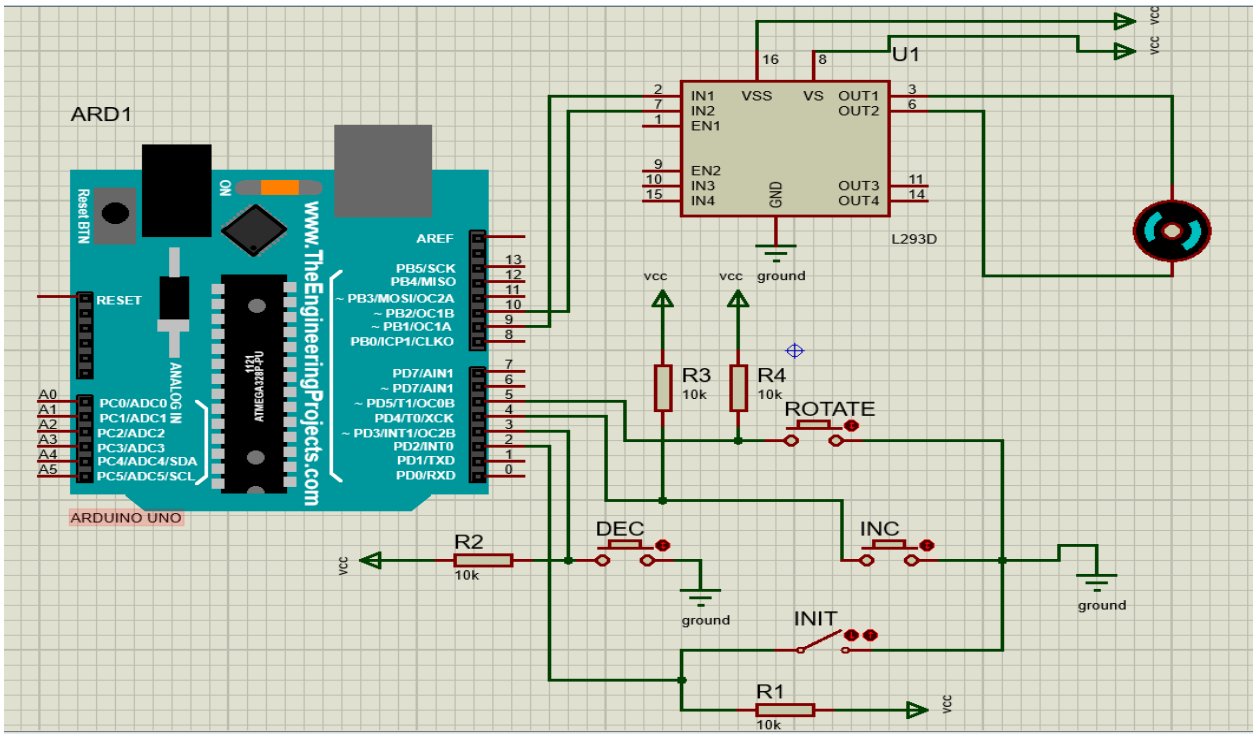
گزارش موتور DC:

تصویر مدل مفهومی مدار را در زیر می‌بینیم:



همانطور که در شکل بالا مشخص می‌باشد این مدار از دو جز اصلی تشکیل شده است. قسمت اول بورد arduino بوده که کد برنامه روی آن آپلود شده و قسمت دوم نیز قطعه L293 می‌باشد. قطعات دیگر شامل موتور و کلیدها در کنار مقاومت‌ها می‌باشند. مقاومت‌ها نیز به عنوان pull up هستند که علت آن نیز جلوگیری از اتصال کوتاه بین منبع و زمین و همچنین ایجاد امکان مقداردهی ۰ و ۱ به پین‌های مورد نظر است. در صورتی که از مقاومت استفاده نشود زمانی که کلید زده می‌شود کار انجام می‌شود اما زمانی که کلید را نمی‌زنیم آنگاه اتفاقی نخواهد افتاد و مدار از کار خواهد افتاد.

* قسمت تعاریف و setup کردن کد:

```
int inPinOnOff = 2;
int inPinDec = 3;
int inPinInc = 4;
int inPinChangeDir = 5;
int outPinDC1 = 9;
int outPinDC2 = 10;
int isFunctional = 0;
int Maxpwm = 255;
int Minpwm = 0;
int velocity = Maxpwm/2;
int pwm_cnt = 0;
char spin_direction = 'r';
```

```
void setup() {
  // denoting input pins for setup:
  pinMode(inPinOnOff, INPUT);
  pinMode(inPinDec, INPUT);
  pinMode(inPinInc, INPUT);
  pinMode(inPinChangeDir, INPUT);

  // denoting output pins for setup:
  pinMode(outPinDC1,  OUTPUT);
  pinMode(outPinDC2, OUTPUT);

  // starting...
  Serial.begin(9600);
}
```

در این قسمت که در دو عکس بالا قابل مشاهده می‌باشد متغیرها تعریف و مقداردهی شده‌اند. به این صورت که پایه‌های ۲ تا ۵ برای فرمان‌های مختلف به موتور, پایه‌های ۹ و ۱۰ برای خروجی دادن به موتور و بقیه موارد نیز برای مشخص کردن سرعت, جهت و روشن بودن موتور می‌باشند. در بخش setup نیز که یکبار اجرا می‌شود ۶ پین معرفی شده در بالا به صورت input و output مشخص شده و مدار شروع به کار می‌کند.

* توابع کمکی:

```
void turnMotor(char dir){
  if (dir == 'r') {
    turnRight();
  }
  else {
    turnLeft();
  }
}

void stopMotor(){
  digitalWrite(outPinDC1, LOW);
  digitalWrite(outPinDC2, LOW);
}
```

```
void turnRight(){
  digitalWrite(outPinDC1, HIGH);
  digitalWrite(outPinDC2, LOW);
}

void turnLeft(){
  digitalWrite(outPinDC1, LOW);
  digitalWrite(outPinDC2, HIGH);
}
```

در این توابع عملیات نوشتن روی خروجی به منظور چرخش موتور (در دو جهت مختلف توسط دو ترکیب مختلف) و متوقف کردن موتور توسط نوشتن دو مقدار LOW در خروجی انجام شده است.

* قسمت کد اصلی و loop:

```
void loop() {
  // Reading the current state of all keys:
  int InitKeyVal = digitalRead(inPinOnOff);
  int DecKeyVal = digitalRead(inPinDec);
  int IncKeyVal = digitalRead(inPinInc);
  int ChangeKeyVal = digitalRead(inPinChangeDir);
```

در ابتدا ۴ کلیدی که برای کنترل کردن موتور در نظر گرفته‌ایم به صورت دیجیتالی از روی پین‌ها خوانده می‌شوند.

```
if (isFunctional == 1) {
  if (pwm_cnt <= velocity) {
    turnMotor(spin_direction);
  }
  else {
    stopMotor();
  }
}
```

در صورتی که موتور در حال کار کردن باشد; اگر مقدار pwm count که در آخر لوپ هر دفعه آپدیت می‌شود کمتر از سرعت حاضر باشد اقدام به ادامه چرخش موتور کرده و در غیر این صورت موتور را متوقف می‌کنیم. (روش pwm)

```
else if (DecKeyVal == LOW) {
    while (digitalRead(inPinDec) == LOW){};
    velocity = velocity - 10;
    if (velocity <= Minpwm) {
      velocity = Minpwm;
    }
}

else if (ChangeKeyVal == LOW) {
    while (digitalRead(inPinChangeDir) == LOW){};
    if (spin_direction == 'r') {
      spin_direction = 'l';
    }
    else {
      spin_direction = 'r';
    }
}
}
```

```
if (isFunctional == 1) {
  if (InitKeyVal == LOW) {
    while (digitalRead(inPinOnOff) == LOW){};
    isFunctional = 0;
    stopMotor();
  }

  else if (IncKeyVal == LOW) {
    while (digitalRead(inPinInc) == LOW){};
    velocity = velocity + 10;
    if (velocity >= Maxpwm) {
      velocity = Maxpwm;
    }
  }
```

در مرحله بعدی حالت های مختلفی که زدن دکمه‌ها در مدار ما به وجود می‌آورند را هندل می‌کنیم.در هر یک از موارد در اول کار یک while تو خالی میگذاریم که تا زمانی که دکمه رها نشده است تاثیر آن دکمه نمایان نشود که به ناگهان مقادیر تغییر زیادی نکنند(با نگه داشتن دکمه مثلا). در توابع تغییر سرعت و جهت نیز به طور منطقی متغیرهای velocity و spin_direction تغییر می‌کنند.

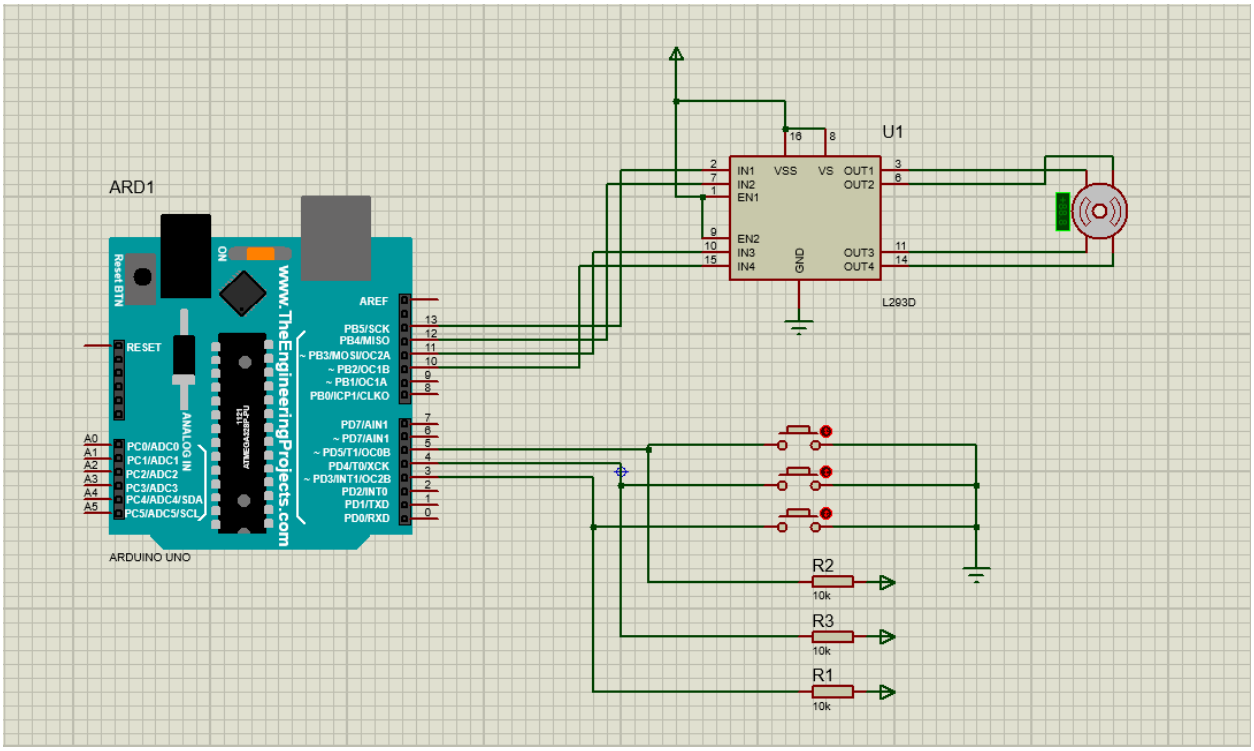
```
else {
  if (InitKeyVal == LOW) {
    while (digitalRead(inPinOnOff) == LOW){};
    isFunctional = 1;
  }
}

pwm_cnt = (pwm_cnt + 1) % (Maxpwm - Minpwm);
```

در مرحله آخر نیز در صورتی که موتور در حال کارکردن نباشد; اگر کلید روشن/خاموش فشرده شود موتور به راه می‌افتد. همچنین در خطر آخر نیز مقدار pwm_cnt را یک واحد زیاد کرده و چون طبق pwm این مقدار باید بین ۰ تا ۲۵۵ باشد آن را نسبت به اختلاف min و max (که همان ۲۵۵ است) mod می‌گیریم.

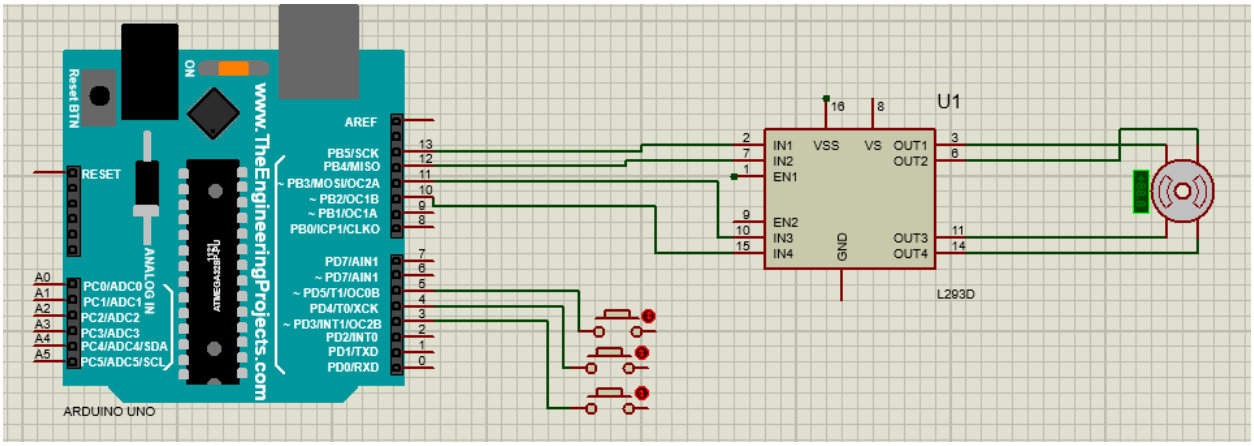
گزارش موتور Stepper

در این بخش ابتدا مدل مفهومی مداری را که پیاده‌سازی نموده‌ایم بررسی می‌کنیم و سپس به بخش کد مدار می‌پردازیم.

تصویر مدار را در زیر می‌بینیم:



همانگونه که در مدار دیده می‌شود، اجزای اصلی مدار شامل برد آردوینو UNO، ماژول L293، ماژول stepper و همچنین تعدادی کلید برای صدور فرمان‌های لازم است. نکته‌ی مهم این بخش مدار مقاومتی pull up است که برای کلیدها پیاده‌سازی شده است که علت آن نیز جلوگیری از اتصال کوتاه بین منبع و زمین و همچنین ایجاد امکان مقداردهی ۰ و ۱ به پین‌های مورد نظر است. در کنار این مورد ماژول L293 نیز مانند حالت نخست همچنان به عنوان یک درایور برای موتور استفاده می‌شود. برد آردوینو نیز به منظور پیاده‌سازی برنامه اصلی ماژول استفاده شده است. طرح مفهومی مدار نیز در شکل زیر آمده است:



توضیح کد

در اینجا ابتدا شبه کد تابع پیاده‌سازی شده را می‌آوریم و سپس با توجه به کد اصلی به توضیح بخش‌های مختلف می‌پردازیم:

```
Main function def {
        Initialize pins and motor state;
        While (true) {
                Get values from pins
                If (stop_button_pushed)
                        stop motor rotation;
                If (rotate_right_button_pushed)
                        start rotate right button;
                If (rotate_left_button_pushed)
                        start rotate left button;
                Update state of motor
        }
```

}

با توجه به آنچه که در شبه کد هم می‌بینیم حلقه‌ی اصلی اجرای برنامه سه بخش اصلی دارد که نخست دریافت ورودی و سپس تغییر استیت و در نهایت نیز به روز رسانی استیت در موتور است.

کد زده شده شامل دو تابع اصلی setup و همچنین loop است. در تابع نخست به مقداردهی اولیه و آماده‌سازی مدار برای استفاده می‌پردازیم و در تابع دوم نیز که به صورت تکراری همواره اجرا می‌شود به پیاده‌سازی عملیات اصلی می‌پردازیم.

تابع setup

در این تابع ابتدا سرعت هر دور چرخش موتور stepper را تعیین می‌کنیم و سپس نیز پین‌های ورودی را مشخص می‌کنیم که بتوانیم دکمه‌ها را به آن‌ها متصل کنیم. در ادامه شبه کد مرتبط به این بخش آمده است.

```
void setup(){
    stepper_motor.setSpeed(RPM);
    pinMode(btn_ccw,INPUT);
    pinMode(btn_cw,INPUT);
    pinMode(btn_off,INPUT);
}
```

تابع loop

برای این بخش که بدنه‌ی اصلی تابع را تشکیل می‌دهد سه بخش عمده داریم:

۱. Perception: در این بخش اطلاعاتی را که از طریق پین‌های ورودی دریافت می‌شوند می‌خوانیم و آن‌ها را در متغیرهایی ذخیره می‌کنیم. این اطلاعات شامل سه پینی می‌شود که برای هر یک از دکمه‌هایی که برای صدور فرمان در مدار قرارداده‌شده‌اند استفاده شده‌اند. در اینجا باید دقت شود که برخلاف حالت DC از یک while استفاده نشده است زیرا این که کاربر دکمه را همچنان بفشارد یا نه اهمیتی ندارد و دلیل آن نیز این است که با فشردن دکمه

کارایی آن تغییر نمی‌کند و همان کارایی قبلی انجام می‌شود در نتیجه نیازی به این که منتظر بمانیم تا کاربر دکمه را رها کند نیست.

۲. Update State: در اینجا منظور از استیت یک متغیر dir است که به منظور تعیین جهت و حرکت یا عدم حرکت موتور استفاده می‌شود. برای به روز رسانی استیت نیز باید با توجه به مقادیری که از محیط دریافت شده است مقدار dir را ۱ یا ۰ یا ۱- قرار دهیم. مقدار ۱ به معنای حرکت ساعت‌گرد و ۱- به معنای حرکت پادساعتگرد است و ۰ نیز به معنای عدم حرکت موتور می‌باشد. همچنین شروط نیز در صورتی درستند که مقدار ورودی پین ۰ باشد چرا که مدار کلیدها با مقاومت pull up بسته شده و به نوعی می‌توان گفت که به صورت active low عمل می‌کند.

۳. Actuating: در این فاز با توجه به مقدار dir عملیاتی را که مدنظرمان است انجام می‌دهیم که در اینجا در واقع این عملیات همان حرکت دادن موتور محسوب می‌شود. برای انجام این کار از تابع step کلاس Stepper استفاده می‌کنیم که در واقع تعیین می‌کند که موتور چه مقدار حرکت کند. ورودی این تابع تعداد چرخش‌هایی را که موتور در یک بار حرکت باید انجام دهد مشخص می‌کند و در واقع می‌توان گفت میزان حرکت موتور برابر است با حاصلضرب ورودی در سرعت حرکت که در تابع setup تعیین شده است. نکته‌ای که در این تابع وجود دارد این است که این تابع به صورت blocking عمل می‌کند که این باعث می‌شود تا زمانی که چرخش انجام نشده به خطوط اولیه‌ی کد که مربوط به perception است نرسیم و همین مورد باعث می‌شود که گاهی با فشردن یک دکمه تغییری در عملیات انجام شده ایجاد نشود و باید دکمه را مدت زمانی نگه داریم تا تغییر را مشاهده کنیم.

در تصویر زیر شبه کد این بخش را مشاهده می‌کنید:

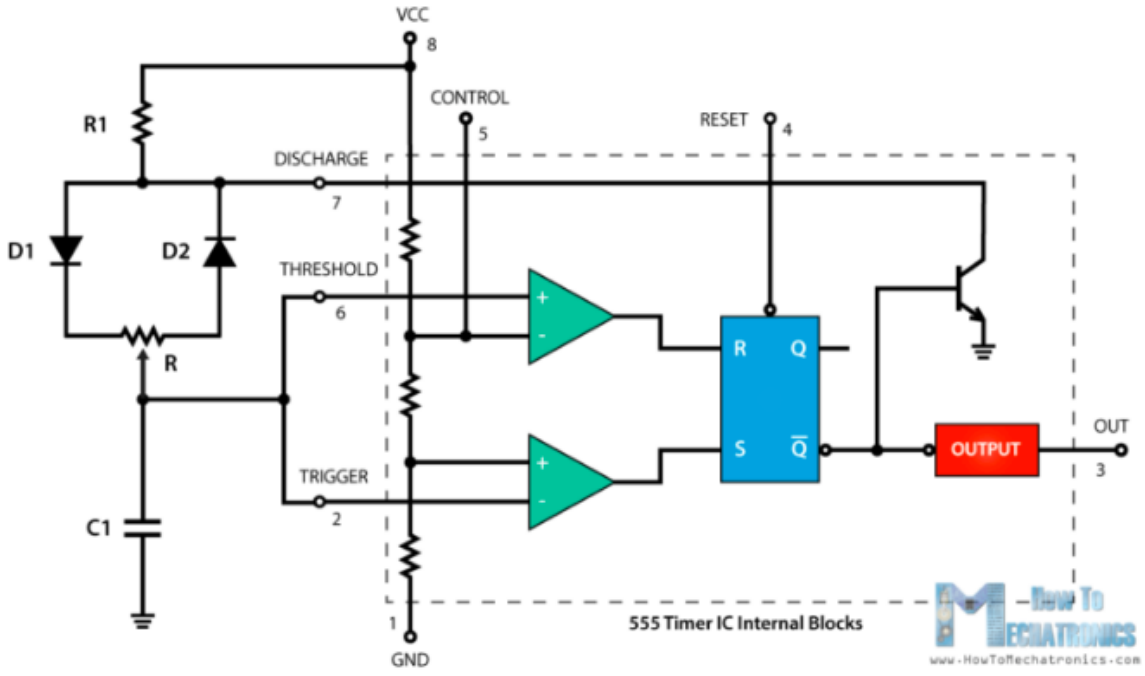
```
void loop(){
    //percept
    int btn_cw_val = digitalRead(btn_cw);
    int btn_ccw_val = digitalRead(btn_ccw);
    int btn_off_val = digitalRead(btn_off);
    //update state
    if (btn_cw_val == LOW){
        dir = 1;
    }
    if (btn_ccw_val == LOW) {
        dir = -1;
    }
    if (btn_off_val == LOW){
        dir = 0;
    }
    // actuate
    if (dir != 0){
        stepper_motor.step(dir);
    }
}
```

پاسخ سوالات متن

سوال اول:

برای ساخت PWM می‌توان به دو شکل سخت افزاری و نرم‌افزاری عمل کرد. برای ساخت آن به صورت نرم‌افزاری همانطور که در بخش اول گزارش کد آن را زدیم، با استفاده از یک شمارنده که در هر لحظه مقدار آن افزایش می‌یابد استفاده می‌کنیم. در حقیقت وظیفه اصلی PWM آن است که یک سیگنال دیجیتال به شکلی رفتار کند که گیرنده آن بتواند آن را آنالوگ ببیند به بیان دیگر هدف آن تبدیل سیگنال دیجیتال به آنالوگ است. برای شبیه سازی این رفتار کافی است در بازه‌های زمانی معین، مقدار سیگنال را صفر و یک کرد و به این شکل سنسور یا موتور یا هر شی دریافت کننده این سیگنال آن را یک سیگنال تناوبی با مقدار متوسط ولتاژ در آن دوره تناوب می‌بینند. بنابراین در پیاده سازی نرم‌افزاری آن هنگامی که شمارنده از عدد مشخصی بیشتر شد، مقدار سیگنال صفر می‌شود و قبل از آن مقدار یک است. همچنین برای ایجاد تناوب یک سقف بالا و کف پایین برای شمارنده در نظر گرفته می‌شود که هنگامی

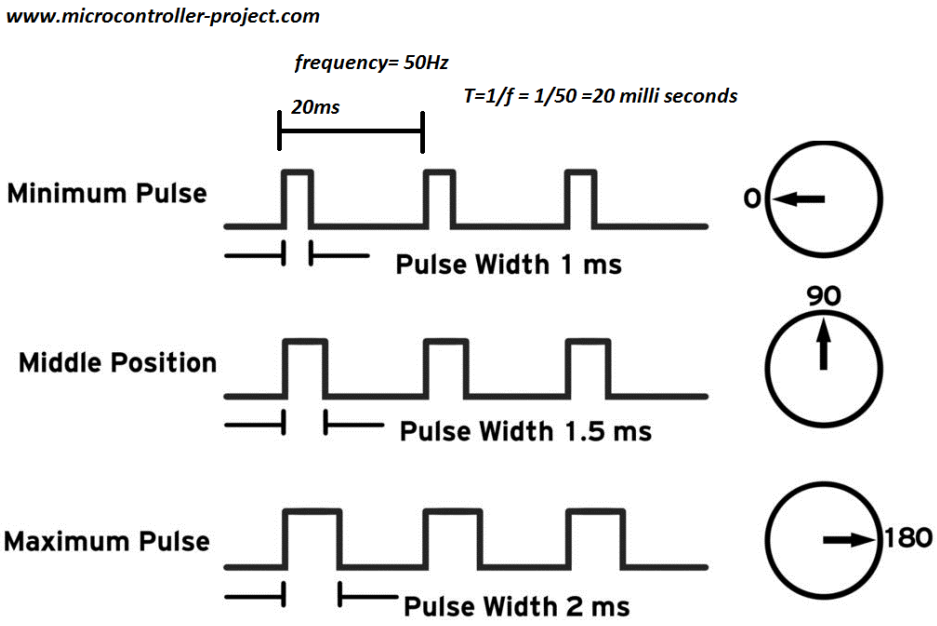
که از سقف بالا عبور کرد دوباره به کف برگردد و چون از حد مشخص کمتر شده است، دوباره سیگنال خروجی یک می‌شود. به این ترتیب رفتار تناوبی نیز شبیه سازی می‌گردد. از طرفی برای مشخص کردن مقدار قدرت سیگنال می‌توان آن حدی که قبل از آن یک و بعد از آن صفر است را مشخص و تنظیم کرد تا مقدار ولتاژ مناسب ایجاد شود. برای پیاده سازی سخت افزاری PWM یکی از راه‌ها این است که با استفاده از روش‌های مشخصی یک موج متناوب (به صورت موج مربعی) ایجاد کرد و سپس اگر بتوان duty cycle آن را عوض کرد، می‌توان عملکرد PWM را ایجاد کرد. برای ایجاد چنین مداری از IC LM555 استفاده می‌شود. مدار زیر برای ایجاد یک موج مربعی استفاده می‌شود. همچنین با تغییر مقدار مقاومت R1 می‌توان مدت زمان صفر و یک بودن را کنترل کرد.



سوال دوم:

PWM در موتور های DC به این صورت عمل می‌کند که duty cycle هر مقدار باشد با آن درصد از سرعت خود حرکت میکند. به طور مثال اگر duty cycle مقدار ۵۰ درصد داشته باشد موتور با ۵۰ درصد از حداکثر سرعت خود حرکت میکند.

اما در موتور های Servo دوره ی پالس مثبت pwm تغییر مکان موتور را مشخص می‌کند. همچنین اندازه ی پالس مشخص می‌کند که موتور در جهت ساعت‌گرد یا پادساعت‌گرد بچرخد. اگر مقدار پالس کم باشد موتور در جهت پادساعت‌گرد و اگر زیاد باشد به صورت ساعت‌گرد می‌چرخد.



سوال سوم:

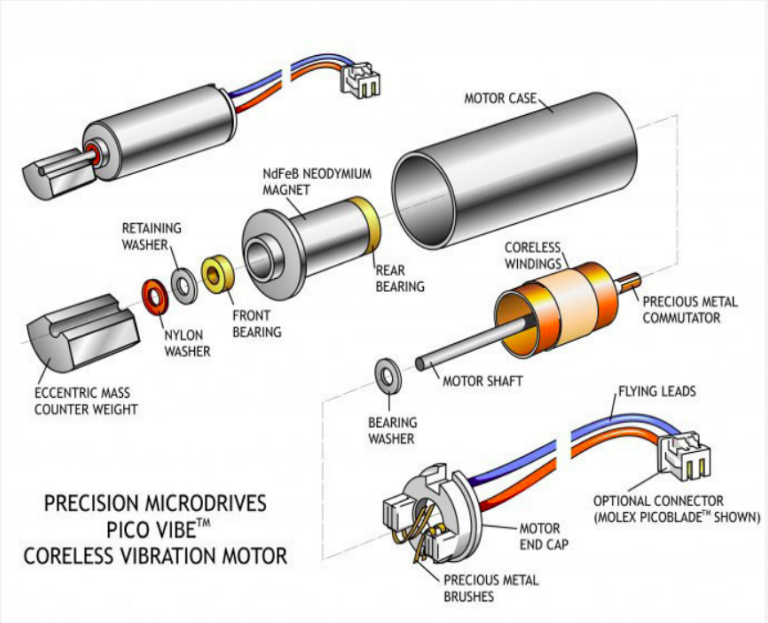
در ابتدای تمرین درباره شیوه کار هر یک از این دو نوع موتور توضیحاتی داده شد بنابراین ما از ارائه اطلاعات تکراری خودداری می‌کنیم و اطلاعاتی که بیان نشده را بازگو می‌کنیم. تفاوت اصلی دو موتور stepper و servo در شیوه چرخش آن‌ها است. در stepper چرخش به گونه‌ای انجام می‌شود که موتور در تعداد step مشخصی چرخش خود را انجام می‌دهد. برای مثال یک چرخش کامل به n مرحله تقسیم می‌شود و موتور فقط در زاویه‌های $\frac{360}{n} + initial\ \theta$ قرار می‌گیرد. موتورهای servo همانند موتورهای stepper هستند اما با این تفاوت که چرخش آن‌ها لزوما از تعداد مشخصی مرحله تشکیل نمی‌شود. در موتورهای servo با توجه به مقداری که از PWM دریافت می‌کند زاویه خود را تنظیم می‌کند، بنابراین می‌تواند هر زاویه دلخواهی بین ۰ تا ۳۶۰ درجه را داشته باشد. این موتورها در حقیقت از موتور DC، یک سنسور تشخیص موقعیت (معمولا potentiometer) و یک مجموعه

چرخدنده و یک مدار کنترلی تشکیل شده‌اند. نمونه‌ای از کاربرد موتور stepper در پرینترها، ماشین‌های تصویر برداری پزشکی و آینه ماشین‌ها استفاده می‌شود. نمونه‌ای از کاربرد موتور servo در بازوهای الکترونیک در روبات‌ها است.

سوال چهارم:

به طور کلی دو نوع موتور برای ایجاد ویبره در موبایل‌ها وجود دارد. دسته اول که موتورهای جرم چرخان خارج از مرکز یا به اختصار (ERM) هستند که در آن‌ها که یک جرم غیر متعادل (جرمی که بالانس نیست) با کمک یک موتور DC می‌چرخد و یک نیرو ایجاد می‌کند که به ویبره تبدیل می‌شود (بیشتر در ادامه توضیح داده می‌شود). دسته دوم ممموتورها تولید کننده ویبره موتورهای ویبره خطی هستند که یک جرم به یک فنر درحال ارتعاش و ایجاد موج متصل است و با تکان خوردن فنر نیرو توسط جرم ایجاد شده و ویبره ایجاد می‌شود. به طور دقیق‌تر به توضیح ERM می‌پردازیم:

این موتورها که از نوع موتورهای DC هستند، که یک جسم غیر بالانس (از نظر شکل ظاهری) به آن متصل است و با چرخیدن موتور، نیرویی توسط جسم غیر بالانس ایجاد شده و با انتقال این نیرو به جسم ویبره وارد می‌شود. تصویر زیر ساختار کلی این موتورها را بیان می‌کند:



از مزایای این نوع از موتورهای ویبراتور می‌توان به سادگی در ایجاد و کم هزینه بودن آن‌ها اشاره کرد به شکلی که تقریبا به طور کامل بازار موبایل‌های هوشمند را در دست گرفته‌اند. همچنین این موتورها در اندازه‌های متنوع قابل ایجاد هستند، بنابراین استفاده از آن‌ها در مکان‌های گوناگون ممکن است.