UNIVERSITY of GUELPH

Mechatroncis

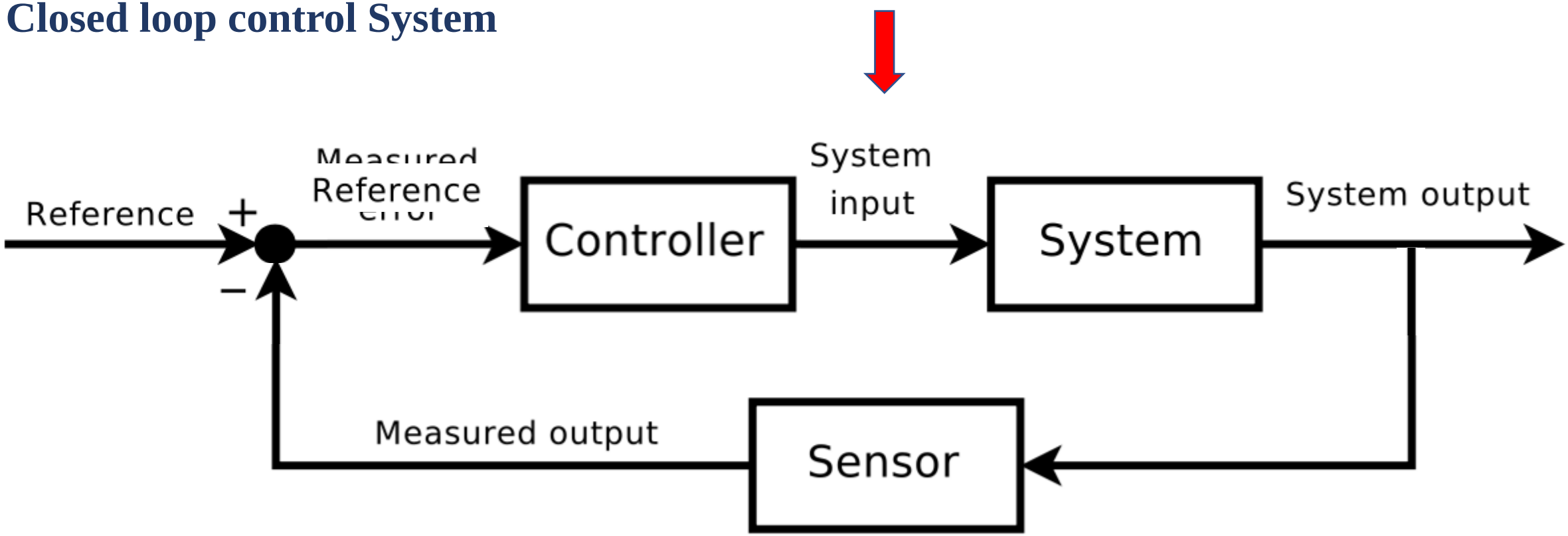
Info Session:

Pulse Width Modulation (PWM)

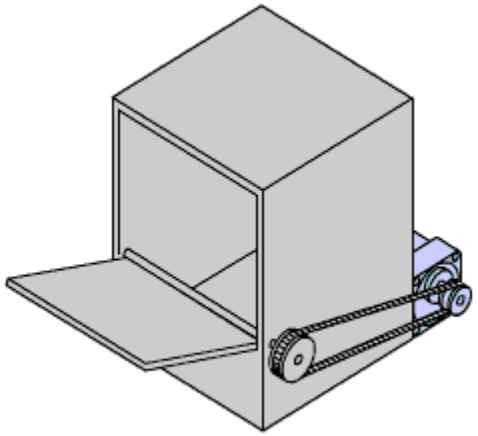
Michael Pumphrey & Dixit Patel

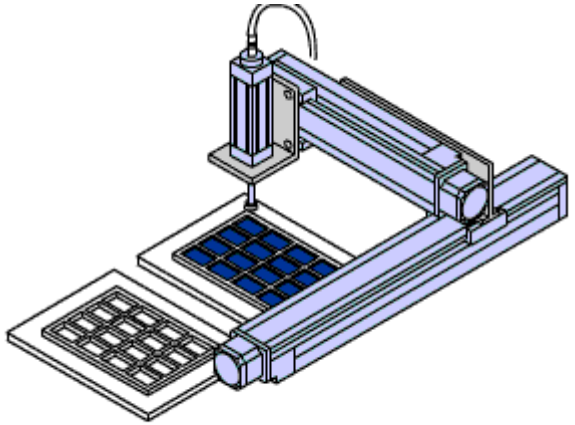
What is PWM? Why is PWM useful?

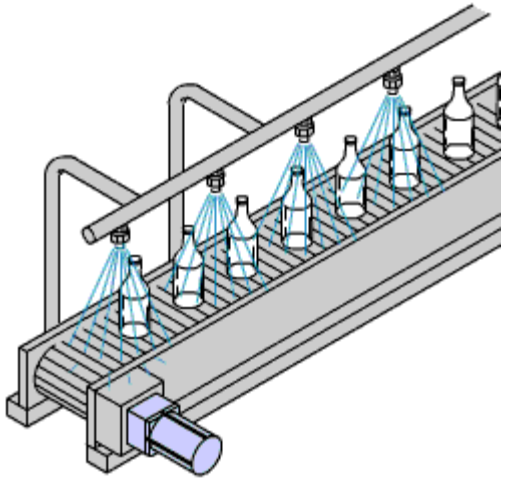
Closed loop control System

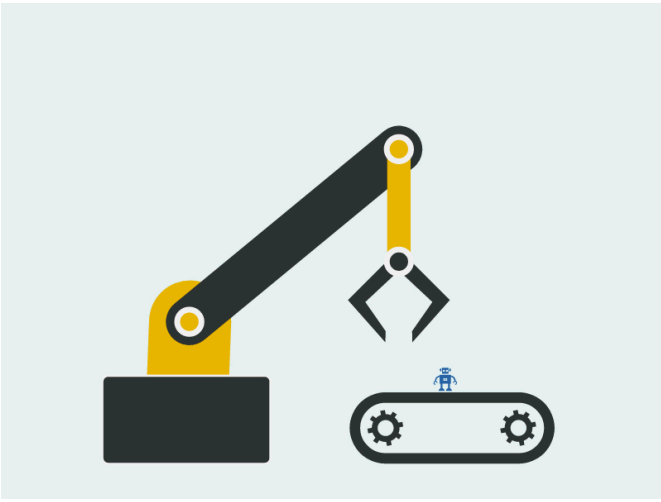


Mechatronics Systems









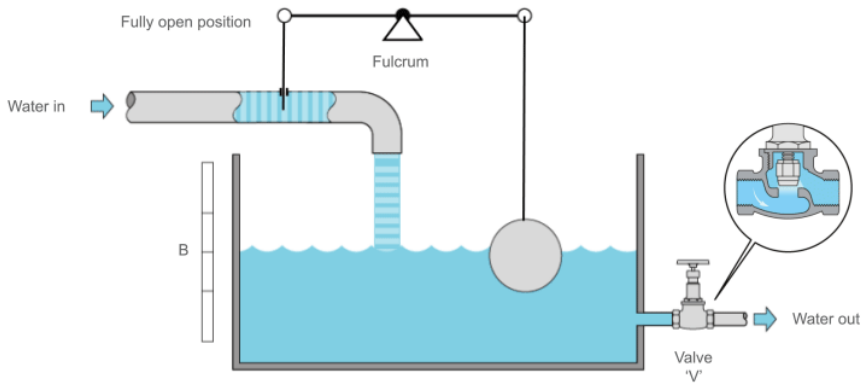
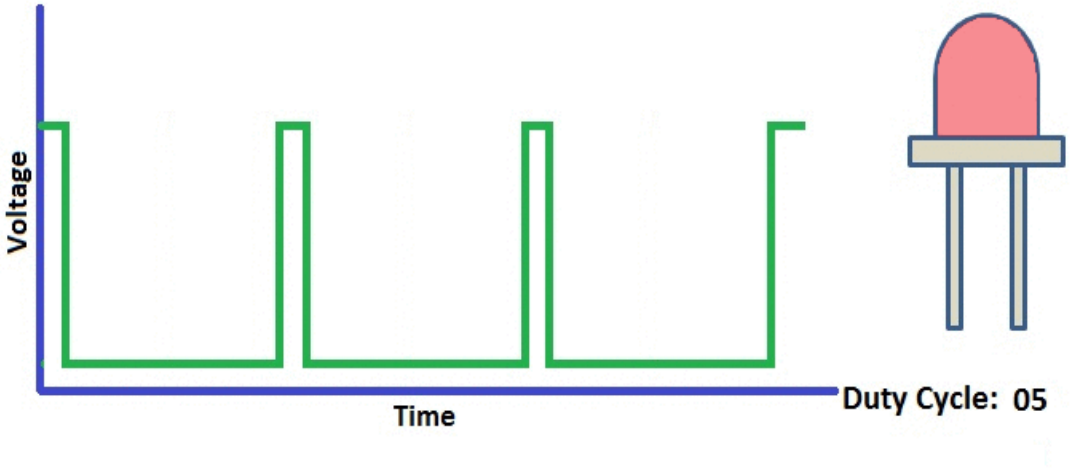
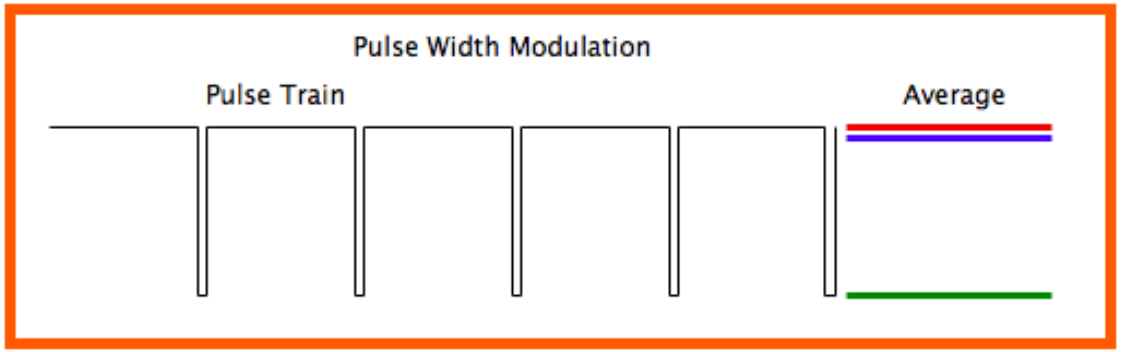


Fig. 5.2.6 Valve open

Pulse Width Modulation

PWM: is a technique which allows us to adjust the average value of the voltage that's going to the electronic device by turning on and off the power at a fast rate.





Pulse Width Modulation

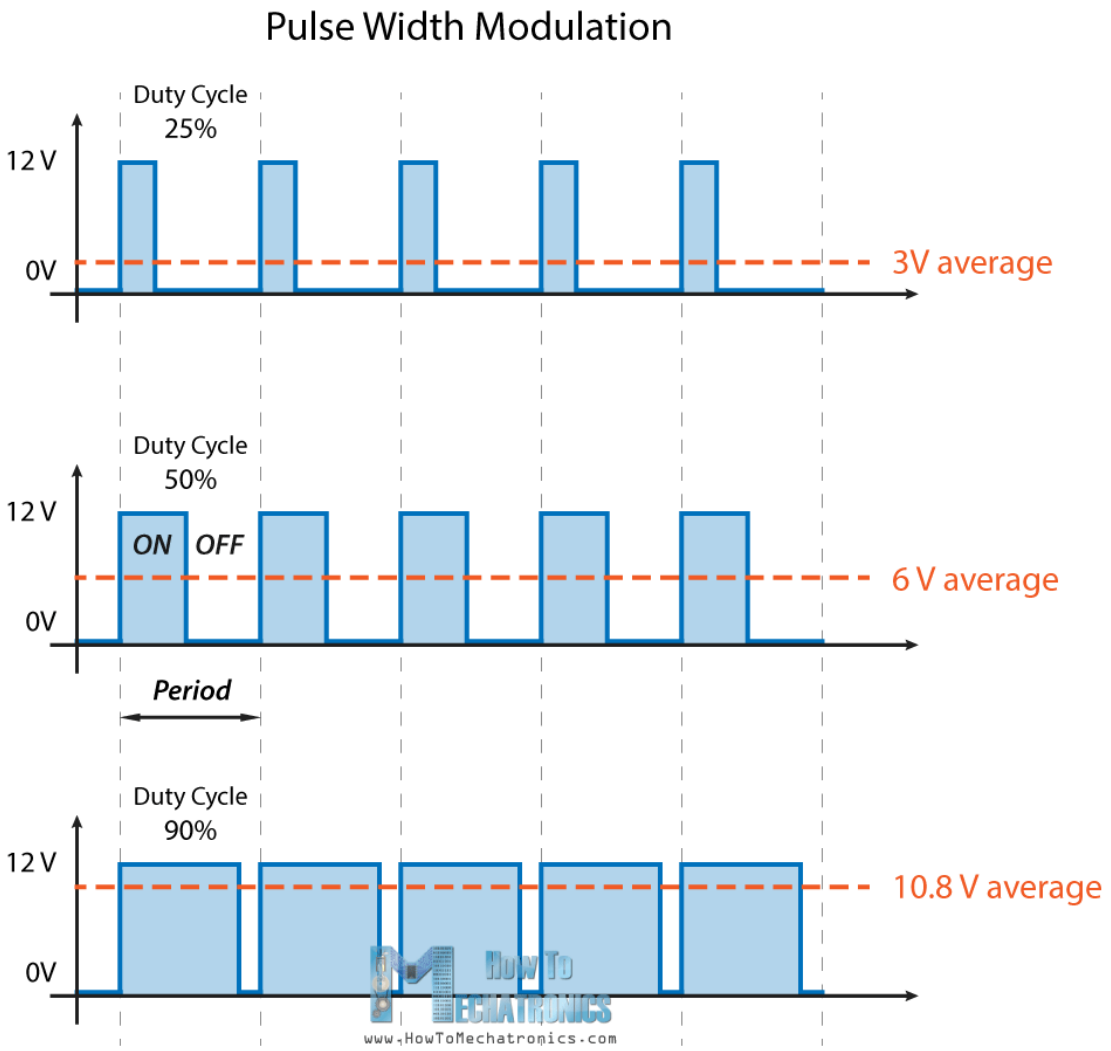
The average voltage depends on the duty cycle, or the amount of time the signal is ON versus the amount of time the signal is OFF in a single period of time.

Duty cycle

$$D = \frac{T_{ON}}{T_{period}}$$

Average voltage

$$V_{AVG} = D\ V_{Hi} + (1 - D)V_{Low}$$



Pulse Width Modulation

Ex:1 Find the average voltage where:

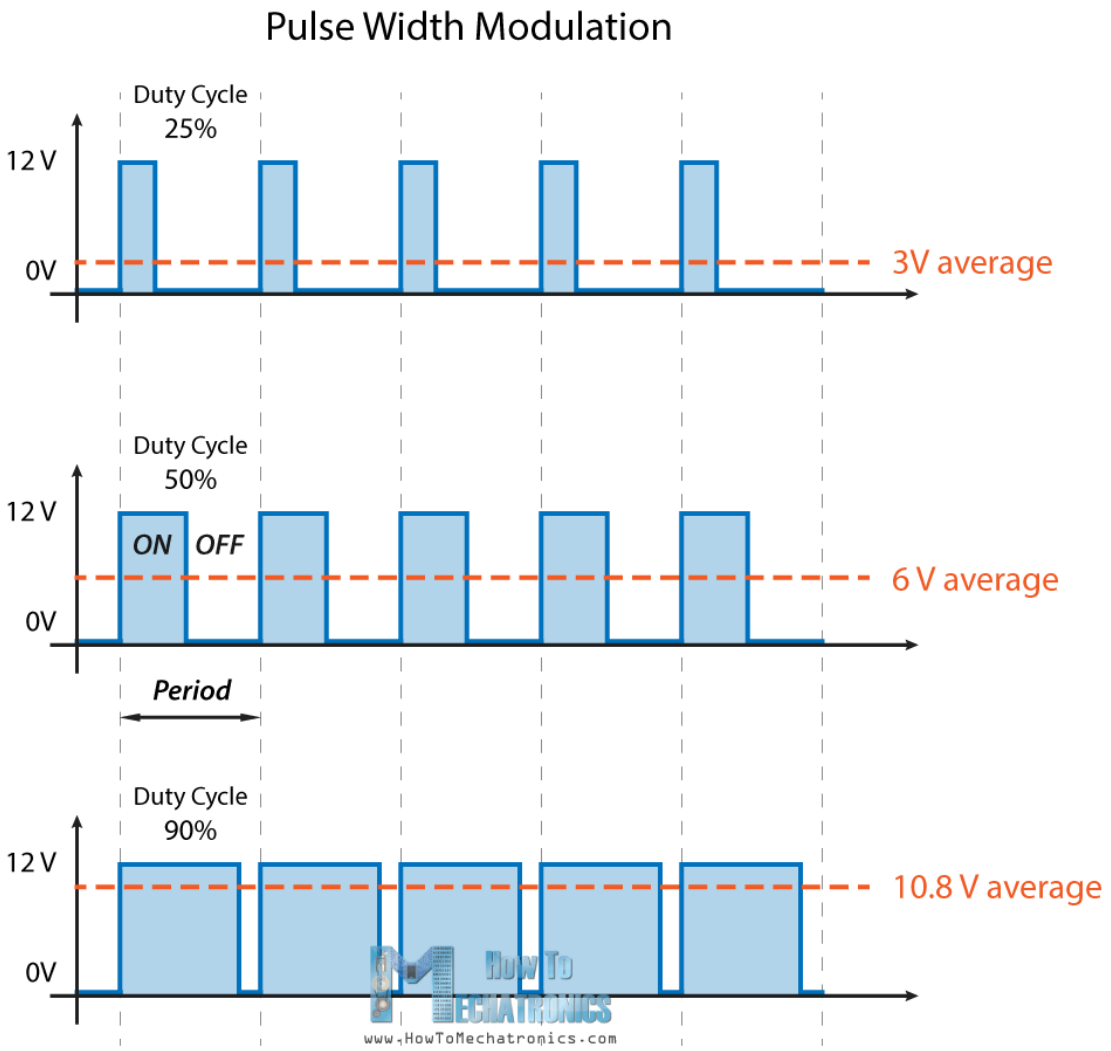
$$D = 25\%$$

$$V_{Hi} = 12V \quad V_{Low} = 0V$$

Solution

$$V_{AVG} = 0.25 * 12 + (1 - 0.25) * 0$$

$$V_{AVG} = 3\ V$$



Pulse Width Modulation

Ex:2 Find the average voltage where:

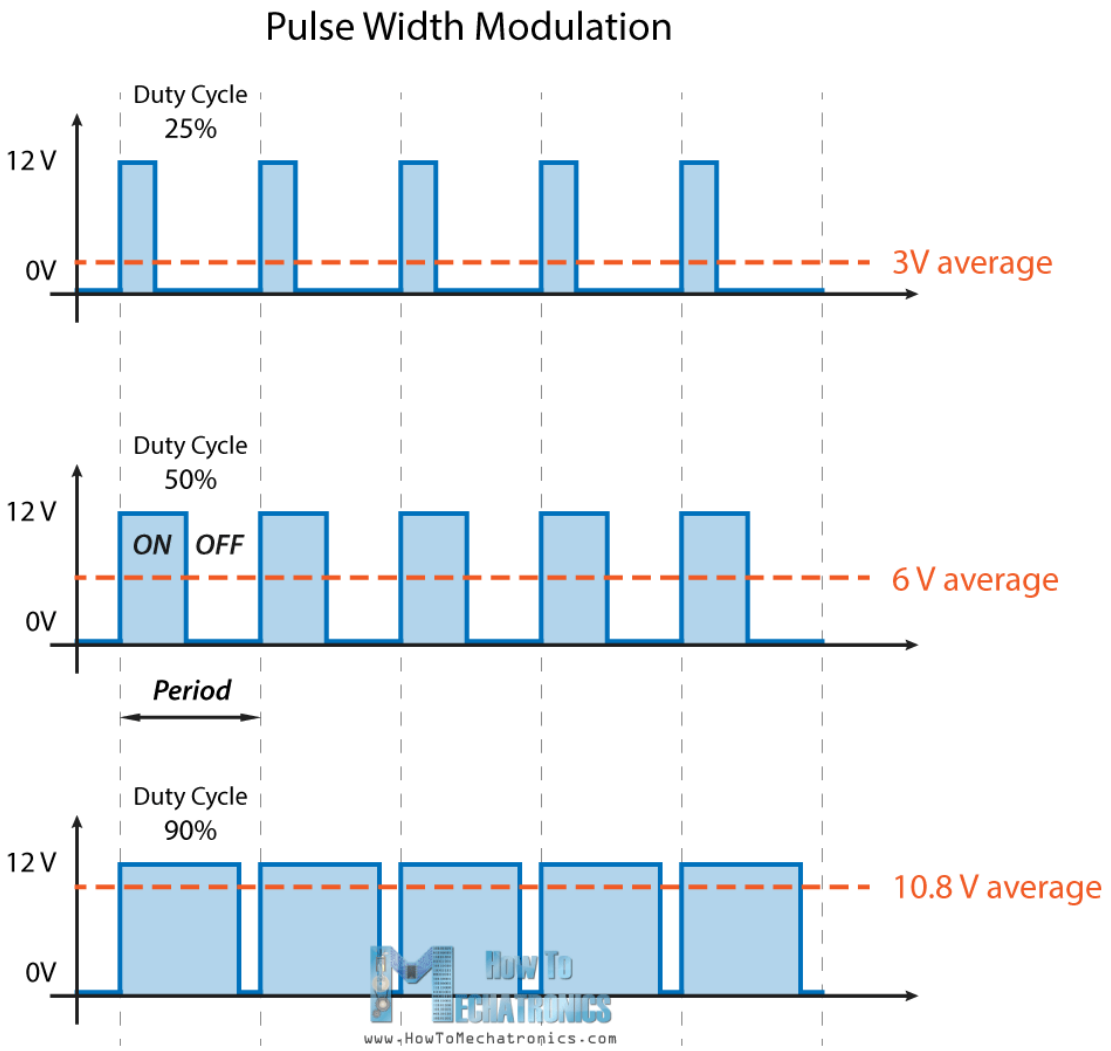
$$D = 90\%$$

$$V_{Hi} = 12V \quad V_{Low} = 0V$$

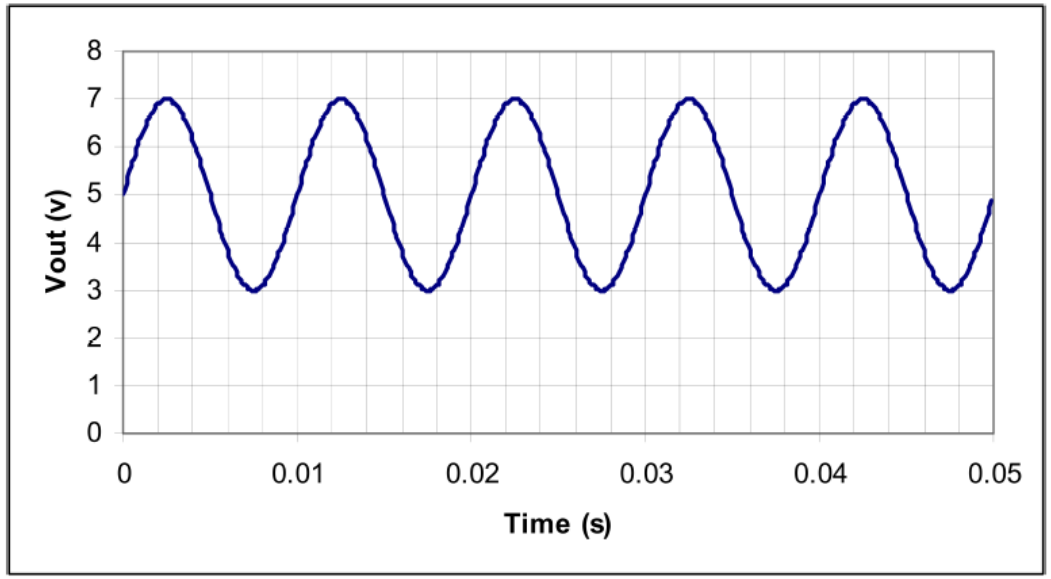
Solution

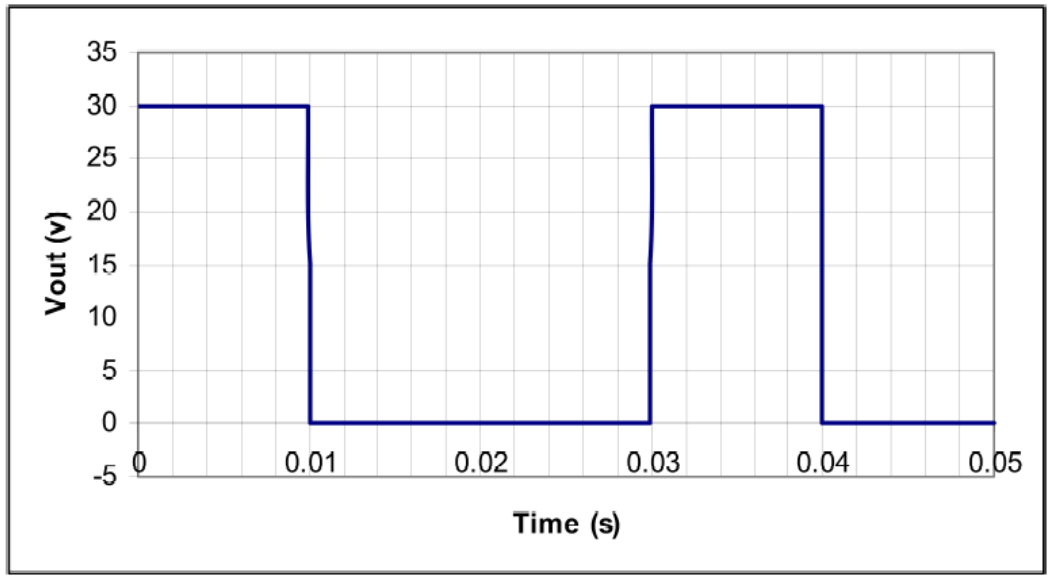
$$V_{AVG} = 0.9 * 12 + (1 - 0.25) * 0$$

$$V_{AVG} = 10.8\,V$$



Pulse Width Modulation

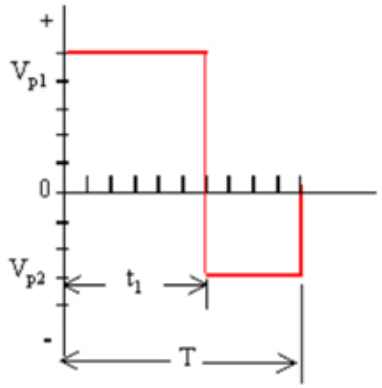




$$V_{avg} = V_{DC} = 5V$$

$$V_{rms} = V_{AC} = \frac{V_p}{\sqrt{2}} = \frac{2}{\sqrt{2}} = 1.414V$$

$$True\ RMS = \sqrt{{V_{AC}}^2 + {V_{DC}}^2} = 5.2V$$



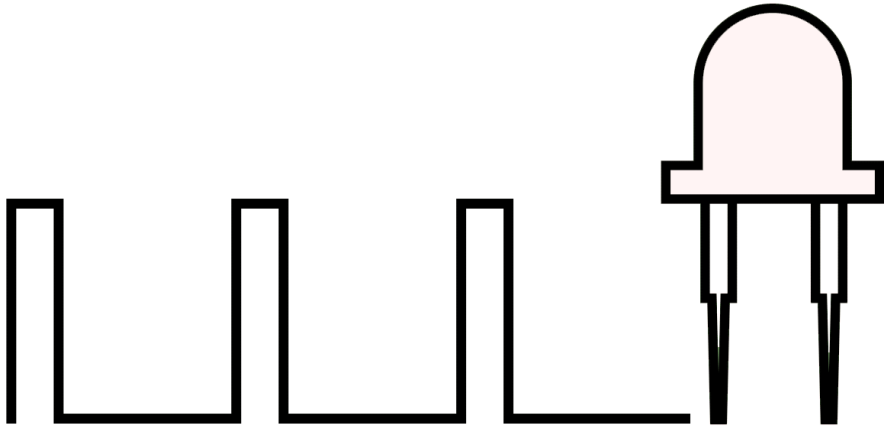
$$V_{avg} = V_{DC} = V_p * D = 30 * 0.333 = 9.99V$$

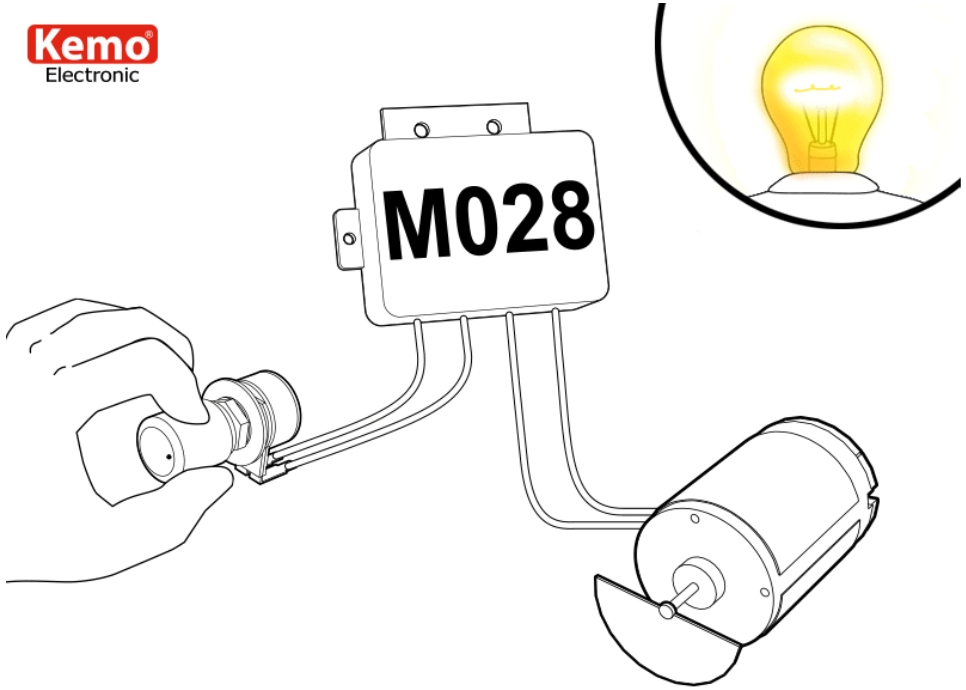
$$V_{rms} = V_{AC} = \sqrt{{V_{rms1}}^2 + {V_{rms2}}^2} = 14.14v$$

$$True\ RMS = \sqrt{{V_{AC}}^2 + {V_{DC}}^2} = V_p * \sqrt{D} = 17.32V$$

Why is PWM useful?

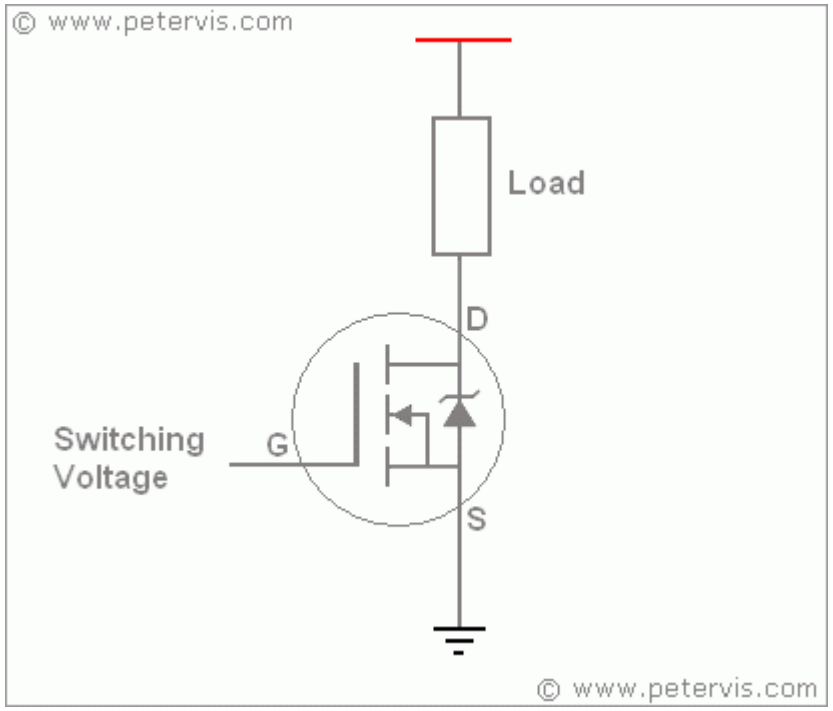
We can use PWM to control the power delivered to an electrical load, e.g. control the speed of the DC motor by simply controlling the input voltage to the motor or the brightness of light bulb or LED.

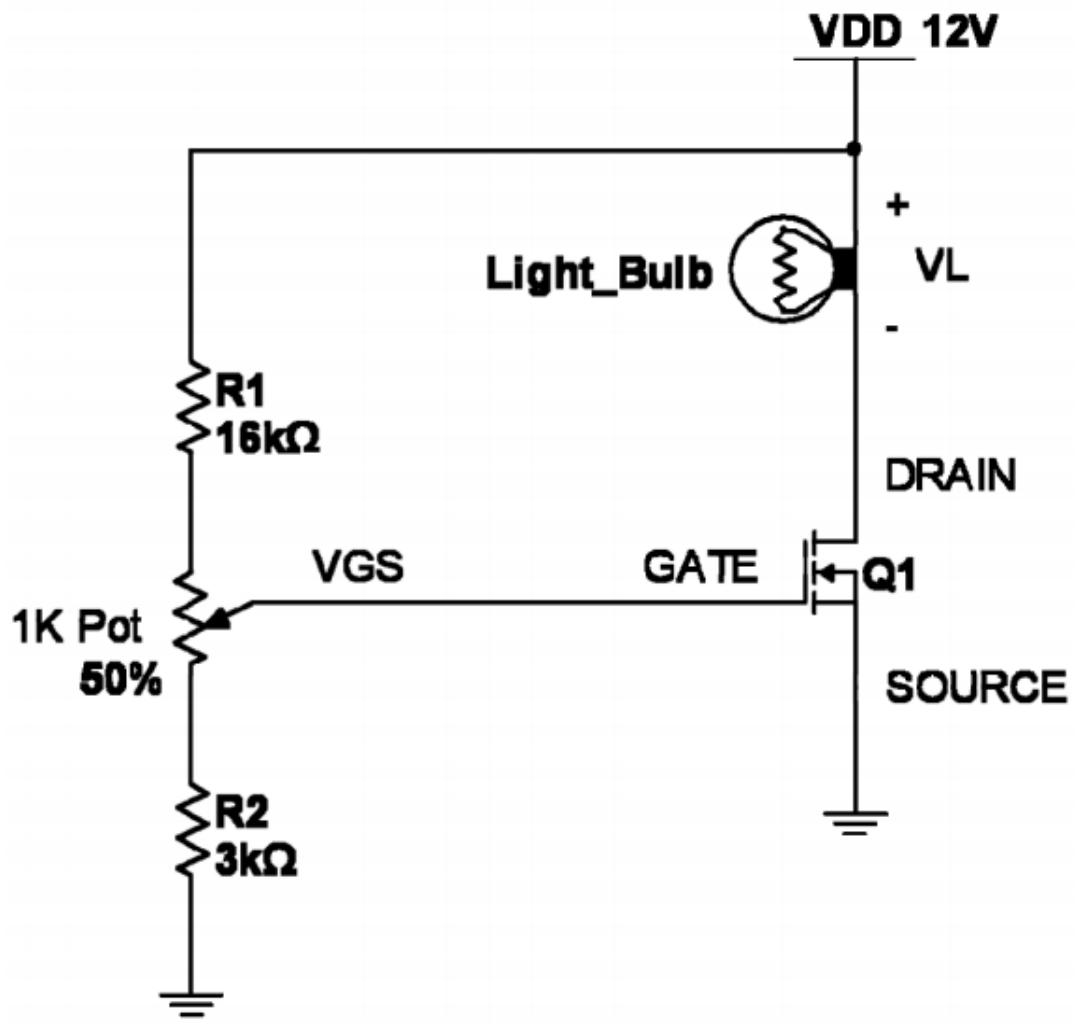




How to generate PWM?

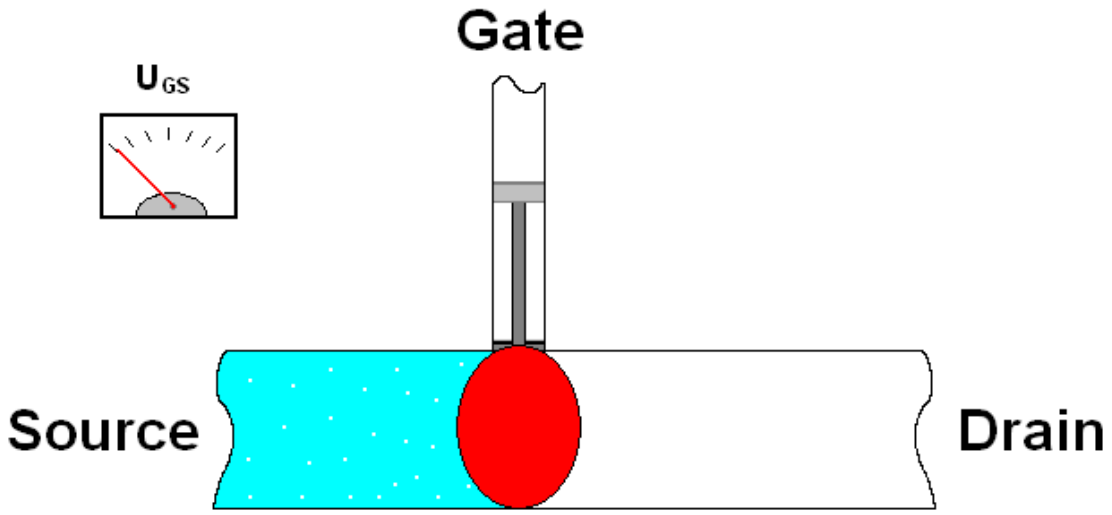
1- Linear Control Strategy (Using MOSFET)

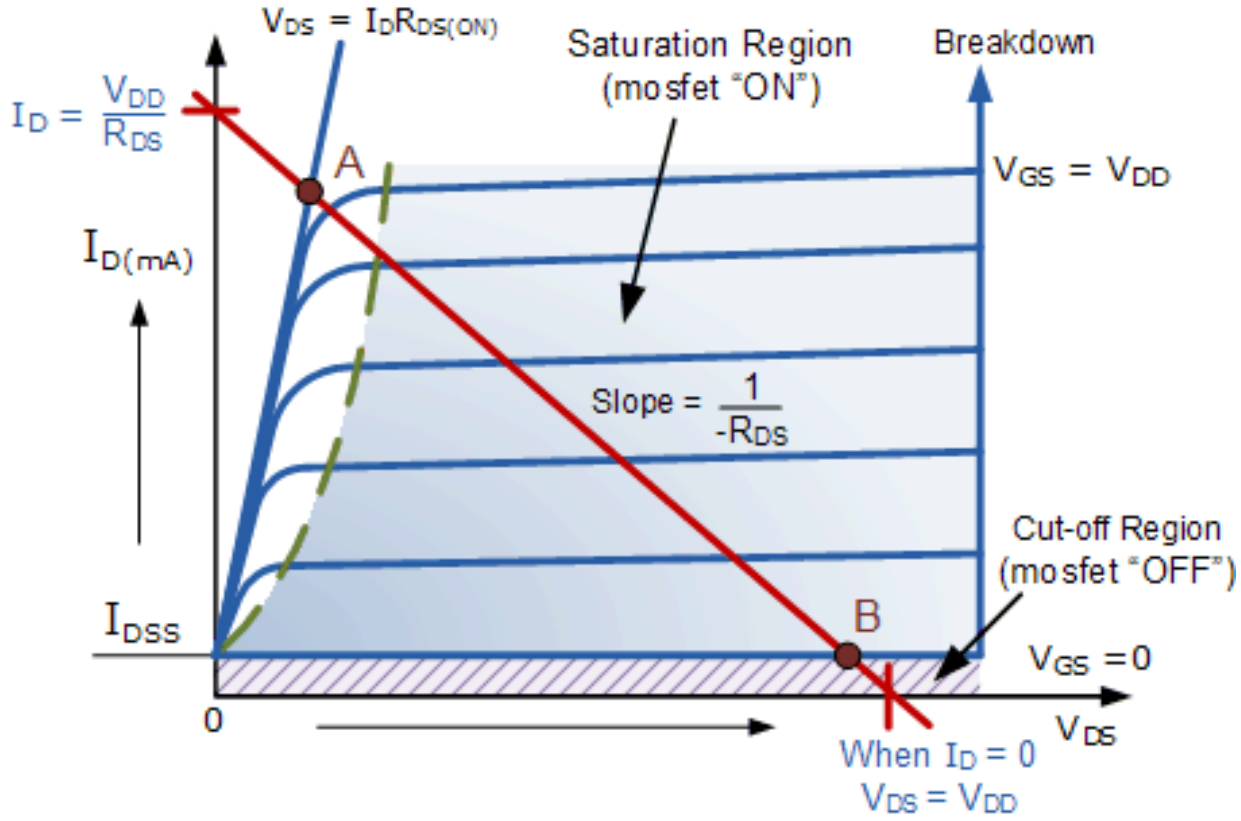




How to generate PWM?

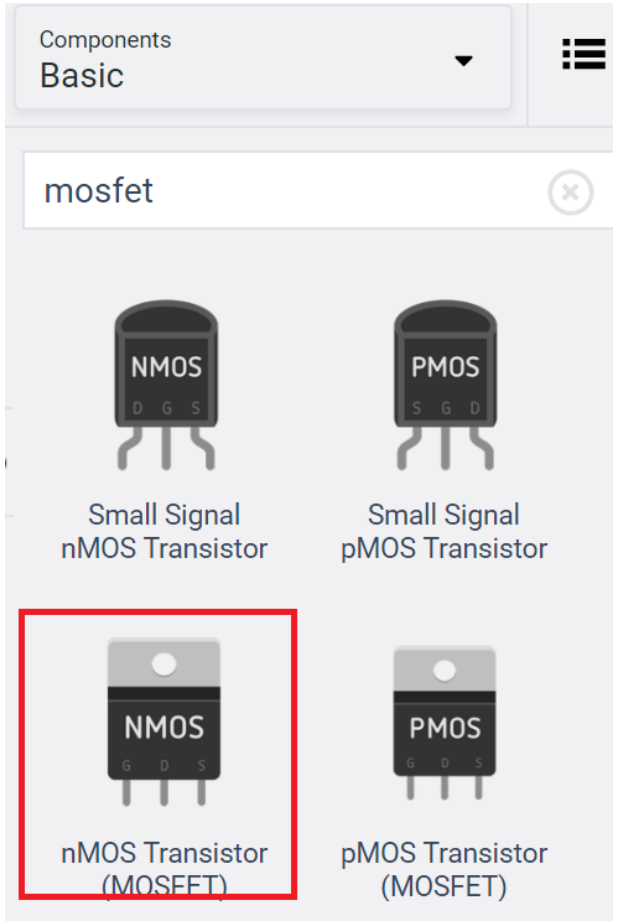
1- Linear Control Strategy (Using MOSFET)

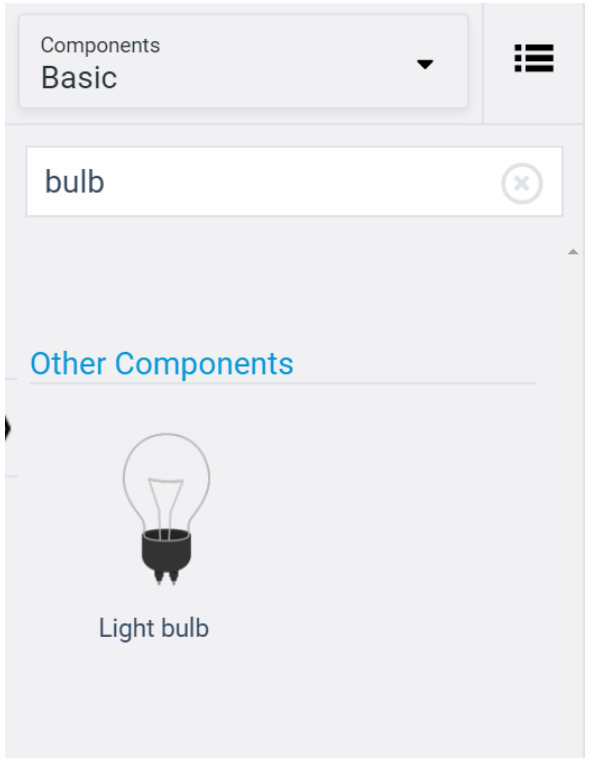


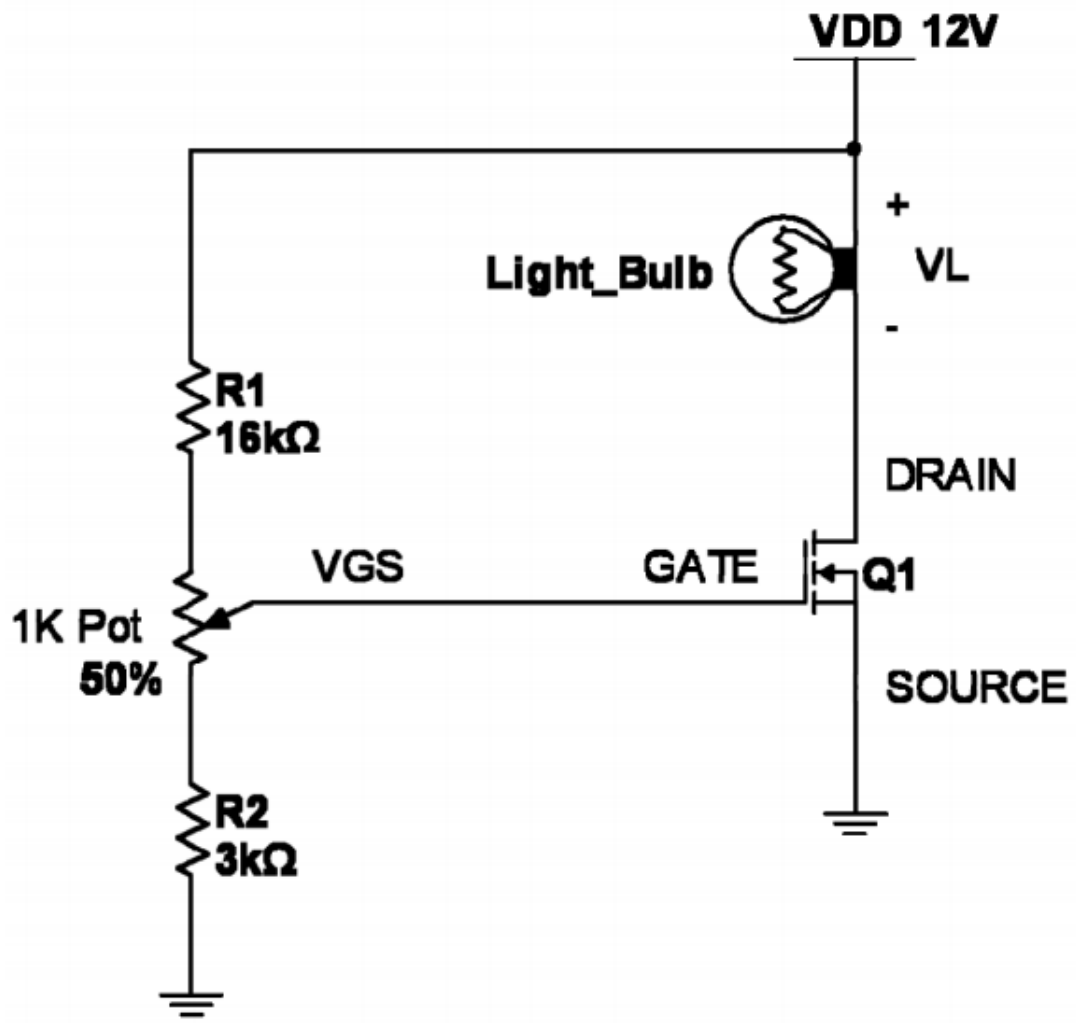


How to generate PWM?

1- Linear Control Strategy (Using MOSFET)

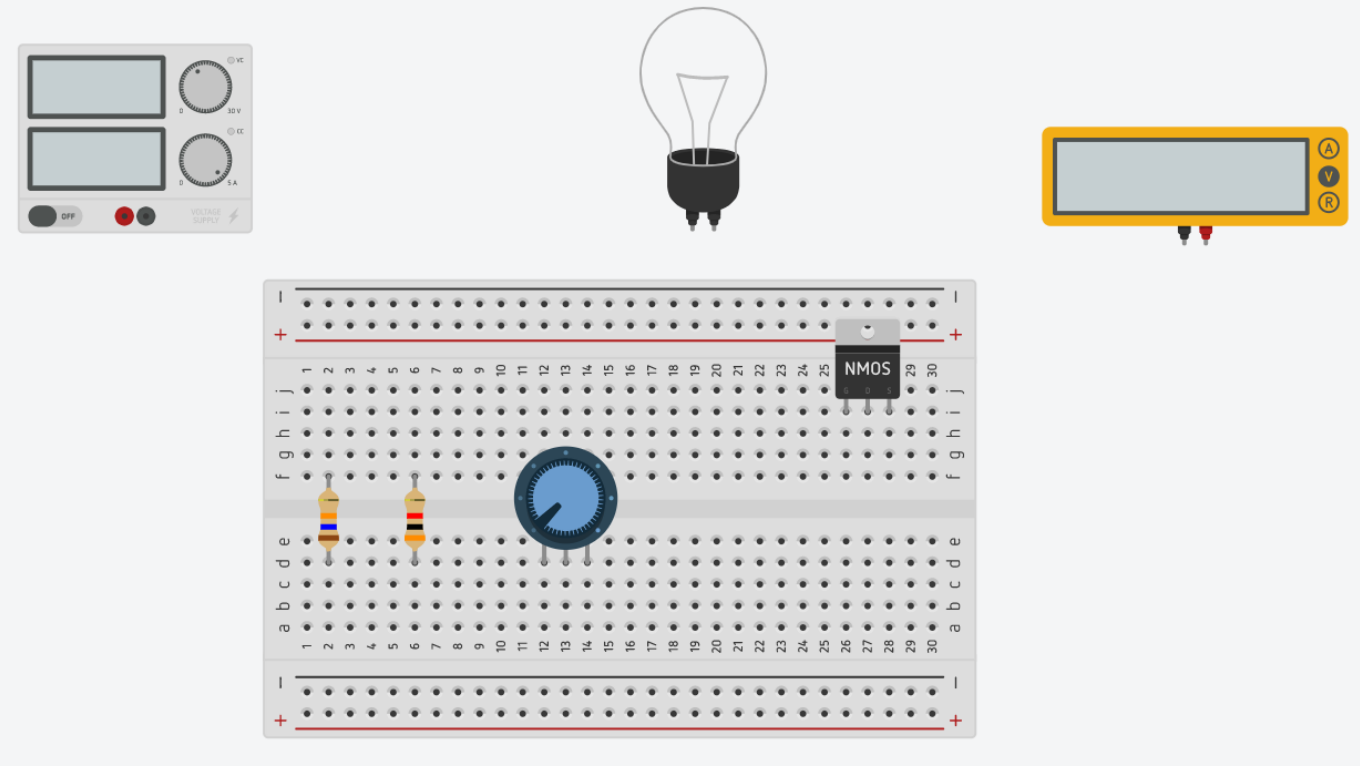


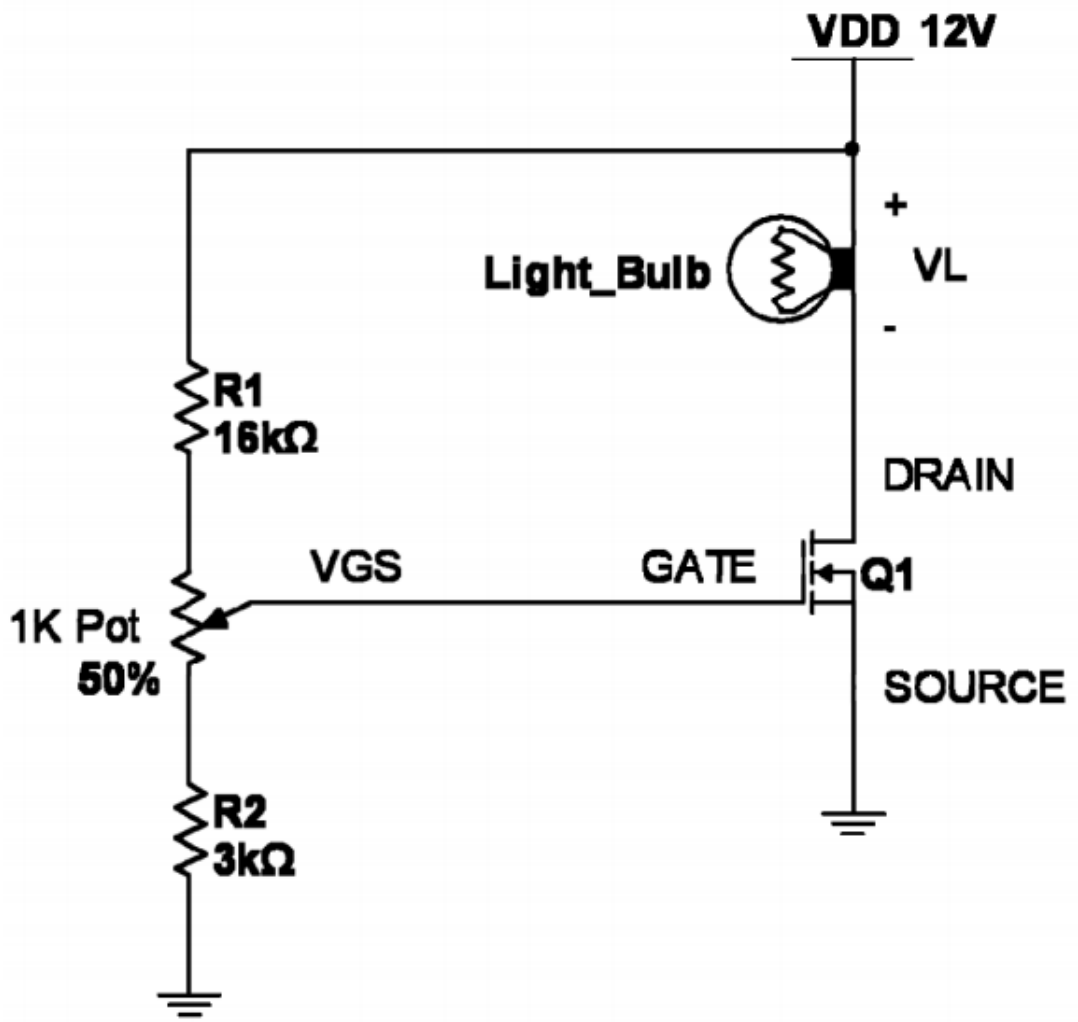




How to generate PWM?

1- Linear Control Strategy (Using MOSFET)



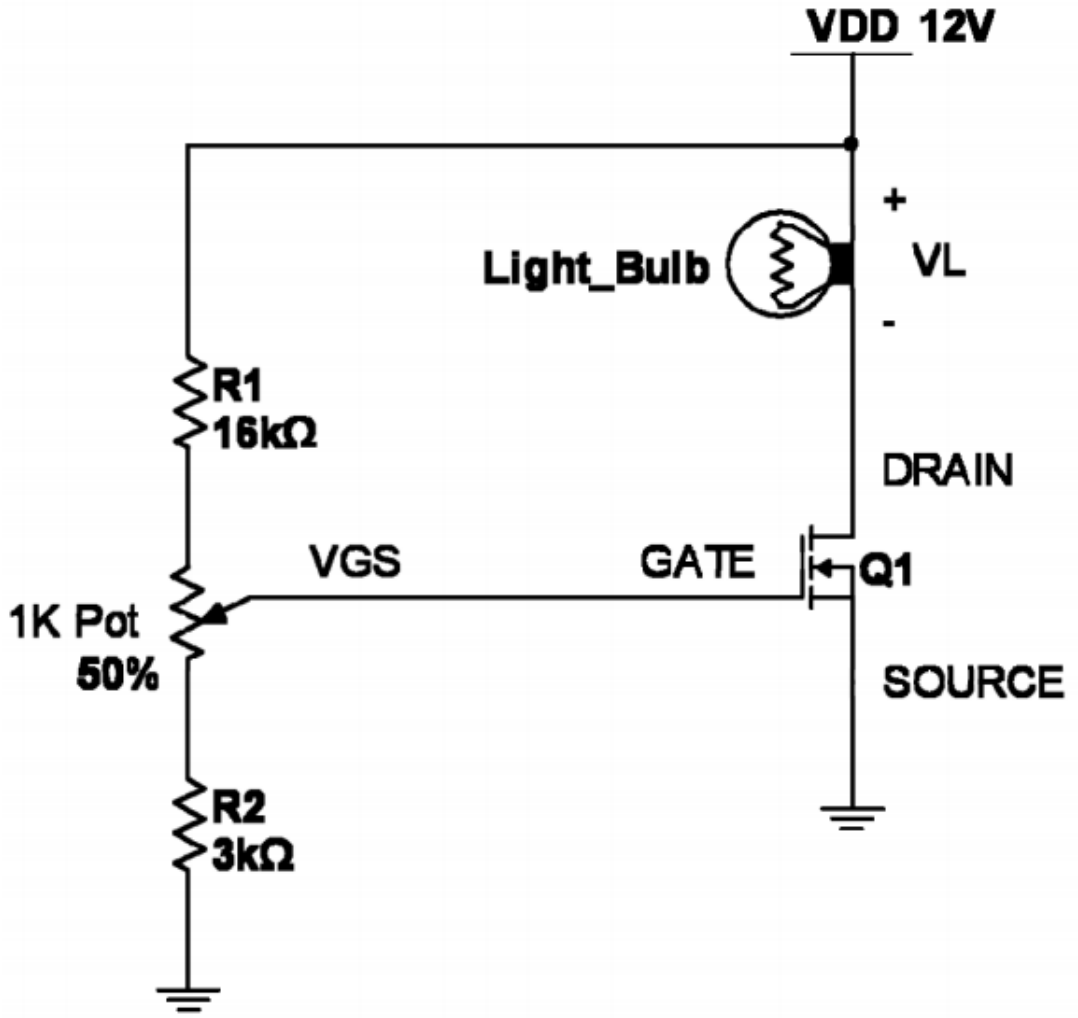


How to generate PWM?

1- Linear Control Strategy (Using MOSFET)

$$\eta = \frac{P_{\mathrm{L}}}{P_{\mathrm{S}}} = \frac{V_{\mathrm{L}} I_{\mathrm{D}}}{V_{\mathrm{DD}} I_{\mathrm{D}}} = \frac{V_{\mathrm{DD}} I_{\mathrm{D}} - V_{\mathrm{DS}} I_{\mathrm{D}}}{V_{\mathrm{DD}} I_{\mathrm{D}}}$$

V_{GS}	I_D	V_L	$P_S = V_{DD} I_D$	$P_L = V_L I_D$	$\eta = \frac{P_L}{P_S}$
...					
...					



How to generate PWM?

2- Digital PWM (using digital controllers, e.g. Arduino)

a) Bit-banging Pulse Width Modulation with digitalWrite.

b) Simple Pulse Width Modulation with analogWrite.

c) Using the ATmega PWM registers directly.

How to generate PWM?

a) Bit-banging Pulse Width Modulation with digitalWrite.

We can ”manually” implement PWM on any pin by repeatedly turning the pin on and off for the desired times.

What is the value of duty cycle?

How much is the frequency?

```
void setup()
{
  pinMode(led, OUTPUT);
}

void loop()
{
  digitalWrite(led, HIGH);
  delay(700);
  digitalWrite(led, LOW);
  delay(300);
}
```

How to generate PWM?

a) Bit-banging Pulse Width Modulation with digitalWrite.

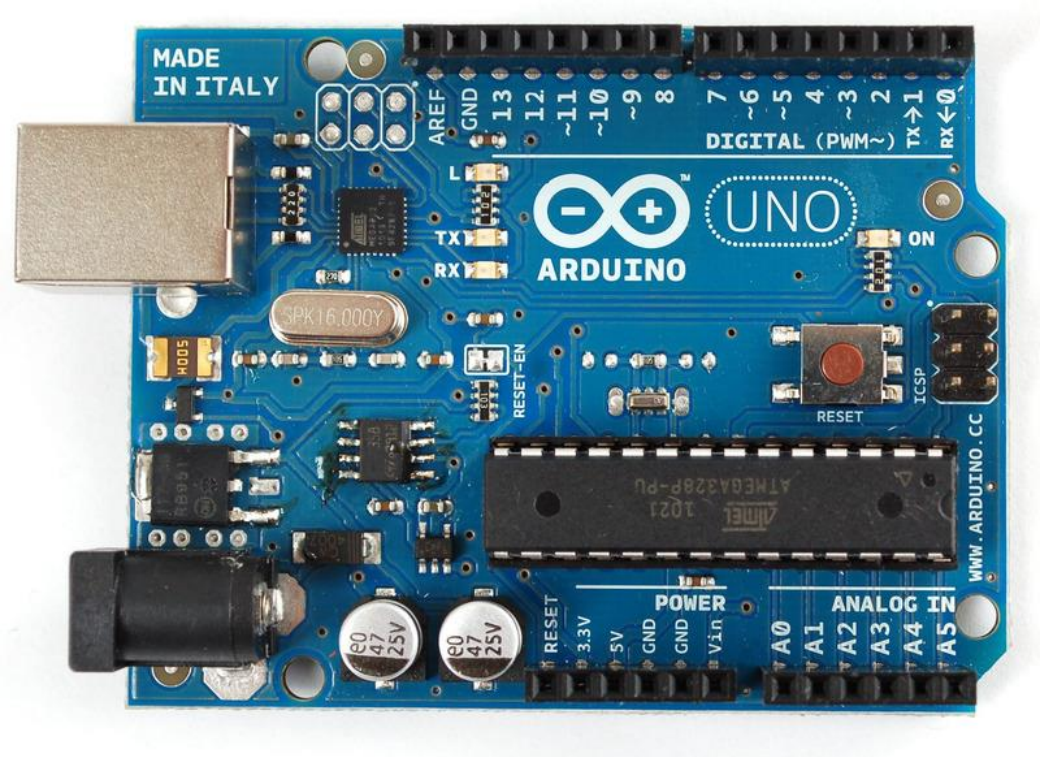
Advantage:

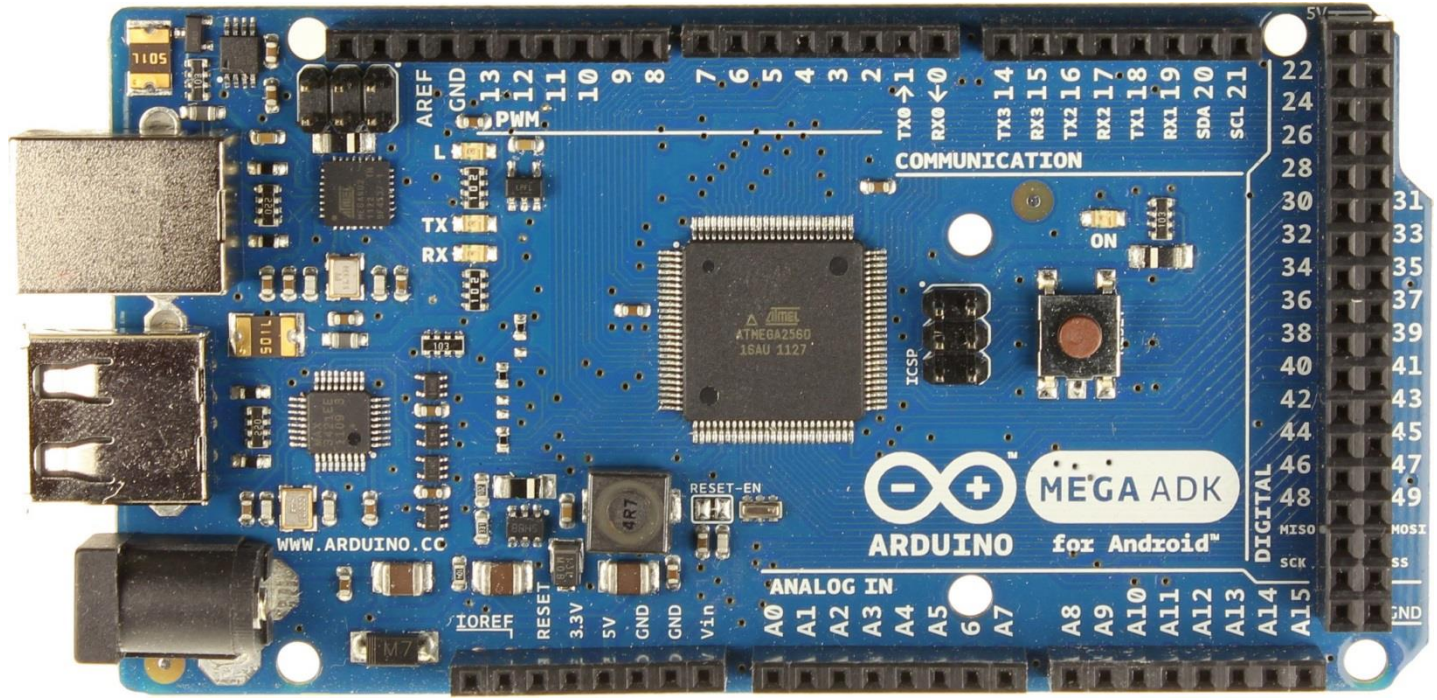
- We can use any digital output pin.
- We have full control the duty cycle and frequency.

Disadvantage:

- We can’t leave the output running while the processor does something else.
- Any interrupts will affect the timing

How to generate PWM?





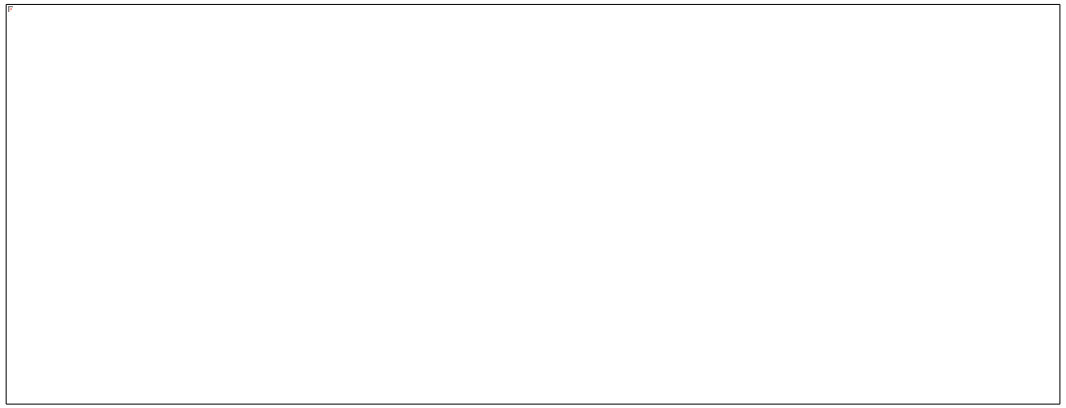
How to generate PWM?

b) Simple Pulse Width Modulation with analogWrite.

Simply we can call analogWrite(pin, dutyCycle), where dutyCycle is a value from 0 to 255, and pin is one of the PWM pins (pins 2:13,44,45,46).

This method is usfel to make the RGB LED light different colors as,

SALMON RGB(250, 128,114) – CHOCOLATE RGB(210, 105, 30).



How to generate PWM?

b) Simple Pulse Width Modulation with analogWrite.

Advantage:

- We can leave the output running while the processor does something else.
- We have full control the duty cycle only.

Disadvantage:

- We don’t have full control the frequency (490 Hz or 980 Hz based on the selected pin).

How to generate PWM?

c) Using the ATmega PWM registers directly.

- The ATmega328P chip has 3 PWM timers (2 8-bit and 1 16-bit), controlling 6 PWM o/p's. The ATmega2560 chip has 6 PWM timers (2 8-bit and 4 16-bit), controlling 15 PWM o/p's.
- By manipulating the chip's timer registers directly, you can obtain more control than the analogWrite function provides.
- Timers known as Timer 0, Timer 1,.., and Timer 6.
- Each timer has two or more output compare registers that control the PWM width for the timer's outputs.
- When the timer reaches the compare register value, the corresponding output is toggled.
- Each of the timers has a prescaler that generates the timer clock.
- The Arduino has a system clock of 16MHz.

How to generate PWM?

Timers (ATmega328P):

timer	bits	channel	UNO pin
timer0	8	A	6
timer0	8	B	5
timer1	16	A	9
timer1	16	B	10
timer2	8	A	11
timer2	8	B	3

How to generate PWM?

Timers (ATmega2560):

timer	bits	channel	Mega pin
timer0	8	A	13
timer0	8	B	4
timer1	16	A	11
timer1	16	B	12
timer2	8	A	10
timer2	8	B	9
timer3	16	A	5
timer3	16	B	2
timer3	16	C	3
timer4	16	A	6
timer4	16	B	7
timer4	16	C	8
timer5	16	A	44
timer5	16	B	45
timer5	16	C	46

How to generate PWM?

- The outputs of each timer will normally have the same frequency but can have different duty cycles (depending on the respective output compare register).
- Each of the timers has a prescaler that generates the timer clock by dividing the system clock by a prescale factor such as 1, 8, 64, 256, or 1024.
- The Arduino has a system clock of 16MHz and the timer clock frequency will be the system clock frequency divided by the prescaler factor.

How to generate PWM?

The Timer/Counter Control Registers for timer 2:

Bit	7	6	5	4	3	2	1	0	
(0xB0)	**COM2A1**	**COM2A0**	**COM2B1**	**COM2B0**	**–**	**–**	**WGM21**	**WGM20**	**TCCR2A**
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit	7	6	5	4	3	2	1	0	
(0xB1)	**FOC2A**	**FOC2B**	**–**	**–**	**WGM22**	**CS22**	**CS21**	**CS20**	**TCCR2B**
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

For more details, please check LAB1 manual or Atmega328p & Atmega2560 datasheets

How to generate PWM?

Clock Select Bit Description

CS22	CS21	CS20	Description
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk_{T2S}/(No prescaling)
0	1	0	clk_{T2S}/8 (From prescaler)
0	1	1	clk_{T2S}/32 (From prescaler)
1	0	0	clk_{T2S}/64 (From prescaler)
1	0	1	clk_{T2S}/128 (From prescaler)
1	1	0	clk_{T2S}/256 (From prescaler)
1	1	1	clk_{T2S}/1024 (From prescaler)

How to generate PWM?

Waveform Generation Mode Bit

Mode	WGM2	WGM1	WGM0	Timer/Counter Mode of Operation	TOP	Update of OCRx at	TOV Flag Set on (1)(2)
0	0	0	0	Normal	0xFF	Immediate	MAX
1	0	0	1	PWM, Phase Correct	0xFF	TOP	BOTTOM
2	0	1	0	CTC	OCRA	Immediate	MAX
3	0	1	1	Fast PWM	0xFF	BOTTOM	MAX
4	1	0	0	Reserved	–	–	–
5	1	0	1	PWM, Phase Correct	OCRA	TOP	BOTTOM
6	1	1	0	Reserved	–	–	–
7	1	1	1	Fast PWM	OCRA	BOTTOM	TOP

How to generate PWM?

Compare Output Mode, Fast PWM Mode.

Output A

COM2A1	COM2A0	Description
0	0	Normal port operation, OC2A disconnected
0	1	WGM22 = 0: Normal Port Operation, OC2A Disconnected WGM22 = 1: Toggle OC2A on Compare Match
1	0	Clear OC2A on Compare Match, set OC2A at BOTTOM (non-inverting mode)
1	1	Set OC2A on Compare Match, clear OC2A at BOTTOM (inverting mode)

Output B

COM2B1	COM2B0	Description
0	0	Normal port operation, OC2B disconnected
0	1	Reserved
1	0	Clear OC2B on Compare Match, set OC2B at BOTTOM (non-inverting mode)
1	1	Set OC2B on Compare Match, clear OC2B at BOTTOM (inverting mode)

How to generate PWM?

The frequency of generated form

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

Where:

$f_{clk\_I/O}$	The Arduino clock frequency.
TOP	The top value for the counter, based on the Timer/Counter mode of operation
N	Variable represents the prescaler factor.

How to generate PWM?

Fast-PWM mode:

- Is the simplest PWM mode, the timer repeatedly counts from 0 to 255 : 65,536. Why?
- The output turns on when the timer is at 0 and turns off when the timer matches the output compare register.

The following code fragment sets up non-inverting fast PWM on pins 9 and 10 (Timer 2) with prescalar=64 (976.5625Hz).

```
pinMode( 9, OUTPUT);
pinMode( 10, OUTPUT);
TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS22);
OCR2A = 180;
OCR2B = 50;
```

BV: bit value

Timer 2 – Arduino Mega

How to generate PWM?

Example:

4.2.1 Pre_experiment

1. Clock frequency of the Mega board is 16MHZ. It is necessary to generate a PWM signal around 2 kHz with 10 bit resolution. Calculate the prescalar value for the operation.

2. Assuming the above (1) PWM signal is on OCR1A pin Write down the values that should be in the Timer/Counter 1 control registers (TCCR1A, TCCR1B, TCCR1C).

```
pinMode( 9, OUTPUT);
pinMode( 10, OUTPUT);
TCCR2A = _BV(COM2A1) | _BV(COM2B1) | _BV(WGM21) | _BV(WGM20);
TCCR2B = _BV(CS22);
OCR2A = 180;
OCR2B = 50;
```

BV: bit value

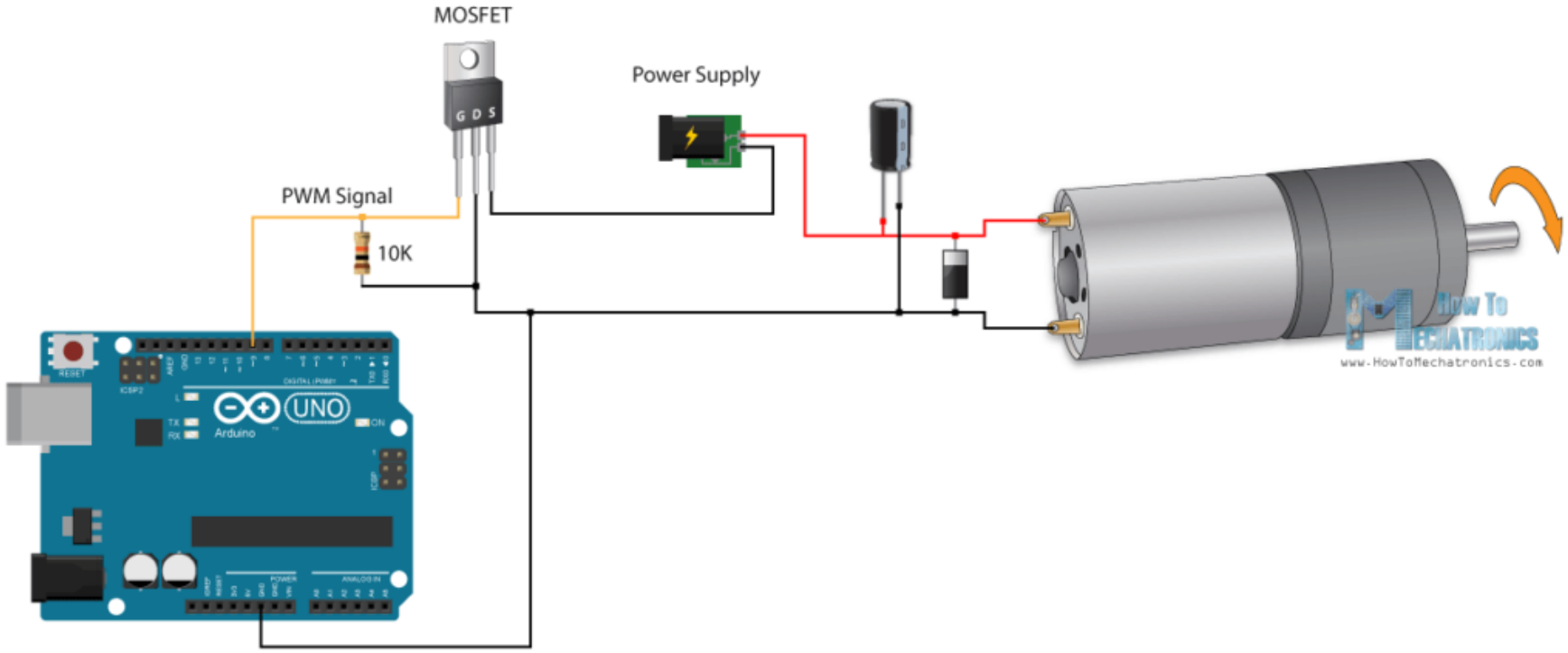
Timer 2 – Arduino Mega

How to generate PWM?

3. Develop the Arduino program that would initiate the PWM with given configurations in (2).

PWM DC Motor Control

- Depending on the size of the motor, we can simply connect an Arduino PWM output to the base of transistor or the gate of a MOSFET and control the speed of the motor by controlling the PWM output.



PWM DC Motor Control

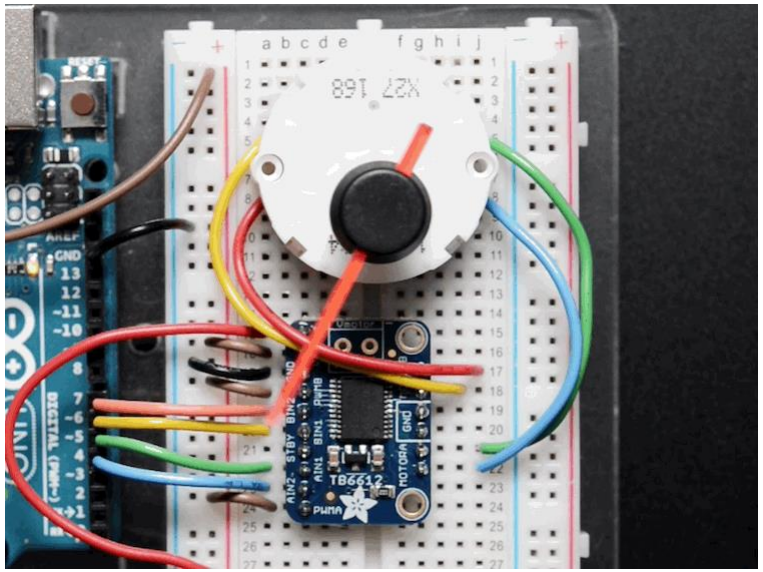
Advantages:

- The low power Arduino PWM signal switches on and off the gate at the MOSFET through which the high-power motor is driven.

Disadvantages:

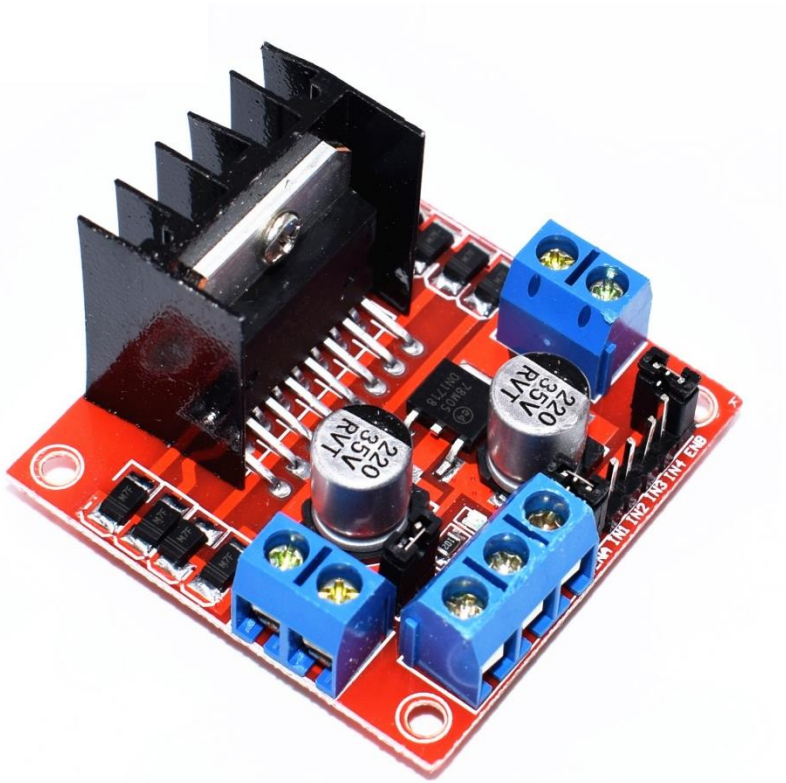
- This MOSFET connection can't be used for controlling the rotation direction.

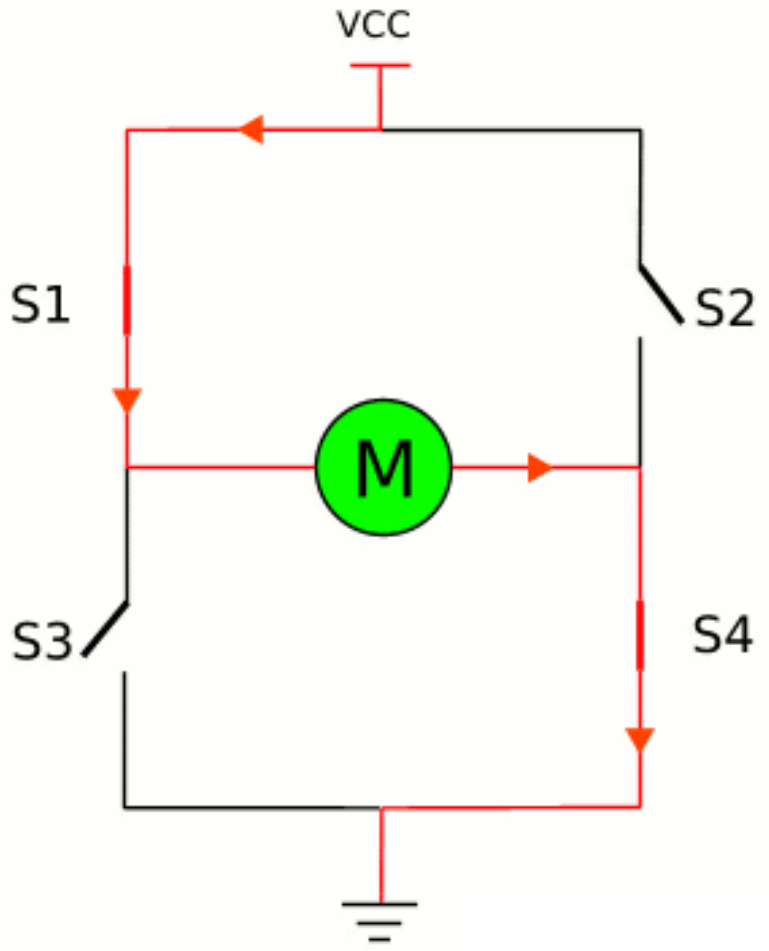
The motor polarity should be reversed in order to change the rotation direction.



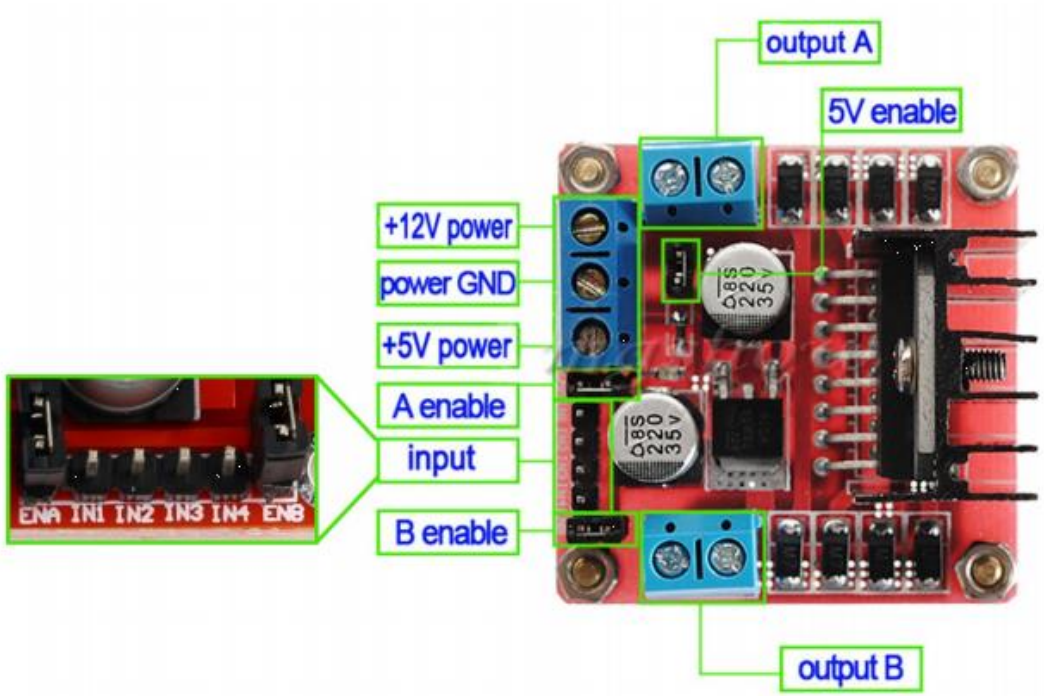
PWM DC Motor Control

L298N Driver (dual H-Bridge driver)



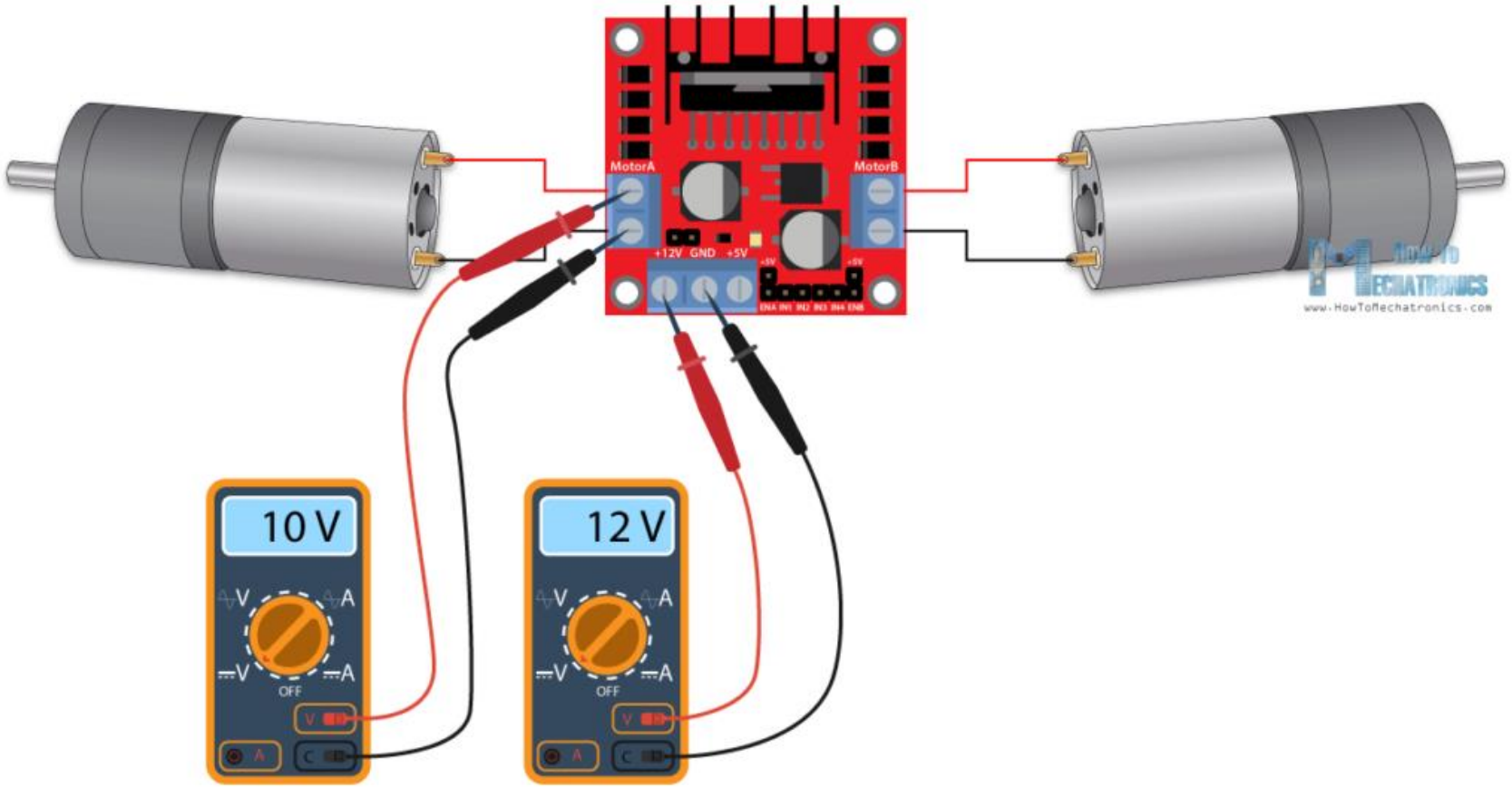


L298N Driver (dual H-Bridge driver)

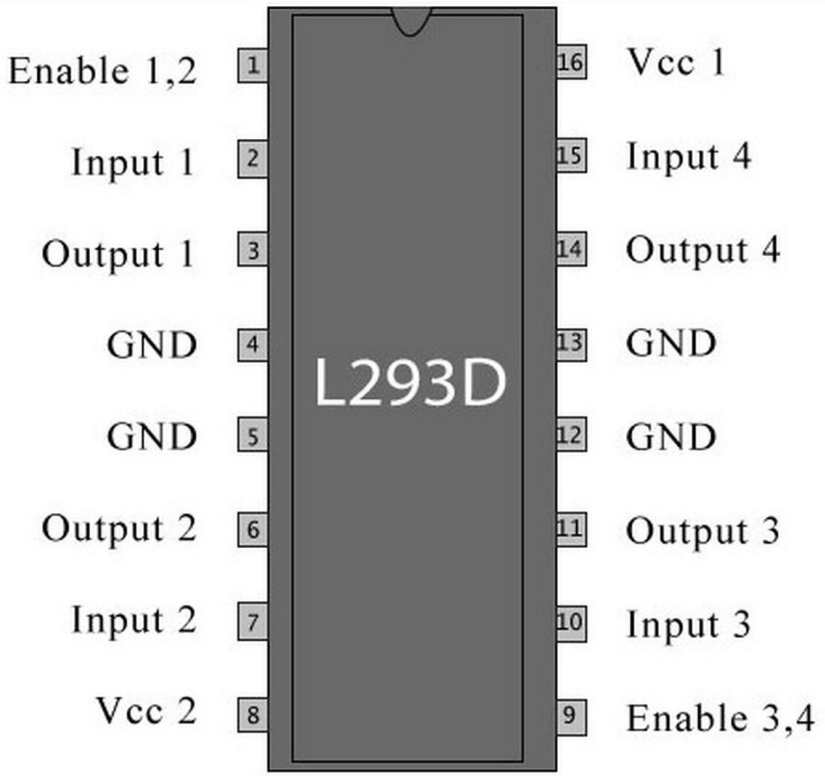


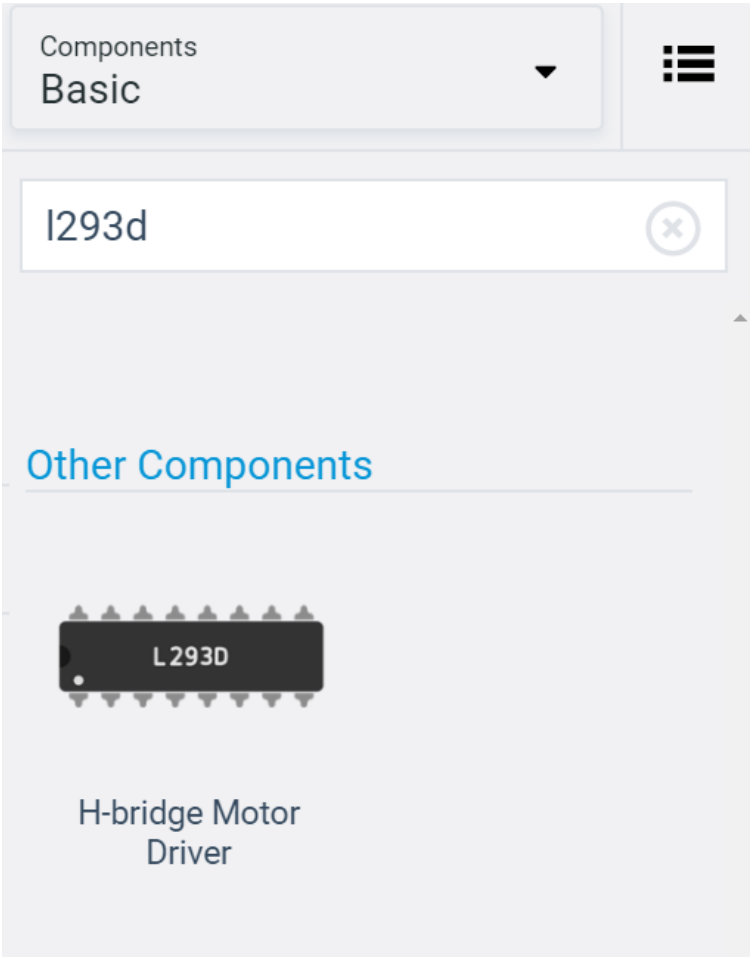
PWM DC Motor Control

L298N Driver (dual H-Bridge driver)

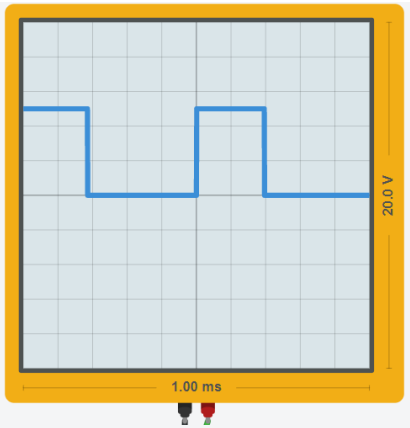


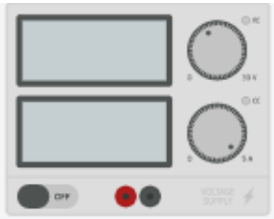
L298D Driver (dual H-Bridge driver)

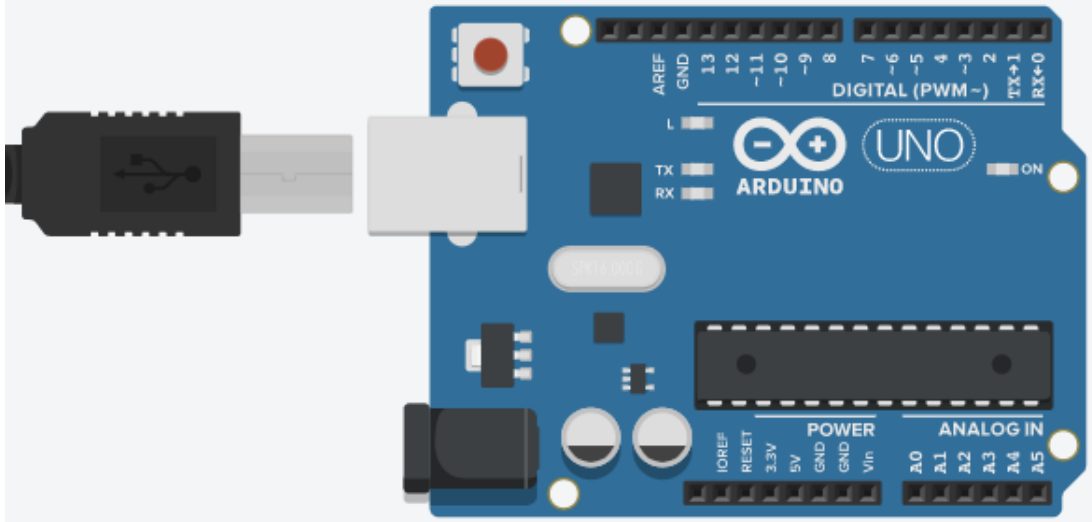


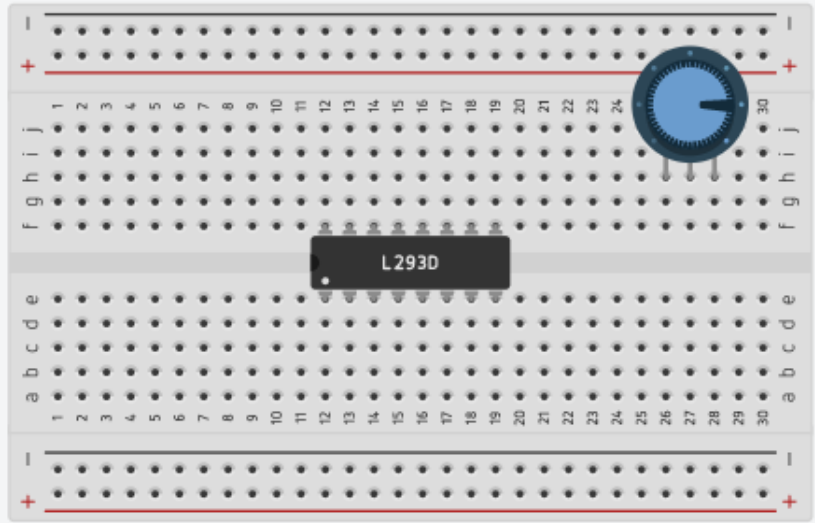


PWM DC Motor Control









PWM DC Motor Control

$$\eta = \frac{P_{\mathrm{L}}}{P_{\mathrm{S}}} = \frac{V_{\mathrm{L(RMS)}} I_{\mathrm{D(RMS)}}}{V_{\mathrm{DD}} I_{\mathrm{DC}}}$$

8. Prepare a spread sheet with the following column entires:
   a. *Positive Width* of PWM signal in microseconds (from the oscilloscope)
   b. *Period* of PWM signal in microseconds (from the oscilloscope)
   c. $Duty\ cycle = \frac{Positive\ Width}{Period} \times 100\%$
   d. I_{DC} = load current with DMM on DC setting (average current)
   e. I_{AC} = load current with DMM on AC setting
   f. $I_{D(RMS)} = \sqrt{(I_{AC})^2 + (I_{DC})^2}$ = true RMS value of load current
   g. V_{DC} = load voltage with DMM on DC setting
   h. V_{AC} = load voltage with DMM on AC setting
   i. $V_{L(RMS)} = \sqrt{(V_{AC})^2 + (V_{DC})^2}$ = true RMS value of load voltage
   j. $P_S = V_{DD} I_{DC}$ = power drawn from the source
   k. $P_L = V_{L(RMS)} I_{D(RMS)}$ = power delivered to the load
   l. $\eta = \frac{P_L}{P_S}$ = efficiency of the power delivery to the load

Q&A