9.1 Introduction

In one cycle period of a digital signal, there is an ON-period and an OFF-period (Fig-9.1); where, the ON-period is known as 'duty cycle' or 'Pulse Width' or 'Mark' and the OFF-period is known as 'Space'. In Fig-9.1a, we have a 10 kHz signal of which the duty cycle is 50%. When the ON-period of the signal changes, we get a signal what is known as 'Pulse Width Modulated (PWM)' signal (Fig-9.1b). The PWM signal is widely used to regulate the speed of a servo motor or to position the shaft of a stepper motor.





Figure-9.1a: PWM signal with 50 µs pulse width

Figure-9.1b: PWM signal with 68 µs pulse width

9.2 Pre-defined PWM Signals in Arduino UNO



Figure-9.2: Pre-defined PWM signals of Arduino UNO

(1) In Arduino UNO, the execution of this instruction: analogRead(3, dutyCycle) automatically delivers 490 Hz PWM signal at DPin-3 (Fig-2). The duty cycle (ON-period) of the signal can be varied from 0% to 100% by changing the value of arg2 (dutyCycle, an 8-bit variable) from 0x00 to 0xFF.

```
analogWrite(DPin-n, arg2); //arg1 = DPin-n = 3, 11, 10, 9, 5, 6; arg2 is an 8-bit value

⇒ analogWrite(3, 0x45); //duty cycle = 2040/256 * 0x45 ≈ 612 µs (30%)

void setup()

{

analogWrite(3, 0x45);

}

void loop()

{

}
```

If we connect an oscilloscope at DPin-3 of the UNO, we will see a 490 Hz square wave signal with about $612 \ \mu s$ ON-period.

(2) As indicated in Fig-2, we can generate four 490 Hz PWM signals at DPin-3, 11, 10, 9 and two 980 PWM signals at DPin-5, 6 by executing the *analogWrite()* function. There is no need of any initialization. The *analogWrite()* automatically assigns TC0 for DPin-5, 6; TC1 for DPin-9, 10; TC2 for DPin-3, 11 for the generation of the respective PWM signals. The DPins are also automatically configured to work as output lines.

(3) The value of arg2 of the *analogWrite(DPin-n, arg2)* function can be assigned directly (a fixed value) or through an external user controlled variable to change/regulate the duty cycle of the PWM signal. In the following example, the duty cycle of the PWM signal of DPin-3 changes as the voltage value of the wiper point of Pot (potentiometer) changes. The affect of the pulse width change can be visualized in the continuous dimming/intensifying of LED1 as the Pot value is changed.



Figure-9.3: 490 Hz PWM signal with varying duty cycle

(a) Connect R1-LED1 circuit and Pot circuit with UNO as per Fig-9.3.

```
(b) Upload the following sketch.
Byte dutyCycle = 0;
int x;
void setup()
{
    analogWrite(3, dutCycle);
}
void loop()
{
    x = analogRead(A0);
    dutyCycle = map(x, 0, 1023, 0, 255);//map() function compresses scale (0,1023)→(0,255)
    analogWrite(3, dutCycle); //dutyCycle value is updated
    delay(1000); //test interval; the affect will appear after 1-sec
}
```

- (c) Slowly rotate Pot and observe that the intensity of LED1 changes which indicates that duty cycle of the PWM signal is changing.
- (4) A 50 Hz PWM signal can also be generated using **Servo.h** Library. This signal is useful to position the shaft of stepper motor like T_{OWER} P_{PRO} SG90 (Fig-9.4). The PWM signal becomes available at any valid DPin when the following codes are uploaded in the UNO.



Figure-9.4: Tower Pro SG90 stepper motor

#include <Servo.h>
Servo myservo; // create servo object to control a servo

```
void setup()
    {
      Myservo.attach(2);
                            //PWM signal at Din-2
      myservo.write(0x45); //at DPin-2, there is 50 Hz PWM signal with ON-period 20/255*69 = 5 ms
    }
    void loop()
    {
    }
(a) Remove R1-LED1 circuit from DPin-2 of Fig-9.3,
(b) Connect servo motor of Fig-9.4 with at DPin-2 of the UNO of Fig-9.3.
(c) Upload the following sketch:
    #include <Servo.h>
    Servo myservo; // create servo object to control a servo
    int potpin = A0;
                        // analog pin used to connect the potentiometer
    byte val;
                        // variable to read the value from the analog pin
    void setup()
    {
      myservo.attach(3); // attaches the servo on pin 9 to the servo object
    }
    void loop()
    {
      val = analogRead(potpin); // reads value of Pot (value between 0 and 1023)
      val = map(val, 0, 1023, 0, 180); // scale it to use with servo (value between 0 and 180)
                                        // sets the servo position according to the scaled value
      myservo.write(val);
      delay(15);
                                       // waits for the servo to get there
    3
```

(d) Slowly turn Pot and check that the shaft position of the servo motor changes accordingly.

9.3 Dual Slope Phase Correct (Mode-10) PWM Signal using TC1







Figure-9.6: PWM Hardware of TC1 Module

(1) In Fig-9.5, we observe that there is a 'periodic digital signal' at DPin-9 of the Arduino UNO. This is a PWM signal as its duty cycle (pulse width) can be changed. The period of the PWM signal is confined within the rising and falling slopes of the reference signal ACE and hence the PWM signal is known as 'dual slope' signal. Why is it called 'phase correct'?

(2) Frequency Determination: The

(a) Frequency of the PWM signal is the inverse of the period of the signal. The period (as we can see in Fig-9.5) is equal to the time required for the TCNT1 Register to 'count up from point-A (initial count = 0) to point-C (count = content of ICR1 Register)' + 'count down from point-C (initial count = content of ICR1 Register) to point-E (final count = 0)'. Based on this principle, let us find the equations for the frequency and the duty cycle of the PWM signal as a function of clkTC1 (driving clock of TC1, Fig-9.6), N (prescale divider), ICR1 (content of ICR1 Register = TOP value), and OCR1A (content of OCR1A Register).

(b) With initial value of 0x0000, the TCNT1 Register begins counting up from point-A to point-C along the rising slope at clocking speed of clkTC1. When the content of TCNT1 Register becomes equal to the content of OCR1A Register at point-B, the logic level of OC1A line at DPin-9 assumes LOW state and remains LOW until point-D arrives. TCNT1 continues counting up from Point-B until its content becomes equal to the content of ICR1 Register at point-C.

(c) Now, the TCNT1 starts counting down from point-C towards point-E along the falling slope at the same clocking speed of clkTC1. When the content of TCNT1 Register becomes equal to the content of OCR1A Register at point-D, the logic level of OC1A line assumes HIGH state and remains HIGH until point-F arrives. We have got one period of the PWM Wave over the points: B, D, and F (or can be said as: over the points: A, C, and E); where, BD is OFF-period and DF is the ON-period/dutyCycle. As the PWM signal involves 2 slopes, it is called dual-slope PWM signal. It is also known as 'phase correct' PWM signal. Why is it called phase correct PWM signal?

(d) Now:

Period (T) of the PWM signal = Time-AC + Time-CE; where, Time-AC = Time (for TCNT1) to count up from point-A to point-C = ICR1*1/clkTC1 Time-CE = Time (for TCNT1) to count down from point-C to point-E = ICR1*1/clkTc1

 $\begin{array}{l} \mbox{Period (T) = Time-AC + Time-CE = ICR1*1/clkTC1 + ICR1*1/clkTC1 = 2*ICR1*1/clkTC1 \\ f_{OC11APCPWM} (frequency of the phase correct PWM signal) = 1/Period = clkcTC1/2*ICR1 \\ f_{OC11APCPWM} = (clkSYS/N)/(2*ICR1) = clckSYS/(2*N*ICR1) = clckSYS/(2*N*TOP) \\ \end{array}$

TOP has been defined as the highest value in the counting sequence. **BOTTOM** has been defined as the lowest value (0x0000) in the counting sequence.

The frequency of the PWM signal can be changed by changing the content of ICR1 Register. The duty cycle can be changed by changing the content of OCR1A Register and OCR1B Register.

(3) **Duty Cycle Determination:** From the above discussion and from Fig-9.5, it appears that the duty cycle is NIL (0%) when the content of OCR1A Register is 0x0000 and the duty cycle id FULL (100%) when the content of OCR1A Register is equal to the content of ICR1 Register (the TOP value). For a given frequency of the PWM signal, the duty cycle can be varied by changing the content of OCR1A Register.

(4) Flags Activation:

(a) At point-B/D of Fig-9.5, compare match occurs between TCNT1 and OCR1 Registers when their contents are equal; as a result, the 'Output Compare A Match Flag (OCF1A)' of TIFR1 Register becomes HIGH; this flag can be used to interrupt the MCU after putting 1s into the OCIE1A bit of TIMSK1 and I bit of SREG Registers. After interruption, the MCU vectors at 0x0016 from where it jumps at the following handler to accomplish interrupt services. The OCF1A flag is automatically cleared when the MCU vectors at the ISR; else, it can cleared by writing LH (*bitSet(TIFR1, 1)*) at this bit position.

```
ISR(TIMER1_CMPA_vect)
{
```

}

(b) At point-C of Fig-9.5, compare match occurs between TCNT1 and ICR1 Registers when their contents are equal; as a result, the 'Input Capture Flag (ICF1)' of TIFR1 Register becomes HIGH; this flag can be used to interrupt the MCU after putting 1s into the ICIE1A bit of TIMSK1 and I bit of SREG Registers. After interruption, the MCU vectors at 0x0014 from where it jumps at the following handler to accomplish interrupt services. The ICF1 flag is automatically cleared when the MCU vectors at the ISR; else, it can cleared by writing LH (*bitSet(TIFR1, 5)*) at this bit position.

```
ISR(TIMER1_CAPT_vect)
{
```

}

(c) At point-E of Fig-9.5, the content of the TCNT1 Register becomes zero; as a result, the TOV1 flag of the TIFR1 Register becomes HIGH; this flag can be used to interrupt the MCU after putting 1s into the TOIE1 bit of TIMSK1 and I bit of SREG Registers. After interruption, the MCU vectors at 0x001A from where it jumps at the following handler to accomplish interrupt services. The TOV1 flag is automatically cleared when the MCU vectors at the ISR; else, it can cleared by writing LH (*bitSet(TIFR1, 0)*) at this bit position.

```
ISR(TIMER1_OVF_vect)
{
}
```

9.4 Sketch for 20 kHz dual-slope phase correct PWM signal using TC1 in Mode-10

9.5 Exercises

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	тор	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, phase correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, phase correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, phase correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, phase and frequency correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, phase and frequency correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, phase correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, phase correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
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

Waveform Generation Mode Bit Description:



[b](1)[/b] In Fig-1, we observe that we can generate PWM signals of desired frequencies and widths at Dpin: ~6, ~5 of Arduino UNO through the programming of TCNT0 (TC0 Module) of the MCU. In this case, we have to write our own codes with strict reference to the data sheets for the bit values of various PWM related registers.

[b](2)[/b] Similarly, we can generate PWM signals of desired frequencies and widths at Dpin: ~9, ~10 of Arduino UNO through the programming of TCNT1 (TC1 Module) of the MCU.

[b](3)[/b] Similarly, we can generate PWM signals of desired frequencies and widths at Dpin: ~11, ~3 of Arduino UNO through the programming of TCNT2 (TC2 Module) of the MCU.

[img]https://forum.arduino.cc/index.php?action=dlattach;topic=568966.0;attach=274401[/img] Figure-1: Generation of PWM signals by programming of the TCX modules of ATmega328 MCU

[b]B:[/b] Generation of PWM signals (fixed frequency and variable width) at DPins: ~6, ~5; ~9, ~10; ~11, ~3 using Arduino commands

[b](1)[/b] In Fig-2, we observe that we can create PWM signals of about 1000 Hz frequency at Dpin: ~6, ~5 of Arduino UNO by executing these Arduino instructions: [b]analogWrite(6, pulseWidth);[/b] and [b]analogWrite(5, pulseWidth);[/b]. The width of the PWM signal can be dynamically varied by changing the value of the 8-bit valued 2nd argument (pulseWidth) of the instruction; the argument can assume direct values from 0x00 to 0xFF or from an analog channel after mapping.

[b](2)[/b] Similarly, we can create PWM signals of about 500 Hz frequency at Dpin: ~9, ~10 of Arduino UNO by executing these Arduino instructions: [b]analogWrite(9, pulseWidth);[/b] and [b]analogWrite(10, pulseWidth);[/b].

[b](3)[/b] Similarly, we can create PWM signals of about 500 Hz frequency at Dpin: ~11, ~3 of Arduino UNO by executing these Arduino instructions: [b]analogWrite(11, pulseWidth);[/b] and [b]analogWrite(3, pulseWidth);[/b].

[img]https://forum.arduino.cc/index.php?action=dlattach;topic=568966.0;attach=274407[/img] Figure-2: PWM signals of known frequencies using Arduino commands

[b]C:[/b] Generation of PWM Signal at any permissible DPin of UNO of known frequency (50 Hz) and variable width using [b]Servo.h[/b] Library Functions for stepper servo SG90 and the like [b](1)[/b] To lock the shaft of the SG90 stepper servo motor at a desired position (say, 90[sup]0[/sup] from the reference position), we need to maintain a continuous signal of 50 Hz with 2 ms ON-period and 18 ms OFF-period at the Control Pin of the servo. By varying the ON-period, the shaft position can also be changed. (Continuous injection of the PWM signal at the Control Pin of the servo is maintained by the Servo.h Library through interrupts.)

[b](2)[/b] This signal is automatically created and sustained at DPin-X (X = 0 to 19) of the UNO when the following codes are included in the sketch. [code]#include<Servo.h> Servo myServo; myServo.attach(DPin-X); //DPin-X=0 to 19 with which the Control Pin of the servo is connected. myServo.write(value); //value determines the ON-period of the 50 Hz PWM signal.[/code]

[b]BTW:[/b] Is Servo.h Library using TCX Module of the MCU for the generation of the PWM signal at the Control Pin of the servo? I have no information about it. Probably, it does not utilize the TCX Module; because, DPin-7 has no relation with any of the TCX Modules; but, we can still drive a servo (SG90) via DPin-7?







