MCU Lab 3 - Week 9
MCU: PWM for a Servo

This computer lab will introduce a Timer 1 and its PWM (pulse width modulation) signal generation. Timer 1 is a 16-bit timer which can count from 0 to 65535 (2^{16}-1). It can be used to generate a wide range of PWM signals suitable for many servo motor controls. Timer 1 also has a unique functionality, called input capture, which can be used to precisely measure the pulse width or period of incoming signals. This "input capture" will be used for MCU Project in measureing the distance in a ultrasonic sensor.

1. Objectives

- To use the Fast PWM mode operation of Timer 1 and generate a PWM signal to control a servo motor provided in the kit (Model: SG90).
- To use a potentiometer (connected to ADC) to control the servo motor's angular position.
- (Extension) To be able to control the servo without using the Fast PWM mode but directly generating periodic pulses with required duty cycles.

2. Marking and Due Date

- Students can form a group (maximum of **two** students) to complete the lab tasks but the demonstration will be assessed **individually**. There is a total of **2.5 marks** allocated to this lab. You need to complete the tasks during the lab session or at latest before the following week lab (in which case, students need to upload a short video to Canvas capturing the operational tasks).

3. Activity #1: Preparation

- Construct a circuit, as shown in Fig. 1, connecting a servo motor (SG90) to PB1 (or UNO pin-9) and a potentiometer to an analog input. Note that the middle pin of the servo is 5V power, and the orange-colour cable is the PWM signal connected to PB1.
- Connect the potentiometer output (pin 2) to one of the ADC input pins. The pin number is selected based on the last digit of your student ID number. Take the % (modulus, or remainder operator) of 6 of the last digit. For example, if your last digit of SID is 8, then 8 % 6 gives 2 (the remainder). Connect the potentiometer to A2 pin (in UNO board). Connect an LED at PB4 (not shown in the figure).

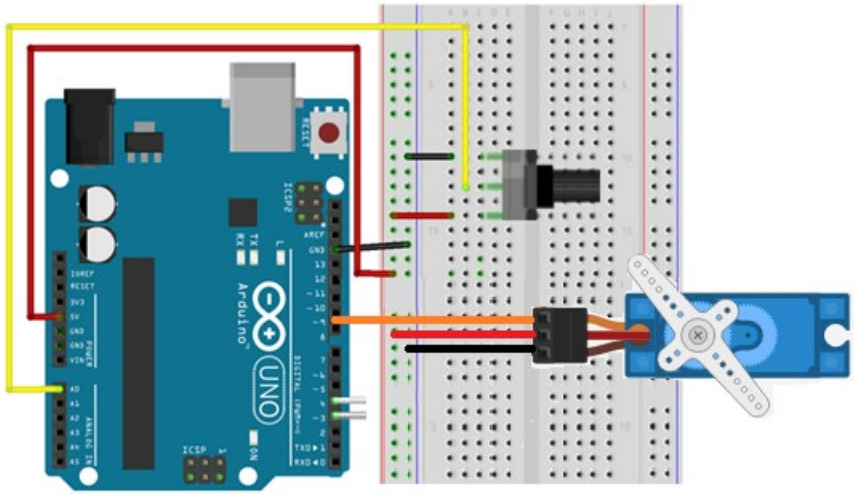

Figure 1. A Servo and a potentiometer connected to Atmega328P

- Note that a servo motor can hold its angular position with high torque (which is one of the main reasons why they are popular for RC toys). However, high torque requires high current and thus require a separate servo power source. In our experiment, we will use the UNO power with no load to the servo (be careful not to force the servo to turn while running as it can cause a reset in the UNO board and, in worst case corrupted flash memory faults).

4. Activity #2: Timer 1 in Fast PWM mode to control a Servo motor

- Download "***mcu3a.cpp***" from Canvas.
- Configure Timer1 in a Fast PWM mode (Mode 14) and program that it can generate a PWM signal with 50Hz frequency with a duty cycle varying from 1ms to 2ms. This is required to drive the servo motor (SG90).
- Refer to the datasheet and Appendix on how to computer values for the ICR1 register (Top for the frequency) and OCR1A register (for the duty cycle).
- In your code, use a for-loop to change the OCR1A value to swipe the servo from 0 degrees (or minimum angle) to 180 degrees (or maximum angle) and then from 180 to 0 degrees.
- Show the results to tutors to get marked:

Output	**Marking**
Servo swipes from 0 deg (or minimum) to 180 deg (or maximum) using a for-loop	/1 mark

5. Activity #3: Servo control using a potentiometer

- Download "***mcu3b.cpp***" from Canvas and upload to the UNO board. The LED connected to PB4 will blink at 1 Hz.
- Modify the code so that the potentiometer output can control the servo's angular position. Find the OCR1 values in the table below and find the linear relation.

POT output	ADC reading	Duty cycle	OCR1 value
0V	0	0.5ms (Servo 0 deg)	
5V	1023	2ms (Servo 180 deg)	

- (Extension) Drive the servo without using the Timer1's PWM mode but using Timer 0 and a DIO pin. This is called a software PWM, and you can connect many servos to DIO pins.

- Show the results to tutors to get marked:

Output	**Marking**
The potentiometer controls the servo position from 0 deg (or minimum) to 180 deg (or maximum) while LED (PB4) blinks at 1 Hz	/1 mark
Extension) The servo is controlled without using the PWM output but generating the signal from a digital output	/0.5 marks

Appendix: Setting up Timer1 for Fast PWM Mode

Timer1 is a 16-bit counter (0 – 65535) in ATmega328P, which can be used for handling slow events or servo control with high-resolution. To generate 50Hz (period is 20ms) PWM, we can use the fast-PWM mode (#14) of the Timer 1 as shown below:

Table 16-4. Waveform Generation Mode Bit Description[1] (Continued)

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

In this mode, the TOP value of the timer can be set from ICR1 (Input Capture Register1 – note that Timer0 doesn't have this register) which can control the frequency of the PWM signal. The duty cycle is controlled by OCR1A (Output Compare Register 1A) as illustrated in Figure 2.

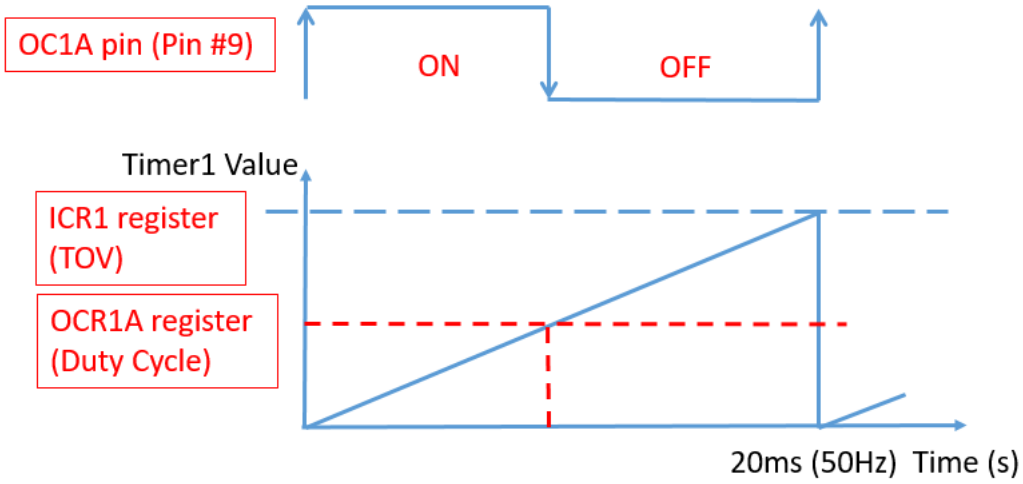

Figure 2. The Fast PWM mode (#14) of Timer1, in which the TOP value is controlled by ICR1 register, and the pulse width is controlled by OCR1A register. The PWM signal is generated through the OC1A pin (Pin#9).

To have 50Hz PWM frequency, first, we need to choose the timer pre-scaler (or clock divider). From the datasheet (pg. 134), the fast PWM mode frequency becomes

$$f_{OCnxPWM} = \frac{f_{clk\_I/O}}{N \cdot (1 + TOP)}$$

This equation tells you the relationship between the system clock frequency ($f_{clk\_I/O}$), pre-scaler (N = 1, 8, 64, 256, or 1024), ICR1 (TOP) and the output PWM frequency ($f_{OCnxPFCPWM}$). To get 50 Hz with a system clock frequency of 16 MHz, you would need to use the following TOP (ICR1) values for the following pre-scalers:

Prescaler N	TOP (ICR1)	Comment
1	320000-1	> 65535 (too big for ICR1)
8	40000-1	Okay
64	5000-1	Okay
256	1250-1	Okay
1024	311.5-1	Too small and fractional

As the ICR1 is a 16-bit register, the maximum raw value is 65535. Thus the Prescaler-1 cannot fit into ICR1. Also, Prescaler-1204 results in a too small and fractional timer frequency, and thus not ideal. We can use the prescalers-8/64/256 depending on the applications. I would suggest pre-scaler 8 and set ICR1 to 40000-1 for maximum resolution, which will be useful to control a servo motor.

As the frequency is set to 50Hz in ICR1 register, the duty cycle is controlled by changing the OCR1A register value. It would be convenient to convert the percentage duty cycle to the OCR1A value. For example, a 50% duty cycle will be converted to OCR1A = 20000-1 as shown in Figure 3 (note that the OCR1A value starts from 0. Thus minus 1 appears).

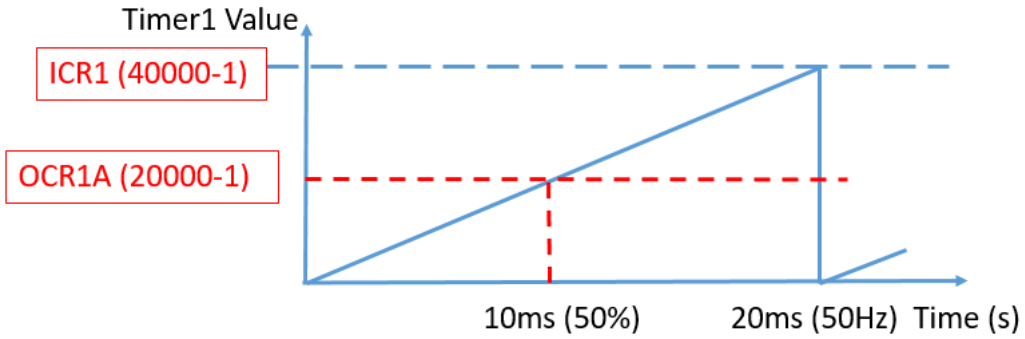

Figure 3. ICR1-TOV value for 20ms period, and OCR1A value for 50% duty cycle.