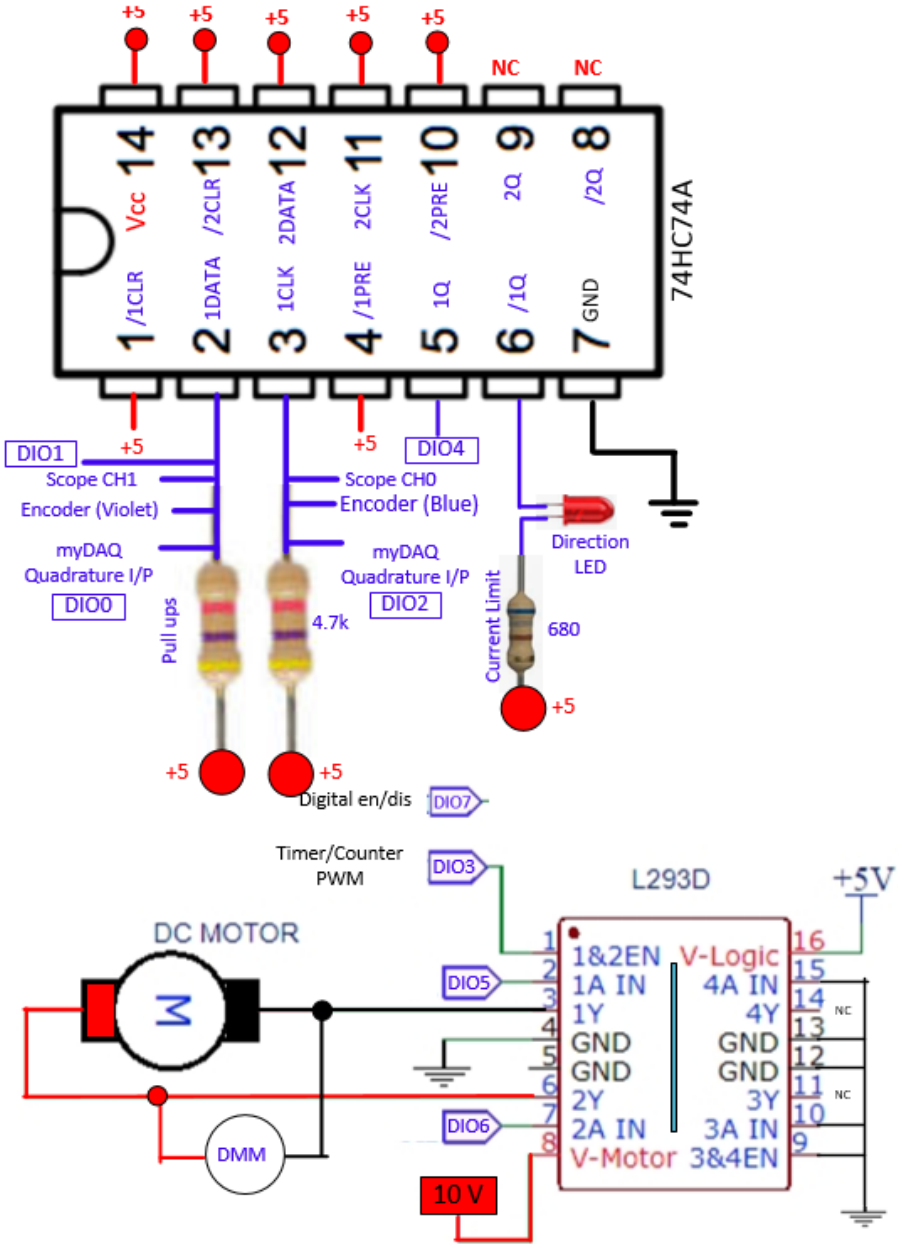


In Part A of lab 6:

DIO7 controls the L293D enable signal
DIO5 connects to IN1 on the L293D
DIO6 connects to IN2 on the L293D

In Part B of Lab 6:

DIO7 controls the L293D enable signal
DIO5 connects to IN1 on the L293D to control mode
DIO6 connects to IN2 on the L293D to control mode
DIO0 connects to one of the motor quadrature encoders
DIO2 connects to the other motor quadrature encoder signal

In Part C of Lab 6:

DIO3 (PWM) control the L293 Enable signal
DIO5 connects to IN1 on the L293D to control mode
DIO6 connects to IN2 on the L293D to control mode

In Part D of Lab 6:

myDAQ DIO7 controls the L293D enable signal
myDAQ DIO5 connects to IN1 on the L293D to control mode
myDAQ DIO6 connects to IN2 on the L293D to control mode
myDAQ DIO4 connects to the Q output of the 74HC74A to read direction
myDAQ DIO1 connects to the one of the motor quadrature encoder signal to measure speed

The 74HC74A data connects to one of the motor quadrature encoders to measure speed and determine direction.
The 74HC74A connects to the other motor quadrature encoder signal to determine direction.

DC Motor Types

Brushed DC

Advantages

- Cheapest and simplest motor
- Speed linear to applied voltage
- Simple motor control

Disadvantages

- High maintenance
- Low life-span (due to physical wear on brushes)

Brushless DC

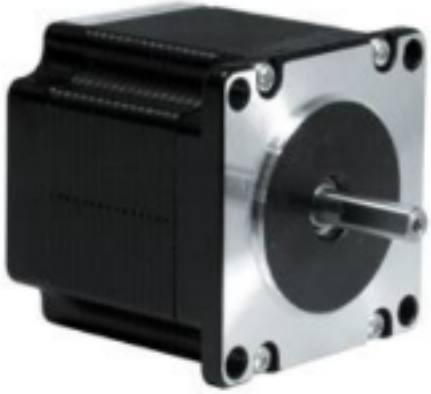
Advantages

- High efficiency
- Little to no maintenance
- Long life span
- High output power per frame size

Disadvantages

- More complicated motor control
- Large initial costs

Stepper



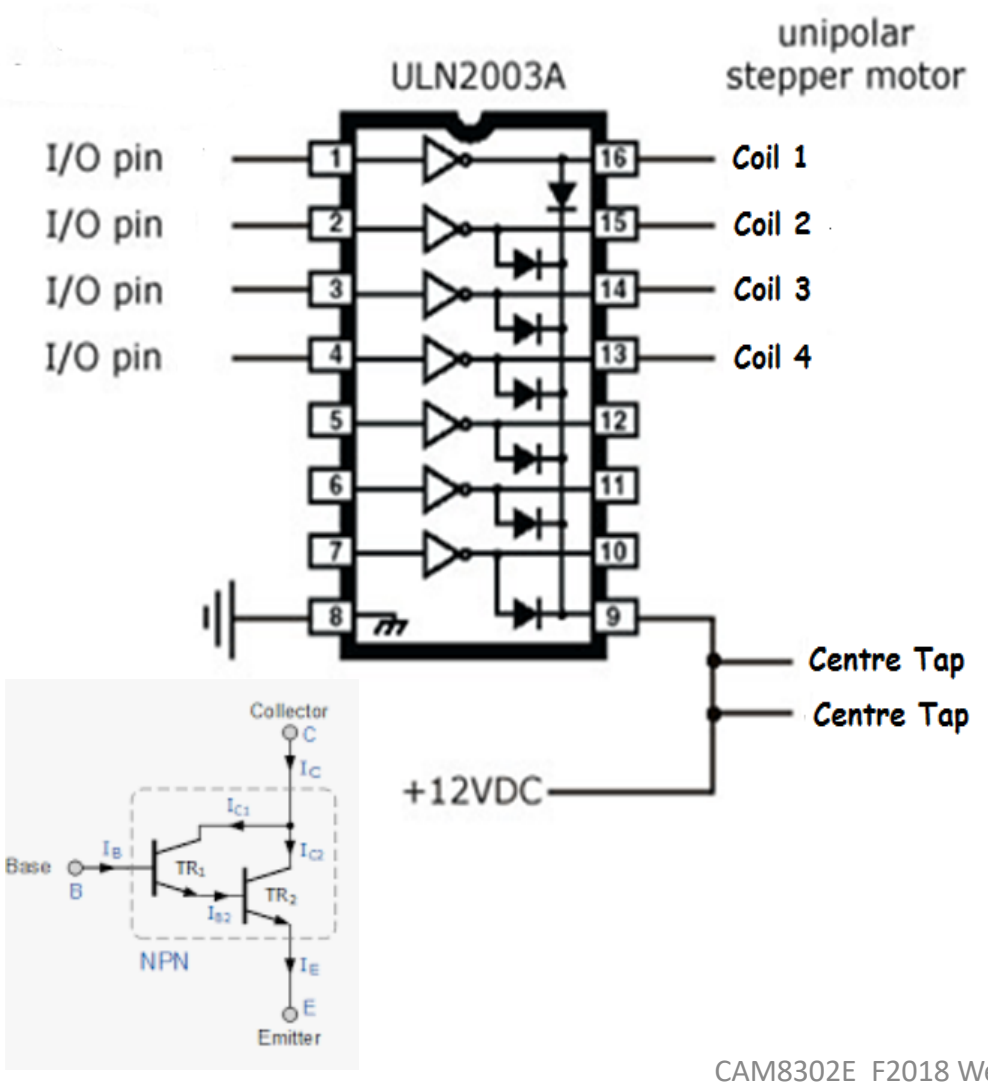
Advantages

- Accurate position control
- Excellent low speed torque
- Long life

Disadvantages

- Low efficiency
- Prone to resonances, noise, and torque ripple
- Cannot accelerate loads rapidly

Unipolar Stepper Motor Driver



ULN 2003A Driver IC

This IC drives inductive loads such as DC motors, solenoids and relays.

The output of the circuit includes suppression diodes to prevent damage to the electronics. The device that appears as an inverter is actually a Darlington Pair transistor drive circuit. When the input is a logic high the output is driven to 0 volts. The Darlington Pair transistor has a much greater Beta than a single NPN transistor.

The input is typically less than 1 mA. Output can handle at least 100 mA per channel.

ULN2003A Driver

1 Features

- 500-mA-Rated Collector Current (Single Output)
- High-Voltage Outputs: 50 V
- Output Clamp Diodes
- Inputs Compatible With Various Types of Logic
- Relay-Driver Applications

2 Applications

- Relay Drivers
- Stepper and DC Brushed Motor Drivers
- Lamp Drivers
- Display Drivers (LED and Gas Discharge)
- Line Drivers
- Logic Buffers

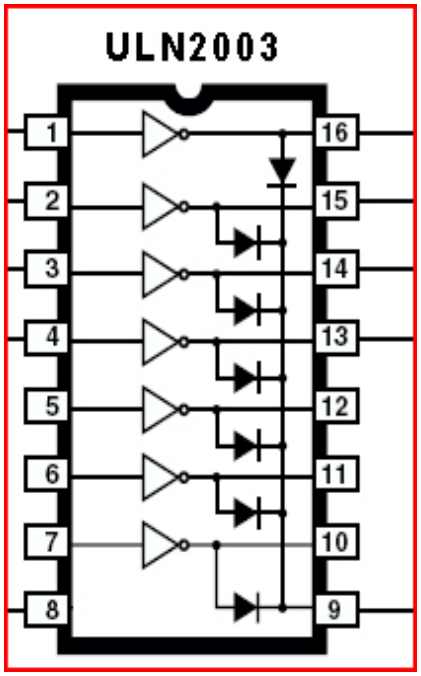
3 Description

The ULx200xA devices are high-voltage, high-current Darlington transistor arrays. Each consists of seven NPN Darlington pairs that feature high-voltage outputs with common-cathode clamp diodes for switching inductive loads.

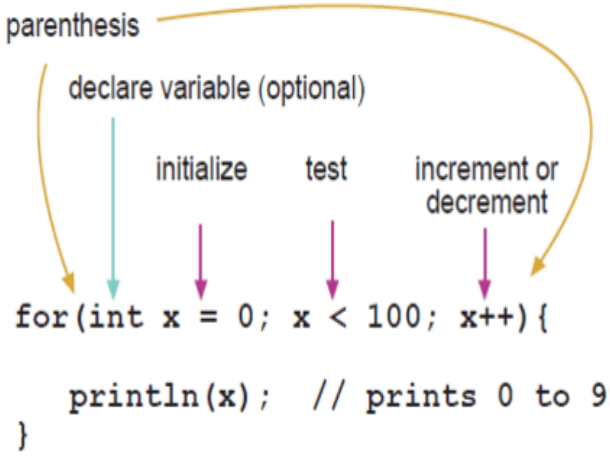
The collector-current rating of a single Darlington pair is 500 mA. The Darlington pairs can be paralleled for higher current capability. Applications include relay drivers, hammer drivers, lamp drivers, display drivers (LED and gas discharge), line drivers, and logic buffers. For 100-V (otherwise interchangeable) versions of the ULx2003A devices, see the SLRS023 data sheet for the SN75468 and SN75469 devices.

The ULN2002A device is designed specifically for use with 14-V to 25-V PMOS devices. Each input of this device has a Zener diode and resistor in series to control the input current to a safe limit. The ULx2003A devices have a 2.7-kΩ series base resistor for each Darlington pair for operation directly with TTL or 5-V CMOS devices.

The ULx2004A devices have a 10.5-kΩ series base resistor to allow operation directly from CMOS devices that use supply voltages of 6 V to 15 V. The required input current of the ULx2004A device is below that of the ULx2003A devices, and the required voltage is less than that required by the ULN2002A device.



For Loop – Used in Stepper Program

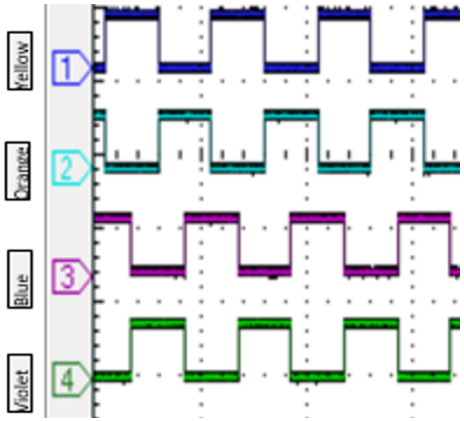


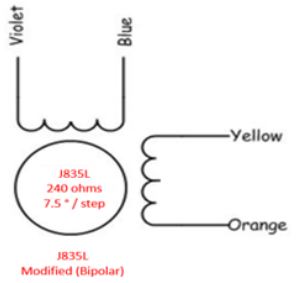

```
19  for (unsigned int x = 0; x < 12; x++)
20
21  {
22    digitalWrite(orange, HIGH);
23    digitalWrite(blue, LOW);
24    digitalWrite(yellow,LOW);
25    digitalWrite(violet, HIGH);
26    delay(delayTime);
```

The **initialization** happens first and exactly once. Each time through the loop, the **condition** is tested; if it's true, the statement block, and the **increment** is executed, then the **condition** is tested again. When the **condition** becomes false, the loop ends.

Arduino Control of a Unipolar Motor in Full Step

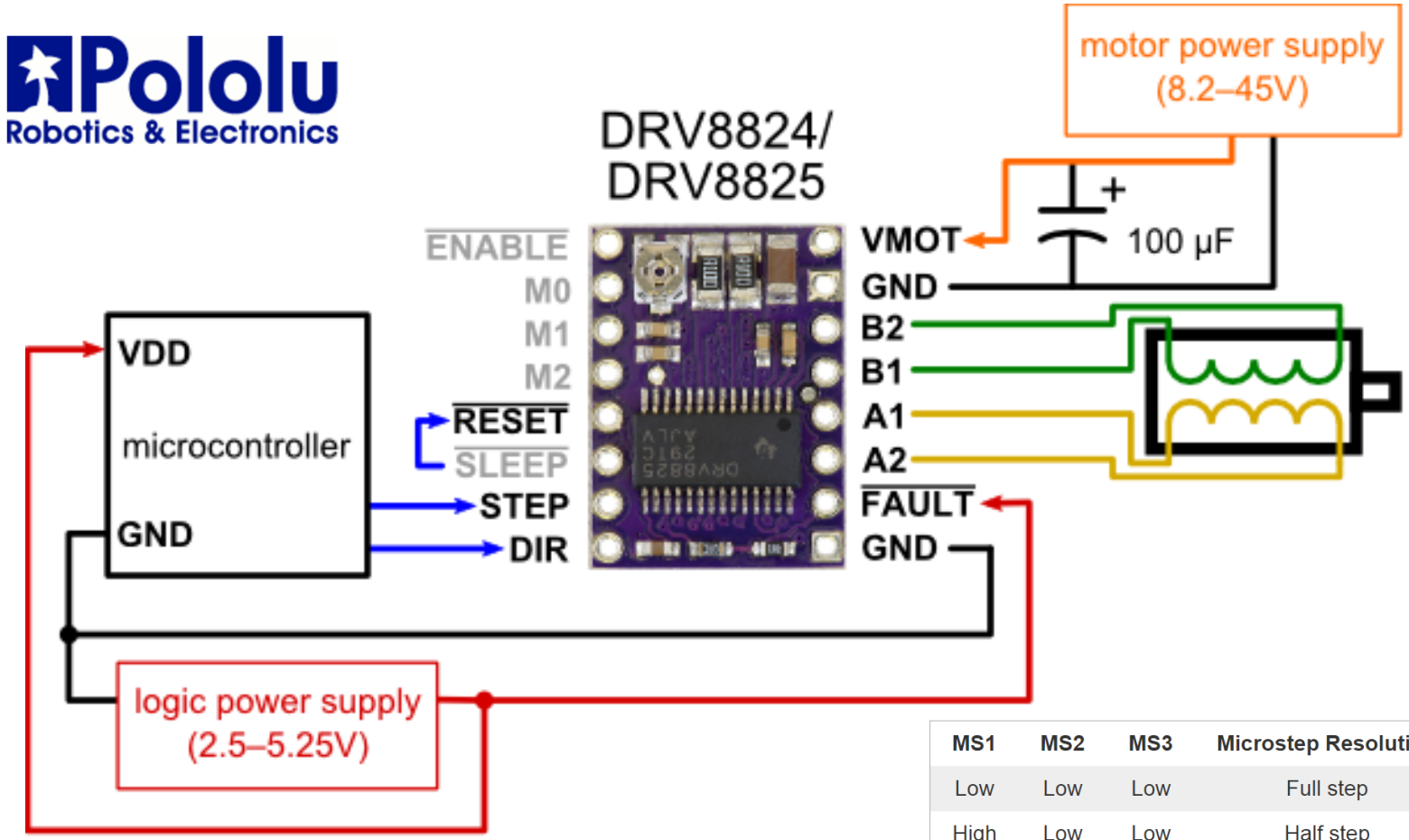
```
/* Stepper Control Unipolar Full Step */

int orange = 8;
int blue = 9;
int yellow = 10;
int violet = 11;

int delayTime = 20;    // milliseconds

void setup() {
  pinMode(orange, OUTPUT);
  pinMode(blue, OUTPUT);
  pinMode(yellow, OUTPUT);
  pinMode(violet, OUTPUT);
}
```





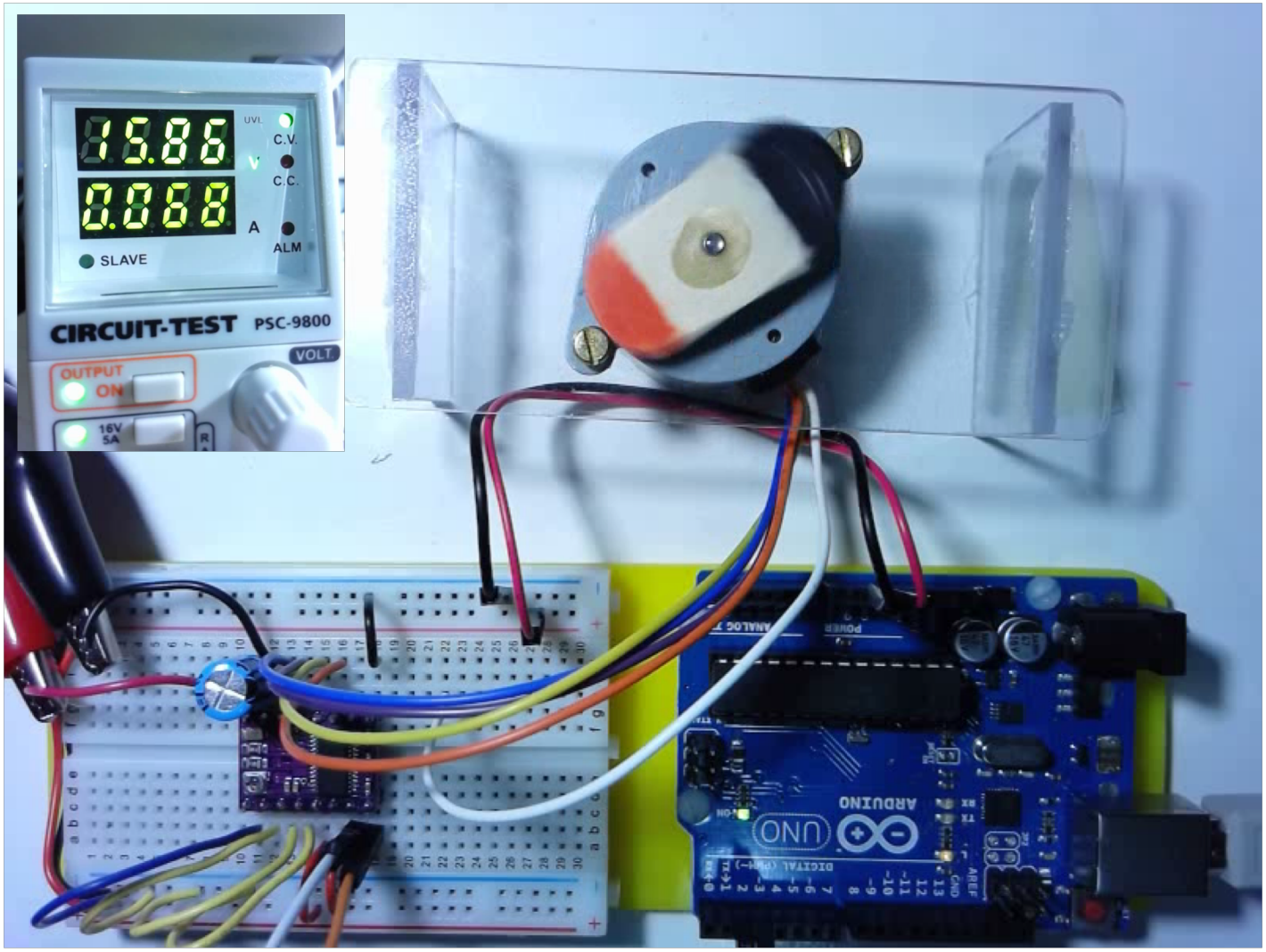

```
 for (unsigned int x = 0; x < 12; x++)

{
  digitalWrite(orange, HIGH);
  digitalWrite(blue, LOW);
  digitalWrite(yellow,LOW);
  digitalWrite(violet, HIGH);
  delay(delayTime);

  digitalWrite(orange, LOW);
  digitalWrite(blue, LOW);
  digitalWrite(yellow, HIGH);
  digitalWrite(violet, HIGH);
  delay(delayTime);

  digitalWrite(orange, LOW);
  digitalWrite(blue, HIGH);
  digitalWrite(yellow, HIGH);
  digitalWrite(violet, LOW);
  delay(delayTime);

  digitalWrite(orange, HIGH);
  digitalWrite(blue, HIGH);
  digitalWrite(yellow, LOW);
  digitalWrite(violet, LOW);
  delay(delayTime);

}
delay(1000);
}
```

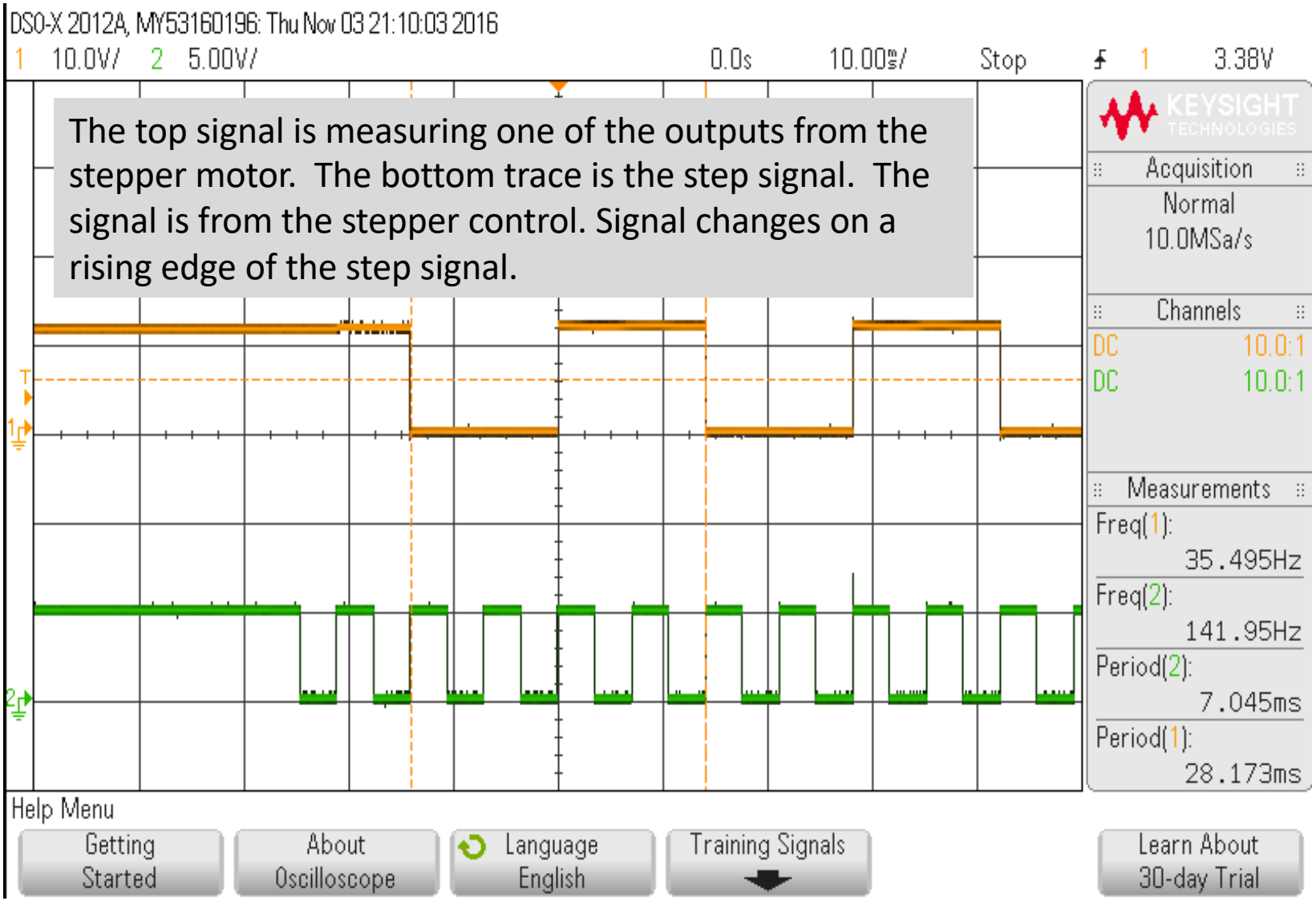
Bipolar DC Stepper Motor Controller



MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step

Bipolar DC Stepper Motor Controller (video)

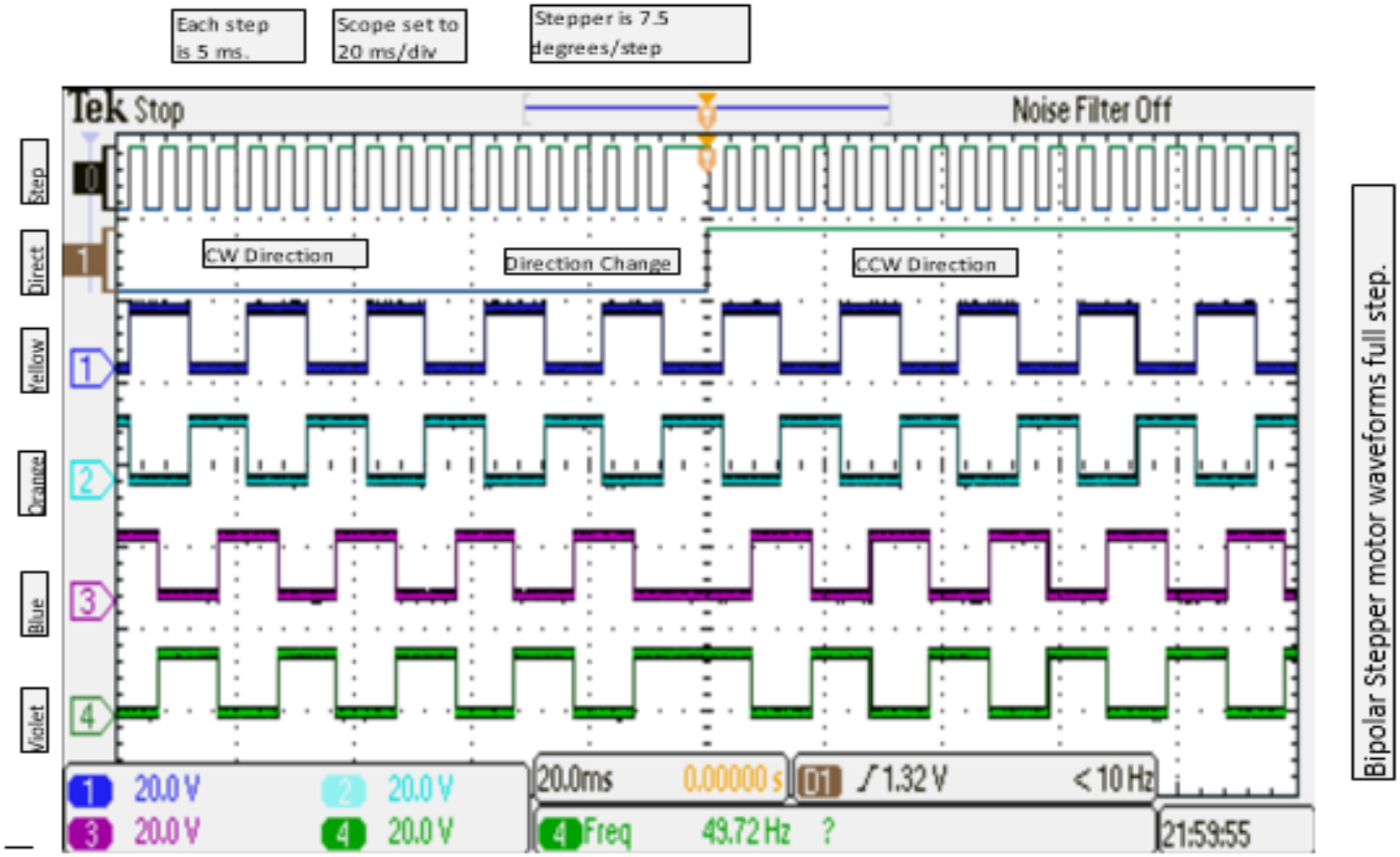



DSO-X 2012A, MY53160196: Thu Nov 03 21:10:03 2016
1 10.0V/ 2 5.00V/
0.0s
10.00ms/
Stop
1 3.38V
The top signal is measuring one of the outputs from the stepper motor. The bottom trace is the step signal. The signal is from the stepper control. Signal changes on a rising edge of the step signal.
KEYSIGHT TECHNOLOGIES
Acquisition
Normal
10.0MSa/s
Channels
DC 10.0:1
DC 10.0:1
Measurements
Freq(1):
35.495Hz
Freq(2):
141.95Hz
Period(2):
7.045ms
Period(1):
28.173ms
Help Menu
Getting Started
About Oscilloscope
Language English
Training Signals
Learn About 30-day Trial

Stepper_FWD_REV_DRV8825_F2016

```
/* Stepper Motor Control using a
 *  POlolu DRV8825 Controller
 *  October 31st 2016
 *  Michel Hanbury from CAM8302E
 */
void setup() {
  pinMode(2, OUTPUT);  // step pulse
  pinMode(3, OUTPUT);  // direction
 }

void loop()
{
   digitalWrite(3, LOW); //direction

for (int x=0; x < 200; x++)
{
   digitalWrite(2, LOW);
   delayMicroseconds(400);
   delay(2);

   digitalWrite(2, HIGH);
   delayMicroseconds(400);
   delay(2);
}
delay(1000);

   digitalWrite(3, LOW); //direction

for (int x=0; x < 200; x++)
{
   digitalWrite(2, LOW);
   delayMicroseconds(400);
   delay(2);

   digitalWrite(2, HIGH);
   delayMicroseconds(400);
   delay(2);
}
delay(1000);
}
```

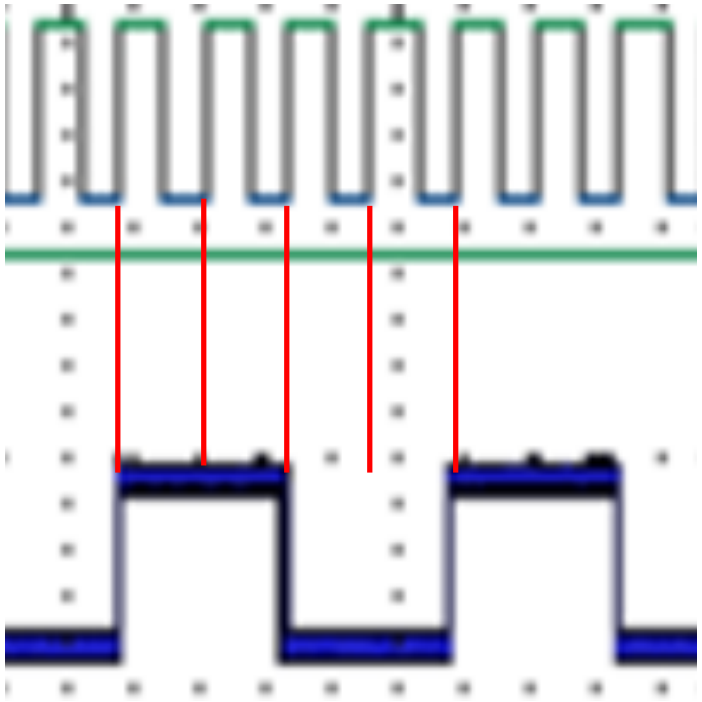
Stepper Full Step Sequence Unipolar Stepper



Clockwise

Counter Clockwise

Stepper CW and CCW Direction

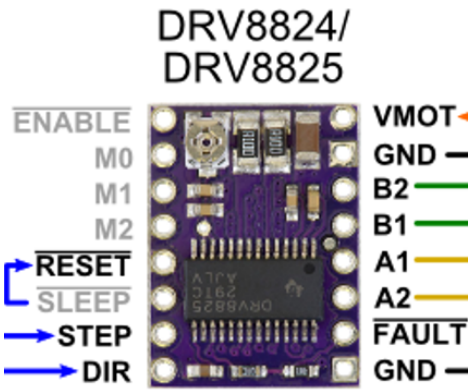


4 steps * 7.5 deg./step = 30 degrees.

Controller clock signal compared to coil signal.

The output changes on the rising edge of the clock.

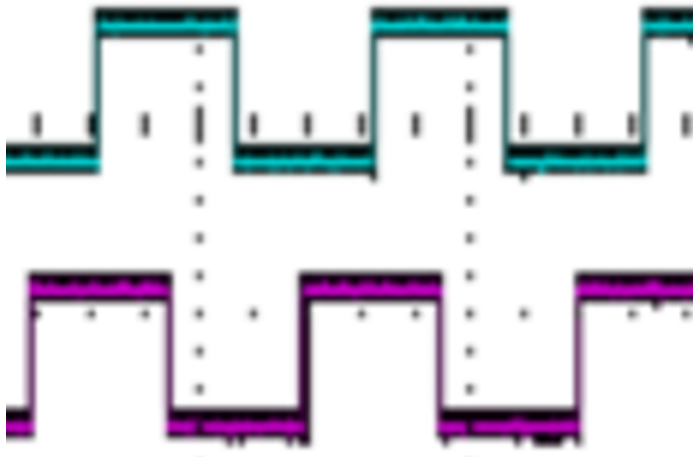
There are 4 clock periods for 4 steps. One high and one low on the coil represents four steps.



Stepper CW and CCW Direction

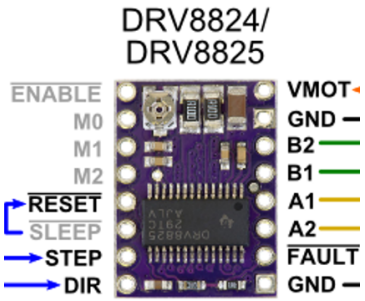
CW direction: (direction = 0)

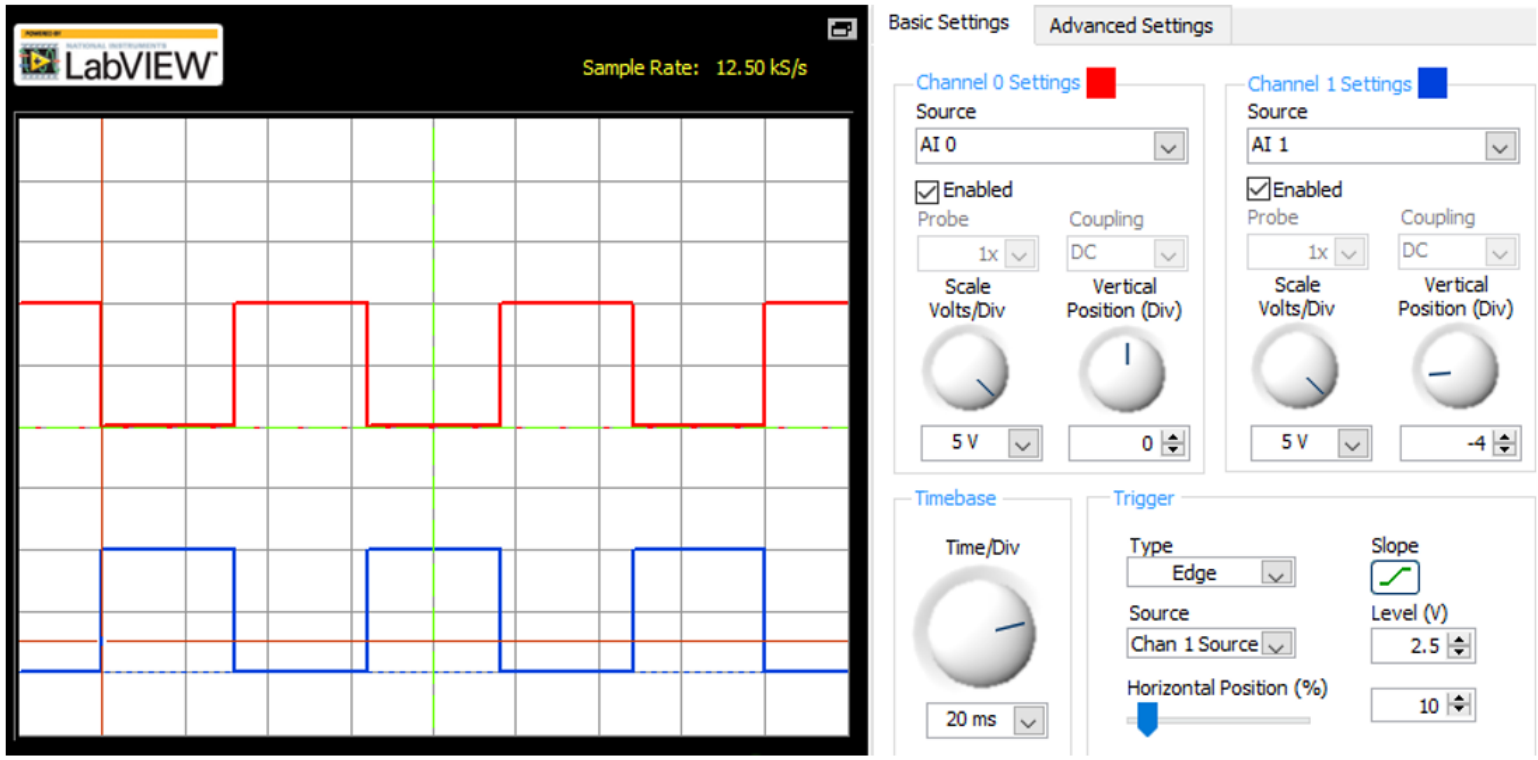
Lower signal rising edge leads the rising edge of the upper signal by 90 degrees.



CCW direction: (direction = 1)

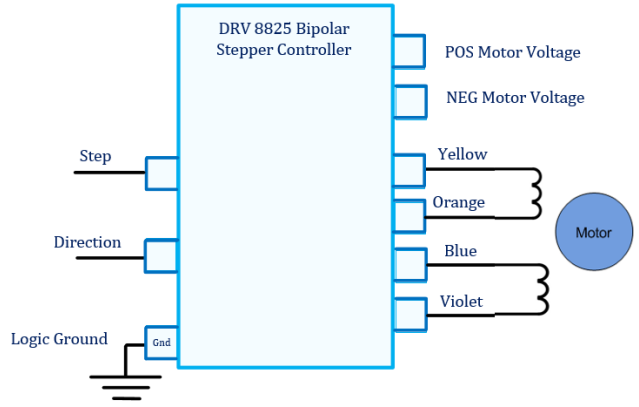
Top signal rising edge leads the rising edge of the lower signal by 90 degrees.

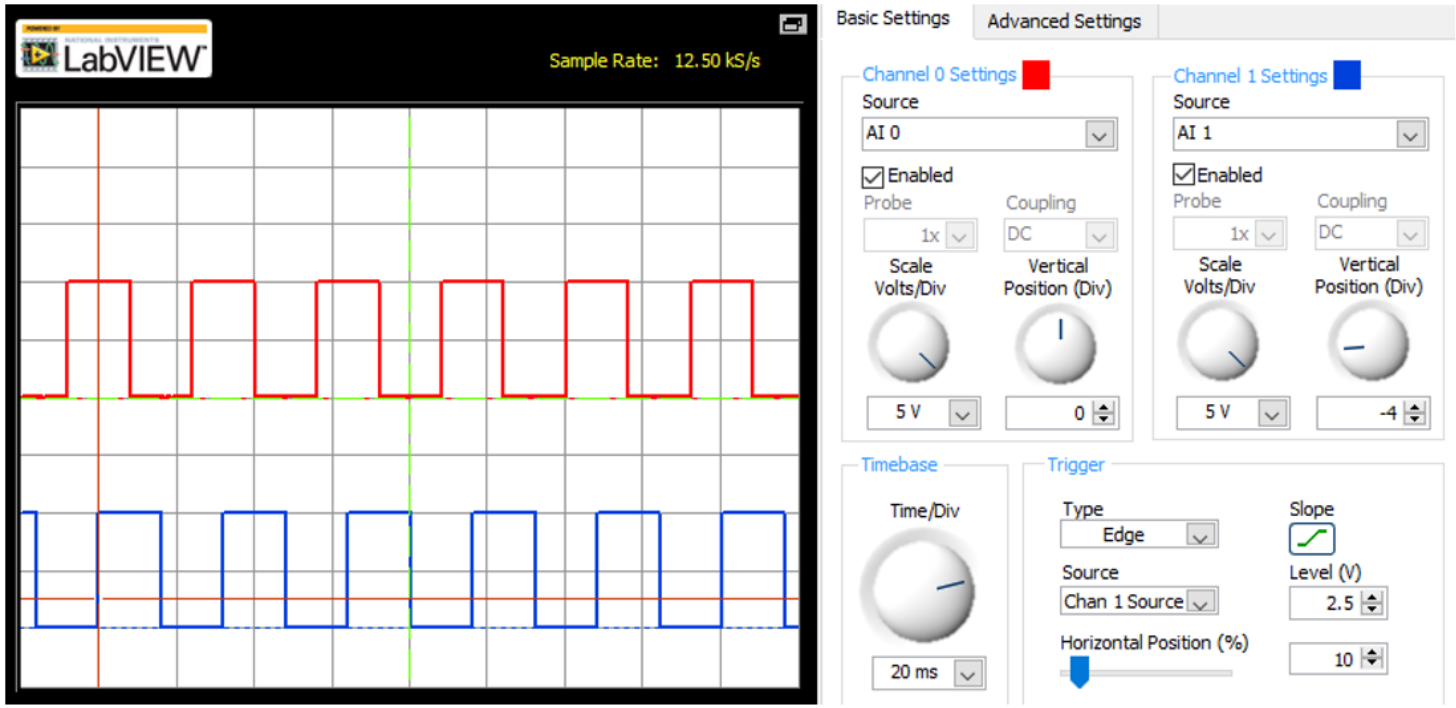




The two traces above are measuring the signal on the driver outputs connected to the yellow wire and the orange wire.

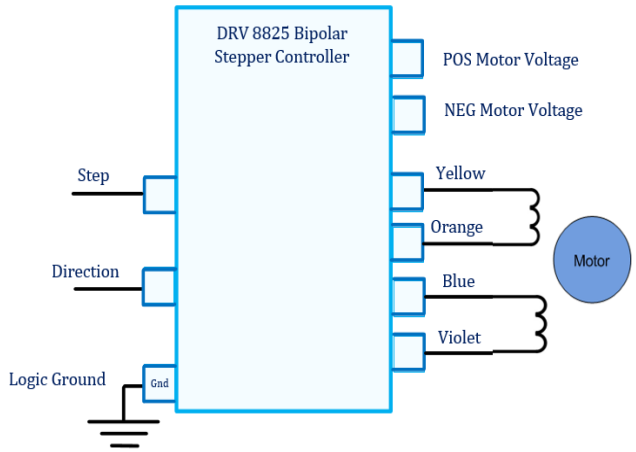
The direction of current flow in the coil changes on every two steps.



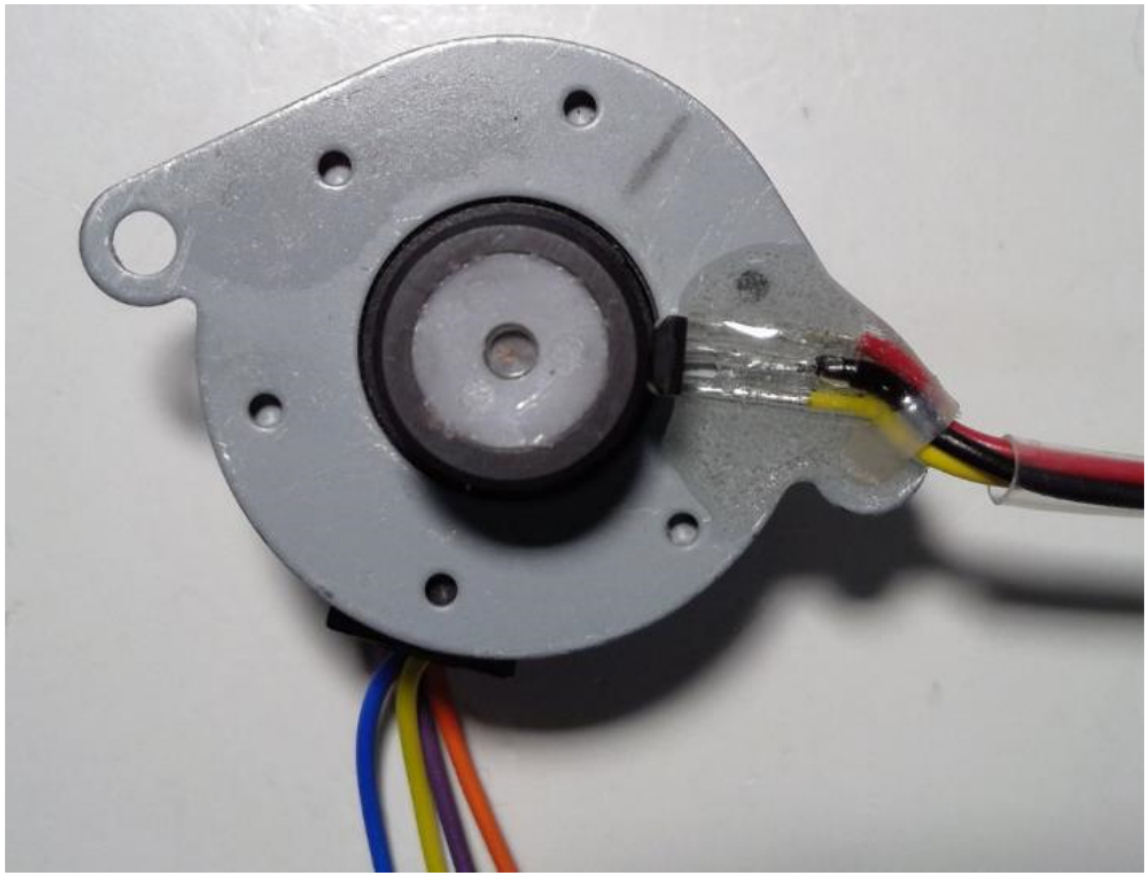


The two traces above are measuring the signal on the driver outputs connected to the yellow wire and the blue wire.

The two signals are 90 degrees out of phase. When the motor is turning in one direction the signal is leading by 90 degrees, in the opposite direction the signal is lagging by 90 degrees to the other coil.



Modified Stepper motor with Hall Effect Sensor and magnet.



240 ohms / coils

7.5 degrees / step

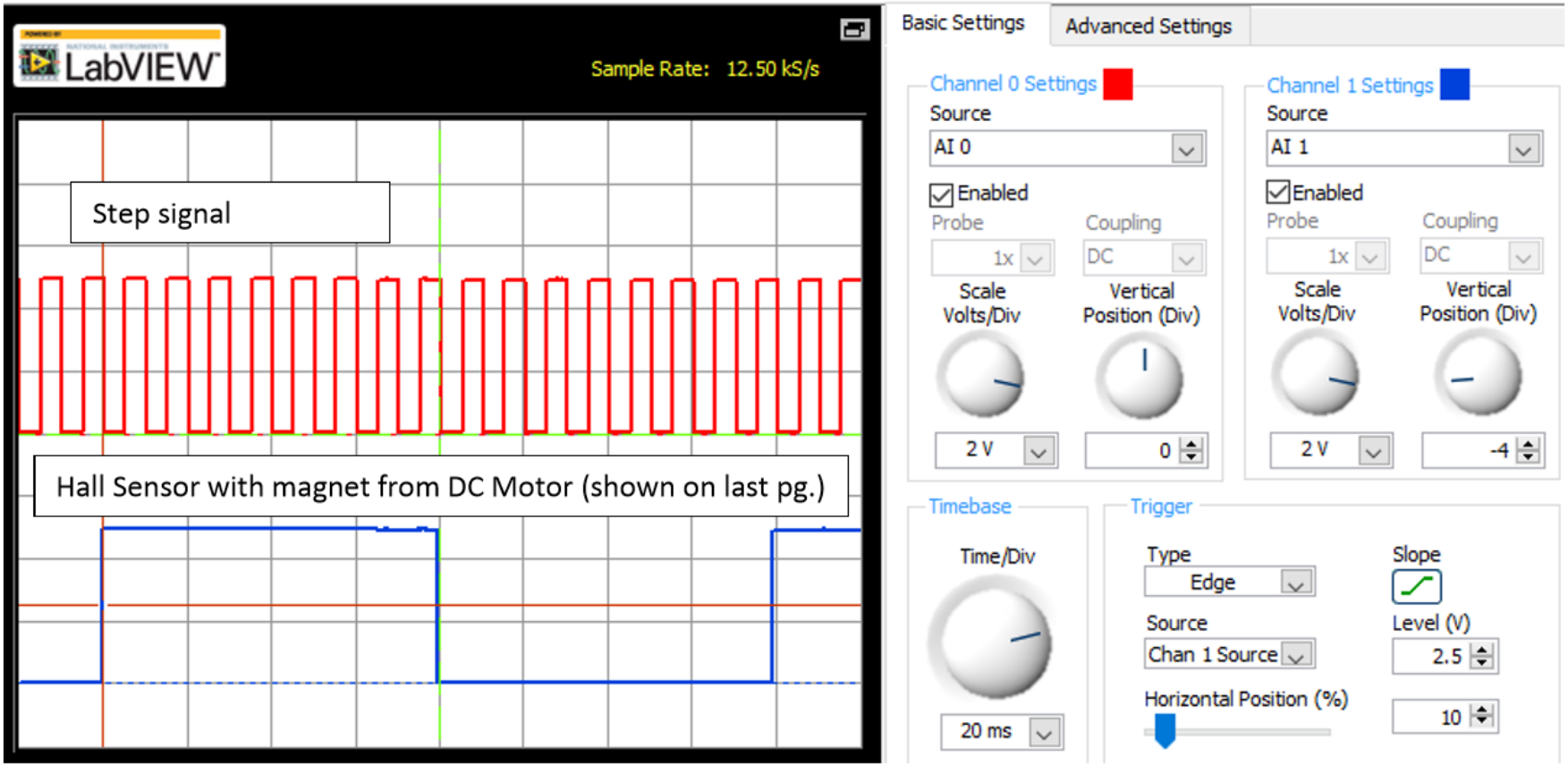
Bi-polar

A Hall Effect sensor is attached near the magnet.

The magnet is the same one used on the DC motor with three NS magnets that will generate 3 pulses per revolution.

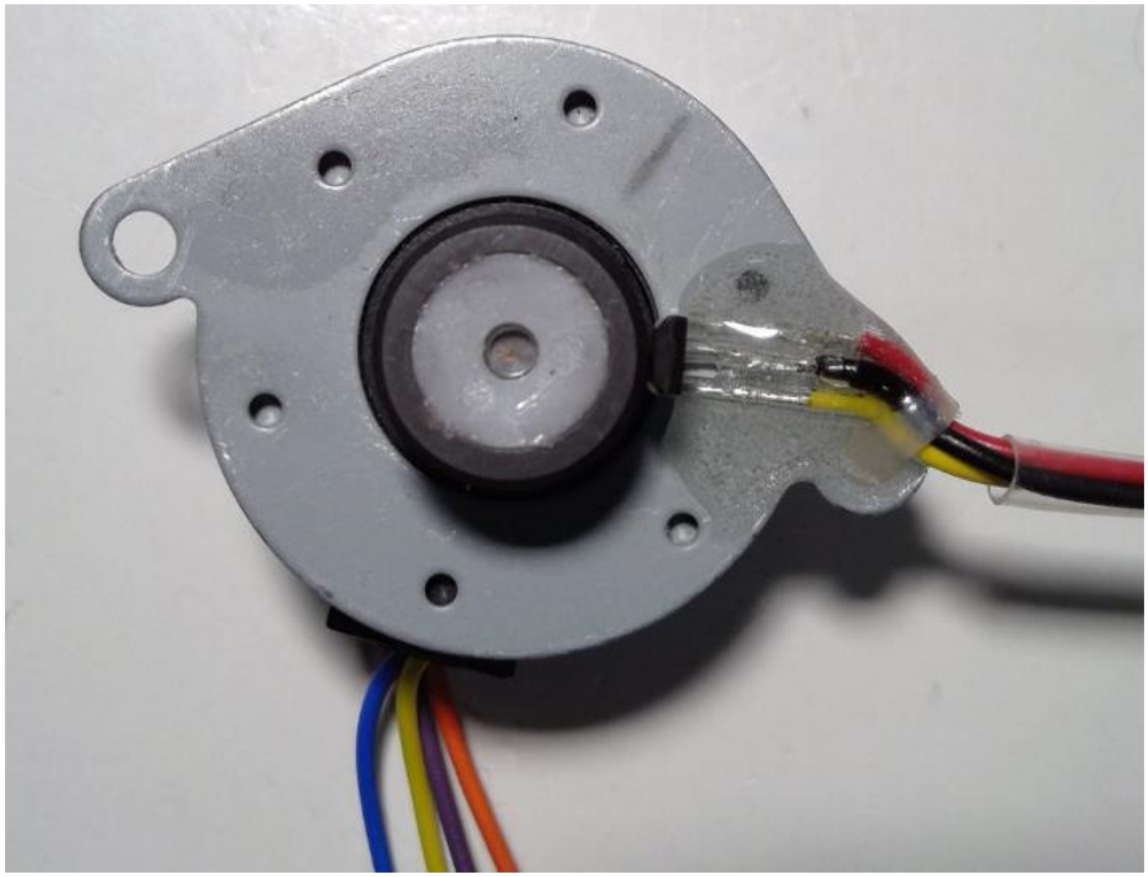
The Hall sensor requires a 5 volt supply and a 10 k pull up resistor attached to the open collector output of the Hall sensor.

The Hall Effect sensor, and the magnet were added to measure the RPMs of the stepper motor. The magnet has three N/S poles that generate three pulses per revolution.



The motor takes 480 ms. per revolution, 1/480 ms. = 2.08 revolutions per second or about 120 RPM.

Modified Stepper motor with Hall Effect Sensor and magnet.



240 ohms / coils

7.5 degrees / step

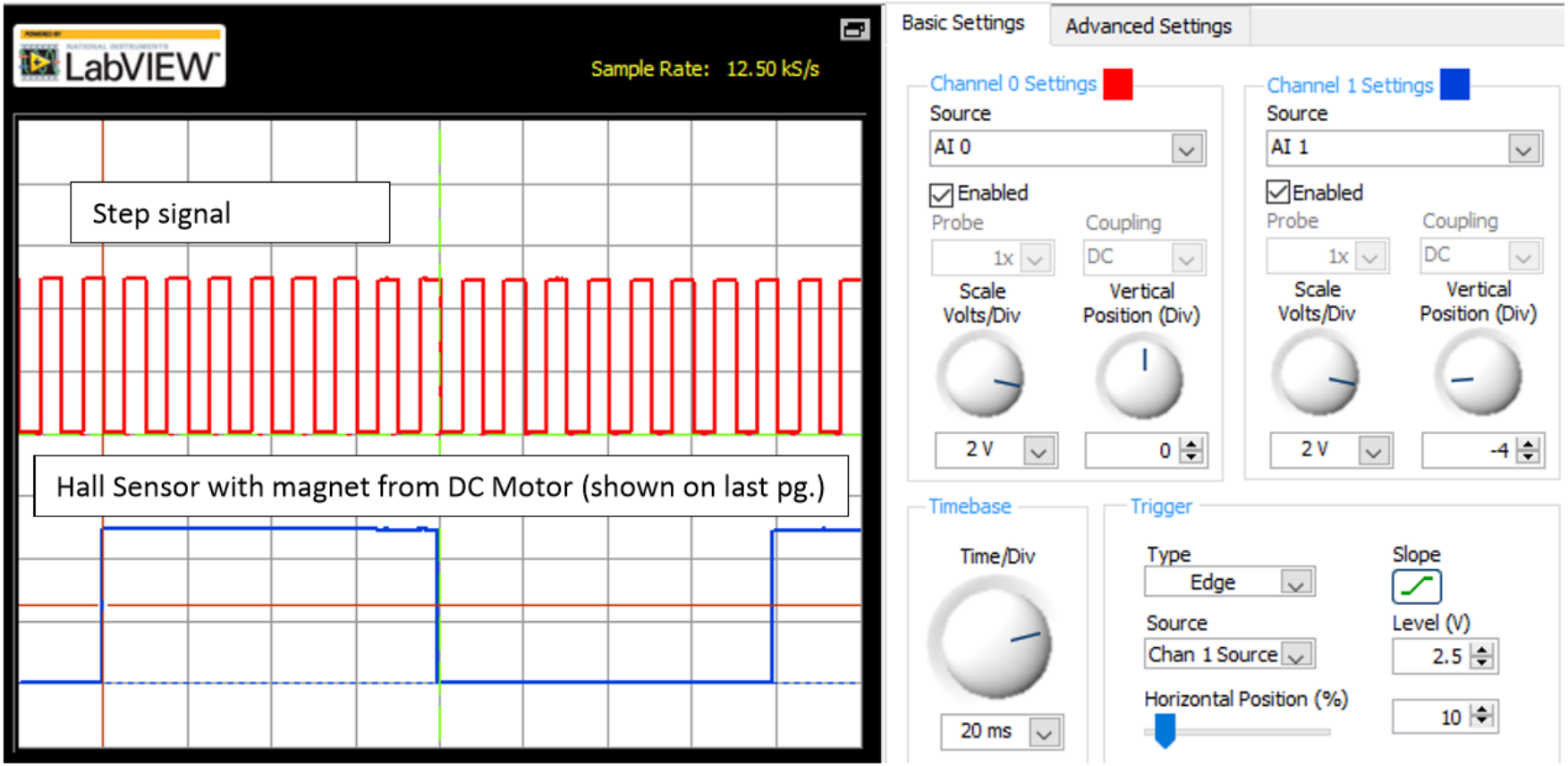
Bi-polar

A Hall Effect sensor is attached near the magnet.

The magnet is the same one used on the DC motor with three NS magnets that will generate 3 pulses per revolution.

The Hall sensor requires a 5 volt supply and a 10 k pull up resistor attached to the open collector output of the Hall sensor.

The Hall Effect sensor, and the magnet were added to measure the RPMs of the stepper motor. The magnet has three N/S poles that generate three pulses per revolution.



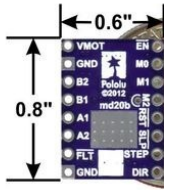
The motor takes 480 ms. per revolution, 1/480 ms. = 2.08 revolutions per second or about 120 RPM.

Micro stepping improves step resolution and allows the stepper motor to turn smoother than in full step. Micro stepping improves the motor also torque. The current through the coil changes using micro stepping.

Overview

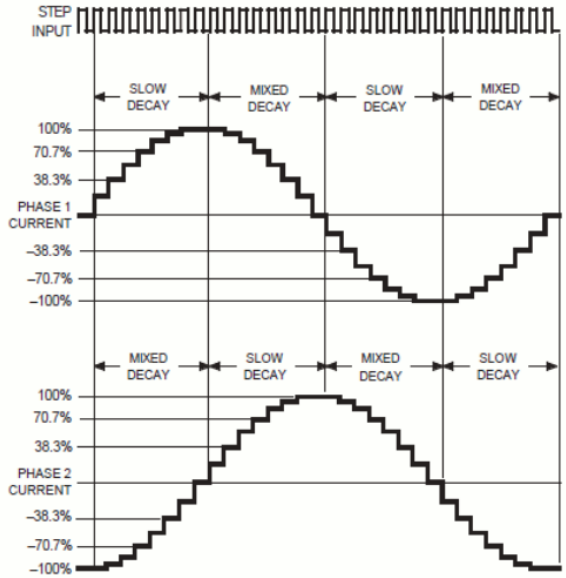
This product is a carrier board or breakout board for TI's DRV8825 stepper motor driver; we therefore recommend careful reading of the DRV8825 datasheet (1MB pdf) before using this product. This stepper motor driver lets you control one bipolar stepper motor at up to 2.2 A output current per coil (see the *Power Dissipation Considerations* section below for more information). Here are some of the driver's key features:

- Simple step and direction control interface
- Six different step resolutions: full-step, half-step, 1/4-step, 1/8-step, 1/16-step, and 1/32-step
- Adjustable current control lets you set the maximum current output with a potentiometer, which lets you use voltages above your stepper motor's rated voltage to achieve higher step rates
- Intelligent chopping control that automatically selects the correct current decay mode (fast decay or slow decay)
- 45 V maximum supply voltage
- Built-in regulator (no external logic voltage supply needed)
- Can interface directly with 3.3 V and 5 V systems
- Over-temperature thermal shutdown, over-current shutdown, and under-voltage lockout



DRV8824/DRV8825 s
driver carrier with

MODE0	MODE1	MODE2	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	1/4 step
High	High	Low	1/8 step
Low	Low	High	1/16 step
High	Low	High	1/32 step
Low	High	High	1/32 step
High	High	High	1/32 step



Assume Mode0 =H and Mode 2 = H.

This places the motor in 1/32 step micro step mode.

The motor will rotate at a step angle of (Motor step angle/32)

For example a 1.8 degree/step motor will have a step angle of 1.8 deg/32 = 0.056 degrees/step. (200 x 32 x 0.056) = 360 degrees.

To rotate the motor at 60 RPM or 1 REV/SEC a step time of 1 second/(200 * 32) = 156 microseconds.

MODE0	MODE1	MODE2	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	1/4 step
High	High	Low	1/8 step
Low	Low	High	1/16 step
High	Low	High	1/32 step
Low	High	High	1/32 step
High	High	High	1/32 step

Interrupts vs Polling

```
33  reading = digitalRead(inPin);
34
35
36  if (reading == HIGH)
37  {
38  digitalWrite(led, HIGH);   // turn the LED on (HIGH is the voltage level)
39    }
40  else
41  {
42   digitalWrite(led, LOW);    // turn the LED off by making the voltage LOW
43   }
```

The section of code above uses input polling to test the state of a mechanical switch. If the switch is a logic high an LED is turned on otherwise it is turned off. This method is not efficient and most of the loop time is consumed in testing the switch. If the loop if long then if may take time before the switch is tested and an event may be missed.

Interrupts Setup

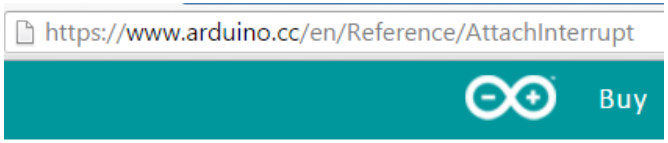
```
21 void setup() {
22
23   attachInterrupt(0, slow, RISING); // on pin 2
24   attachInterrupt(1, fast, RISING); // on pin 3
25
```

```
101 void fast(){                    // Interrupt service routine
102   delayTime = 10;
103 }
104
105 void slow(){                    // Interrupt service routine
106   delayTime = 20;
107 }
```

Lines 21-24 configure the interrupt. The program is telling the Arduino that when a rising edge occurs on pin 2 (interrupt 0) then stop what the program is currently doing, execute the “slow” routine then return to where it was before the interrupt.

Interrupts vs Polling

Reference Language | Libraries | Comparison | Changes



attachInterrupt()

Description

Digital Pins With Interrupts

The first parameter to attachInterrupt is an interrupt number. Normally you should use digitalPinToInterrupt(pin) to translate the actual digital pin to the specific interrupt number. For example, if you connect to pin 3, use digitalPinToInterrupt(3) as the first parameter to attachInterrupt.

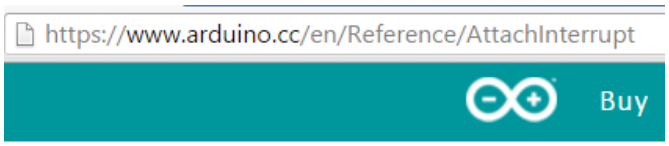
Board	Digital Pins Usable For Interrupts
Uno, Nano, Mini, other 328-based	2, 3
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21

Using Interrupts

Using Interrupts

Interrupts are useful for making things happen automatically in microcontroller programs, and can help solve timing problems. Good tasks for using an interrupt may include reading a rotary encoder, or monitoring user input.

If you wanted to insure that a program always caught the pulses from a rotary encoder, so that it never misses a pulse, it would make it very tricky to write a program to do anything else, because the program would need to constantly poll the sensor lines for the encoder, in order to catch pulses when they occurred. Other sensors have a similar interface dynamic too, such as trying to read a sound sensor that is trying to catch a click, or an infrared slot sensor (photo-interrupter) trying to catch a coin drop. In all of these situations, using an interrupt can free the microcontroller to get some other work done while not missing the input.



Interrupts Syntax

Syntax

attachInterrupt(digitalPinToInterrupt(pin), ISR, mode); (*recommended*)

attachInterrupt(interrupt, ISR, mode); (*not recommended*)

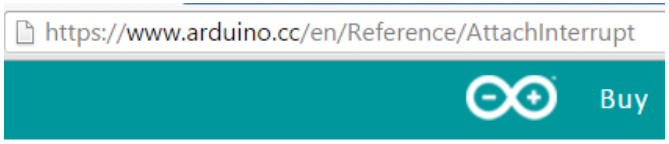
attachInterrupt(pin, ISR, mode) ; (*not recommended Arduino Due, Zero only*)

Parameters

interrupt: the number of the interrupt (*int*)

pin: the pin number (*Arduino Due, Zero only*)

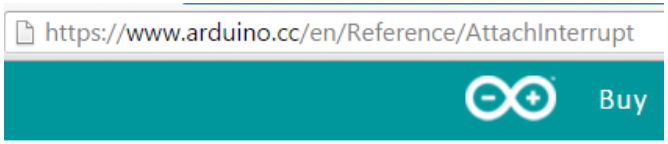
ISR: the ISR to call when the interrupt occurs; this function must take no parameters and return nothing. This function is sometimes referred to as an *interrupt service routine.*



Interrupts Modes

mode:

defines when the interrupt should be triggered. Four contstants are predefined as valid values:



- **LOW** to trigger the interrupt whenever the pin is low,
- **CHANGE** to trigger the interrupt whenever the pin changes value
- **RISING** to trigger when the pin goes from low to high,
- **FALLING** for when the pin goes from high to low.

Interrupt Program for Stepper

```
5  #define direction 4
6  #define step 5
7  #define steptime1 20   // micro seconds  10.4 high, 10.4 low
8  #define steptime2 0    // milli seconds  period = 20.8ms * 48 = 1 socond/rev.
9
10 void setup() {
11   pinMode(step, OUTPUT);  // step pulse
12   pinMode(direction, OUTPUT);  // direction
13   attachInterrupt(digitalPinToInterrupt(2),direction1, FALLING); // from optical
14   attachInterrupt(digitalPinToInterrupt(3),direction2, RISING); // from push button
15   digitalWrite(direction, LOW); //start in one direction
16  }
```

Interrupt Functions

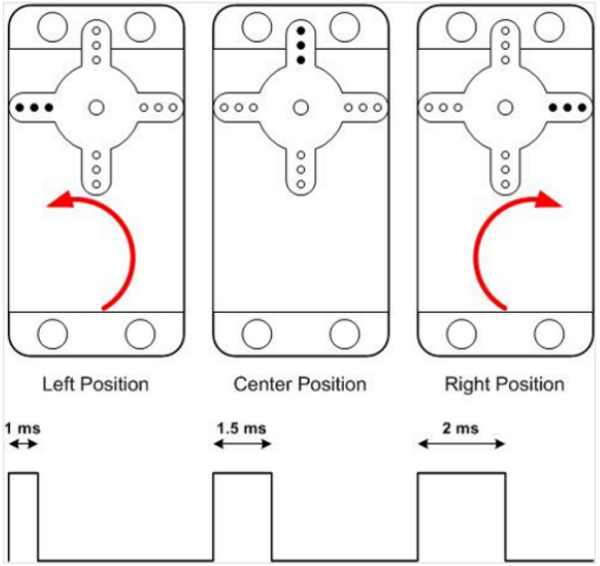
This section of the code are the two functions that are executed when an interrupt occurs.

```
32 void direction1() {
33 digitalWrite(direction, LOW); //direction
34 }
35
36 void direction2() {
37 digitalWrite(direction, HIGH); //direction
38
39 }
```

2nd Interrupt Program for Stepper

```
21 void loop() {
22  tone(4,freq); // 31 to 65,535 Hertz
23  digitalWrite(5,direction_1);
24 }
25
26 void toggle_s() {    // pin 2 interrupt 0 toggle speed
27    digitalWrite(LED, HIGH);
28
29    if (freq == 125)
30      freq = 100;      // change speed if interrupted
31    else if (freq == 100)
32      freq = 125;
33     x++; // inx x
34 }
35 void toggle_d() {    // pin 3 interrupt 1  toggle direction
36
37    detachInterrupt(1);
38    digitalWrite(LED, LOW);
39
40     if (direction_1 == HIGH)
41       direction_1 = LOW;  // change direction if interrupted
42    else if (direction_1 == LOW)
43       direction_1 = HIGH;
44     x--; // dec x
45     attachInterrupt(1, toggle_d, FALLING);
46 }
```

Servo Motor



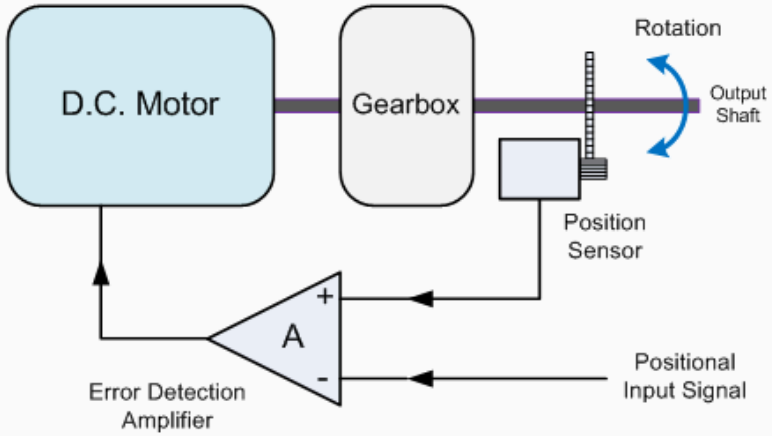
RC Servos has been designed to accept the RC PWM signal which is nothing more than a periodic pulse with a width of anything between 1.0 ms and 2.0 ms. Some systems with more resolution will have allow for pulses in the range of 0.5 ms to 2.5 ms. However, 1 ms to 2 ms is pretty much standard. The idea behind this position protocol is that 1.5 ms commands the servo to go to the center position. A 1.0 ms pulse commands the motor to attempt to reach its leftmost position and 2.0 ms to its rightmost position. Any pulse measuring in between 1.0 ms and 2.0 ms is decoded as a position in between leftmost and rightmost. Since the remote control is analog, practically any position can be attained.

Servo Motor – used in robotics and remote controlled devices.
Position is controlled using PWM signal. The motor has a feedback potentiometer to determine the current position of the output control arm. The servo motor moves CW and CCW with about 180 degrees of movement. The servo has a gear train to give it a greater torque. The variable resistor acts as a feedback signal.

The DC Servo Motor

DC Servo motors are used in closed loop type applications were the position of the output motor shaft is fed back to the motor control circuit. Typical positional "Feedback" devices include Resolvers, Encoders and Potentiometers as used in radio control models such as airplanes and boats etc. A servo motor generally includes a built-in gearbox for speed reduction and is capable of delivering high torques directly. The output shaft of a servo motor does not rotate freely as do the shafts of DC motors because of the gearbox and feedback devices attached.

DC Servo Motor Block Diagram

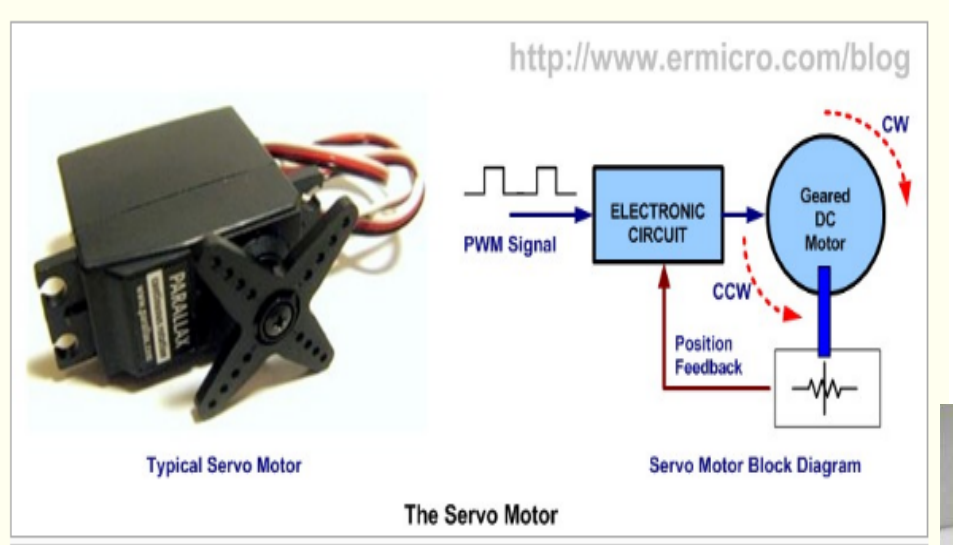


A servo motor consists of a DC motor, reduction gearbox, positional feedback device and some form of error correction. The speed or position is controlled in relation to a positional input signal or reference signal applied to the device. The error detection amplifier looks at this input signal and compares it with the feedback signal from the motors output shaft and determines if the motor output shaft is in an error condition and, if so, the controller makes appropriate corrections either speeding up the motor or slowing it down. This response to the positional feedback device means that the servo motor operates within a "Closed Loop System".

Servo motors are also used in remote control models with most servo motors being able to rotate up to about 180 degrees in both directions making them ideal for accurate angular positioning. However, these RC type servos are unable to continually rotate at high speed like conventional DC motors unless specially modified. A servo motor consist of several devices in one package, motor, gearbox, feedback device and error correction for controlling position, direction or speed. They are controlled using just three wires, Power, Ground and Signal Control.

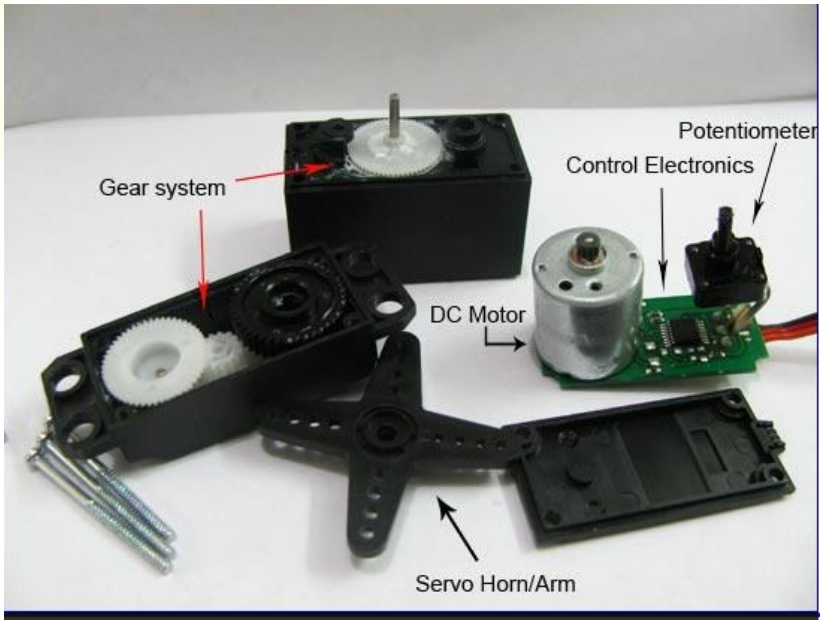
- A servo motor consists of four main sections:
  - DC motor
  - Gear set
  - Potentiometer for feedback.
  - Electronic Circuit

DC Servo Motors



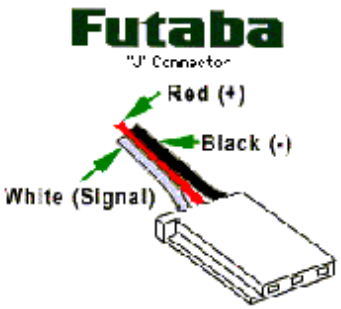
A servo Motor uses a feedback signal to control positioning.

The feedback can be as simple as a variable resistor or as complex as an absolute incremental optical encoder.



Servo Motor Block Diagram (2nd image)

DC Servo Motors



- A servo motor connects to a controller using three wires.
- One of the wires connects to ground a second wire to 4.5 – 6.3 volts DC.
- The 3rd wire is the input signal. The signal originated from a controller producing a PWM output signal.
- The PWM signal has a period of 20 ms. The time high ranges from about 1 to 2 ms.

Incremental versus Absolute Encoder

INCREMENTAL VS. ABSOLUTE ENCODERS CLOSE

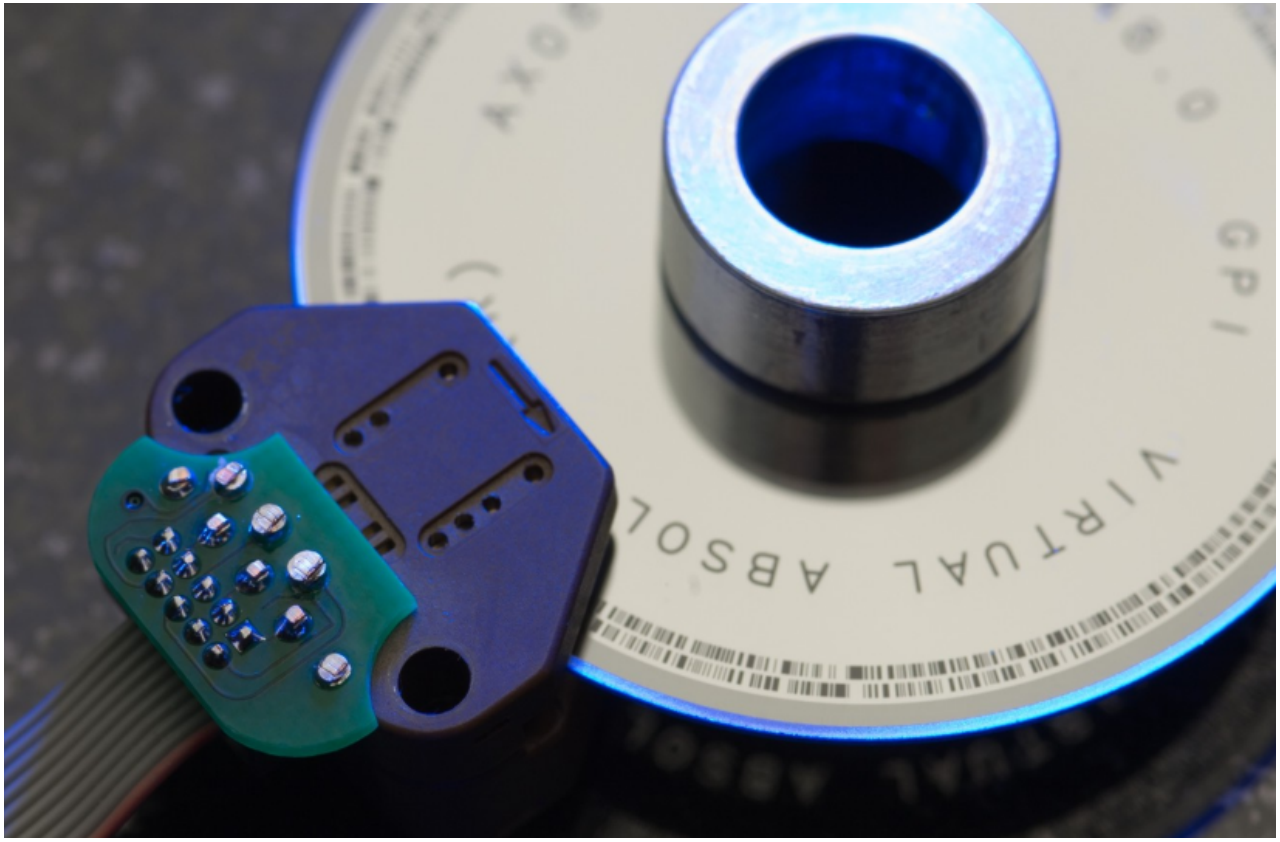
Incremental Encoders

A device that generates electrical signals by means of a rotating disk that passes between a light source and photo detectors. Incremental encoders have two output signals, or channels, commonly referred to as channel A and B. The A and B outputs are nominally 90° out of phase with each other and are interpreted by a motion controller to determine position/velocity information. The lead/lag relationship between the A and B channels provides directional information. It is important to understand that each mechanical position is not uniquely defined. When the incremental encoder is powered on, the position of an incremental encoder is not known, since the output signals are not unique to any singular position. Incremental encoders often provide a third output that pulses once per revolution of the disk. This is typically called the Index, or Z-channel, and is commonly used for homing/reference moves.

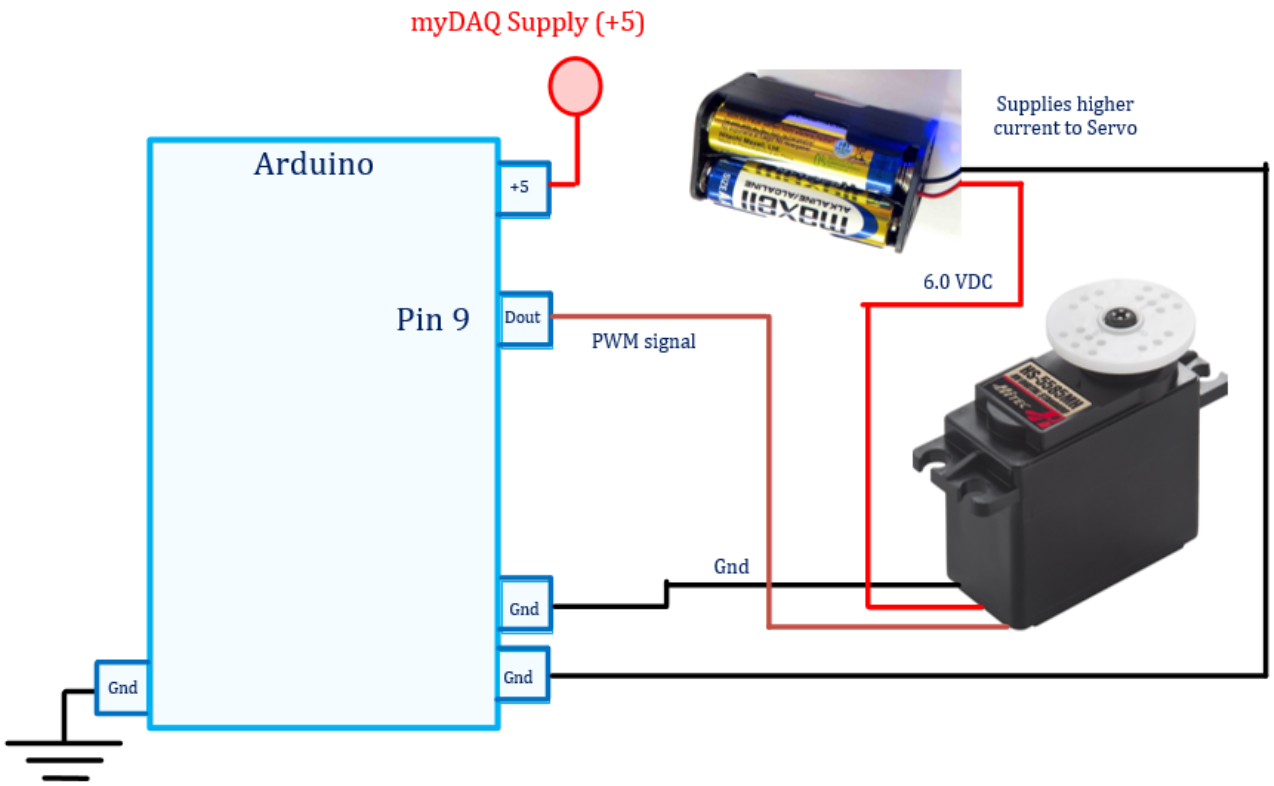
Absolute Encoders

Absolute encoders have a unique value (voltage, binary count, etc.) for each mechanical position. When an absolute encoder is powered on, the position is known. Absolute encoders most commonly provide digital data in a parallel or serial format to the motion controller which is used to determine position/velocity information. Since they provide absolute position information when powered on they eliminate the need for a homing/reference move in a motion system.

Absolute Encoder



The absolute encoder knows the current position on startup of the system. The encoder uses a pattern on the rotating disk and optics to determine position. A quadrature encoder is an incremental encoder.

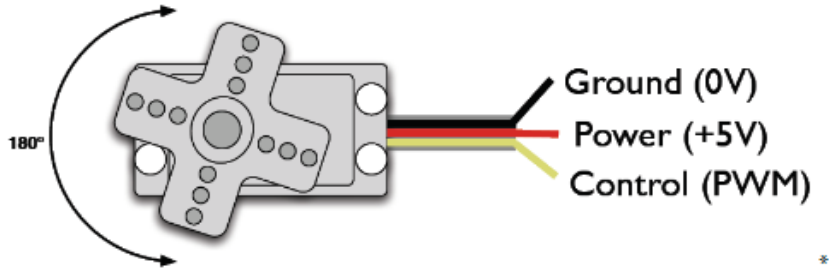


Servo hardware configuration.

Servo Motor Connections to Arduino: Connect the power for the servo to a 4.5 to 6.5 volt battery or DC power supply. The signal controlling the servo motor shaft position is controlled using an Arduino PWM output.

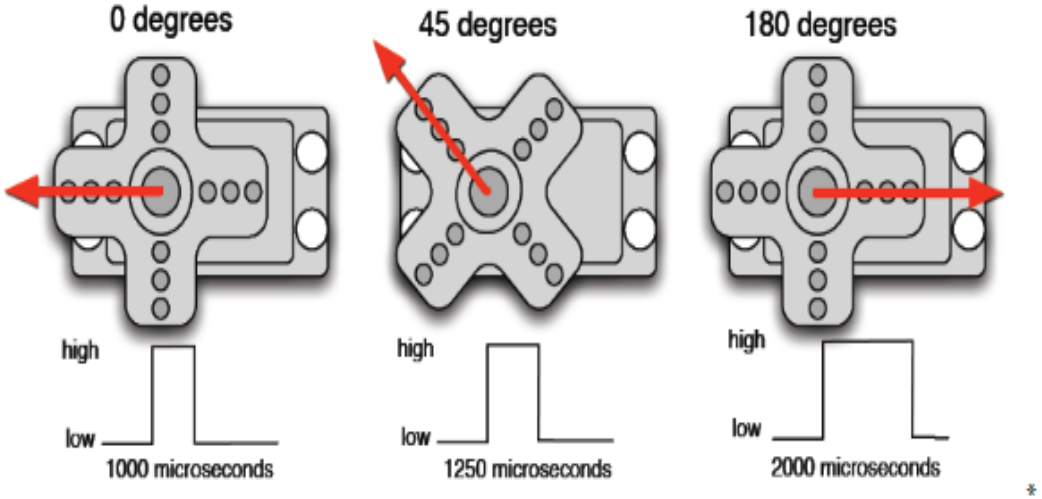
Make the servo movement

Of course you need to connect the servo motor. Now get your servo motor, and you can find three pin hole. They are:



As we know, we are going to send a pulse to make the servo move. The pulse should be ranges from 1 to 2 millisecs.

Servo Motor Wiring

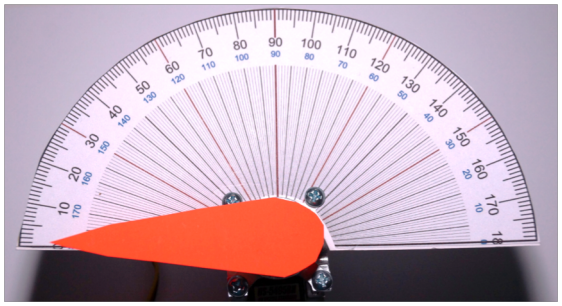


Servo Motor Shaft Position versus signal timing.

servo_motor_test_with_slider_Nov_11_2016

```
// Controlling a servo position using a potentiometer (variable resistor)
// by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>
// Modified by Michel Hanbury
// For CAM8302E Nov. 12, 2016

#include <Servo.h>    // add servo library

Servo myservo;  // create servo object to control a servo

int slider = 0;    // analog pin used to connect the potentiometer
int Analog_In;     // variable to read the value from the analog pin
int Servo_Angle;
void setup()
{
  Serial.begin (9600);   // 9600 bits/second to display variable data
  myservo.attach(9);     // attaches the servo on pin 9 to the servo object
}

void loop()
{

  Analog_In = analogRead(slider);            // reads the value of the potentiometer (value between 0 and 1023)
  Servo_Angle = map(Analog_IN, 0, 1023, 0, 180);      // scale it to use it with the servo (value between 0 and 180)
                                                      // adjusted for min of 0 deg to 180 deg.
  myservo.write(Servo_Angle);                         // sets the servo position according to the scaled value

    Serial.print("Analog In  ");
    Serial.print(Analog_In);
    Serial.print("    Servo Angle   ");
    Serial.println(Servo_Angle);

    delay(15);                          // waits for the servo to get there
}
```

Servo Motor test program.

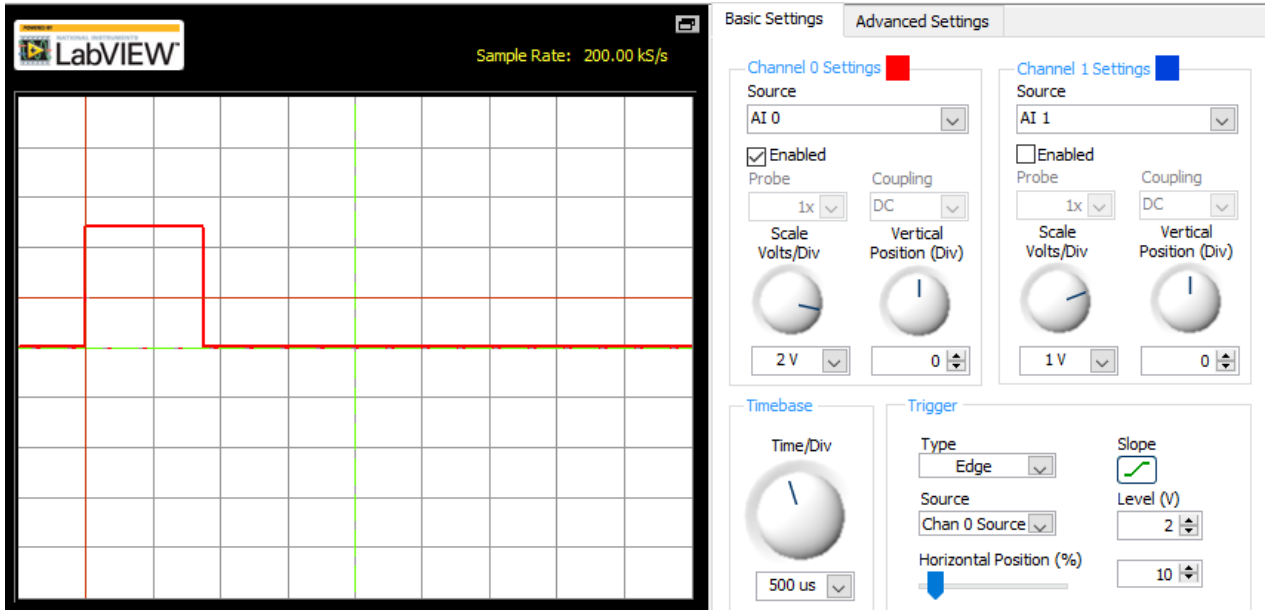


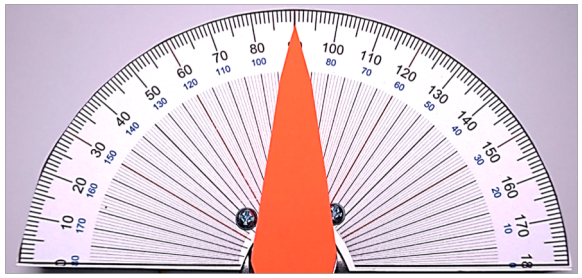
Servo motor fully CCW. There are hard limits to the servo position, do not drive the motor to the maximum or minimum positions.

This screen captures shows a time high of ~0.9 ms.

Program Output

```
Analog In  186    Servo Angle   32
Analog In  186    Servo Angle   32
Analog In  186    Servo Angle   32
```



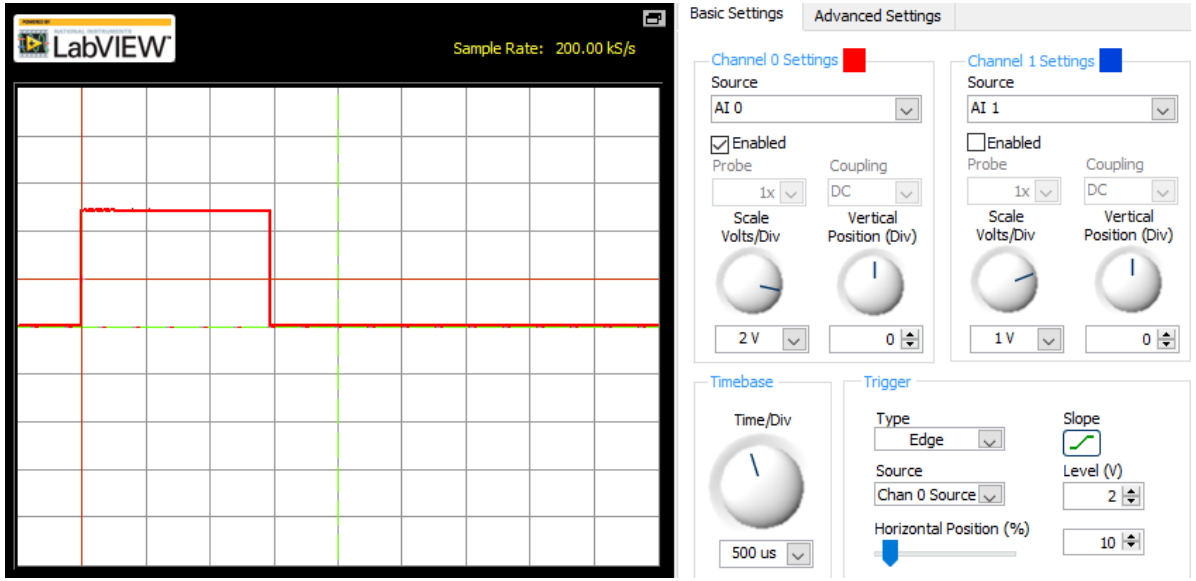


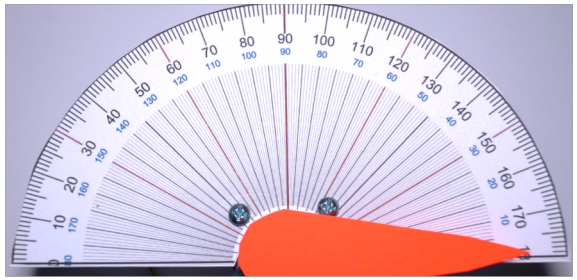
Servo motor at the middle neutral position. This is normally about 1.5 milliseconds.

This screen captures shows a time high of ~1.5 ms.

Program Output

```
Analog In   515     Servo Angle    90
Analog In   515     Servo Angle    90
Analog In   515     Servo Angle    90
```



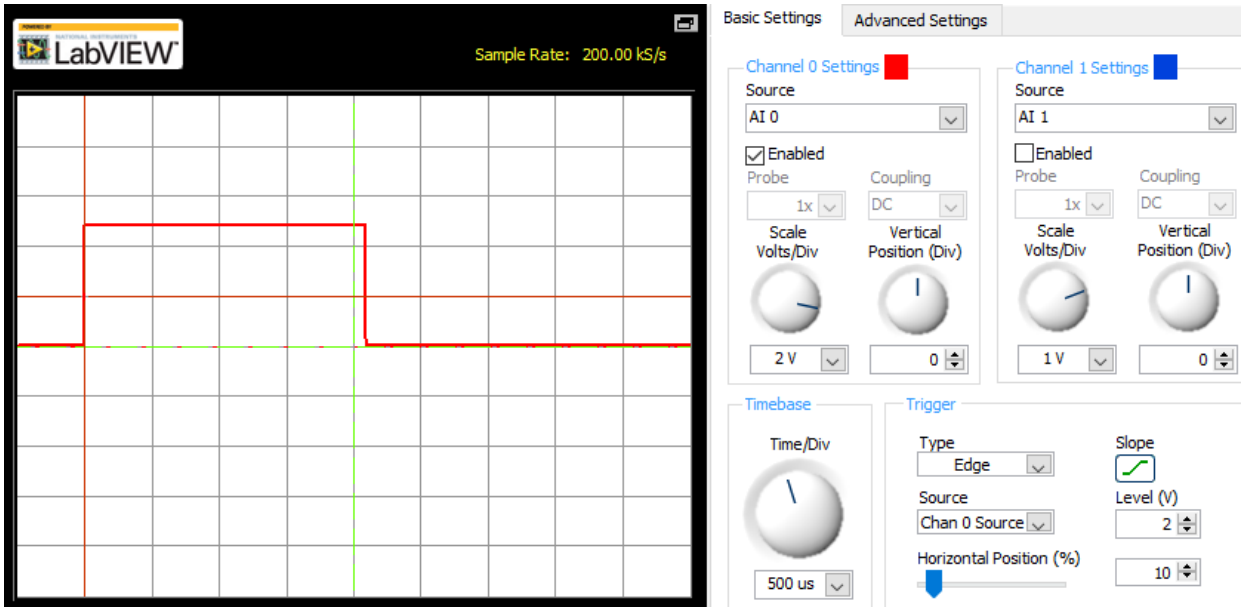


Servo motor fully CW. There are hard limits to the servo position, do not drive the motor to the maximum or minimum positions.

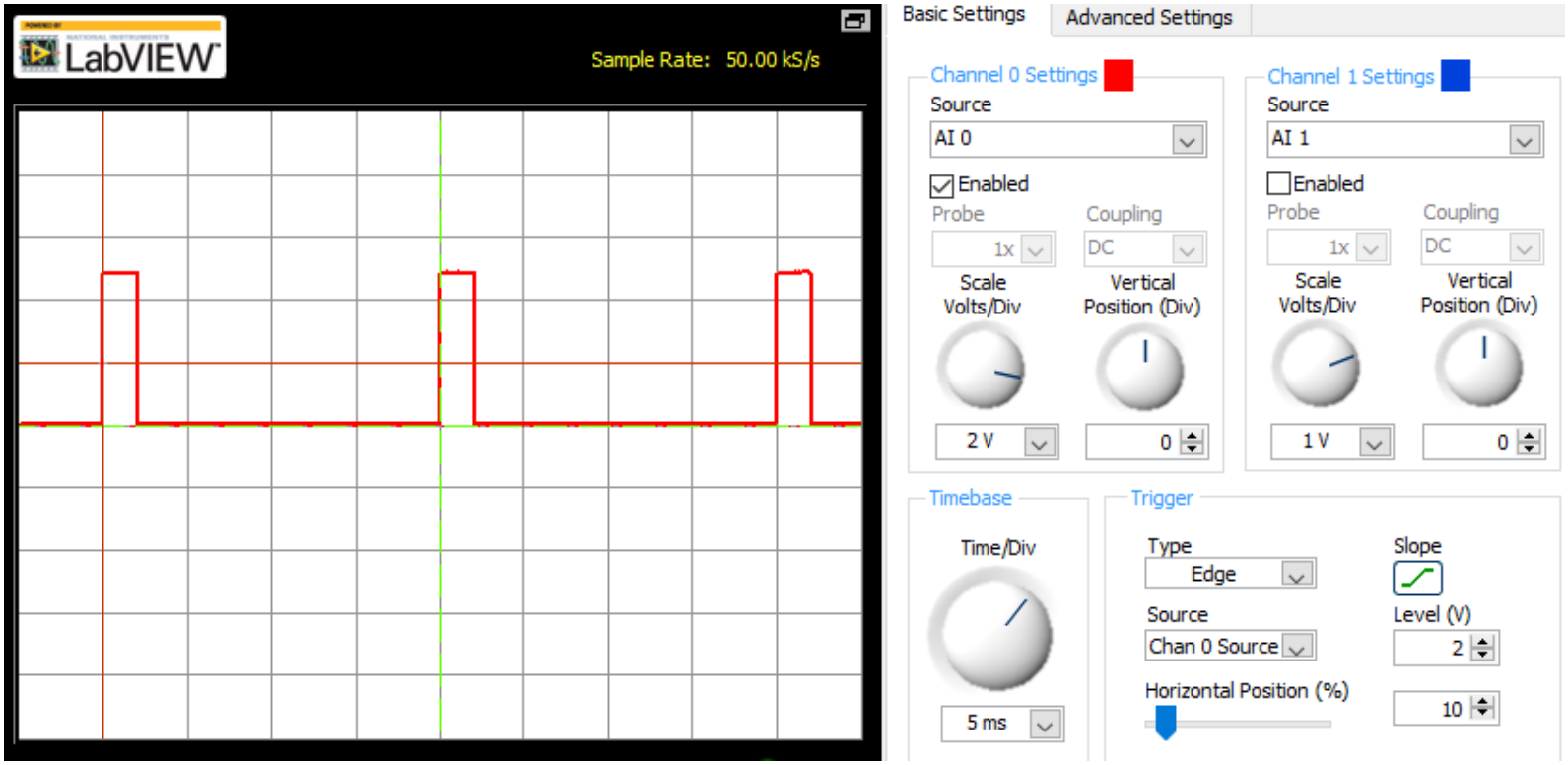
This screen captures shows a time high of ~2.1 ms.

Program Output

```
Analog In   856     Servo Angle    150
Analog In   855     Servo Angle    150
Analog In   855     Servo Angle    150
```



Screen capture showing the period of the PWM for the Servo. The period equals 20 milliseconds. It is the time high that is changed to control position.



Brushless DC Motor (BLDC)

Introduction

This is a new brushless DC motor with the added bonus of a built-in motor driver - this means it doesn't need any external motor drivers and you can connect it to an Arduino board directly!

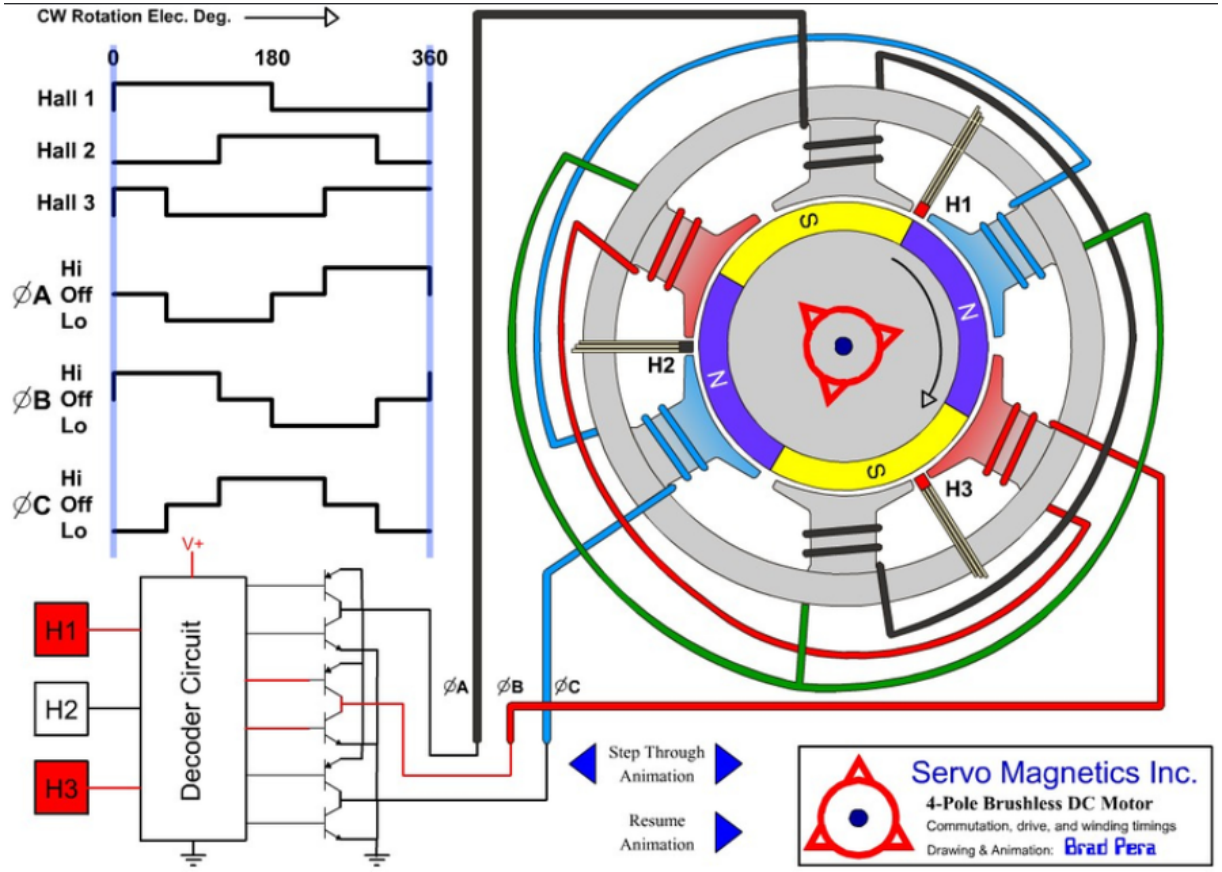
The brushless motor comes with direction control, PWM rotational speed control and frequency feedback output. It is suitable for miniature-sized mobile robotic platforms. With the motor speed feedback signal, it is useful in cyclic control systems.

Specification

- Operating Voltage: 12V
- Motor Rated Speed: 7100-7300rpm
- Torque: 2.4kg*cm
- Speed: 159 rpm/min approx.
- Reduction ratio: 45:1
- Signal cycle pulse number: 45*6 (Each cycle outputs 6 pulse)
- Control mode:
  - PWM speed control
  - Direction control
  - Feedback pulse output

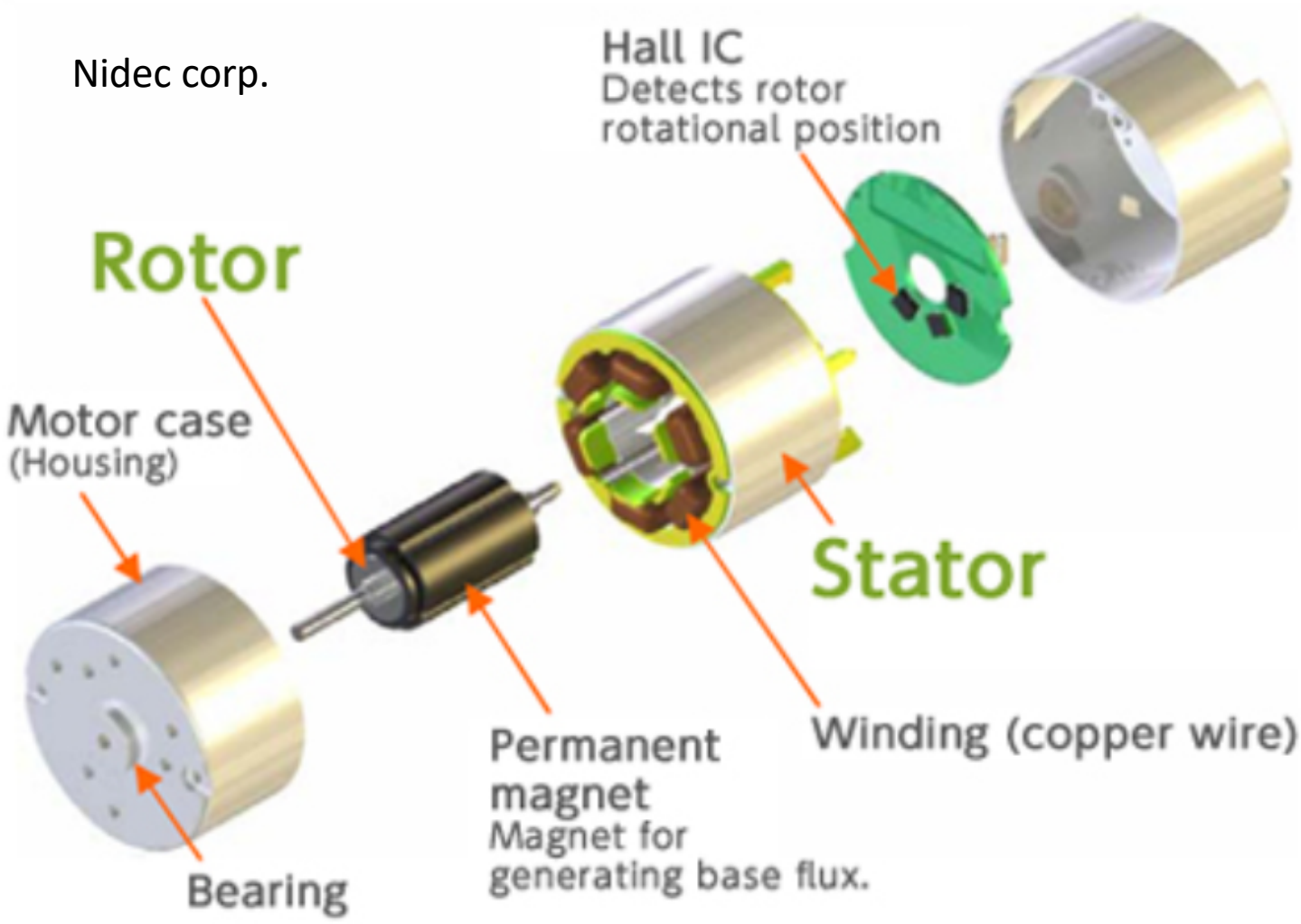
Brushless DC Motor (BLDC)

Hall sensors detect the current position of the motor and electronic circuit energizes the coils in sequence.

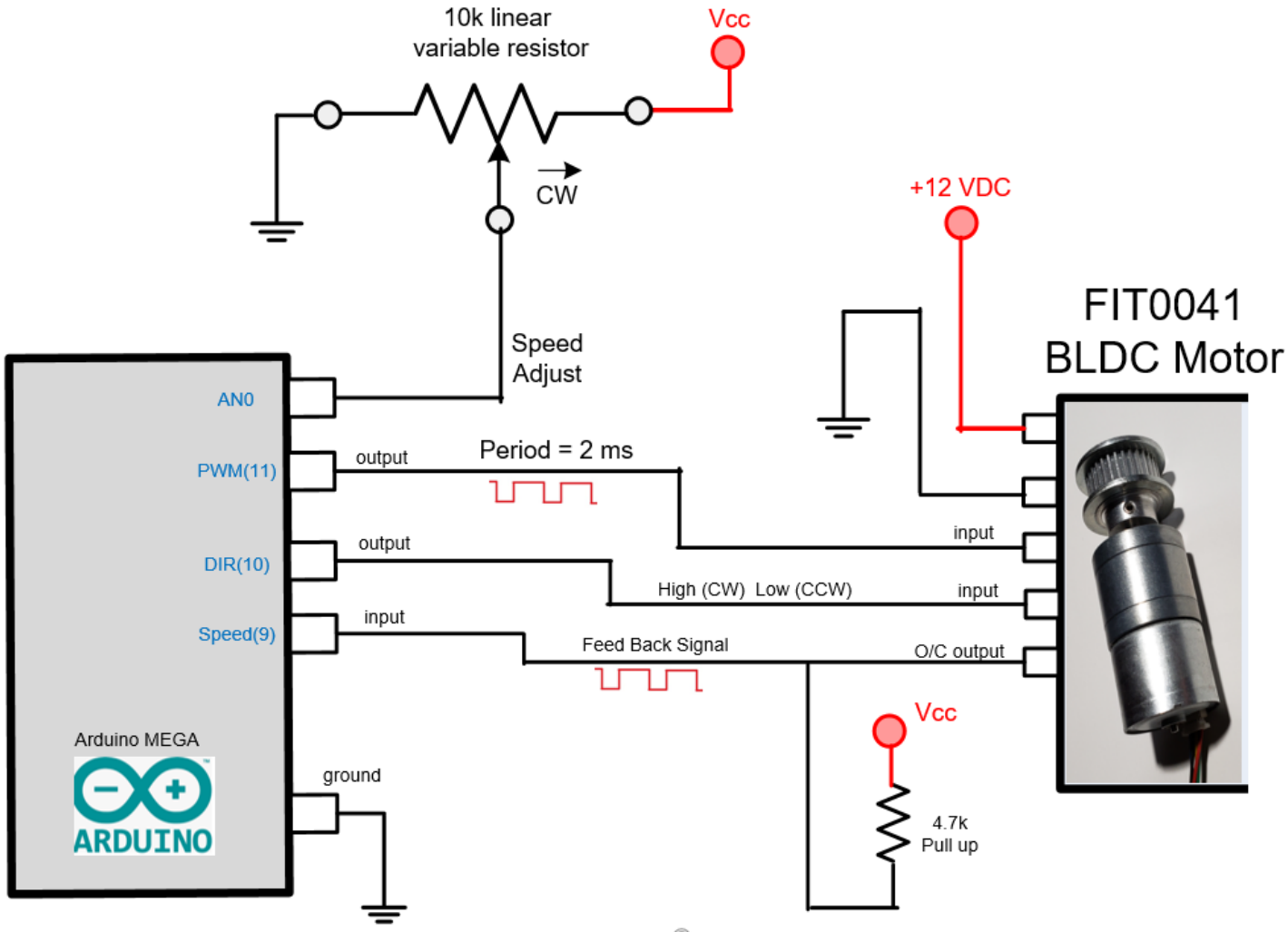


Brushless DC Motor (BLDC)

Nidec corp.



Brushless DC Motor (BLDC)



Brushless DC Motor (BLDC)

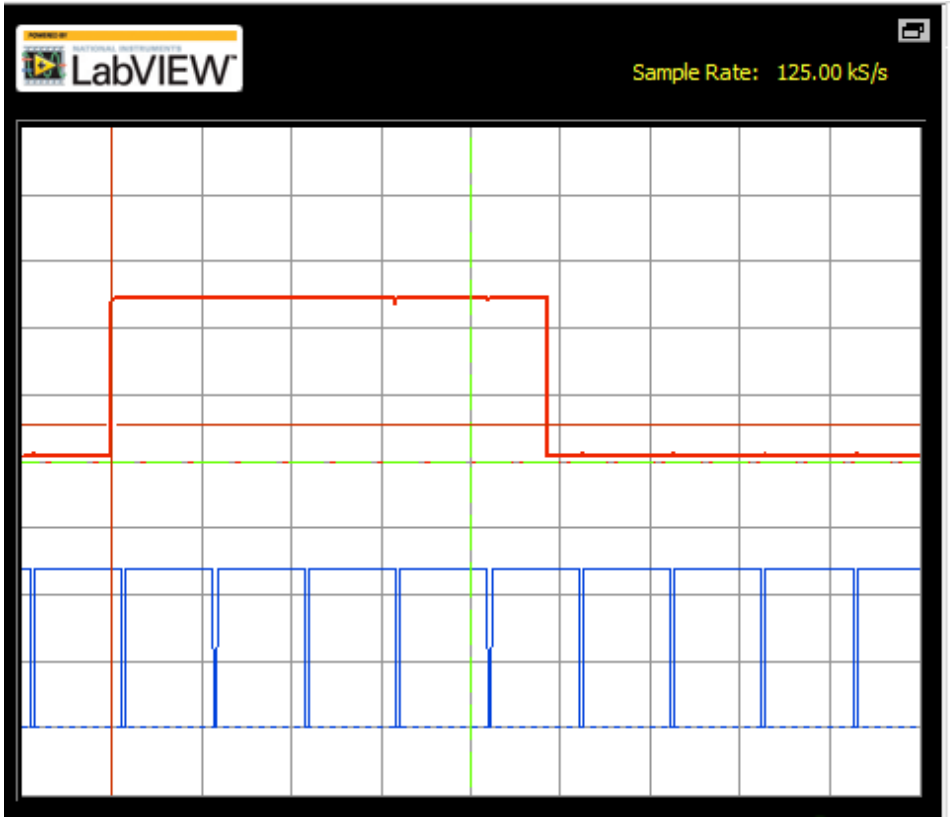
Label	Name	Description
1	PWM	PWM Control, 0-5V (20~30KHz)
2	Power -	POWER- (GND)
3	Direction	Direction Pin: +5V or dangling, motor moves forward; GND or cathode, motor moves backward.
4	FG	FG signal pin(need a pull-up resister-5k)
5	Power +	POWER+ (12V)

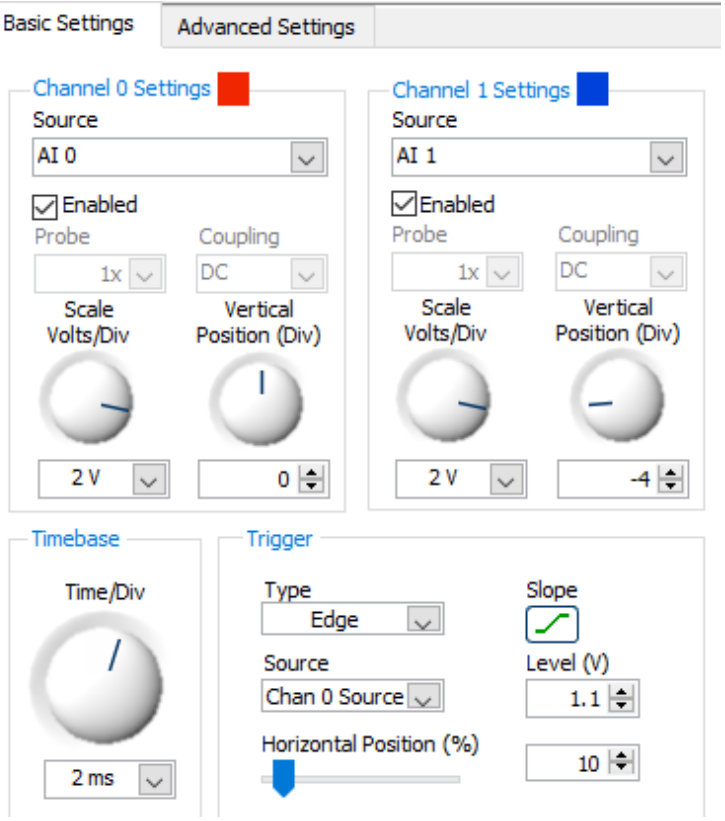
Brushless DC Motor (BLDC)

- More Efficient than Brushed motors
- Less maintenance
- Easy to control, built-in
- Some have feedback

```
22 int AIN_0 = analogRead(0);  // value between 0 and 1023
23 int PWM1 = 127;
24 PWM1 = map(AIN_0,0,1023,0,255);
25
26 digitalWrite(10,HIGH);   //high = CW, low = CCW
27 analogWrite(11,PWM1);   // PWM runs at 500 Hz  duty cycle 0 to 255
28 int TH = pulseIn(9,HIGH);   // measures time high in microseconds
29
30 Serial.print(PWM1); Serial.print("  ");
31 Serial.print(AIN_0); Serial.print("  ");
32 Serial.println(TH);
```

Brushless DC Motor (BLDC)



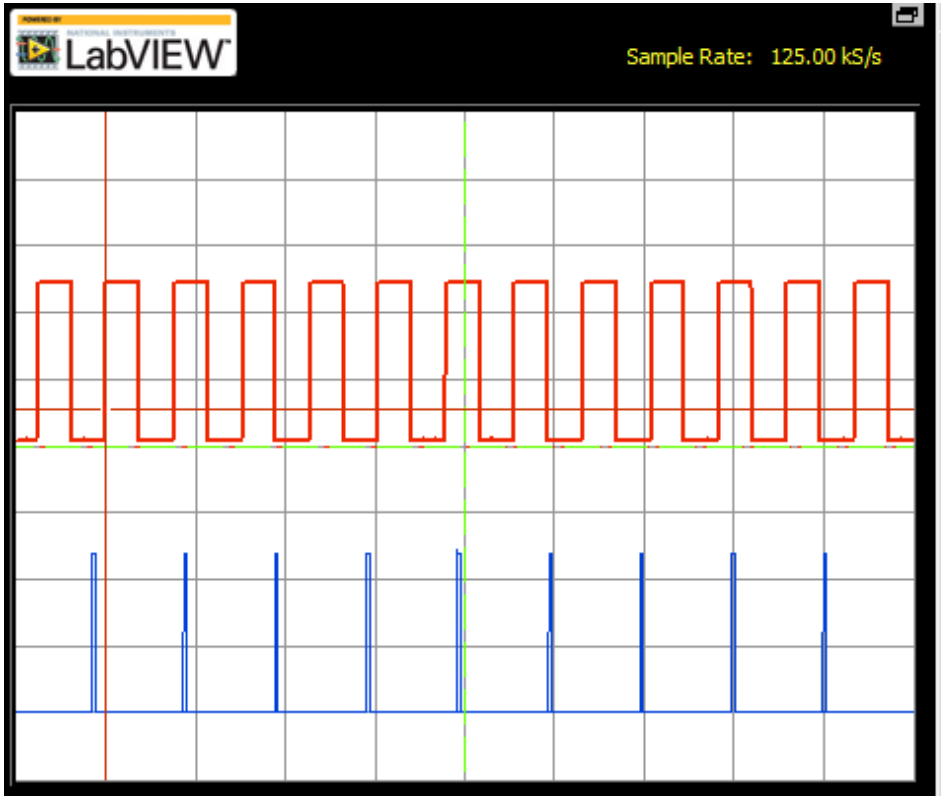


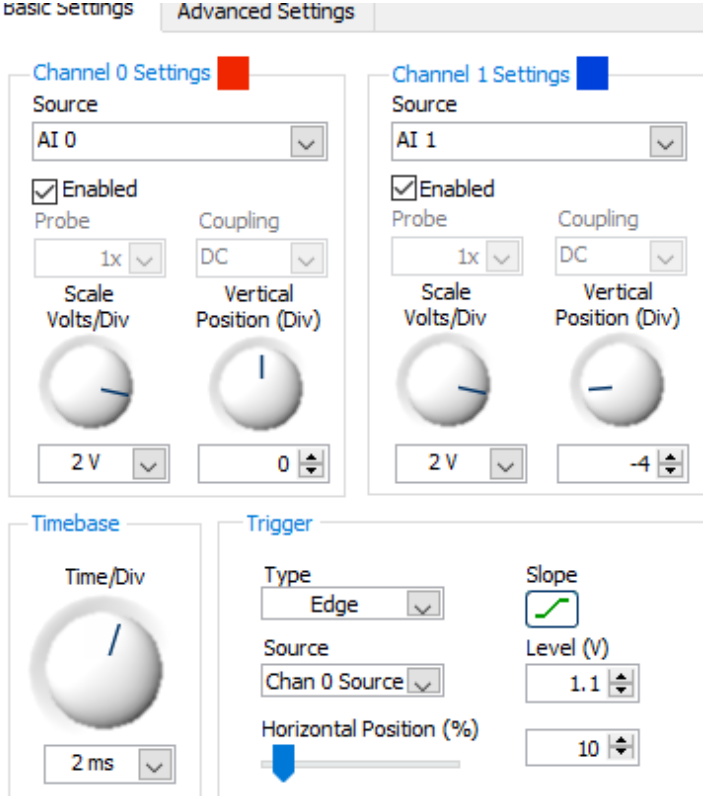

```
244   982   9873
244   982   10250
244   982   11908
244   982   10285
```

```
Serial.print(PWM1);
Serial.print(AIN_0);
Serial.println(TH);
```

Takes 5.65 sec/rev.

Brushless DC Motor (BLDC)






```
8   36   735
8   36   753
8   36   731
8   36   746
```

```
Serial.print(PWM1);
Serial.print(AIN_0);
Serial.println(TH);
```

Takes 8 sec for 20 revolutions.
0.4 sec/rev

Brushless DC Motor (BLDC)

DC_brushless_control §

```
// Brushless DC motor Control  Michel Hanbury
// November 20th 2018

int TH = 0;
unsigned long time = 0;

void setup() {
  Serial.begin(9600);
  pinMode(10, OUTPUT); //PWM PIN 11  with PWM wire
  pinMode(11, OUTPUT);//direction control PIN 10 with direction wire
}

void loop() {

// Note: PWM on motor +5 is off, 0.0 volts 100% on
// motor produces 6 pulses per revolution of the DC motor
// Motor gear ratio is 45 to 1
// Motor is a FIT0441 Brushless  12 volts DC supply
// Red + 12, Black is ground and motor(-), yellow direction,
// Green feedback pulse, blue PWM (gnd = 100%)

int AIN_0 = analogRead(0);  // value between 0 and 1023
int PWM1 = 127;
PWM1 = map(AIN_0,0,1023,0,255);

digitalWrite(10,HIGH);   //high = CW, low = CCW
analogWrite(11,PWM1);   // PWM runs at 500 Hz  duty cycle 0 to 255
int TH = pulseIn(9,HIGH);   // measures time high in microseconds

Serial.print(PWM1); Serial.print("  ");
Serial.print(AIN_0); Serial.print("  ");
Serial.println(TH);

}
```

```
174  702  971
174  701  934
174  701  941
174  702  939
174  701  944
174  702  926
174  701  950
174  701  955
174  702  923
174  702  962
174  701  928
174  702  926
174  701  951
174  701  941
```

```
9  40  735
9  40  724
9  40  751
9  40  735
9  40  738
9  40  732
9  40  715
9  40  732
9  40  734
9  40  737
9  40  742
9  39  734
9  40  726
9  40  744
```

```
64  260  751
64  260  770
64  260  757
64  260  768
64  260  751
64  260  757
64  260  770
```

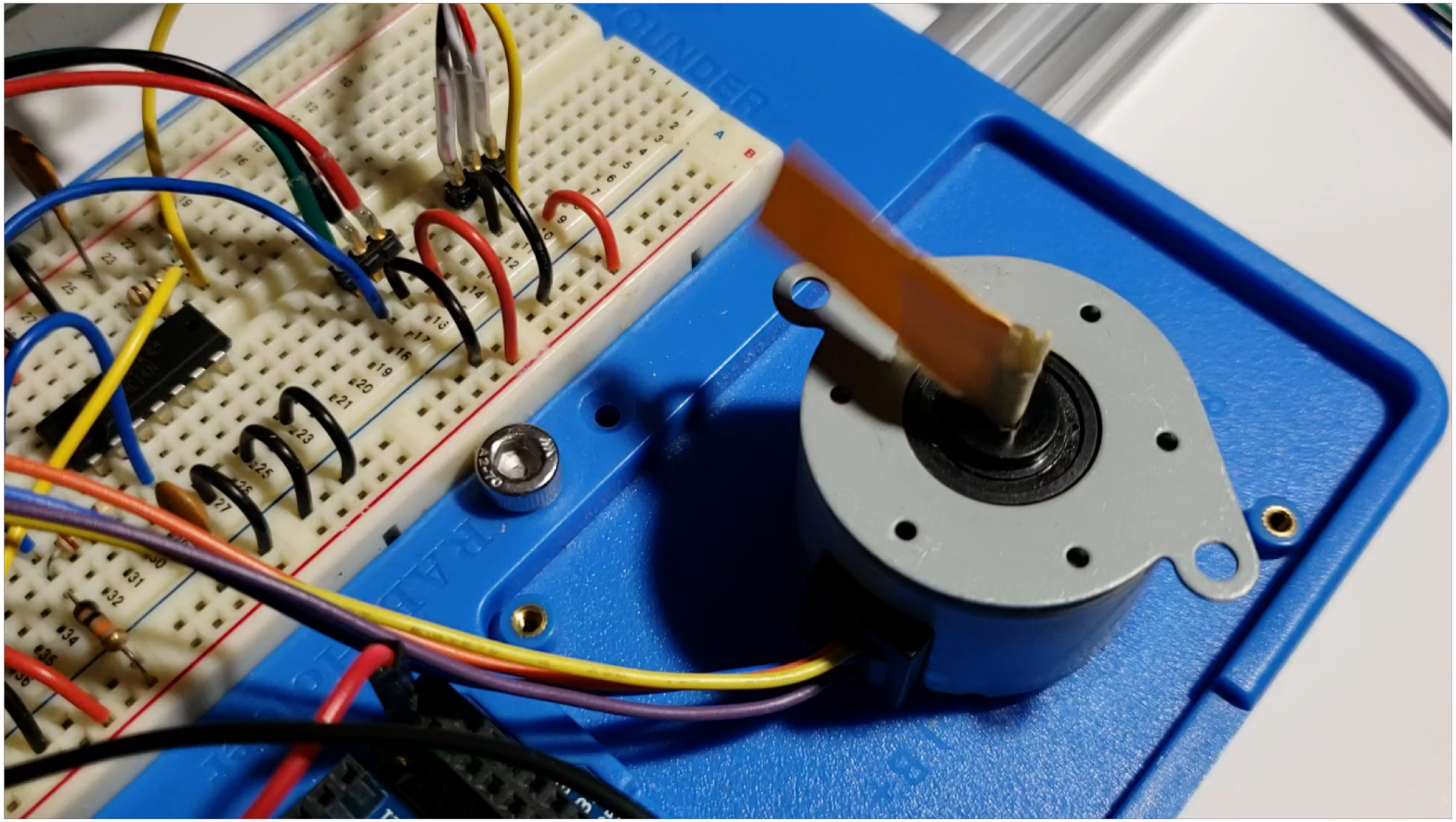
Small Stepper FWD and REV

```
#define direction 4
#define step 5
#define delaym 10    // 10.4 milliseconds high and 10.4 ms low
#define delayu 400

void setup() {
  pinMode(step, OUTPUT);  // step pulse
  pinMode(direction, OUTPUT);  // direction
}

void loop()
{

digitalWrite(direction,LOW); //clockwise
stepperslow();
delay(500);
digitalWrite(direction,HIGH); // counter clockwise
stepperslow();
delay(500);

}
```

Small Stepper FWD and REV

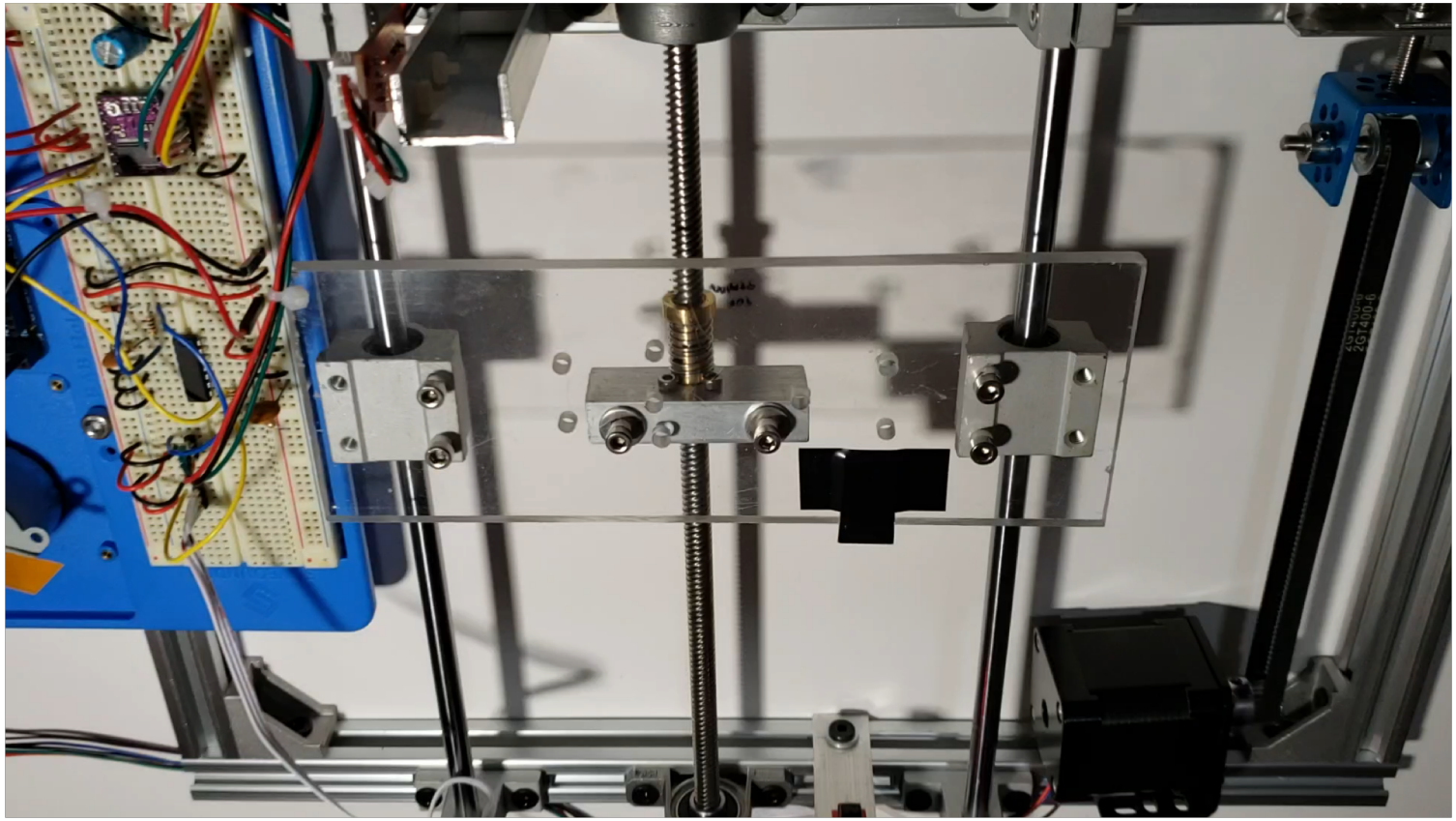
```
28 void stepperslow()   // stepper function
29 {
30 for (int x=0; x < 96; x++) // execute for two rotations
31 {
32    digitalWrite(step, LOW);
33    delayMicroseconds(delayu);  //maximum 16,383
34    delay(delaym);
35
36    digitalWrite(step, HIGH);
37    delayMicroseconds(delayu);
38    delay(delaym);
39 }
40 }
```

Small Stepper FWD and REV



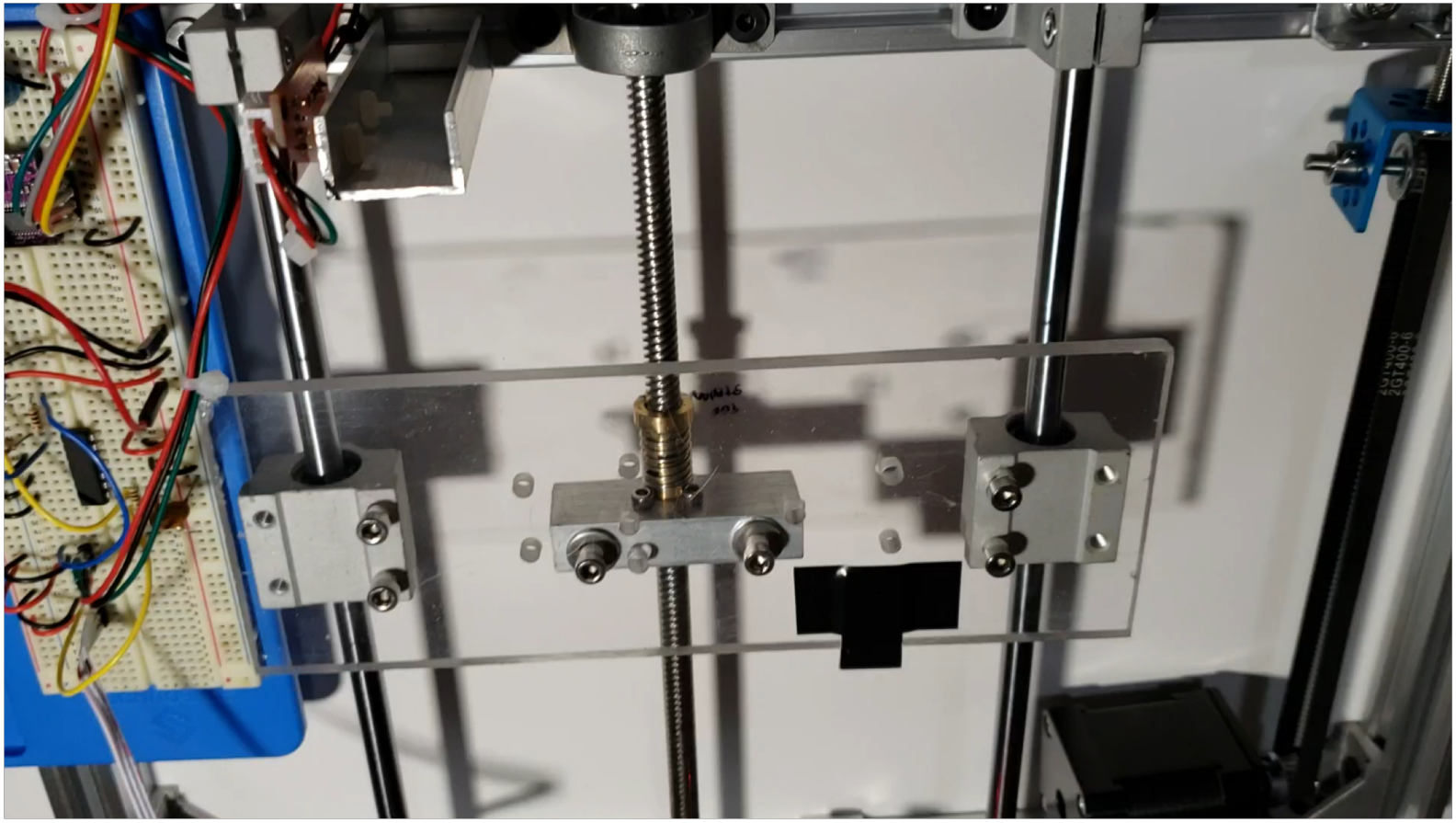
Smaller Stepper Forward and Reverse. (Video)

Small Stepper FWD and REV



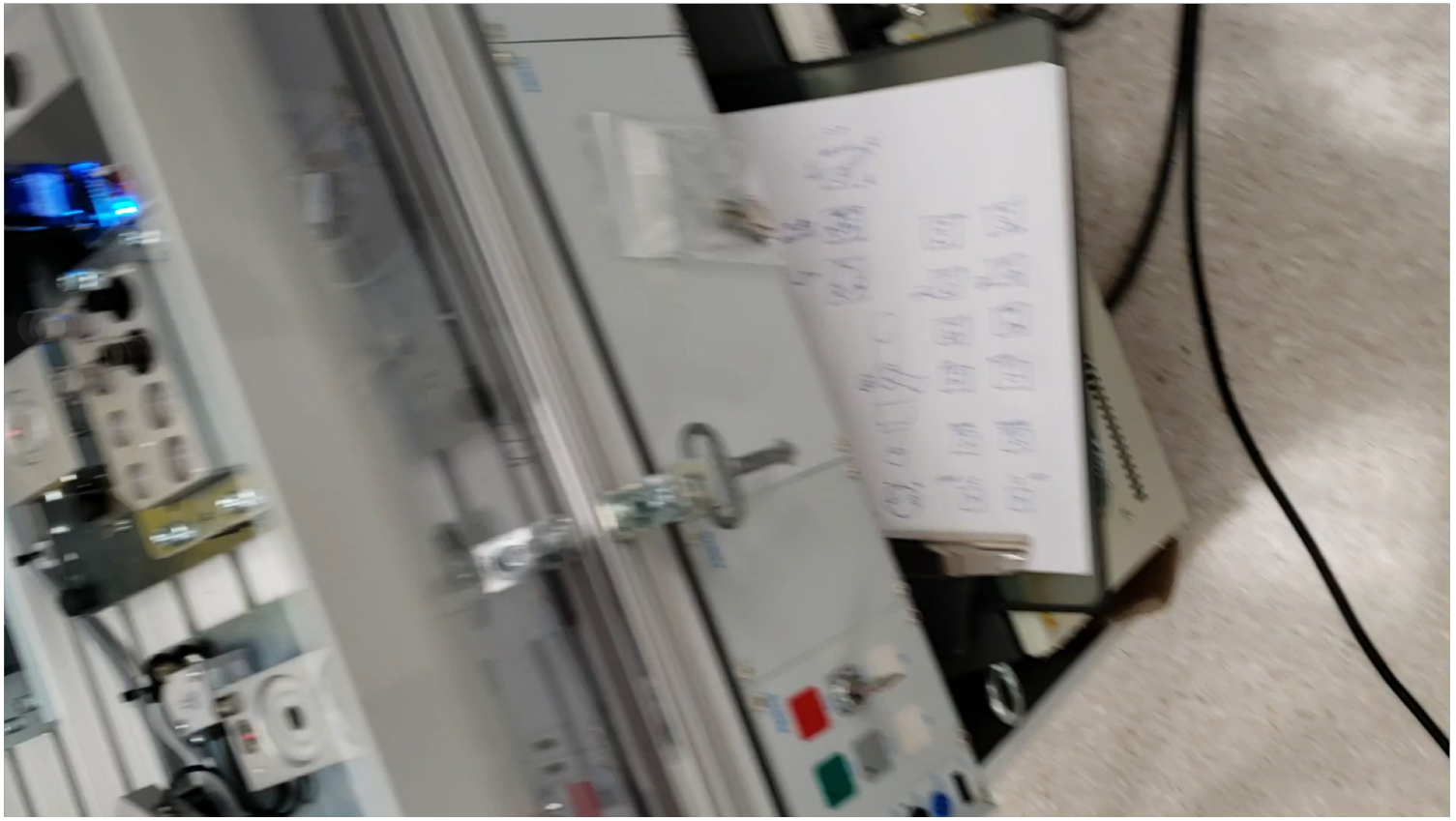
Lead Screw connected to stepper with flexible coupler. (Video)
Program uses interrupts.

Small Stepper FWD and REV

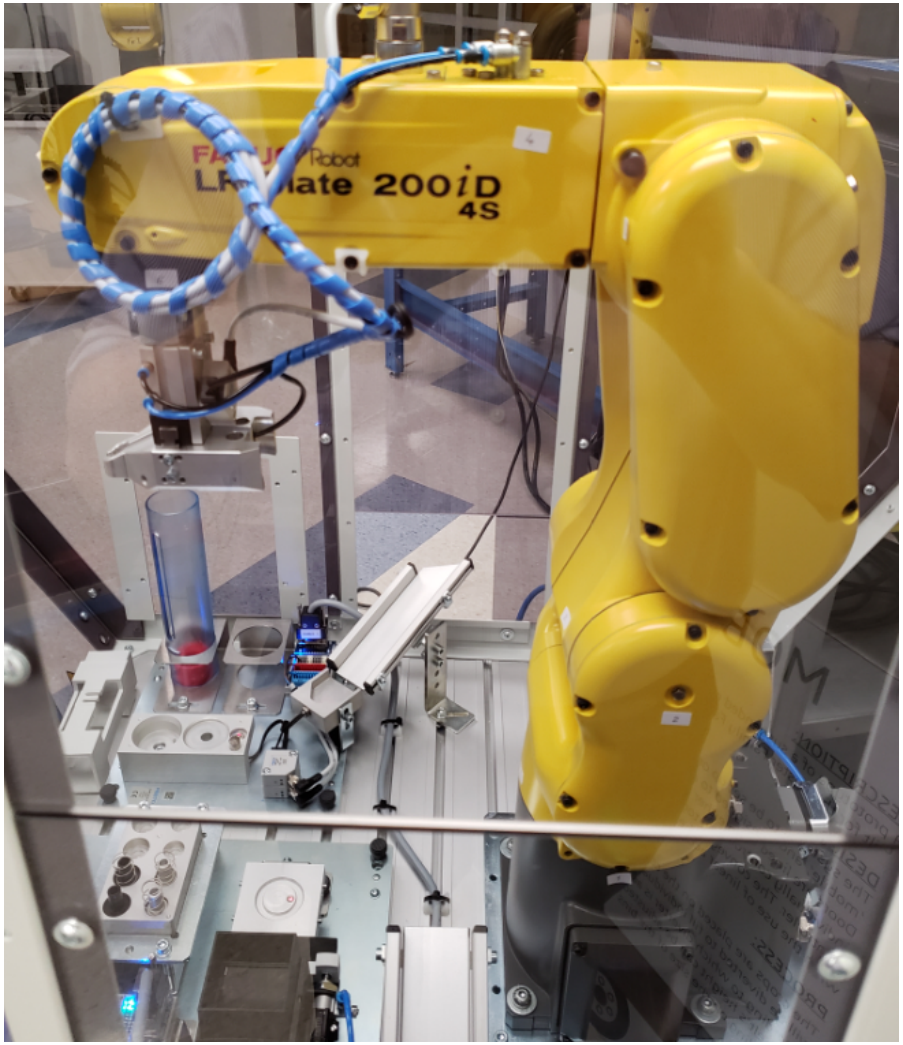


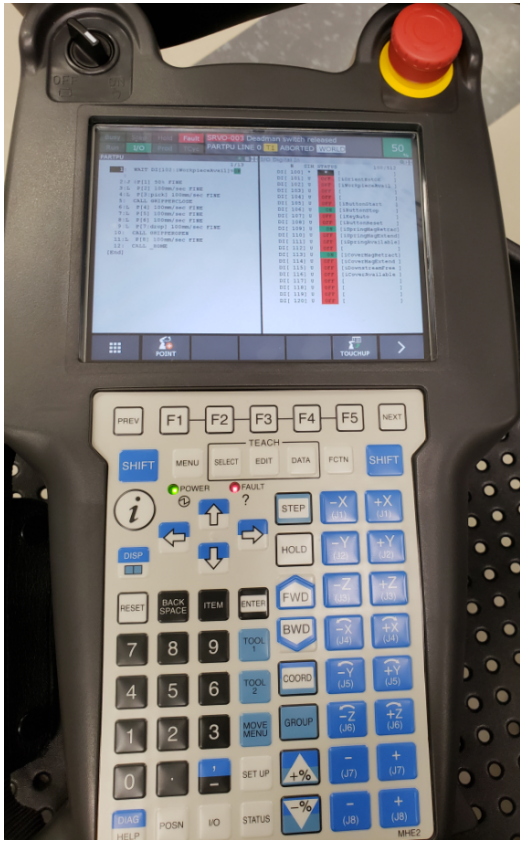
Forward and reverse.

Robot Arm T213

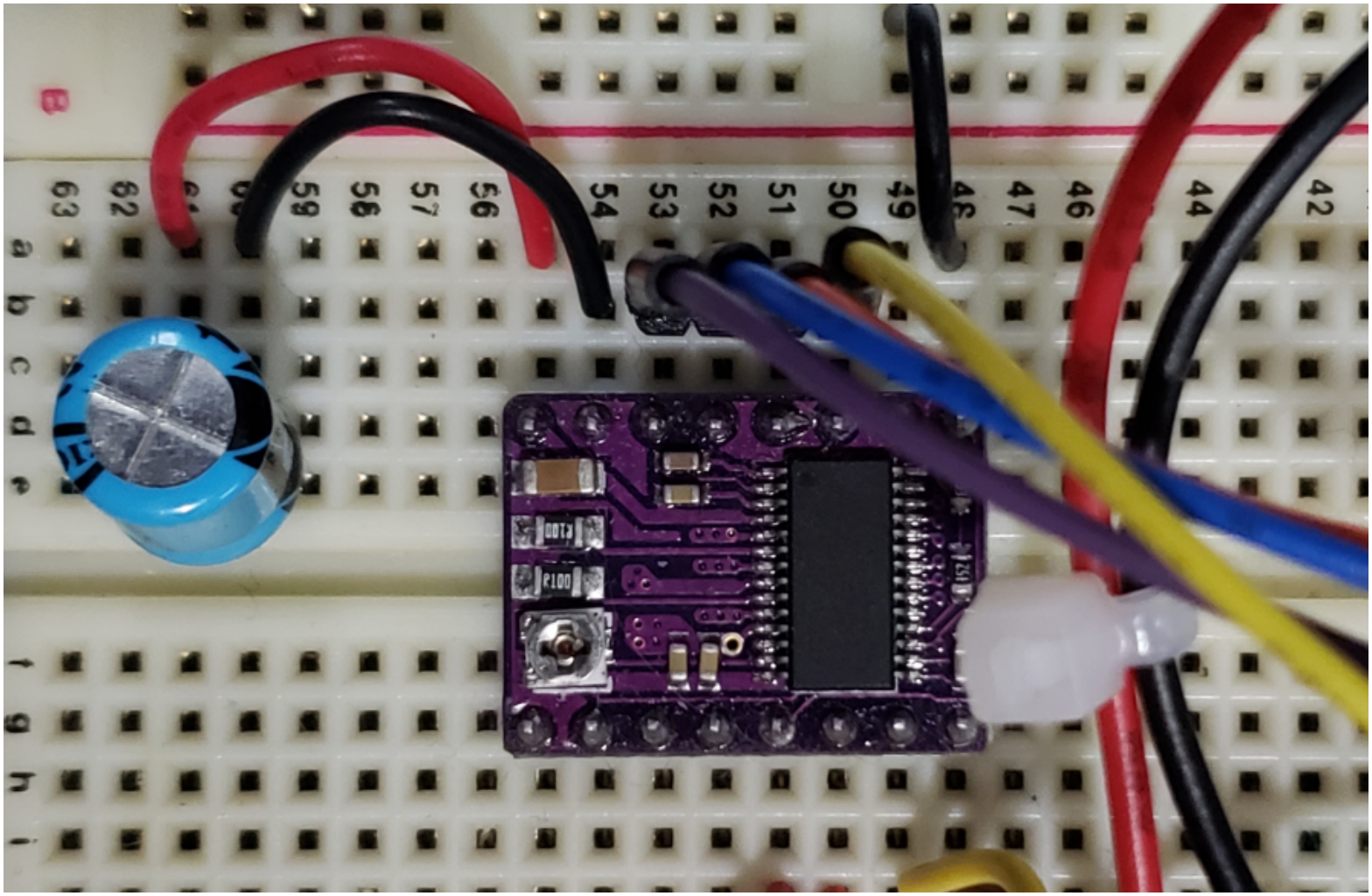


Small Stepper FWD and REV





Small Stepper FWD and REV



Small Stepper FWD and REV