Getting Started with the Timer/Counter Type E (TCE) and Waveform Extension (WEX) Peripherals

TB3339

Introduction

Author: Teodor Lina Microchip Technology Inc.

The AVR® EB family of microcontrollers uses powerful timers structured to cover several applications, from signal measurement to event synchronization and waveform generation. The Timer/Counter type E (TCE) is a 16-bit Timer/Counter peripheral that can generate up to four Pulse-Width Modulation (PWM) signals and has multiple waveform generation modes. The Waveform Extension (WEX) is a waveform extension peripheral that enhances the capabilities of the TCE. The TCE and WEX peripherals can generate up to eight PWM signals.

The capabilities of the 16-bit PWM TCE include accurate program execution timing, command execution, frequency and waveform generation. The TCE consists of a base counter and a set of compare channels. The base counter is used to count clock cycles and events or enable the events to count the clock cycles. The compare channels can be used with the base counter to perform a compare match control, frequency generation and pulse-width modulation.

The WEX enhances the capabilities of the 16-bit PWM TCE. This peripheral can produce complementary waveform output signals, generate a dead time between these signals, generate and manage Fault events, and override port outputs. The key feature is that the WEX overcomes common problems when using timers for applications such as motor control. With the dead time added, the signals are not overlapping, avoiding shoot-throughs caused by the transistor's switching. WEX can also make Fault conditions, driving all the signals into a user-defined state, a significant feature in applications where safety is crucial. Every functionality from WEX occurs inside the hardware, making the waveforms generated predictable in every situation.

This technical brief familiarizes the reader with several operating modes for the TCE and WEX, emphasizing the timer's particularities. For a better understanding of the functionality, refer to the data sheet.

The structure of the document covers three specific use cases:

- **Generate Four PWM Signals Using TCE Independently of WEX:**
  Initialize the TCE in Dual-Slope mode to generate four PWM signals with different duty cycles and scaling options.
- **Generate a Pattern for Eight Signals Using WEX:**
  Configure the WEX to generate a patterns succession of patterns for each output pin that toggles periodically.
- **Generate Eight Complementary Waveform Output Signals with Fault Detection Using TCE and WEX:**
  Initialize the TCE in Single Slope mode and generate four PWM signals. Configure the WEX to split the four PWM signals into eight complementary signals with dead time. Each PWM signal has a different duty cycle, between 0% and 100%. During each compare register, interrupt duty cycle increments happen in the Interrupt Service Routines (ISRs). A Fault event is triggered every 250 µs, driving all the outputs to low '0' logic. Then, the fault event is resolved, and the ordinary operation resumes. After that, the process repeats.

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

Bare metal code examples for AVR16EB32 with the same functionality as the ones described in this technical brief can be found here:

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

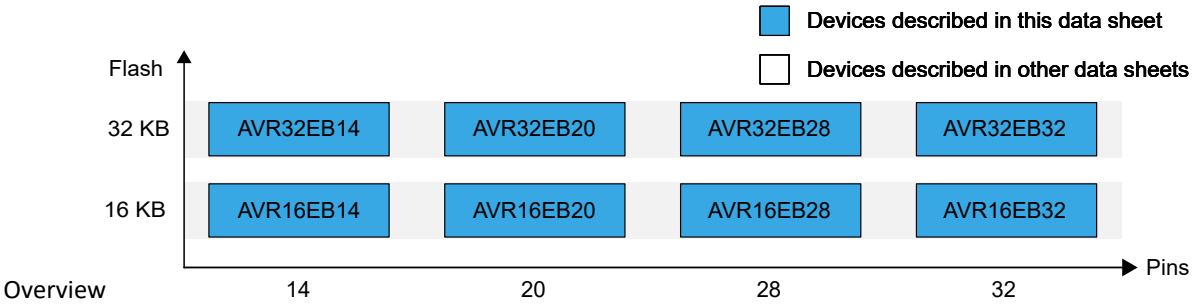
Table of Contents

1. Relevant Devices

This section lists the relevant devices for this document. The following figure shows the different 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 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

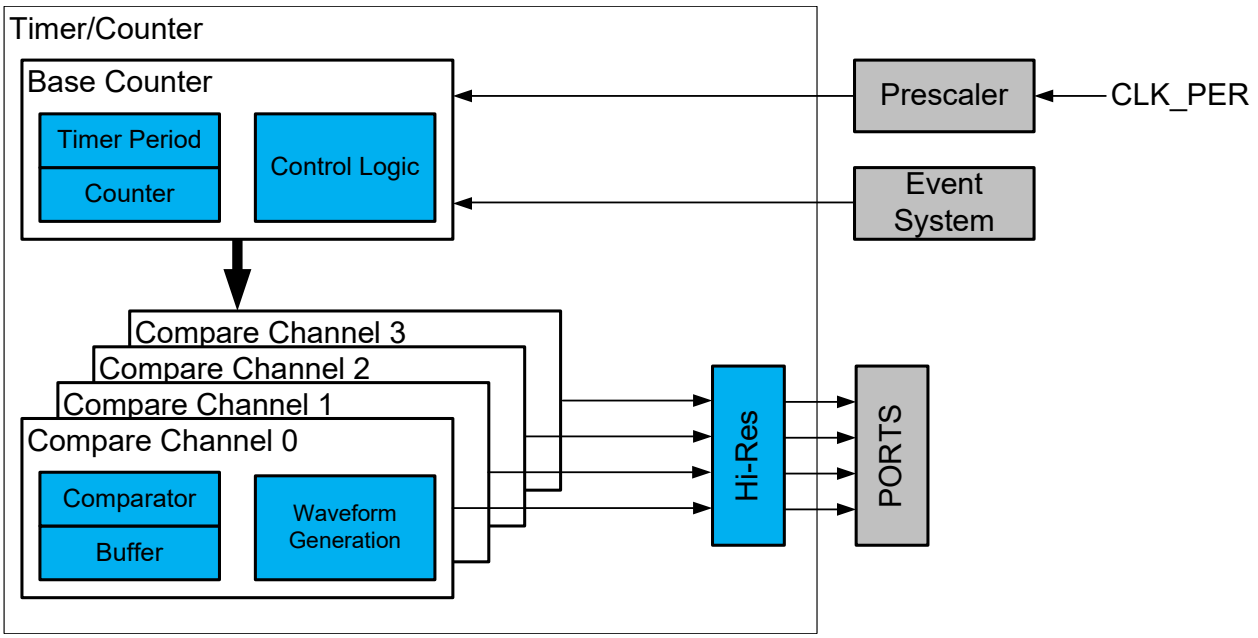
Timer/Counter Overview

The flexible 16-bit TCE provides accurate program execution timing, frequency and waveform generation, as well as command execution.

A TCE instance consists of a base counter and four compare channels. Based on the clock ticks (timer) or different events (counter), the user can set the base counter to count upwards or downwards. A timer/counter can be clocked and timed from the peripheral clock, with optional prescaling, or from the EVSYS. The EVSYS can also be used for direction control or operation synchronization.

The counter includes a high-resolution option which can increase the duty cycle resolution to eight times the input clock. Additionally, a prescaled peripheral clock and events from the EVSYS can be used to control the counter.

Figure 2-1. Timer/Counter Block Diagram



The counter value is continuously compared to zero and the period (PER) value to determine when the counter has reached BOTTOM or TOP, respectively. If one of the conditions is met, the counter is updated, and an interrupt is generated. The counter is also compared with the Compare registers. The counter uses these comparisons to generate interrupts and set the waveform and period of the pulse-width, if a Waveform Generation mode is selected.

The Counter, Period, and Compare registers and all their buffers are 16-bits wide. The buffers are part of a scheme that ensures the respective registers are only updated when the Counter register is updated. Each buffer has a Buffer Valid (BV) bit used to determine if the respective register requires updates.

The TCE can be configured to use the EVSYS to count the rising and/or falling edges of the event signal or use it to enable clock tick counting.

TCE offers the option of writing a fractional value between 0 and 2 to the compare registers if the SCALEMODE option is enabled. The compare value is calculated based on the period, amplitude and offset values.

Waveform Extension Overview

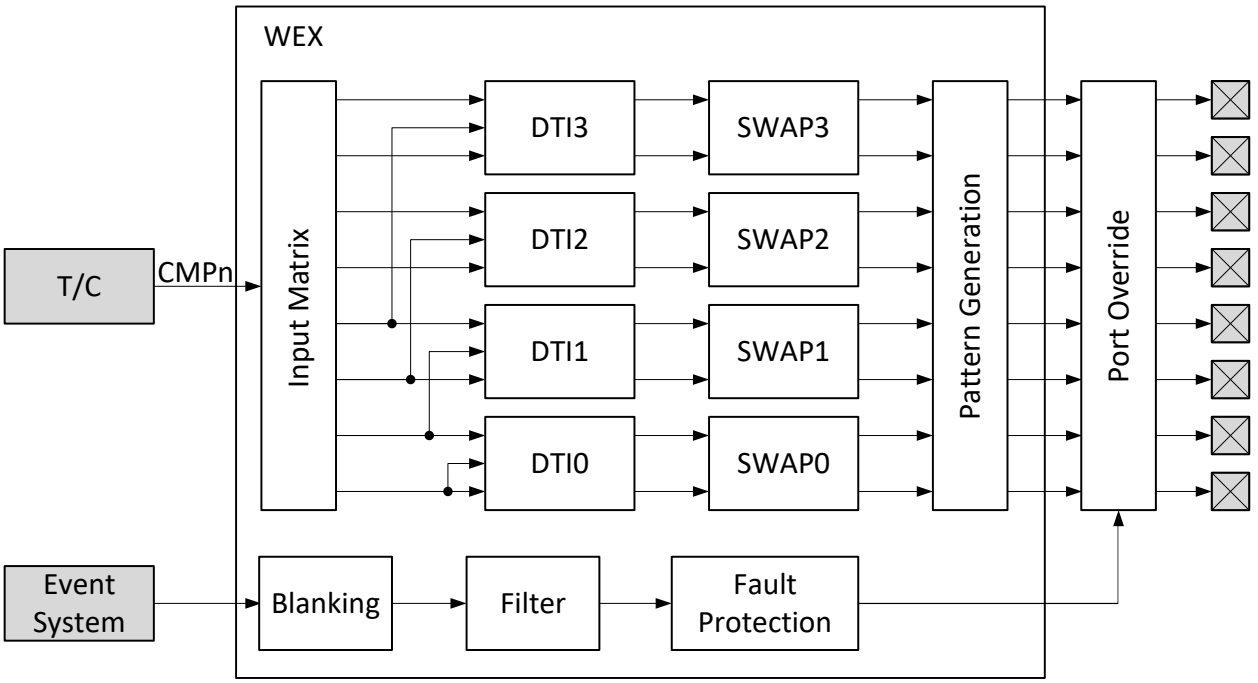
The WEX provides extra functions to the timer/counter in Waveform Generation (WG) modes, such as complementary signal generation with dead time. These functions are helpful for motor control and power control applications.

A WEX instance consists of five independent units:

- Input matrix
- Dead time insertion unit
- Swap unit
- Pattern generation unit
- Fault protection unit

The user can route out the waveform outputs from the timer/counter to the port pins in different configurations when using the input matrix unit. The dead-time insertion unit can split the timer's four outputs into two non-overlapping signals, low-side and high-side. The WEX can make eight PWM output signals from the four PWM output signals of the timer. The low-side and high-side pin positions can be swapped using the swap unit, an important feature for fast motor decay. The pattern generation unit can override the port pins with a constant logic level, and may be used independently of the timer/counter. The user can configure a pattern for a maximum of eight pins. The Fault protection unit can set the WEX outputs to a defined state when a Fault event is detected. The Fault protection unit is connected to the EVSYS, enabling the events to trigger faults.

Figure 2-2. Waveform Extension Block Diagram



3. Generate PWM Signals Using TCE

A TCE key characteristic, when compared to timers such as TCB or TCA, is the versatility and precision of the PWM generation. The user can choose from various configurations according to the complexity of the application. The TCE can be configured in both Single-Slope and Dual-Slope PWM Generation modes, which allows the trade-off between a constant phase, Correct Phase PWM, and a higher maximum operation frequency, Fast PWM. Also, the TCE has a buffering scheme that ensures a glitch-free PWM. Another feature of the TCE is the possibility of scaling the PWM signal's duty cycle using amplitude and offset.

The TCE and TCB can both be used to generate a PWM signal with a high maximum operating frequency. Only the TCE can be used in critical applications due to its dual-slope PWM capabilities based on its selectable direction. Dual-slope PWM does not modify the pulse center position when the duty cycle is changed. Thus, the phase is always constant. This feature is essential for motor control applications because it will avoid the switching noise caused by the simultaneous commutation of multiple PWM signals. The noise comes from transistors switching. Using dual-slope PWM signals with different duty cycles can prevent this noise type.

The buffering scheme contains a buffer for each Compare and Period register. It is essential to use these buffers in critical applications where an unexpected long pulse can lead to a short circuit. Moreover, the buffer presence can prevent the loss of synchronization between two peripherals that use the same timer but different compare channels. However, given the fact that the Period and Compare registers can be updated directly, the buffering scheme can be avoided by the user. The following waveforms illustrate the difference between the buffered and unbuffered operations.

Figure 3-1. Unbuffered Dual-Slope Operation

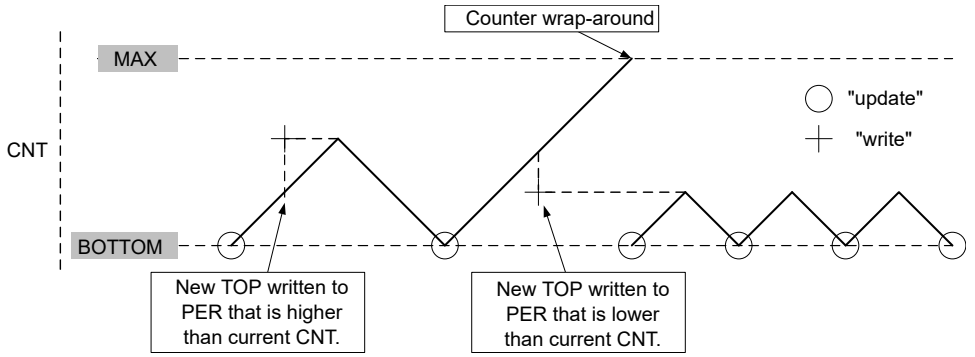
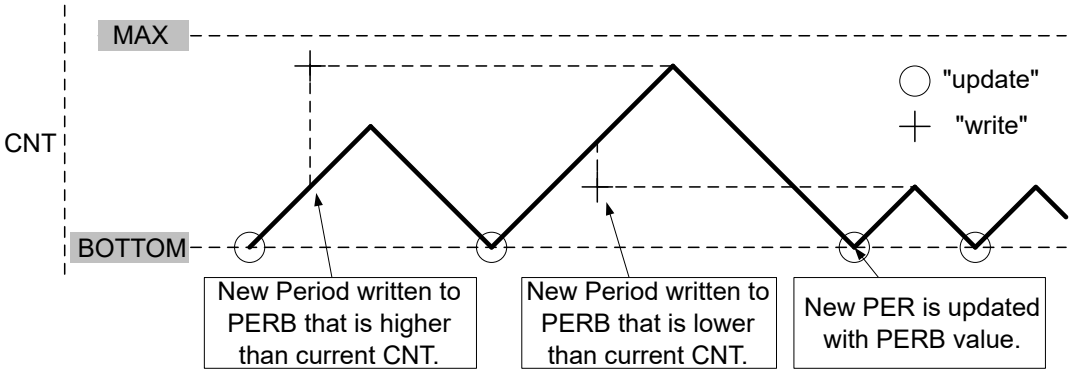


Figure 3-2. Changing the Period Using Buffering



If the user changes the Period register directly (unbuffered operation), the timer may already have passed the new threshold, so it will continue counting to the maximum value. That will cause an unusually long pulse that can cause further problems. Also, if using two or more compare channels and one is updated, the sync between the triggers may be lost. To prevent all these possible problems, using the buffering scheme is required. The buffers hold the new value and transfer it to the Compare or the Period register accordingly when the timer is updated. With all values changed at the same time, the problems mentioned disappear.

Below is an example of how to set a TCE instance to generate four PWM signals at 10 kHz with 20%, 40%, 60% and 80% duty cycles using the buffering scheme described above. The values scaling feature set in compare registers and the high-resolution feature are highlighted in this example. The user can increase the PWM signal resolution up to three bits.

In this example, the user can change the maximum range of the duty cycles from 0-100% to 0-75%, 0-50% or 0-150%. The duty cycles are calculated based on the period value. The period will always be 100%, the maximum value of the duty cycle. All the duty cycles over 100% will be saturated to 100%, which is the value of the period. When the user selects the range of duty cycles to 0-50% of the period, the duty cycles generated initially for the four PWM signals with duty cycle values of 20%, 40%, 60% and 80% will now have the values scaled to 10%, 20%, 30% and 40% duty cycles.

The proposed example modifies the scaling value of the compare registers of TCE at run time every 10 ms. The range of duty cycles will be modified depending on the scaling value, and the following two subsections detail how to configure the TCE peripheral to do the desired behavior. The first subsection explains how to configure the TCE using bare metal code. The following two subsections detail how to configure the TCE using MCC Melody. The last subsection contains the results.

3.1 Bare Metal Implementation

1. The CLKCTRL peripheral must be configured for using the high-resolution feature of the TCE0. The following registers settings are required:

Figure 3-3. MCLKCTRLA Register

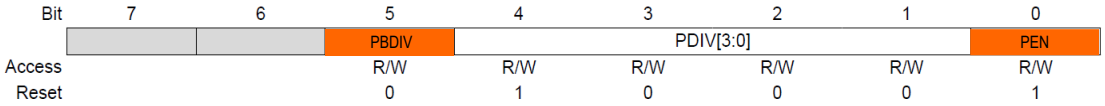
Bit	7	6	5	4	3	2	1	0
					CLKSEL[3:0]			
Access					R/W	R/W	R/W	R/W
Reset					0	0	0	0

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

Figure 3-4. MCLKCTRLB Register



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.

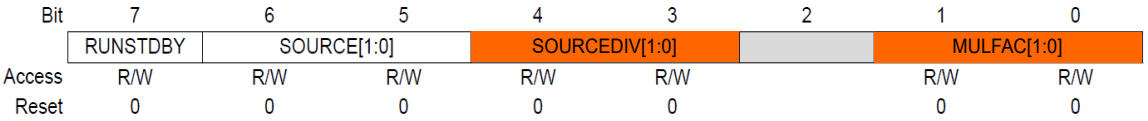
Value	Description
0x0	(NONE) = No Division
0x1	(DIV4) = Divide by 4

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

Figure 3-5. PLLCTRLA Register



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. The TCE corresponding register in Port Multiplexer can be set to route the module outputs to different ports. In this case, Port A is chosen, which is also the default port.

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

Figure 3-6. TCE PORTMUX Register

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

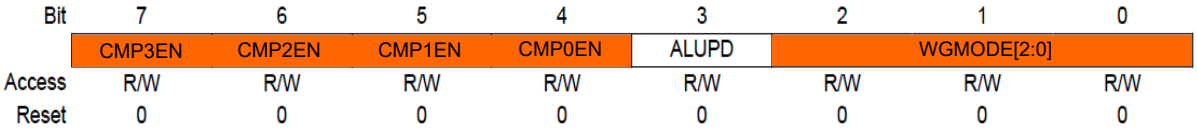
Value	Name	Description							
		WO0	WO1	WO2	WO3	WO4	WO5	WO6	WO7
0x0	PORTA	PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7
0x1	-	Reserved							
0x2	PORTC	PC0	PC1	PC2	PC3	-	-	-	-
0x3	PORTD	PD0	PD1	PD2	PD3	PD4	PD5	PD6	PD7
0x4	-	Reserved							
0x5	PORTF	PF0	PF1	PF2	PF3	PF4	PF5	-	-
0x6 - 0x7	-	Reserved							
0x8	PORTC2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

3. The CTRLB register contains the Enable bits of the compare channels and the bit field determining the Waveform Generation mode. In this example, channels 0, 1, 2 and 3 are used with Dual-Slope PWM mode.

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

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

Figure 3-7. CTRLB Register



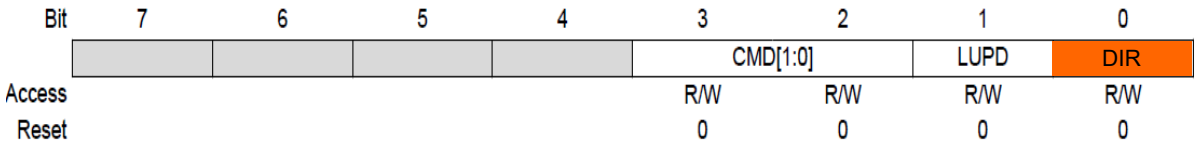
Bits 2:0 – WGMODE[2:0] Waveform Generation Mode
This bit field selects the Waveform Generation mode and controls the counting sequence of the counter, TOP value, UPDATE condition, interrupt condition, and the type of waveform generated.
No waveform generation is performed in the Normal mode of operation. 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. Set the DIR bit of the CTRLECLR register to '1' to set the timer to count clock ticks down (decrementing). The default value of the DIR bit is '0'.

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

Figure 3-8. CTRLECLR Register



5. Enable the Scaling mode amplitude and offset register usage, and set the scaling method to the bottom. The duty cycle values are scaled from 0% duty cycle up to 100% duty cycle. Instead of writing the absolute value to the Compare registers, it is possible to write this as a fractional value between 0 and 2.

   When writing a fractional value to the compare or compare buffer registers, first, the values are multiplied by the amplitude and then the offset is added. The high-resolution feature can increase the PWM's resolution four times, with two extra bits, or eight times, with three. This feature is helpful at very low frequencies. The high-resolution feature shortens the clock period steps by four or eight times. For example, for a clock with a frequency of 20 MHz with high resolution set to off, one clock period is 50 ns (T[s] = 1/f[Hz] = 1/20,000,000). For this clock with high-resolution 4X, one clock period is 12.5 ns. This clock has, with a high resolution of 8X, one clock period is 6.25 ns.

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

   The formulas for scaling the values written in the Compare registers are explained below:
   EQ3.1:

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

   Scaled value using the Center Scale mode.

   EQ3.2:

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

   Scaled value using the Bottom Scale mode.

   EQ3.3:

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

   Scaled value using the Top Scale mode.

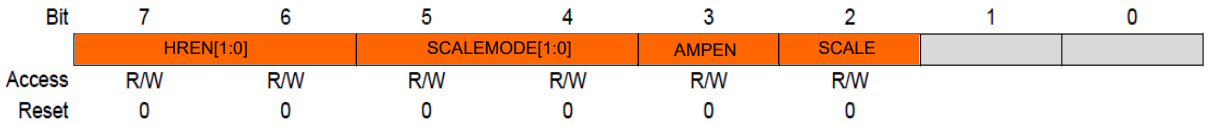
   EQ3.4

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

   Scaled value using the Top-Bottom Scale mode.

   The CMP_{frac} uses four different values mentioned in the AVR16EB16/20/28/32 AVR® EB Family Data Sheet, *Table 23-6. Effective Compare Value in Scaled Mode*.

Figure 3-9. CTRLD Register



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 is the buffer of the Period register. It is used to set the frequency of the PWM signal using EQ3.5:

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

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

In this example, the resolution is increased by four. Thus, the value written in the PER register is calculated using EQ3.6:

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

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

7. The compare registers are used to set the duty cycle. The values in the Compare registers are 20%, 40%, 60% and 80% of the one in the Period register because 20%, 40%, 60% and 80% duty cycles are desired. When using the buffer registers to write fractional values inside the compare registers, these values will be overridden at run time. These values can be set to 0 during initialization to highlight the difference between writing values ranging from 0 to PER and writing fractional values in the CMP registers. For example, the value written in CMP0 for 20% duty cycle PWM without Scaling mode is 0x320 for the given PER of 0xFA0, equivalent to 10 kHz frequency or 100 µs period, and high-resolution 4X. The value written in CMP0 for 20% duty cycle PWM, with Scaling mode enabled, is 0x1999 - the duty is calculated in hardware using the value of PER, AMPLITUDE and OFFSET registers.

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

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

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

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

8. Set the amplitude by writing a desired value to the AMP register.

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

9. Set the offset by writing a desired value to the OFFSET register.

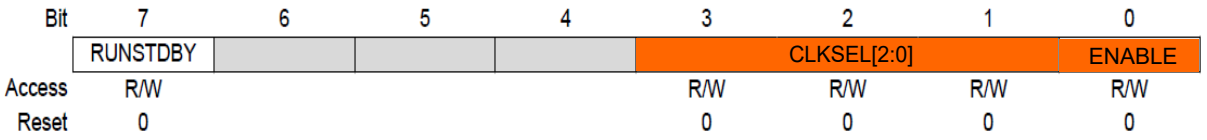
```
TCE0.OFFSET = 0x00;
```

10. Set the prescaler to 1 by changing the CLKSEL bit field in the CTRLA register. To start the counter, the user must set the Enable bit in the same register.

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

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

Figure 3-10. CTRLA Register



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. Then, the Port A Pins 0-3 (PA0-3) are set as output by writing a '1' to the corresponding bit in the Direction register of the port.

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

12. Add another pin as output to toggle it when a change in amplitude and compare values is happening. This way, the user will observe the changes. The pin toggles during run time.

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

13. To see the duty cycles scaling, the user must change the value of the Amplitude register during the run time. After writing a new value in the Amplitude register, the user must write the old values in the Compare registers. This because the duty cycle scaling update only happens when a new value is written in the Compare registers.

3.2 MCC Melody Implementation

To generate this project using MPLAB Code Configurator Melody, 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. Go to *Project Resources>System> CLKCTRL*.
   – Disable the Prescaler enable button
   – Clock Select: PLL Oscillator
   – PLL Multiplication Factor: Multiply by 16
   – PLL Source Division: DIV4
   – Prescaler B division PDIVB: DIV4

5. From *Device Resources>Drivers>Timer*, add the TCE module, then do the following configuration:
   – Module Enable: Must be enabled by default. If not, toggle the button (it turns blue if enabled).
   – Clock Selection: System clock (by default, the divider must be 1 - System clock)
   – Waveform Generation Mode: Dual-Slope PWM mode with overflow on TOP and BOTTOM (DSBOTH)
   – Requested Period [s]: 0.0001
   – Duty Cycle 0 [%]: 20
   – Duty Cycle 1 [%]: 40
   – Duty Cycle 2 [%]: 60
   – Duty Cycle 3 [%]: 80
   – Waveform Output n: Check the boxes from the Enable column for Waveform Output 0, 1, 2, 3
   – Duty Cycle High Resolution: Resolution increased by 4X
   – Scale mode: CMP values are scaled from the Bottom, 0% DC
   – Scaled Writing to registers: Fractional
   – Amplitude Control Enable: Toggle the button (it turns blue if enabled)
   – Amplitude Value: 1
6. Check if the TCE_WO[0-3] pins are locked as outputs on PORTA in the **Pin Grid View** tab. When the boxes from the 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**. Select the PIN5 of PORTD as an output, then toggle it to see the amplitude and compare value changes.
7. In the **Project Resources** tab, click the **Generate** button so that MCC will generate all the specified drivers and configurations.
8. Edit the `main.c` file, as indicated below.
   Add in the include files section:

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

   Add macro definitions:

```
/* Calculated values for Period, CMP, Amplitude and OFFSET registers */
#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)
```

   Add function:

```
void Amplitude_Value_Set(uint16_t value)
{
    /* Set the value of amplitude */
    TCE0_AmplitudeSet(value);

    /* Rewrite the values in CMPBUF registers so that the duty cycle get scaled according
     * to the new value of amplitude */
    TCE0_CompareAllChannelsBufferedSet(DUTY_CYCLE_20_PERCENT, DUTY_CYCLE_40_PERCENT,
                                       DUTY_CYCLE_60_PERCENT, DUTY_CYCLE_80_PERCENT);

    /* Toggle PD5 pin to know when the value of amplitude has changed */
    IO_PD5_Toggle();
}
```

Edit the main function:

```
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. Now, the project can be built and run from MPLAB X IDE. At run time, the scaling value for the compare registers modifies every 10 ms, and the range of the duty cycles will be modified accordingly.

3.3 Results

Below are illustrated some logic analyzer captures to show how the four PWM signals look and how the scaling of the duty cycles is created using the amplitude and offset.

Figure 3-11. Range of the Duty Cycles Scaled to 0-100% of the PERIOD

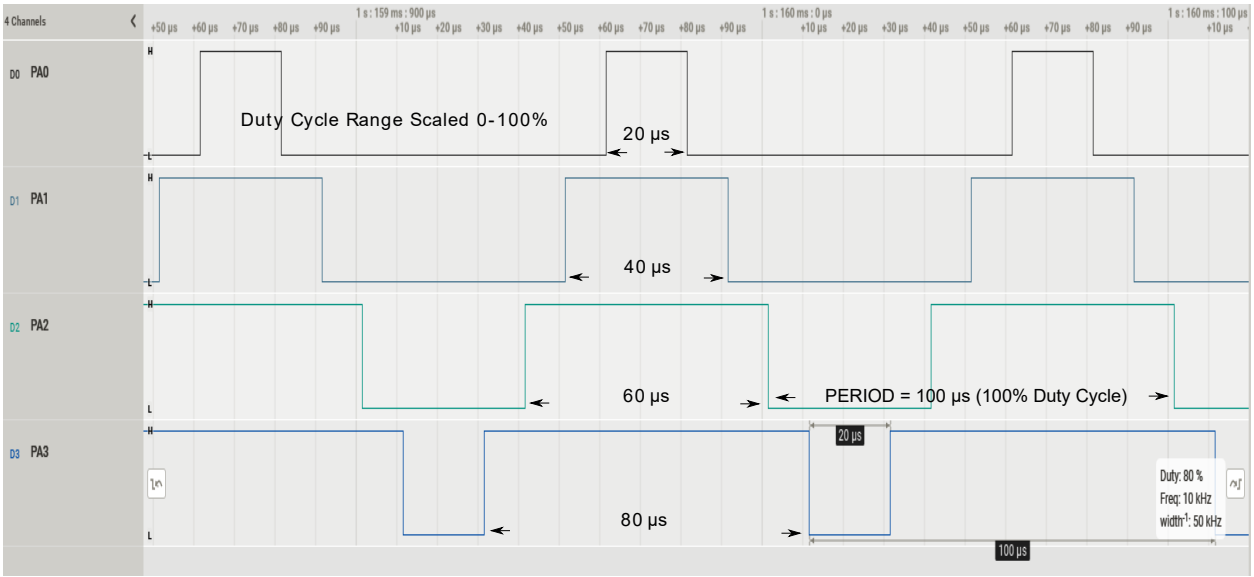


Figure 3-12. Range of the Duty Cycles Scaled to 0-50% of the PERIOD

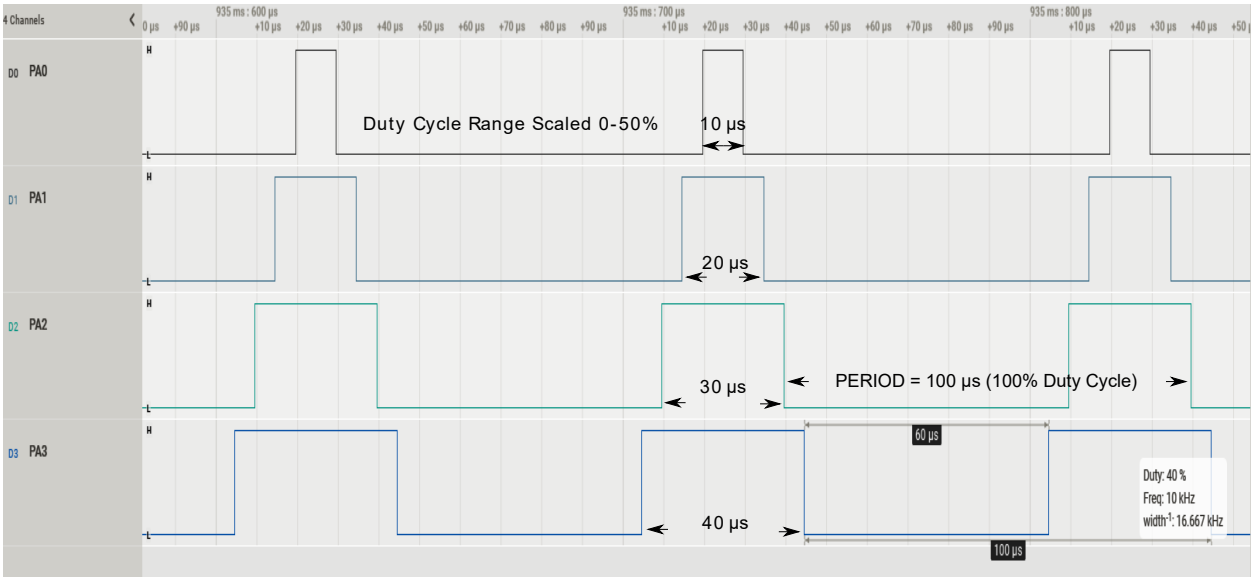
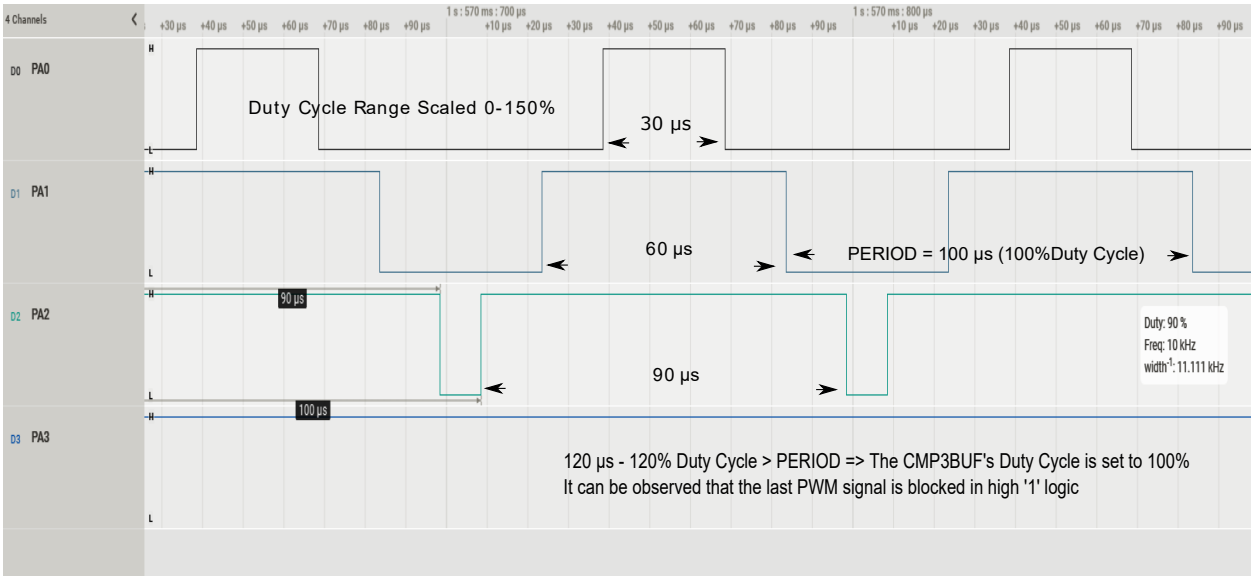


Figure 3-13. Range of the Duty Cycles Scaled to 0-150% of the PERIOD



Find a bare metal code example for the AVR16EB32 with the same functionality as described in this section here:

Click to view code examples on MPLAB DISCOVER

Find an MCC-generated code example for the AVR16EB32 with the same functionality as described in this section here:

Click to view code examples on MPLAB DISCOVER

4. Pattern Generation Using WEX

A fundamental use case of the waveform extension is to set a pattern for the output signals that change periodically. This Pattern Generation mode is helpful if the user wants to use the WEX to override the port pins. When using the WEX in Pattern Generation mode, the output pins from TCE are controlled. The user must enable the Pattern Generation mode and set a desired pattern containing the appropriate code.

A basic example containing the initialization is provided below. The program will toggle the pins (complementary pattern) every 25 µs using software delay ten times. After another 250 µs, a stairs pattern is generated with a 5 µs delay between each step. After another 250 µs delay, the process repeats.

The pins must be configured as outputs by setting the corresponding bits of the Direction register before the initialization of the waveform extension, as described below. Here, Port A pins 0-7 (PA0-7) were chosen. The patterns are best observed using a logic analyzer or eight LEDs to see the signal outputs.

The following two subsections explain configuring the WEX peripheral to do the desired behavior. The first subsection describes how to configure WEX using bare metal code. The second subsection explains how to configure WEX using MCC Melody. The last subsection contains the results.

4.1 Bare Metal Implementation

1. Enabling Pattern Generation mode in the CTRLA register will override the TCE-generated waveform.

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

Figure 4-1. CTRLA Register

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. Every signal output must be enabled to be overridden in Pattern Generation mode. Every signal has a one-to-one relationship with the bits from the PGMOVR register.

```
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;
```

Figure 4-2. Pattern Generation Override Register

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. The PGMOUT register holds the pattern for every signal. It is worth mentioning that every signal has a one-to-one relationship with the bits from the PGMOUT register. For example, to achieve the pattern 0xAA, set bits 1, 3, 5, and 7 to high ('1') logic in the PGMOUT register:

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

Figure 4-3. Pattern Generation Output Register

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. After configuring the WEX in Pattern Generation mode, the user can toggle each signal state using the PGMOUT register.

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

5. Add a delay to see when the toggling occurs.

```
_delay_us(25);
```

6. Port A pins 0-7 (PA0-7) are set as outputs by writing a '1' to the corresponding bits in the Direction register of the port. These GPIOs are configured only to obtain a visible output.

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

7. Make a setting in the PORTMUX to select the default PORT A pins.

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

Figure 4-4. WEX PORTMUX Register

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. Enable the WEX's outputs by setting the corresponding bits in the Output Override Enable register to enable the PORT A pins override.

```
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;
```

Figure 4-5. Output Override Enable Register

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. The last step for this example is to enable the TCE module because the source clock that passes from TCE to WEX is needed to update the defined patterns.

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

4.2 MCC Melody Implementation

To generate this project using MPLAB Code Configurator Melody, 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. Go to *Device Resources>Drivers>Timer*. Add the TCE module, then do the following configuration:
   – Module Enable: Toggle the button (it turns blue if enabled)
5. Go to *Device Resources>Drivers*. Add the WEX module, then do the following configuration:
   – Input Matrix: Direct
   – Update Source: TCE (the update condition for the output signals will be the TCE module)
   – Override Settings: Check all the boxes from the Output Enable column for the Waveform Output[0-7]
   – Pattern Generation Mode Enable: Toggle the button (it turns blue if enabled)
   – Pattern Generation Actions: Check all the boxes from the Override Enable column and set for each output a desired state (LOW or HIGH), to set a pattern
6. Check if the WEX_WO[0-7] pins are locked as outputs on PORTA in the **Pin Grid View** tab. The pins are secured when the boxes from the Output Enable column from Override Settings are checked. To change the PORT, click on a pin from another PORT in **Pin Grid View**.
7. In the **Project Resources** tab, click the **Generate** button so that MCC will generate all the specified drivers and configurations.
8. Edit the `main.c` file, as indicated below.
   Add in the include files section:

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

   Add macro definitions:

```
/* Patterns that are written in the PGMOUT register */
#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)
```

Add functions:

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

    /* Complementary signals pattern */
    for(uint8_t i = 0; i < 9; i++)
    {
        /* Complementary_pattern variable changes at every step */
        complementary_pattern = ~complementary_pattern;

        /* Toggle the pattern for each of the WEX's output */
        WEX0_PatternGenerationOutputSet(complementary_pattern);

        /* Software delay added in order for the toggle to be visible */
        _delay_us(25);
    }

    /* Put all signals in low '0' logic and wait 250us to see the transition from one of
     * the complementary patterns to the stairs pattern*/
    WEX0_PatternGenerationOutputSet(PATTERN_RESET);
}

void Stairs_Pattern_Set(void)
{
    /* Each of the signals switch from low to high one at a time in increasing order
     * generate a stairs increment pattern*/
    WEX0_PatternGenerationOutputSet(STAIRCASE7_PATTERN);

    /* Software delay added in order for the increment to be visible */
    _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);

    /* Each of the signals switch from low to high one at a time in decreasing order
     * generate a stairs decrement pattern*/
    WEX0_PatternGenerationOutputSet(STAIRCASE0_PATTERN);

    /* Software delay added in order for the decrement to be visible */
    _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);

    /* Put all signals in low '0' logic and wait 250us to see the transition from the
     * stairs pattern to one of the complementary patterns */
    WEX0_PatternGenerationOutputSet(PATTERN_RESET);
}
```

Edit the main function:

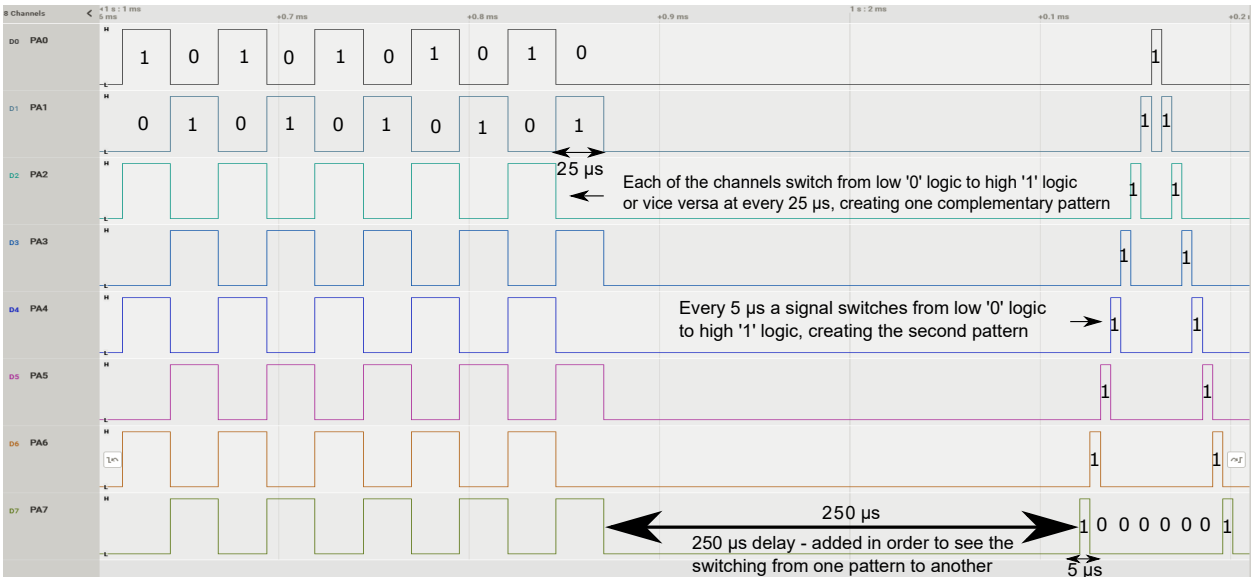
```
int main(void)
{
    SYSTEM_Initialize();

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

9. The project can now be built and run using MPLAB X IDE.

4.3 Results

A logic analyzer capture is illustrated below to show how the generated patterns look.

Figure 4-6. Switching From Complementary Pattern to Stairs Pattern



Find a bare metal code example for the AVR16EB32 with the same functionality as described in this section here:

Click to view code examples on MPLAB DISCOVER

Find an MCC-generated code example for the AVR16EB32 with the same functionality as described in this section here:

Click to view code examples on MPLAB DISCOVER

5. Generate Eight PWM Signals with TCE and WEX

The TCE can generate precise and versatile PWM signals on all four channels and generate complementary waveforms that do no overlap.

The WEX can be configured to extend the four compare channels given by TCE up to eight channels capable of generating a PWM signal.

Below is an example of how to set a TCE and a WEX instance to generate eight complementary PWM signals at 20 kHz with variable duty cycles using the buffering - use the Generate PWM Signals with TCE and WEX section for reference. The signals are in pairs of two without overlapping due to the added dead time, an essential feature in motor control for avoiding the shoot-through current in transistor switching. The update of the compare registers will happen during the compare match interrupts for each channel. In this example, the Fault protection feature is highlighted as well. When a software event is triggered, it emulates a Fault event, and all the signals are driven low, which happens every 1 ms. Configure the WEX for Fault event detection and the EVSYS to generate a software event to do this.

The following two subsections explain how to configure the TCE and WEX peripherals to perform the desired behavior. The first subsection clarifies the TCE and WEX configuring using bare metal code. The second subsection explains the TCE and WEX configuring using MCC Melody. The last subsection contains the results.

5.1 Bare Metal Implementation

1. The Clock peripheral must run at a frequency of 20 MHz. The default Clock speed is 3.33 MHz and has, by default, a prescaler division of six. This code example needs to disable the prescaler and use the clock to maximum speed.

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

Figure 5-1. MCLKCTRLB Register of CLKCTRL

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. The TCE and WEX corresponding registers in the Port Multiplexer can route the module outputs to different ports. In this case, Port A is chosen, which is also the default port.

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

Figure 5-2. PORTMUX Control for TCE and WEX

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

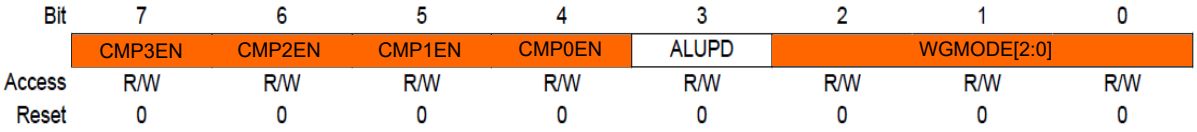
Value	Name	Description							
		WO0	WO1	WO2	WO3	WO4	WO5	WO6	WO7
0x0	PORTA	PA0	PA1	PA2	PA3	PA4	PA5	PA6	PA7
0x1	-	Reserved							
0x2	PORTC	PC0	PC1	PC2	PC3	-	-	-	-
0x3	PORTD	PD0	PD1	PD2	PD3	PD4	PD5	PD6	PD7
0x4	-	Reserved							
0x5	PORTF	PF0	PF1	PF2	PF3	PF4	PF5	-	-
0x6 - 0x7	-	Reserved							
0x8	PORTC2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

3. The CTRLB register of TCE contains the Enable bits of the compare channels and the bit field that determines the Waveform Generation mode. In this example channels 0, 1, 2, and 3 are used with a Single-Slope PWM mode.

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

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

Figure 5-3. CTRLB Register of TCE



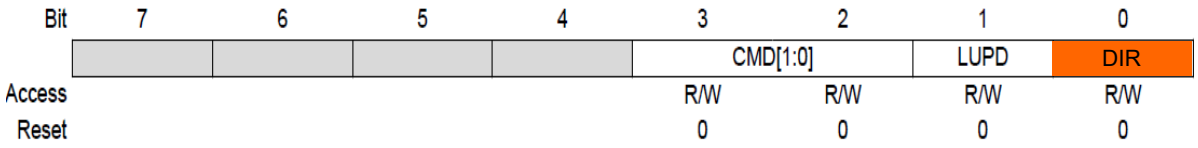
Bits 2:0 – WGMODE[2:0] Waveform Generation Mode
This bit field selects the Waveform Generation mode and controls the counting sequence of the counter, TOP value, UPDATE condition, interrupt condition, and the type of waveform generated.
No waveform generation is performed in the Normal mode of operation. 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. Set the DIR bit of the CTRLECLR register of TCE to '1' to set the timer to count clock ticks down (decrementing). The default value of the DIR bit is '0'.

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

Figure 5-4. CTRLECLR Register of TCE



5. PER is the buffer of the Period register of TCE and is used to set the frequency of the PWM signal using EQ5.1:

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

Considering the targeted values for this example:

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

```
/* the PER register is always set with the desired value -1, 1000 - 1 = 999, the desired
value for a 50us period */
TCE0.PER  = 0x3E7;
```

6. The Compare registers are updated using its buffer capabilities to set the duty cycle. The values in the Compare registers can be set to have random duty cycles such as 20%, 40%, 60% and 80%. However, these values are overridden at run time by updating the CMP registers using their buffers after the first interrupt occurs. So, we can set these values at '0'.

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

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

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

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

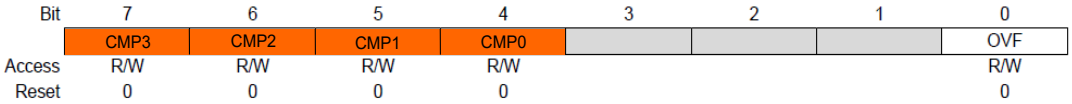
7. After starting the timer, the duty cycles will increase every time a compare register's value matches the counter value. The duty cycles' range will be between 0-100%. During the ISR, this happens for every compare channel. ISR routine for Compare 0 channel:

```
ISR(TCE0_CMP0_vect)
{
    /* clear the interrupt flag */
    TCE0.INTFLAGS = TCE_CMP0_bm;

    static uint16_t duty_cycle = 0;

    /* duty cycle update in interrupt */
    duty_cycle += 5;
    if(duty_cycle >= MAX_DUTY_CYCLE)
    duty_cycle = 0;
    TCE0.CMP0BUF = duty_cycle;
}
```

8. Enable interrupts on compare match for each of the compare registers of TCE.

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

Figure 5-5. INTCTRL Register of TCE



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. Set the initial value for the TCE counter to '0'.

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

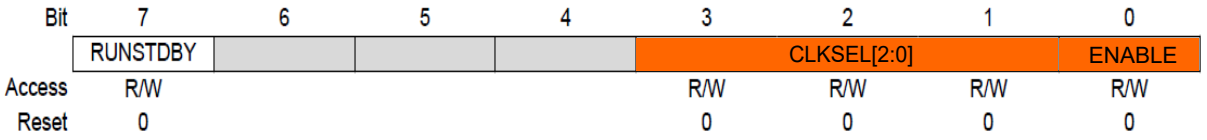
10. Set the prescaler to '1' by changing the CLKSEL bit field in the CTRLA register. To start the counter, the user has to set the Enable bit in the same register.

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

The last step for configuring the TCE is to enable the module.

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

Figure 5-6. CTRLA Register of TCE



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. Set the output routing matrix for the WEX in the Direct mode and enable dead-time insertion for each pair of two complementary signals. After making these settings, the WEX uses the four generated PWM signals from TCE to generate eight complementary signals.

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

Figure 5-7. CTRLA Register of WEX

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. Set the desired values for dead-time using the dead-time registers. The dead-time is measured in peripheral clock ticks. The equation below explains how to calculate the dead-time:

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

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

Figure 5-8. Dead-Time Registers of WEX: Dead-Time Low-Side, High-Side and Both Sides

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

To achieve symmetrical dead time for the application, use the last register. In this example, the low-side and high-side dead-time registers are used independently to create asymmetrical dead-time.

Equations used to calculate the dead time:

EQ5.2

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

Converting clock ticks to nanoseconds, for example:

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

$$TCE_{prescaler} = 4$$

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

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

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

13. Set the Fault Restart mode condition and Fault signal driving pattern in the FAULTCTRL register.

```
/* set fault restart mode to latched mode - fault is active as long as the fault condition
is active */
/* to restart from fault in latched mode the user must use a software fault clear command
*/
WEX0.FAULTCTRL = WEX_FDMODE_LATCHED_gc;
```

```
/* drive all pins to low '0' logic when a fault is detected */
WEX0.FAULTCTRL |= WEX_FDACT_LOW_gc;
```

Figure 5-9. FAULTCTRL Register of WEX

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.

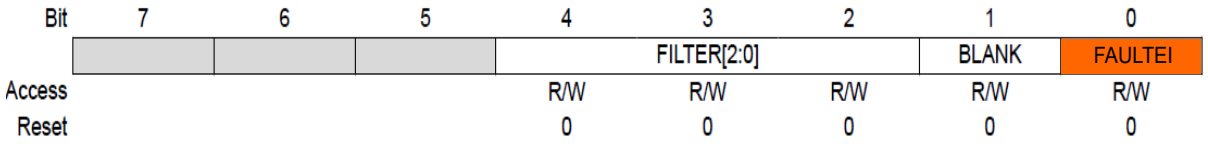
Bits 1:0 – FDACT[1:0] Fault Detection Action

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. Enable the event input A of WEX using the EVCTRLA register.

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

Figure 5-10. EVCTRLA Register of WEX



Bit 0 – FAULTEI Fault Event Input Enable

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

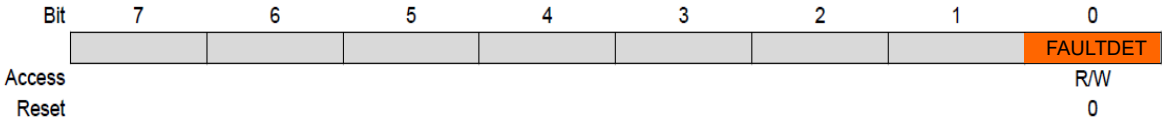
Note: Fault event inputs are not taken into account before Timer/Counting driving a waveform is enabled

15. Enable Fault interrupt (Fault detection) and clear the Fault flags during the WEX's ISR.

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

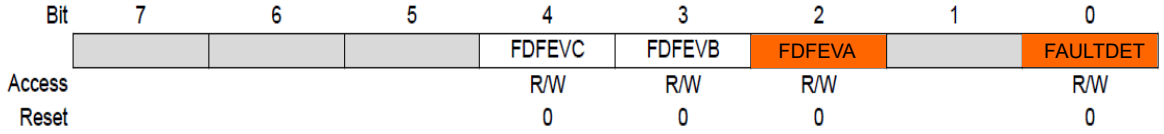
```
/* WEX fault ISR */
ISR(WEX0_FDFEVA_vect)
{
    /* clear the interrupt flag and fault event flag */
    WEX0.INTFLAGS = WEX_FDFEVA_bm | WEX_FAULTDET_bm;
}
```

Figure 5-11. INTCTRL and INTFLAGS Registers of WEX



Bit 0 – FAULTDET Fault Detection Interrupt Enable

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



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. Configure the EVENT SYSTEM (EVSYS) peripheral to generate events on Channel 0. Configure WEX to be the user of the event generator from Channel 0. Generate a Fault using a software event. To stop the Fault, turn off the Fault generator channel from EVSYS and clear the Fault state using a software command.

```
/* set the WEX0 peripheral as the user of the event generator from channel 0, which is a
software event in this example*/
EVSYS.USERWEXA = EVSYS_USER_CHANNEL0_gc;
```

```
/* software fault creation, repeat in main loop to see it on LogicA. This is an event
generated using a software command */
EVSYS.SWEVENTA = EVSYS_SWEVENTA_CH0_gc;
```

```
/* clear fault condition using a software command */
WEX0.CTRLC = WEX_CMD_FAULTCLR_gc;
```

17. Enable WEX's outputs to generate the eight PWM signals using the OUTOVEN register.

```
/* enables WEX's outputs */
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;
```

Figure 5-12. OUTOVEN Register of WEX

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. Then, the Port A Pins 0-7 (PA0-7) are set as output by writing a '1' to the corresponding bit in the Direction register of the port.

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

19. After all the modules are configured, enable global interrupts.

```
/* enable global interrupts */
    sei();
```

5.2 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. Go to *Project Resources>System>Interrupt Manager*. Toggle the Global Interrupt Enable button.
5. Go to *Project Resources>System>CLKCTRL*. Disable the Prescaler enable button.
6. From *Device Resources>Drivers>Timer*, add the TCE module, then do the following configuration:
   – Enable Timer: This may be enabled by default. If not, toggle the button (it turns blue if enabled).
   – Clock Divider: System clock (by default, the divider should be 1 - System clock)
   – Waveform Generation Mode: Single-Slope PWM mode
   – Requested Period [s]: 0.00005

– Waveform Output n: Check the boxes from the Enable column for Waveform Output 0, 1, 2, 3
– Generate ISR: Toggle the button (it turns blue if enabled)
– Enable Compare 0 Interrupt: Toggle the button (it turns blue if enabled)
– Enable Compare 1 Interrupt: Toggle the button (it turns blue if enabled)
– Enable Compare 2 Interrupt: Toggle the button (it turns blue if enabled)
– Enable Compare 3 Interrupt: Toggle the button (it turns blue if enabled)

7. From *Device Resources>Drivers*, add the WEX module, then do the following configuration:
   – Input Matrix: Direct
   – Update Source: TCE (the update condition for the output signals will be dictated by TCE)
   – Override Settings: Check all the boxes from the Output Enable column for the Waveform Output [0-7]
   – Dead-time Insertion Channel 0 Enable: Toggle the button (it turns blue if enabled)
   – Dead-time Insertion Channel 1 Enable: Toggle the button (it turns blue if enabled)
   – Dead-time Insertion Channel 2 Enable: Toggle the button (it turns blue if enabled)
   – Dead-time Insertion Channel 3 Enable: Toggle the button (it turns blue if enabled)
   – Requested Dead-time Low Side (µs): 0.150
   – Requested Dead-time High Side (µs): 0.250
   – Fault Event Input A: Toggle the button (it turns blue if enabled)
   – Fault Enable: Toggle the button (it turns blue if enabled)
   – Fault Interrupt Enable: Toggle the button (it turns blue if enabled)
   – Fault Detection Restart Mode: Cycle-by-Cycle
   – Fault Detection Action: Low
8. From *Device Resources>Drivers*, add the EVSYS module, then follow the below configuration:
   – CHANNELS: CHANNEL0
   – USERS: WEXA (Select the CHANNEL0 rectangle from the **CHANNELS** tab, hold and drag the cursor to the WEXA rectangle from the **USERS** tab)
9. Check if the WEX_WO[0-7] pins are locked as outputs on PORTA in the **Pin Grid View** tab. The pins are also locked when checking the boxes from the Enable column from Waveform Output n. To change the PORT, click on a pin from another PORT in **Pin Grid View**.
10. In the **Project Resources** tab, click the **Generate** button so that MCC will generate all the specified drivers and configurations.
11. Edit the `main.c` file, as indicated below.
    Add in the include files the following:

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

    Add macro definitions:

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

Add ISR callback functions:

```
/* Callback function that is called in the ISR routine */
void UserCallback_CMP0(void)
{
    static uint16_t duty_cycle = 0;

    /* Duty cycle update in interrupt */
    duty_cycle += 5;
    if(duty_cycle >= MAX_DUTY_CYCLE)
      duty_cycle = 0;
    TCE0_PWM_BufferedDutyCycle0Set(duty_cycle);
}

/* Callback function that is called in the ISR routine */
void UserCallback_CMP1(void)
{
    static uint16_t duty_cycle = 0;

    /* Duty cycle update in interrupt */
    duty_cycle += 15;
    if(duty_cycle >= MAX_DUTY_CYCLE)
      duty_cycle = 0;
    TCE0_PWM_BufferedDutyCycle1Set(duty_cycle);
}

/* Callback function that is called in the ISR routine */
void UserCallback_CMP2(void)
{
    static uint16_t duty_cycle = 0;

    /* Duty cycle update in interrupt */
    duty_cycle += 25;
    if(duty_cycle >= MAX_DUTY_CYCLE)
      duty_cycle = 0;
    TCE0_PWM_BufferedDutyCycle2Set(duty_cycle);
}

/* Callback function that is called in the ISR routine */
void UserCallback_CMP3(void)
{
    static uint16_t duty_cycle = 0;

    /* Duty cycle update in interrupt */
    duty_cycle += 35;
    if(duty_cycle >= MAX_DUTY_CYCLE)
      duty_cycle = 0;
    TCE0_PWM_BufferedDutyCycle3Set(duty_cycle);
}
```

Add functions:

```
void Create_Fault(void)
{
    /* Fault creation, repeat in main loop to see it on Logic Analyzer. This is an event
     * generated using a software command */
    EVSYS_SoftwareEventASet(EVSYS_SWEVENTA_CH0_gc);
}

void Clear_Fault(void)
{
    /* Clear fault condition using a software command */
    WEX0_SoftwareCommand(WEX_CMD_FAULTCLR_gc);
}
```

Edit the main function:

```
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. Now, the project can be built and run using MPLAB X IDE. During the run time, the scaling value for the compare registers updates every 10 ms, adjusting the range of the duty cycles.

5.3 Results

The figure below shows some logic analyzer captures to demonstrate the eight generated complementary PWM signals, the duty cycles increasing from 0-100%, what happens during a Fault event, and how the dead-time periods look.

Figure 5-13. Eight PWM Signals With Variable Duty Cycles, Fault Detection Actions and Restoring to Normal Operation After the Fault Is Cleared

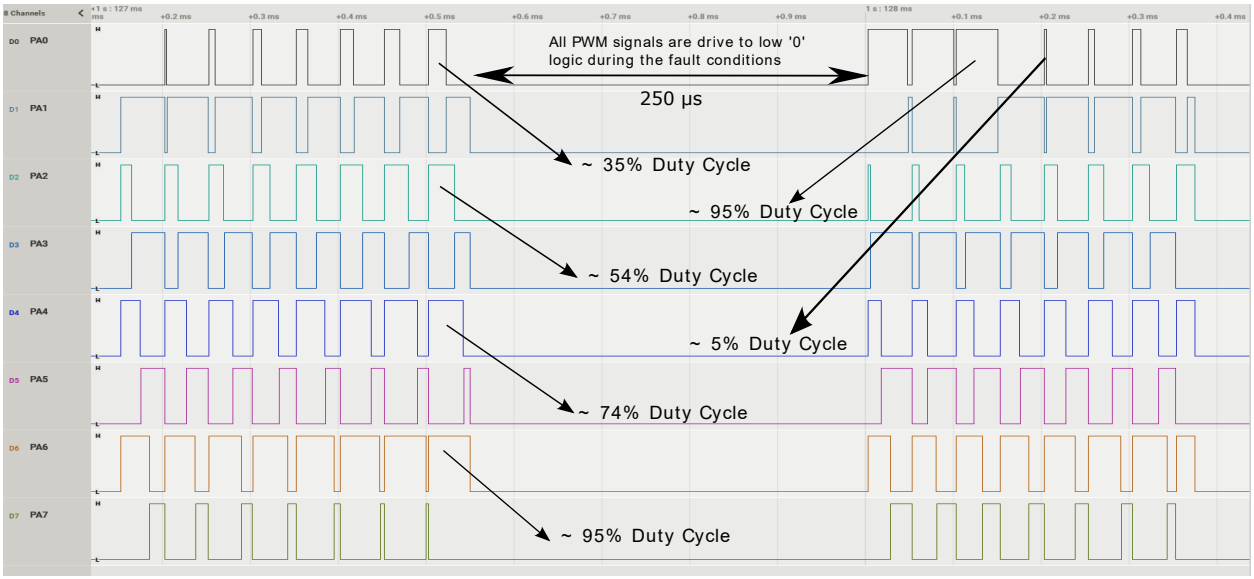


Figure 5-14. Dead-Time Periods Highlighted

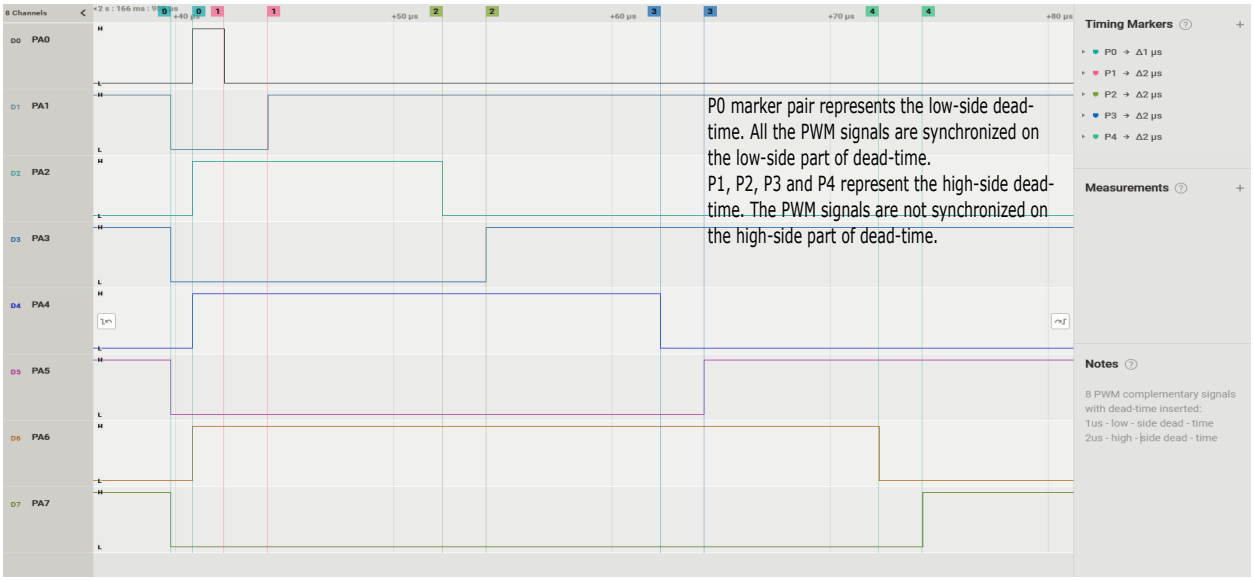
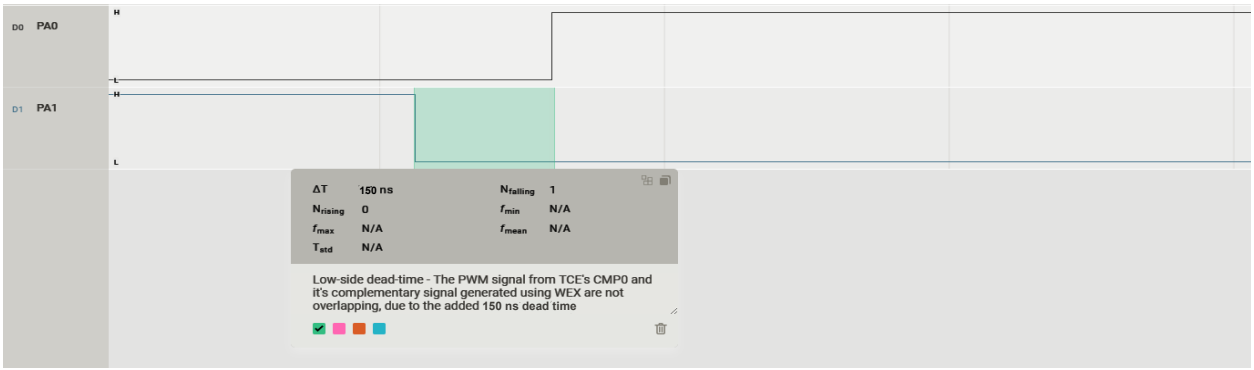


Figure 5-15. A More In-Depth Look at the Low-Side Dead-Time Period



Find a bare metal code example for the AVR16EB32 with the same functionality as described in this section here:

Click to view code examples on MPLAB DISCOVER

Find an MCC-generated code example for the AVR16EB32 with the same functionality as described in this section here:

Click to view code examples on MPLAB DISCOVER

6. References

Use the following links for more information about the TCE operation modes:

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

7. Revision History

Document Revision	Date	Comments
B	08/2024	Update image with MULFAC[1:0] register table
A	11/2023	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: 978-1-6683-0093-0

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 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com	**Australia - Sydney** Tel: 61-2-9868-6733	**India - Bangalore** Tel: 91-80-3090-4444	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393
Atlanta Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455	**China - Beijing** Tel: 86-10-8569-7000	**India - New Delhi** Tel: 91-11-4160-8631	**Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829
Austin, TX Tel: 512-257-3370	**China - Chengdu** Tel: 86-28-8665-5511	**India - Pune** Tel: 91-20-4121-0141	**Finland - Espoo** Tel: 358-9-4520-820
Boston Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088	**China - Chongqing** Tel: 86-23-8980-9588	**Japan - Osaka** Tel: 81-6-6152-7160	**France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79
Chicago Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075	**China - Dongguan** Tel: 86-769-8702-9880	**Japan - Tokyo** Tel: 81-3-6880- 3770	**Germany - Garching** Tel: 49-8931-9700
Dallas Addison, TX Tel: 972-818-7423 Fax: 972-818-2924	**China - Guangzhou** Tel: 86-20-8755-8029	**Korea - Daegu** Tel: 82-53-744-4301	**Germany - Haan** Tel: 49-2129-3766400
Detroit Novi, MI Tel: 248-848-4000	**China - Hangzhou** Tel: 86-571-8792-8115	**Korea - Seoul** Tel: 82-2-554-7200	**Germany - Heilbronn** Tel: 49-7131-72400
Houston, TX Tel: 281-894-5983	**China - Hong Kong SAR** Tel: 852-2943-5100	**Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906	**Germany - Karlsruhe** Tel: 49-721-625370
Indianapolis Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380	**China - Nanjing** Tel: 86-25-8473-2460	**Malaysia - Penang** Tel: 60-4-227-8870	**Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44
Los Angeles Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800	**China - Qingdao** Tel: 86-532-8502-7355	**Philippines - Manila** Tel: 63-2-634-9065	**Germany - Rosenheim** Tel: 49-8031-354-560
Raleigh, NC Tel: 919-844-7510	**China - Shanghai** Tel: 86-21-3326-8000	**Singapore** Tel: 65-6334-8870	**Israel - Hod Hasharon** Tel: 972-9-775-5100
New York, NY Tel: 631-435-6000	**China - Shenyang** Tel: 86-24-2334-2829	**Taiwan - Hsin Chu** Tel: 886-3-577-8366	**Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781
San Jose, CA Tel: 408-735-9110 Tel: 408-436-4270	**China - Shenzhen** Tel: 86-755-8864-2200	**Taiwan - Kaohsiung** Tel: 886-7-213-7830	**Italy - Padova** Tel: 39-049-7625286
Canada - Toronto Tel: 905-695-1980 Fax: 905-695-2078	**China - Suzhou** Tel: 86-186-6233-1526	**Taiwan - Taipei** Tel: 886-2-2508-8600	**Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340
	China - Wuhan Tel: 86-27-5980-5300	**Thailand - Bangkok** Tel: 66-2-694-1351	**Norway - Trondheim** Tel: 47-72884388
	China - Xian Tel: 86-29-8833-7252	**Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Poland - Warsaw** Tel: 48-22-3325737
	China - Xiamen Tel: 86-592-2388138		**Romania - Bucharest** Tel: 40-21-407-87-50
	China - Zhuhai Tel: 86-756-3210040		**Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91
			Sweden - Gothenberg Tel: 46-31-704-60-40
			Sweden - Stockholm Tel: 46-8-5090-4654
			UK - Wokingham Tel: 44-118-921-5800 Fax: 44-118-921-5820