Tutorial 7

Pulse Width Modulation (PWM) for AVR Microcontroller

# Basic Pulse Width Modulation (PWM) – Manual PWM

In this task we will learn how to create a PWM signal using CTC mode of the Timer. This is considered the “manual” way to create a PWM signal. More will be discussed in the next tasks. For example, we want to create the **0.25Hz** frequency and play around with different DUTY cycle

Note that in this formula **we don’t divide by TWO** since we might generate the Time ON different with TIME OFF. We will use **OCR1A** to set up the frequency of the PWM and **OCR1B** for the Time ON

1. Connect a LED to PORTB0 – we will use this to indicate the frequency of the PWM.
2. We will start with some configuration for the Timer 1 and setup for the CTC Interrupts for both channel A and channel B.



1. In the ISR for the CTC Interrupt for Channel A, turn off the LED.
2. Add also another ISR to service the CTC Interrupt for Channel B, turn on the LED.
3. The final code looks like below.



1. Build and download the code onto the board.
2. You shall see the time for the LED to be
   1. ON for 2 seconds
   2. OFF for 2 seconds
3. We will try the duty cycle to be 25 percent
   1. We will change the OCR1B to be
   2. Build and download the code onto the board. You shall see the time for the LED to be
      1. ON for 1 second
      2. OFF for 3 seconds

Note that in the above code, we used the CTC Interrupts to generate delay time for TIME ON (duty cycle) and TIME OFF, and in turn create the PWM signal. You can use other techniques for time delay creation to generate PWM signals as well – such as **\_delay\_ms(),** **normal counting mode**, **CTC mode without interrupt (like in Lab 1)**.

# Fast PWM Mode – Mode 7

In the previous task, we were “manually” creating PWM signal using the CTC interrupts. However, the AVR MCU has its own PWM modes. In this task, we will explore the first mode called Fast PWM Mode and change the Duty cycle

1. Connect an LED to the pin OC1A (Pin 15).

A picture containing graphical user interface

Description automatically generated

1. Let’s develop from the code from the previous task, starting by removing the interrupt related code segments



1. First to use the PWM, we need to set the mode. The **Fast PWM mode** is one of the working modes of the Timer so we need to configure corresponding registers in the Timer. From the datasheet, we can have a number of variation of fast PWM like below

A picture containing table

Description automatically generated

1. In this task, we will try the Fast PWM, 10 bit (Mode 7) by configuring **WGM1x** bits in TCCR1A and **TCCR1B** registers

Diagram

Description automatically generated with medium confidence

1. Next, we need to set the **Compare Output Mode** for OCR1A and OC1B bit. Let’s stick with the non-inverting mode for now by setting

A picture containing table

Description automatically generated

Text

Description automatically generated

1. Finally, to set the duty cycle for the PWM, we need to set the value for the register OC1RA. Few calculations:
   1. The top value of the counter is
   2. The PWM frequency for the output can be calculated by the following equation:

* 1. Duty cycle

Let’s say we start with the value of **OCR1A = 155,** the duty cycle is expected to be

1. The output from the PWM mode is taken from the pin **OC1A** (and/or **OC1B**) – or pin **PORTB1** (and/or **PORTB2**). To generate output, we need to set the pin as the output in **DDRB** register.
2. The final code looks like below.



Note in this code, we group all the configuration and initialisation into a routine

1. Build and download the code onto the board and then observe your LED, it should blinks quite fast but there is a still a distinction between ON and OFF.

*Note – some measurement was done last semester can be seen below to confirm the result*

Graphical user interface

Description automatically generated

1. Let’s change the duty cycle to 75% by replacing **OCR1A = 768**

Build and download the code onto the board and then observe your LED, it should blinks even faster. You shall see the time for turning OFF is quite short.

1. Let’s change the duty cycle to 90% by replacing **OCR1A = 920**

Build and download the code onto the board and then observe your LED, it should blinks even faster. You shall see the time for turning OFF is quite short, this is due to the higher duty cycle

In summary

|  |  |
| --- | --- |
| Fast PWM - MODE 7 | |
| OCR1A | Duty cycle |
| **920** |  |
| **768** |  |
| **155** |  |

As noticed above, the PWM frequency is defined by the TOP value of the counter. For the fix bit width (8-bit, 9-bit, 10-bit), the frequencies are fixed.

# Fast PWM Mode – Mode 15

In the previous section, we use Mode 7 where the fixed frequency is set due to the TOP value is fixed at maximum of the Timer’s resolution. To adjust the PWM frequency, we can use the mode 14 and 15 where the TOP value is set by OCR1A or ICR1. The next task, we will work with the mode 15.

Calendar

Description automatically generated

1. Change the Timer to **mode 15** by setting the **WGM1x** bits
2. It is important to highlight if OCR1A is used to set as the TOP, we can only create PWM signal with the **duty cycle of 50%**. We need to change the setting of the Compare Match Output.

A picture containing table

Description automatically generatedText

Description automatically generated with medium confidence

1. Let’s say we are trying to the PWM of 10Hz, we will use the same formula that is applied for CTC mode.
2. Modified code looks like (the old settings are commented).



1. Rebuild the code and download it onto the board. You can observe the LED and compare with the previous mode.

*Note – some measurement was done last semester can be seen below to confirm the result*

Graphical user interface

Description automatically generated

Optional Exercises

**You are recommended to attempt the following exercises and compare these with the Fast PWM mode earlier.**

# Phase Correct PWM Mode – Mode 3

In this task, we will work with the 2nd mode called Phase Correct PWM.

1. Keep the same setup as above
2. We start from the code from the previous task.
3. Like Fast PWM, we have several Phase Correct PWM modes. From the datasheet, we can have a number of variations of fast PWM like below

A screenshot of a computer

Description automatically generated with medium confidence

1. First, let’s try the Phase Correct, 10 bit (Mode 3) by configuring **WGM1x** bits in TCCR1A and **TCCR1B** registers

Diagram

Description automatically generated with medium confidence

1. Next, we need to set the **Compare Output Mode** for OCR1A and OC1B bit. Let’s stick with the non-inverting mode for now by setting

A picture containing table

Description automatically generated

Text

Description automatically generated

1. Finally, to set the duty cycle for the PWM, we need to set the value for the register OC1RA. Few calculations:
   1. The top value of the counter is
   2. The PWM frequency for the output can be calculated by the following equation:

* 1. Duty cycle

Let’s say we start with the value of **OCR1A = 155,** the duty cycle is expected to be

1. The output from the PWM mode is taken from the pin **OC1A** (and/or **OC1B**) – or pin **PORTB1** (and/or **PORTB2**). To generate output, we need to set the pin as the output in **DDRB** register.
2. The final code looks like below.



1. Build and download the code onto the board.

|  |  |
| --- | --- |
| Fast PWM - MODE 3 | |
| OCR1A | Duty cycle |
| **920** |  |
| **768** |  |
| **155** |  |

# Phase Correct PWM Mode – Mode 11

Similarly to the Fast PWM, Phase Correct PWM does also have the mode 11 in which the TOP value is set by OCR1A.

Calendar

Description automatically generated

Also. we can only create PWM signal with the **duty cycle of 50%**. We need to change the setting of the Compare Match Output like what we did in the fast PWM mode.

A picture containing table

Description automatically generatedText

Description automatically generated with medium confidence

1. We will use the OCR1A of 585 as we did in the previous task. Note that in this Phase Correct mode, the frequency is calculated differently
2. Modified code looks like



1. Rebuild the code and download it onto the board. Do you observe any difference?

# Fast PWM Mode

Check this link and analyse how people use Arduino built-in function to generate PWM signals

<https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM>