Atmel

AVR 8-bit Microcontrollers

AVR131: Using the AVR's High-speed PWM

APPLICATION NOTE

Introduction

This application note is an introduction to the use of the high-speed Pulse Width Modulator (PWM) available in some Atmel® tinyAVR® microcontrollers such as Atmel ATtiny26, Atmel ATtiny15, etc.

The advantage of high-speed PWM is the increased bandwidth of the analog output signal. The high frequency further allows for smaller and less expensive filter components to be used in shaping the signal.

- The fast PWM is used to generate a pulse train with varying duty-cycle
- The PWM output is generated on the OC1A output pin (PB1)
- An analog filter can be used to shape the digital PWM output to obtain an analog signal such as a sine wave
- Assembly and C code examples are provided to show the usage of high-speed PWM in the ATtiny26

Features

- Analog waveform generation using high-speed PWM
- Example application to generate sine wave
- High-speed pre-scalable PWM clock

Table of Contents

1. Glossary

DAC	Digital to Analog Converter
ISR	Interrupt Service Routine
kHz	KiloHertz
MHz	MegaHertz
PLL	Phase-Locked Loop
PWM	Pulse Width Modulation
RC-filter	Resistor-Capacitor filter

2. Description

This chapter provides an insight to the theory on high-speed PWM and its usage in generating analog waveform. The high-speed PWM output, combined with an analog filter can be used to generate analog output signals, i.e. a digital to analog converter (DAC). A digital pulse train with a constant period (fixed base frequency) is used to generate an analog signal. Different analog levels can be generated by varying the duty cycle (and thereby the pulse width) of the digital signal. If a high analog level is needed, the duty cycle is increased and vice versa. The analog waveform considered in this application note is Sine Wave.

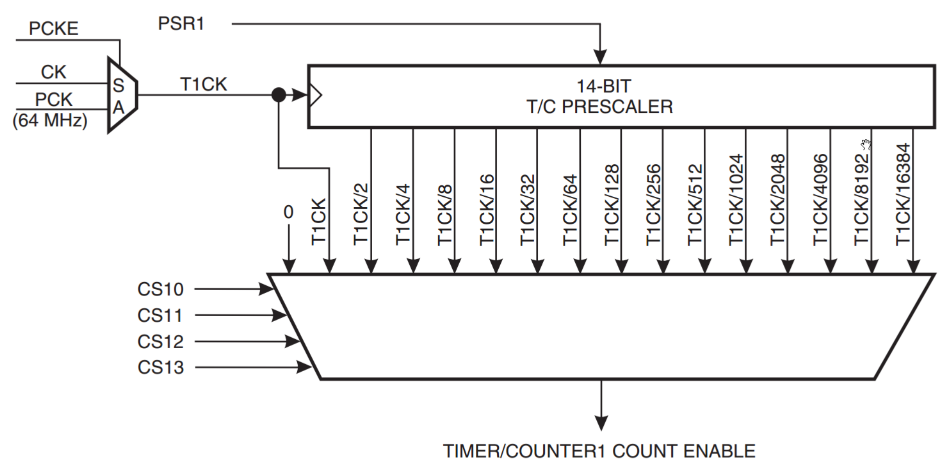
2.1. High-speed PWM

In the AVR®, the timer/counters can be used to generate PWM signals. PWM base frequency is determined by timer clock frequency and top counter value. Faster clock frequency will increase the PWM base frequency and vice versa. Lower top value will reduce the time for overflow, which thereby increases the PWM frequency. The PWM base frequency can be calculated as follows:

PWM Frequency = Timer clock speed / Timer resolution

ATtiny26 devices features the high-speed PWM, which allows the user to run the timer at a higher speed than the CPU clock. Timer/Counter1 supports two accurate, high speed, 8-bit Pulse Width Modulators using clock speeds up to 64MHz. It has two clocking modes; synchronous mode and asynchronous mode. The synchronous mode uses the system clock (CK) as the clock timebase and asynchronous mode uses the fast peripheral clock (PCK) as the clock time base. Asynchronous mode can be enabled by setting the PCKE bit from the PLLCSR Register (PLLCSR > PCKE = 1). Timer/Counter1 features a prescaler setting, which provides clock selections between PCK to PCK/16384 in asynchronous mode. Setting the PSR1 bit in TCCR1B Register resets the prescaler. The block diagram of Timer/Counter1 Prescaler is shown in the following diagram:

Figure 2-1. Timer/Counter1 Prescaler



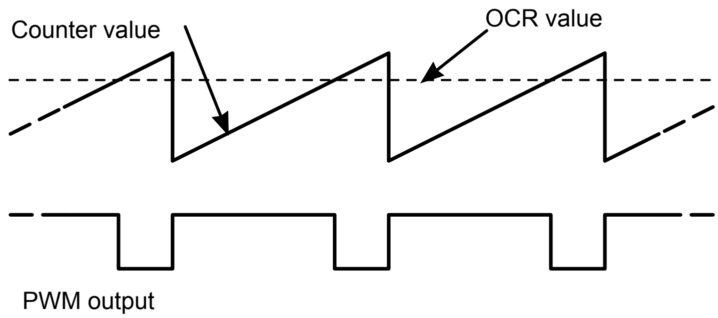
Note: Refer the device datasheet for more details on configuration and usage of internal RC Oscillator and PLL.

The PLL of Timer/Counter1 needs a 1MHz reference clock. We can use the internal RC oscillator to generate 1MHz by configuring the OSCCAL register. This clock output will be used as reference to PLL which can generate a recommended maximum frequency of 64MHz. Hence, the maximum PWM frequency that could be generated with best resolution is given as follows:

PWM Frequency = 64MHz / 256 = 250kHz

Increasing the base frequency beyond this will be at the expense of reduced resolution, since fewer steps are available from 0% to 100% duty cycle. Altering the value of the Output Compare Registers (OCR) changes the duty cycle. Increasing the OCR value increases the duty cycle. The PWM output is high until the OCR value is reached, and low until the timer reaches the top value and wraps back to 0. This is shown in following figure.

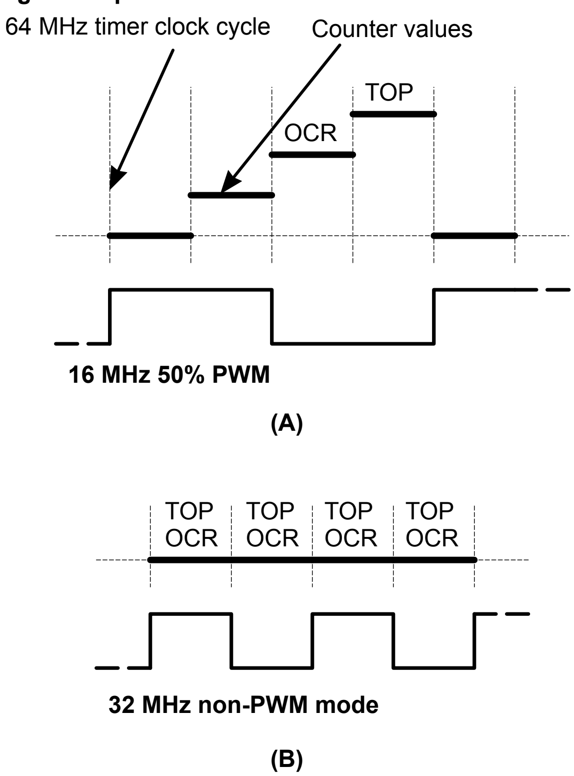
Figure 2-2. Counter Values and PWM Output



With 64MHz timer clock and a top value 3, PWM base frequency of 16MHz can be achieved. However, the OCR value is now limited to 0, 1 (25% duty cycle), 2 (50% duty cycle), or 3 (100% duty cycle). This shows that lowering the top value can increase the PWM base frequency, but reduces the resolution.

To achieve the maximum output frequency from the timer, it must be run in non-PWM mode. Both the OCR value and the top value must be set to 0. The counter is then stuck at 0. Setting the Output Compare Match action to ‘toggle output’ makes the timer toggle the output on every timer clock tick. The result is a 32MHz signal, as shown in following figures.

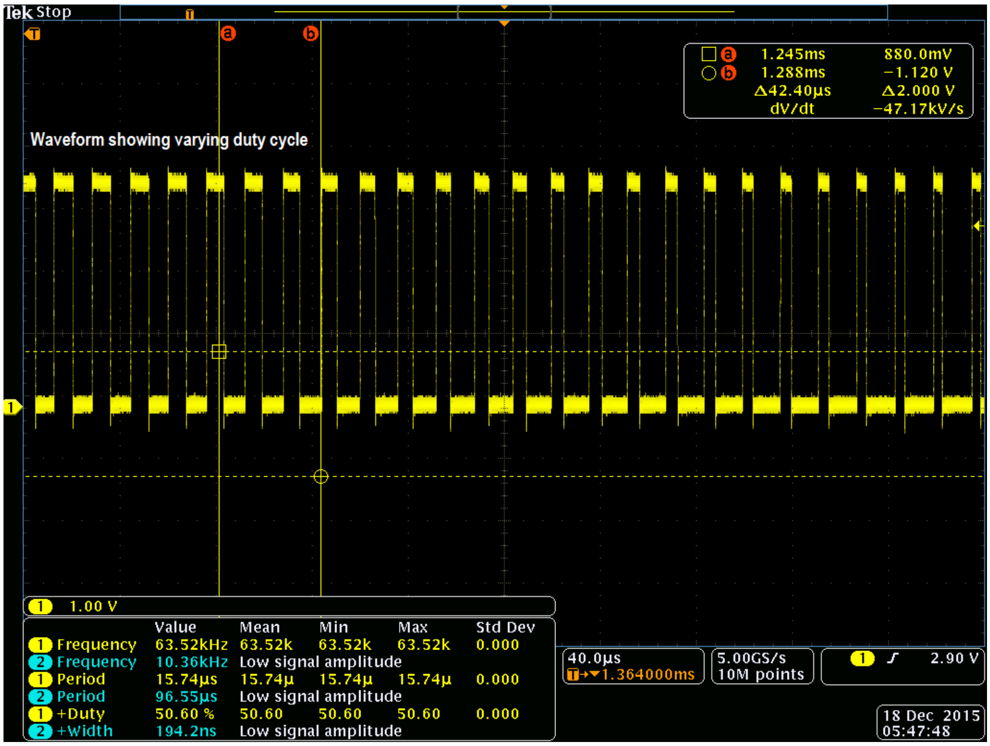
Figure 2-3. High Frequency Digital Output



2.2. Analog Waveform Generation from PWM Signal

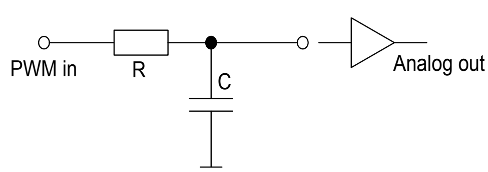
Analog waveforms can be generated by averaging the PWM signals over one period using simple low-pass filters. In this application note, implementation of sine wave generation from high-speed PWM output is explained. If high-speed PWM is used to generate analog signals, the step-size between the analog levels depends on the resolution of the PWM signal. Duty cycle of the PWM signal determines the amplitude of the analog waveform. A duty cycle of 50% gives an analog signal with half the supply voltage, while 75% duty cycle gives an analog signal with 75% supply voltage. A real-time example of PWM waveform with varying duty cycle is shown in the following figure.

Figure 2-4. PWM Output with Varying Duty Cycle



The analog low-pass filter could be a simple passive RC-filter for instance. The filter removes the high PWM base frequency and lets through the analog signal. The filter crossover frequency must be chosen high enough to not alter the analog signal of interest. At the same time, it must be as low as possible to minimize the ripple from the PWM base frequency. The higher the base frequency is, the easier it is to attenuate the base frequency and thereby minimize the signal ripple. The selection of resolution versus base frequency is thus an application dependent trade-off.

Figure 2-5. Low-Pass RC-Filter



If the analog signal is fed to a low-impedance input, a buffer amplifier should be connected between the filter output and the load. This will prevent the load from discharging the capacitor and creating ripple voltages.

2.3. Sine Wave Generation

As seen in the previous section, amplitude of the analog waveform is directly proportional to duty cycle of PWM signal. So, if a sine wave is to be generated, the duty cycle of the PWM signal has to be varied accordingly. In ATtiny26, the OCR value determines the duty cycle of the PWM signal. In order to control the variation in amplitude, the duty cycle has to be varied, thereby generating a sine wave. The application uses a sine table that has the sample values (duty cycle of PWM), which will be loaded at every Timer overflow. To generate a sine table, the following parameters must be considered:

Table 2-1. Configuration Parameters

Parameter	Value used in the application
CPUFreq	8MHZ
TimerFreq	16MHZ
TimerTop	255
PWMFreq	62.5kHz
SineFreq	~244Hz

1. **CPUFreq:** This is the frequency at which the system is operated. In the application demonstrated, the CPU runs at 8MHz using Internal RC Oscillator.
2. **TimerFreq:** This is the frequency at which the Timer/Counter1 is operated. In the application demonstrated, 1MHz output of Internal RC Oscillator acts as Timer1 PLL reference to generate a 64MHz clock. This clock is then prescaled down to 16MHz to demonstrate the prescaling capability of Timer/Counter1.
3. **TimerTop:** This is the Timer/Counter1 top value that is one of the deciding parameters of PWM frequency. In the application demonstrated 255 is loaded to 8-bit register OCR1C.
4. **PWMFreq:** This is the frequency of the PWM output. It can be calculated by **PWMFreq = TimerFreq / TimerTop**.
5. **SineFreq:** This is the frequency of Sine Wave generated after passing the PWM signal to a Low-Pass filter. This can be calculated by the formula, **PWMFreq / TimerTop**.
6. **SampleValue:** This is the instantaneous sample of Sine Wave that will be loaded to OCR1A on every Timer overflow. The sample value decides the duty cycle of PWM which in turn controls the amplitude of the analog waveform (sine wave) generated. The formula to generate **SampleValue** is shown as follows:

$$SampleValue_n = \left(\frac{TimerTop - 1}{2}\right) * Sin\left(\frac{2 * \pi * n}{TimerTop}\right) + \left(\frac{TimerTop + 1}{2}\right)$$

Note:

1. Here, **n** is the number of samples in one cycle of sine wave. It ranges from **0** to **TimerTop**.
2. It has to be noted that the CPU should be running fast enough to load the duty cycle values to the OCR1A register after every Timer1 overflow.

Configurations to generate different frequencies of sine wave are listed in the following table:

Table 2-2. Various Combinations to Generate Desired Frequency of Sine Wave

CPUFreq [MH]	TimerFreq [MH]	PWMFreq [kH]	SineFreq [Hz]	TimerTop	Number of CPU cycles before overflow
8	16	62.5	250	249	125
8	16	62.5	500	124	62.5
8	16	62.5	800	77	39
16	16	62.5	800	77	78
16	16	62.5	1000	62	63
16	16	62.5	1500	41	42

Note: The values shown in the table has some restrictions on CPU speed and PWM speed. For more details, refer Interrupt Service Routine on page 10.

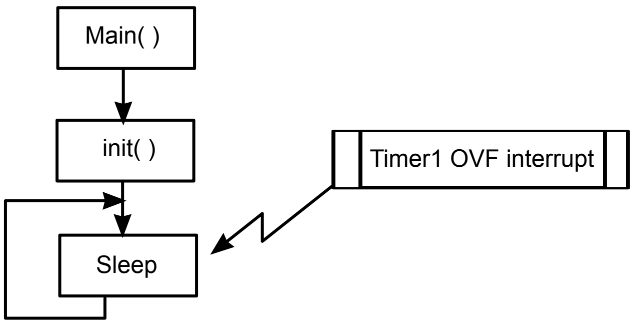
3. Application Example

This application note comes with an example code in both **C & Assembly** language to generate sine wave from High-Speed PWM of Atmel ATtiny26. This section discusses more about the firmware, limitations, and design parameters that user shall consider before using the code in their application.

3.1. Firmware - Assembly Code

The implementation is such that CPU runs at 8MHz and Timer1 is clocked by 16MHz clock. Necessary fuses are to be programmed accordingly. The code consists of three parts: Initialization, Timer1 overflow interrupt service routine, and a sleep loop.

Figure 3-1. Flowchart of main() - Sine Wave Generator Code

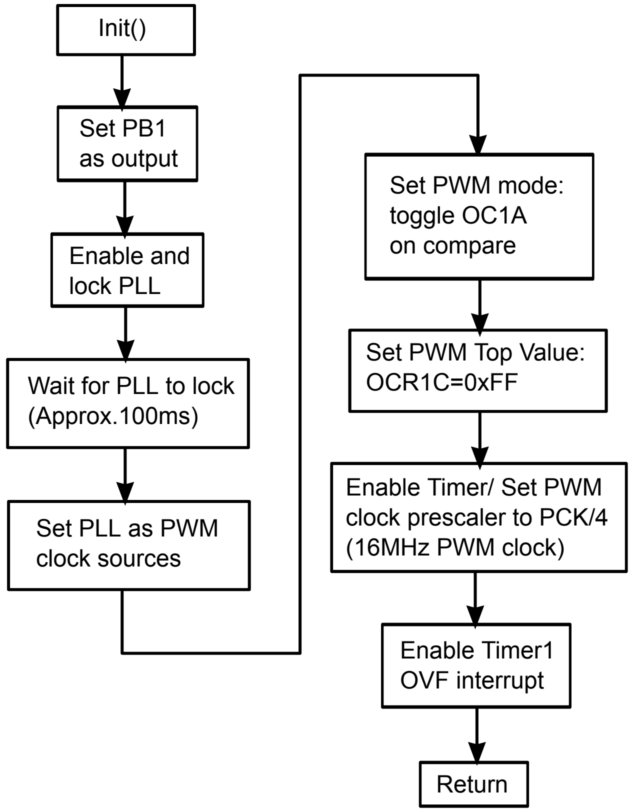


3.1.1. Initialization

To generate an output from the PWM the Output Compare Pin of Timer1 (OC1A) is set up as output.

1. The clock source for the timer is prepared by starting the PLL and locked to the system clock (required).
2. The PLL takes approximately 100ms to lock onto the system clock and it is therefore necessary to wait for the PLL lock flag before proceeding.
3. When the PLL is locked it is selected as clock source for the timer.
4. The PWM mode is selected so that the OC1A pin toggles on compare match.
5. Top value of the timer is set to 0xFF. The Top value affects the resolution and the base frequency of the PWM – the higher the Top value is the higher resolution and the lower base frequency.
6. The prescaler is set, which also starts the timer.
7. The Overflow interrupt is enabled.

Figure 3-2. Flowchart of init() - Initialize Timer1 and I/O Pin for Fast PWM Mode



3.1.2. Interrupt Service Routine

When the Timer1 value reaches the OCR1C value (0xFF), the Timer Overflow interrupt service routine (ISR) is executed. This happens at a constant interval, since OCR1C is constant. This interval is the base frequency of the fast PWM output signal.

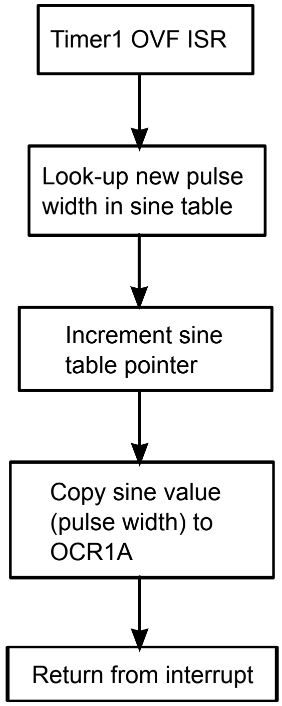
The sine table generated as explained in Sine Wave Generation on page 7 is stored in the Flash memory as a Look-Up Table. When the ISR is executed, the value from sine table is loaded to OCR1A register. On each look-up the index to the look-up table is incremented so that consecutive values can be loaded. In this way the pulse width is modulated to the sine wave.

Note: The OCR1A register is buffered and that the latching of the buffer into the actual OCR1A register takes place on the timer overflow.

The interrupt routine in assembly language takes 13 clock cycles to execute. The call to and return from the interrupt comes in addition – in total 21 system clock cycles. Since Timer1 is an 8-bit timer with top value as 255, the interrupt occurs every **TimerTop / (TimerFreq / CPUFreq) > 256 / 2 = 128 cycles**.

If the Timer1 clock is configured to 64MHz and CPU is operated at 8MHz, timer overflow interrupt occurs every 32 cycles of system clock (which is within the limits of the application). Though it is possible to clock the Timer1 with the maximum frequency of 64MHz, the PLL output is prescaled by 4 to 16MHz, to illustrate the use of the prescaler.

Figure 3-3. Flowchart of OC1A_isr() - Timer1 ISR to Load Sine Table in OCR1A



3.1.3. Sleep Mode

The sleep mode **Idle** is used to put the device into power reduction state while waiting for the Interrupt to occur. When the interrupt is serviced, it returns back to sleep.

3.2. Firmware - C Code

Implementation of `Initialization` and `Interrupt Service Routine` in **C** language is the same as that of **Assembly** language. For more details, refer Firmware - Assembly Code on page 9. It is to be noted that ISR implemented in **C** language takes 46 CPU cycles with Optimization set to **Os**.

3.2.1. Sleep Mode - C Code

The following code snippet is added to enable Sleep mode for the device.

```
sleep_enable();
while(1)
{    //Enter Sleep mode
     sleep_mode();
}
```

The functions `sleep_enable() & sleep_mode()` are predefined directives, defined in `sleep.h`.

3.3. Application Considerations

The following considerations must be taken care of while implementing the application

1. The RC filter used in this application is the simplest component to generate analog signal from PWM output. For analog signal output with lesser ripples, noise components, etc., better signal conditioning must be performed.
2. The output of the RC Filter will be an analog signal with reduced amplitude. Based on the need, a suitable amplifier circuit can be added to solve the purpose.

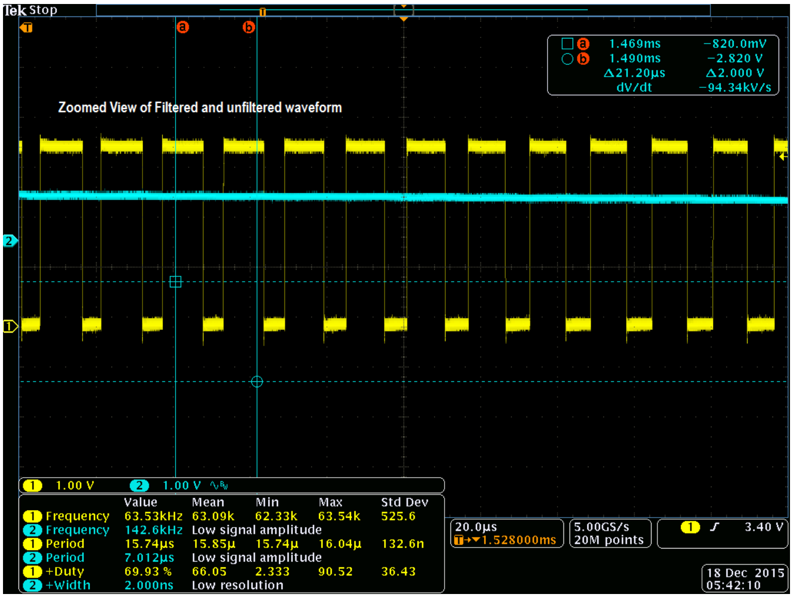
3. The PWM frequency can be increased further by reducing the resolution of the sine-wave. However, reduced resolution results in increased step size of the waveform, which would result in ripples and non-smooth waveform.
4. The design parameters should be configured such that **TimerTop / (TimerFreq / CPUFreq)** shall be greater than 21 cycles for application in assembly language and greater that 46 cycles for application in C language. Breaching this may result in sine wave generated with undesired frequency. For different configurations, refer Table 2-2 Various Combinations to Generate Desired Frequency of Sine Wave on page 8.
5. Frequency of the output sine wave depends on the CPU frequency and Timer1 Clock frequency. The clock used from Timer1 is a PLL with internal RC oscillator as reference. Hence, it is necessary to calibrate the RC oscillator to generate the desired frequency of sine wave.

4. Results and Conclusion

The following screen-shots are examples of sine wave signals generated by Atmel ATtiny26 PWM for the configuration mentioned in Table 2-1 Configuration Parameters on page 7.

Figure 4-1. PWM Output and Sine Wave

Figure 4-2. PWM Output and Sine Wave – Zoomed View



Note:

1. Waveform in blue color is unfiltered PWM signal.
2. Waveform in Yellow color is filtered Sine signal.

The screen-shots show the output on the OC1A pin, which is the digital pulse modulated signal, and the filtered/shaped PWM signal. A simple RC filter is used to shape the PWM signal to a sine wave – an analog signal where the amplitude is controlled by the duty cycle of the PWM output. The RC filter used has R = 10kΩ and C = 100nF, resulting in a filter crossover frequency of 1kHz, which will let the low frequency sine wave pass while filtering out the high frequency PWM base.

5. References

1. AVR130: Using the timers on tinyAVR and megaAVR devices
2. AVR135: Using Timer Capture to Measure PWM Duty Cycle on tinyAVR and megaAVR devices
3. AVR205: Frequency measurement made easy with Atmel tinyAVR and Atmel megaAVR
4. AVR504: Migrating from ATtiny26 to ATtiny261/461/861

6. Revision History

Doc Rev.	Date	Comments
2542B	03/2016	Updated for Atmel Studio 7 and added C example code
2542A	09/2003	Initial document release

Atmel | Enabling Unlimited Possibilities®

Atmel Corporation 1600 Technology Drive, San Jose, CA 95110 USA **T:** (+1)(408) 441.0311 **F:** (+1)(408) 436.4200 | **www.atmel.com**