Brushless DC Motor Controller

REPORT

Abstract

Drive and Control of a Brushless DC Motor (BLDC)

This project deals with the drive and control method of a brushless DC motor (BLDC).
This type of motors represents the most recent end of a long evolution of motors technology.
It is known for its increased efficiency, increased reliability, reduced noise, and longer lifetime over brushed DC motors.
In this work, we discuss in the first part the theory of brushless DC motors and its advantages over other types of motors especially brushed DC motors. Then we realize the construction and operating principle of this motor.
In the second part we develop the hardware implementation of the control and drive board of a sensored BLDC motor.

Acknowledgements

The following report, while belonging to Project Garuda, benefited from the insight and direction of several people.

First, we wish to thank the Faculty Advisor Dr. R S Kulkarni and Dr. Dinesh M N, for their amazing support throughout the project.
Next, we wish to thank the Head of the Electrical Engineering department, Dr. Jaypal R, and rest of the Electrical department staff for their unrivalled support in providing us with all equipment that we're used to make this possible.

We wish also thank the members of the team and beyond: for the time and effort they provided in judging this work.

My gratitude goes to my team for their continuous encouragement and faith during prolonged failures and successes in pursue our goal further.

Table of Contents

I. Introduction 5
II. Methods and Results 7
A. Motor Description
..... 7
B. The Motor Controller Board
……..... 8
C. Data Flow Structure 10
D. Electrical Flow Structure 11
E. PCB Layouts 12
Discussion 10
IV. Appendices 14

I. Introduction

In the world of automobiles, there are only a finite number of ways to actuate them. When the goal is to minimize the cost of transport and improve the energy efficiency, the most logical selection for propulsion is electric motors. Other methods of propulsion, like gasoline or other related fuel, have major issues preventing energy efficiency and sustainability. A hydraulic engine is extremely powerful, but consumes large amounts of energy and requires continuous running, even if the vehicle is not moving. Gasoline systems require large, heavy storage tanks for compressed fuel, necessitating large air compressors to fill up the tanks and often the energy consumed compressing the air is not considered in the cost of transport. Conversely, electric motors are capable of large amounts of power, and are relatively light weight. A typical electric motor is capable of 200 watts per kilogram, while a typical human muscle is only capable of about 50 watts per kilogram of power.

Electric motors come in two major varieties: brushed and brushless. Brushed motors are the most common and the easiest to use. Brushed motors work by having two magnets in the stator and an electromagnet in the rotor. The motor continues to spin by the electromagnet changing orientation, which is controlled by the brushes that act as contact switches. Brushed motors are very easy to use, and will run by simply applying a dc voltage across the motor leads. The current Electric Vehicles in the college uses exclusively brushed motors, with motor controllers set up for the brushed motors.

There are several issues with brushed motors. The constant contact of the brushes will limit the top speed of the motor and eventually the brushes wear out, necessitating replacement. The brushes also result in sparking and electrical noise, which leads to variation in the motor constants, as observed by the laboratory. Another disadvantage is that the component of the motor that heats up, the electromagnet, is in the rotor and is nearly impossible to cool without opening the motor to blow air through it.

Conversely, brushless motors do not contain brushes. The electromagnet for the motor is in the stator and does not rotate, while the rotor contains rare earth magnets. The changing of the polarity to continue to make the motor spin is controlled by a computer, making the control more precise and efficient, since the computer can factor in

the speed of the motor. Without the brushes, the motor has less frictional losses and less variable motor constants and properties. Also, with the electromagnets in the stator the motor will be much easier to cool, which means the motor will be able to run longer at higher power values without overheating.

The major drawback of the brushless motor is that they are much more difficult to control. A brushless motor requires a computer to sense the position of the rotor and apply the appropriate voltage across the appropriate electromagnet to force the motor to spin. Hence, the goal of this project is to design and build a motor controller board to enable control of a brushless motor and create a motor controller to run the vehicle.

II. Methods and Results

A. Motor Description

The motor used is a Yu Feng, brushless, 1000 Watt, corresponding to the 48 volt motor. The spec sheet for the motor was not found. A picture of the motor is displayed below in Figure 1. The motor is approximately 4.5 kilograms.

Figure 1: Picture of the brushless motor supplied.

As depicted in the figure, the motor has eight wires attaching to it. There is one power wire for each of the three phase windings, one power wire for the Hall effect sensors, one ground wire for the Hall effect sensors, and one signal wire for each of the three Hall effect sensors. The Hall sensors are built into the motor to detect the position of the rotor. The Hall sensors return a high value when rotor is in one orientation and a low when the rotor is 180 degrees away from the high. In actual use, the Hall sensor returns a high for the 180-degree section while the rotor is closest to the sensor and a low

voltage of zero for the rest of the time. The motor uses three Hall sensors, and by determining which sensors are on at any given time, the appropriate orientation of the electromagnetic field can be applied.

B. Motor Controller

The basic principal of the motor controller is to use the Hall effect sensors to determine the orientation of the rotor and apply a magnetic field perpendicular to the rotor magnets. To control the speed of the motor, the voltage applied is Pulse Width Modulated (PWM). It is also necessary to determine the speed of the motor, which can be accomplished by measuring the time for the Hall sensors to go through one full rotation, corresponding to one revolution of the motor. The diagram below shows the possible sequence of MOSFET activity to achieve one full rotation of the rotor. In the diagram, each winding is attached between two MOSFETs, and by controlling which MOSFETs are active controls the flow of current into the motor and ultimately the direction of the magnetic field.

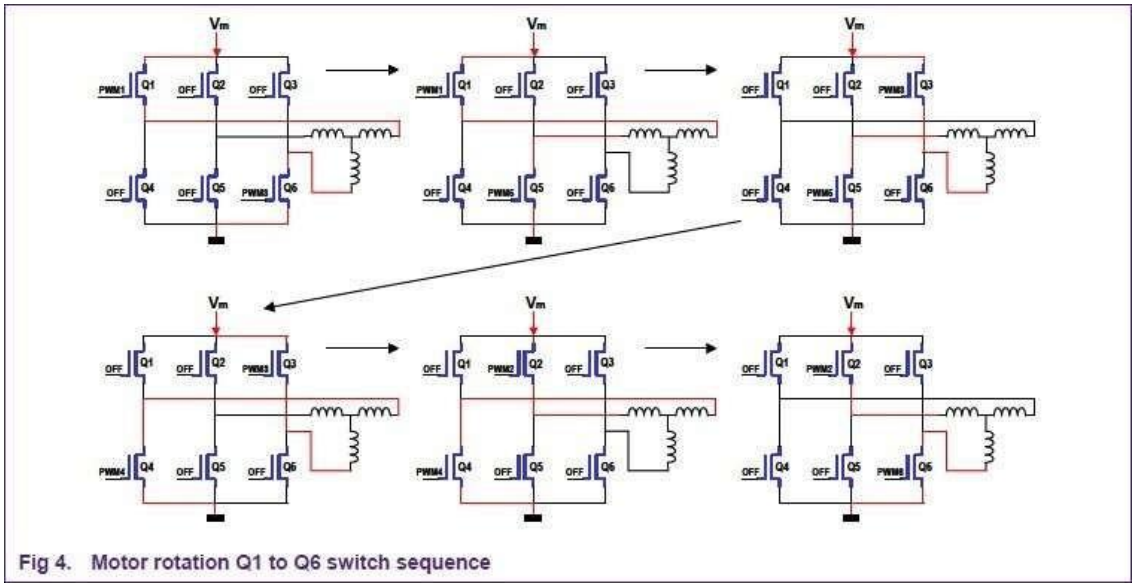


Fig 4. Motor rotation Q1 to Q6 switch sequence

Figure 6: Depicts the flow of current through the MOSFETs and motor windings, as a progression of the rotation of the motor. The figure is from the Brushless DC motor control using the LPC2141 article from the NPX website.

The actual implementation of this strategy utilizes a block commutation scheme provided in the Maxon EC motor guide. This strategy is depicted in Figure 7, below. As

depicted below there are six cases, where the high sensor is: 1, 1 and 2, 2, 2 and 3, 3, 3 and 1. In the code, the variable hall_sensor_code is comprised of three bits: sensor 3 is assigned bit 0, sensor 2 is assigned bit 1, and sensor 1 is assigned bit 2. The actual motor code takes on a switch-case structure, which allows for the six cases listed above, as well as the potential to go in the reverse direction. In each case, it is first determined which winding is getting the voltage applied to it. Next, it is determined which two MOSFETs are set to the on position. For example, phase 1 in the diagram below has Hall sensors 1 and 3 set to on. This gives a value of 101 for hall_sensor_code, in binary, or 5 in decimal. In the diagram, U_{1-2} is on, with the shaded portion on the positive side. This means that the lowside MOSFETs for windings 1 and 2 are set to on, and the lowside MOSFET for winding 3 is off. Also, since the top is shaded, the first number of the subscript has the voltage applied to it, so in the code, winding 1 gets the voltage. This example is depicted as the second state in Figure 6. The specific section of code is as follows:

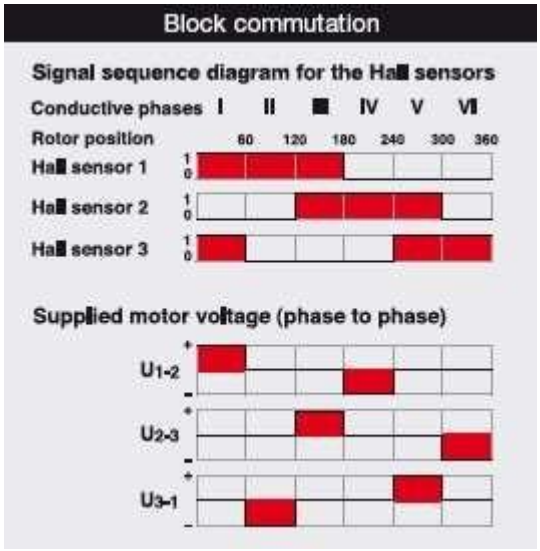
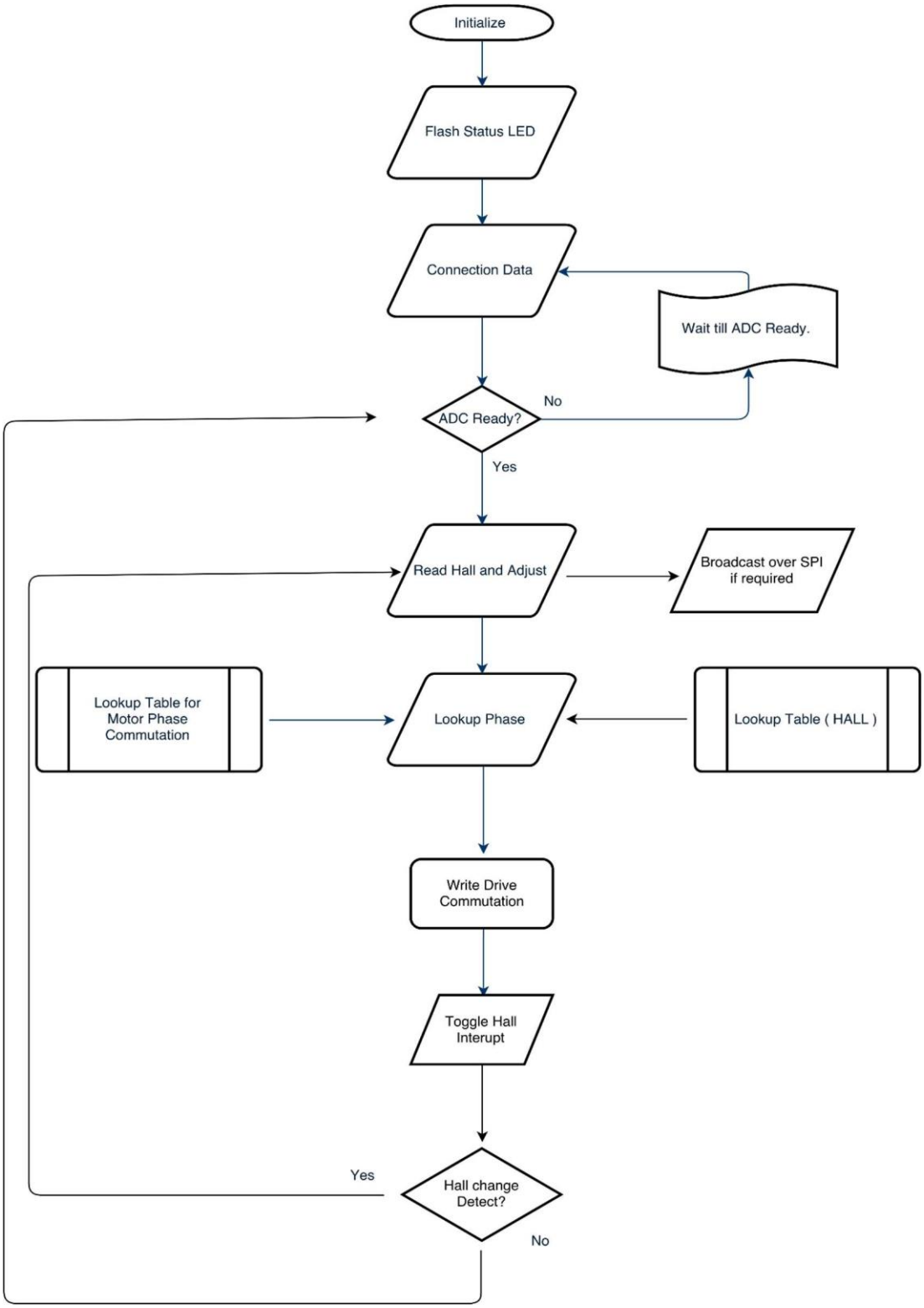
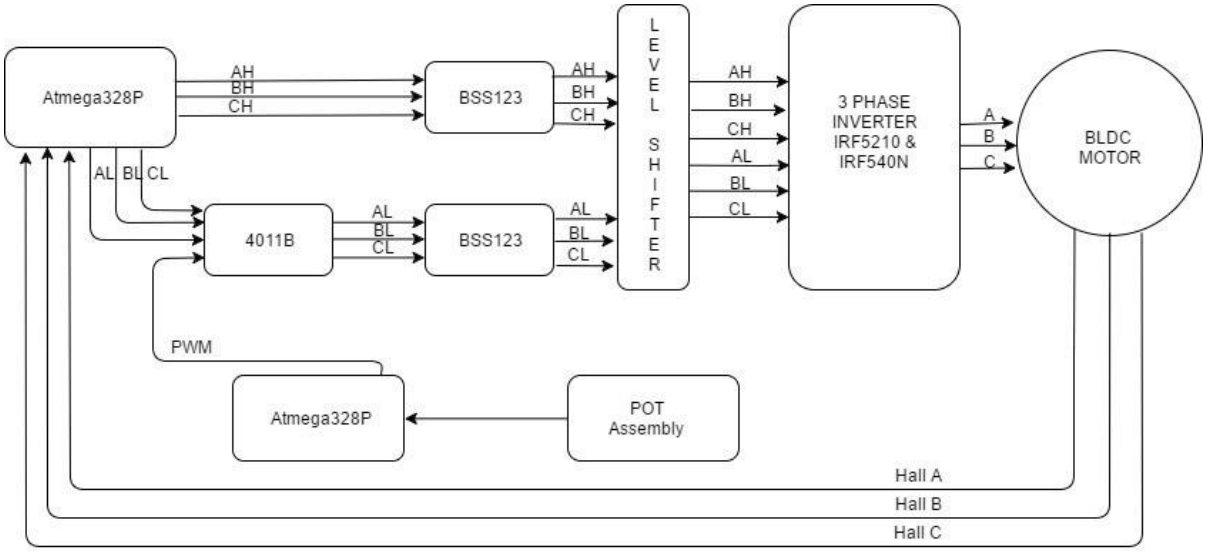


Figure 7: Block commutation scheme from Maxon's motor guide

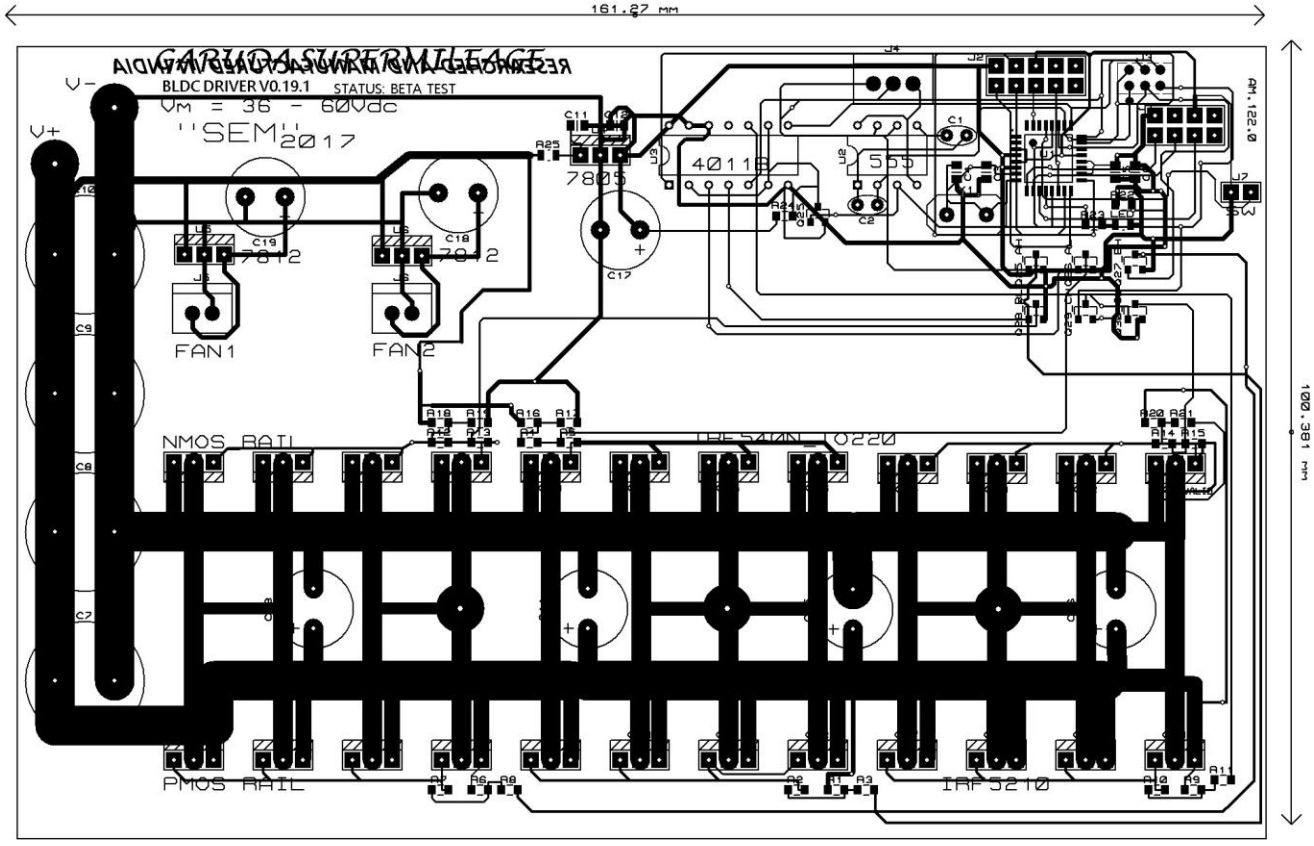
C. Data Flow structure



D. Electrical Flow structure



E. PCB layout.



Prototype V19.1A

**Please find attached the Schematics attached under Appendix.

III. Discussion

Upon the completion of the project, we have a working motor controller for a brushless dc motor; however, the motor controller can be improved. Currently all of the code for the motor controller is in the main loop of the c file. This may cause a problem at high speed, since the motor may be turning faster than the main loop can keep up with. This problem could be solved by putting the code into an interrupt; let the motor run until a hall sensor switches on, then apply the necessary voltage to the necessary windings. This would also be useful for determining the speed of the motor by counting the interrupts. A working motor controller has been made, but many modifications are necessary before the code is good enough to use on a vehicle.

IV. Appendices

Motor Controller Code:

```
volatile uint8_t phase = 0; // motor firing phase volatile long position = 0; // current
position
volatile bool sample_and_hold_pin = false; // flag to tell loop to sample current position
volatile byte position_buffer[] = {0, 0, 0, 0}; // buffer for transfering single bytes of position over spi


void setup() {

// configure motor control outputs
DDRC |= 0x3F;  // A0-A5 as outputs
// configure uC control pins
DDRD &= !0xEF; // 0-3, 5-7 as inputs (will break serial communication and require ISP programmer to
undo)
DDRD |= 0x10;  // 4 as output, used for software pin change interrupt      pinMode(0, INPUT);  // sample
and hold      pinMode(1, INPUT);  // motor direction      pinMode(2, INPUT);  // encoder channel B
pinMode(3, INPUT);  // encoder channel A
pinMode(4, OUTPUT); // used for software initiated pin change interrupt
pinMode(5,OUTPUT); digitalWrite(5, HIGH);      pinMode(6, INPUT);  //
hall sensor 3      pinMode(7, INPUT);  // hall sensor 2      pinMode(8,
INPUT);  // hall sensor 1      pinMode(9,OUTPUT); digitalWrite(9, LOW);
//pinMode(8, OUTPUT); // status LED
```

```
pinMode(10, INPUT);  // pwm pin, master controls this      pinMode(MISO,
OUTPUT);      pinMode(SS, INPUT); // /SS pin
SPCR |= 0b00;       // SPI speed f_osc/4 (4MHz with 16MHz chip)
SPCR &= ~_BV(DORD); // MSB first
SPCR &= ~_BV(MSTR); // slave mode
SPCR &= ~_BV(CPOL); // clock low when idle
SPCR &= ~_BV(CPHA); // sample on clock rising edge
SPCR |=  _BV(SPE);  // enable SPI
SPCR |=  _BV(SPIE); // enable SPI interrupts

// initialize interrupts   initInterrupt();

// setup phase to correct motor position by triggering pin change interrupt a few times
// fills in hall variable in pin change ISR   delay(1);   digitalWrite(4,
LOW);   delay(1);
digitalWrite(4, HIGH);   delay(1);
digitalWrite(4, LOW);

// flash LED thrice on reset   for (int i = 0; i < 6;
i++) {
PINB |= _BV(0);     delay(80);
}
}


void loop() {

static long sample_position = 0;   static bool
sample_and_hold_set = false;

// sample and hold event
if (sample_and_hold_pin == true && sample_and_hold_set == false) {      sample_position = position;
// sample position      position_buffer[0] = sample_position >> 24; // divide sampled position
position_buffer[1] = sample_position >> 16; // into byte sized chuncks      position_buffer[2] =
sample_position >> 8;  // for SPI transfer      position_buffer[3] = sample_position;       //
sample_and_hold_set = true;                // set flag that sample has been taken
PINB |= _BV(0);                          // toggle LED    } else if (sample_and_hold_pin == false &&
sample_and_hold_set == true) {      sample_and_hold_set = false;                // remove sample flag
}

// commutate motor   commutate(phase);
}


void initInterrupt() {
cli();               //disable interrupts while changing settings   PCICR |= 1 << PCIE2;  // set bit 2 in PCICR
for PCINT23:16 (port D)   PCMSK2 |= 0xFF;       // enable pin change interrupts on port D (all pins)
sei();
```

```
}


ISR(PCINT2_vect) { // run every time there is a pin change on port D pin change

// lookup tabel for position incrementing (3-phase version of quadrature)
static int8_t hall_inc[] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, -
1, 0, 0, 0, 0, 0, -1, 0, 0, 1, 0, 0, -1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 1, -1, 0, 0, 1, 0, 0, -1, 0, 0, 0, 0, 0, -1, 0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0};

// lookup table for setting next motor phase, extended to allow reverse direction
static uint8_t phase_lookup[] = {0, 2, 4, 3, 6, 1, 5, 2, 4, 3};
static int hall = 0;   static uint8_t dir
= 0;
static bool sample_and_hold_set = false;

int port = PIND; // read the port

sample_and_hold_pin = port & 1; // flag to tell loop to store current position in buffer
dir = (port >> 1) & 1; // direction value, 1 or 0   hall = hall << 3;      // preserve last read
for position in/decrement
hall |= port >> 5;     // shift to read only hall sensors

if (dir == 0) {
phase = phase_lookup[(hall & 0x07)];     // determine next phase to fire on the motor   } else {
phase = phase_lookup[(hall & 0x07) + 3]; // adding 3 to lookup index has the effect of reversing the motor
(MAGIC!)
}    position += hall_inc[hall & 0x3F]; // use <hall_prev><hall_current> as lookup index to which will
increment, decrement or do nothing to position value

PINB |= _BV(0);                          // toggle LED
}



ISR(SPI_STC_vect) { // spi handler
byte position_byte = SPDR; // read in command from master (which byte of position to send next)
SPDR = position_buffer[position_byte]; // send a single byte from position_buffer

}


void commutate(uint8_t _phase) {
```

```
//                          {lowers on, Bh:Al,  Ch:Al,  Ch:Bl,  Ah:Bl,  Ah:Cl,  Bh:Cl}
static uint8_t phase_to_port_ABC[] = {0b000000, 0b010001, 0b100001,
0b100010, 0b001010, 0b001100, 0b010100};
static uint8_t phase_to_port_ACB[] = {0b000111, 0b100001, 0b010001,
0b010100, 0b001100, 0b001010, 0b100010};
static uint8_t phase_to_port_BAC[] = {0b000111, 0b001010, 0b100010,
0b100001, 0b010001, 0b010100, 0b001100};
static uint8_t phase_to_port_BCA[] = {0b000111, 0b001100, 0b010100,
0b010001, 0b100001, 0b100010, 0b001010};
static uint8_t phase_to_port_CAB[] = {0b000111, 0b100010, 0b001010,
0b001100, 0b010100, 0b010001, 0b100001};
static uint8_t phase_to_port_CBA[] = {0b000111, 0b010100, 0b001100,
0b001010, 0b100010, 0b100001, 0b010001};

// the order of the 3 motor wires will dictate which of the above lookup tables to use.

PORTC = phase_to_port_ABC[_phase];

}
```