International
IOR Rectifier

APPLICATION NOTE

Application Note AN-1187

IR3230 Sensorless BLDC Motor Drive

By Alex Lollio

Table of Contents

Introduction

The IR3230 is a three-phase brushless DC motor controller/driver with many integrated features. They provide large flexibility in adapting the IR3230 to a specific system requirement and simplify the system design.

Purpose of this document is to provide all the information for realizing a sensorless 3-phase DC motor driver using IR3230 and Microchip PIC16F1937.

To rotate the BLDC motor, the stator windings should be energized in a sequence. It is important to know the rotor position in order to understand which winding will be energized following the energizing sequence. Rotor position is sensed using Hall effect sensors embedded into the stator.

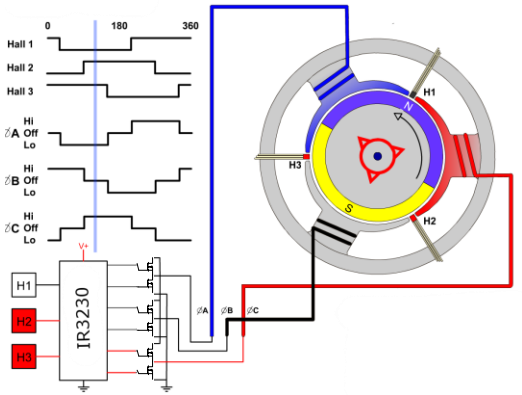


Figure 1: **IR3230 typical connection to a BLDC motor with Hall effect sensors embedded in the stator**

Most BLDC motors have three Hall sensors embedded into the stator on the non-driving end of the motor. Whenever the rotor magnetic poles pass near the Hall sensors, they give a high or low signal, indicating the N or S pole is passing near the sensors. Based on the combination of these three Hall sensor signals, the exact sequence of commutation can be determined. Figure 1 shows the typical connection of the IR3230 to a 2-pole BLDC motor with Hall sensors. Based on the physical position of the Hall sensors, there are two versions of output. The Hall sensors may be at 60° or 120° phase shift to each other. This job is referred to a 60° phase shift configuration.

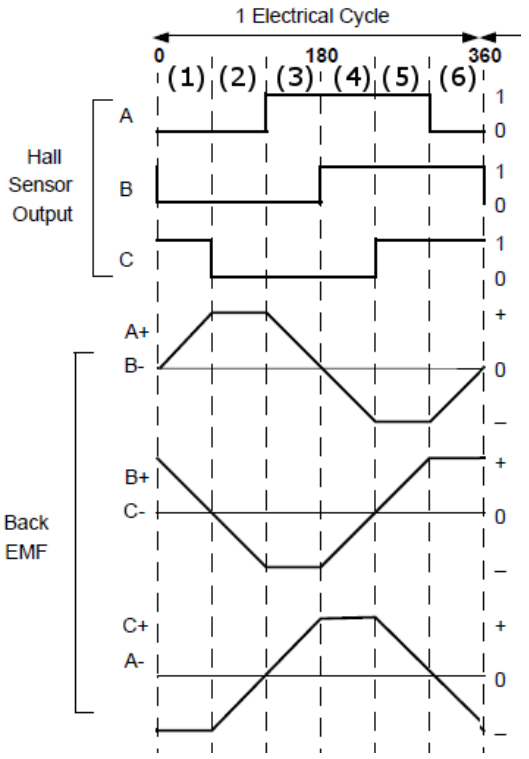


Figure 2: **Hall Sensor Signals and Back EMF**

In more and more applications we need to drive a BLDC motor with no Hall sensor embedded into the stator. In this condition we have to find a way to generate the equivalent digital word that the Hall sensors would generate if present. When a BLDC motor rotates, each winding generates a voltage known as back Electromotive Force or back EMF, which opposes the main voltage supplied to the windings according to Lenz's Law. The polarity of this back EMF is in opposite direction of the energized voltage.

Figure 2 shows an entire electrical Cycle of a BLDC motor where we can see the Hall sensor output and the generated Back EMF.

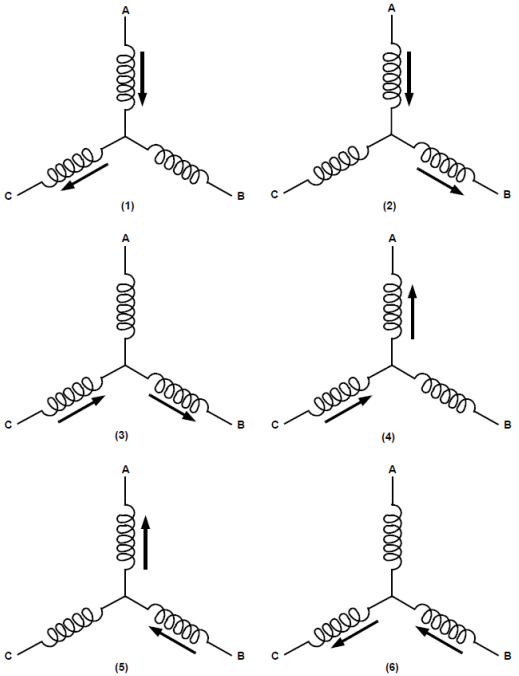


Figure 3: **Winding energizing sequence in respect to the Hall Sensor**

Figure 3 shows the driving sequence in respect to the Hall sensor positions of Figure 2. This is known as the six-step-sequence.

The goal of this work is to sense the Back EMF signal in order to generate the equivalent Hall Sensor digital signals and provide them to IR3230 for the motor driving.

Basic Working Principle

Figure 4 is a simplified schematic diagram of how a sensorless, 3-phase brushless motor is configured with the drive and control circuitry. Each phase of the motor is connected to three circuits:

• a FET driver to the motor supply,

• a FET driver to the motor supply return, and

• a voltage divider.

The voltage divider is necessary to reduce the phase voltage down to a range acceptable to the microcontroller

input. During each commutation period the microcontroller provides a sensor code to IR3230 in order to drive one motor phase high, one motor phase low, and compares the third undriven motor phase to a fourth voltage divider connected to the motor supply.

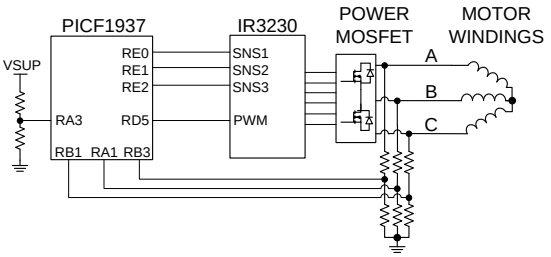


Figure 4: **Simplified BLDC Drive and Control Circuitry**

Voltage to the motor from the motor supply is varied by pulse-width modulating the driver FETs. Only low side supply return is modulated by a PWM signal provided by the Micro to IR3230. The microcontroller comparator is used to determine when the undriven phase voltage reaches the mid-point between the motor supply and motor return potentials. Since there are three phase voltages and one reference, this requires multiplexing each of the three phase outputs to one input of the comparator while the other comparator input remains on the fixed reference.

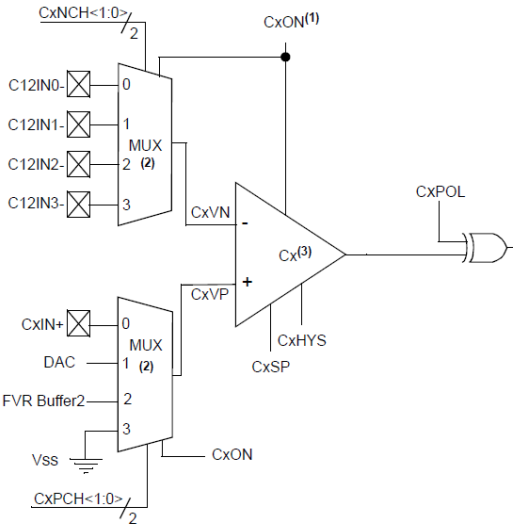


Figure 5: **Microcontroller Comparator Module**

Figure 5 shows a block diagram of the comparator configuration. As seen in the diagram, the non-inverting input can be configured to sense one I/O pin, and the inverting input can be configured to sense one of four I/O pins. Sensorless brushless motor control needs four

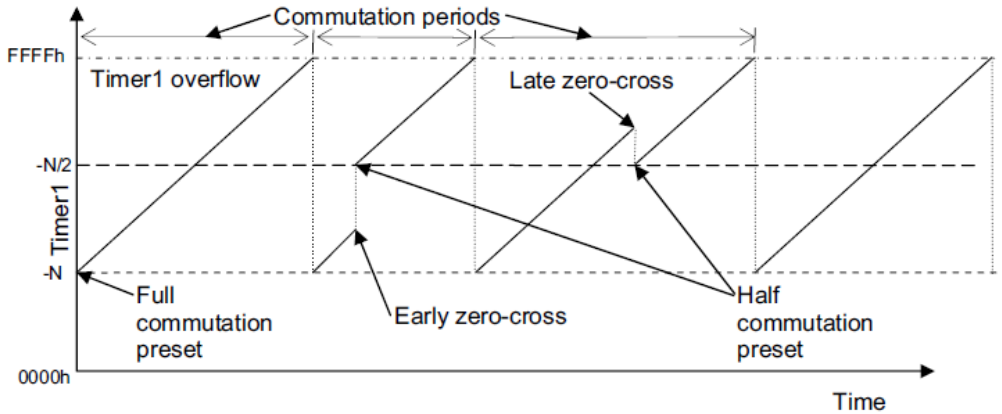
comparator connections: the single non-inverting comparator input is connected to a voltage divider off of the high motor supply rail, and a voltage divider on each motor terminal is connected to the input MUX of the inverting comparator input. Two bits in the comparator control register select which motor terminal is directed to the inverting comparator input. At each commutation state, when the high and low motor drivers are configured, so are the comparator inputs, so that the floating motor terminal can be compared to the fixed reference.

Motor Control

The motor is controlled by synchronizing the commutation with both the motor position and the motor speed. Motor speed at various supply voltage and load conditions is a function of the motor design. The control algorithm searches for this intrinsic speed and adjusts the commutation period to match. The control algorithm is similar to a Phase Lock Loop (PLL) in that error between the commutation period and what the motor needs is computed and added back into the commutation period. The error eventually accumulates to zero. The biggest difference between motor control and other Phase Lock Loop systems is that the motor control becomes discontinuous at any commutation rate above the ideal rate. In other words, the motor cannot keep up at any commutation rate above the ideal rate. This discontinuity results in an abrupt loss of lock and causes the motor to stop abruptly. The control algorithm must avoid crossing this discontinuity boundary by instantly responding to any condition that may cause this breech. This is accomplished by resetting the time to the next commutation at each zero-crossing event. By definition, zero-crossing occurs in the middle of the commutation period so the time to the next commutation is set to one half of the last computed commutation period. If the commutation period becomes too short, then the commutation will be early, resulting in a late zero-crossing event. When this occurs, then the current commutation period is instantly extended by the amount of the delay. If the commutation period becomes too long, then the zero-crossing event will occur early and the current commutation will be shortened. This keeps the commutation cycles tightly coupled to the motor position and allows for a narrow bandwidth error feedback loop for better stability.

Commutation

When commutation is synchronized with the motor position then the voltage of the undriven phase transitions through the point at which it is equal to half the motor supply voltage at the mid-point of the commutation period. This is sometimes referred to as the zero-crossing event. The supply voltage must be applied to the driven phases to bias the undriven phase to the proper level for zero-crossing detection. The control algorithm measures the time from commutation to the zero-crossing event and computes the difference between the actual and expected as the error. In any case, the time from the zero-crossing event to the next commutation is always half the uncorrected commutation period. In other words, even if the zero-crossing is detected immediately after commutation, the time to the next commutation will be half the previously calculated commutation time. Timer1 is used to both measure the time to the zero-crossing event and to time the commutation events. Figure 2 shows a graphical representation of this. The commutation time is N. Timer1 is preset to –N so that it overflows after N counts. At the zero-crossing event, the Timer1 count is captured; let's call this time X. The time to zero-crossing can then be calculated as X – (-N) or X+N. At the zero-crossing event Timer1 is also preset to –N/2, which will



cause an overflow, thereby triggering the next commutation one-half commutation period later.

Zero Cross Detection

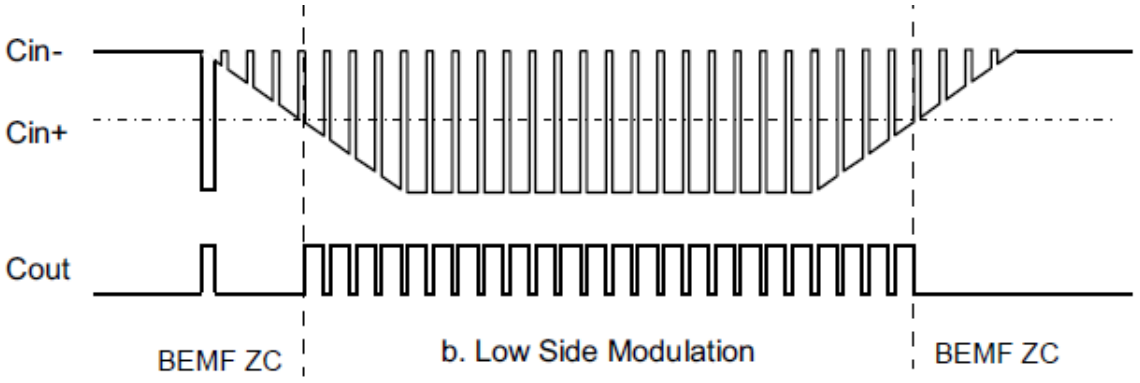
The zero-crossing event cannot be detected on every commutation period because of the nature of the drive and detection circuitry. Consider Figure below (Low Side Modulation) which shows one phase of the motor drive waveform for a low-side modulated drive. In the Figure, a dotted line represents the comparator reference input, which in our case, is half the motor supply voltage. The motor terminal voltage is represented by the waveform and is on the other comparator input. For the low-side modulated system, when the motor terminal BEMF is falling, the comparator output is steady in the first half and transitions in the last half of the commutation. Clearly, it is easier to detect the zero-crossing event at the first comparator change than it is to detect when the comparator stops changing. For this reason, the zero-crossing event is detected only on falling BEMF periods on low-side modulated systems. Since the BEMF alternates between rising and falling in each commutation cycle, zero-crossing is detected every second cycle. The periods in which the zero-crossing events are not detected are commutation only. The commutation-only interval is also when the new commutation time is computed using the zero-crossing event time captured during the previous commutation and zero-cross period.

Commutation States

There are six commutation states in each electrical revolution. Each state generates a digital code and send it to IR3230. IR3230 drives one motor phase high and one motor phase low. The third undriven motor phase is directed to the comparator inverting input through a voltage divider. These three actions require setting the PSTRCON register for the modulated drive, a PORT latch register for the unmodulated drive, and the CCP1CON register for comparator BEMF detection.

Motor Control Code

When the motor is running the motor control code has two major functions: status monitoring and motor control interrupts. The overall view of the motor control code is shown in Figure 6. The main System Service loop, in addition to status monitoring, controls the start-up sequence. Three interrupt types are enabled to control motor operation: Capture, Timer1 and Comparator. Capture interrupt is enabled just at the start up and it is used to capture the initial motor speed value. Timer1 interrupts are always enabled and invoke each and every motor commutation. Comparator interrupts are only enabled every second commutation period to capture the zero-crossing event. We will start analyzing the System Service routine and then the Commutation and Zero Cross interrupt routine (this refers to normal operation mode when the motor is in steady state). At the end we will see the starting procedure which describes how to lock the phase when the motor is already running (Capture Interrupt). This code does not implement a starting procedure when the motor is completely stopped. This can be easily added to the system service routine as discussed in the Microchip AN1305 Application Note.



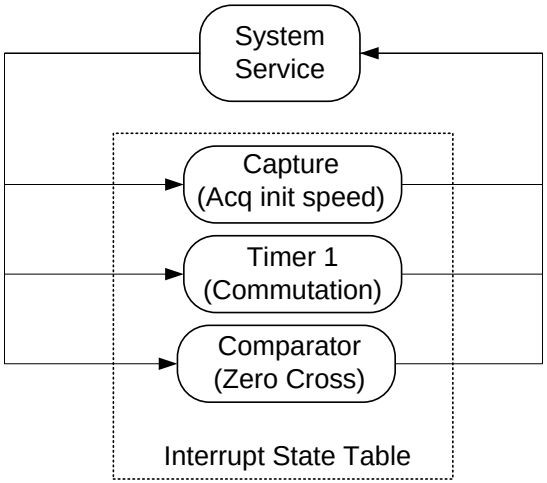


Figure 6: **Motor control code**

System Service

System services control the start-up sequence and monitor system status while the motor is running. S tart-up consists of system initialization and Phase Lock search. Status monitoring includes stall monitoring and speed control. The frequency of each start-up event and status check is determined by the time base manager.

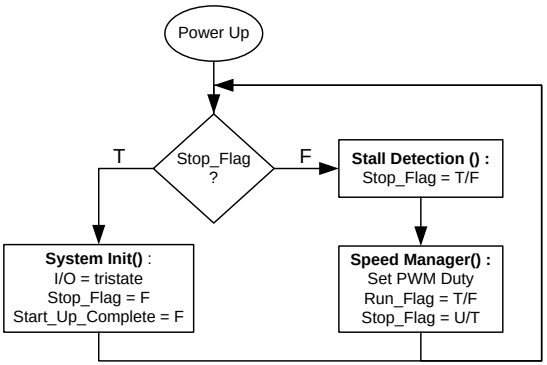


Figure 7: **System Service Flow-Chart**

Time Base Manager

The time base manager is a 10 millisecond period timer based on Timer0. Each status monitor subroutine keeps track of the time base manager time-ticks by means of a flag. Every 10 milliseconds, the time base manager sets a series of flags, one for each start-up control and status monitor. The flag is an indication to the respective control or status monitor to update its own internal counter. The control or status function is serviced when the corresponding counter reaches zero. The counter is then reset to the time count for that monitor or control and the service is performed.

System Initialization

System Initialization sets the Special Function Registers (SFRs) to configure the microcontroller and sets all ports to their initial off state. Initialization occurs at power-up and whenever the motor is stopped. The motor may be stopped as a result of a Fault or because the speed request is below the lowest run speed. When initialization is active, all of the status functions, other than speed request, are disabled.

Control Start-Up

Start-up looks slightly different than the normal run condition. The only difference between normal run mode and start-up is the `startup_complete` flag.

The `startup_complete` flag is cleared during initialization and remains clear until the zero-crossing event is detected within +/- 12% of the commutation mid-point. The `startup_complete` flag is necessary to prevent system acceleration before locking the motor phase.

If zero-crossing fails to set the `startup_complete` flag within the allowed stall time, then the motor is stopped and the start-up steps repeat from system initialization.

Stall Monitoring

Stall monitoring checks to make sure that the motor responded properly to the start-up conditions and is actually rotating. If a stall is detected then the motor is stopped and a restart is attempted. There are various reasons why the motor may not be rotating. For example, the rotor could be held in position by some blockage.

In this case commutation will go through the acceleration process without the motor following. A stalled motor will produce a zero-crossing event almost immediately after commutation. We allow for early zero-crossing for a short while during start-up, because a starting motor is by definition stalled. If the motor does not sense the zero-crossing event in the middle of the commutation period in a reasonable amount of time, then the motor is assumed to be stalled. What sometimes happens though, is that the control algorithm continues to shorten the commutation period as a result of acceleration until the commutation period just happens to be twice the time to the zero-crossing. The time from commutation to zero-crossing did not change, only the commutation period did. The commutation period is compared to the speed control speed request and, if the commutation period is

much shorter than expected, then a stall condition is assumed.

Speed Manager

The speed manager varies the voltage applied to the motor. This is accomplished by pulse-width modulating the motor driver switches. Even in systems where the motor always runs at full speed, speed control is needed to soft start the motor. Without soft start, the start-up currents will be excessive and starting torque could cause system damage. The speed control manager always starts the motor at the same voltage. The speed control manager starts increasing or decreasing the applied voltage up to the desired level when the zero-crossing detection senses that commutation is synchronized with the motor. Voltage is varied by varying the on-period of the ECCP PWM. The value in CCPR1L sets 8 Most Significant bits of the on-period. Two more Least Significant bits of resolution are set by the DC1B[1:0] bits in the CCP1CON register. The duty cycle percentage of the PWM is calculated as (CCPR1L: DC1B[1:0])/(PR2:0b00).

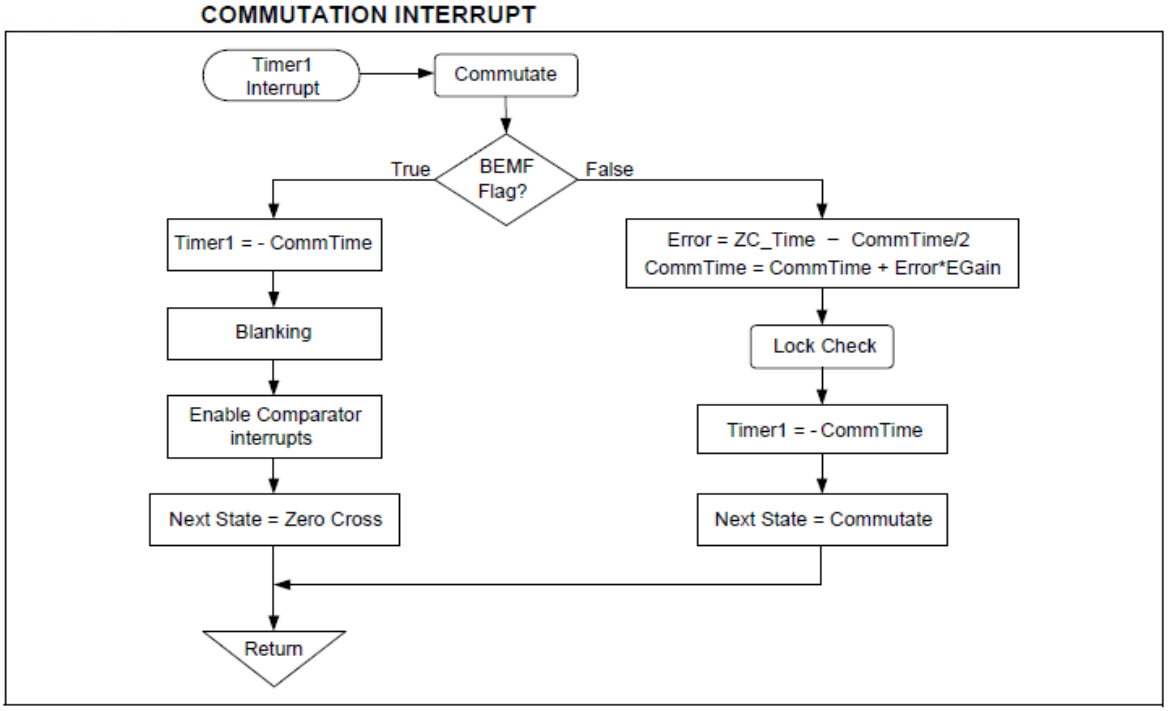
Commutation Interrupt

At the beginning of each commutation interrupt the commutation subroutine is called in which the motor drivers and BEMF sense lines are switched for the upcoming commutation period. Commutation time is controlled by Timer1. Timer1 is preset so that overflow will occur when the commutation time has elapsed. The interrupt occurs when Timer1 overflows one commutation period later and the process is repeated for the next commutation phase. Timer1, in effect, always contains a negative number representing the time remaining until the next commutation event.

At each commutation event, the commutation subroutine is called to switch the motor drivers. The commutation subroutine also sets a flag to let the Interrupt Service Routine (ISR) know whether to setup for a zero-crossing event or make error correction calculations.

The commutation interrupt has two purposes other than switching the drivers: Zero-crossing setup and commutation time calculation. In the first instance, if the zero-crossing event will be captured in the upcoming period, then the comparator interrupt must set up for that occurrence. In the other instance, the zero-crossing event cannot be captured so there is more time available to make calculations to correct the commutation period. In both cases, Timer1 is preset with the negative of the last calculated commutation time.

For more information about the commutation period math



refers to Microchip AN1305.

Zero-Crossing Interrupt

The zero-crossing event is captured by a comparator interrupt. The comparator interrupts are sensed by an exclusive-or gate comparing two mismatch latch outputs. The two mismatch latches consist of a holding latch and a temporary latch. The holding latch is set to the comparator output value when the comparator control register (CCPxCON) register is accessed (read or written). The temporary latch is set to the comparator output value at each instruction cycle. If the comparator output changes after the holding latch is set, an exclusive-or gate will signal the difference between the holding and temporary latches and set the interrupt. Therefore, it is imperative that when the comparator interrupt is enabled, the comparator output is in the state opposite the state it will change to when the zero-crossing event occurs. The CCPxCON register is accessed as part of the commutation switching routine at which time the comparator output is indeterminate. For this reason it is necessary to access the CCPxCON register again later when setting up the comparator interrupt.

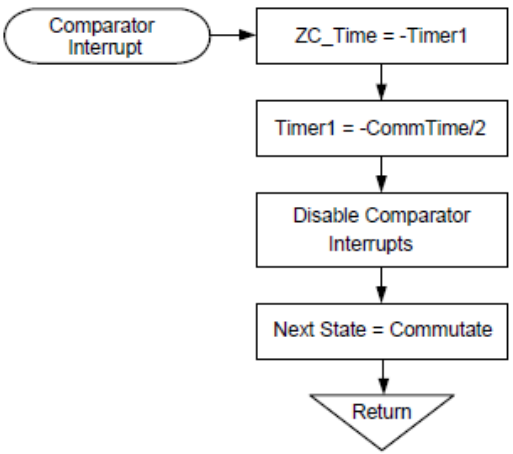


Figure 8: **Zero Crossing Interrupt**

There is one major obstacle that can prevent properly setting up the comparator for the zero-crossing event. At commutation, one motor coil is detached from the supply and another is attached. The current in the detached coil does not stop immediately because of the coil inductance. The current must flow somewhere so it flows through the body diode of either the high-side switch or low-side switch. When the broken connection is from the negative supply side then affected current is flowing out of the coil towards the driver switches. Both switches are off so the only path through which the current can continue is the high-side switch body diode. The current continues from there on the motor supply line and back to the motor through the high-side drive switch that is still on. This causes a high transient on the BEMF sense line. A negative spike similarly occurs when the broken connection is from the positive supply side. As motor current varies with the load, so does the energy that must be dissipated in this spike. During acceleration and high load conditions, the energy in the coil inductance increases causing the width of the transient to lengthen. Conversely, during deceleration and light loads the width shortens. The comparator interrupt cannot be setup until all the energy in the transient has been dissipated.

Avoiding the transient period is called blanking.

Blanking

There are two ways to handle blanking: timed and dynamic. The timed method is just as it sounds. After commutation, a specific time is allowed before the comparator interrupt is setup. The problem with timed blanking is that the wait time for all blanking events must be long enough to accommodate the worst case spike. At high RPM rates, this could be a significant percentage of the commutation period and may limit the maximum speed attainable. Dynamic blanking solves this problem. In dynamic blanking, a short minimum blanking time is allowed to elapse, to make sure the commutation switch is complete, and then the level of the comparator output is tested until the inductive transient is no longer present. The level of the comparator output is in a known good state immediately after the transient, at which time the control register can be read to set the mismatch holding latch. Only then can the comparator interrupt be cleared and enabled.

Capture Interrupt

In some application it is very important to start driving a motor which is already rotating. For example, let's imagine a man who uses an electric bike: while he is pedaling he wants to accelerate by using the electric motor. In this condition the motor control needs to understand the wheel rotation speed and start driving the motor after being synchronized with it.

Otherwise, if the driver starts to drive the motor with a wrong switching frequency or with a wrong alignment with the stator magnets this will produce a strong motor breaking.

The reason why this happens is reported in Figure 9. We can see a low side modulated BLDC motor with a PWM equal to zero.

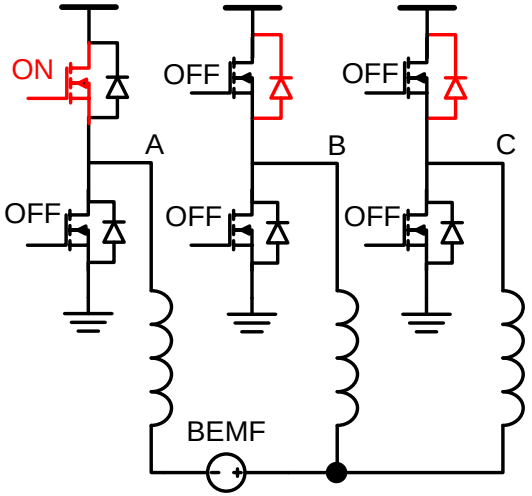


Figure 9: **Motor breaking effect if the driver is not synchronized with the BEMF produced by the rotating stators**

In this case all the low side transistors are OFF and just one high side transistor is switched ON. When the BLDC motor rotates, each winding generates BEMF voltage. When the driver is not synchronized with the motor rotation we will have some condition where the generated BEMF is positive in respect to the supply voltage. This will produce a switching ON of the high side diodes shorting the motor terminals together.

In order to avoid this problem before starting with normal operation condition we have to initialize the switching frequency and getting synchronized with the motor. Capture Interrupt is the ISR routine with this goal. It starts immediately after system initialization.

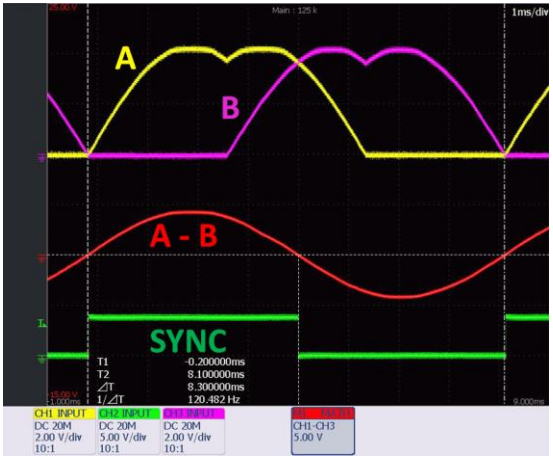


Figure 10: **PhaseA and PhaseB of BLDC motor which is rotating but not electrical driven**

Figure 10 shows two phases of an undriven and rotating BLDC motor. The difference between phase A and phase B is a perfect sinusoid as shown by the red waveform in the figure. This sinusoid contains all the information we need for getting synchronization with the motor. The period of the sinusoid is exactly one electrical cycle.

Figure 11 shows the simple circuit used to generate the green sync signal of Figure 10. Just using two external resistor connected to two pin of the Micro we can obtain the synching signal. The rising edge of the comparator output is used for obtaining the time of one electrical cycle. This can be done using the Capture hardware of the Micro which saves the Timer1 value into a register each rising edge of the comparator output. This time has to be divided by six in order to obtain the commutation time for each step.

In order to avoid this “breaking” effect when the motor is not directly driven by IR3230 we decide to force the IC into a latched fault condition. We can simply do that by applying a wrong sensor code (for example “000”). After that, IR3230 will switch off immediately all the MOSFET avoiding any blockage of the wheel.

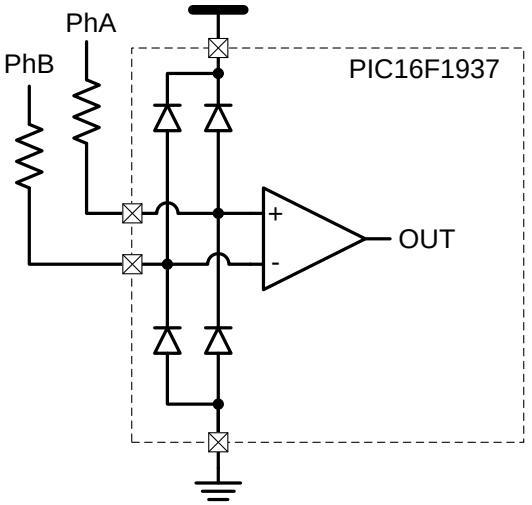


Figure 11: **The internal Micro comparator is used to generate the green sync signal of Figure 10**

Fault Reset

All the time we want to start driving the motor we have to reset the IR3230 fault. We have two main reasons to be in latched fault condition:

- First, the accelerator was at zero and the IR3230 didn't drive the motor. The microcontroller applied a wrong sensor code in order to avoid any wheel blockage (as discussed in previous section);
- Second, an overcurrent shutdown.

IR3230 has a programmable overcurrent shutdown which generates a latched fault when the input current is higher than a programmed threshold.

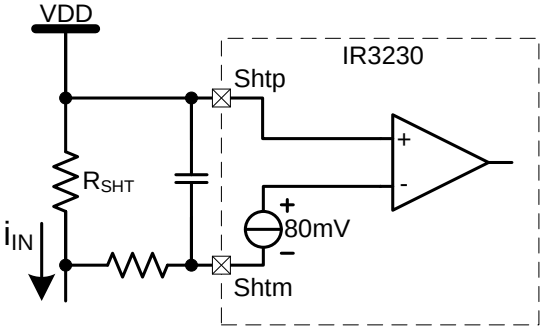


Figure 12: **IR3230 internal overcurrent shutdown**

Figure 12 shows the overall shunt circuit: R_{SHT} value stays between 1mΩ and 10mΩ, the low pass filter cut off frequency is around 1kHz and the internal comparator stage has 80mV of voltage threshold. If we suppose to use 2mΩ shunt resistor we have a peak current limit of 40A.

When the input current gets higher than this value, IR3230 generates a latched fault and immediately disconnects all the power Mosfet to the three phases motor.

When this happens, the Micro needs to reset the fault by applying a digital one to the Flt_rst pin of IR3230. The input pin RB4 of the micro is used to sense the IR3230 fault and pin RD6 is used to reset the fault.

The routine FLTCheck() is used to check if some faults appears, reset the IR3230 and restart the PIC control algorithm. This routine is enabled after the Capture Interrupt event.

Figure 13 shows the input current profile when applying a PWM ramp of 125ms. While the motor is accelerating the peak current can be very high in respect to the steady state current value. This means that IR3230 can easily generate an overcurrent fault. The simplest way to limit the peak input current is to slow down the PWM ram (if using PWM ramp of 500ms the peak current becomes 20A). To generate a slower PWM ramp we can simply increase the parameter TIMEBASE_DUTY_RAMP contained in file "eBike_Motor.h".

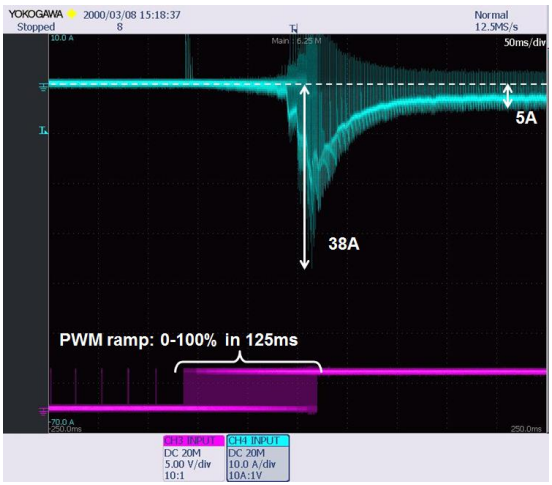
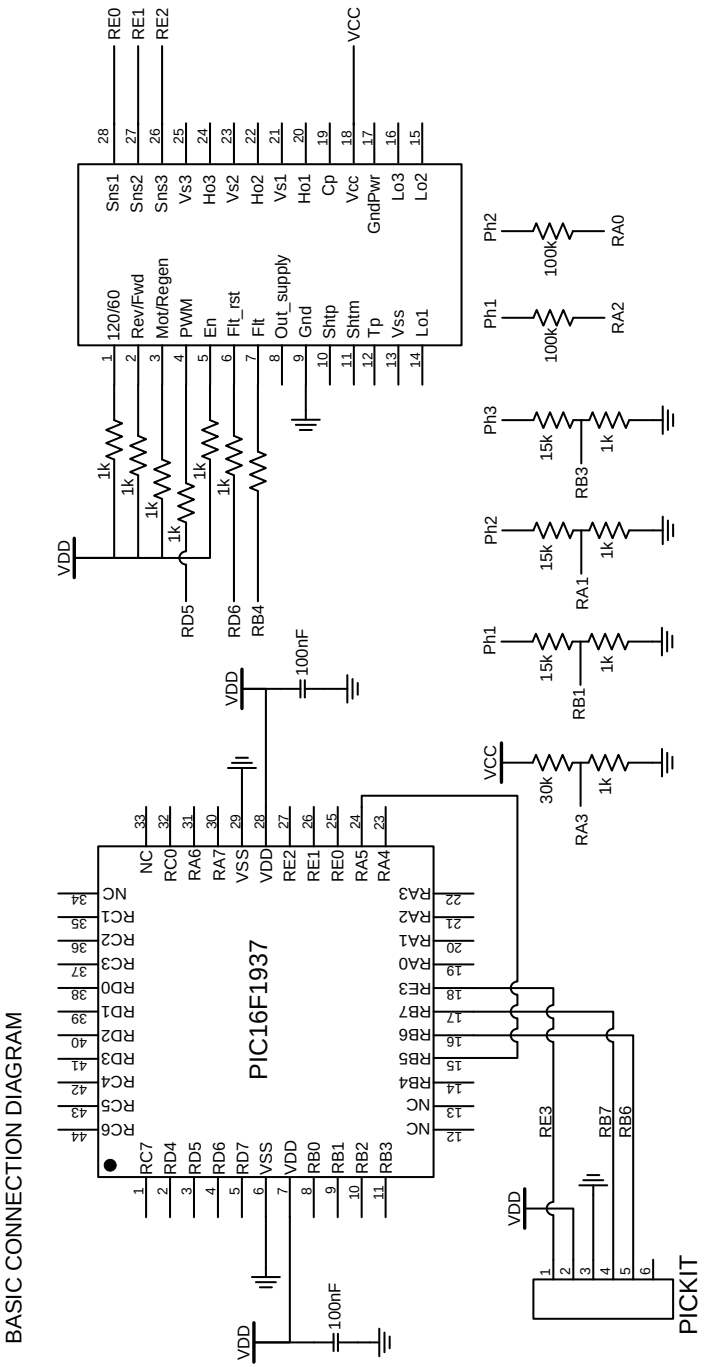


Figure 13: **Input current profile when applying a PWM ramp of 125ms.**


BASIC CONNECTION DIAGRAM
PIC16F1937
PICKIT
VDD
VCC
100nF
1k
15k
30k
100k
RE0
RE1
RE2
RD5
RD6
RB4
RB3
RB2
RB1
RA3
RA2
RA1
RA0
RE3
RB7
RB6
Sns1
Sns2
Sns3
Vs3
Ho3
Vs2
Ho2
Vs1
Ho1
Cp
Vcc
GndPwr
Lo3
Lo2
120/60
Rev/Fwd
Mot/Regen
PWM
En
Flt_rst
Flt
Out_supply
Gnd
Shtp
Shtm
Tp
Vss
Lo1
Ph1
Ph2
Ph3