Contents

Designing an open-source brushless motor controller, supporting sensored and sensorless commutation

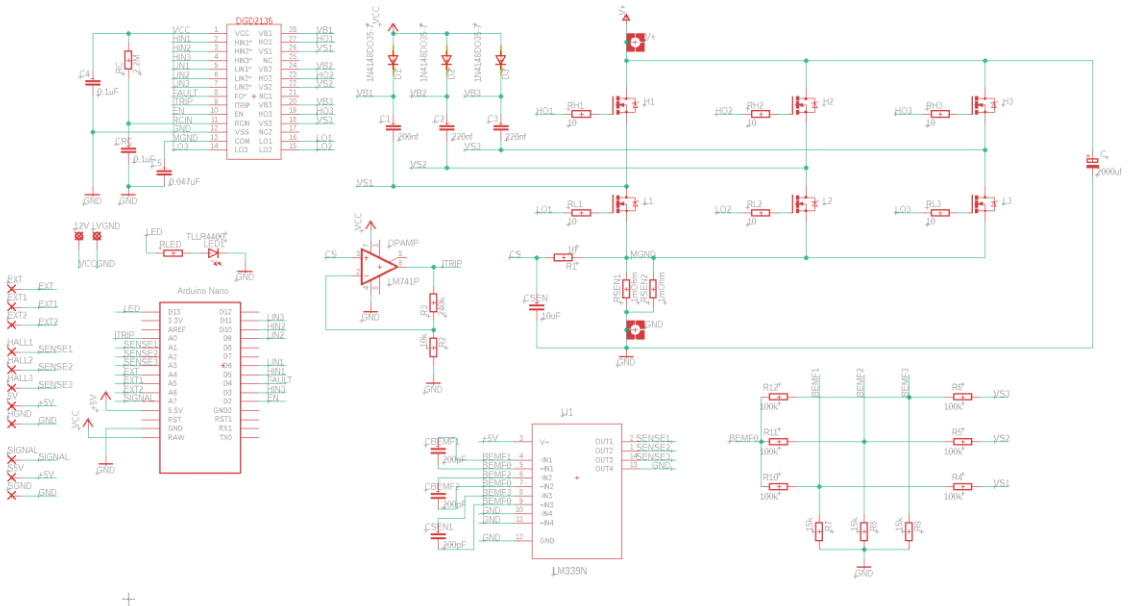
Brushless motors are a form of synchronous motors which allow its permanent magnet rotors to rotate without the use of brushes. This therefore enables them to be highly efficient, have a long lifespan and have an impressive weight to power output ratio. However, the drawback of using such motors is that they cannot be powered as simply as brushed dc motors, in which a dc power supply is applied across the motor's terminals. Brushless motors require a complicated controller, commonly referred to as an ESC (Electronic speed controller), in order to sequentially switch dc currents across the motors three windings at specific times in order to rotate the motor. Fortunately, due to the widespread availability and cost reduction of microcontrollers and MOSFETS, brushless motor controllers have never been more accessible, especially for hobbyists.

In this documentation I will explain the design decisions and considerations of brushless motor controller with the following requirements:

- It will support sensored commutation by enabling a brushless motor's hall sensors to be utilised to sense the rotor's current rotation.
- It will support sensorless commutation by detecting the back EMF generated by the motor windings in order to determine the rotation of the motor's rotor.
- It will support of an input voltage up to 42v, allowing up to a 10s lithium battery to be used.
- It will support a maximum instantaneous discharge current of around 70 amps.
- It will support a continuous discharge of 30 amps.
- Most of the components will be through hole, enabling a novice electronics hobbyist to easily construct the motor controller.
- The speed controller will be cheaper than most commercial alternatives offering the same specifications (especially when the parts are ordered from Chinese distributors).

Hardware

The schematic

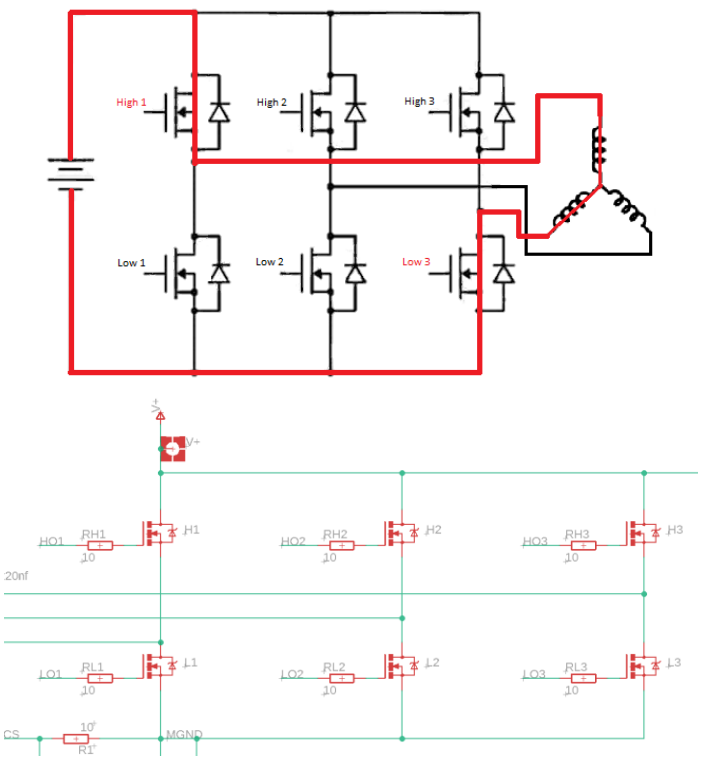


The speed controller's schematic can be observed in Figure 1. This includes all the components that will be used in order to construct the speed controller, however certain components can be omitted if only sensored commutation is required (this will be discussed later in the documentation). The schematic consists of four primary areas:

- The microcontroller (an Arduino Nano)
- The triple half bridge MOSFET driver IC (an IR2136 or DGD2136)
- The triple half bridge MOSFET configuration
- The back EMF detection circuitry

The Tripple half bridge MOSFET configuration

This arrangement of the MOSFETs enables any one of the three motor's phases to be connected to either the supply voltage or to ground. By connecting one of the phases to the supply, and another to ground, a current is conducted through specific motor windings in a specific direction. The energised windings subsequently apply a force to the motor's rotor through an attraction/repulsion force with the rotor's permanent magnets.



The MOSFET of choice is the IRF3205. This MOSFET boasts a maximum drain-source voltage of 55v and a maximum continuous drain current of 98A. This MOSFET subsequently supports our voltage requirement of 42v and can handle high continuous currents as long as the MOSFETs are sufficiently cooled.

6 step commutation

In order for a brushless motor to spin most efficiently, the motors windings must be energised in a specific order, and at specific times. This ensures that the motors torque is largest and that the current direction of rotation is maintained. The windings to be energised in order for a four-pole motor to rotate anti-clockwise can be observed in Figure x.

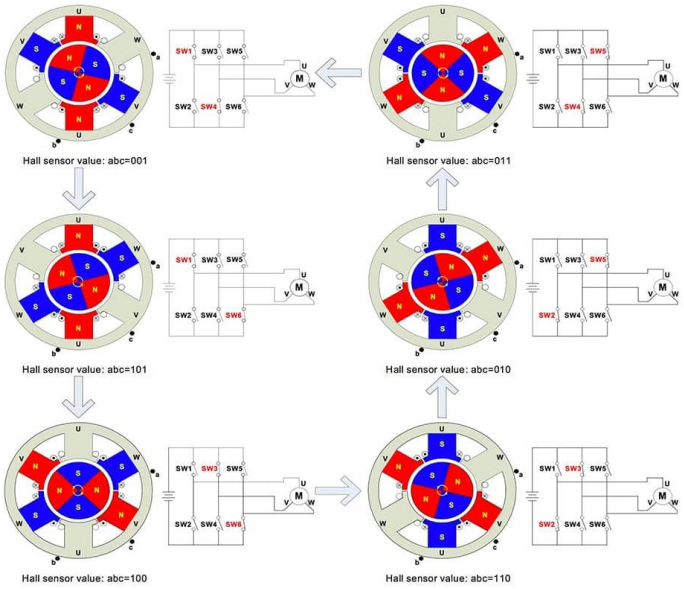
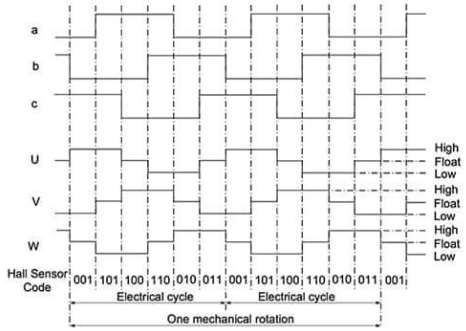


Figure 1 https://www.digikey.com/en/articles/how-to-power-and-control-brushless-dc-motors

Figure x demonstrates that there are 6 combinations of two MOSFETS to be activated in order for one electrical revolution to be performed (a half revolution for a 4-pole motor). Therefore, if these 6 steps are repeated, the rotor will continuously rotate. In order to adjust the speed of the motor's rotation, a PWM waveform can be applied to the MOSFETs. The motor's speed will adjust proportionally to the duty cycle applied to the MOSFETs; which is due to the fact that a motors speed is proportional to the average voltage applied across the motors windings.

Sensored commutation theory

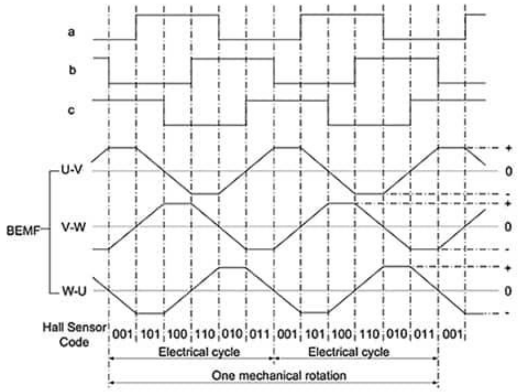
As explored above, the position of the motor's rotor is required in order to determine the two MOSFETs to be switched. One common approach of detecting the rotor's current rotation is through the use of hall sensors integrated within the motor. These hall sensors are spaced either 60 or 120 degrees around the motors stator and detect the presence of the magnetic field from the motor's rotor. The majority of the hall sensors output a 5v logic signal in order to indicate the magnetic pole that is present. These hall sensors are strategically spaced in order to output three logic signals which correlate to the windings that need to be energised in order to continue the motors rotation. A table showing the windings to be excited with the corresponding hall signals can be observed in Figure x.



Sensorless commutation theory

The sensored approach highlighted above however has the disadvantage of adding additional cost and complexity to the motor. To mitigate this downside, an alternative method of detecting the

motors rotation has been devised and is used extensively with hobbyist motors. This sensorless commutation approach relies on the fact that as the rotor rotates after a current is applied to two of the motor's phases, the rotors permanent magnets pass by the motors un-excited winding. This action of passing a permanent magnet near a winding results in the generation of an EMF. As this winding is currently un-energised, the EMF can be sensed by the speed controller and can be correlated to the current position of the rotor. The waveform generated by the back EMF can be observed in Figure x below:

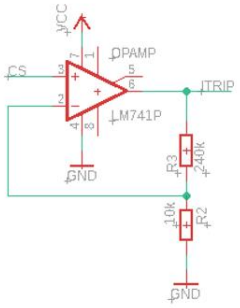


This diagram demonstrates that the step of the 6-step commutation is to be advanced when the BEMF of the floating phase crosses the zero point. Subsequently, this zero-crossing event can be monitored for each specific step of the 6-step commutation in order to continuously rotate the rotor.

This commutation method however consists of its own downsides. The primary downside is that the magnitude of the BEMF waveform is proportional to the rate of change of the magnetic field across the motors winding. This therefore means that at a low rotational speed, the BEMF waveform is hard to detect; and at a standstill there is no BEMF waveform at all. Therefore, the windings need to be initially energised in a sequence that is not correlated to the rotors position, until enough movement is created for a BEMF signal to be observed.

Current sensing

In order for the current draw of the BLDC controller to be measured, a shunt resistor is utilised. The voltage across this shunt resistor is amplified and measured in order to accurately determine the amount of current passing through the motor. This ensures that in situations in which the motor has stalled or is drawing more current than the BLDC controller is designed for, the PWM signal applied to the MOSFETs can be stopped in order to prevent damage. A simple non-inverting amplifier allows the small voltage across the shunt resistor to be amplified to a more appropriate voltage range for interpretation for a microcontroller or IC.

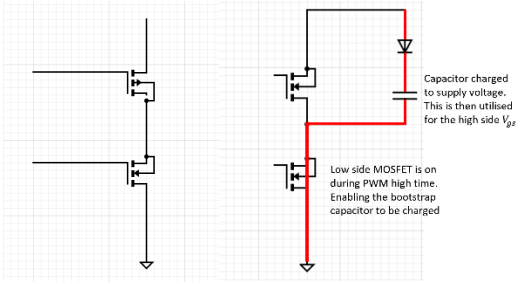


Bootstrapping

Theory

Each half bridge utilises two N channel MOSFETs. Initially it may seem counter intuitive to utilise two N channel FETs due to the fact that the high side MOSFET would need a very high gate voltage, relative to the circuits ground, in order to sufficiently activate the MOSFET. This is due to the fact that there will be a voltage across the motor's windings, therefore the high side MOSFET will require a Vgs (with respect to ground) which is equal to the MOSFETS saturation voltage plus this voltage drop across the windings. Therefore, the high side MOSFETs gate voltage must be higher than its drain voltage, meaning that this voltage will most likely have to be provided from a source external to this circuit. An alternative to using two N channel MOSFETs would be to use an N channel for the low side MOSFET and a P channel for the high side. However, using only N channel MOSFETs not only reduces the cost of the circuitry, but it also results in a reduction of the losses due to the inherent attribute of P channel MOSFETs consisting of a higher Rdson resistance.

A solution which enables the utilisation of two N channel MOSFETs in a half bridge configuration is referred to as MOSFET bootstrapping. Bootstrapping enables a capacitor, connected between the circuits supply and the high side MOSFETs drain, to be charged when the low side MOSFET is activated. The voltage across this charged capacitor can then be used in order to provide a sufficient voltage (relative to the high side MOSFETs source) to the gate of the high side MOSFET.



Implementation

An 1N1418 diode has been used as the bootstrap diode due to its high speeds; this is necessary as the bootstrap capacitor will need to be sufficiently charged within the very small timeframe of the low side MOSFETs PWM on time.

A specific bootstrap capacitance is required to be tailored around the PWM frequency used to drive the MOSFETS and the MOSFETS gate capacitance.

The charge that must be stored by the bootstrap capacitor is expressed in the following equation:

$$Q_{CB} = Q_G + (t_{Lmax} \times I_B)$$

Where:

- Q_{CB} is the total charge to be stored.
- Q_G is the MOSFETs typical gate capacitance.
- t_{Lmax} is the maximum time that the high side MOSFET PWM waveform will be high.
- I_B is the maximum quiescent supply current stated in the MOSFET driver's datasheet.

In the case of this speed controller:

- The IRF3205 has a gate charge of 146nC, therefore $Q_{CB} = 146nC$
- A 31kHz PWM frequency is utilised so $t_{Lmax} = 32\mu s$

- The IR2136's maximum quiescent Vbs supply current is $I_B = 70 \times 10^{-6}$

$$Q_{CB} = 147 \times 10^{-9} + (32 \times 10^{-6} \times 70 \times 10^{-6}) = 1.4924 \times 10^{-7}$$

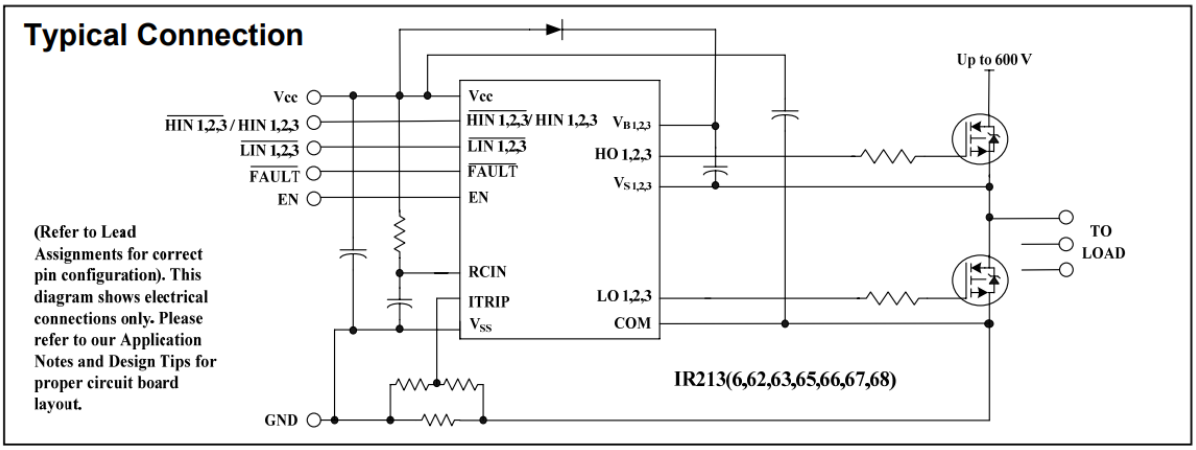
Finally, we need to tailor the value of C_B around the maximum allowable ripple. In this case I will allow a 5% ripple of the 12v supply.

$$C_B \geq \frac{Q_{CB}}{\Delta V}$$

$$\frac{1.4924 \times 10^{-7}}{\Delta V} = 248\ \text{nF}$$

The MOSFET driver IC

The IR2136 enables the Arduino's 6 PWM signals to be utilised in order to switch the 6 MOSFETs within the triple half bridge configuration. Using a MOSFET driver is preferable to simply using the Arduino's logic level PWM signals due to the fact that the Arduinos low voltage and high impedance output results in considerably slower switching times, consequentially causing MOSFET heat losses. This IC supports a supply voltage of 10-20v; I will be powering it used an automotive switching dc regulator with an output of 12v.



Pin descriptions

Pins	
HIN1, 2, 3 LIN1, 2, 3	The HIN pins receive the PWM signal for the high side MOSFETs of each half bridge and the LIN pins receive the PWM signals for the low side FETs. Depending on the variant of the ir2136, these inputs can be inverted. In my case I am using the ir2136S which has both the HIN and LIN signals inverted.
FAULT	The fault pin indicates an overcurrent or undervoltage event in the form of an open drain signal. This is connected to the Arduinos digital input pin 4 which is in pullup mode. The fault signal is inverted.
RCIN	A pin which connects to a series RC network in order to define the amount of time for the FAULT signal to clear. This time is approximately equal to R*C.
ITRIP	This is the analog input for the overcurrent detection. When the voltage at this pin exceeds 4.3V the chips operation is halted and the FAULT signal output. For this application a simple amplifier is used in order to amplify the voltage drop across the shunt resistor. The gain of the amplifier is strategically chosen in order to result in a 4.3v output when the maximum current is reached.
EN	A logic input to enable the ir2136 to operate.

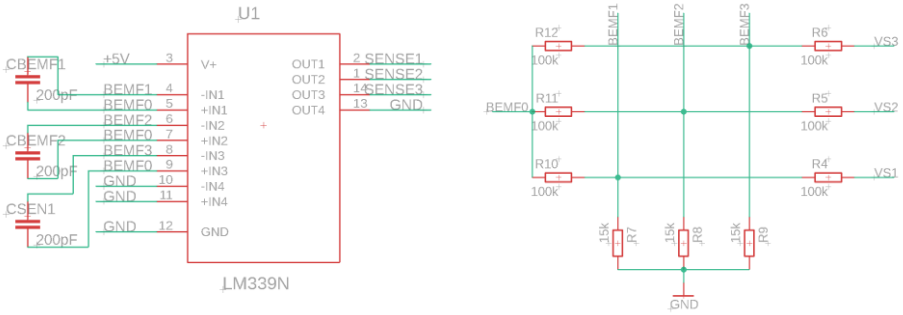
HO1, 2, 3 LO1, 2, 3	The output pins in order to drive the high and low side MOSFETs. This output logic signal will consist of the same PWM waveform applied to its corresponding input pin, however depending on the ir2136 variant it may be inverted.
VB1, 2, 3 VS1, 2, 3	The VB pin connects to the node attaching the bootstrap capacitor to the supply voltage. The VS pin connects to the node connecting the bootstrap capacitor to the drain of the high side MOSFET. These pins enable the bootstrap capacitor to be charged and discharged in order to drive the high side N channel MOSFET.

Deadtime

The IC provides a typical dead time value of 290 nanoseconds. This means that there will be a 290ns delay between one of a half bridges MOSFETs being turned off and the other MOSFET being turned on. This minimises the risk of both MOSFETs being activated at the same time, which would result in a direct short circuit between the supply and ground through both MOSFETs.

Sensorless commutation hardware

An effective but simple method of measuring the BEMF zero crossing events (highlighted in Sensorless Commutation Theory) is to use a resistor network composed of nine resistors, this can be observed in the Figure below.



Resistors R6, R5 and R4 are connected to the three motor phases. R7, R8 and R9 create a potential divider with R6, R5 and R4, allowing the voltage from each phase to be attenuated to within a 5-volt range. The attenuated voltages are then combined at the BEMF0 node in order to simulate the motors neutral point (a point where a brushless motors phases join internally when in a Y configuration but is usually inaccessible externally). This point now allows us to sense the voltage across each of the motors windings by measuring the phase voltage to this virtual neutral point. This subsequently means that the BEMF induced within the motor's windings can be detected. As we are only concerned as to whether a BEMF zero crossing event has occurred, an LM339N 4 channel comparator has been utilized. This comparator outputs a logic signal corresponding to whether the voltage across a phase winding is above or below the virtual neutral voltage. Therefore, by inspecting these signals for a change in states, we can determine that a BEMF crossing event has occurred for a particular phase, and therefore advance the commutation sequence step.

A capacitor is required in order to smooth each attenuated voltage signal in order to reduce any high frequency noise superimposed on the BEMF signal. Any superimposed noise can cause lots of bouncing and erroneous state changes to be detected. This capacitor is connected between the virtual neutral point and each of the phases and consists of a value of 200 pF.

The microcontroller

The brains of the speed controller are in the form of an Arduino Nano, which is a breadboard-friendly board based on the ATMega328 microcontroller. These boards are widely used within the hobbyist electronic space and are incredibly cheap, especially when purchased in the form of a third-

party clone. This microcontroller typically runs at 16MHz which is relatively slow compared to other microcontrollers used to control BLDC speed controllers. Therefore, the nano must be programmed as efficiently as possible to make use of the limited processing power. As a result, most of the advantages provided by the Arduino framework are not utilised and port/register manipulation was an effective programming tool.

Timer utilisation and timekeeping

The ATMega328 only has three hardware timers, all of which are used to generate the PWM signal for two of the 6 PWM pins. As the BLDC controller requires 6 PWM signals to the MOSFETs, all of the timers were already being used and therefore I could not have a timer dedicated to the task of timekeeping. The PWM frequency is set to 32kHz, and the timer resolution restricted to 8 bits, subsequently meaning that the timers value would increment and overflow extremely fast, far too fast to utilise the timer's values for timekeeping. This issue was mitigated by adding an overflow interrupt to one of the timers, in which a global 16-bit integer is incremented at each overflow event, resultantly giving me a 32kHz timer. This timer variable would take around two seconds to overflow which is a perfectly reasonable time range.

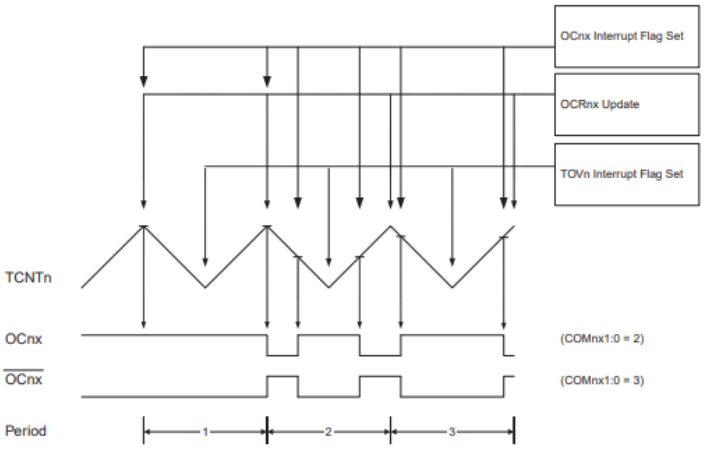
The main requirement for timekeeping is so that there is a timeout for the speed controller being in one of the commutation steps. If the controller is in one of the commutation steps for an unordinary amount of time, then this suggests that the motor has stalled, or some other fault has occurred. As a result, the PWM duty cycle for the MOSFETs needs to be set to 0 in order to prevent a sustained high current draw. This timeout value is set to 1072 clocks of the make-shift timer, corresponding to around 32 milliseconds.

Generating the PWM waveform

The PWM frequency used to generate the PWM waveform for the MOSFETs is 32KHz. This PWM frequency is rather high, however it means that the speed controller will happily support motors that consist of high pole counts and spin at a high RPM. By default, the Arduino supports a PWM frequency of 500hz. This frequency is far too small for this application, therefore it needed to be increased by manipulating the registers that determine the timers prescaling.

The prescaler was set to 1 and the timers resolution set to 8 bits, meaning that the overflow frequency is $\frac{16000000}{256} = 62500$. Phase correct PWM was then applied by manipulating the timer mode registers. This means that the timer would first increment until its maximum value before decrementing back to its lowest value, resulting in a PWM frequency of 31250. The two Output Compare Registers are then set in order to toggle the PWM pins output at each matching comparison; thus, generating a PWM waveform which can be observed in the figure below.

Figure 18-7. Phase Correct PWM Mode, Timing Diagram



Sensored logic flowchart

The logic for sensored operation is rather simple. The throttle is sampled every 20 milliseconds, in which the throttle is mapped to an 8-bit range before being assigned as the PWM duty cycle. The hall sensors logic levels are read before being utilised in order to determine the commutation step that should be currently applied. A timeout is used in order to ensure that a commutation step is not applied for too long, as this suggests the motor has been stalled or the wrong commutation step is being applied.

Sensorless logic flow chart

The sensorless logic is slightly more complex. As the magnitude of the BEMF signal is proportional to the speed in which the motor is running, this means that there will not be any way to determine the rotors position while stationary. As a result, we are required to perform open loop commutation in order to get the motor to briefly move and generate a BEMF signal. As can be observed in the flowchart, the six-step commutation sequence is performed, with 5 millisecond intervals, a total of 12 times. Once this has been completed, the microcontroller attempts to detect a phase 3 BEMF event (which would be the next BEMF event to be observed). If this is successfully detected then this means that the rotors position has been determined and we can perform closed loop, sensorless commutation.

Once the rotor is spinning, the sensorless commutation is rather similar to sensored commutation. During each commutation step, the microcontroller will monitor the BEMF channels until it detects the expected rise or fall signal for that specific commutation step. This indicates that the commutation step is to be advanced, and the process is continued.

Reflections, issues and improvements

To reflect on the speed controller as a whole, I am very satisfied with the outcome, and I believe that it meets all of my initially stated requirements. I have utilised this brushless speed controller to drive a large range of brushless motors (from small hobbyist sensorless motors to huge 1.5kW+ rated sensored motors). The sensorless commutation has greatly exceeded my expectations and is able to very reliably start and continuously rotate sensorless brusless motors. I am very pleased with the PCB and the general package of the final product, especially as the PCB is able to be utilised for both sensored and sensorless applications.

I have occasionally run into issues where two of the MOSFETs from a single phase fail and develop a short. This has only occurred when running with relatively high loads and at voltages above 36v. I

believe that this issue could be a result of possibly large BEMF voltage spikes; of which are exceeding the ratings of the MOSFETs and resulting in its failure. Another possible cause of its failure could be an insufficient bootstrap capacitor, in which it is not able to contain enough charge to fully open the gate of the high side MOSFET. I will continue to investigate and rectify any issues found with this speed controller.

If I were to completely resign this speed controller then I would definitely utilise another microcontroller. As briefly explored in this documentation, the AtMega328p is not a very suitable microcontroller for this role. The primary reasoning behind this would be its limitation on speed and the availability of timers. I would also design the PCB to be able to support multiple parallel MOSFETs for each side of each phase. Such a feature would result in a dramatic reduction of MOSFET heating and would enable the speed controller to tolerate much higher currents. This would be due to the fact that the total MOSFET resistance would half and any heat as a result of this resistance would be distributed over multiple MOSFET packages.