#define STAIRCASE1_PATTERN              (WEX_PGMOUT1_bm)
#define STAIRCASE2_PATTERN              (WEX_PGMOUT2_bm)
#define STAIRCASE3_PATTERN              (WEX_PGMOUT3_bm)
#define STAIRCASE4_PATTERN              (WEX_PGMOUT4_bm)
#define STAIRCASE5_PATTERN              (WEX_PGMOUT5_bm)
#define STAIRCASE6_PATTERN              (WEX_PGMOUT6_bm)
#define STAIRCASE7_PATTERN              (WEX_PGMOUT7_bm)
#define PATTERN_RESET                   (0x00)
```

添加函数：

```
void Complementary_Pattern_Set(void)
{
    WEX0_PatternGenerationOutputSet(COMPLEMENTARY_PATTERN);
    uint8_t complementary_pattern = COMPLEMENTARY_PATTERN;
    _delay_us(25);

    /* 互补信号模式 */
    for(uint8_t i = 0; i < 9; i++)
    {
        /* 每个步长 complementary_pattern 变量都发生变化 */
        complementary_pattern = ~complementary_pattern;

        /* 翻转每个 WEX 的输出的模式 */
        WEX0_PatternGenerationOutputSet(complementary_pattern);

        /* 添加软件延时以便观察翻转 */
        _delay_us(25);
    }

    /* 将所有信号置于逻辑低电平 0，然后等待 250 µs 以观察从其中一个
     * 互补模式切换到阶梯模式 */
    WEX0_PatternGenerationOutputSet(PATTERN_RESET);
}

void Stairs_Pattern_Set(void)
{
    /* 每个信号按照递增顺序依次从低电平变为高电平，
     * 以生成阶梯递增模式 */
    WEX0_PatternGenerationOutputSet(STAIRCASE7_PATTERN);

    /* 添加软件延时以便观察递增 */
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE6_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE5_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE4_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE3_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE2_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE1_PATTERN);
    _delay_us(5);

    /* 每个信号按照递减顺序依次从低电平变为高电平，
     * 以生成阶梯递减模式 */
    WEX0_PatternGenerationOutputSet(STAIRCASE0_PATTERN);

    /* 添加软件延时以便观察递减 */
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE1_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE2_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE3_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE4_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE5_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE6_PATTERN);
    _delay_us(5);
    WEX0_PatternGenerationOutputSet(STAIRCASE7_PATTERN);
    _delay_us(5);

    /* 将所有信号置于逻辑低电平 0，然后等待 250 µs 以观察从
     * 阶梯模式切换到其中一个互补模式 */
    WEX0_PatternGenerationOutputSet(PATTERN_RESET);
}
```

编辑主函数：

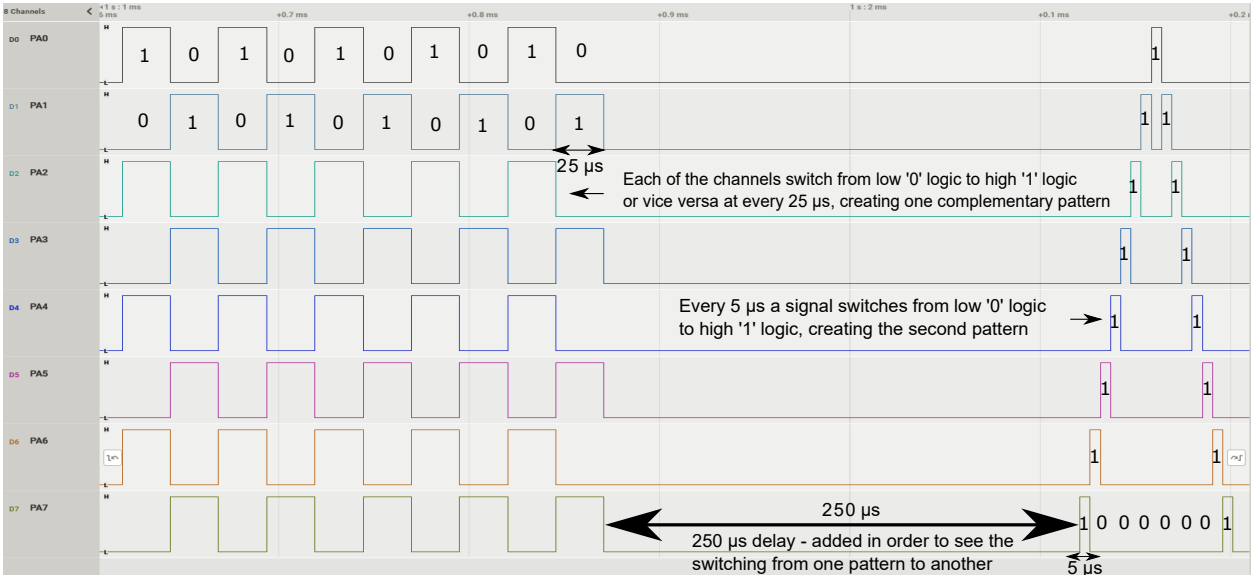
```
int main(void)
{
    SYSTEM_Initialize();

    while(1)
    {
        Complementary_Pattern_Set();
        _delay_us(250);
        Stairs_Pattern_Set();
        _delay_us(250);
    }
}
```

9. 现在即可使用 MPLAB X IDE 编译并运行该项目。

4.3. 结果

下图给出了逻辑分析器捕捉到的一些波形，以展示所生成模式的外观。

图 4-6. 从互补模式切换为阶梯模式



如需本部分所述功能的 AVR16EB32 裸机代码示例，可单击此处：

Click to view code examples on MPLAB DISCOVER

如需本部分所述功能的使用 MCC Melody 生成的 AVR16EB32 代码示例，可单击此处：

Click to view code examples on MPLAB DISCOVER

5. 搭配使用 TCE 和 WEX 生成八个 PWM 信号

TCE 可以在所有四个通道上生成精确且多功能的 PWM 信号，并产生不重叠的互补波形。

WEX 可配置为将 TCE 提供的四个比较通道扩展至最多八个能够生成 PWM 信号的通道。

下面举例说明了如何设置 TCE 和 WEX 实例，以使用缓冲方案生成八个可变占空比的 20 kHz 互补 PWM 信号——请参考搭配使用 TCE 和 WEX 生成 PWM 信号部分。由于添加了死区，因此信号成对出现且不会重叠，该功能对于电机控制来说至关重要，其可有效避免晶体管开关引起的直通电流。比较寄存器将在每个通道的比较匹配中断期间进行更新。在本例中，还重点介绍了故障保护功能。当触发软件事件时，将仿真故障事件，并且所有信号都会驱动为低电平，每 1 ms 发生一次。为此，将 WEX 配置为进行故障事件检测，并将 EVSYS 配置为生成软件事件。

下面两个小节介绍如何配置 TCE 和 WEX 外设以实现所需的行为。第一小节介绍如何使用裸机代码配置 TCE 和 WEX。第二小节介绍如何使用 MCC Melody 配置 TCE 和 WEX。最后一个小节提供结果。

5.1. 裸机实现

1. 时钟外设必须以 20 MHz 的频率运行。默认时钟速度为 3.33 MHz，预分频器默认采用六分频。该代码示例需禁止预分频器并使时钟达到最高速度。

```
_PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, CLKCTRL.MCLKCTRLB & ~CLKCTRL_PEN_bm);
```

图 5-1. CLKCTRL 的 MCLKCTRLB 寄存器

Bit	7	6	5	4	3	2	1	0
			PBDIV	PDIV[3:0]				PEN
Access			R/W	R/W	R/W	R/W	R/W	R/W
Reset			0	1	0	0	0	1

Bit 0 – PEN Prescaler Enable

This bit controls whether the Main Clock (CLK_MAIN) prescaler is enabled or not.

Value	Description
0	The CLK_MAIN prescaler is disabled
1	The CLK_MAIN prescaler is enabled and the division ratio is controlled by the Prescaler Division (PDIV) bit field

2. 可在端口多路开关中设置 TCE 和 WEX 对应的寄存器，以将模块输出路由至不同端口。本例中选择端口 A，即默认端口。

```
PORTMUX.TCEROUTEA = 0x0;
```

图 5-2. TCE 和 WEX 的 PORTMUX 控制

Bit	7	6	5	4	3	2	1	0
					TCE0[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

Bits 3:0 – TCE0[3:0] TCE0 Signals
This bit field controls the pin positions for TCE0 signals.

Value	Name	Description							
		WO0	WO1	WO2	WO3	WO4	WO5	WO6	WO7
0x0	PORTA	PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7
0x1	-	Reserved							
0x2	PORTC	PC0	PC1	PC2	PC3	-	-	-	-
0x3	PORTD	PD0	PD1	PD2	PD3	PD4	PD5	PD6	PD7
0x4	-	Reserved							
0x5	PORTF	PF0	PF1	PF2	PF3	PF4	PF5	-	-
0x6 - 0x7	-	Reserved							
0x8	PORTC2	PA0	PA1	PC0	PC1	PC2	PC3	-	-
0x9	PORTA2	PA2	PA3	PA4	PA5	PA6	PA7	-	-
Others	-	Reserved							

3. TCE 的 CTRLB 寄存器包含比较通道的使能位以及决定波形生成模式的位域。在本例中，使用通道 0、1、2 和 3 以及单斜率 PWM 模式。

```
TCE0.CTRLB |= (TCE_CMP0EN_bm | TCE_CMP1EN_bm | TCE_CMP2EN_bm | TCE_CMP3EN_bm);
```

```
TCE0.CTRLB |= TCE_WGMODE_SINGLESLOPE_gc;
```

图 5-3. TCE 的 CTRLB 寄存器

Bit	7	6	5	4	3	2	1	0
	CMP3EN	CMP2EN	CMP1EN	CMP0EN	ALUPD	WGMODE[2:0]		
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

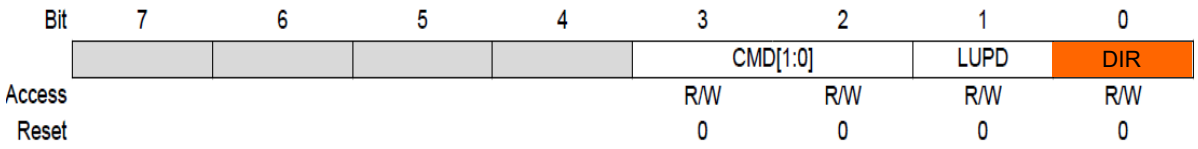
Bits 2:0 – WGMODE[2:0] Waveform Generation Mode
This bit field selects the Waveform Generation mode and controls the counting sequence of the counter, TOP value, UPDATE condition, interrupt condition, and the type of waveform generated.
No waveform generation is performed in the Normal mode of operation. For all other modes, the waveform generator output will only be directed to the port pins if the corresponding CMPnEN bit has been set. The port pin direction must be set as output.

Value	Name	Description
0x0	NORMAL	Normal operation mode
0x1	FRQ	Frequency mode
0x2	-	Reserved
0x3	SINGLESLOPE	Single-slope PWM mode
0x4	-	Reserved
0x5	DSTOP	Dual-slope PWM mode with overflow on TOP
0x6	DSBOTH	Dual-slope PWM mode with overflow on TOP and BOTTOM
0x7	DSBOTTOM	Dual-slope PWM mode with overflow on BOTTOM

4. 将 TCE 的 CTRLECLR 寄存器的 DIR 位设置为 1，以将定时器设置为对时钟节拍进行向下计数（递减）。DIR 位的默认值为 0。

```
TCE0.CTRLECLR = TCE_DIR_bm;
```

图 5-4. TCE 的 CTRLECLR 寄存器



5. PER 为 TCE 周期寄存器的缓冲区， EQ5.1 用于设置 PWM 信号的频率：

$$f_{SS\ PWM}(Hz) = \frac{f_{CLK}(Hz)}{TCE_{prescaler} \times (TCE_{period} + 1)}$$

假设本例的目标值如下：

$$TCE_{period} = \frac{f_{CLK}(Hz)}{TCE_{prescaler} \times f_{SS\,PWM}(Hz)} = \frac{20000000}{1 \times 20000} \cong 1000 = 0x3E8$$

```
/* PER 寄存器始终设置为所需值 - 1，1000 - 1=999，所需值对应于 50 µs 周期 */
TCE0.PER  = 0x3E7;
```

6. 使用缓冲功能更新比较寄存器以设置占空比。可将比较寄存器中的值设置为随机占空比，例如 20%、40%、60%和 80%。但在运行时，发生第一次中断后更新 CMP 寄存器（使用其缓冲区）时会改写这些值。因此，我们可将这些值设置为 0。

```
TCE0.CMP0 = 0x00;
```

```
TCE0.CMP1 = 0x00;
```

```
TCE0.CMP2 = 0x00;
```

```
TCE0.CMP3 = 0x00;
```

7. 启动定时器后，每次比较寄存器的值与计数器值匹配时都会增加占空比。占空比的范围在 0-100%之间。在 ISR 期间，每个比较通道都会发生这种情况。比较 0 通道的 ISR 程序：

```
ISR(TCE0_CMP0_vect)
{
    /* 清零中断标志 */
    TCE0.INTFLAGS = TCE_CMP0_bm;

    static uint16_t duty_cycle = 0;

    /* 中断中的占空比更新 */
    duty_cycle += 5;
    if(duty_cycle >= MAX_DUTY_CYCLE)
    duty_cycle = 0;
    TCE0.CMP0BUF = duty_cycle;
}
```

8. 为 TCE 的每个比较寄存器允许比较匹配中断。

```
TCE0.INTCTRL = (TCE_CMP0_bm | TCE_CMP1_bm | TCE_CMP2_bm | TCE_CMP3_bm);
```

图 5-5. TCE 的 INTCTRL 寄存器

Bit	7	6	5	4	3	2	1	0
	CMP3	CMP2	CMP1	CMP0				OVF
Access	R/W	R/W	R/W	R/W				R/W
Reset	0	0	0	0				0

Bits 4, 5, 6, 7 – CMPn Compare Channel n Interrupt Enable
Writing the CMPn bit to '1' enables the interrupt from Compare Channel n.

Bit 0 – OVF Timer Overflow/Underflow Interrupt Enable
Writing the OVF bit to '1' enables the overflow/underflow interrupt.

9. 将 TCE 计数器的初始值设置为 0。

```
TCE0.CNT = 0x00;
```

10. 通过更改 CTRLA 寄存器中的 CLKSEL 位域将预分频值设置为 1。要启动计数器，用户必须将同一寄存器中的使能位置 1。

```
TCE0.CTRLA = TCE_CLKSEL_DIV1_gc;
```

配置 TCE 的最后一步是使能该模块。

```
TCE0.CTRLA |= TCE_ENABLE_bm;
```

图 5-6. TCE 的 CTRLA 寄存器

Bit	7	6	5	4	3	2	1	0
	RUNSTDBY				CLKSEL[2:0]			ENABLE
Access	R/W				R/W	R/W	R/W	R/W
Reset	0				0	0	0	0

Bits 3:1 – CLKSEL[2:0] Clock Select
These bits select the clock frequency for the timer/counter.

Value	Name	Description
0x0	DIV1	$f_{TCE} = f_{CLK\_PER}$ (no prescaling)
0x1	DIV2	$f_{TCE} = f_{CLK\_PER}/2$
0x2	DIV4	$f_{TCE} = f_{CLK\_PER}/4$
0x3	DIV8	$f_{TCE} = f_{CLK\_PER}/8$
0x4	DIV16	$f_{TCE} = f_{CLK\_PER}/16$
0x5	DIV64	$f_{TCE} = f_{CLK\_PER}/64$
0x6	DIV256	$f_{TCE} = f_{CLK\_PER}/256$
0x7	DIV1024	$f_{TCE} = f_{CLK\_PER}/1024$

11. 将 WEX 的输出路由矩阵设置为直接模式，并为每对互补信号使能死区插入。完成这些设置后，WEX 基于 TCE 生成的四个 PWM 信号来生成八个互补信号。

```
WEX0.CTRLA = WEX_INMX_DIRECT_gc | WEX_DTI0EN_bm | WEX_DTI1EN_bm |
                    WEX_DTI2EN_bm | WEX_DTI3EN_bm;
```

图 5-7. WEX 的 CTRLA 寄存器

Bit	7	6	5	4	3	2	1	0
	PGM	INMX[2:0]			DTI3EN	DTI2EN	DTI1EN	DTI0EN
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Bit 7 – PGM Pattern Generation Mode
Writing this bit to '1' enables the Pattern Generation mode. In Pattern Generation mode, the dead-time buffer registers are used for storing the pattern so these buffers are not available in this mode.

Bits 6:4 – INMX[2:0] Input Matrix
This bit field defines the matrix routing of the timer/counters Waveform Generation outputs to the WEX internal module. Refer to table Table 24-1 for more information.

Value	Name	Description
0x0	DIRECT	Direct from TCE0
0x1	-	Reserved
0x2	CWCMA	Common Waveform Channel Mode A. Waveform output (WO) on a single Pulse-Width Modulation (PWM) channel.
0x3	CWCMB	Common Waveform Channel Mode B. WO on two PWM channels.

Bits 0, 1, 2, 3 – DTIEN Dead-Time Insertion CMPn Enable

Value	Name	Description
0x0	DISABLED	Dead-time not inserted
0x1	ENABLED	Dead-time inserted

12. 使用死区寄存器设置所需的死区值。死区以外设时钟节拍来测量。下面的公式说明了如何计算死区：

```
WEX0.DTLS = 0x03;
```

```
WEX0.DTHS = 0x05;
```

图 5-8. WEX 的死区寄存器：下桥臂死区、上桥臂死区和上下桥臂死区

DTLS	7:0	DTLS[7:0]
DTHS	7:0	DTHS[7:0]
DTBOTH	7:0	DTBOTH[7:0]

若要在应用中实现对称死区，请使用最后一个寄存器。在本例中，单独使用下桥臂死区寄存器与上桥臂死区寄存器以创建非对称死区。
用于计算死区的公式：

EQ5.2

$$clock\_tick(ns) = \frac{1}{f_{CLK}(Hz)} \times TCE_{prescaler} \times 10^9$$

将时钟节拍转换为纳秒，例如：

$f_{clk}(Hz) = 20\ MHz$

$TCE_{prescaler} = 4$

$$desired\_dead\_time(ns) = 500$$

$$clock\_tick(ns) = \frac{1}{20000000} \times 4 \times 10^9 = 250$$

$$register\_value = dead\_time(ns) \div clock\_tick(ns) = 500 \div 250 = 2$$

13. 在 FAULTCTRL 寄存器中设置故障重启模式条件和故障信号驱动模式。

```
/* 将故障重启模式设置为锁存模式——只要故障条件有效，故障状态即保持激活 */
/* 要从锁存模式下的故障状态重启，用户必须使用软件故障清除命令 */
WEX0.FAULTCTRL = WEX_FDMODE_LATCHED_gc;
```

```
/* 当检测到故障时，将所有引脚驱动至逻辑低电平 0 */
WEX0.FAULTCTRL |= WEX_FDACT_LOW_gc;
```

图 5-9. WEX 的 FAULTCTRL 寄存器

Bit	7	6	5	4	3	2	1	0
	FDDBD					FDMODE	FDACT[1:0]	
Access	R/W					R/W	R/W	R/W
Reset	0					0	0	0

Bit 7 – FDDBD Fault Detection on Debug Break Detection

Value	Name	Description
0x0	FAULT	OCD Break request is treated as a fault if fault protection is enabled
0x1	IGNORE	OCD Break request will not trigger a fault

Bit 2 – FDMODE Fault Detection Restart Mode

Value	Name	Description
0x0	LATCHED	Latched mode. Output will remain in fault state until fault condition is no longer active and Fault Detection Flag Event Input (FDFEVx in INCTRL register) is cleared by software.
0x1	CBC	Cycle-by-cycle mode. Waveform output will remain in fault state until fault condition is no longer active.

Bits 1:0 – FDACT[1:0] Fault Detection Action

Value	Name	Description
0x0	NONE	None. Fault Protection Disabled
0x1	LOW	Drive all pins low
0x2	-	Reserved
0x3	CUSTOM	Drive all pins to setting defined by FAULTDRV and FAULTVAL

14. 使用 EVCTRLA 寄存器使能 WEX 的事件输入 A。

```
WEX0.EVCTRLA = WEX_FAULTEI_bm;
```

图 5-10. WEX 的 EVCTRLA 寄存器

Bit	7	6	5	4	3	2	1	0
				FILTER[2:0]			BLANK	FAULTEI
Access				R/W	R/W	R/W	R/W	R/W
Reset				0	0	0	0	0

Bit 0 – FAULTEI Fault Event Input Enable

This bit enables event input A as trigger for fault condition.

Note: Fault event inputs are not taken into account before Timer/Counting driving a waveform is enabled

15. 在 WEX 的 ISR 期间允许故障中断（故障检测）并清零故障标志。

```
WEX0.INTCTRL = WEX_FAULTDET_bm;
```

```
/* WEX 故障 ISR */
ISR(WEX0_FDFEVA_vect)
{
    /* 清零中断标志和故障事件标志 */
    WEX0.INTFLAGS = WEX_FDFEVA_bm | WEX_FAULTDET_bm;
}
```

图 5-11. WEX 的 INTCTRL 和 INTFLAGS 寄存器

Bit	7	6	5	4	3	2	1	0
								FAULTDET
Access								R/W
Reset								0

Bit 0 – FAULTDET Fault Detection Interrupt Enable

Writing this bit to '1' enables fault detection interrupt.

Bit	7	6	5	4	3	2	1	0
				FDFEVC	FDFEVB	FDFEVA		FAULTDET
Access				R/W	R/W	R/W		R/W
Reset				0	0	0		0

Bit 4 – FDFEVC Fault Detection Flag Event Input C

This bit is set on a fault detection on Event input C. The bit it cleared by writing a '1' to its bit location.

Bit 3 – FDFEVB Fault Detection Flag Event Input B

This bit is set on a fault detection on Event input B. The bit it cleared by writing a '1' to its bit location.

Bit 2 – FDFEVA Fault Detection Flag Event Input A

This bit is set on a fault detection on Event input A. The bit it cleared by writing a '1' to its bit location.

Bit 0 – FAULTDET Fault Detection Interrupt Flag

This bit is set on a positive edge of fault detection. The bit it cleared by writing a '1' to its bit location.

16. 配置事件系统（EVSYS）外设以在通道 0 上生成事件。将 WEX 配置为通道 0 的事件生成器的用户。使用软件事件生成故障。要停止故障，需关闭 EVSYS 的故障发生器通道并使用软件命令清除故障状态。

```
/* 将 WEX0 外设设置为通道 0 的事件生成器的用户，在本例中为软件事件 */
EVSYS.USERWEXA = EVSYS_USER_CHANNEL0_gc;
```

```
/* 创建软件故障，在主循环中重复以在逻辑分析器上进行查看。这是使用软件命令生成的事件 */
EVSYS.SWEVENTA = EVSYS_SWEVENTA_CH0_gc;
```

```
/* 使用软件命令清除故障条件 */
WEX0.CTRLC = WEX_CMD_FAULTCLR_gc;
```

17. 使用 OUTOVEN 寄存器使能 WEX 的输出以生成八个 PWM 信号。

```
/* 使能 WEX 的输出 */
WEX0.OUTOVEN = WEX_OUTOVEN0_bm | WEX_OUTOVEN1_bm | WEX_OUTOVEN2_bm | WEX_OUTOVEN3_bm |
               WEX_OUTOVEN4_bm | WEX_OUTOVEN5_bm | WEX_OUTOVEN6_bm | WEX_OUTOVEN7_bm;
```

图 5-12. WEX 的 OUTOVEN 寄存器

Bit	7	6	5	4	3	2	1	0
	OUTOVEN7	OUTOVEN6	OUTOVEN5	OUTOVEN4	OUTOVEN3	OUTOVEN2	OUTOVEN1	OUTOVEN0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

18. 然后，通过向端口的方向寄存器中的相应位写入 1 将端口 A 的引脚 0-7（PA0-7）设置为输出。

```
PORTA.DIRSET = PIN0_bm | PIN1_bm | PIN2_bm | PIN3_bm |
                         PIN4_bm | PIN5_bm | PIN6_bm | PIN7_bm;
```

19. 当所有模块配置完成后，允许全局中断。

```
/* 允许全局中断 */
    sei();
```

5.2. MCC Melody 实现

要使用 MPLAB 代码配置器（即 MCC Melody，不支持 MCC Classic）生成该项目，请按照以下步骤操作：

1. 为 AVR16EB32 新建一个 MPLAB X IDE 项目。
2. 从工具栏中打开 MCC（有关安装 MCC 插件的更多信息，请单击此处）。
3. 在 MCC 内容管理器向导中选择 **MCC Melody**，然后单击 **Finish**。
4. 转至 *Project Resources > System > Interrupt Manager*（项目资源 > 系统 > 中断管理器）。切换全局中断允许按钮。
5. 转至 *Project Resources > System > CLKCTRL*。禁用预分频器使能按钮。
6. 从 *Device Resources > Drivers > Timer* 中添加 TCE 模块，然后进行以下配置：
   - 使能定时器：可能默认使能。如果未使能，请切换按钮（使能后会变成蓝色）。
   - 时钟分频器：系统时钟（默认情况下，分频值应为 1——系统时钟）
   - 波形生成模式：单斜率 PWM 模式
   - 请求的周期[s]：0.00005
   - 波形输出 n：勾选波形输出 0、1、2、3 的使能列中的框
   - 产生 ISR：切换按钮（使能后会变成蓝色）
   - 允许比较 0 中断：切换按钮（使能后会变成蓝色）
   - 允许比较 1 中断：切换按钮（使能后会变成蓝色）

- 允许比较 2 中断：切换按钮（使能后会变成蓝色）
- 允许比较 3 中断：切换按钮（使能后会变成蓝色）

7. 从 *Device Resources > Drivers*（器件资源 > 驱动程序）中添加 WEX 模块，然后进行以下配置：
   - 输入矩阵：直接
   - 更新源：TCE（输出信号的更新条件将由 TCE 决定）
   - 改写设置：勾选波形输出[0-7]的输出使能列中的所有框
   - 死区插入通道 0 使能：切换按钮（使能后会变成蓝色）
   - 死区插入通道 1 使能：切换按钮（使能后会变成蓝色）
   - 死区插入通道 2 使能：切换按钮（使能后会变成蓝色）
   - 死区插入通道 3 使能：切换按钮（使能后会变成蓝色）
   - 请求的下桥臂死区（µs）：0.150
   - 请求的上桥臂死区（µs）：0.250
   - 故障事件输入 A：切换按钮（使能后会变成蓝色）
   - 故障使能：切换按钮（使能后会变成蓝色）
   - 故障中断允许：切换按钮（使能后会变成蓝色）
   - 故障检测重启模式：逐周期
   - 故障检测操作：低电平
8. 从 *Device Resources > Drivers* 中添加 EVSYS 模块，然后进行以下配置：
   - 通道：CHANNEL0
   - 用户：WEXA（从 **CHANNELS**（通道）选项卡中选择 CHANNEL0 矩形，按住并拖动光标至 **USERS**（用户）选项卡中的 WEXA 矩形）
9. 在 **Pin Grid View** 选项卡中检查端口 A 上的 WEX_WO[0-7]引脚是否锁定为输出。当勾选波形输出 n 的使能列中的框时，也会锁定引脚。要更改端口，请单击 **Pin Grid View** 中另一个端口的引脚。
10. 在 **Project Resources** 选项卡中单击 **Generate** 按钮，以便 MCC 生成所有指定的驱动程序和配置。
11. 编辑 `main.c` 文件，如下所示。
    在包含文件中添加以下内容：

```
#include "mcc_generated_files/system/system.h"
#include <util/delay.h>
```

添加宏定义：

```
#define TCE_PERIOD                              (0x3E8)
#define MAX_DUTY_CYCLE                          (0x3DE)
```

添加 ISR 回调函数：

```
/* 在 ISR 程序中调用的回调函数 */
void UserCallback_CMP0(void)
{
    static uint16_t duty_cycle = 0;

    /* 中断中的占空比更新 */
    duty_cycle += 5;
    if(duty_cycle >= MAX_DUTY_CYCLE)
      duty_cycle = 0;
    TCE0_PWM_BufferedDutyCycle0Set(duty_cycle);
}

/* 在 ISR 程序中调用的回调函数 */
void UserCallback_CMP1(void)
{
    static uint16_t duty_cycle = 0;

    /* 中断中的占空比更新 */
    duty_cycle += 15;
    if(duty_cycle >= MAX_DUTY_CYCLE)
      duty_cycle = 0;
    TCE0_PWM_BufferedDutyCycle1Set(duty_cycle);
}

/* 在 ISR 程序中调用的回调函数 */
void UserCallback_CMP2(void)
{
    static uint16_t duty_cycle = 0;

    /* 中断中的占空比更新 */
    duty_cycle += 25;
    if(duty_cycle >= MAX_DUTY_CYCLE)
      duty_cycle = 0;
    TCE0_PWM_BufferedDutyCycle2Set(duty_cycle);
}

/* 在 ISR 程序中调用的回调函数 */
void UserCallback_CMP3(void)
{
    static uint16_t duty_cycle = 0;

    /* 中断中的占空比更新 */
    duty_cycle += 35;
    if(duty_cycle >= MAX_DUTY_CYCLE)
      duty_cycle = 0;
    TCE0_PWM_BufferedDutyCycle3Set(duty_cycle);
}
```

添加函数：

```
void Create_Fault(void)
{
    /* 创建软件故障，在主循环中重复以在逻辑分析器上进行查看。这是使用
     * 软件命令生成的事件 */
    EVSYS_SoftwareEventASet(EVSYS_SWEVENTA_CH0_gc);
}

void Clear_Fault(void)
{
    /* 使用软件命令清除故障条件 */
    WEX0_SoftwareCommand(WEX_CMD_FAULTCLR_gc);
}
```

编辑主函数：

```
int main(void)
{
    SYSTEM_Initialize();

    TCE0_Compare0CallbackRegister(UserCallback_CMP0);
    TCE0_Compare1CallbackRegister(UserCallback_CMP1);
    TCE0_Compare2CallbackRegister(UserCallback_CMP2);
    TCE0_Compare3CallbackRegister(UserCallback_CMP3);

    while(1)
    {
      Create_Fault();
      _delay_us(250);
      Clear_Fault();
      _delay_us(250);
    }
}
```

12. 现在即可使用 MPLAB X IDE 编译并运行该项目。运行期间，每 10 ms 更新一次比较寄存器的缩放值，以调整占空比的范围。

5.3. 结果

下图给出了逻辑分析器捕捉到的一些波形，以展示生成的八个互补 PWM 信号、从 0%增加至100%的占空比、故障事件期间的状态以及死区的外观。

图 5-13. 八个具有可变占空比的 PWM 信号、故障检测操作以及清除故障后恢复正常工作

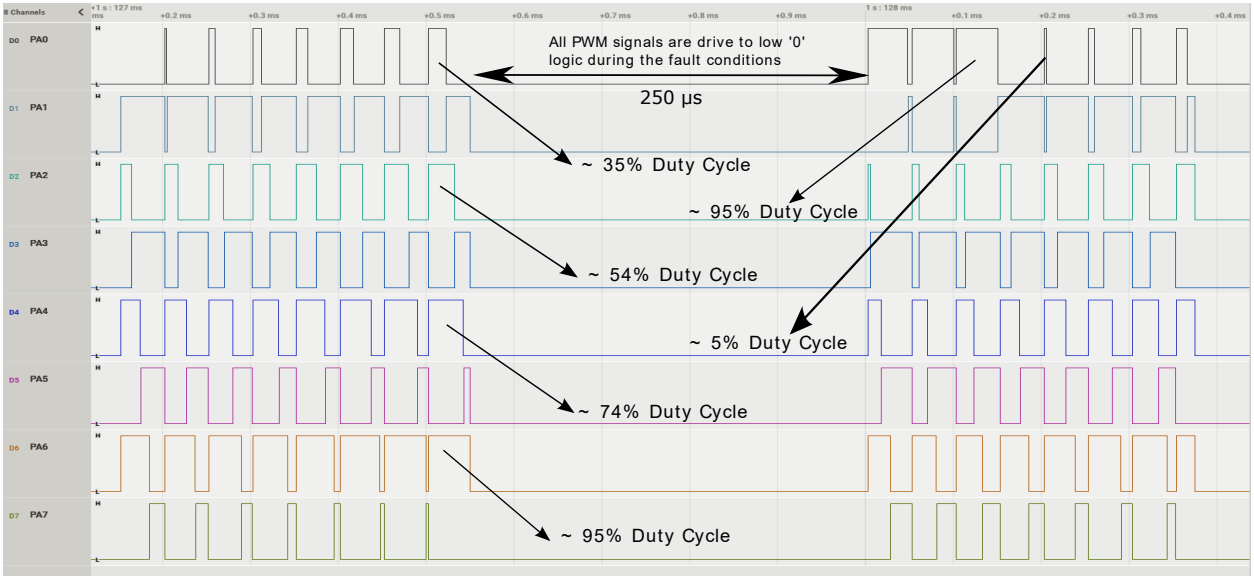


图 5-14. 突出显示的死区

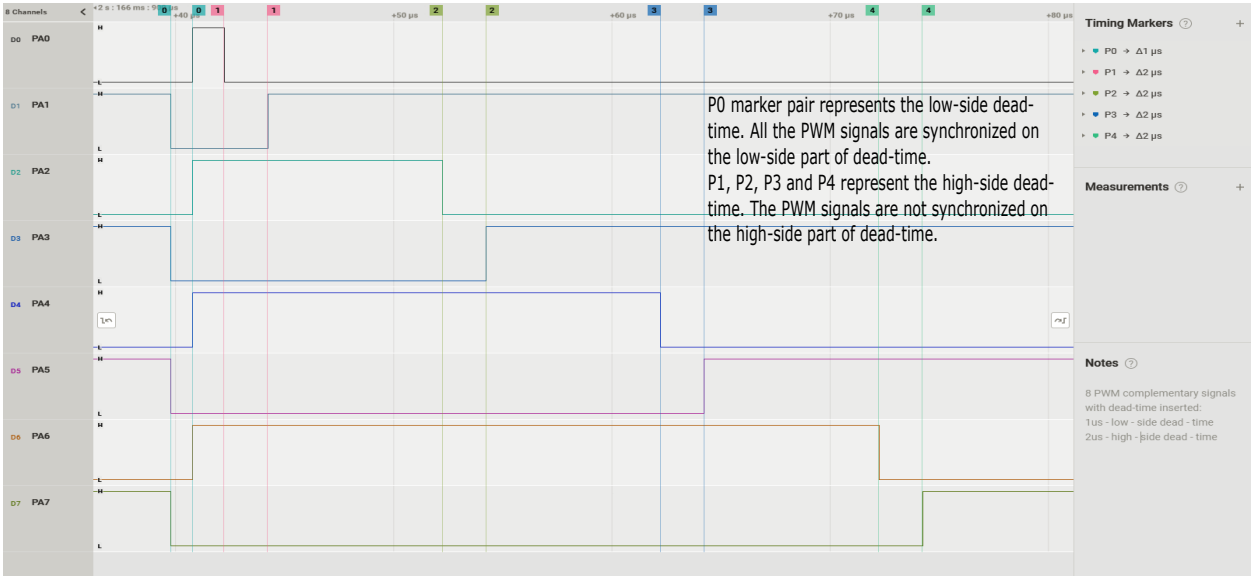


图 5-15. 下桥臂死区的详细信息

如需本部分所述功能的 AVR16EB32 裸机代码示例，可单击此处：

Click to view code examples on MPLAB DISCOVER

如需本部分所述功能的使用 MCC Melody 生成的 AVR16EB32 代码示例，可单击此处：

Click to view code examples on MPLAB DISCOVER

6. 参考资料

有关 TCE 工作模式的更多信息，请访问以下链接：

1. AVR16EB32 产品页面。
2. AVR16EB32 Curiosity Nano 评估工具包。
3. AVR16EB16/20/28/32 AVR® EB Family Data Sheet。

7. 版本历史

文档版本	日期	注释
B	2024 年 8 月	更新了 MULFAC[1:0]寄存器表的图片
A	2023 年 11 月	文档初始版本

Microchip 信息

商标

ISBN：979-8-3371-2321-9

法律声明

Microchip 器件代码保护功能

请注意以下有关 Microchip 产品代码保护功能的要点：

- Microchip 的产品均达到 Microchip 数据手册中所述的技术规范。
- Microchip 确信：在正常使用且符合工作规范的情况下， Microchip 系列产品非常安全。
- Microchip 注重并积极保护其知识产权。严禁任何试图破坏 Microchip 产品代码保护功能的行为，这种行为可能会违反 《数字千年版权法案》（Digital Millennium Copyright Act）
- Microchip 或任何其他半导体厂商均无法保证其代码的安全性。代码保护并不意味着我们保证产品是“牢不可破”的。代码保护功能处于持续发展中。 Microchip 承诺将不断改进产品的代码保护功能。