Name: ..... Student ID: .....
Name: ..... Student ID: .....

Laboratory 06
Introduction to PWM and Motor Control

1. Pulse Width Modulation

PWM control is used in a variety of applications, ranging from telecommunications to robotic control.

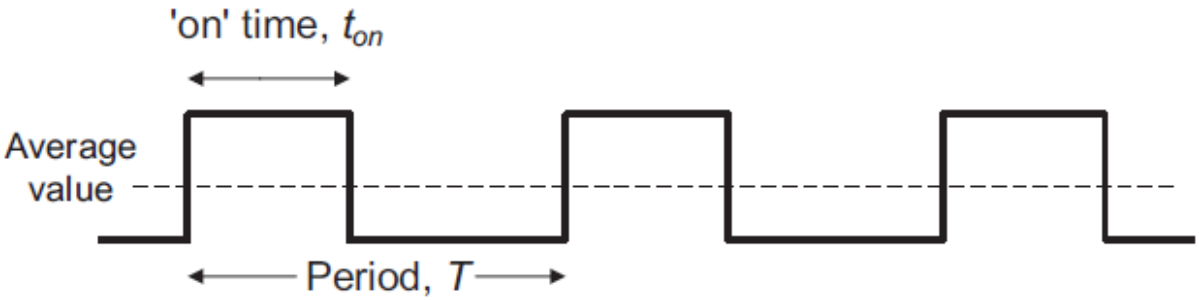


Fig.1. PWM signal

The period is normally kept constant, and the pulse width, or 'on' time, is varied, hence the name. The duty cycle is the proportion of time that the pulse is 'on' or 'high', and is expressed as a percentage, i.e.:

Duty cycle = (pulse on time / pulse period) * 100%

A 100% duty cycle therefore means 'continuously on' and a 0% duty cycle means 'continuously off '. PWM streams are easily generated by digital counters and comparators, which can readily be designed into a microcontroller. They can also be produced simply by program loops and a standard digital output, with no dedicated hardware at all.

2. Using the PWM Sources

The LPC1769 microcontroller has six PWM outputs, on P2.0- P2.5. Each has two variables, **period** and **pulse width**, or **duty cycle** derived from these. A PWM output can be established, named and allocated to a pin using **PwmOut.** All PWM outputs share the same period/frequency; if the period is changed for one, then it is changed for all.

Functions	Usage
PwmOut	Create a PwmOut object connected to the specified pin
write	Set the output duty-cycle, specified as a normalised float (0.0 – 1.0)
read	Return the current output duty-cycle setting, measured as a normalised float (0.0 – 1.0)
period	Set the PWM period, specified in seconds (float), keeping the duty cycle the same.
period_ms	Set the PWM period, specified in milliseconds (int), keeping the duty cycle the same.
period_us	Set the PWM period, specified in microseconds (int), keeping the duty cycle the same.
pulsewidth	Set the PWM pulse width, specified in seconds (float), keeping the period the same.
pulsewidth_ms	Set the PWM pulse width, specified in milliseconds (int), keeping the period the same.
pulsewidth_us	Set the PWM pulse width, specified microseconds (int), keeping the period the same.
operator =	An operator shorthand for write()

3. A trial PWM output

```
/*Sets PWM source to fixed frequency and duty cycle. */

#include "mbed.h"
PwmOut PWM1(P2_0);          //create a PWM output called PWM1 on P2.0

int main() {
  PWM1.period(0.010);       // set PWM period to 10 ms
  PWM1=0.5;                 // set duty cycle to 50%
}
```

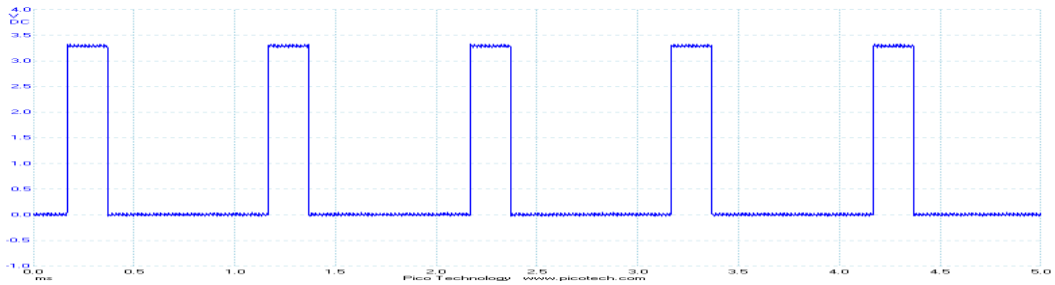
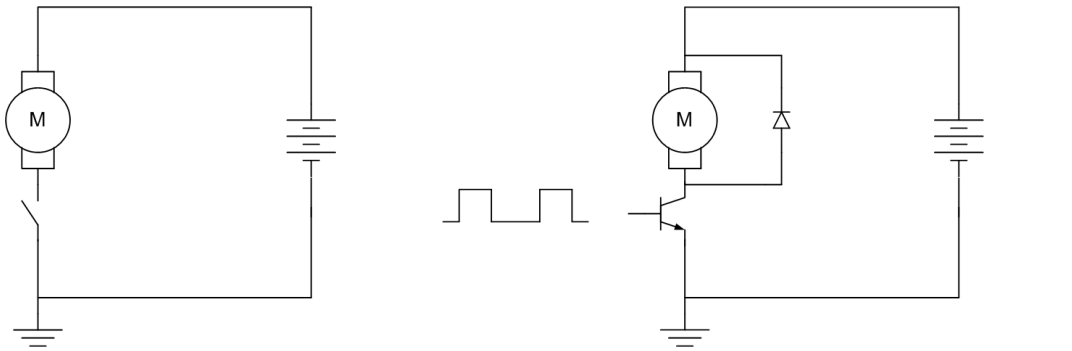


Fig.2. Example of PWM

```
/* PWM control to DC motor is repeatedly ramped      */
#include "mbed.h"
PwmOut PWM1(P0_8);
float i;
int main() {
  PWM1.period(0.010);                  //set PWM period to 10 ms
  while(1) {
    for (i=0;i<1;i=i+0.01) {
      PWM1=i;                          //update PWM duty cycle
      wait(0.2);
      }
  }
}
```

4. Using PWM to Control DC Motor

DC Motor rotates as DC current flow through it. The rotation speed is proportional to amount of the current and the direction of rotation depends on direction of the current. We can connect DC motor to a DC supply and a switch as shown in Figure 3(a). When switch is closed and current flows motor will begin to rotate. Its rotating direction depends the direction of current. If we make current to flow in opposite direction, motor will rotate in opposite direction as well.



(a) Using a switch **(b) Using a transistor and PWM signal**

Fig.3. Connecting a DC motor

In stead of using switch which is either turn on or turn off all the time, we can use transistor as a switch and control it by PWM signal as shown in Figure 3(b). With this technique, the average amount of current flow through motor will be proportional to duty cycle of the PWM signal. Note that there is a **freewheeling** diode connecting across the motor. This is to protect the transistor from being damaged by high voltage transient occurs when current flowing through motor is interrupted.

To make a motor rotate bidirectional, we can connect motor in H-Bridge configuration as shown in Figure 4. In this configuration, turning on A and D while turning off B and C

causes motor to rotate clockwise. On the other hand, turning on B and C while turning off A and D causes motor to rotate counter clockwise.

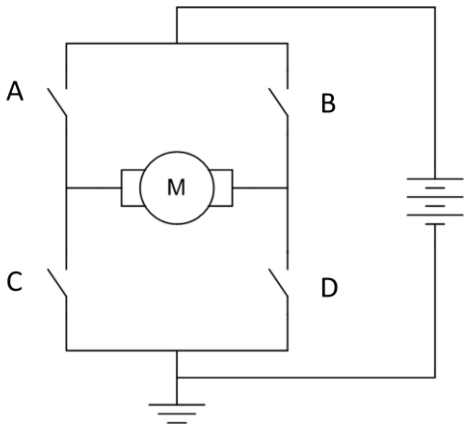
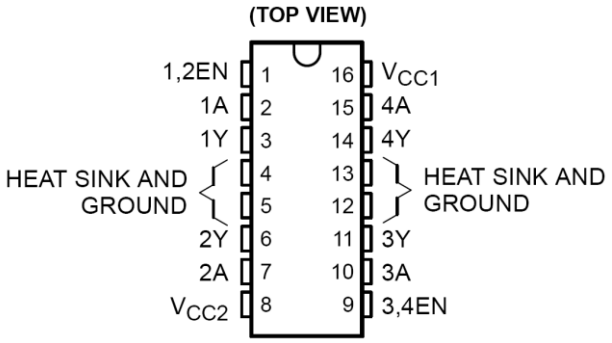


Fig.4 H-bridge configuration

4.1 Using L293D Driving DC Motor



$4.5 < V_{CC1} < 7$

$V_{CC1} < V_{CC2} < 36$

$2.3 < V_{IH} < V_{CC1}$

$-0.3 < V_{IL} < 1.5$

Continuous output current 600 mA

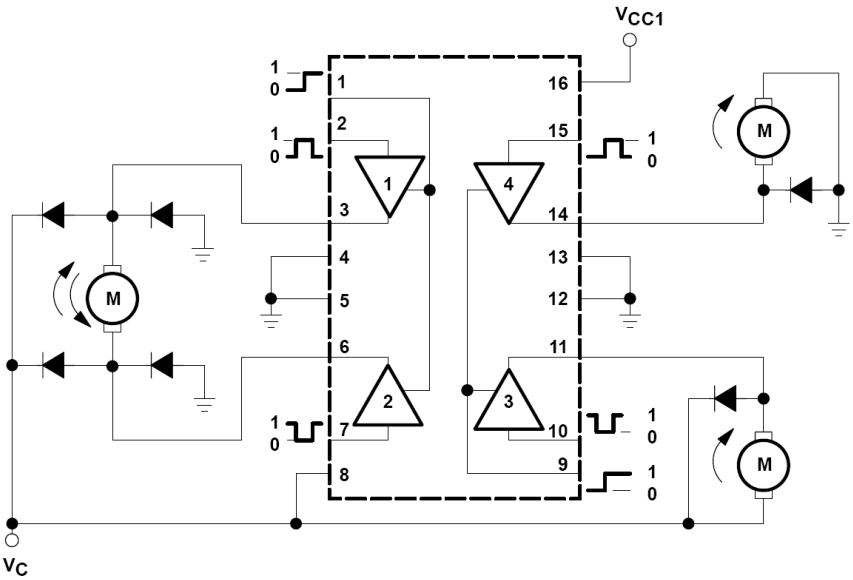
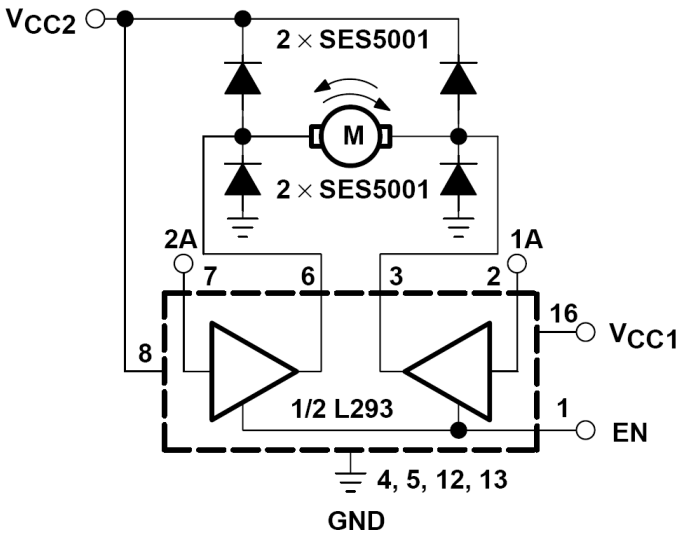


Fig. 5 Diagram of L293D

Because amount of current needed to a drive motor is mostly greater than the current driving capacity of microcontroller. Therefore in many applications, it needs an IC designed particularly to drive motor such as L293D. This type of IC typically can provide a large current from hundreds of mA to several amperes. It usually requires some digital logic signals to enable and allows PWM signals to control speed of the motor. It also can be configured as an H-bridge. Therefore it is very convenient to controlled both speed and rotation direction of a motor.

From Figure 5, a package of L293D contains 4 drivers. Each driver is connected to 3 pins, **Enable (EN)**, **Control (xA)** and **Output (xY)**. It operates as follows. When connecting logic **High** to **EN**, it enables the driver. If the **xA** receives logic High, it will make the driver **sourcing** current from the chip to motor via the **xY** pin. On the other hand, receiving logic **Low** will make the driver **sinking** current from the motor into the chip.

L293D requires 2 power supplies, V_{CC1} and V_{CC2}. V_{CC1} is used for operation of internal control circuits (not shown in Fig.5) of L293D. V_{CC2} is used for operation of the driver circuits as well as the motor. Thus V_{CC2} can be a high voltage for a large size motor. Summary of the recommended ranges of the supplies and logic levels are shown in Fig.5



EN	1A	2A	FUNCTION
H	L	H	Turn right
H	H	L	Turn left
H	L	L	Fast motor stop
H	H	H	Fast motor stop
L	X	X	Fast motor stop

L = low, H = high, X = don't care

Fig.6. Circuit connection for Bidirectional control using L293D

The circuit connection and table of control signals for bidirectional motor control is illustrated in Figure 6. In this case, driver 1 and 2 are connected to operate in H-bridge configuration.

5. Servo Control

A servo motor as shown in Figure 7 is a small **rotary position control device**, often used in radio-controlled cars and aircraft to control angular position of variables such as steering, elevators and rudders. Servos are now popular in certain robotic applications. The servo shaft can be positioned to specific angular positions by sending the servo a PWM signal. As long as the modulated signal exists on the input line, the servo will maintain the angular position of the shaft. As the modulated signal changes, the angular position of the shaft changes. This is illustrated in Figure 8. Many servos use a PWM signal with a 20 ms period, as is shown here. In this example, the pulse width is modulated from 1.25 ms to 1.75 ms to give the full 180 degree range of the servo.

The servo requires a higher current than the USB standard can provide, and so it must be powered using an external supply, e.g. with a 4xAA (6 V) battery pack. The LPC1769 itself can still be supplied from the USB. Alternatively, because the pack voltage lies within the permissible input voltage range for pin 2, the LPC1769 could be supplied from the battery pack, through this pin. The LPC1769 then regulates the incoming 6 V to the 3.3 V that it requires.

Fig.7 Servo motor and its connection

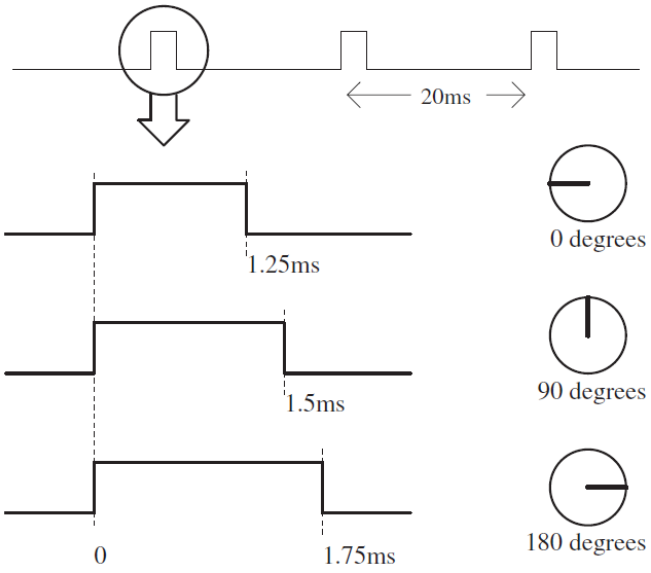


Fig. 8 Control servo motor with PWM

5. Using mbed API

Download mbed_API from
http://www.kmitl.ac.th/~kvkasin/courses/Micro/lecture_notes/mbed_Lxpresso.zip

1) Start LPCXpresso
2) Select an empty workspace
3) Go to Project Explorer panel -> Import -> General -> Import existing projects
4) Browse to mbed_lXpresso folder
5) Unselect "copy projects into workspace" and then click Finish
6) Configure Projects
    a. Right-click on the project (app_led)
    b. Click Build Configurations -> Set Active
7) Configure MCU
    a. Right-click on the project (app_led)
    b. Click Properties -> C/C++ Build -> MCU Settings
8) Configure the indexer
    a. Click Window -> Preferences -> C/C++ -> Indexer -> Use active build configuration
9) Modify app_led's main.cpp

6. Experiment

Create a project and write a program that sets a PWM output**.**

1. Using LPC1769 and L293 control speed of a motor. Use VCC1 = VCC2 =6 V

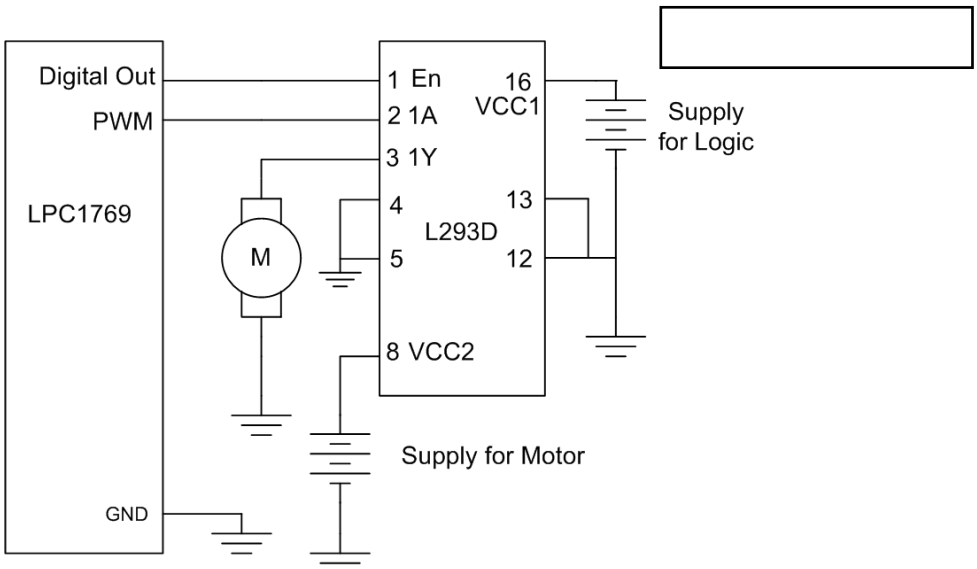


Fig. P1

2. Using LPC1769 and L293 control both speed and rotating direction of a motor. Use VCC1=VCC2=6V

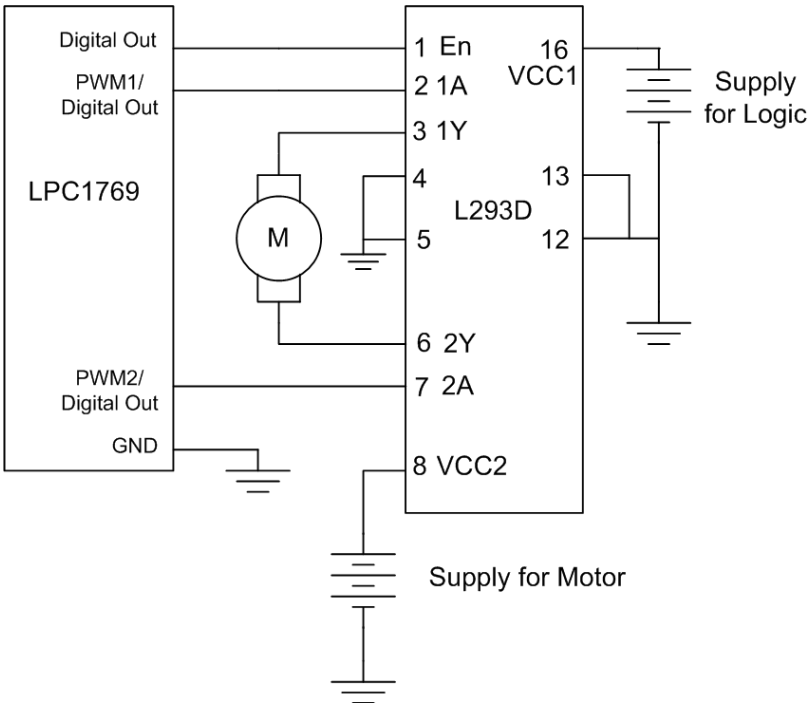


Fig. P2

3. Set a PWM output on pin P2_0. Set the PWM period to 20 ms. Try a number of different duty periods, taking values from the examples, and observe the servo' s position. Then write a program that continually moves the servo shaft from one limit to the other. Use 6V supply.

7. References

[1] Rob Toulson and Tim Wilmshurst, "Fast and Effective Embedded Systems Design Applying the ARM mbed", Elsevier 2012.

[2] L93D Data sheet, Texas Instrument 2002