TB - Getting Started with Timer/Counter Type F (TCF)

TB3341

Introduction

Author: Bogdan-Alexandru Mariniuc Microchip Technology Inc.

The AVR® EB family of devices contains powerful timers that enable them to cover a range of applications. The Timer/Counter type F (TCF) functions include frequency and waveform generation.

The timer is asynchronous and can be connected to an external clock source of up to 80 MHz. One of the main characteristics which differentiates it from other timers is the Numerical Controller Oscillator (NCO) operation mode.

This technical brief familiarizes the reader with TCF operating modes, emphasizing the timer's unique features, such as NCO mode and 24-bit resolution. To better understand the functionality, consult the data sheet in the References section.

This document covers two specific use cases:

- **Generate Two Constant-On-Time PWM Signals:**
  Initialize the timer in NCO Pulse-Frequency Mode to generate two PWM signals with a variable duty cycle that creates a Constant-On-Time effect.
- **Generate Two Variable-Frequency Signals:**
  Initialize the timer in NCO Fixed Duty Cycle Mode to generate two variable frequency signals.

Note: There are two code examples for each use case described in this document. One bare metal developed on AVR16EB32, and one generated with MPLAB® Code Configurator (MCC) developed on AVR16EB32.

Find bare metal code examples for AVR16EB32 with the same functionality as the ones described in this technical brief here:

Click to view code examples on MPLAB DISCOVER

Find MCC Melody generated code examples for AVR16EB32 with the same functionality as the ones described in this technical brief here:

Click to view code examples on MPLAB DISCOVER

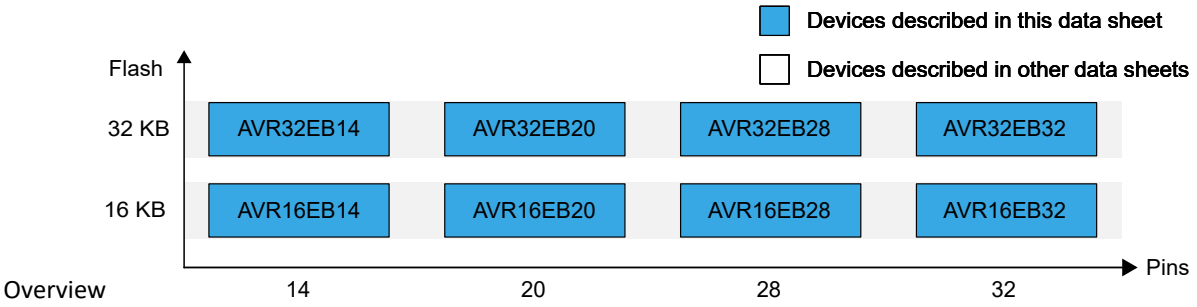
Table of Contents

1. Relevant Devices

This section lists the relevant devices for this document. The following figure shows various family devices, laying out pin count variants and memory sizes:

- Vertical migration upwards is possible without code modification, as these devices are pin-compatible and provide the same or more features. Downward migration on the AVR EB series devices may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and the available features
- Devices with different Flash memory sizes usually have different SRAM and EEPROM

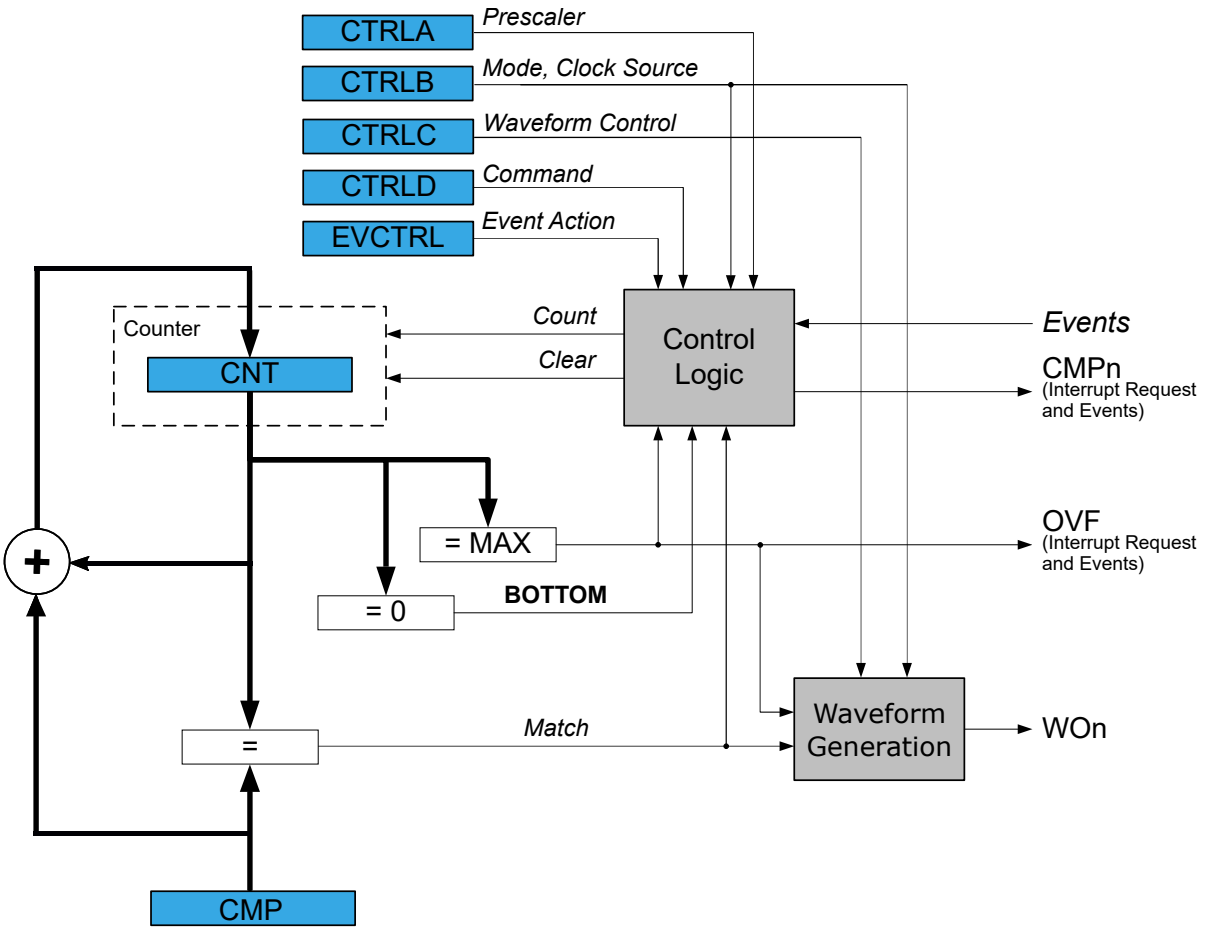
Figure 1-1. AVR® EB Family



2. Overview

The TCF consists of a base counter and control logic, which can be set in different waveform generation modes like frequency generation, NCO Pulse Frequency, NCO Fixed Duty Cycle and 8-bit Pulse-Width Modulation (PWM). Similar to other timer/counter modules, it can generate interrupts or events based on specific conditions like timer overflows or compare matches.

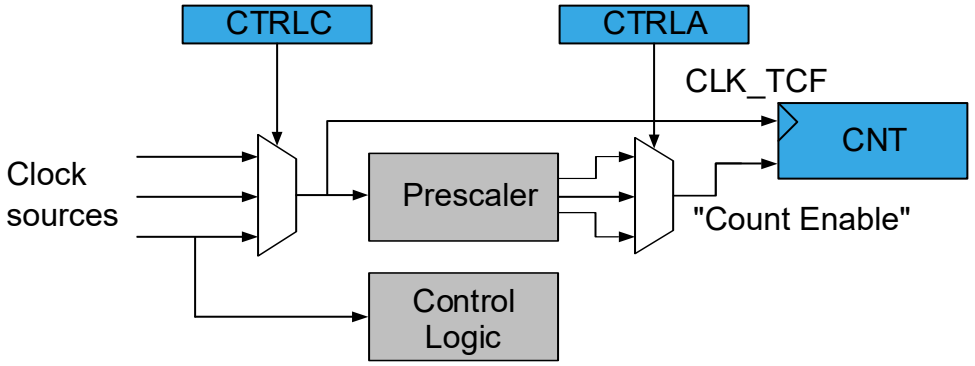
Figure 2-1. Block Diagram



The timer/counter can be clocked from the multiple clock sources, including Peripheral Clock (CLK_PER), internal oscillators, and Event System (EVSYS).

The Clock Select (CLKSEL) bit field in the Control B (TCFn.CTRLB) register selects one of the clock sources used as an input for the prescaler. The selected clock source can be divided by a maximum of 128 using the 7-bit prescaler.

Figure 2-2. Clock Selection



Using TCF in Frequency Generation Mode

In Frequency Waveform Generation mode, the CMP register controls the period time (T). The corresponding Waveform Generator output is toggled for each compare match between the CNT and CMP registers. The functionality is similar to what TCA and TCB timers/counters have, but the TCF has a 24-bit wide count register, instead of 16-bit width.

Using TCF in 8-bit PWM Mode

The TCF can be configured to run in 8-bit PWM mode where the register pairs in the lowest 16-bit Compare register (CMP0 and CMP1) are used as individual compare registers. The 8-bit PWM functionality is a helpful addition to the TCF. The TCF 8-bit PWM output is single-slope only.

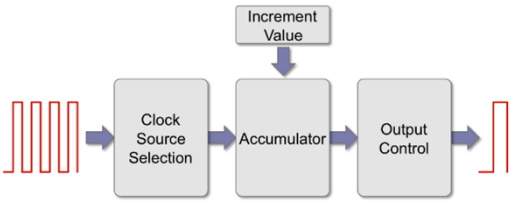
Using TCF in NCO Pulse-Frequency Mode

The NCO mode uses the overflow of an accumulator to create an output signal.

The accumulator overflow is controlled by an adjustable increment value rather than a single clock pulse or postscaler increment, offering an advantage over a simple timer-driven counter because the division resolution does not vary with the limited prescaler/postscaler divider value. The NCO is most useful for applications that require frequency accuracy and high resolution at a fixed duty cycle.

The NCO operates by repeatedly adding a fixed value to an accumulator. Additions occur at the input clock rate. The accumulator will overflow with a carry periodically, the raw NCO output. This reduces the input clock by the added value ratio to the maximum accumulator value.

Figure 2-3. NCO Logic



The NCO output can be modified by stretching the pulse or toggling a flip-flop. Then, the modified NCO output is distributed to other peripherals and optionally output to an I/O pin. The accumulator overflow can also generate an interrupt. The NCO period changes in discrete steps to create an average frequency. This output depends on the ability of the receiving circuit to average the NCO output to reduce uncertainty.

The waveform frequency (f_{FRQ}) is defined by Equation 1:

Note: The prescaler (N) is disabled in this mode.

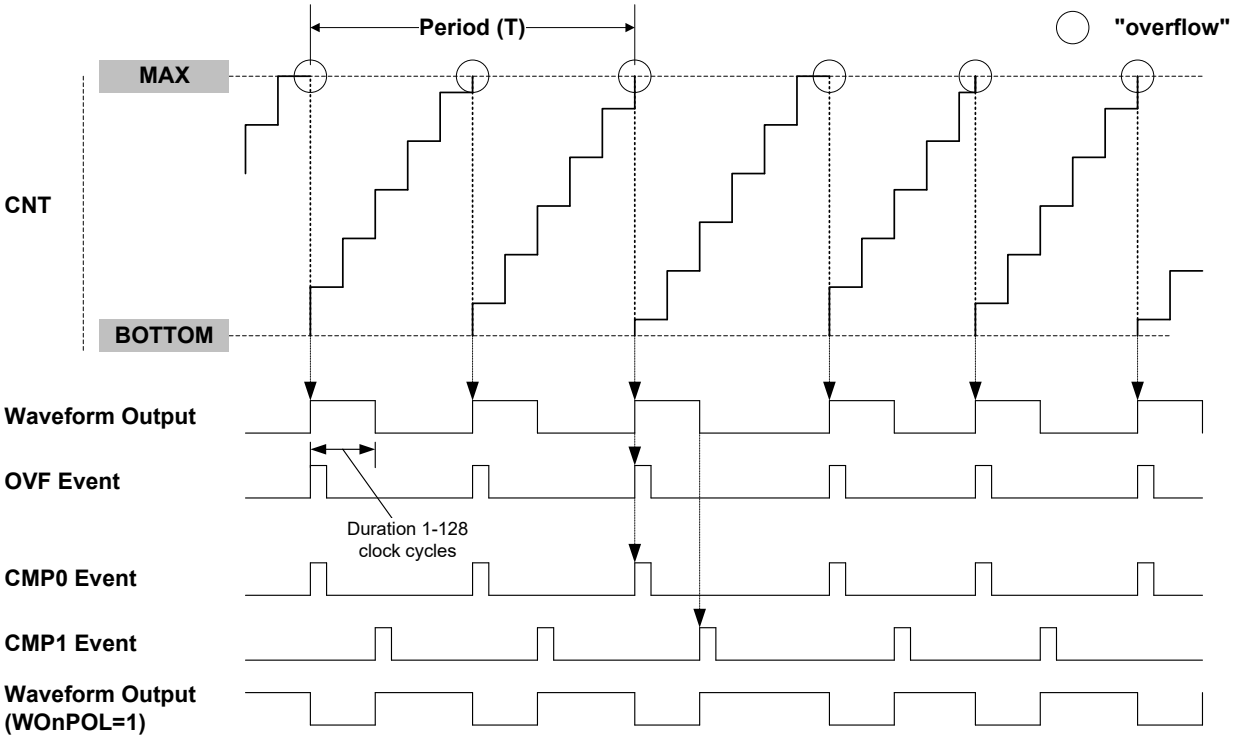
Equation 1. FRQ Frequency

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

In this mode, the output becomes active on the clock rising edge immediately following the overflow event, and goes inactive from 1 to 128 clock periods later, determined by the WGPULSE bit field in TCFn.CTRLB, giving a waveform output at the frequency.

The Compare-Match 0 (CMP0) event is generated at every start of the waveform pulse, while the Compare-Match 1 (CMP1) event is generated when the pulse ends. The interrupt flags are set at the same time as the pulse events.

Figure 2-4. NCO Pulse-Frequency Waveform Generation



Using TCF in the NCO Fixed Duty-Cycle Frequency Waveform Generation

In NCO Fixed Duty Cycle Frequency Waveform Generation mode, the TCF operates like in NCO PWM mode, but the output is toggled every time the accumulator overflows.

Given that the increment value remains constant, this provides a 50% duty cycle at half the frequency for the Pulse-Frequency mode.

The CMP0 and CMP1 events are generated at alternating overflows.

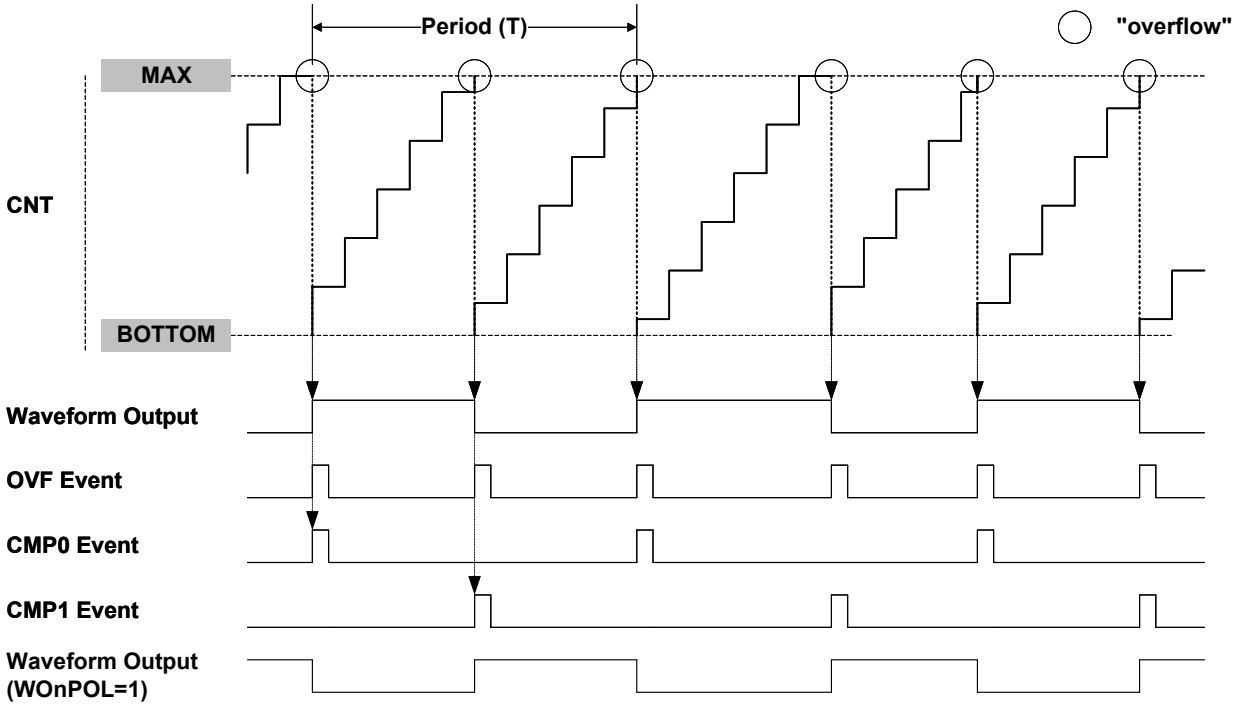
The interrupt flags are set at the same time as the pulse events, as shown in Equation 2.

Note: N represents the prescaler.

Equation 2. FRQ Frequency

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

Figure 2-5. NCO Fixed Duty-Cycle Frequency Waveform Generation



3. Generate Two Fixed Frequency PWM Signals with Variable Duty Cycle

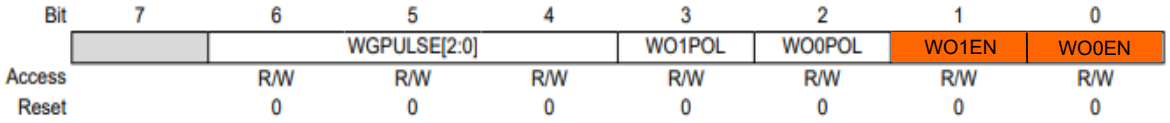
Use case description: Configure the TCF to generate an overflow event on the compare registers at a fixed frequency while increasing the waveform duty cycle to all available steps.

Result: The TCF will generate an output on the Channel 0 and Channel 1 at a fixed frequency, while the waveform duty cycle will increase.

Embedded Bare Metal Implementation

1. **Configuring the Output Signals**
   Enable Waveform Output 0 and Waveform Output 1.

Figure 3-1. CTRLC Register




```
The CTRLC register is double buffered. It is recommended to wait in a loop for bit
CTRLCBUSY in the Status register to be cleared every time the CTRLC register is written.
Create a function called TCF0_OutputsSet that returns void and takes an argument of type
uint8_t.
void TCF0_OutputsSet(uint8_t value)
    {
        while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
        TCF0.CTRLC = value;
    }
```

2. **Configuring the TCF Clock**
   The AVR16EB32 board runs default with a system clock of 3.33 MHz. For the TCF to run at 20 MHz , the system-clock prescaler must be disabled.

```
Create a function take returns void and take an argument of type void.
void CLOCK_Initialize(void)
{
    _PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, 0x0);
}
```

Figure 3-2. CTRLB Register

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

Figure 3-3. Clock Select Bit Mask

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

The TCF provides a range of clock options to choose from. For this application select the default option, which is the Peripheral Clock running at 20 MHz. The CTRLB register controls the clock selection.

```
Create a function that returns void and takes an argument of type TCF_CLKSEL_t.
void TCF0_ClockSet(TCF_CLKSEL_t config)
{
    TCF0.CTRLB |= config;
}
```

The Prescaler is disabled in this mode.

3. **Configuring the Waveform Generation Mode**

The mode in which the timer works is selected using the CTRLB register. NCOPF will be selected.

Figure 3-4. Waveform Generation Mode Bit Mask

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

```
Create a function that returns void and takes an argument of type TCF_WGMODE_t.
void TCF0_ModeSet(TCF_WGMODE_t mode)
{
    TCF0.CTRLB |= mode;
}
```

4. **Setting the Frequency**

After configuring the TCF clock, choose the frequency at which the timer will operate - 125 kHz using the CMP register.

Figure 3-5. CMP Register

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

Using Equation 1, the Increment can be calculated as follows:

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

Using the desired values results in:

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

The hexadecimal result is 0x0001999A.

```
The above formula cand be translated into the formula:
#define TCF0_NCOPL_HZ_TO_INCREMENT(HZ, F_CLOCK)    (uint32_t)(((float)(HZ) * 16777216.0) /
(float)(F_CLOCK) + 0.5)
```

```
The CMP register is double buffered. It is recommended to wait in a loop for bit CMP0BUSY
in the Status register to be cleared every time the CMP register is written.
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. Configuring the Waveform Generation Pulse Length

Figure 3-6. CTRLC Register

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

Figure 3-7. Waveform Generation Pulse Length Bit Mask

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

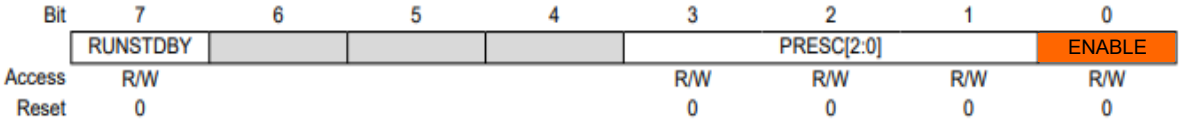
Set the pulse length to be a single clock increment, which is 50 ns. 1 clock cycle takes 50 ns beacuse the timer is running at 20 MHz. 1 divided by 20 MHz equals 50 ns.

```
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. Start and Stop the Timer

Figure 3-8. CTRLA Register




```
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;
}
```

The TCF will be initialized with the following settings:

- Both outputs enabled
- Peripheral clock selected
- NCO Pulse-Length mode set
- Set a frequency of 125 kHz
- Set a waveform generation pulse length of 1 clock cycle
- Set the timer to start counting from 0

```
Create a function that returns void and takes an argument of type void.
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);
}
```

```
The util/delay.h header file requires to define the F_CPU frequency.
#define F_CPU 20000000UL
Include the util/delay.h header file
#include <util/delay.h>
```

```
Create a function that returns void and takes an argument of type void. Use the
TCF0_NCO_Pulse_Length_Demo function to change the pulse-length with a delay in-between the
changes.
void NCO_Pulse_Length_Demo(void)
{
    /* Configure the TCF to start counting from 0 */
    TCF0_CounterSet(0);

    /* Enable the TCF */
    TCF0_Start();

    /* Delay for 20 us */
   _delay_us(20);
```

```
    /* Configure the pulse-length to 2 clock cycles */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK2_gc);

    /* Delay for 20 us */
    _delay_us(20);

    /* Configure the pulse-length to 4 clock cycles */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK4_gc);

    /* Delay for 20 us */
    _delay_us(20);

    /* Configure the pulse-length to 8 clock cycles */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK8_gc);

    /* Delay for 20 us */
    _delay_us(20);

    /* Configure the pulse-length to 6 clock cycles */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK16_gc);

    /* Delay for 20 us */
    _delay_us(20);

    /* Configure the pulse-length to 32 clock cycles */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK32_gc);

    /* Delay for 20 us */
    _delay_us(20);

    /* Configure the pulse-length to 64 clock cycles */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK64_gc);

    /* Delay for 20 us */
    _delay_us(20);

    /* Configure the pulse-length to 128 clock cycles */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK128_gc);

    /* Delay for 25 us */
    _delay_us(25);

    /* Stop the timer */
    TCF0_Stop();

    /* Configure the pulse-length to 1 clock cycle */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK1_gc);
}
```

The main function looks like this:

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

MCC Melody Implementation

To generate this project using MPLAB Code Configurator, MCC Melody (MCC Classic is not supported), follow the next steps:

1. Create a new MPLAB® X IDE project for AVR16EB32.
2. Open MCC from the toolbar find more information on installing the MCC plug-in here.
3. In **MCC Content Manager Wizard**, select **MCC Melody**, then click **Finish.**
4. From **Device Resources**, go to **System**, click the **CLCKCTRL** window and disable the Prescaler.
5. From **Device Resource**, go to **Drivers**, click the Timer window, add the TCF module, then do the following configuration:
   – Clock Divider: System clock (by default, the divider will be 1 - System clock)
   – Waveform Generation Mode: NCO Pulse-Length mode
   – Waveform Generation Pulse Length: 1 Clock Period
   – Requested Period[s]: 0.000008
   – Waveform Output n: check the boxes from the Enable column for Waveform Output 0 and Waveform Output 1
6. Select the PA0 and PA1 pins in the **Pin Grid View**. When the boxes from Enable column and Waveform Output n are checked, the pins are also locked. To change the PORT, click on a pin from another PORT in **Pin Grid View**.
7. In the **Project Resources** window, click the **Generate** button so that MCC will generate all the specified drivers and configurations.
8. Edit the main.c file,as following:
   Include the util/delay.h header file.

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

Create a function called NCO_Pulse_Length_Demo. Use the TCF0_NCO_PulseLengthSet function to change the pulse length.

The code is as follows:

```
void NCO_Pulse_Length_Demo(void)
{
     /* Configure the TCF to start counting from 0 */
    TCF0_CounterSet(0);

    /* Enable the TCF */
    TCF0_Start();

    /* Delay for 20 us */
   _delay_us(20);

   /* Configure the pulse-length to 2 clock cycles */
   TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK2_gc);

   /* Delay for 20 us */
   _delay_us(20);

   /* Configure the pulse-length to 4 clock cycles */
   TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK4_gc);

   /* Delay for 20 us */
   _delay_us(20);

   /* Configure the pulse-length to 8 clock cycles */
   TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK8_gc);

   /* Delay for 20 us */
   _delay_us(20);

   /* Configure the pulse-length to 6 clock cycles */
   TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK16_gc);
```

```
    /* Delay for 20 us */
    _delay_us(20);

    /* Configure the pulse-length to 32 clock cycles */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK32_gc);

    /* Delay for 20 us */
    _delay_us(20);

    /* Configure the pulse-length to 64 clock cycles */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK64_gc);

    /* Delay for 20 us */
    _delay_us(20);

    /* Configure the pulse-length to 128 clock cycles */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK128_gc);

    /* Delay for 25 us */
    _delay_us(25);

    /* Stop the timer */
    TCF0_Stop();

    /* Configure the pulse-length to 1 clock cycle */
    TCF0_NCO_PulseLengthSet(TCF_WGPULSE_CLK1_gc);
}
```

9. The main function looks like this:

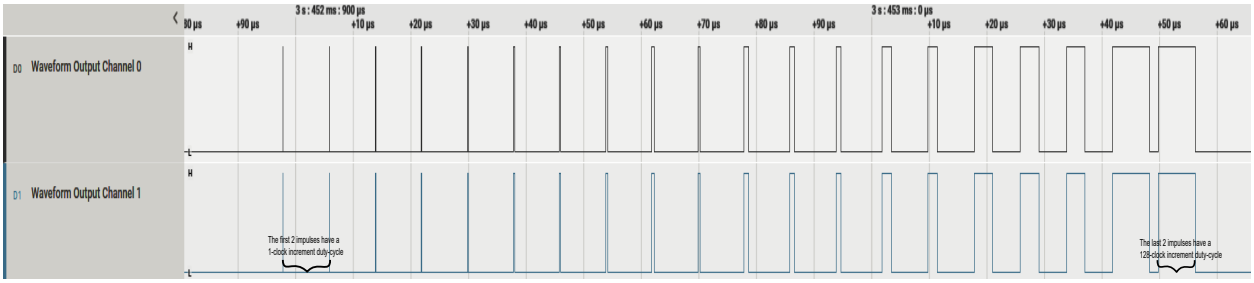
```
int main(void)
{
    SYSTEM_Initialize();

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

10. Flash the project.
11. Result:

TCF generates two identical signals with a frequency of 125 kHz and PWM with a variable duration ranging from 1 to 128 clock cycles. In this case, one clock cycle takes 50 ns.

Figure 3-9. Result



Click to view code examples on MPLAB DISCOVER

4. Generate Two Variable-Frequency Signals in NCO Fixed Duty Cycle Waveform Generation

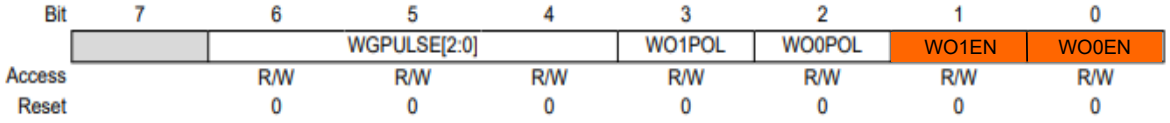
Use case description: Configure TCF to generate an overflow event on the compare registers on a range of frequencies, from 10 Hz to 100 kHz.

Result: TCF will generate an output on Channel 0 and Channel 1 on a range of frequencies, which toggles every time the accumulator overflows.

Embedded Bare Metal Implementation

1. **Configuring the Output Signals**
   Enable Waveform Output 0 and Waveform Output 1.

Figure 4-1. CTRLC Register




```
The CTRLC register is double buffered. It is recommended to wait in a loop for bit
CTRLCBUSY in the Status register to be cleared every time the CTRLC register is written.
Create a function called TCF0_OutputsSet that returns void and takes an argument of type
uint8_t.
void TCF0_OutputsSet(uint8_t value)
    {
        while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
        TCF0.CTRLC = value;
    }
```

2. **Configuring the TCF Clock**
   The AVR16EB32 board runs default with a system clock of 3.33 MHz. For the TCF to run at 20 MHz , the system-clock prescaler must be disabled.

```
Create a function take returns void and take an argument of type void.void
CLOCK_Initialize(void)
{
    _PROTECTED_WRITE(CLKCTRL.MCLKCTRLB, 0x0);
}
```

Figure 4-2. CTRLB Register

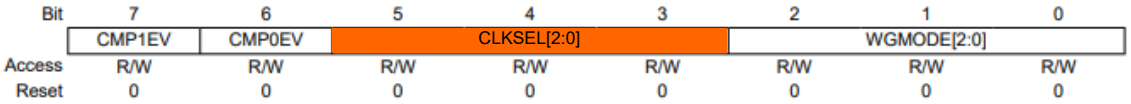


Figure 4-3. Clock Select Bit Mask

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.

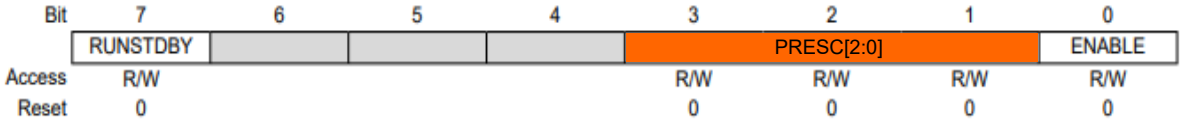
Value	Name	Description
0x0	CLKPER	Peripheral clock
0x1	EVENT	Event edge
0x2	OSCHF	Internal high-frequency oscillator
0x3	OSC32K	Internal 32.768 kHz oscillator
0x4	XOSCHF	High-frequency crystal oscillator
0x5	PLL	Phase Locked Loop
Others	-	Reserved

The TCF provides a range of clock options to choose from. For this application, select the default option, which is the Peripheral Clock running at 20 MHz. The CTRLB register controls the clock selection.

```
Create a function that returns void and takes an argument of type TCF_CLKSEL_t.
void TCF0_ClockSet(TCF_CLKSEL_t config)
{
    TCF0.CTRLB |= config;
}
```

3. **Selecting the Prescaler**

Figure 4-4. CTRLA Register




```
The CTRLA register is double buffered. It is recommended to wait in a loop for bit
CTRLABUSY in the Status register to be cleared every time the CTRLA register is written.
Create a function that returns void and takes an argumnet of type TCF_PRESC_t.
void TCF0_PrescalerSet(TCF_PRESC_t config)
{
    while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
    TCF0.CTRLA  |= config;
}
```

4. **Configuring the Waveform Generation Mode**

The mode in which the timer works is selected using the CTRLB register. NCOFDC will be selected.

Figure 4-5. CTRLB Register

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

Figure 4-6. Waveform Generation Mode Bit Mask

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

```
Create a function that returns void and takes an argument of type TCF_WGMODE_t.
void TCF0_ModeSet(TCF_WGMODE_t mode)
{
    TCF0.CTRLB |= mode;
}
```

5. **Setting the CMP Register**

Using Equation 2, the Increment can be calculated as follows:

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

Using the desired values results in:

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

The result is hexadecimal, 0x00000011.

```
The above formula cand be translated into the formula:
#define TCF0_NCOFD_HZ_TO_INCREMENT(HZ, F_CLOCK, TCF0_PRESCALER)    (uint32_t)(((float)(HZ)
* 33554432.0 * (TCF0_PRESCALER)) / ((float)(F_CLOCK)) + 0.5)
```

Before the timer starts, the CMP is written with the value of 11.

Figure 4-7. CMP Register

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

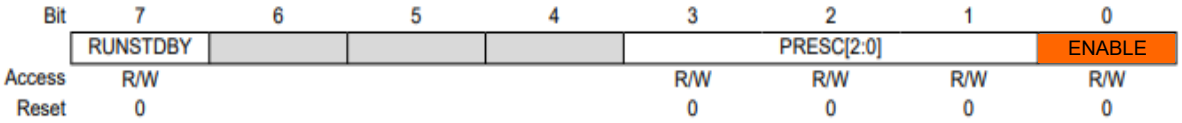
```
The CMP register is double buffered. It is recommended to wait in a loop for bit CMP0BUSY
in the Status register to be cleared every time the CMP register is written.
Create a function that returns void and takes an argument of type uint32_t.
void TCF0_CompareSet(uint32_t value)
{
    while((TCF0.STATUS & TCF_CMP0BUSY_bm) != 0){};
    TCF0.CMP = value;
}
```

```
The CNT register is double buffered. It is recommended to wait in a loop for bit CNTBUSY
in the Status register to be cleared every time the CNT register is written.
Create a function that returns void and takes an argument of type uint32_t.
void TCF0_CounterSet(uint32_t value)
{
    while((TCF0.STATUS & TCF_CNTBUSY_bm) != 0){};
```

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

6. Start and Stop the Timer

Figure 4-8. CTRLA Register




```
The CTRLA register is double buffered. It is recommended to wait in a loop for bit
CTRLABUSY in the Status register to be cleared every time the CTRLA register is written.
Create a function that returns void and takes an argument of type void.
void TCF0_Start(void)
{
    while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
    TCF0.CTRLA |=  TCF_ENABLE_bm;
}

Create a function that returns void and takes an argument of type void.
void TCF0_Stop(void)
{
    while((TCF0.STATUS & TCF_CTRLABUSY_bm) != 0){};
    TCF0.CTRLA &= ~TCF_ENABLE_bm;
}
```

The TCF will be initialized with the following settings:

- Both outputs enabled
- Peripheral clock selected
- NCO Fixed Duty Cycle mode set
- Set a waveform generation pulse length of 1 clock cycle
- Set a frequency of 10 Hz
- Set the timer to start counting from 0

```
Create a function that returns void and takes an argument of type void.
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));
}
```

```
The util/delay.h header file requires to define the F_CPU frequency.
#define F_CPU 20000000UL
Include the util/delay.h header file
#include <util/delay.h>
```

```
Create a function that returns void and takes an argument of type uint_32t. Use the
TCF0_CompareSet function to change the frequency.
void NCO_Fixed_DutyCycle_Demo(void)
{
    /* Configure the TCF to start counting from 0 */
    TCF0_CounterSet(0);

    /* Enable the TCF */
    TCF0_Start();

    /* Delay for 600 ms */
```

```
    _delay_ms(600);

    /* Load the CMP register with a frequency of 100 Hz */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(100, 20000000, 1));

     /* Delay for 60 ms */
    _delay_ms(60);

    /* Load the CMP register with a frequency of 1 KHz */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(1000, 20000000, 1));

     /* Delay for 6 ms */
    _delay_ms(6);

    /* Load the CMP register with a frequency of 10 KHz */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(10000, 20000000, 1));

     /* Delay for 600 us */
    _delay_us(600);

    /* Load the CMP register with a frequency of 100 KHz */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(100000, 20000000, 1));

     /* Delay for 60 us */
    _delay_us(60);

     /* Stop the TCF */
    TCF0_Stop();

    /* Load the CMP register with a frequency of 10 Hz */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(10, 20000000, 1));
}
```

The main function looks this:

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

MCC Melody Implementation

To generate this project using MPLAB Code Configurator, MCC Melody (MCC Classic is not supported), follow the next steps:

1. Create a new MPLAB X IDE project for AVR16EB32.
2. Open MCC from the toolbar find more information on installing the MCC plug-in here.
3. In **MCC Content Manager Wizard**, select **MCC Melody**, then click **Finish.**
4. From **Device Resources** go to **System**, click the **CLCKCTRL** window and disable the Prescaler.
5. From **Device Resource**, go to **Drivers**, click the Timer window, add the TCF module, then do the following configuration:
   – Clock Divider: System clock (by default, the divider should be 1 - System clock
   – Waveform Generation Mode: NCO Fixed Duty-Cycle mode
   – Requested Period[s]: 0.1
   – Waveform Output n: check the boxes from the Enable column for Waveform Output 0 and Waveform Output 1
6. Select the PA0 and PA1 pins in the **Pin Grid View**. When the boxes from Enable column from Waveform Output n are checked, the pins are also locked. To change the PORT, click on a pin from another PORT in **Pin Grid View**.
7. In the **Project Resources** window, click the **Generate** button so that MCC will generate all the specified drivers and configurations.
8. Edit the main.c file, as following:
   Include the util/delay.h header file.

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

Create a function called NCO_Fixed_DutyCycle_Demo. Use the TCF0_CompareSet function to change the frequency.

The code is as follows:

```
Create a function that retuns void and takes an argument of type void.
void NCO_Fixed_DutyCycle_Demo(void)
{
    /* Configure the TCF to start counting from 0 */
    TCF0_CounterSet(0);

    /* Enable the TCF */
    TCF0_Start();

    /* Delay for 600 ms */
   _delay_ms(600);

   /* Load the CMP register with a frequency of 100 Hz */
   TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(100, 20000000, 1));

    /* Delay for 60 ms */
   _delay_ms(60);

   /* Load the CMP register with a frequency of 1 KHz */
   TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(1000, 20000000, 1));

    /* Delay for 6 ms */
   _delay_ms(6);

   /* Load the CMP register with a frequency of 10 KHz */
   TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(10000, 20000000, 1));

    /* Delay for 600 us */
   _delay_us(600);

   /* Load the CMP register with a frequency of 100 KHz */
   TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(100000, 20000000, 1));
```

```
    /* Delay for 60 us */
    _delay_us(60);

    /* Stop the TCF */
    TCF0_Stop();

    /* Load the CMP register with a frequency of 10 Hz */
    TCF0_CompareSet(TCF0_NCOFD_HZ_TO_INCREMENT(10, 20000000, 1));
}
```

9. The main.c file looks like this:

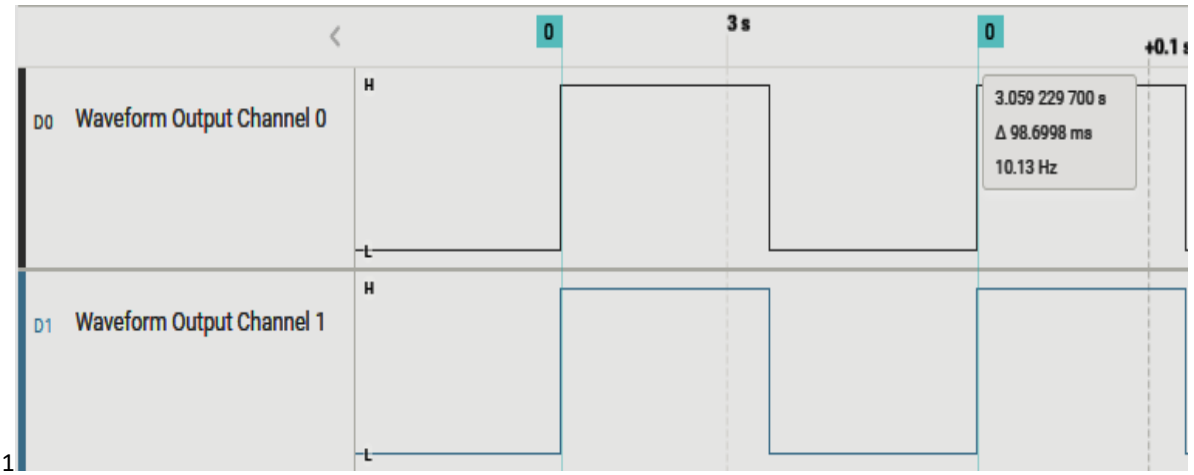
```
int main(void)
{
    SYSTEM_Initialize();

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

10. Flash the project.

Result 1: two identical signals are generated with a frequency of 10 Hz and duty cycle of 50%.

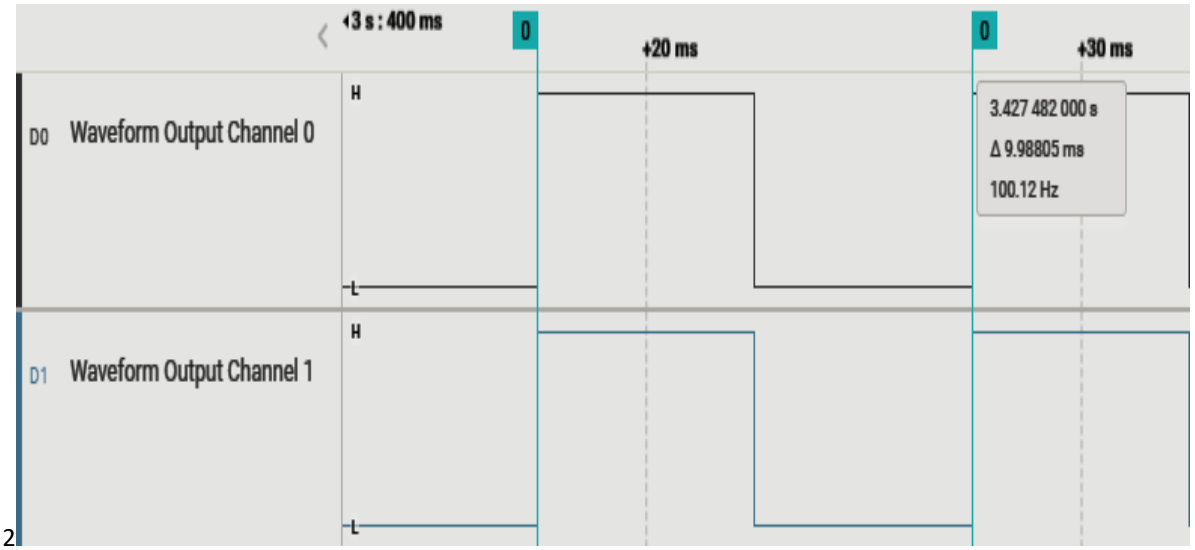
Figure 4-9. Result



1

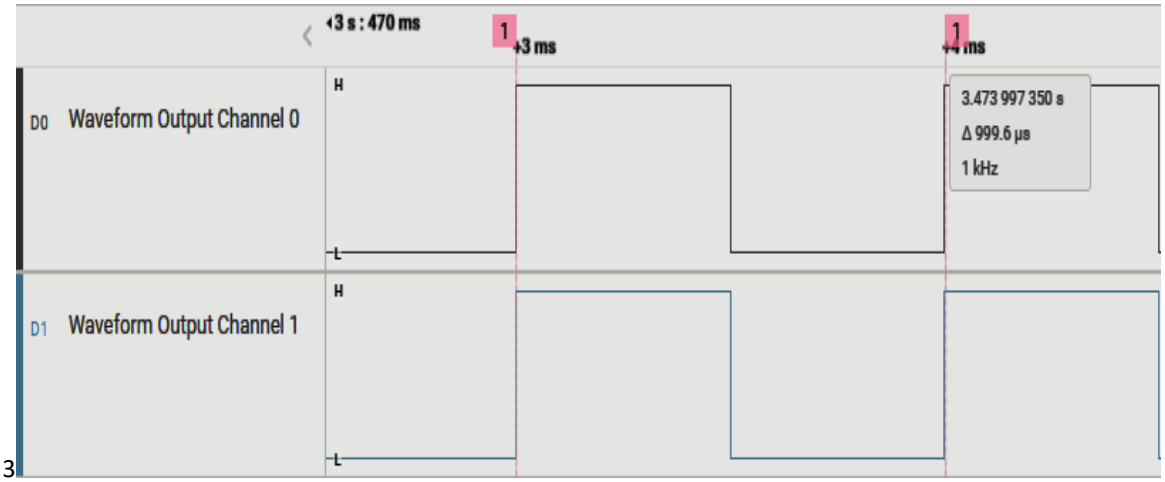
Result 2: two identical signals are generated with a frequency of 100 Hz and duty cycle of 50%.

Figure 4-10. Result



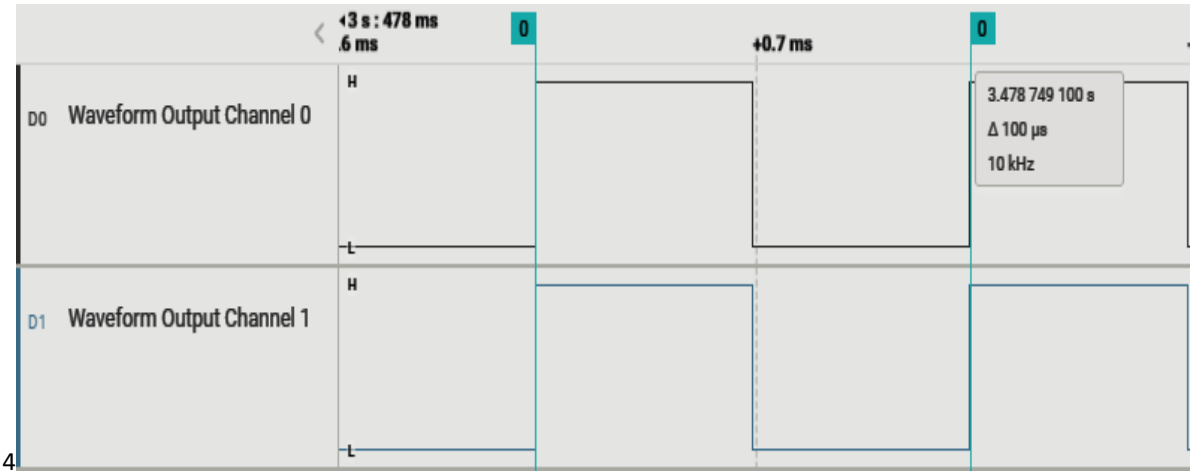
Result 3: two identical signals are generated with a frequency of 1 kHz and duty cycle of 50%.

Figure 4-11. Result



Result 4: two identical signals are generated with a frequency of 10 kHz and duty cycle of 50%.

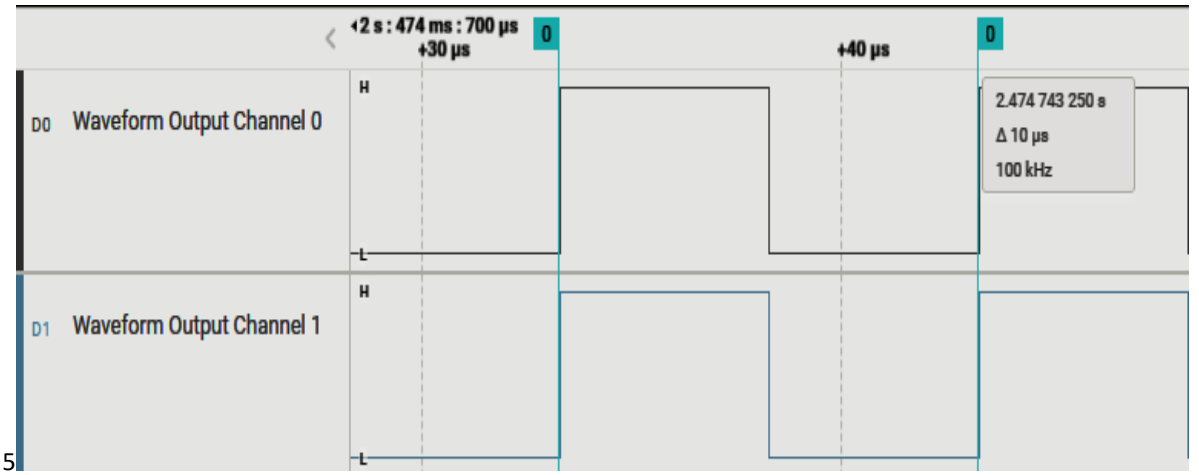
Figure 4-12. Result



4

Result 5: two identical signals are generated with a frequency of 100 kHz and duty cycle of 50%.

Figure 4-13. Result



5

Click to view code examples on MPLAB DISCOVER

5. References

Use the following links for more information about the TCF operation modes.

These links are incorrect. They must be updated once the web pages will actually exist.

1. AVR16EB32 Product Page
2. AVR16EB32 Curiosity Nano Evaluation Kit
3. AVR16EB16/20/28/32 AVR® EB Family Data Sheet

6. Revision History

Document Revision	Date	Comments
A	10/2023	Initial document release

Microchip Information

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip product is strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable". Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure

that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, CryptoMemory, CryptoRF, dsPIC, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, TimeCesium, TimeHub, TimePictra, TimeProvider, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, Clockstudio, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, EyeOpen, GridTime, IdealBridge, IGaT, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, IntelliMOS, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, MarginLink, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, mSiC, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, Power MOS IV, Power MOS 7, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SmartHLS, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, Trusted Time, TSHARC, Turing, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

ISBN: 978-1-6683-3059-3

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office	**Australia - Sydney**	**India - Bangalore**	**Austria - Wels**
2355 West Chandler Blvd.	Tel: 61-2-9868-6733	Tel: 91-80-3090-4444	Tel: 43-7242-2244-39
Chandler, AZ 85224-6199	**China - Beijing**	**India - New Delhi**	Fax: 43-7242-2244-393
Tel: 480-792-7200	Tel: 86-10-8569-7000	Tel: 91-11-4160-8631	**Denmark - Copenhagen**
Fax: 480-792-7277	**China - Chengdu**	**India - Pune**	Tel: 45-4485-5910
Technical Support:	Tel: 86-28-8665-5511	Tel: 91-20-4121-0141	Fax: 45-4485-2829
www.microchip.com/support	**China - Chongqing**	**Japan - Osaka**	**Finland - Espoo**
Web Address:	Tel: 86-23-8980-9588	Tel: 81-6-6152-7160	Tel: 358-9-4520-820
www.microchip.com	**China - Dongguan**	**Japan - Tokyo**	**France - Paris**
Atlanta	Tel: 86-769-8702-9880	Tel: 81-3-6880- 3770	Tel: 33-1-69-53-63-20
Duluth, GA	**China - Guangzhou**	**Korea - Daegu**	Fax: 33-1-69-30-90-79
Tel: 678-957-9614	Tel: 86-20-8755-8029	Tel: 82-53-744-4301	**Germany - Garching**
Fax: 678-957-1455	**China - Hangzhou**	**Korea - Seoul**	Tel: 49-8931-9700
Austin, TX	Tel: 86-571-8792-8115	Tel: 82-2-554-7200	**Germany - Haan**
Tel: 512-257-3370	**China - Hong Kong SAR**	**Malaysia - Kuala Lumpur**	Tel: 49-2129-3766400
Boston	Tel: 852-2943-5100	Tel: 60-3-7651-7906	**Germany - Heilbronn**
Westborough, MA	**China - Nanjing**	**Malaysia - Penang**	Tel: 49-7131-72400
Tel: 774-760-0087	Tel: 86-25-8473-2460	Tel: 60-4-227-8870	**Germany - Karlsruhe**
Fax: 774-760-0088	**China - Qingdao**	**Philippines - Manila**	Tel: 49-721-625370
Chicago	Tel: 86-532-8502-7355	Tel: 63-2-634-9065	**Germany - Munich**
Itasca, IL	**China - Shanghai**	**Singapore**	Tel: 49-89-627-144-0
Tel: 630-285-0071	Tel: 86-21-3326-8000	Tel: 65-6334-8870	Fax: 49-89-627-144-44
Fax: 630-285-0075	**China - Shenyang**	**Taiwan - Hsin Chu**	**Germany - Rosenheim**
Dallas	Tel: 86-24-2334-2829	Tel: 886-3-577-8366	Tel: 49-8031-354-560
Addison, TX	**China - Shenzhen**	**Taiwan - Kaohsiung**	**Israel - Ra'anana**
Tel: 972-818-7423	Tel: 86-755-8864-2200	Tel: 886-7-213-7830	Tel: 972-9-744-7705
Fax: 972-818-2924	**China - Suzhou**	**Taiwan - Taipei**	**Italy - Milan**
Detroit	Tel: 86-186-6233-1526	Tel: 886-2-2508-8600	Tel: 39-0331-742611
Novi, MI	**China - Wuhan**	**Thailand - Bangkok**	Fax: 39-0331-466781
Tel: 248-848-4000	Tel: 86-27-5980-5300	Tel: 66-2-694-1351	**Italy - Padova**
Houston, TX	**China - Xian**	**Vietnam - Ho Chi Minh**	Tel: 39-049-7625286
Tel: 281-894-5983	Tel: 86-29-8833-7252	Tel: 84-28-5448-2100	**Netherlands - Drunen**
Indianapolis	**China - Xiamen**		Tel: 31-416-690399
Noblesville, IN	Tel: 86-592-2388138		Fax: 31-416-690340
Tel: 317-773-8323	**China - Zhuhai**		**Norway - Trondheim**
Fax: 317-773-5453	Tel: 86-756-3210040		Tel: 47-72884388
Tel: 317-536-2380			**Poland - Warsaw**
Los Angeles			Tel: 48-22-3325737
Mission Viejo, CA			**Romania - Bucharest**
Tel: 949-462-9523			Tel: 40-21-407-87-50
Fax: 949-462-9608			**Spain - Madrid**
Tel: 951-273-7800			Tel: 34-91-708-08-90
Raleigh, NC			Fax: 34-91-708-08-91
Tel: 919-844-7510			**Sweden - Gothenberg**
New York, NY			Tel: 46-31-704-60-40
Tel: 631-435-6000			**Sweden - Stockholm**
San Jose, CA			Tel: 46-8-5090-4654
Tel: 408-735-9110			**UK - Wokingham**
Tel: 408-436-4270			Tel: 44-118-921-5800
Canada - Toronto			Fax: 44-118-921-5820
Tel: 905-695-1980			
Fax: 905-695-2078			