AN1305

Sensorless 3-Phase Brushless Motor Control with the PIC16FXXX

Author: *Ward Brown*
Microchip Technology Inc.

OVERVIEW

In 2002, I wrote my first application note on brushless motor control, AN857, which described the operation of sensored and sensorless brushless motors. The basic motor operation concepts described in that application note still apply today, therefore, I will not repeat them here. In the time since then, enhancements to two of the microcontroller peripherals have been made that dramatically improve the microcontroller capabilities for sensorless motor control applications. This application note will describe those enhancements and how they are used to create a sensorless motor control solution with an 8 MHz PIC16FXXX device that has truly stellar performance with seamless operation from 100 RPM to over 90,000 RPM.

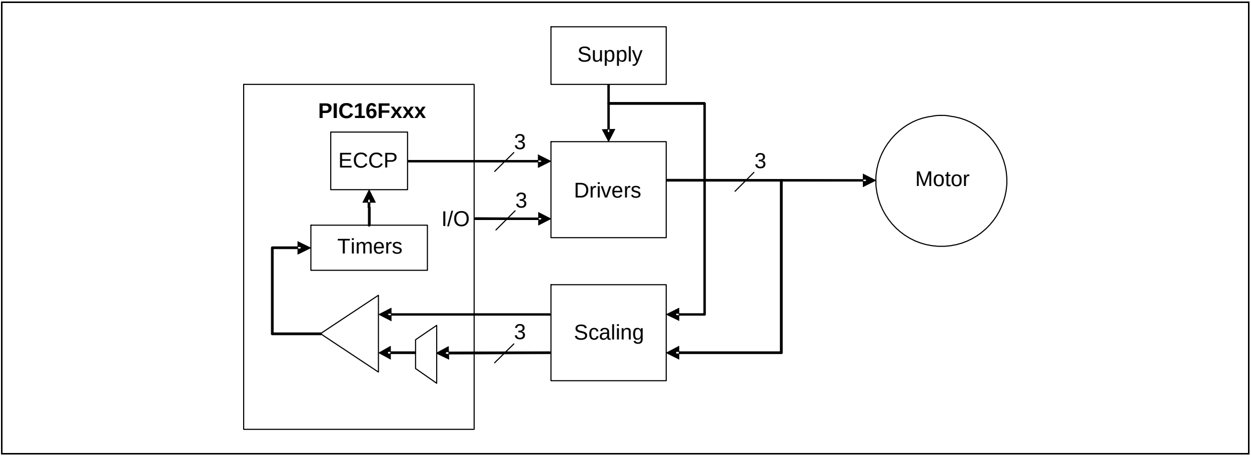
TYPICAL MOTOR CONNECTION

Figure 1 is a simplified block 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 drives 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 1: DRIVE AND CONTROL CIRCUITRY



Voltage to the motor from the motor supply is varied by pulse-width modulating the driver FETs. Only one side of the drive needs to be modulated: either the high side motor supply or the low side supply return. This requires that three of the microcontroller outputs to the FET drivers have PWM capability, while the other three can be general purpose output pins.

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.

We'll now take a look at the new peripheral features to see how they meet the FET driver and comparator input requirements and how they are controlled.

MICROCONTROLLER PERIPHERAL ENHANCEMENTS

Two relatively minor enhancements are now incorporated in some PIC16FXXX devices that significantly improve the capabilities of the comparator and the enhanced capture, compare, and PWM (ECCP) peripherals. These enhancements are particularly useful for brushless motor control applications. The comparator now has four independently selectable inputs to the inverting input and the ECCP has up to four individually selectable outputs in single PWM mode.

COMPARATOR ENHANCEMENTS

Figure 1 shows a block diagram of the new 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.

ECCP ENHANCEMENTS

The new Enhanced Capture Compare and PWM peripheral (ECCP) has the capability to direct the PWM output to four I/O pins. The PWM output for each pin is individually selected by a bit in the PSTRCON control register. In single PWM mode, the PWM signal can be directed to any combination of the four outputs. For brushless motor control three of the PWM outputs are connected to three of the motor driver devices. The PWM outputs are connected to the high side drivers for high side modulation or the low side drivers for low side modulation. For each of the six commutation states, one PWM output is enabled to drive one motor terminal with modulation, while another motor terminal is driven steady state. The third motor terminal floats and is used to detect motor position.

MOTOR CONTROL

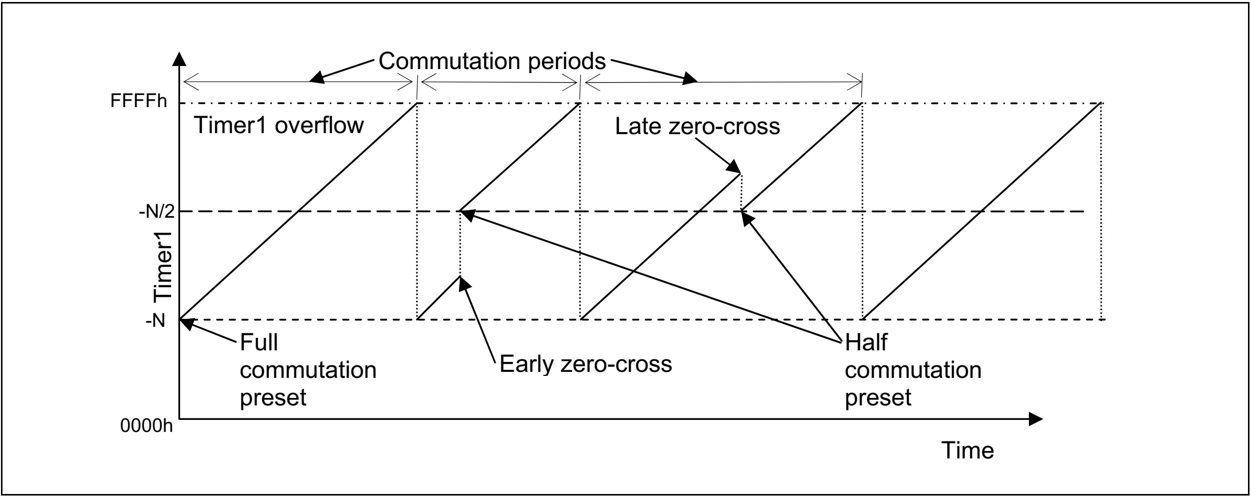
The motor is controlled by synchronizing the commutation with both the motor position and the motor speed. Motor speed at various supply voltage and load conditions is a function of the motor design. The control algorithm searches for this intrinsic speed and adjusts the commutation period to match. 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.

The motor position can be determined by sensing the voltage on the undriven motor phase. The first half of the section on Sensorless Motor Control in AN857 covers this in detail. One of the differences between the approach of AN857 and this application note is that we will be using a comparator to determine the voltage instead of using an analog to digital converter.

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.

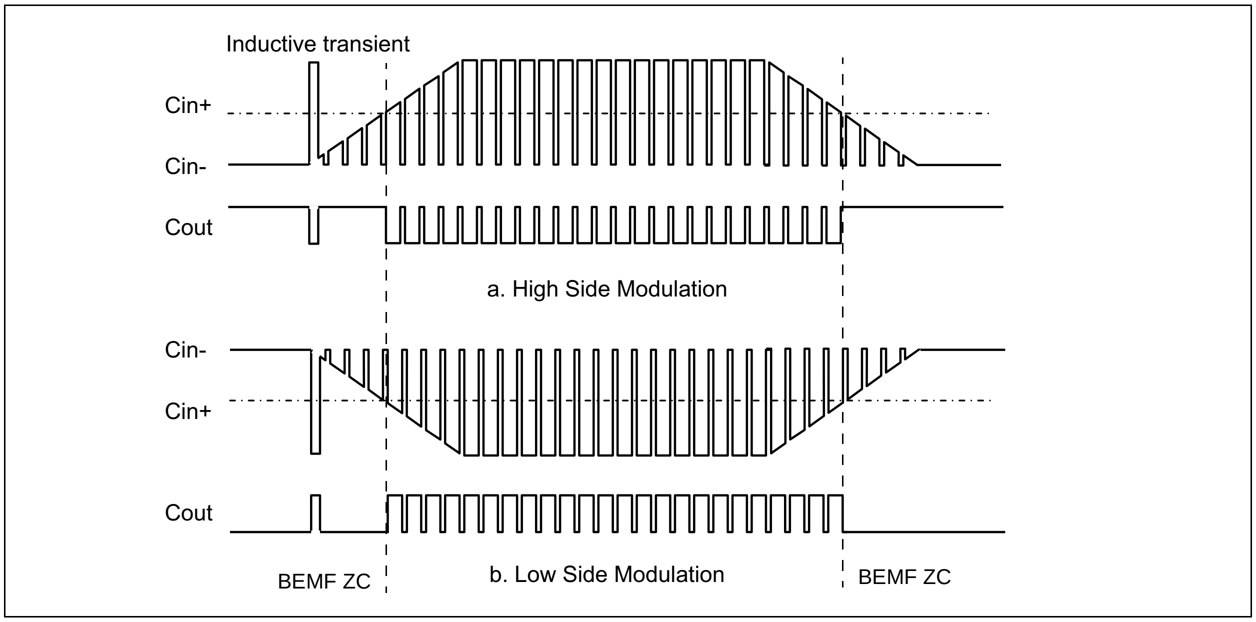
FIGURE 2: COMMUTATION



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 pictures 3a and 3b in Figure 3. Picture 3a (High Side Modulation) shows one phase of a motor drive waveform for a high-side modulated drive. Picture 3b (Low Side Modulation) shows one phase of the motor drive waveform for a low-side modulated drive.

FIGURE 3: ZERO-CROSS DETECTION



In Figure 3, 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 high-side modulated system, when the motor terminal BEMF voltage is rising the comparator output, after the inductive transient, is steady until half-way through the commutation period. Once the comparator transitions the first time, it continues to transition at every PWM drive cycle for the remainder of the commutation period, as shown in the lower half of pictures 3a and 3b. On the same high-side modulated system, when the BEMF is falling, the comparator continues to transition at every PWM cycle until the last half of the commutation period. The opposite is true for low-side modulated systems, that is, when the 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 during rising BEMF periods on high-side modulated systems and 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 DRIVE

For speed control and soft start-up, the motor supply voltage is pulse-width modulated. The modulation is applied to the motor driver switches and needs to be applied only to either the high-side or low side drivers. Sometimes it is necessary to modulate the low-side drivers in order to cycle the high-side gate driver charge pumps. Otherwise, high-side or low-side modulation is a matter of personal preference. In each commutation period, one motor terminal is driven high, one terminal is driven low, and the third remaining terminal is left floating. The modulated drive comes from an ECCP PWM output pin. The unmodulated drive comes directly from an output pin. There are four ECCP PWM pins designated: P1A, P1B, P1C, and P1D. The PWM output pins are individually controlled by bits in the PSTRCON register. When the PWM is active a `1` in the PSTRCON register corresponding to the desired active output will modulate that pin. All other PWM pins will output the level in the corresponding PORT output latch. It is important not to perform any output operations directly to the port sharing the PWM outputs. The reason is that all writes to a register are performed as read-modify-write operations. If the active PWM is high when an operation, such as a bit-set, is performed on a non-PWM output in the same port as the PWM, then the output latch for the active PWM pin will be changed to a high level. When the PWM output eventually moves to the next drive phase the output latch will drive the unmodulated pin high.

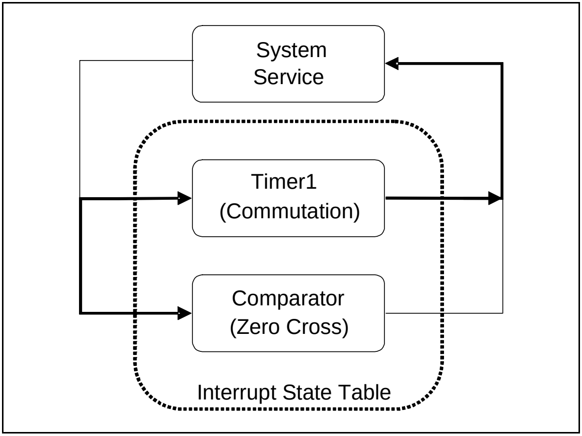
COMMUTATION STATES

There are six commutation states in each electrical revolution. Each state 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 4. The main System Service loop, in addition to status monitoring, controls the start-up sequence. Two interrupt types are enabled to control motor operation: Timer1 and comparator. 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.

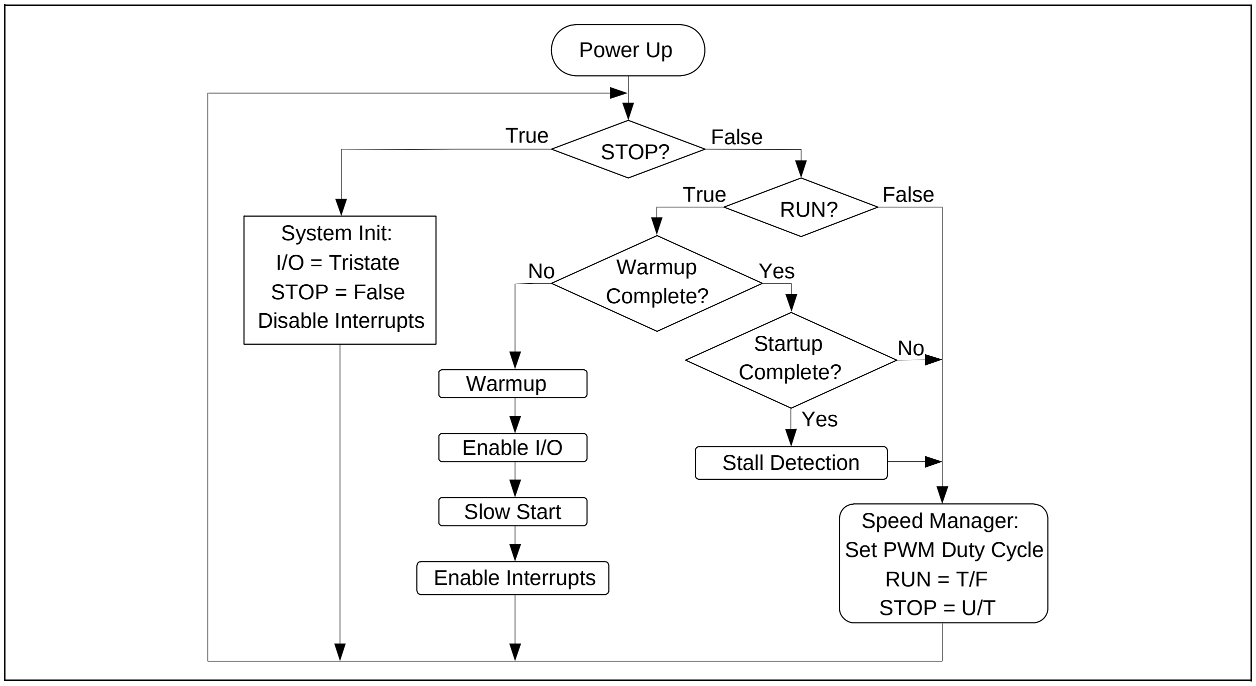
FIGURE 4: MOTOR CONTROL CODE



SYSTEM SERVICES

System services control the start-up sequence and monitor system status while the motor is running. start-up consists of system initialization, warm-up, slow start, 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 5: SYSTEM SERVICES



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.

WARM-UP

Warm-up allows the system to settle after initialization. Warm-up commences immediately after initialization and lasts for the warm-up time specified by the `TIMEBASE_WARMUP_ms` constant. When warm-up is complete, the motor drivers are enabled and slow start commences.

CONTROL SLOW START

Control slow start prepositions the motor to a known commutation state so that subsequent commutations can start to accelerate the motor up to speed. Recall that the drive voltage is applied to two of the three motor windings. This causes the permanent magnet rotor to align with the created magnetic field which is between the active windings. Some motors have a strong tendency to align the rotor magnets with one of the motor windings when power is not applied. There are 60 electrical degrees between windings, so that applying power to such a motor will cause the rotor to move at least 30 electrical degrees, or more if the rotor was aligned to the winding not being powered. Other motors have a weak alignment tendency, so that the rotor may be aligned 180 degrees out of phase with the generated magnetic field when power is applied. In this case, the rotor will not move at all and will be in an improper position to accelerate properly when commutation begins. For this reason, motors that have a weak alignment tendency must receive two slow start positioning drive periods of sequential commutation drive states to ensure that the rotor is not stuck 180 degrees out of phase when the actual commutation starts. The amount of time to dwell in each slow start state depends on the inertia of the motor and the drive voltage applied. High inertia motors need more dwell time than low inertia motors. This allows the rotor to settle to a known position which makes the acceleration response to the first commutation drive more consistent and predictable. The dwell time for each slow start step is specified by the `TIMEBASE_SLOW_STEP` constant. The start-up commutation period is set in Timer1 and interrupts are enabled at the conclusion of slow start. At this point, Timer1 interrupts have complete control of the motor commutation and comparator interrupts determine how to adjust Timer1 to achieve Phase Lock with the BEMF signal.

CONTROL START-UP

Start-up commences immediately after slow start and looks only 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 warm-up 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 start-up from being detected as a stall condition. The motor accelerates during start-up and the commutation period shortens to catch up, because zero-crossing is detected almost immediately after commutation. See the **Section "Zero-Crossing During Start-up"** for more detail on how this works.

Control start-up routine does nothing more than check that Phase Lock to the BEMF signal is achieved in a reasonable amount of time. It does this by setting a timer and checking that the `startup_complete` flag is set when the start-up time is complete. If zero-crossing fails to set the `startup_complete` flag within the allowed start-up 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, or the rotor did not settle fully during slow start-up. In both those cases commutation will go through the acceleration process without the motor following. As we will see later, 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).

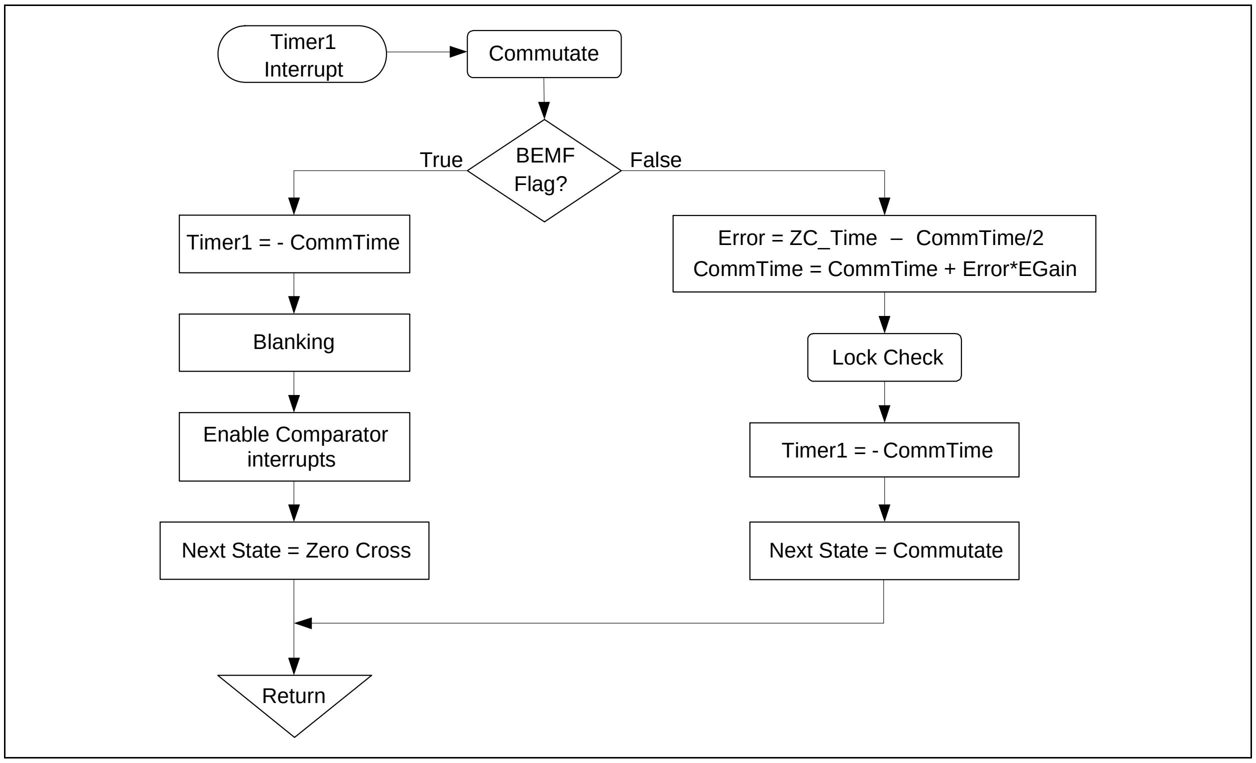
INTERRUPTS

Two types of interrupts are used to control the motor: Timer1 interrupts set the commutation period and comparator interrupts capture the time of the zero-crossing event.

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.

FIGURE 6: COMMUTATION INTERRUPT



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.

COMMUTATION PERIOD MATH

Since Timer1 is a 16-bit timer, the Timer1 count registers (TMR1H:TMR1L) can contain a number from 0x0000 to 0xFFFF. Timer1 values from 0x8000 to 0xFFFF are treated as negative signed integers. However, Timer1 values from 0x0000 to 0x7FFF are treated as positive signed integers. Normally, this wouldn't matter because Timer1 could be treated as an unsigned integer, but that causes problems when performing math on partial commutation periods such as the mid-commutation point, otherwise known as the expected zero-crossing point. Signed integers are necessary to accommodate positive and negative error calculations. To avoid the need to do all commutation period calculations with long signed integers, one simple trick can keep the math to the size of a signed integer and at the same time realize the full 16-bit count capability of Timer1.

Consider that the commutation period as written to Timer1 is a 16-bit negative integer. If we assume that the sign bit is the 17th unimplemented bit, then that bit will always be '1'. The only time the commutation period is measured is at the zero-crossing event. By definition the zero-crossing time is half the commutation period. The expected zero-crossing value can be calculated from the signed 16-bit commutation period value by shifting the commutation value right by '1' and always setting the Most Significant bit to '1' after the shift. Therefore, values that have no sign bit to extend will assume the negative sign of the unimplemented 17th bit which is always '1'. Negative integer values will have their sign bit extended during the shift, so setting the Most Significant bit makes no difference. Once we have the expected zero-crossing value in signed 16-bit integer format, all other calculations, such as the zero-crossing error and next commutation period are calculated with simple 16-bit signed integer math functions. However, remember that the commutation time is stored and used as a signed 16-bit number which is always negative, even when the sign bit says otherwise, because the implied 17th bit makes it so.

Since the value in Timer1 is always negative, the Timer1 value captured at zero-crossing will also be negative. The error between the expected zero-crossing and the actual is computed, scaled, and accumulated in the stored commutation time as follows:

EQUATION 1:

$$ZCE = ZCT - CT/2$$

$$CT = CT + ZCE*EGain$$

Where:

ZCE = Zero-Crossing error

CT = Commutation time

CT/2 = Expected zero-crossing

ZCT = Zero-Crossing time

EGain = Feedback Gain Factor

When zero-crossing occurs early, then ZCT will be a larger negative number than CT/2 and the error will reduce the commutation time. The opposite will be true when zero-crossing occurs late. The scaling factor determines how quickly the system responds to commutation errors. A large scaling factor will provide a slower response. However, a large scaling factor will also introduce larger truncation errors since we are performing all calculations in 16-bit integer math. Small, low-inertia systems respond best to a small scaling factor whereas high-inertia systems work better with a large scaling factor.

SETTING UP TO CAPTURE THE ZERO-CROSSING EVENT

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.

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 as shown in Figure 3a (High Side Modulation). 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.

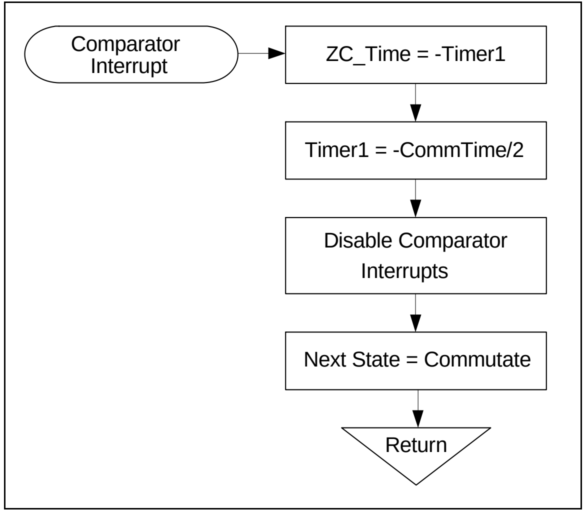
BLANKING

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.

ZERO-CROSSING INTERRUPT

The zero-crossing interrupt occurs when the BEMF voltage reaches the BEMF reference voltage. Two things happen in the zero-crossing interrupt service: First, the value of Timer1 is read, and then Timer1 is preset with half the previously computed commutation time. During steady state operation the value read from Timer1 is essentially the same as the value written to Timer1. Regardless of when zero-crossing occurs, early or late, the value written to Timer1 will always be half the previously computed commutation time. During acceleration zero-crossing will occur early and the value read from Timer1 will be a larger negative number than expected. The difference between the read number and the expected number is the zero-crossing error which will be used to correct the commutation time when the new commutation time is computed as part of the next commutation interrupt.

FIGURE 7:



ZERO-CROSSING DURING START-UP

When the motor is not rotating, it is not generating any voltage. The undriven motor terminal then will be approximately half the motor supply voltage because the high-side and low-side drivers form a low-impedance voltage divider. This is the voltage that will be presented to the zero-crossing comparator input. The inter-winding capacitance of the motor coils will cause a small overshoot in the PWM drive signal so the zero-crossing comparator will sense every PWM pulse from the beginning of each commutation period. This means that when the motor is starting the zero-crossing interrupt will occur almost immediately after the blanking period. At the same time, the motor will start to accelerate and the calculated period for the next commutation will be shorter because of the early zero-crossing event. Until the motor comes up to speed, it will be operating in step response to the applied power. As the motor approaches the ideal commutation rate, and the motor generated voltage grows, the zero-crossing events will move toward the center of the commutation period. When the period is twice the time to zero-crossing, the control will be synchronized with the motor.

HARDWARE

The hardware for sensorless brushless motor control is relatively simple. Six supply switches are needed as are four voltage dividers. Of the six supply switches, three are for switching the plus motor supply to the three motor terminals, and the other three are for switching the three motor terminals to the supply return. Of the four voltage dividers, three scale the motor terminal voltage for measurement by the microcontroller and the fourth is used to scale the motor supply voltage as a reference to which the motor terminal voltages are compared.

PWM FREQUENCY AND PERIOD

The PWM frequency is a function of Timer2 and the PR2 register. Timer2 is an 8-bit counter. When the count reaches the value of PR2, then Timer2 is reset to zero and the cycle repeats. The duty cycle period of the PWM is a function of Timer2 and the CCPR1L register. At the beginning of each PWM period, the PWM output is high. When the Timer2 count equals the CCPR1L register, the output is set low. Smaller values in PR2 will produce higher PWM frequencies at the cost of duty cycle resolution because CCPR1L does not have as many bits to work with. There are actually two more bits of duty cycle resolution in the CCP1CON0 register, labeled DC1B1 and DC1B0. Timer2 increases by one count every fourth system clock, or FOSC/4. The DC1B0 bit changes with every FOSC clock and the DC1B1 bit changes every second FOSC clock.

There are several factors to consider when choosing the PWM frequency. Low frequencies will be audible at slow motor speeds. High frequencies will cause greater switching losses, especially under high load high current conditions. A general rule of thumb is to choose a PWM frequency from 16 kHz to 20 kHz.

COMPUTING THE VOLTAGE DIVIDERS

Resistive voltage dividers are used to scale the motor supply and motor terminal voltages within a range acceptable to the microcontroller inputs. The dividers will be used to compare the motor BEMF voltage to the motor supply.

Compute the voltage divider for the reference such that the reference voltage is well within the common mode range of the microcontroller comparator input. A good rule of thumb is to set the reference voltage to half the microcontroller supply voltage. The reference is connected to the motor voltage so the equation for the reference divider is:

EQUATION 2:

$$V_{DD}/2 = Vmotor*Y/(X+Y)$$

Where:

VDD = microcontroller supply

Vmotor = Motor supply

X = Resistor from Vmotor to non-inverting comparator input

Y = Resistor from non-inverting comparator input to ground

Compute the BEMF voltage dividers from the values of the reference divider:

EQUATION 3:

$$A = X$$

$$B = 2 * Y$$

Where:

A = Resistor from motor terminal to inverting comparator input

B = Resistor from inverting comparator input to ground

X and Y are resistor values determined in Equation 2.

TROUBLESHOOTING

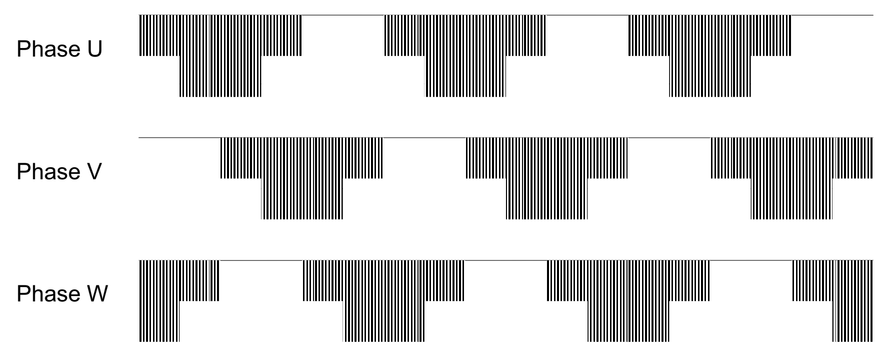
The most common troubles encountered when attempting to use a new motor or driver circuit are driver configuration, motor acceleration, and zero-crossing detection. The easiest to detect and remedy is the driver configuration. Start-up and zero detect are more difficult because, unless the motor is running, you cannot tell which is the problem and the motor will not run unless the problem is corrected. Debugging then becomes an iterative process by holding one parameter fixed then trying a range of the other parameter. The following sections deal with each aspect separately.

DEBUGGING DRIVER CONFIGURATION

There are too many possible issues with driver configuration to deal with individually. Here is a simple and safe method to determine that all circuit elements are working properly from the motor terminal back to the microcontroller. The first step is to replace the motor with a resistor network. Construct the network with three resistors. One end of each resistor connects to each of the driver outputs. The other end of all three resistors is connected to a common node. The resistor values must all be the same and high enough to prevent damaging current flow at the maximum motor voltage, but low enough to allow proper biasing of the output driver devices. A good rule of thumb value for a 15 to 20-volt system is 270 Ohms.

With the network in place on the driver outputs, invoke motor start-up and observe the waveform on each driver output. For a low side modulated system, the waveforms should look like those shown in Figure 8, if they are then proceeded with debugging start-up with the motor. If they are not, then use standard circuit debugging techniques to determine where the fault lies. For a high side modulated system rotate the figure right by 180 degrees.

FIGURE 8: DEBUGGING DRIVER CONFIGURATION



DEBUGGING START-UP

The most difficult aspect of sensorless motor control is starting the motor. Before the motor starts spinning, and generating a BEMF voltage, the motor is responding to commutation more like a stepping motor than like synchronous motor. Because every motor is different the response to applied voltage will also be different. We want to start the motor as quickly as possible to avoid excessive current in the stepping mode. However, if the ever shortening steps between commutation times are too large, then the steps can easily skip over the zone where BEMF crosses near the middle of the commutation period. If the steps are too small, then more time is spent in Stepping mode with the associated excessive current.

The start-up algorithm always begins with a fixed PWM duty cycle. The challenge is to determine what the duty cycle should be for optimum start-up performance when the desired nominal motor voltage is applied. One method is to set the PWM duty cycle to an arbitrary value that will prevent excessive current damage even when the motor is stopped. A good starting point is 25%. Then apply a voltage to the motor below where the motor will operate. Try starting the motor with this combination. It is unlikely that the motor will start reliably. Gradually increase the applied voltage and retry to start the motor at each level until the motor starts reliably. Use the combination of fixed PWM duty cycle and the reliable starting voltage to compute the duty cycle needed to start the motor when the desired nominal voltage is applied.

DEBUGGING ZERO-CROSS

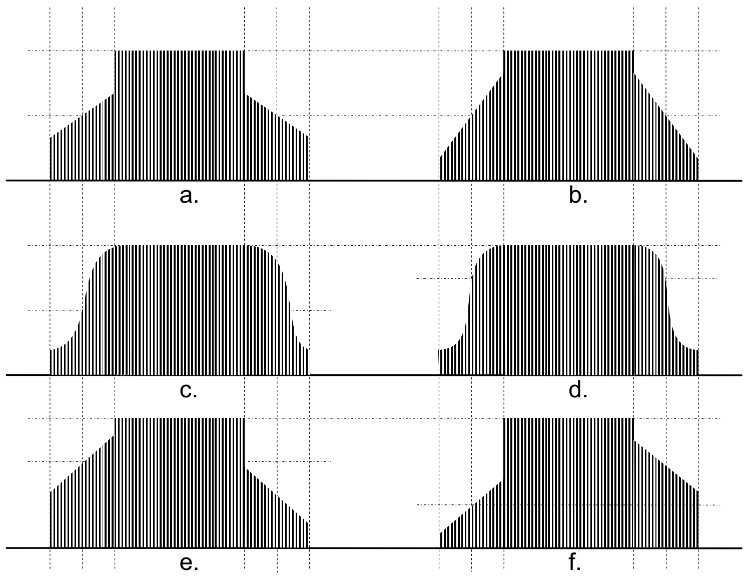
A perfectly sinusoidal motor with perfectly balanced windings and perfectly equal drivers will produce a back EMF on the open terminal that is exactly equal to half the motor supply voltage at exactly midway through the commutation period. Nothing in the real world is perfect but it’s usually close enough. However, in some cases adjustments need to be made. This is especially true for non-sinusoidal wave shapes. Offsetting the zero-crossing reference voltage can compensate for just about every imperfection.

Consider the waveforms in Figure 9. Pictures 9a and 9b show the phase wave shapes for a balanced sinusoidal motor running at about 30% and 60% of maximum speed, respectively. These diagrams are for a high-side modulated system. Notice the symmetry of the wave shapes: the slope and relative position to mid-voltage of the rising voltage on the left is reflected mirror image on the right.

Now consider the waveforms in pictures 9c and 9d. In picture 9c, the left and right slopes are equal, but appear to be offset to the right. In fact, the left slope in picture 9c is offset down and the right side is offset up by the same amount. This is because the zero-crossing is being detected when the motor is only about 25% through the commutation period. In other words, zero-cross is being detected early. That moves the left slope down and the right slope up by an equal amount. Moving the zero-cross reference voltage up, as shown in picture 9d, moves the zero-cross detection later in the commutation period and aligns the drive cycle with the maximum motor generated back EMF for optimum performance.

If there is a problem with the zero-cross reference voltage, then that will also offset the BEMF waveform. Consider pictures 9e and 9f. Picture 9e shows the response when the reference is too high, and picture 9f shows the response when the reference is too low.

FIGURE 9: DEBUGGING DRIVER CONFIGURATION



PERFORMANCE

Maximum speed is a function of how quickly the micro-controller can capture and process zero-crossing events and compute the necessary corrections. The time it takes to commutate and compute will determine the minimum commutation period. The time it takes to commutate and setup for zero-cross will determine the minimum half commutation period. The shorter of these two will determine the maximum speed. Minimum speed will be determined on the longest period that can be measured by Timer1.

I claimed at the beginning a speed range from 100 RPM to over 90,000 RPM with a system clock of 8 MHz. This is easily verified with the MPLAB® simulator. There are six commutations in one electrical revolution. RPM is converted to commutation time with the formula: 10/RPM = Seconds/Commutation Period. At 90,000 RPM, the commutation period is 111.11 µs and half the commutation period is 55.56 µs. Using the MPLAB simulator with options set for 8 MHz system clock, the debug stopwatch indicates a total elapsed time 100 µs for the commutation interrupt and 55 µs for the commutation with blanking setup. Those times include context saving and restore before and after the interrupt service.

DRIVER CIRCUIT AND SOFTWARE

An example circuit for evaluating the concepts of this application note is shown in **Appendix A. “Schematics”**. This circuit is of the PIC16F1937 evaluation board (F1 Starter Kit) with BLDC add-on. The software to operate a brushless motor with the starter kit is available for download from the Microchip web site in the same location as this application note. The BLDC add-on board is supplied with a motor to ensure the best possible initial experience. The default code configuration is for the supplied motor. A terminal strip on the BLDC board permits connection of any BLDC motor requiring less than 7 amps peak current. Driving motors other than the supplied motor will most likely require some modifications to the motor header file as mentioned in the Debugging Start-up section of this document.

The software is divided into several separate files to simplify mixing and matching different drivers, motors, speed controllers, and microcontrollers. In most applications, changing from one speed control technique or microcontroller is simply a matter of swapping out the corresponding file in the software project.

The software has three header files: one for the motor type, one for the driver circuit, and the main header file. The header files are easily identified in the Project View window of MPLAB. The motor type and driver header files contain definitions unique to the motor type or driver circuit. The main header file has definitions common to all motor types and drivers. The main header file also determines which motor and driver header files to include in the project by means of definition macros. Those definition macros are defined in the build options dialog of MPLAB. The dialog is accessed with the pull-down menu *Project->Build Options->Project*. Select the **Compiler** tab and *Add or Remove* defines as appropriate. See the main header file, `BLDC.h`, for examples.

User adjustable parameters are contained, and explained, in the motor header file.

NOTES:

Appendix A. Schematics

FIGURE A-1: F1 EVALUATION PLATFORM

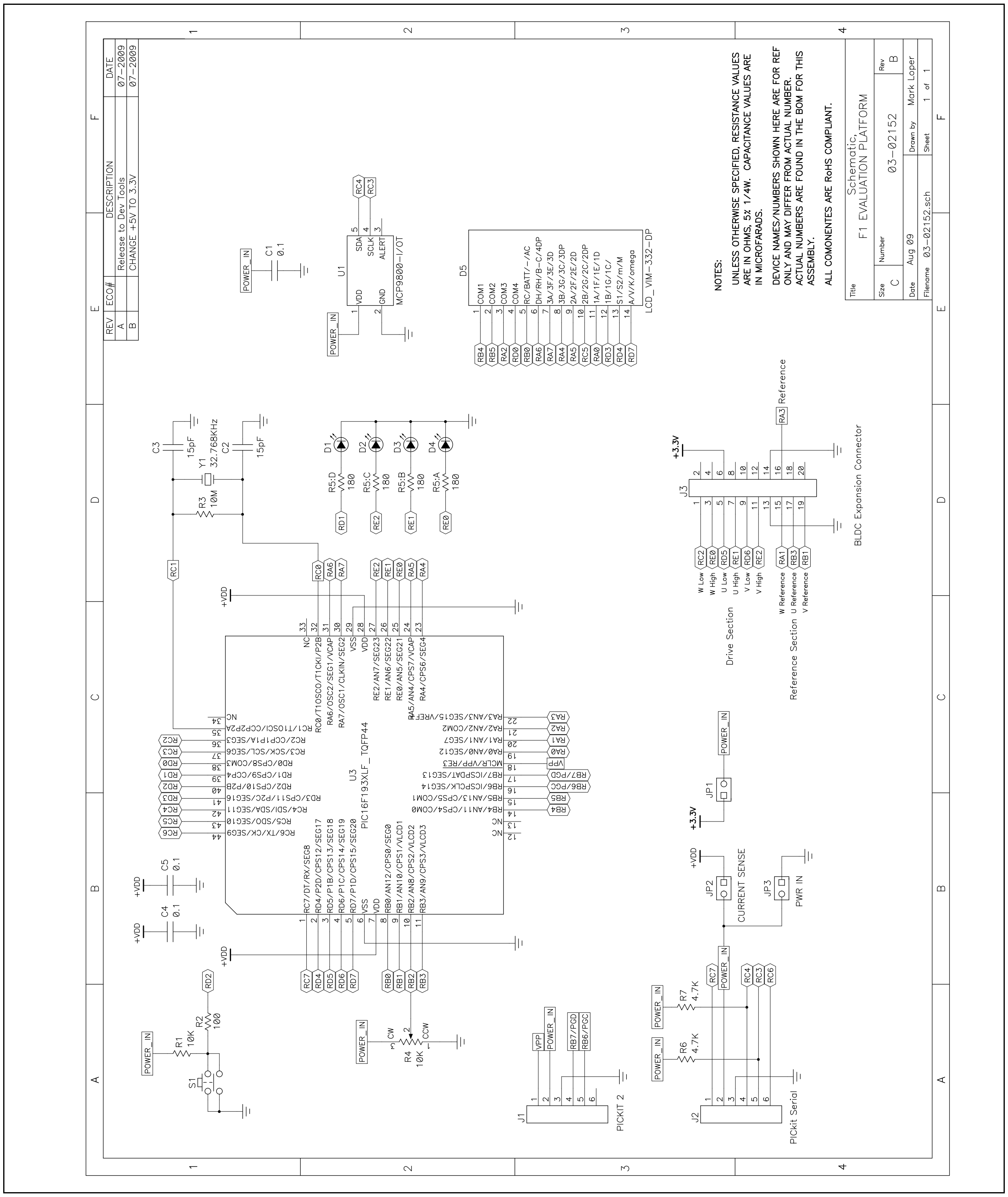
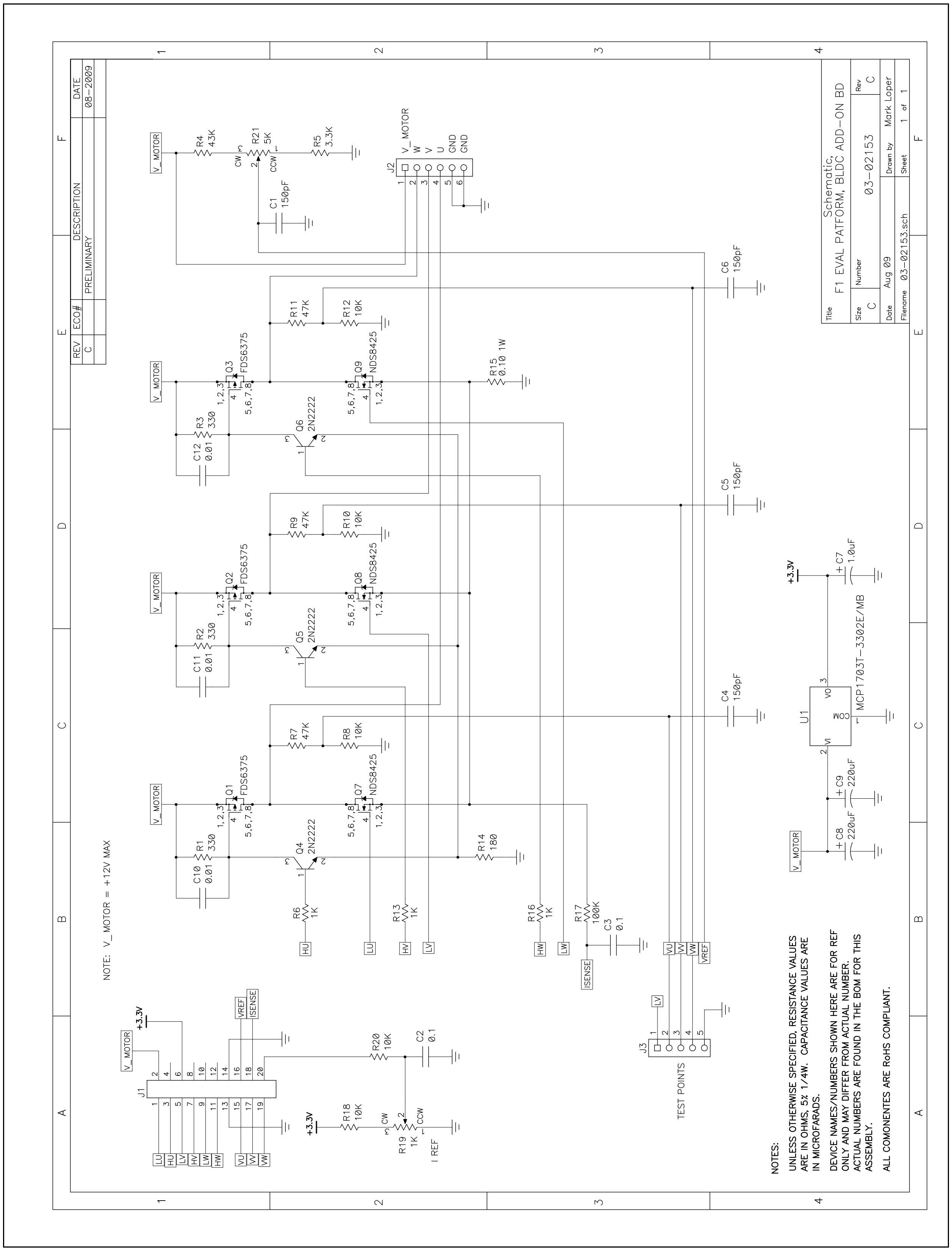


FIGURE A-2: F1 EVALUATION PLATFORM – BLDC ADD-ON BD


Schematic, F1 EVAL PATFORM, BLDC ADD-ON BD
Number 03-02153
Rev C
Size C
Date Aug 09
Drawn by Mark Loper
Filename 03-02153.sch
Sheet 1 of 1
REV C ECO# DESCRIPTION PRELIMINARY DATE 08-2009
NOTE: V_MOTOR = +12V MAX
NOTES:
UNLESS OTHERWISE SPECIFIED, RESISTANCE VALUES ARE IN OHMS, 5% 1/4W. CAPACITANCE VALUES ARE IN MICROFARADS.
DEVICE NAMES/NUMBERS SHOWN HERE ARE FOR REF ONLY AND MAY DIFFER FROM ACTUAL NUMBER. ACTUAL NUMBERS ARE FOUND IN THE BOM FOR THIS ASSEMBLY.
ALL COMONENTES ARE RoHS COMPLIANT.
TEST POINTS
MCP1703T-3302E/MB
Q1 FDS6375
Q2 FDS6375
Q3 FDS6375
Q7 NDS8425
Q8 NDS8425
Q9 NDS8425
Q4 2N2222
Q5 2N2222
Q6 2N2222
R15 0.10 1W
R14 180
R17 100K
I REF
V_MOTOR
+3.3V

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4080

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-6578-300
Fax: 886-3-6578-370

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

03/26/09