定时器/计数器 F 型 （TCF）入门指南

TB3341

简介

作者：Bogdan-Alexandru Mariniuc，Microchip Technology Inc.

AVR® EB 系列器件配备了功能强大的定时器，广泛适用于各种应用。F 型定时器/计数器（Timer/Counter type F，TCF）的功能包括生成频率和波形。

该定时器采用异步设计，可连接最高 80 MHz 的外部时钟源。它区别于其他定时器的主要特性之一是具备数控振荡器（Numerical Controller Oscillator，NCO）工作模式。

本技术简介帮助读者熟悉 TCF 的工作模式，重点介绍该定时器的独有特性，例如 NCO 模式和 24 位分辨率。若要更好地了解功能，请参见参考资料部分的数据手册。

本文档涵盖两个具体用例：

- **生成两个恒定导通时间的 PWM 信号：**
  将定时器初始化为 NCO 脉冲频率模式以生成两个具有可变占空比的 PWM 信号，从而实现恒定导通时间效果。
- **生成两个可变频率信号：**
  将定时器初始化为 NCO 固定占空比模式以生成两个可变频率信号。

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：

Click to view code examples on MPLAB DISCOVER

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：

Click to view code examples on MPLAB DISCOVER

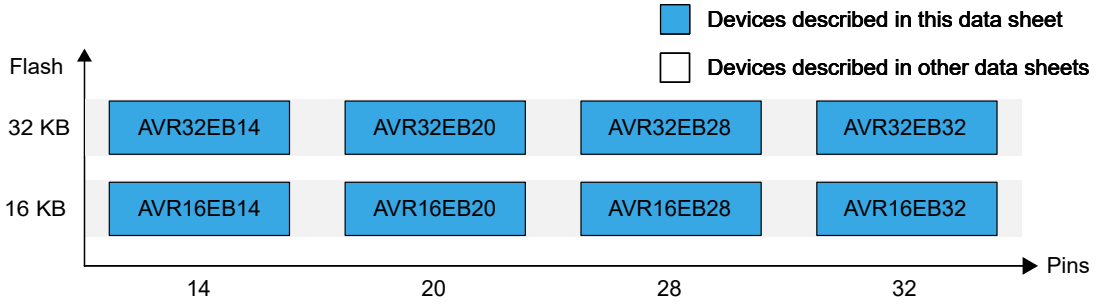
目录

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

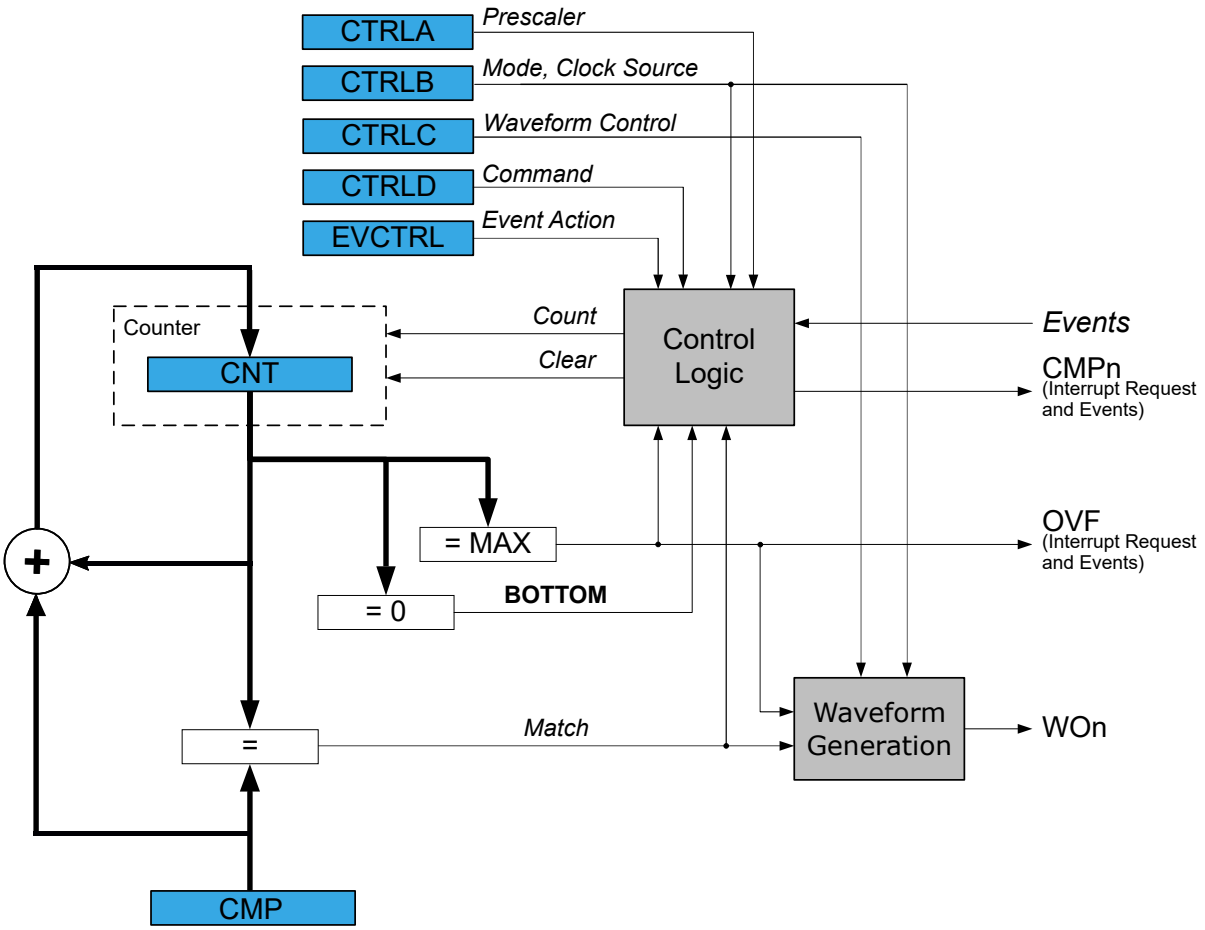
图 1-1. AVR® EB 系列概览



2. 概述

TCF 由基本计数器和控制逻辑组成，可设置为不同的波形生成模式，例如频率生成、NCO 脉冲频率、NCO 固定占空比和 8 位脉宽调制（Pulse-Width Modulation，PWM）。与其他定时器/计数器模块类似，它可基于特定条件（如定时器溢出或比较匹配）产生中断或事件。

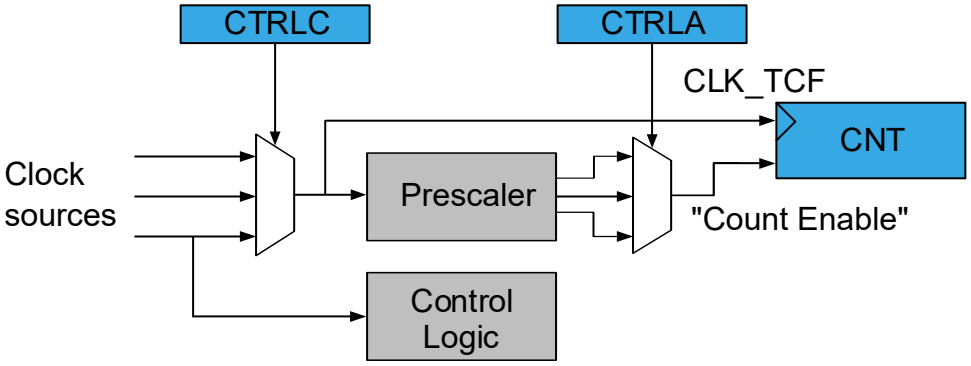
图 2-1. 框图



该定时器/计数器支持多种时钟源，其中包括外设时钟（CLK_PER）、内部振荡器和事件系统（EVSYS）。

可通过控制 B（TCFn.CTRLB）寄存器中的时钟选择（CLKSEL）位域选择其中一个时钟源，以用作预分频器的输入。可使用 7 位预分频器对所选的时钟源进行最高 128 分频。

图 2-2. 时钟选择



使用 TCF 的频率生成模式

在频率波形生成模式下，CMP 寄存器控制周期时间（T）。每次 CNT 与 CMP 寄存器之间发生比较匹配时，都会翻转相应的波形发生器输出。其功能与 TCA 和 TCB 定时器/计数器类似，但 TCF 的计数寄存器宽度为 24 位，而非 16 位。

使用 TCF 的 8 位 PWM 模式

TCF 可配置为以 8 位 PWM 模式运行，在该模式下，最低 16 位比较寄存器中的寄存器对（CMP0 和 CMP1）分别用作独立的比较寄存器。8 位 PWM 功能对于 TCF 来说是一个有益的补充。TCF 8 位 PWM 输出仅为单斜率输出。

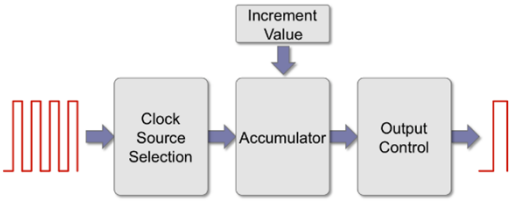
使用 TCF 的 NCO 脉冲频率模式

NCO 模式基于累加器的溢出来生成输出信号。

累加器溢出由可调增量值（而非单个时钟脉冲或后分频器递增）控制，与简单的定时器驱动式计数器相比更具优势，因为分频分辨率不会随着有限的预分频器/后分频器分频值而变化。NCO 对于要求在固定占空比条件下确保频率精度和高分辨率的应用最为有用。

NCO 的工作方式是重复向累加器增加一个固定值。达到输入时钟速率时会进行加法运算。累加器会定期发生进位溢出，该进位为原始的 NCO 输出。这将通过增加值与最大累加器值的比率来降低输入时钟速率。

图 2-3. NCO 逻辑



NCO 输出可以通过延长脉冲或翻转触发器进行修正。之后，修正后的 NCO 输出分配到其他外设，也可输出到 I/O 引脚上。累加器溢出也可以产生中断。NCO 周期以离散步阶进行变化，从而产生一个平均频率。该输出依靠接收电路平均 NCO 输出的能力来降低不确定性。

波形频率（f_{FRQ}）由公式 1 定义：

注：在该模式下，禁止预分频器（N）。

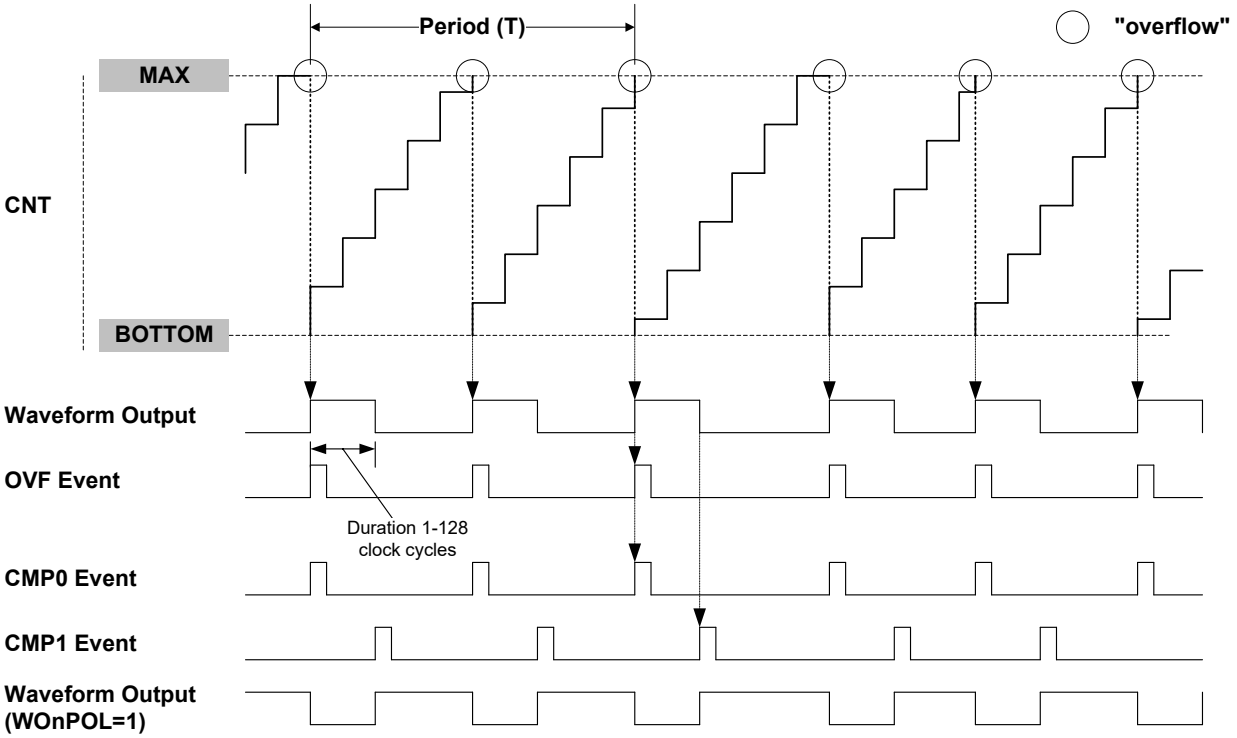
公式 1.FRQ 频率

$$f_{FRQ}\left(Hz\right) = \frac{TCF_{clock}(Hz) \times Increment}{2^{SIZE\_CNT}}$$

在该模式下，输出会在溢出事件发生后的时钟上升沿立即变为有效状态，并在 1 到 128 个时钟周期后变为无效状态，具体的时钟周期数由 TCFn.CTRLB 中的 WGPULSE 位域决定，从而按照设置的频率生成波形输出。

每次波形脉冲开始时都会生成比较匹配 0（CMP0）事件，每次波形脉冲结束时都会生成比较匹配 1（CMP1）事件。中断标志在脉冲事件发生的同时置 1。

图 2-4. NCO 脉冲频率波形生成



使用 TCF 的 NCO 固定占空比频率波形生成模式

TCF 在 NCO 固定占空比频率波形生成模式下的工作方式与 NCO PWM 模式下类似，但是每次累加器溢出时都会翻转输出。

鉴于增量值保持不变，因此会在脉冲频率模式下提供一半频率的 50%占空比。

溢出时会交替生成 CMP0 和 CMP1 事件。

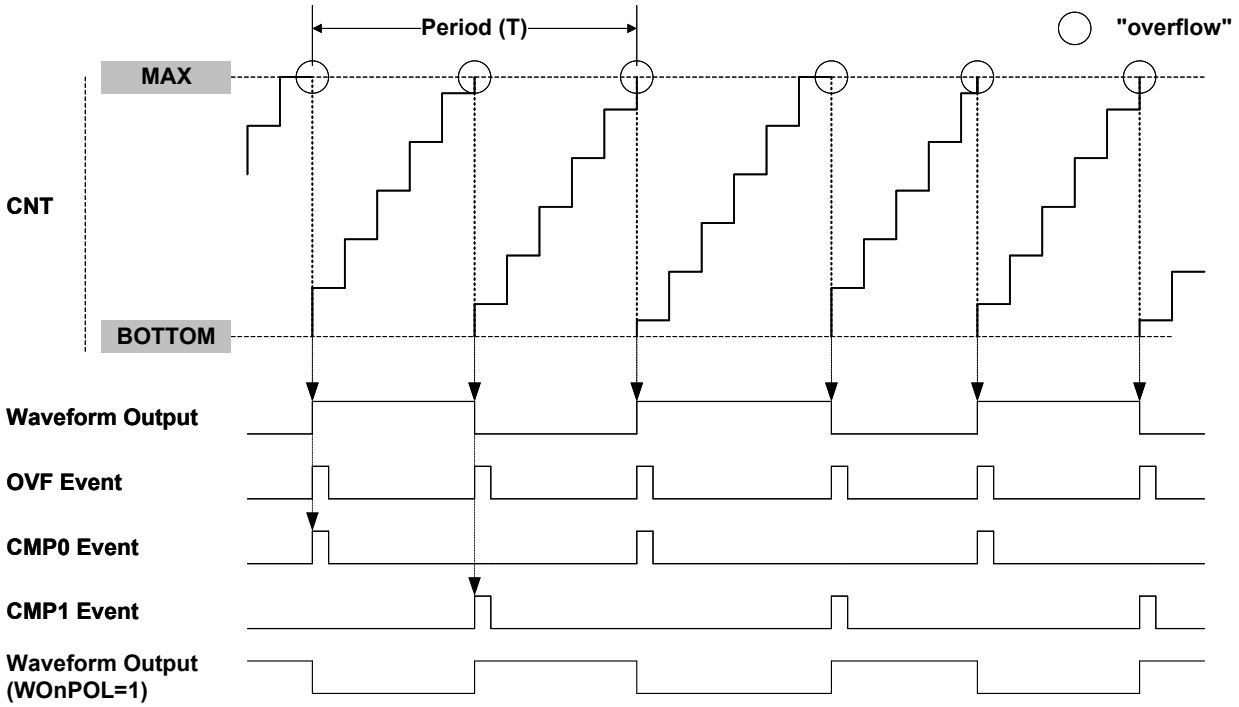
中断标志在脉冲事件发生的同时置 1，如公式 2 所示。

注： N 表示预分频器。

公式 2.FRQ 频率

$$f_{FRQ}\left(Hz\right) = \frac{TCF_{clock}(Hz) \times Increment}{2 \times N \times 2^{SIZE\_CNT}}$$

图 2-5. NCO 固定占空比频率波形生成



3. 生成两个具有可变占空比的固定频率 PWM 信号

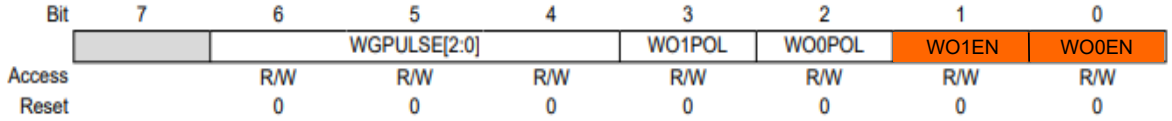
用例说明：将 TCF 配置为以固定频率在比较寄存器上生成溢出事件，同时将波形占空比增加到所有可用的级别。

结果：TCF 将以固定频率在通道 0 和通道 1 上生成输出，同时波形占空比将会增加。

嵌入式裸机实现

1. **配置输出信号**
   使能波形输出 0 和波形输出 1。

   图 3-1. CTRLC 寄存器




```
CTRLC 寄存器为双缓冲寄存器。建议每次写入 CTRLC 寄存器时，都在一个循环中等待状态寄存器中的 CTRLCBUSY 位清零。
创建一个名为 TCF0_OutputsSet 的函数，该函数返回 void 并接受一个 uint8_t 类型的参数。
void TCF0_OutputsSet(uint8_t value)
    {
        while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
        TCF0.CTRLC = value;
    }
```

2. **配置 TCF 时钟**
   AVR16EB32 板默认以 3.33 MHz 的系统时钟运行。为了使 TCF 以 20 MHz 运行，必须禁止系统时钟预分频器。

```
创建一个函数，该函数返回 void 并接受一个 void 类型的参数。
void CLOCK_Initialize(void)
{
    _PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, 0x0);
}
```

图 3-2. CTRLB 寄存器

Bit	7	6	5	4	3	2	1	0
	CMP1EV	CMP0EV	CLKSEL[2:0]			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

图 3-3. 时钟选择位掩码

Bits 5:3 – CLKSEL[2:0] Clock Select

This bit field controls the Timer/Counter clock source and cannot be changed while the Timer/Counter is enabled.

Value	Name	Description
0x0	CLKPER	Peripheral clock
0x1	EVENT	Event edge
0x2	OSCHF	Internal high-frequency oscillator
0x3	OSC32K	Internal 32.768 kHz oscillator
0x4	XOSCHF	High-frequency crystal oscillator
0x5	PLL	Phase Locked Loop
Others	-	Reserved

TCF 提供了多个时钟选项。对于本应用，选择默认选项，即以 20 MHz 运行的外设时钟。可通过 CTRLB 寄存器选择时钟。

```
创建一个函数，该函数返回 void 并接受一个 TCF_CLKSEL_t 类型的参数。
void TCF0_ClockSet(TCF_CLKSEL_t config)
{
    TCF0.CTRLB |= config;
}
```

在该模式下，禁止预分频器。

3. **配置波形生成模式**
   使用 CTRLB 寄存器选择定时器的工作模式。本用例将选择 NCOPF。

图 3-4. 波形生成模式位掩码

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

This bit field selects the Waveform mode.

Value	Name	Description
0x0	FRQ	Frequency
0x1	NCOPF	Numerical Controlled Oscillator Pulse-Frequency
0x2	NCOFDC	Numerical Controlled Oscillator Fixed Duty-Cycle
0x3-0x6	-	Reserved
0x7	PWM8	8-Bit PWM mode

```
创建一个函数，该函数返回 void 并接受一个 TCF_WGMODE_t 类型的参数。
void TCF0_ModeSet(TCF_WGMODE_t mode)
{
    TCF0.CTRLB |= mode;
}
```

4. **设置频率**
   配置 TCF 时钟后，使用 CMP 寄存器选择定时器的运行频率——125 kHz。

图 3-5. CMP 寄存器

0x14	CMP	7:0	CMP[7:0]
		15:8	CMP[15:8]
		23:16	CMP[23:16]
		31:24	

可使用公式 1 计算增量，具体如下：$Increment = \frac{f_{FRQ}(Hz) \times 2^{SIZE\_CNT}}{TCF_{clock}(Hz)}$

代入所需的值后：

$Increment = \frac{125.000 \times 16.777.216}{20.000.000} = 104,857.6$

十六进制结果为 0x0001999A。

```
上述公式可以转化为以下公式：
#define TCF0_NCOPL_HZ_TO_INCREMENT(HZ, F_CLOCK)    (uint32_t)(((float)(HZ) * 16777216.0) /
(float)(F_CLOCK) + 0.5)
```

```
CMP 寄存器为双缓冲寄存器。建议每次写入 CMP 寄存器时，都在一个循环中等待状态寄存器中的 CMP0BUSY 位清零。
void TCF0_CompareSet(uint32_t value)
{
    while((TCF0.STATUS & TCF_CMP0BUSY_bm) != 0){};
    TCF0.CMP = value;
}
```

```
void TCF0_CounterSet(uint32_t value)
{
    while((TCF0.STATUS & TCF_CNTBUSY_bm) != 0){};
    TCF0.CNT0 = (uint8_t)value;
}
```

5. **配置波形生成脉冲长度**

图 3-6. CTRLC 寄存器

Bit	7	6	5	4	3	2	1	0
		WGPULSE[2:0]			WO1POL	WO0POL	WO1EN	WO0EN
Access		R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset		0	0	0	0	0	0	0

图 3-7. 波形生成脉冲长度位掩码

Bits 6:4 – WGPULSE[2:0] Waveform Generation Pulse Length

This bit field controls the generated waveform high time in Pulse-Frequency mode.

Value	Name	Description
0x0	CLK1	High pulse is 1 Timer/Counter clock period
0x1	CLK2	High pulse is 2 Timer/Counter clock periods
0x2	CLK4	High pulse is 4 Timer/Counter clock periods
0x3	CLK8	High pulse is 8 Timer/Counter clock periods
0x4	CLK16	High pulse is 16 Timer/Counter clock periods
0x5	CLK32	High pulse is 32 Timer/Counter clock periods
0x6	CLK64	High pulse is 64 Timer/Counter clock periods
0x7	CLK128	High pulse is 128 Timer/Counter clock periods

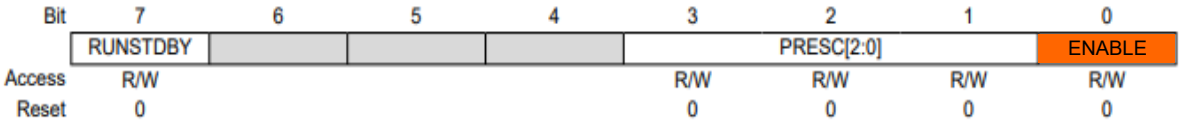
将脉冲长度设置为单个时钟增量，即 50 ns。由于定时器以 20 MHz 运行，因此 1 个时钟周期为 50 ns。1 除以 20 MHz 等于 50 ns。

```
void TCF0_NCO_PulseLengthSet(TCF_WGPULSE_t config)
{
    uint8_t temp;
```

```
    while((TCF0.STATUS & TCF_CTRLCBUSY_bm) != 0){};
    temp = (TCF0.CTRLC & ~TCF_WGPULSE_gm) |
               (  config   &  TCF_WGPULSE_gm);
    TCF0.CTRLC = temp;
}
```

6. **启动和停止定时器**

图 3-8. CTRLA 寄存器




```
void TCF0_Start(void)
{
    while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
    TCF0.CTRLA |=  TCF_ENABLE_bm;
}
void TCF0_Stop(void)
{
    while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
    TCF0.CTRLA &= ~TCF_ENABLE_bm;
}
```

TCF 将进行以下初始化设置：

- 使能两个输出
- 选择外设时钟
- 设置为 NCO 脉冲长度模式
- 将频率设置为 125 kHz
- 将波形生成脉冲长度设置为 1 个时钟周期
- 将定时器设置为从 0 开始计数

```
创建一个函数，该函数返回 void 并接受一个 void 类型的参数。
void TCF0_Initialize(void)
{
   TCF0_OutputsSet(TCF_WO0EN_bm | TCF_WO1EN_bm);
   TCF0_ClockSet(TCF_CLKSEL_CLKPER_gc);
   TCF0_ModeSet(TCF_WGMODE_NCOPF_gc);
   TCF0_CompareSet(TCF0_NCOPL_HZ_TO_INCREMENT(125000, 20000000));
   TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK1_gc);
   TCF0_CounterSet(0);
}
```

```
util/delay.h 头文件需要定义 F_CPU 频率。
#define F_CPU 20000000UL
包含 util/delay.h 头文件
#include <util/delay.h>
```

```
创建一个函数，该函数返回 void 并接受一个 void 类型的参数。使用 TCF0_NCO_Pulse_Length_Demo 函数以一定的延时为间隔更改脉冲长度。
void NCO_Pulse_Length_Demo(void)
{
    /* 将 TCF 配置为从 0 开始计数 */
    TCF0_CounterSet(0);

    /* 使能 TCF */
    TCF0_Start();

    /* 20 µs 延时 */
   _delay_us(20);
```

```
    /* 将脉冲长度配置为 2 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK2_gc);

    /* 20 µs 延时 */
    _delay_us(20);

    /* 将脉冲长度配置为 4 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK4_gc);

    /* 20 µs 延时 */
    _delay_us(20);

    /* 将脉冲长度配置为 8 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK8_gc);

    /* 20 µs 延时 */
    _delay_us(20);

    /* 将脉冲长度配置为 16 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK16_gc);

    /* 20 µs 延时 */
    _delay_us(20);

    /* 将脉冲长度配置为 32 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK32_gc);

    /* 20 µs 延时 */
    _delay_us(20);

    /* 将脉冲长度配置为 64 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK64_gc);

    /* 20 µs 延时 */
    _delay_us(20);

    /* 将脉冲长度配置为 128 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK128_gc);

    /* 25 µs 延时 */
    _delay_us(25);

    /* 停止定时器 */
    TCF0_Stop();

    /* 将脉冲长度配置为 1 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK1_gc);
}
```

main 函数如下所示：

```
void main(void)
{
    CLOCK_Initialize();
    TCF0_Initialize();
    while(1)
    {
        NCO_Pulse_Length_Demo();
        _delay_ms(20);
    }
}
```

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 Content Manager Wizard**（MCC 内容管理器向导）中选择 **MCC Melody**，然后单击 **Finish**（完成）。
4. 从 **Device Resources**（器件资源）中转到 **System**（系统），然后单击 **CLCKCTRL** 窗口并禁止预分频器。
5. 从 **Device Resource** 中转到 **Drivers**（驱动程序），然后单击定时器窗口，添加 TCF 模块，并进行以下配置：
   - 时钟分频器：系统时钟（默认情况下，分频值将为 1——系统时钟）
   - 波形生成模式：NCO 脉冲长度模式
   - 波形生成脉冲长度：1 个时钟周期
   - 请求的周期[s]： 0.000008
   - 波形输出 n：勾选波形输出 0 和波形输出 1 的使能列中的框
6. 在 **Pin Grid View**（引脚网格视图）中选择 PA0 和 PA1 引脚。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。
7. 在 **Project Resources**（项目资源）窗口中单击 **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，具体如下所述：
   包含 util/delay.h 头文件。

```
#include <util/delay.h>
```

   创建一个名为 NCO_Pulse_Length_Demo 的函数。使用 TCF0_NCO_PulseLengthSet 函数更改脉冲长度。
   代码如下：

```
void NCO_Pulse_Length_Demo(void)
{
     /* 将 TCF 配置为从 0 开始计数 */
    TCF0_CounterSet(0);

    /* 使能 TCF */
    TCF0_Start();

    /* 20 µs 延时 */
   _delay_us(20);

   /* 将脉冲长度配置为 2 个时钟周期 */
   TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK2_gc);

   /* 20 µs 延时 */
   _delay_us(20);

   /* 将脉冲长度配置为 4 个时钟周期 */
   TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK4_gc);

   /* 20 µs 延时 */
   _delay_us(20);

   /* 将脉冲长度配置为 8 个时钟周期 */
   TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK8_gc);

   /* 20 µs 延时 */
   _delay_us(20);

   /* 将脉冲长度配置为 16 个时钟周期 */
```

```
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK16_gc);

    /* 20 µs 延时 */
    _delay_us(20);

    /* 将脉冲长度配置为 32 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK32_gc);

    /* 20 µs 延时 */
    _delay_us(20);

    /* 将脉冲长度配置为 64 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK64_gc);

    /* 20 µs 延时 */
    _delay_us(20);

    /* 将脉冲长度配置为 128 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK128_gc);

    /* 25 µs 延时 */
    _delay_us(25);

    /* 停止定时器 */
    TCF0_Stop();

    /* 将脉冲长度配置为 1 个时钟周期 */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK1_gc);
}
```

9. main 函数如下所示：

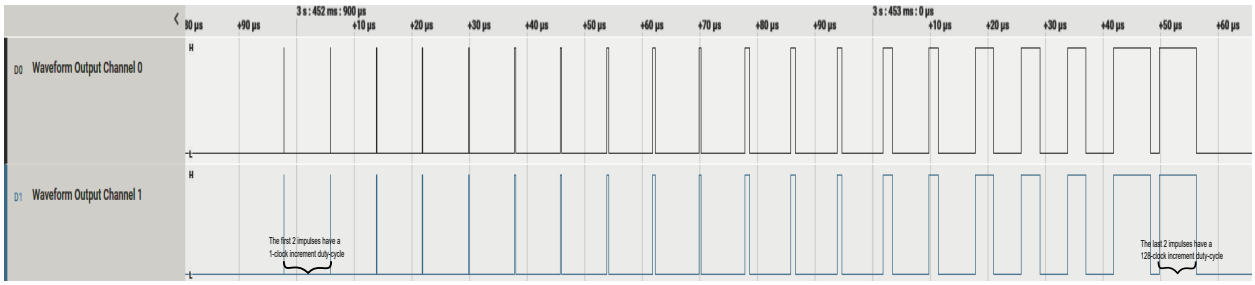
```
int main(void)
{
    SYSTEM_Initialize();

    while(1)
    {
        NCO_Pulse_Length_Demo();
        _delay_ms(20);
    }
}
```

10. 刷写项目。

11. 结果：

TCF 生成两个频率为 125 kHz 的相同信号，并生成脉冲长度范围为 1 至 128 个时钟周期的 PWM 信号。在本用例中，一个时钟周期为 50 ns。

图 3-9. 结果



Click to view code examples on MPLAB DISCOVER

4. 在 NCO 固定占空比波形生成模式下生成两个可变频率信号

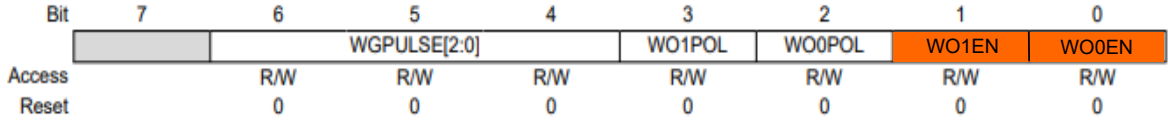
用例说明：将 TCF 配置为以 10 Hz 至 100 kHz 范围内的频率在比较寄存器上生成溢出事件。

结果：TCF 将在通道 0 和通道 1 上以某个范围的频率生成输出，每次累加器溢出时都会翻转输出。

嵌入式裸机实现

1. **配置输出信号**
   使能波形输出 0 和波形输出 1。

   图 4-1. CTRLC 寄存器




```
CTRLC 寄存器为双缓冲寄存器。建议每次写入 CTRLC 寄存器时，都在一个循环中等待状态寄存器中的 CTRLCBUSY 位清零。
创建一个名为 TCF0_OutputsSet 的函数，该函数返回 void 并接受一个 uint8_t 类型的参数。
void TCF0_OutputsSet(uint8_t value)
    {
        while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
        TCF0.CTRLC = value;
    }
```

2. **配置 TCF 时钟**
   AVR16EB32 板默认以 3.33 MHz 的系统时钟运行。为了使 TCF 以 20 MHz 运行，必须禁止系统时钟预分频器。

```
创建一个函数，该函数返回 void 并接受一个 void 类型的参数。
void CLOCK_Initialize(void)
{
    _PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, 0x0);
}
```

图 4-2. CTRLB 寄存器

Bit	7	6	5	4	3	2	1	0
	CMP1EV	CMP0EV	CLKSEL[2:0]			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

图 4-3. 时钟选择位掩码

Bits 5:3 – CLKSEL[2:0] Clock Select

This bit field controls the Timer/Counter clock source and cannot be changed while the Timer/Counter is enabled.

Value	Name	Description
0x0	CLKPER	Peripheral clock
0x1	EVENT	Event edge
0x2	OSCHF	Internal high-frequency oscillator
0x3	OSC32K	Internal 32.768 kHz oscillator
0x4	XOSCHF	High-frequency crystal oscillator
0x5	PLL	Phase Locked Loop
Others	-	Reserved

TCF 提供了多个时钟选项。对于本应用，选择默认选项，即以 20 MHz 运行的外设时钟。可通过 CTRLB 寄存器选择时钟。

```
创建一个函数，该函数返回 void 并接受一个 TCF_CLKSEL_t 类型的参数。
void TCF0_ClockSet(TCF_CLKSEL_t config)
{
    TCF0.CTRLB |= config;
}
```

3. **选择预分频器**

图 4-4. CTRLA 寄存器

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

```
CTRLA 寄存器为双缓冲寄存器。建议每次写入 CTRLA 寄存器时，都在一个循环中等待状态寄存器中的 CTRLABUSY 位清零。
创建一个函数，该函数返回 void 并接受一个 TCF_PRESC_t 类型的参数。
void TCF0_PrescalerSet(TCF_PRESC_t config)
{
    while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
    TCF0.CTRLA  |= config;
}
```

4. **配置波形生成模式**

使用 CTRLB 寄存器选择定时器的工作模式。本用例将选择 NCOFDC。

图 4-5. CTRLB 寄存器

Bit	7	6	5	4	3	2	1	0
	CMP1EV	CMP0EV	CLKSEL[2:0]			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

图 4-6. 波形生成模式位掩码

Bits 2:0 – WGMODE[2:0] Waveform Generation Mode
This bit field selects the Waveform mode.

Value	Name	Description
0x0	FRQ	Frequency
0x1	NCOPF	Numerical Controlled Oscillator Pulse-Frequency
0x2	NCOFDC	Numerical Controlled Oscillator Fixed Duty-Cycle
0x3-0x6	-	Reserved
0x7	PWM8	8-Bit PWM mode

```
创建一个函数，该函数返回 void 并接受一个 TCF_WGMODE_t 类型的参数。
void TCF0_ModeSet(TCF_WGMODE_t mode)
{
    TCF0.CTRLB |= mode;
}
```

5. **设置 CMP 寄存器**

可使用公式 2 计算增量，具体如下：

$$Increment = \frac{f_{FRQ}(Hz) \times 2^{SIZE\_CNT} \times N}{TCF_{clock}(Hz)}$$

代入所需的值后：

$$Increment = \frac{10 \times 16.777.216 \times 2 \times 1}{20.000.000} = 104,857.6$$

结果为十六进制，即 0x00000011。

```
上述公式可以转化为以下公式：
#define TCF0_NCOFD_HZ_TO_INCREMENT(HZ, F_CLOCK, TCF0_PRESCALER)    (uint32_t)(((float)(HZ)
* 33554432.0 * (TCF0_PRESCALER)) / ((float)(F_CLOCK)) + 0.5)
```

启动定时器之前，向 CMP 中写入值 11。

图 4-7. CMP 寄存器

0x14	CMP	7:0	CMP[7:0]
		15:8	CMP[15:8]
		23:16	CMP[23:16]
		31:24	

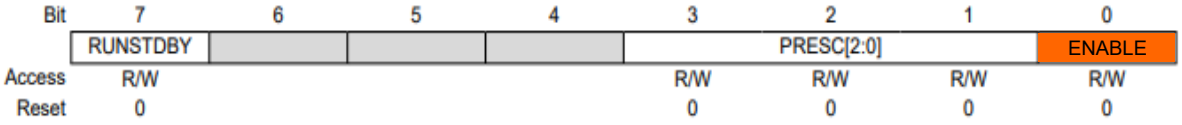
```
CMP 寄存器为双缓冲寄存器。建议每次写入 CMP 寄存器时，都在一个循环中等待状态寄存器中的 CMP0BUSY 位清零。
创建一个函数，该函数返回 void 并接受一个 uint32_t 类型的参数。
void TCF0_CompareSet(uint32_t value)
{
    while((TCF0.STATUS & TCF_CMP0BUSY_bm) != 0){};
    TCF0.CMP = value;
}
```

```
CNT 寄存器为双缓冲寄存器。建议每次写入 CNT 寄存器时，都在一个循环中等待状态寄存器中的 CNTBUSY 位清零。
创建一个函数，该函数返回 void 并接受一个 uint32_t 类型的参数。
void TCF0_CounterSet(uint32_t value)
{
    while((TCF0.STATUS & TCF_CNTBUSY_bm) != 0){};
```

```
    TCF0.CNT0 = (uint8_t)value;
}
```

6. **启动和停止定时器**

图 4-8. CTRLA 寄存器




```
CTRLA 寄存器为双缓冲寄存器。建议每次写入 CTRLA 寄存器时，都在一个循环中等待状态寄存器中的 CTRLABUSY 位清零。
创建一个函数，该函数返回 void 并接受一个 void 类型的参数。
void TCF0_Start(void)
{
    while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
    TCF0.CTRLA |=  TCF_ENABLE_bm;
}

创建一个函数，该函数返回 void 并接受一个 void 类型的参数。
void TCF0_Stop(void)
{
    while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
    TCF0.CTRLA &= ~TCF_ENABLE_bm;
}
```

TCF 将进行以下初始化设置：

- 使能两个输出
- 选择外设时钟
- 设置为 NCO 固定占空比模式
- 将波形生成脉冲长度设置为 1 个时钟周期
- 将频率设置为 10 Hz
- 将定时器设置为从 0 开始计数

```
创建一个函数，该函数返回 void 并接受一个 void 类型的参数。
void TCF0_Initialize(void)
{
   TCF0_OutputsSet(TCF_WO0EN_bm | TCF_WO1EN_bm);
   TCF0_ClockSet(TCF_CLKSEL_CLKPER_gc);
   TCF0_PrescalerSet(TCF_PRESC_DIV1_gc);
   TCF0_ModeSet(TCF_WGMODE_NCOFDC_gc);
   TCF0_CounterSet(0);
   TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(10, 20000000, 1));
}
```

```
util/delay.h 头文件需要定义 F_CPU 频率。
#define F_CPU 20000000UL
包含 util/delay.h 头文件
#include <util/delay.h>
```

```
创建一个函数，该函数返回 void 并接受一个 uint_32t 类型的参数。使用 TCF0_CompareSet 函数更改频率。
void NCO_Fixed_DutyCycle_Demo(void)
{
    /* 将 TCF 配置为从 0 开始计数 */
    TCF0_CounterSet(0);

    /* 使能 TCF */
    TCF0_Start();

    /* 600 ms 延时 */
```

```
    _delay_ms(600);

    /* 向 CMP 寄存器中装载 100 Hz 频率 */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(100, 20000000, 1));

     /* 60 ms 延时 */
    _delay_ms(60);

    /* 向 CMP 寄存器中装载 1 KHz 频率 */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(1000, 20000000, 1));

     /* 6 ms 延时 */
    _delay_ms(6);

    /* 向 CMP 寄存器中装载 10 KHz 频率 */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(10000, 20000000, 1));

     /* 600 µs 延时 */
    _delay_us(600);

    /* 向 CMP 寄存器中装载 100 KHz 频率 */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(100000, 20000000, 1));

     /* 60 µs 延时 */
    _delay_us(60);

     /* 停止 TCF */
    TCF0_Stop();

    /* 向 CMP 寄存器中装载 10 Hz 频率 */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(10, 20000000, 1));
}
```

main 函数如下所示：

```
void main(void)
{
    CLOCK_Initialize();
    TCF0_Initialize();
    while(1)
    {
        NCO_Fixed_DutyCycle_Demo();
        _delay_ms(1000);
    }
}
```

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 Content Manager Wizard** 中选择 **MCC Melody**，然后单击 **Finish**。
4. 从 **Device Resources** 中转到 **System**，然后单击 **CLCKCTRL** 窗口并禁止预分频器。
5. 从 **Device Resource** 中转到 **Drivers**，然后单击定时器窗口，添加 TCF 模块，并进行以下配置：
   - 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：NCO 固定占空比模式
   - 请求的周期[s]：0.1
   - 波形输出 n：勾选波形输出 0 和波形输出 1 的使能列中的框
6. 在 **Pin Grid View** 中选择 PA0 和 PA1 引脚。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。
7. 在 **Project Resources** 窗口中单击 **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，具体如下所述：
   包含 util/delay.h 头文件。

```
#include <util/delay.h>
```

创建一个名为 NCO_Fixed_DutyCycle_Demo 的函数。使用 TCF0_CompareSet 函数更改频率。

代码如下：

```
创建一个函数，该函数返回 void 并接受一个 void 类型的参数。
void NCO_Fixed_DutyCycle_Demo(void)
{
    /* 将 TCF 配置为从 0 开始计数 */
    TCF0_CounterSet(0);

    /* 使能 TCF */
    TCF0_Start();

    /* 600 ms 延时 */
   _delay_ms(600);

   /* 向 CMP 寄存器中装载 100 Hz 频率 */
   TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(100, 20000000, 1));

    /* 60 ms 延时 */
   _delay_ms(60);

   /* 向 CMP 寄存器中装载 1 KHz 频率 */
   TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(1000, 20000000, 1));

    /* 6 ms 延时 */
   _delay_ms(6);

   /* 向 CMP 寄存器中装载 10 KHz 频率 */
   TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(10000, 20000000, 1));

    /* 600 µs 延时 */
   _delay_us(600);

   /* 向 CMP 寄存器中装载 100 KHz 频率 */
   TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(100000, 20000000, 1));

    /* 60 µs 延时 */
   _delay_us(60);

    /* 停止 TCF */
   TCF0_Stop();
```

```
    /* 向 CMP 寄存器中装载 10 Hz 频率 */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(10, 20000000, 1));
}
```

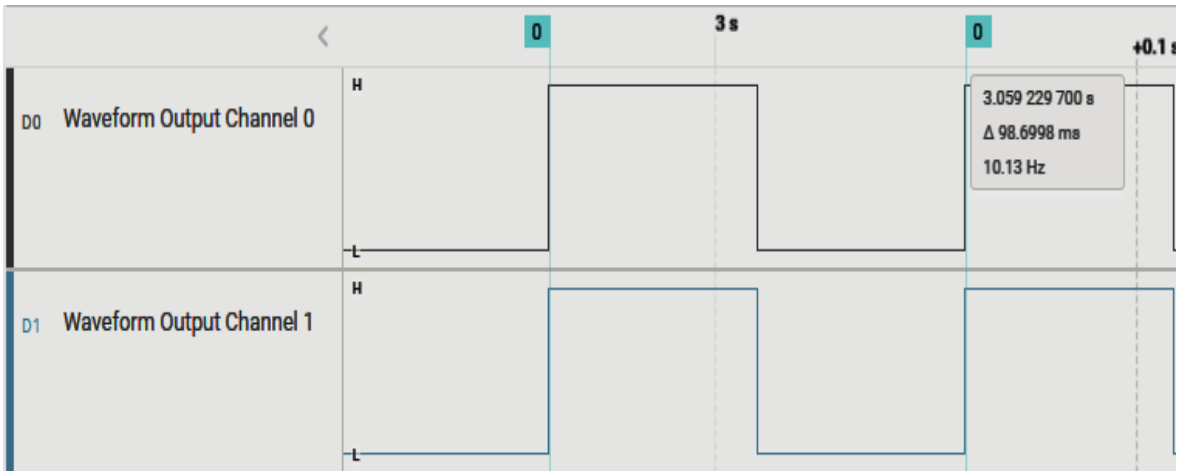
9. main.c 文件如下所示：

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

    while(1)
    {
        NCO_Fixed_DutyCycle_Demo();
        _delay_ms(1000);
    }
}
}
```

10. 刷写项目。

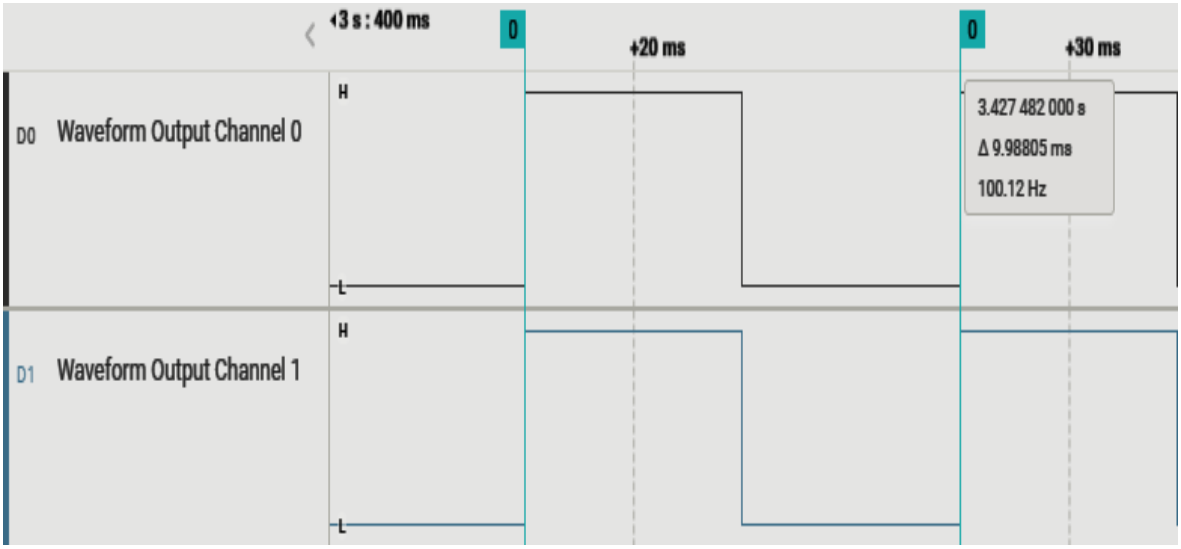
结果 1：生成两个频率为 10 Hz、占空比为 50%的相同信号。

图 4-9. 结果 1



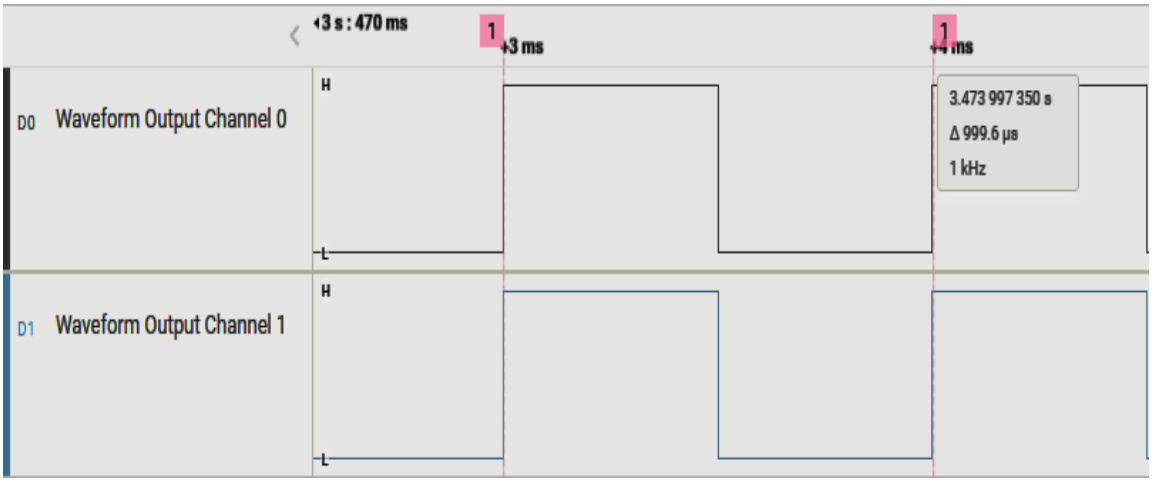
结果 2：生成两个频率为 100 Hz、占空比为 50%的相同信号。

图 4-10. 结果 2



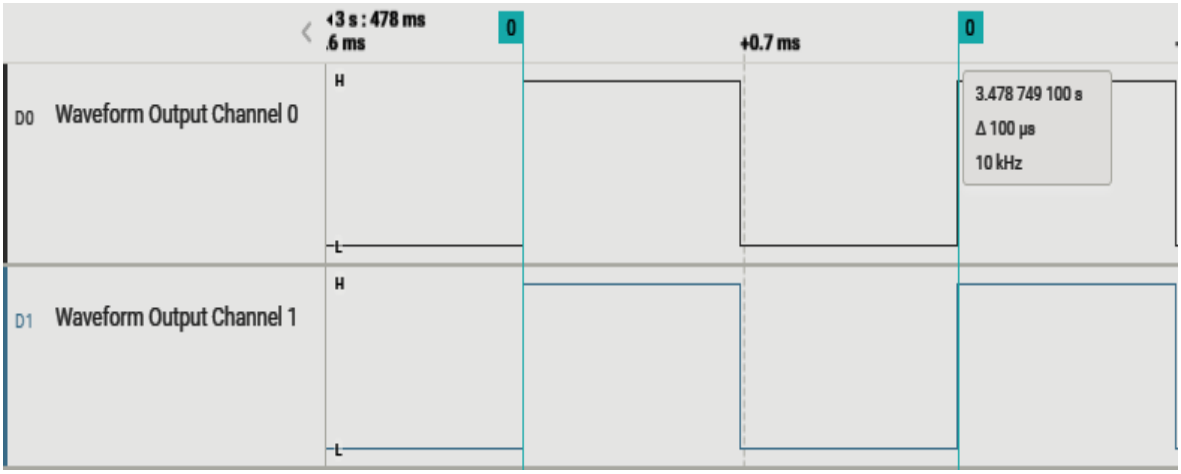
结果 3：生成两个频率为 1 kHz、占空比为 50%的相同信号。

图 4-11. 结果 3



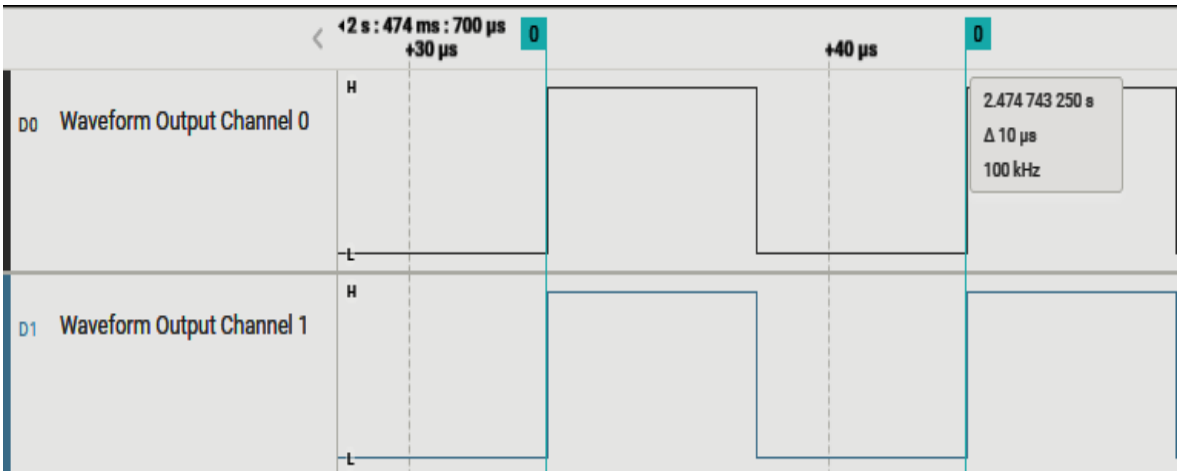
结果 4：生成两个频率为 10 kHz、占空比为 50%的相同信号。

图 4-12. 结果 4



结果 5：生成两个频率为 100 kHz、占空比为 50%的相同信号。

图 4-13. 结果 5



Click to view code examples on MPLAB DISCOVER

5. 参考资料

有关 TCF 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

6. 版本历史

文档版本	日期	注释
A	2023 年 10 月	文档初始版本

Microchip 信息

商标

“Microchip”的名称和徽标组合、“M”徽标及其他名称、徽标和品牌均为 Microchip Technology Incorporated 或其关联公司和/或子公司在美国和/或其他国家或地区的注册商标或商标（“Microchip 商标”）。有关 Microchip 商标的信息，可访问 https://www.microchip.com/en-us/about/legal-information/microchip-trademarks。

ISBN：979-8-3371-2433-9

法律声明

提供本文档的中文版本仅为了便于理解。请勿忽视文档中包含的英文部分，因为其中提供了有关 Microchip 产品性能和使用情况的有用信息。Microchip Technology Inc.及其分公司和相关公司、各级主管与员工及事务代理机构对译文中可能存在的任何差错不承担任何责任。建议参考 Microchip Technology Inc.的英文原版文档。

本出版物及其提供的信息仅适用于 Microchip 产品，包括设计、测试以及将 Microchip 产品集成到您的应用中。以其他任何方式使用这些信息都将被视为违反条款。本出版物中的器件应用信息仅为您提供便利，将来可能会发生更新。您须自行确保应用符合您的规范。如需额外的支持，请联系当地的 Microchip 销售办事处，或访问 www.microchip.com/en-us/support/design-help/client-support-services。

Microchip“按原样”提供这些信息。Microchip 对这些信息不作任何明示或暗示、书面或口头、法定或其他形式的声明或担保，包括但不限于针对非侵权性、适销性和特定用途的适用性的暗示担保，或针对其使用情况、质量或性能的担保。

在任何情况下，对于因这些信息或使用这些信息而产生的任何间接的、特殊的、惩罚性的、偶然的或附带的损失、损害或任何类型的开销，Microchip 概不承担任何责任，即使 Microchip 已被告知可能发生损害或损害可以预见。在法律允许的最大范围内，对于因这些信息或使用这些信息而产生的所有索赔，Microchip 在任何情况下所承担的全部责任均不超出您为获得这些信息向 Microchip 直接支付的金额（如有）。如果将 Microchip 器件用于生命维持和/或生命安全应用，一切风险由买方自负。买方同意在由此引发任何一切损害、索赔、诉讼或费用时，会维护和保障 Microchip 免于承担法律责任。除非另外声明，在 Microchip 知识产权保护下，不得暗中或以其他方式转让任何许可证。

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：在按照操作规范正常使用的情况下，Microchip 系列产品非常安全。
- Microchip 重视并积极保护其知识产权。任何试图破坏 Microchip 产品代码保护功能的行为均可视为违反了《数字器件千年版权法案（Digital Millennium Copyright Act）》并予以严禁。
- 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。