TB3217

Getting Started with Timer/Counter Type A (TCA)

Introduction

Author: Catalin Visan, Microchip Technology Inc.

The AVR® microcontrollers are equipped with powerful timers designed to cover a wide area of applications, from signals measurement to events synchronization and waveforms generation.

The Timer/Counter type A (TCA) is a 16-bit timer that is present in the tinyAVR® 0-series, tinyAVR® 1-series, megaAVR® 0-series, and AVR® DA devices. The main idea behind the TCA is that a very flexible timer is needed to perform convoluted actions as well as the very basic functions of a simple timer. The flexibility comes from the multitude of features provided, such as the possibility of splitting the 16-bit timer into two completely independent 8-bit timers or the built-in Wave Generation modes. Another important characteristic is that the TCA was devised to overcome common problems when using timers, such as the unpredictable behavior of the PWM signal when the duty cycle is changed while the timer is running. The TCA has double-buffered registers that synchronize the updates of different registers, making the waveforms generated predictable in every single situation.

The purpose of this technical brief is to familiarize the reader with some of the operating modes of the TCA, emphasizing this timer's particularities and to provide initialization code snippets. For a deeper understanding of the functionality, please consult the data sheet. The structure of the document covers three specific use cases:

- **Using Periodic Interrupt Mode:**
  Initialize the timer to trigger an interrupt every 250 ms, toggling an example GPIO in the Interrupt Service Routine
- **Generating a Dual-Slope PWM Signal:**
  Initialize the timer to generate a dual-slope 16-bit PWM signal with 1 kHz frequency and 50% duty cycle on a GPIO pin
- **Generating Two PWM Signals in Split Mode:**
  Initialize the timer in Split mode to generate two single-slope 8-bit PWM signals on two GPIO pins, with independent duty cycle and frequency

Note: For each of the use cases described in this document, there are two code examples: One bare metal developed on ATmega4809 and one generated with MPLAB® Code Configurator (MCC) developed on AVR128DA48.

View the ATmega4809 Code Example on GitHub
Click to browse repository

View the AVR128DA48 Code Example on GitHub
Click to browse repository

Table of Contents

1. Relevant Devices

This section lists the relevant devices for this document. The following figures show 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 tinyAVR® 1-series devices may require code modification due to fewer available instances of some peripherals
- Horizontal migration to the left reduces the pin count and, therefore, the available features
- Devices with different Flash memory sizes typically also have different SRAM and EEPROM

Figure 1-1. tinyAVR® 0-series Overview

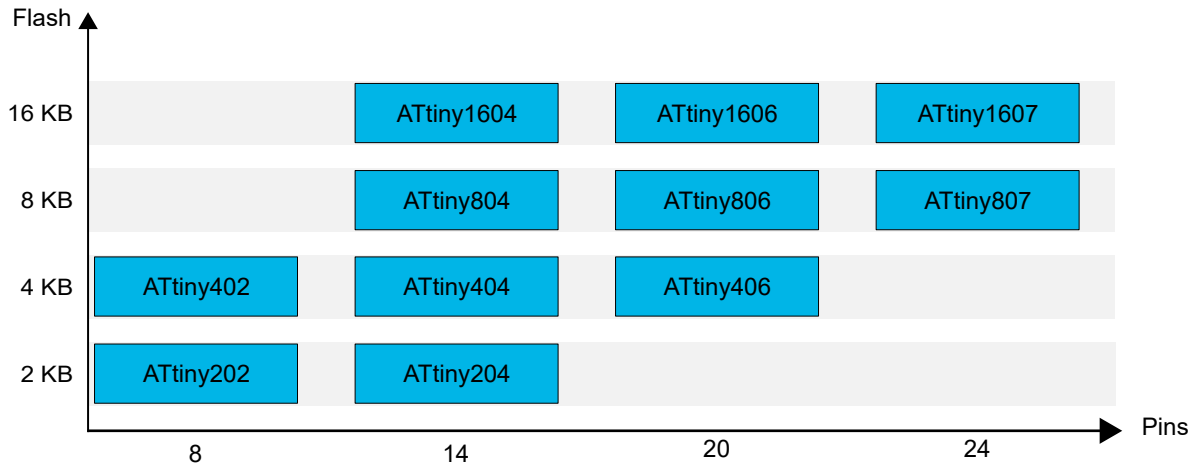


Figure 1-2. tinyAVR® 1-series Overview

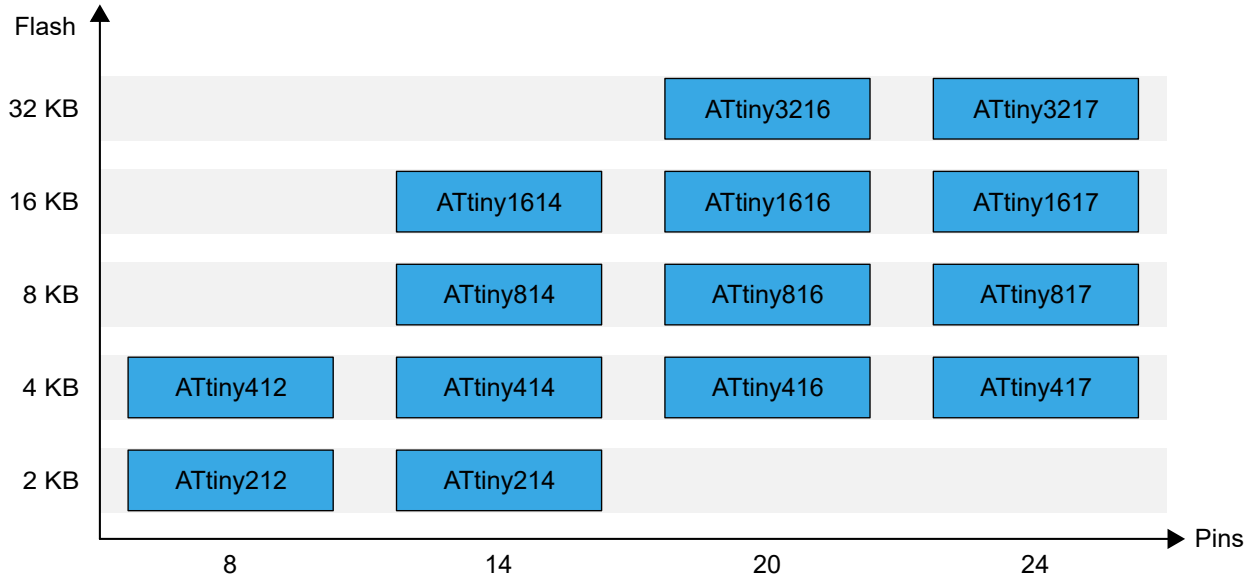


Figure 1-3. megaAVR® 0-series Overview

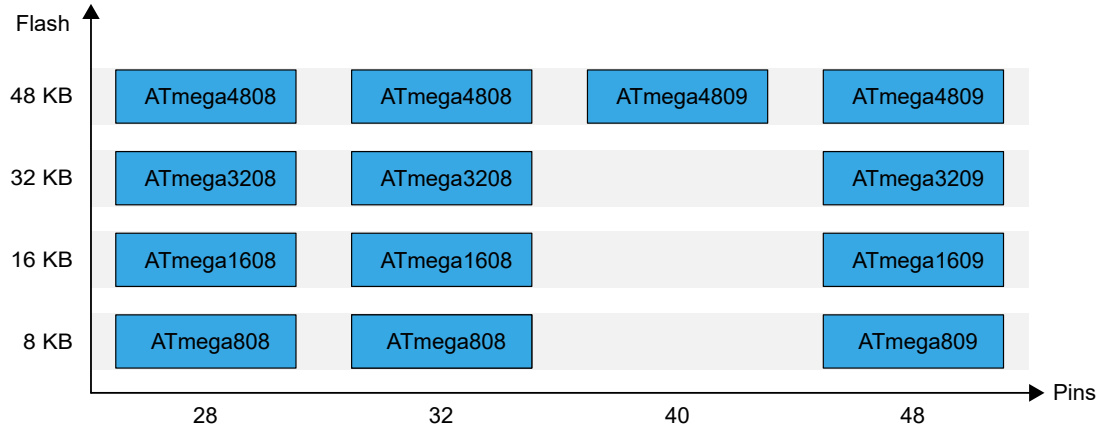
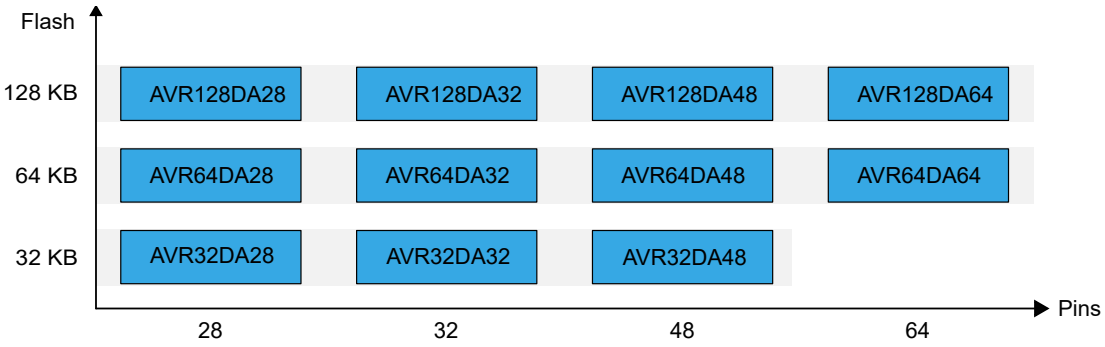


Figure 1-4. AVR® DA Family Overview

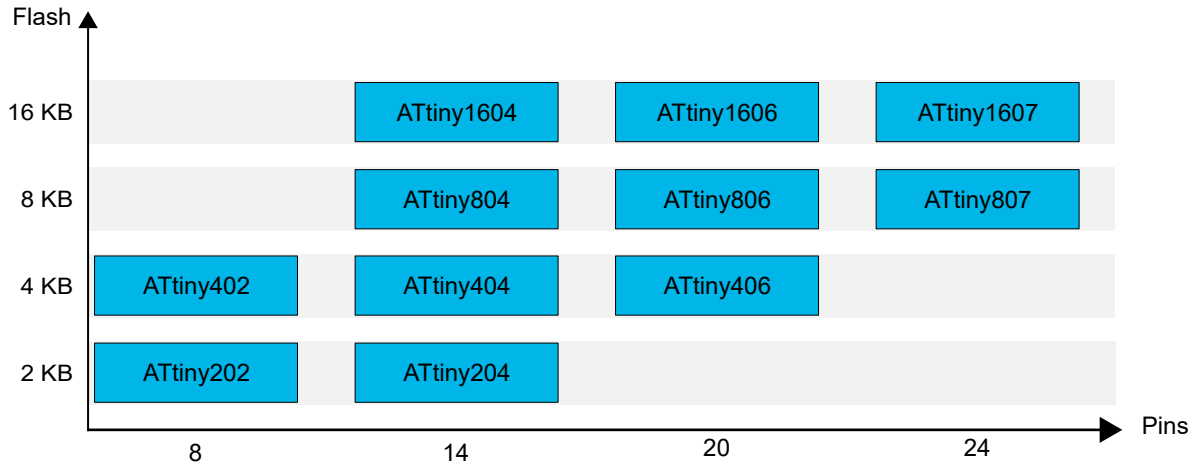


1.1 tinyAVR® 0-series

The figure below shows the tinyAVR® 0-series 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
- Horizontal migration to the left reduces the pin count and, therefore, the available features

Figure 1-5. tinyAVR® 0-series Overview



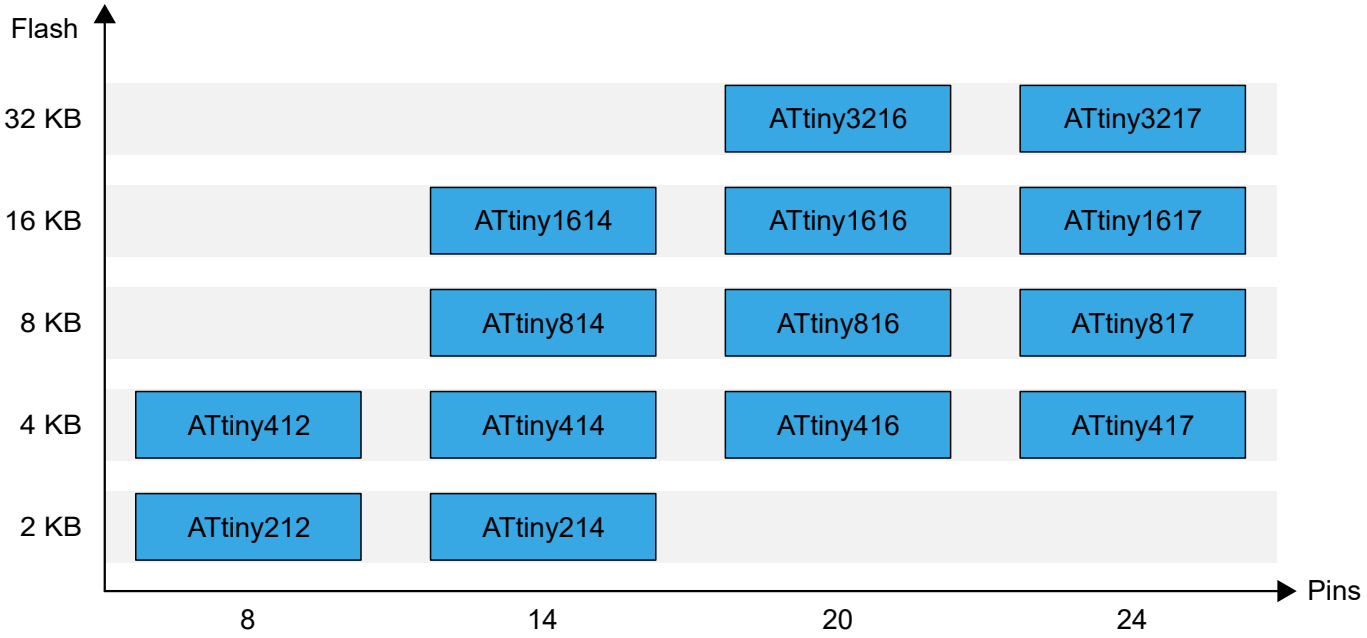
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

1.2 tinyAVR® 1-series

The following figure shows the tinyAVR 1-series 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 may require code modification due to fewer available instances of some peripherals.
- Horizontal migration to the left reduces the pin count and, therefore, the available features

Figure 1-6. tinyAVR® 1-series Overview



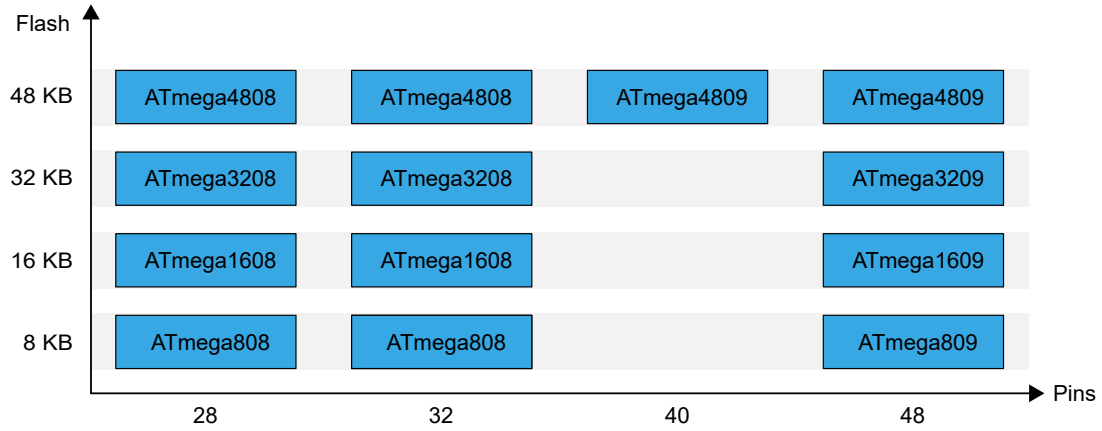
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

1.3 megaAVR® 0-series

The figure below shows the megaAVR® 0-series devices, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin and feature compatible
- Horizontal migration to the left reduces the pin count and, therefore, the available features

Figure 1-7. megaAVR® 0-series Overview



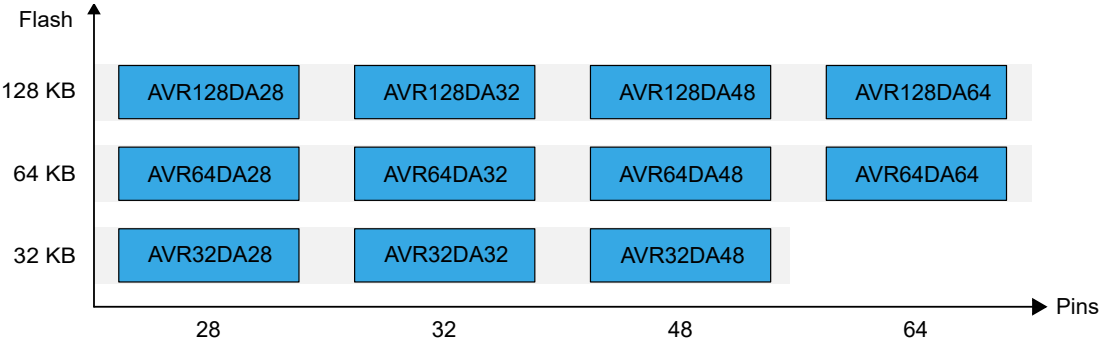
Devices with different Flash memory sizes typically also have different SRAM and EEPROM.

1.4 AVR® DA Family Overview

The figure below shows the AVR® DA devices, laying out pin count variants and memory sizes:

- Vertical migration is possible without code modification, as these devices are fully pin and feature compatible
- Horizontal migration to the left reduces the pin count, and therefore, the available features

Figure 1-8. AVR® DA Family Overview



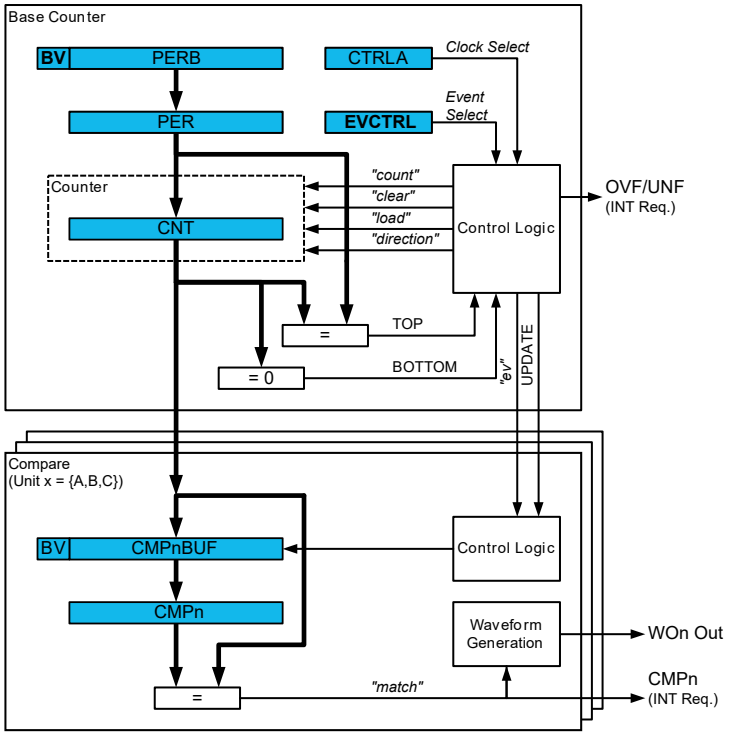
Devices with different Flash memory sizes typically also have different SRAM.

2. Overview

The flexible 16-bit PWM Timer/Counter type A (TCA) provides accurate program execution timing, frequency and waveform generation, and command execution.

A TCA instance consists of a base counter and three compare channels. The user can set the base counter to count upwards or downwards based on clock ticks (timer) or different events (counter). The Event System can also be used for direction control or to synchronize operations. The period can be adjusted from a specific register as well as the compare thresholds that can be used to generate different waveforms or to trigger events. It is worth mentioning that a prescaler can be used to divide the clock source and also that TCA can operate in Idle sleep mode.

Figure 2-1. Timer/Counter Block Diagram



The counter value is compared continuously to zero and the period value. If one of the conditions is met, the control logic block acts according to the configured operation mode, updating the counter and/or generating an interrupt request. The counter is also compared to the Compare registers. Use these comparisons to generate interrupt requests and set the waveform period or the pulse width if a Waveform Generator 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 updated only when the Counter register is updated. Each buffer has a Buffer Valid (BV) bit used by the logic block to determine if the respective register needs to be updated.

The TCA can be configured to use the Event System and can be utilized to count rising and/or falling edges of the event signal or use it to enable clock ticks counting. Also, the polarity of the event signal can be used to control the direction, low signal for up-counting, and high signal for down-counting. Moreover, the TCA can generate one-cycle strobes on the event channel outputs. The trigger for generating one-cycle strobes on the event channel can be the overflow of the timer, a compare channel match, or an underflow in Split mode.

3. Using Periodic Interrupt Mode

A basic use case of the timer is to set it to trigger an interrupt every time it is updated. This mode is useful if a piece of code must be executed repeatedly every few milliseconds. The user must enable the interrupts and set an Interrupt Service Routine (ISR), which will contain the appropriate code. A basic example containing the initialization and an ISR is provided below. The program will toggle a pin every 250 ms using TCA's periodic interrupts. A pin must be configured as an output by setting the corresponding bit of the Direction register before the initialization of the timer as described below. In this case, Port A pin 0 (PA0) was chosen.

1. Setting the corresponding bit in the Interrupt Control register enables the overflow interrupt of TCA.

```
TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;
```

Figure 3-1. Interrupt Control Register

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

2. In this mode, no waveform must be generated, so the Waveform Generation bit field in the CTRLB register must be configured accordingly.

```
TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc;
```

Figure 3-2. CTRLB Register

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

bits 2:0 WGMODE[2:0]: Waveform Generation Mode bits
These bits select the Waveform Generation mode.

WGMODE[2:0]	Group Configuration	Mode of Operation
000	NORMAL	Normal
001	FRQ	Frequency
010	-	Reserved
011	SINGLESLOPE	Single-slope PWM

3. Since the timer may count clock ticks, not events, the CNTEI bit of the EVCTRL register must be set to '0'. It is worth mentioning that this is the default value of the CNTEI bit.

```
TCA0.SINGLE.EVCTRL &= ~(TCA_SINGLE_CNTEI_bm);
```

Figure 3-3. EVCTRL Register

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

4. The value written in the Period register represents the number of clock ticks between the moment when the timer starts and the moment when the first interrupt is triggered, and also the number of clock ticks between two consecutive interrupts. It can be deduced from the following equation.

Note: The value written to the Period register will be one less than the desired count because the counting starts from '0'.

$$time_{TCA_{IRQ}}\left(s\right) = \frac{TCA_{period} + 1}{TCA_{clock}\left(Hz\right)}$$

where the clock of the TCA instance is defined by: $TCA_{clock}\left(Hz\right) = \frac{f_{CLK}(Hz)}{TCA_{prescaler}}$

and the peripheral clock $f_{CLK} = \frac{CLK\_MAIN}{Main\ \ clock\ prescaler}$

Combining these equations, the following result is obtained:

$$time_{TCA_{IRQ}}(s) = \frac{\left(TCA_{period} + 1\right) \times TCA_{prescaler}}{f_{CLK}(Hz)}$$

Note: The Period register is 16 bits wide. Thus the longest achievable interrupt period with no TCA prescaler is listed below.

$$time_{TCA_{IRQ}}(s) = \frac{\left(TCA_{period} + 1\right) \times TCA_{presclaer}}{f_{CLK}(Hz)} = \frac{(0xFFFF + 1) \times 1}{3333333(Hz)} = 19,66 \times 10^{-3}s$$

Considering the targeted values for this example,

$$TCA_{period} = \frac{time_{TCA_{IRQ}}(s)}{TCA_{prescaler}} - 1 = \frac{250 \times 10^{-3}(s) \times 3333333}{256} - 1 \cong 3254 = 0xCB6$$

```
TCA0.SINGLE.PER = 0x0CB6;
```

5. From the CTRLA register, the prescaler is set to '256'. To start the counter, the user must set the Enable bit in the same register.

```
TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV256_gc | TCA_SINGLE_ENABLE_bm;
```

Figure 3-4. Control A Register

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

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

Value	Name	Description
0x5	DIV64	$f_{TCA} = f_{CLK\_PER}/64$
0x6	DIV256	$f_{TCA} = f_{CLK\_PER}/256$
0x7	DIV1024	$f_{TCA} = f_{CLK\_PER}/1024$

6. After the timer is fully configured, the `sei();` macro enables the global interrupts. Always configure the peripherals when the global interrupts are disabled to avoid problems.
In the ISR, the output pin is toggled by setting the corresponding bit in the Input register of the port. Also, the Overflow Interrupt flag is cleared.

Tip: Interrupt flags have to be cleared in software by writing '1' at the respective bit location.

```
ISR(TCA0_OVF_vect)
{
    PORTA.OUTTGL = PIN0_bm;

    TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;
}
```

Figure 3-5. INTCTRL Register

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

An interrupt request is generated when the corresponding interrupt source is enabled, and the Interrupt flag is set. As soon as the flag is set, the microcontroller will start executing the code from the ISR written by the user. The interrupt request remains active until the Interrupt flag is cleared. The parameter of the ISR is the interrupt vector. Therefore, the user can specify what interrupt source the ISR corresponds to. A list of the TCA interrupt vectors is provided below. When programming, it is useful to use the autocomplete function of the IDE to identify the desired interrupt vector.

Figure 3-6. Available Interrupt Vectors and Sources in Normal Mode

Name	Vector Description	Conditions
OVF	Overflow and Compare match interrupt	The counter has reached its top value and wrapped to zero
CMP0	Compare channel 0 interrupt	Match between the counter value and the Compare 0 register
CMP1	Compare channel 1 interrupt	Match between the counter value and the Compare 1 register
CMP2	Compare channel 2 interrupt	Match between the counter value and the Compare 2 register

7. Port A pin 0 (PA0) is set as output by writing a '1' to the corresponding bit in the Direction register of the port. This GPIO is configured only to obtain a visible output, but it has nothing to do with the TCA instance itself in this mode.

```
PORTA.DIR |= PIN0_bm;
```

Tip: The full code example is also available in the Appendix section.

View the ATmega4809 Code Example on GitHub
Click to browse repository

An MCC generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:

View the AVR128DA48 Code Example on GitHub
Click to browse repository

4. Generating a Dual-Slope PWM Signal

One of the most important characteristics of the TCA when compared to other timers, such as TCB, is the versatility and precision of the PWM generation. The user can choose from various configurations according to the complexity of the application. The TCA can be configured in both Single-Slope and Dual-Slope PWM Generation modes, which permits the trade-off between a constant phase (Correct Phase PWM) and a higher maximum operation frequency (Fast PWM). Also, the TCA has a buffering scheme that ensures a glitch-free PWM.

Both the TCA and TCB can be used to generate a PWM signal with a high maximum operating frequency. Only the TCA can be used in critical applications due to its dual-slope PWM capabilities given by 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.

The buffering scheme contains a buffer for each Compare register as well as for the Period register. The use of these buffers is essential in critical applications where an unexpected long pulse can lead to a short circuit. Moreover, the presence of these buffers 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 buffered and unbuffered operations.

Figure 4-1. Unbuffered Dual-Slope Operation

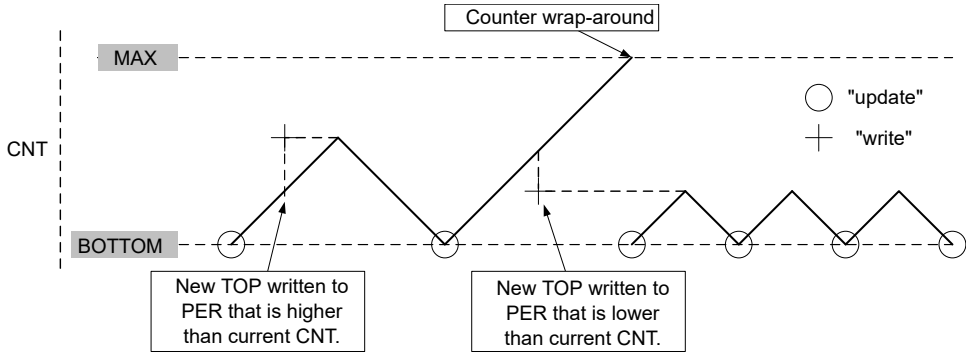
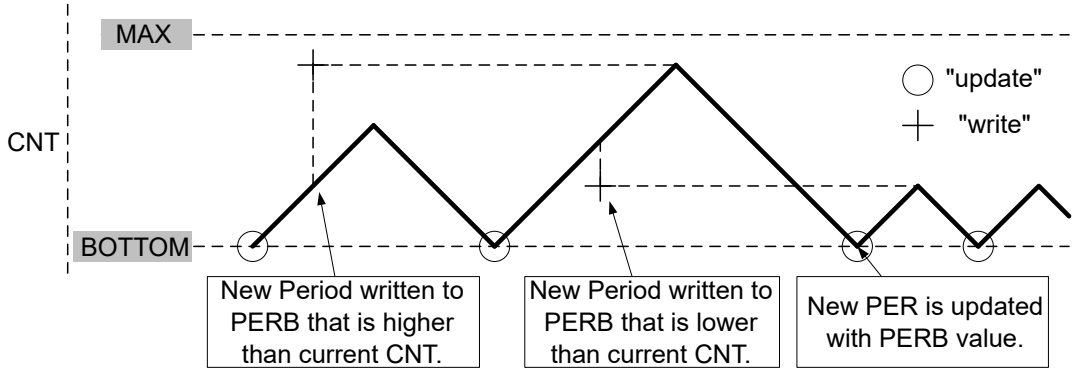


Figure 4-2. Changing the Period Using Buffering



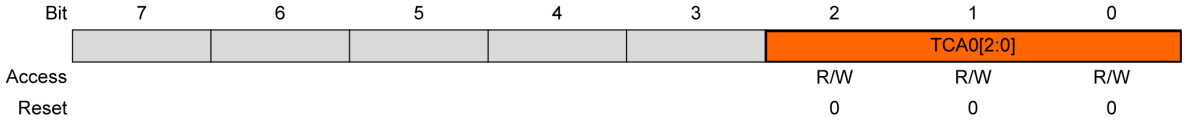
If the user changes the Period register directly, it is possible that the timer has already passed the new threshold, so it will continue counting to the maximum value. That will cause an unusually long pulse that can cause further problems. Also, if two or more compare channels are used and one of them is updated, the sync between the triggers may be lost. To prevent all these possible problems, the use of 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 TCA instance to generate a 1 kHz PWM signal with a 50% duty cycle using the buffering scheme described above.

1. The TCA 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.TCAROUTEA = PORTMUX_TCA0_PORTA_gc;
```

Figure 4-3. PORTMUX Control for TCA



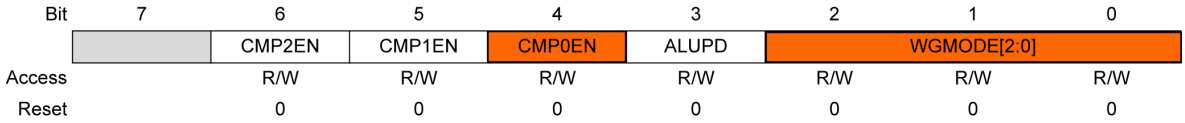
bits 2:0 TCA0[2:0]: TCA0 bits
Write these bits to select alternative output pins for TCA0.

Value	Name	Description
0x0	PORTA	TCA0 pins on PA[5:0]
0x1	PORTB	TCA0 pins on PB[5:0]
0x2	PORTC	TCA0 pins on PC[5:0]
0x3	PORTD	TCA0 pins on PD[5:0]
0x4	PORTE	TCA0 pins on PE[5:0]
0x5	PORTF	TCA0 pins on PF[5:0]
Other	-	Reserved

2. The CTRLB register contains the Enable bits of the compare channels and the bit field that determines the Waveform Generation mode. In this example, channel 0 is used together with a Dual-Slope PWM mode.

```
TCA0.SINGLE.CTRLB = TCA_SINGLE_CMP0EN_bm | TCA_SINGLE_WGMODE_DSBOTTOM_gc;
```

Figure 4-4. CTRLB Register



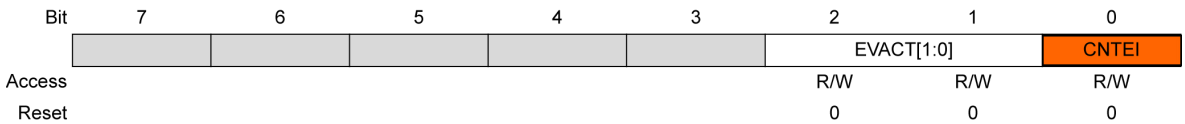
bits 2:0 WGMODE[2:0]: Waveform Generation Mode bits
These bits select the Waveform Generation mode.

Value	Name	Description
0x0	NORMAL	Normal Operation mode
0x1	FRQ	Frequency mode
0x3	SINGLESLOPE	Single-Slope PWM mode
0x5	DSTOP	Dual-Slope PWM mode
0x6	DSBOTH	Dual-Slope PWM mode
0x7	DSBOTTOM	Dual-Slope PWM mode
Other	-	Reserved

3. Set the CNTEI bit of the EVCTRL register to '0' to set the timer to count clock ticks instead of events. It is worth mentioning that this is the default value of the CNTEI bit.

```
TCA0.SINGLE.EVCTRL &= ~(TCA_SINGLE_CNTEI_bm);
```

Figure 4-5. EVCTRL Register



4. PERBUF is the buffer of the Period register. It is used to set the frequency of the PWM signal using the following formula.

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

Considering the targeted values for this example,

$$TCA_{period} = \frac{f_{CLK}(Hz)}{2 \times TCA_{prescaler} \times f_{DS\ PWM}(Hz)} = \frac{3333333}{2 \times 4 \times 1000} \cong 416 = 0x1A0$$

```
TCA0.SINGLE.PERBUF  = 0x01A0;
```

5. Also, the Compare register is updated using its buffer to set the duty cycle. The value in the Compare register is half of the one in the Period register because a 50% duty cycle is desired.

```
TCA0.SINGLE.CMP0BUF = 0x00D0;
```

6. Set the prescaler to 4 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.

```
TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV4_gc | TCA_SINGLE_ENABLE_bm;
```

Figure 4-6. CTRLA Register

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

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

Value	Name	Description
0x1	DIV2	$f_{TCA} = f_{CLK\_PER}/2$
0x2	DIV4	$f_{TCA} = f_{CLK\_PER}/4$
0x3	DIV8	$f_{TCA} = f_{CLK\_PER}/8$

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

```
PORTA.DIR |= PIN0_bm;
```

Tip: The full code example is also available in the Appendix section.

View the ATmega4809 Code Example on GitHub
Click to browse repository

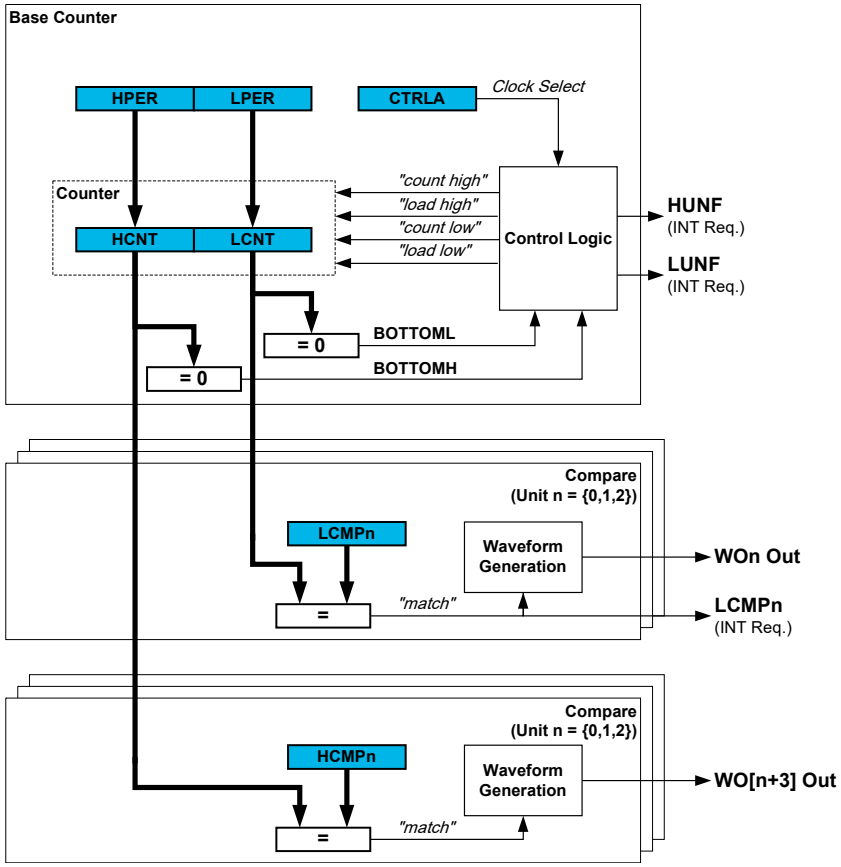
An MCC generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:

View the AVR128DA48 Code Example on GitHub
Click to browse repository

5. Generating Two PWM Signals in Split Mode

A TCA instance can be split into two completely independent 8-bit timers. This feature provides a high degree of flexibility, being extremely helpful in waveform generation applications. Except for the cases where high accuracy signals are required, most of the applications can be designed using 8-bit signal generators, and the possibility of adding one more generator to the design can be a huge advantage. Though, there are more limitations in the Split mode than the Counter registers dimension. The most important one is that both 8-bit timers have only the down-count option, so the Dual-Slope PWM mode becomes unavailable. Also, the buffering scheme cannot be used, and the timers can no longer count events, only clock ticks. Moreover, there are no interrupts or flags for high-byte Compare registers. Regardless of these limitations, the Split mode can be attractive when there is a need for a high number of timers. The block diagram of the TCA in Split mode is provided below.

Figure 5-1. Timer/Counter Block Diagram Spit Mode

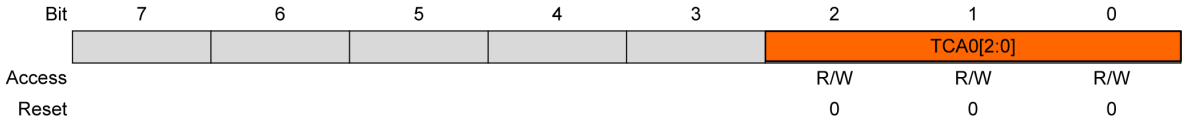


This mode will be put forward by generating two PWM signals with different frequencies and different duty cycles.

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

```
PORTMUX.TCAROUTEA = PORTMUX_TCA0_PORTA_gc;
```

Figure 5-2. TCAROUTEA Register



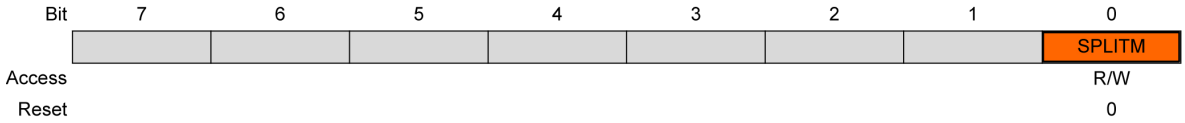
bits 2:0 TCA0[2:0]: TCA0 bits
Write these bits to select alternative output pins for TCA0.

Value	Name	Description
0x0	PORTA	TCA0 pins on PA[5:0]
0x1	PORTB	TCA0 pins on PB[5:0]
0x2	PORTC	TCA0 pins on PC[5:0]
0x3	PORTD	TCA0 pins on PD[5:0]
0x4	PORTE	TCA0 pins on PE[5:0]
0x5	PORTF	TCA0 pins on PF[5:0]
Other	-	Reserved

2. Enable the Split mode by setting the corresponding bit in the CTRLD register.

```
TCA0.SPLIT.CTRLD = TCA_SPLIT_SPLITM_bm;
```

Figure 5-3. CTRLD Register

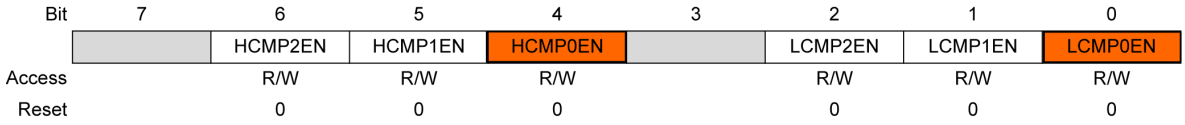


bit 0 SPLITM: Enable Split Mode bit
This bit sets the timer/counter in Split mode operation.

3. The CTRLB register contains the Enable bits of the compare channels. In this example, channel 0 of the lower byte of the timer and channel 0 of the higher byte of the timer are used.

```
TCA0.SPLIT.CTRLB = TCA_SPLIT_HCMP0EN_bm | TCA_SPLIT_LCMP0EN_bm;
```

Figure 5-4. CTRLB Register - Split Mode



bit 4 HCMP0EN: High byte Compare 0 Enable bit

See LCMP0EN.

bit 0 LCMP0EN: Low byte Compare 0 Enable bit

4. In this mode, the Period register and the Compare registers are split in half. Each half of the Period register determines the frequency of the respective PWM signal. Using the desired frequency value, the Period register value can be deduced from the following formula:

$$f_{ss\,PWM} = \frac{f_{CLK}(Hz)}{TCA_{prescaler} \times \left(TCA_{period} + 1\right)}$$

Considering the targeted values for this example,

$$TCA_{period}\,1 = \frac{f_{CLK}(Hz)}{TCA_{prescaler} \times f_{ss\,PWM1}(Hz)} - 1 = \frac{3333333}{16 \times 1000} - 1 \cong 207 = 0xCF$$

$$TCA_{period}\,2 = \frac{f_{CLK}(Hz)}{TCA_{prescaler} \times f_{ssPWM2}(Hz)} - 1 = \frac{3333333}{16 \times 3000} - 1 \cong 68 = 0x44$$

This translates to the following lines of code:

```
TCA0.SPLIT.LPER = 0xCF;

TCA0.SPLIT.HPER = 0x44;
```

5. Each half of the Compare registers determines the duty cycle of the respective PWM signal.

```
TCA0.SPLIT.LCMP0 = 0x68;

TCA0.SPLIT.HCMP0 = 0x11;
```

6. From the CTRLA register, the prescaler is set to 16. To start the counter, the user must set the Enable bit in the same register.

```
TCA0.SPLIT.CTRLA = TCA_SPLIT_CLKSEL_DIV16_gc | TCA_SPLIT_ENABLE_bm;
```

7. The initialization code provided illustrates a simple way of configuring the TCA in Split mode, but some mentions must be made. The Single Slope PWM mode is the only Waveform Generation mode available. Also, it is recommended to stop the timer and to do a hard Reset before switching from Normal mode to Split mode. An example is provided below. Clear the Enable bit in the CTRLA register to stop the counter. Then, in the Command bit field of the CTRLESET register, the user will write the code of the hard Reset command.

```
void TCA0_hardReset(void)
{
    TCA0.SINGLE.CTRLA &= ~(TCA_SINGLE_ENABLE_bm);

    TCA0.SINGLE.CTRLESET = TCA_SINGLE_CMD_RESET_gc;
}
```

8. Then, pins 0 and 3 of Port A (PA0 and PA3) are set as outputs by writing a '1' to each corresponding bit in the Direction register of the port.

```
PORTA.DIR |= PIN0_bm | PIN3_bm;
```

Tip: The full code example is also available in the Appendix section.

View the ATmega4809 Code Example on GitHub
Click to browse repository

An MCC generated code example for AVR128DA48 with the same functionality as the one described in this section can be found here:

View the AVR128DA48 Code Example on GitHub
Click to browse repository

6. References

Use the following links to find more information about the TCA operation modes:

1. ATmega4809 product page: www.microchip.com/wwwproducts/en/ATMEGA4809
2. megaAVR® 0-series Manual (DS40002015)
3. ATmega809/1609/3209/4809 – 48-Pin Data Sheet – megaAVR® 0-series (DS40002016)
4. ATmega4809 Xplained Pro web page: https://www.microchip.com/developmenttools/ProductDetails/atmega4809-xpro
5. AVR128DA48 Product Page: www.microchip.com/wwwproducts/en/AVR128DA48
6. AVR128DA48 Curiosity Nano Evaluation Kit web page: www.microchip.com/Developmenttools/ProductDetails/DM164151
7. AVR128DA28/32/48/64 AVR® DA Family Data Sheet (DS40002183)
8. Getting Started with the AVR® DA Family (DS00003429)

7. Revision History

Document Revision	Date	Comments
B	01/2021	The GitHub repository links are updated. Added the *AVR® DA Family Overview*, *References* and *Revision History* sections. Added MCC versions for each use case running on AVR128DA48. Other minor editorial corrections.
A	11/2018	Initial document release.

8. Appendix

Example 8-1. Using Periodic Interrupt Mode Full Code Example

```
#define PERIOD_EXAMPLE_VALUE    (0x0CB6)

#include <avr/io.h>
#include <avr/interrupt.h>
/*Using default clock 3.33MHz */

void TCA0_init(void);
void PORT_init(void);

void TCA0_init(void)
{
    /* enable overflow interrupt */
    TCA0.SINGLE.INTCTRL = TCA_SINGLE_OVF_bm;

    /* set Normal mode */
    TCA0.SINGLE.CTRLB = TCA_SINGLE_WGMODE_NORMAL_gc;

    /* disable event counting */
    TCA0.SINGLE.EVCTRL &= ~(TCA_SINGLE_CNTEI_bm);

    /* set the period */
    TCA0.SINGLE.PER = PERIOD_EXAMPLE_VALUE;

    TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV256_gc         /* set clock
source (sys_clk/256) */
                      | TCA_SINGLE_ENABLE_bm;               /* start timer */
}
void PORT_init(void)
{
    /* set pin 0 of PORT A as output */
    PORTA.DIR |= PIN0_bm;
}

ISR(TCA0_OVF_vect)
{
    /* Toggle PIN 0 of PORT A */
    PORTA.OUTTGL = PIN0_bm;

    /* The interrupt flag has to be cleared manually */
    TCA0.SINGLE.INTFLAGS = TCA_SINGLE_OVF_bm;
}

int main(void)
{
    PORT_init();

    TCA0_init();

    /* enable global interrupts */
    sei();

    while (1)
    {
        ;
    }
}
```

Example 8-2. Generating a Dual-Slope PWM Signal Full Code Example

```
#define PERIOD_EXAMPLE_VALUE        (0x01A0)
#define DUTY_CYCLE_EXAMPLE_VALUE    (0x00D0)

#include <avr/io.h>
/*Using default clock 3.33MHz */

void TCA0_init(void);
```

```
void PORT_init(void);

void TCA0_init(void)
{
    /* set waveform output on PORT A */
    PORTMUX.TCAROUTEA = PORTMUX_TCA0_PORTA_gc;

    TCA0.SINGLE.CTRLB = TCA_SINGLE_CMP0EN_bm             /* enable compare
channel 0 */
                      | TCA_SINGLE_WGMODE_DSBOTTOM_gc;   /* set dual-slope PWM
mode */

    /* disable event counting */
    TCA0.SINGLE.EVCTRL &= ~(TCA_SINGLE_CNTEI_bm);

    /* set PWM frequency and duty cycle (50%) */
    TCA0.SINGLE.PERBUF  = PERIOD_EXAMPLE_VALUE;
    TCA0.SINGLE.CMP0BUF = DUTY_CYCLE_EXAMPLE_VALUE;

    TCA0.SINGLE.CTRLA = TCA_SINGLE_CLKSEL_DIV4_gc        /* set clock source
(sys_clk/4) */
                      | TCA_SINGLE_ENABLE_bm;            /* start timer */
}

void PORT_init(void)
{
    /* set pin 0 of PORT A as output */
    PORTA.DIR |= PIN0_bm;
}

int main(void)
{
    PORT_init();

    TCA0_init();

    /* Replace with your application code */
    while (1)
    {
        ;
    }
}
```

Example 8-3. Generating Two PWM Signals in Split Mode Full Code Example

```
#define PERIOD_EXAMPLE_VALUE_L        (0xCF)
#define PERIOD_EXAMPLE_VALUE_H        (0x44)
#define DUTY_CYCLE_EXAMPLE_VALUE_L    (0x68)
#define DUTY_CYCLE_EXAMPLE_VALUE_H    (0x11)

#include <avr/io.h>
/*Using default clock 3.33MHz */

void TCA0_init(void);
void PIN_init(void);
void TCA0_hardReset(void);

void TCA0_init(void)
{
    /* set waveform output on PORT A */
    PORTMUX.TCAROUTEA = PORTMUX_TCA0_PORTA_gc;

    /* enable split mode */
    TCA0.SPLIT.CTRLD = TCA_SPLIT_SPLITM_bm;

    TCA0.SPLIT.CTRLB = TCA_SPLIT_HCMP0EN_bm         /* enable compare channel
0 for the higher byte */
                     | TCA_SPLIT_LCMP0EN_bm;        /* enable compare channel
0 for the lower byte */

    /* set the PWM frequencies and duty cycles */
    TCA0.SPLIT.LPER = PERIOD_EXAMPLE_VALUE_L;
    TCA0.SPLIT.LCMP0 = DUTY_CYCLE_EXAMPLE_VALUE_L;
```

```
    TCA0.SPLIT.HPER = PERIOD_EXAMPLE_VALUE_H;
    TCA0.SPLIT.HCMP0 = DUTY_CYCLE_EXAMPLE_VALUE_H;

    TCA0.SPLIT.CTRLA = TCA_SPLIT_CLKSEL_DIV16_gc    /* set clock source
(sys_clk/16) */
                     | TCA_SPLIT_ENABLE_bm;         /* start timer */
}

void PIN_init(void)
{
    PORTA.DIR |= PIN0_bm    /* set pin 0 of PORT A as output */
              | PIN3_bm;    /* set pin 3 of PORT A as output */
}

/* must be used when switching from single mode to split mode */
void TCA0_hardReset(void)
{
    /* stop timer */
    TCA0.SINGLE.CTRLA &= ~(TCA_SINGLE_ENABLE_bm);

    /* force a hard reset */
    TCA0.SINGLE.CTRLESET = TCA_SINGLE_CMD_RESET_gc;
}

int main(void)
{
    PIN_init();

    TCA0_init();

    while (1)
    {
        ;
    }
}
```

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 devices:

- 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 and under normal conditions.
- There are dishonest and possibly illegal methods being used in attempts to breach the code protection features of the Microchip devices. We believe that these methods require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Attempts to breach these code protection features, most likely, cannot be accomplished without violating Microchip's intellectual property rights.
- Microchip is willing to work with any customer who is concerned about the integrity of its code.
- 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. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Legal Notice

Trademarks

ISBN: 978-1-5224-7497-5

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