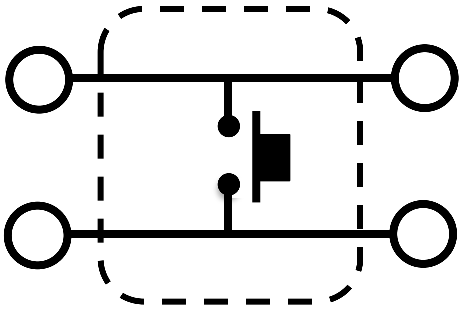
Model Railway Animation: Part 3 Expanded
By David King

Welcome to the expanded instalment of Part 3 of this series as we will some create a couple additional sketches on how to control our small DC motors. The first sketch will allow us to start and stop a motor with the use of a single pushbutton while the second sketch will allow use the added ability to reverse the direction of rotation of the motor using a second pushbutton. We will continue to use the potentiometer to control the speed of the motor.

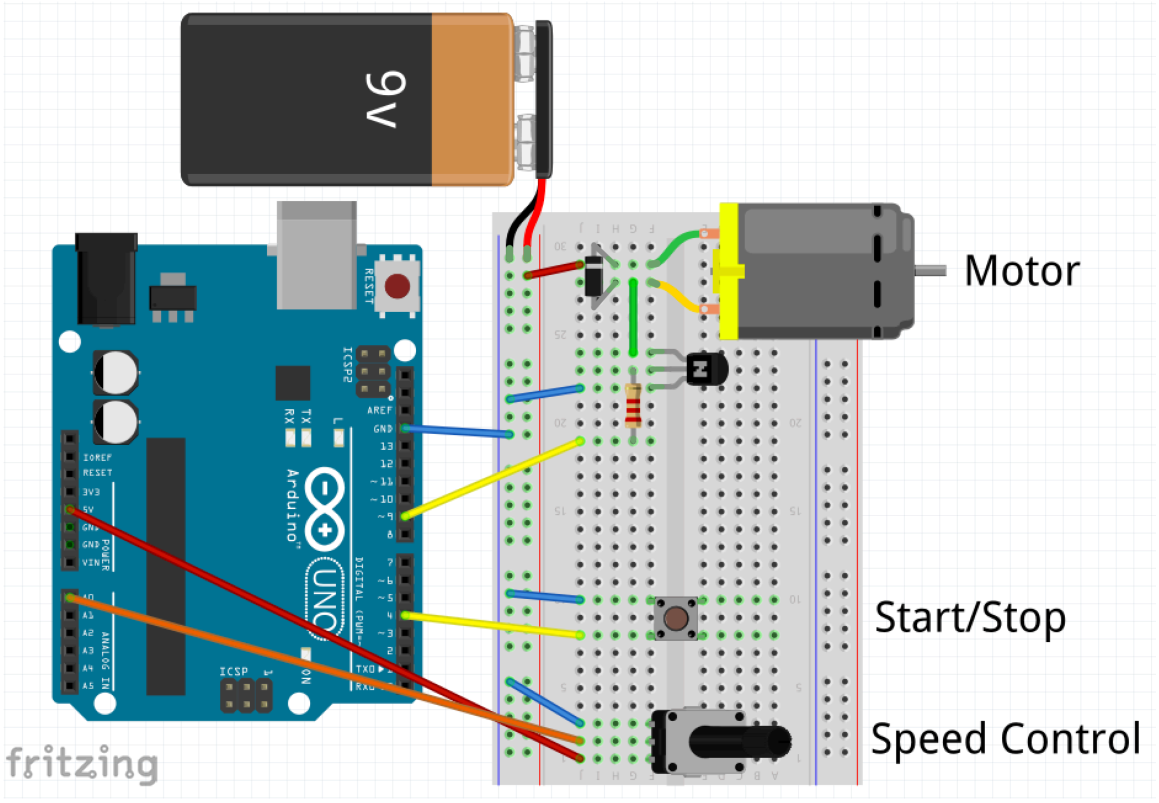
Starting and Stopping the Motor

To modify our existing sketch to allow us to start and stop the small DC motor using a single pushbutton. The only hardware needed will be one of the small pushbuttons included in your kit and a couple of jumper wires.



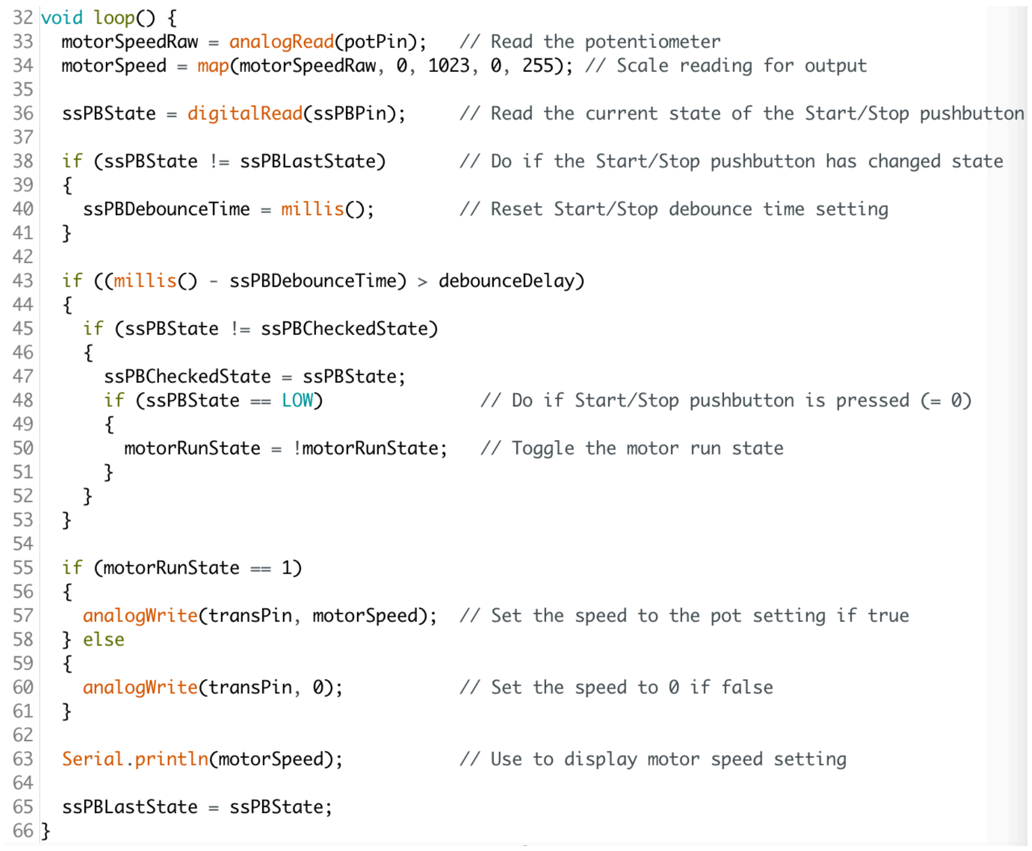
The image here is showing a typical pushbutton switch similar to the ones included in your kit along with the how the pins are connected. It is possible to have other wiring or pinout arrangements than the one I show here but this setup is the most common.

To wire up this pushbutton you will need to connect one end of a jumper from one side of the pushbutton and the other end of the jumper will be connected to ground. The second jumper will be connected from the other side of the pushbutton switch to digital pin 4 on the Uno. The Fritzing image on the next page will show you this wiring. Note that I have removed the second motor from the Fritzing diagram to make this image and wiring easier to view. If you are using the Arduino kit along with the Mosfet to control the motor just use the wiring for that configuration as shown in the article in The Canadian.



Let us look at the coded needed to control this motor while using the Start/Stop pushbutton. The code for the speed control portion remains similar as it was used in the previous sketch when we were only controlling the speed of the motor. Here is the complete code.

```
// DC_Motor_w_Pot_StartStop.io by David King
// This sketch is used for the most basic of speed controls for a DC motor.
// This sketch has the added feature of a Start/Stop pushbutton.
// The value of the pushbutton is as follows since we are using INPUT_PULLUP
// -- button not pressed = 1
// -- button is pressed  = 0

// declare any variables needed in you file here

int transPin = 9;       // This pin is connected to the transistor/mosfet
int potPin = A0;        // This pin is connected to the potentiometer
int ssPBPin = 4;        // This pin is connected to the Start/Stop pushbutton

int motorSpeed = 0;     // This is the speed of the motor on output pin
int motorSpeedRaw = 0;  // This is the raw value from the potentiometer
int motorRunState = 0;  // Set the initial state of the motorRunState
int ssPBState;          // The current state of the Start/Stop pushbutton
int ssPBLastState = 1;  // Set the inital state of the Start/Stop last state
int ssPBCheckedState;   // Set the initial state of the checked state

unsigned long debounceDelay = 50;
unsigned long ssPBDebounceTime = 0;

void setup() {
  // put your setup code here, to run once:
  pinMode(transPin, OUTPUT);      // Set the pin for the transistor
  pinMode(ssPBPin, INPUT_PULLUP); // Set the pin for the Start/Stop pushbutton
  analogWrite(transPin, 0);       // Set the starting speed to 0
  Serial.begin(9600);             // Enable the serial monitor
}

```



```
32 void loop() {
33   motorSpeedRaw = analogRead(potPin);    // Read the potentiometer
34   motorSpeed = map(motorSpeedRaw, 0, 1023, 0, 255); // Scale reading for output
35
36   ssPBState = digitalRead(ssPBPin);      // Read the current state of the Start/Stop pushbutton
37
38   if (ssPBState != ssPBLastState)        // Do if the Start/Stop pushbutton has changed state
39   {
40     ssPBDebounceTime = millis();         // Reset Start/Stop debounce time setting
41   }
42
43   if ((millis() - ssPBDebounceTime) > debounceDelay)
44   {
45     if (ssPBState != ssPBCheckedState)
46     {
47       ssPBCheckedState = ssPBState;
48       if (ssPBState == LOW)              // Do if Start/Stop pushbutton is pressed (= 0)
49       {
50         motorRunState = !motorRunState;  // Toggle the motor run state
51       }
52     }
53   }
54
55   if (motorRunState == 1)
56   {
57     analogWrite(transPin, motorSpeed);  // Set the speed to the pot setting if true
58   } else
59   {
60     analogWrite(transPin, 0);           // Set the speed to 0 if false
61   }
62
63   Serial.println(motorSpeed);           // Use to display motor speed setting
64
65   ssPBLastState = ssPBState;
66 }
```

Some new variables have been added in the declarations area of the code for assignment as needed for the pushbutton and a timer. Also one line of code was added to the void setup() code as we needed to the set the function on the digital I/O pin being used by the pushbutton.

The biggest change is located in the void loop() section of the code. Here we have added a block of code from lines 36 to 53 and line 65 that is used to read the state of the pushbutton, set a delay timer so that we only see the pushbutton once per activation and finally set the motorRunState to true (1) or false (0). The code on line 63 is used to only allow the previous code in the loop to function once until the pushbutton either released or pressed again.

In rungs 55 to 61 we check to see if the motorRunState is true or false. If the state is true we turn on the motor by setting the output pin to a value based on the position of the potentiometer which will power up the motor. If the state is false we set the output pin to a value of zero that will stop the motor.

If you have followed the wiring shown in the Fritzing diagram and created your sketch using similar code to what I have shown above you should now be able to control the operation of your motor by starting and stopping it when you want and also be able to control the speed of the motor.

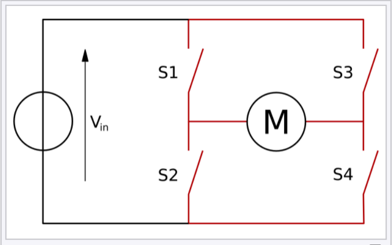
Adding Direction Control

The next logical step in motor control would be to add some sort of directional control to your existing sketch. This sounds easy and it really isn't that hard but it will require some understanding of how we reverse the direction of a small DC motor and the transistor or Mosfet driving the motor will need to be changed to something a little more elaborate.

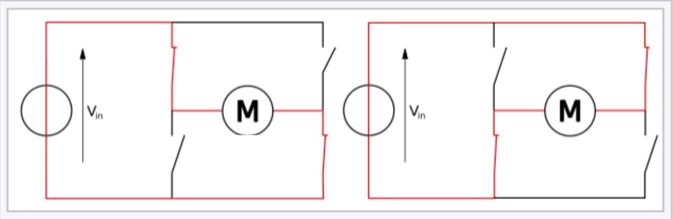
Let us first look at how the motor can have its direction reversed.

Changing the direction of rotation is a simple operation as all that is needed is to reverse the 2 leads of the motor. If you have been following the wiring directions that you have been shown so far the motor is either turning clockwise or counter-clockwise when viewing the shaft end of the motor. Each time you start the motor it rotates in the same direction each time. To reverse the motor simply swap the position of the 2 wires from the motor. Now when you start the motor its direction of rotation will be reversed. That's really how simple it is to reverse the direction of rotation on these small motors. Although this works well it is not practical to swap the motor connections every time you want to reverse the direction of rotation for your motor. We need a better solution and this can be done by changing the electronic components that we are using to drive the motor.

H-Bridge



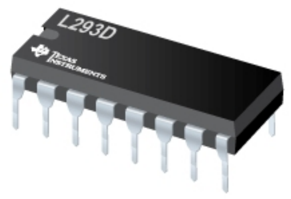
Structure of an H bridge (highlighted in red)



The two basic states of an H bridge

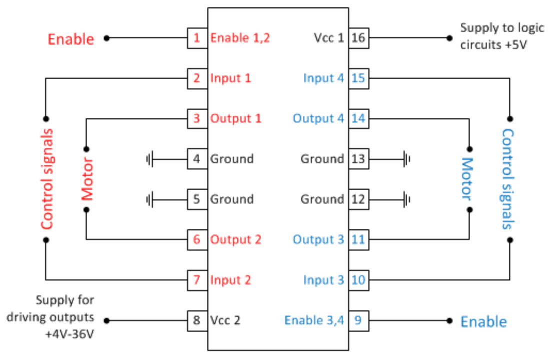
These 2 images from Wikipedia show the basic operation of a H-Bridge circuit. The left image shows us a motor that has been connected to 4 switches and a DC voltage supply which could be our 9-volt battery. The switches (S1, S2, S3 and S4) could all be transistors or Mosfets. The right image shows that if you close switches S1 and S4 we see that the left side of the motor is connected to the top of Vin and the right side of the motor is connected to the bottom of Vin. Now if we only close switches S2 and S3 we see that the left side of the motor is connected to the bottom of Vin and the right side of the motor is connected to the top of Vin. By operating the switches in this fashion we can now control which side of motor is connected to the positive or negative lead of the battery. By doing this we can now control the direction of rotation for our motor.

An easier method of wiring is to use an IC (integrated chip) that contains all of the transistors and other components needed to create the H-Bridge. The IC we will use is a L293D. If you have the Arduino kit the IC has been included. If you have the Adafruit or SparkFun kit this IC is not included but it can be obtained from most electronic supply shops that maybe located locally to you or from other on-line suppliers.



Doing a quick search of the on-line suppliers I found the IC for about $5 Canadian for the DIP-16 type of L293D IC.

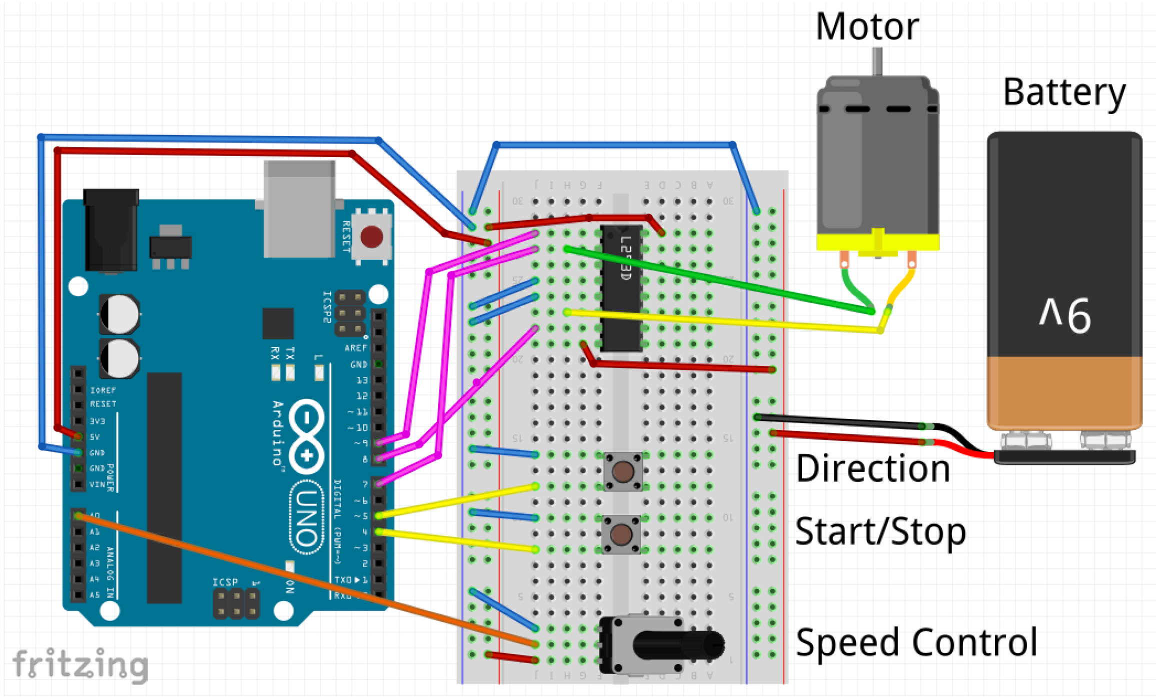
This IC has the ability to drive 2 motors, but we will only drive 1 motor for now. As a result we will be connecting 9 wires to the IC. Pins 1 to 8 and pin 16 will be used. This image shows the pinout connections on this IC and the image below shows the Fritzing diagram for you to follow using your kit components.



🟥 Connections for circuit 1

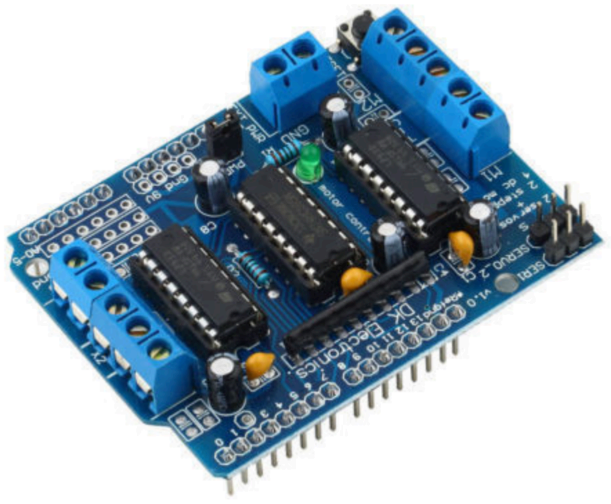
🟦 Connections for circuit 2

⬛ Common connections



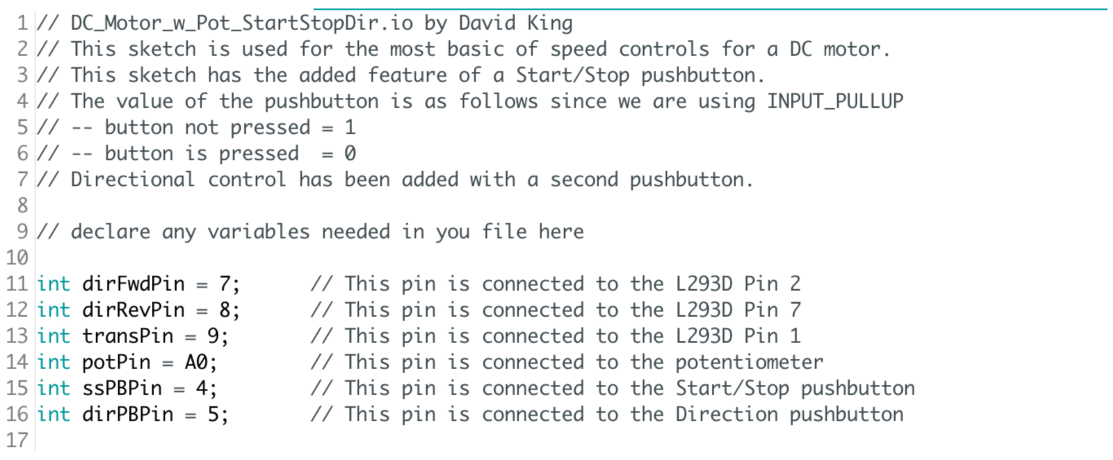
This may look complicated but just take your time making the connections and everything should be fine.

If you are looking for a pre-built solution for driving a DC motor or more than one you can purchase boards that are pre-built and ready for you to connect to your Uno microcontroller board. These pre-built units come in 2 main types, one that you run wires from your microcontroller to the board while the second is a pre-built “shield” that connects directly on top of, stacks, on your Uno. Either of these boards could cost you as much as $30 Canadian locally but if you are willing to wait a few weeks you can order them over eBay or similar service and have them shipped from China for as low as $2.50 Canadian. The images below both types, with the board on the left and the shield on the right.



The New and Complete Code

Based on the hard-wired sample in the Fritzing diagram I’m going to show you the code that I used in my sketch for the completed Start/Stop, Direction and Speed Control. I have broken down the code into 4 images to separate the major sections from each other. The first section of code, lines 1 to 32, includes all of the header comments and all of the variable declarations.



```
 1 // DC_Motor_w_Pot_StartStopDir.io by David King
 2 // This sketch is used for the most basic of speed controls for a DC motor.
 3 // This sketch has the added feature of a Start/Stop pushbutton.
 4 // The value of the pushbutton is as follows since we are using INPUT_PULLUP
 5 // -- button not pressed = 1
 6 // -- button is pressed  = 0
 7 // Directional control has been added with a second pushbutton.
 8
 9 // declare any variables needed in you file here
10
11 int dirFwdPin = 7;      // This pin is connected to the L293D Pin 2
12 int dirRevPin = 8;      // This pin is connected to the L293D Pin 7
13 int transPin = 9;       // This pin is connected to the L293D Pin 1
14 int potPin = A0;        // This pin is connected to the potentiometer
15 int ssPBPin = 4;        // This pin is connected to the Start/Stop pushbutton
16 int dirPBPin = 5;       // This pin is connected to the Direction pushbutton
17
```

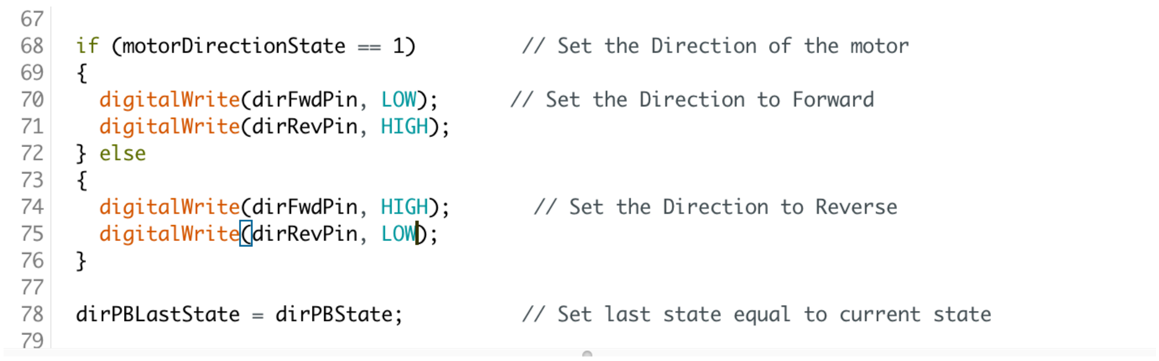
```
int motorSpeed = 0;      // This is the speed of the motor on output pin
int motorSpeedRaw = 0;  // This is the raw value from the potentiometer
int motorRunState = 0;  // Set the initial state of the motorRunState
int ssPBState;          // The current state of the Start/Stop pushbutton
int ssPBLastState = 1;  // Set the inital state of the Start/Stop last state
int ssPBCheckedState;   // Set the initial state of the checked state
int motorDirectionState = 0;  // Set the initial statte of the motorDirectionState
int dirPBState;         // The current state of the Direction pushbutton
int dirPBLastState = 1; // Set the initial state of the Direction last state
int dirPBCheckedState;  // Set the initial state of the checked state

unsigned long debounceDelay = 50;
unsigned long ssPBDebounceTime = 0;
unsigned long dirPBDebounceTime = 0;

```

The second block of code is the void setup() section where the pins are set and any initial values such as setting the speed of the motor to 0 just in case it wants to start moving.

```
void setup() {
  // put your setup code here, to run once:
  pinMode(dirFwdPin, OUTPUT);     // Set the pin for the L293D Forward
  pinMode(dirRevPin, OUTPUT);     // Set the pin for the L293D Reverse
  pinMode(transPin, OUTPUT);      // Set the pin for the L293D Speed
  pinMode(ssPBPin, INPUT_PULLUP); // Set the pin for the Start/Stop pushbutton
  pinMode(dirPBPin, INPUT_PULLUP);// Set the pin for the Direction pushbutton
  analogWrite(transPin, 0);       // Set the starting speed to 0
  Serial.begin(9600);             // Enable the serial monitor
}

```

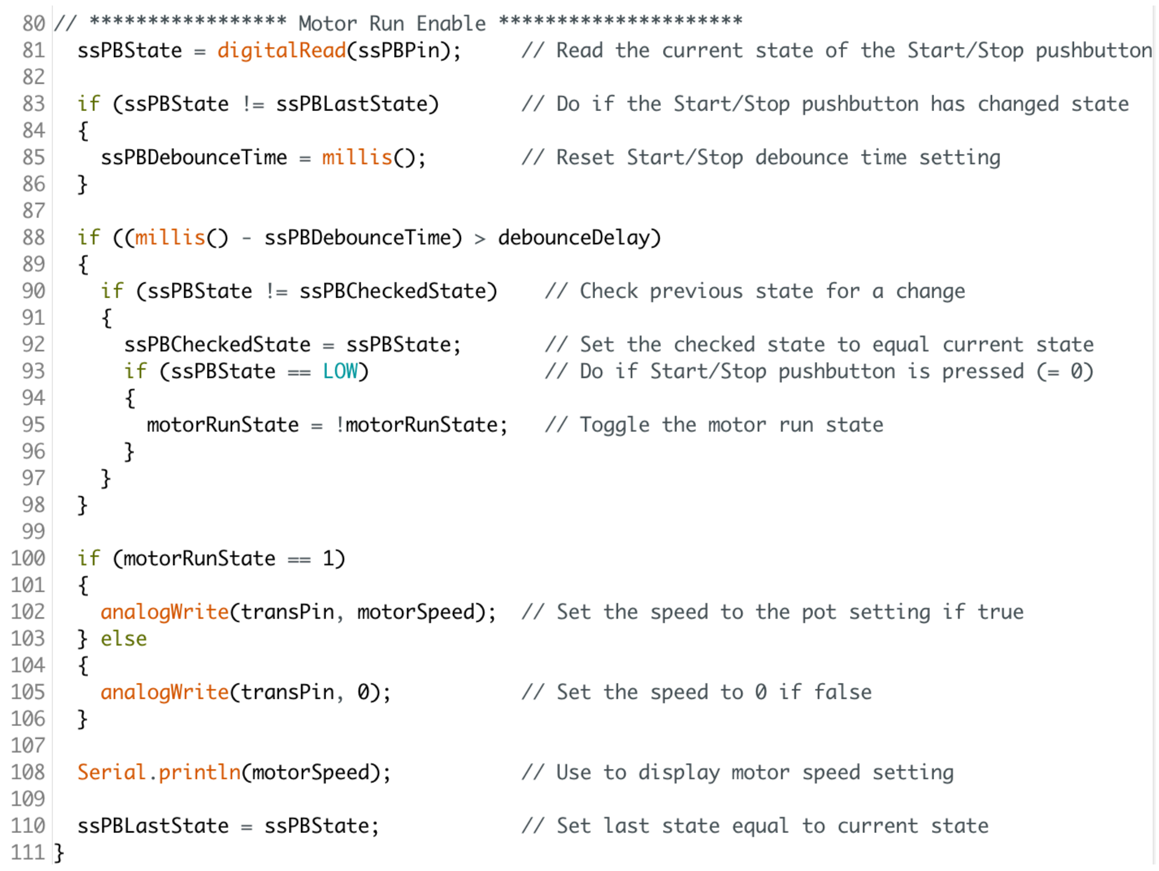
Next we continue in the void loop() section and add all of the coding needed to set the direction of the motor. A single pushbutton is used to toggle between forward and reverse.

```
void loop() {
  motorSpeedRaw = analogRead(potPin);   // Read the potentiometer
  motorSpeed = map(motorSpeedRaw, 0, 1023, 0, 255); // Scale reading for output

// ***************** Motor Direction Set ********************
  dirPBState = digitalRead(dirPBPin);   // Read the current state of the Direction pushbutton

  if (dirPBState != dirPBLastState)       // Do if the Direction pushbutton has changed state
  {
    dirPBDebounceTime = millis();        // Reset Direction debounce time setting
  }

  if ((millis() - dirPBDebounceTime) > debounceDelay)
  {
    if (dirPBState != dirPBCheckedState)    // Check previous state for a change
    {
      dirPBCheckedState = dirPBState;       // Set the checked state to equal current state
      if (dirPBState == LOW)                // Do if Direction pushbutton is pressed (= 0)
      {
        motorDirectionState = !motorDirectionState;   // Toggle the motor direction state
      }
    }
  }
```



```

  if (motorDirectionState == 1)         // Set the Direction of the motor
  {
    digitalWrite(dirFwdPin, LOW);      // Set the Direction to Forward
    digitalWrite(dirRevPin, HIGH);
  } else
  {
    digitalWrite(dirFwdPin, HIGH);       // Set the Direction to Reverse
    digitalWrite(dirRevPin, LOW);
  }

  dirPBLastState = dirPBState;          // Set last state equal to current state

```

Finally, the final portion of code to the Start/Stop control. This control also uses a single pushbutton to toggle the motor on and off.



```
// ***************** Motor Run Enable *********************
  ssPBState = digitalRead(ssPBPin);     // Read the current state of the Start/Stop pushbutton

  if (ssPBState != ssPBLastState)       // Do if the Start/Stop pushbutton has changed state
  {
    ssPBDebounceTime = millis();        // Reset Start/Stop debounce time setting
  }

  if ((millis() - ssPBDebounceTime) > debounceDelay)
  {
    if (ssPBState != ssPBCheckedState)    // Check previous state for a change
    {
      ssPBCheckedState = ssPBState;       // Set the checked state to equal current state
      if (ssPBState == LOW)               // Do if Start/Stop pushbutton is pressed (= 0)
      {
        motorRunState = !motorRunState;   // Toggle the motor run state
      }
    }
  }

  if (motorRunState == 1)
  {
    analogWrite(transPin, motorSpeed);  // Set the speed to the pot setting if true
  } else
  {
    analogWrite(transPin, 0);           // Set the speed to 0 if false
  }

  Serial.println(motorSpeed);           // Use to display motor speed setting

  ssPBLastState = ssPBState;            // Set last state equal to current state
}
```

Well that’s it, all 111 lines of code. This may look a little daunting but just take your time and it should all work out well.

Conclusion

I hope that you have been enjoying these series of articles on using a microcontroller to add animation and possibly some automation to your model train layout. This time we looked at motion by using both servos and small DC motors. If you combine what you have learnt here along with the previous articles and a little imagination you may find yourself extremely pleased on what you can accomplish. For those of you that know me you already know that I work with electronics and computers so combining these interests with my model railroading is just a logical step forward. You too can take this step, I'm sure of it.

Join me in the next article, Part 4, in this series as I take a look at using a LCD display that you can use to display information on your layout for your operators. This information could include information if a staging track is empty, how fast you are travelling along the mainline or any other information. The information can be complex or simple depending on how far you want to go.

Until next time keep playing with all of your new knowledge to create and you will be amazed at how much you improve your skills along the way.

This is a project that you can start at any time even if you have not viewed the previous article from this series. The previous articles are available in the preceding issues of *The Canadian*, so enjoy!

Issue #59, Introduction
Issue #60, Part 1: LED's
Issue #61, Part 2: Input Controls