Application Note

# Sensorless Block Commutation of a BLDC Motor with the Z32F128 MCU Using the ADC for BEMF Detection

AN038301-0516

## Abstract

This MultiMotor Series application note demonstrates the sensorless block commutation of a BLDC motor with Zilog's Z32F128 ARM Cortex M3 architecture, which is specifically designed to meet challenging design demands. The theory of block commutation for a BLDC motor and the method by which the back electromotive force (BEMF) is detected for this type of motor control is discussed. The BEMF can be detected by using either a three-channel ADC module or three comparators to monitor each phase for the BEMF zero crossing.

This application note addresses the implementation of the ADC peripheral for BEMF detection. For convenience and ease of implementation, a BLDC-type 24V/30W motor capable of running up to 3200 RPM is included in the demonstration kit.

**Note:** The source code file associated with this application note, AN0383-SC01, is available free for download from the Zilog website. This source code has been tested with Keil µVision version 5.18.0.0. Subsequent releases of Keil µVision may require you to modify the code supplied with this application note.

**Note:** The term *MultiMotor* refers to the ability to operate the included BLDC motor with multiple driving scheme program modules. The driving scheme implemented in this application note is the AN0383 program module.

### Z32F128 Series Flash Microcontrollers

The Z32F128 Series of Flash MCUs is based on Zilog's advanced 32-bit ARM Cortex M3 CPU core. The MCUs in this series are optimized for motor control applications and support control of single– and multiphase variable-speed motors. Target applications include consumer appliances, HVAC, factory automation, refrigeration, and automotive applications.

To rotate a 3-phase motor, three AC voltage signals must be supplied and phase-shifted 120 degrees from each other and the MCU must provide six Pulse Width Modulation (PWM) outputs. The Z32F128 Series Flash MCU features a flexible PWM module with three complementary pairs supporting deadband operation and fault protection trip input. These features provide multiphase control capability for various motor types, and ensure safe operation of the motor by providing immediate shutdown of the PWM pins during a fault condition.

## Features

The Z32F128 MCU offers the following features:

- High performance low-power Cortex-M3 core
- 128KB code Flash memory with cache function
- 12KB SRAM
- 3-phase motor PWM with ADC triggering function
  - 2 channels
- 1.5MSPS high-speed ADC with burst conversion function
  - 3 units with 16 channel input
- Built-in Programmable Gain Amplifier (PGA) for ADC inputs
  - 4 channels
- Built-in analog comparator
  - 4 channels
- System fail-safe function by clock monitoring
- XTAL OSC fail monitoring
- Precision internal oscillator clock (20MHz ±3%)
- Watchdog timer
- Six 16-bit general purpose timers
- Quadrature encoder counter
- External communication ports: 4 UARTs, 2 $I^2C$s, 2 SPIs
- High current driving port for UART and photo couplers
- Debug and emergency stop function
- Real-time monitoring of peripherals in debug mode
- JTAG and SWD in-circuit debugger
- Various memory size and package options
  - LQFP-80, LQFP-64
- Industrial grade operating temperature (–40°~ +85°)

A block diagram of this feature set is shown in Figure 1.

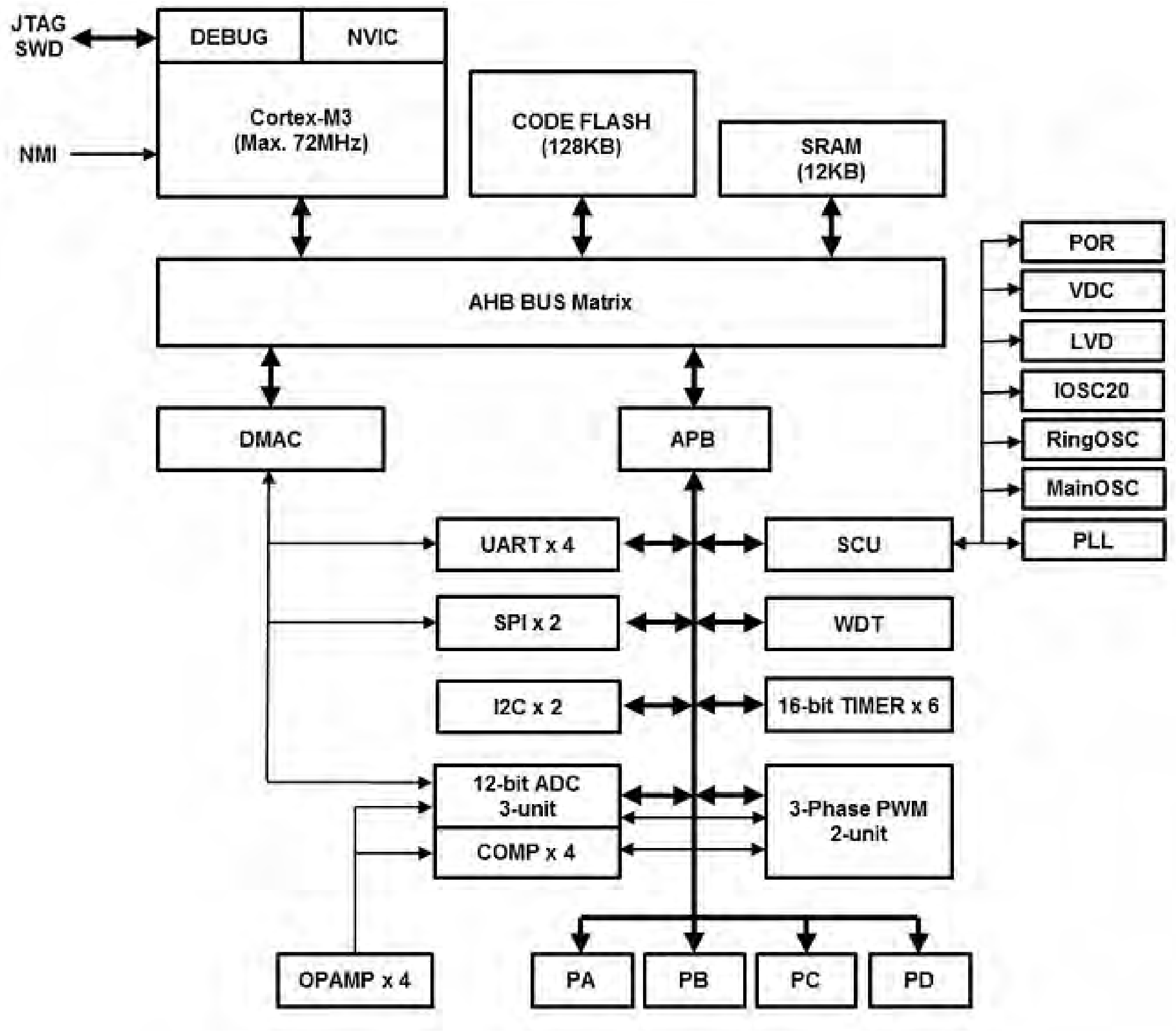


**Figure 1. Block Diagram of the Z32F128 ARM Cortex Architecture**

## Program Features

- Smooth startup with linear speed control characteristics
- Selection for open-loop speed control
- Selection for closed-loop speed control
- Inrush current protection internal to the MCU via PWM shutdown
- Software adjustable over current protection using the ADC
- Selectable motor spin direction
- LEDs for motor operation feedback
- Motor control selectable for Stand-alone or PC-to-hyper terminal control via UART
- IXYS 64N55 N-channel MOSFETS capable of 64A at 50V operating voltage

## Theory of Operation

In block-commutated electric machines, the phase windings are energized every sixty degrees to complete an electrical revolution in six discrete steps. Unlike sinusoidal BLDC machine control, rotor position information is not available during those discrete 60 degree steps. The only rotor position information available in six step commutation is Hall sensor information, or in this application, BEMF information. Either method will provide the rotor position information every sixty degrees. This information is then used to switch the motor windings at the correct commutation angle to achieve the correct alignment between the rotor and stator fields for optimal efficiency in this type of control scheme.

Correct switching of the motors is important because if the motor windings are not switched as close as possible to the rotors' correct position, the motor will not run efficiently and power losses in the motor windings and switching devices will occur. The correct switching of the windings is where the BEMF crosses halfway of the bus voltage on the non-energized phase of the motor. This concept is discussed in further detail later on in this application note.

It is not entirely accurate to state that the BEMF information or Hall information will commutate the motor at the best efficiency and phase-angle because the optimum alignment of the rotor field and the stator field is at 90 degrees to each other, not at 60 degrees. However, since block commutation can only provide rotor information every 60 degrees and nothing in between, there is a commutation-angle error of 30 degrees. Therefore, this error is inherent to this type of motor control. In contrast, sinusoidal commutation does allow for a 90-degree adjustment between the rotor and stator fields and therefore allows for more efficient motor control. However, the phase-angle error in block commutation is accepted for this type of motor control because the means of implementation are relatively simple to commutate a BLDC machine as compared to sinusoidal methods. Additionally, studies of block-commutated machines show that the resulting efficiency and related heat losses due to this phase-angle error are relatively miniscule.

To commutate the BLDC machine, the three phases of the motor are energized so that each phase is shifted 120 degrees to each other, while each phase is not energized for 1/3rd of each electrical revolution. In other words, only two phases are energized in each commutation step, leaving the third phase floating, as shown in Figure 6. The BEMF zero crossing has to be detected at the floating phase to commutate the motor at the correct phase-angle. The knowledge that one electrical revolution consists of six commutation steps is required to ascertain when to detect the BEMF zero crossings. The mechanical revolutions are determined by the number of magnet pole-pairs in the motor, as shown in Equation 1.

**Equation 1.**

Motor frequency as a function of speed and magnet pole pairs:

$$\mathrm{Fmotor} = \frac{\mathrm{RPM} \times \mathrm{N} \times 6}{60}$$

In Equation 1:

*N* is the number of pole pairs
*6* is the number of commutation steps per electrical revolution, and
*60* is seconds per minute.

During each of the six commutation steps, the floating phase is not energized so that there is no current flowing through it and the BEMF from the rotating magnetic field can be induced onto it. The physical relationship of the generated BEMF magnitude and rotational speed is governed by Equation 2.

**Equation 2.**

Magnitude of BEMF voltage as a function of speed:

$$\mathrm{Vbemf} = \mathrm{B} \times \mathrm{L} \times \mathrm{v}$$

In Equation 2:

*B* is the magnetic flux density
*L* is the wire length, and
*v* is the velocity of the rotating magnetic field within the wires of the motor phases.

This equation states that the motor must reach a certain speed before the BEMF voltages are induced onto the floating phase windings so that zero crossings are generated. Therefore, as opposed to Hall commutated machines where the rotor position is known even when the motor is not rotating, the BEMF commutated machine must go through a *forced* commutation ramp until the BEMF is generated.

In this application, a Phase-locked loop (PLL) is implemented using Timer0 to produce the switching frequency until the BEMF is detected.

The implementation of the forced commutation ramp must be such that the rotor can keep up with the rate of speed changes while ensuring that the current is proportional to the rate of speed. Providing the proportionality of the current, the motor has enough current to follow the speed change while avoiding too much current that could heat up the switching devices. The delta between the reference voltage for the zero crossing and the actual zero crossing is fed into this Proportional-Integral Phase-locked loop, whose output produces a value proportional to the BEMF zero crossing error. If the motor is running at the correct phase-angle (commutation point), this error is near zero while the output of the PLL PI is producing a value of some magnitude to become and maintain the switching frequency of the commutation timer and ultimately, the speed of the motor. If the motor is loaded, then initially this BEMF between zero crossing and zero-crossing reference voltage produces an error to produce an offset to this PLL PI output so that the switching frequency resulting in speed is changed to correct the commutation angle.

The quintessence of this PLL PI is that it always satisfies Equation 2 in that the speed changes whenever the motor is loaded to compensate for the resulting change in BEMF. In other words, if the BEMF changes as a result of a load change, so does the speed to ensure the BEMF is always held at the correct phase-angle.

Zero crossings events are located midway between the switching events and are the points at which the floating non-energized phase transitions either from the negative voltage through the neutral point toward the positive voltage, or from the positive voltage through the neutral point toward ground, depending of the direction. These transitions through the neutral point are referred to as *zero-crossing*. To detect the zero-crossings, a voltage divider network must be implemented in the hardware and scaled accordingly to the allowable input voltages of the MCU. Therefore, the voltage dividers implemented in this application consist of a 150K and 4K resistor network from the bus voltage to ground to provide the reference input to one ADC channel, and the remaining three ADC channels are connected each to a voltage divider consisting of a 150K and 10K Ohm resistor network parallel to each of the phases. Each of the BEFM phase voltage transitions from the phase voltage dividers are compared against the midpoint of the bus voltage divider, as shown in Figure 2.

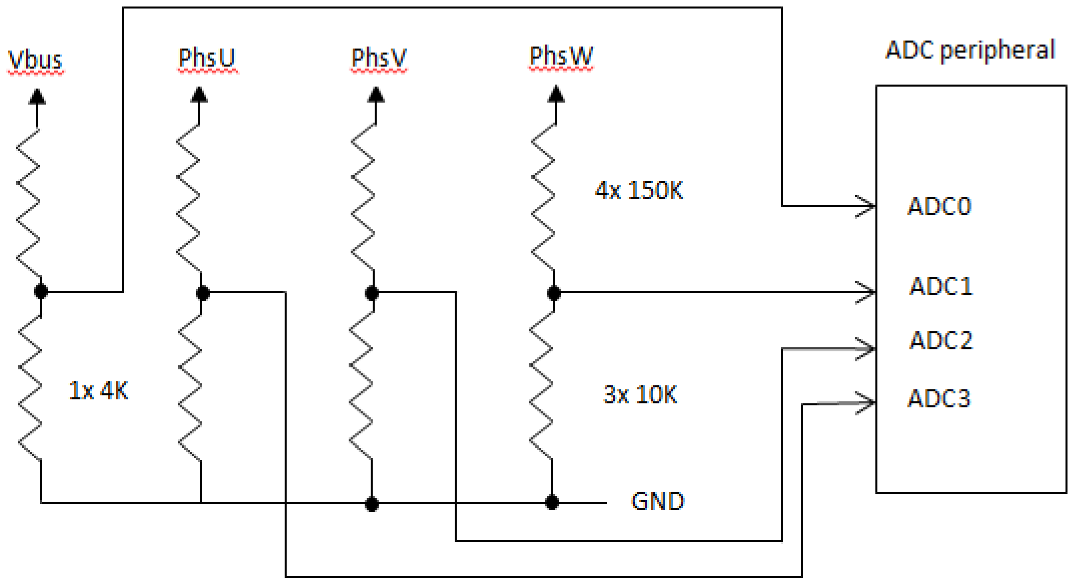


**Figure 2. Three Comparators and the Resistor Network**

In contrast to using the comparators for the BEMF detection when the zero-crossing occurs, the ADC processes the magnitude of the difference between the actual BEMF voltage and the reference voltage so that the output is then used to access the commutation tables for the next commutation step. This concept is further explained under BEMF Detection and Switching on page 10 of this application note.

The commutation sequence for a block commutated BLDC machine is AB-AC-BC- BA-CA-CB so that the phase voltages are either positive or negative with reference to the neutral point. This concept is shown in Figure 5.

Figure 3 shows the resulting three phase voltages across the BEMF dividers and R-C filters (not shown in Figure 2).

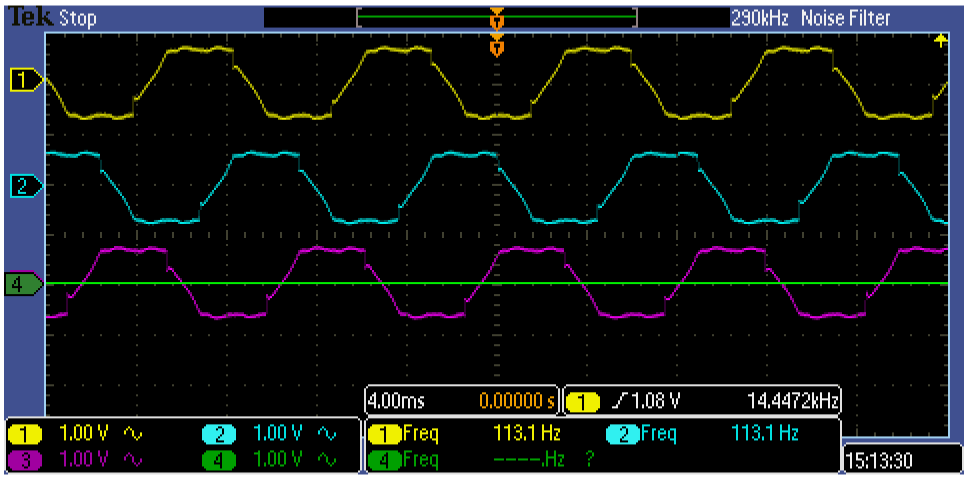


**Figure 3. Three Phase Voltages Across the BEMF Dividers and R-C Filters**

In Figure 3, each of the three BEMF voltages are compared to one reference voltage represented by the green signal trace for zero crossing detection at the BEMF voltages shown in the red circles.

## Implementation

To realize this type of sensorless motor control, Zilog's Z32F128 ARM Cortex M3 PWM peripheral is configured for Up/Down center-aligned PWM, also referred to as *complementary PWM mode*. Alternatively, but not tested to the full extent in this application, the PWM can also be selected for UP-count edge-aligned Independent PWM mode. If the user decides to implement the Independent PWM mode, an R-C filter is used to filter the PWM duty cycle from the BEMF voltages so that the BEMF can be detected.

PWM peripherals can be configured to operate in different modes, all of which have advantages and disadvantages in terms of how the stored energy in the windings is managed, and the effects on the inverter bridge to produce the phase voltages. It is beyond the scope of this application note to discuss all of them. However, it is worthwhile to discuss the Independent versus Complementary PWM mode in more detail.

The term *Independent PWM* was derived from the phenomenon that the phase voltages see only the bus voltage or ground during one PWM period. This occurs because only the high-side duty register is loaded with the duty cycle so that the upper switching device is modulated while the lower switching device on the other phase is fully turned on. The term *complementary PWM* was derived from the phenomenon that the motor phases have modulated both bus voltage polarities during each PWM period, because in addition to loading the high-side duty cycle registers, the low-side duty registers are loaded with the complementary value. Due to this feature, the complementary PWM produces a voltage averaging effect across the motor windings so that the generated BEMF slopes are free of PWM contamination. As a result, the BEMF voltage divider does not necessarily require BEMF R-C filters to detect the BEMF zero-crossings.

The center-aligned PWM mode generated by the PWM peripheral implemented in this application note is shown in Figure 4.

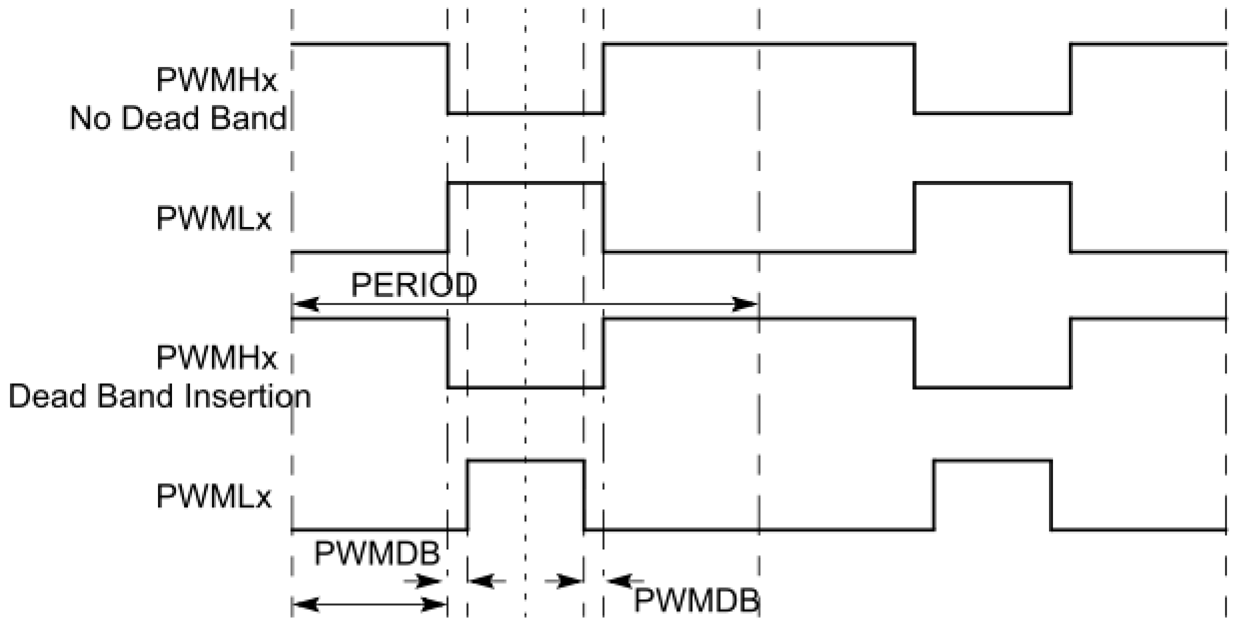


**Figure 4. Relationship of the Duty Cycles with respect to the PWM Period in Center Alignment**

The Independent PWM switching mode is also referred to a *soft-switching* and the Complementary PWM is also referred to as *hard-switching*.

With the PWM implemented and the ability to commutate the motor in either PWM mode, the duty cycles generated by the PWM apply the voltages to the inverter bridge to produce the three 120 degree shifted phase voltages, as shown in Figures 5 and 6.

Figure 5 shows the three phases of the BLDC motor connected to the inverter bridge consisting of six 64N55 IXYS N-channel MOSFETS to energize the motor windings with the

AB-AC-BC-BA-CA-CB sequence to commutate the motor. The arrows indicate the direction of current flow for each commutation step.

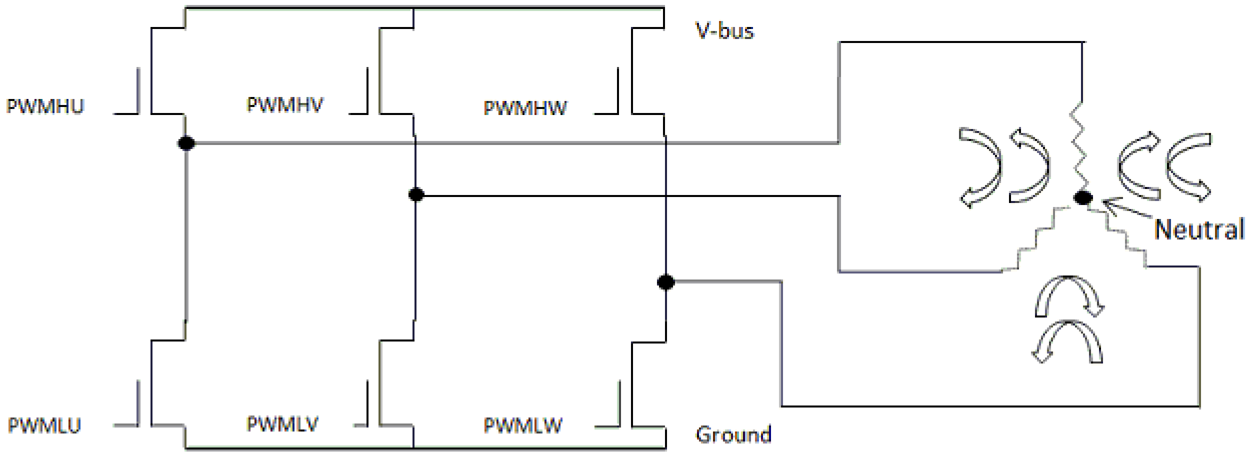


**Figure 5. Inverter Bridge and AB-AC-BC-BA-CA-CB Phase Energizing Sequence with respect to Neutral**

In Figure 6, the three phase voltages are 120 degree shifted to each other, produced by the inverter bridge across the motor windings. The slopes of the phase voltages are the generated BEMFs and are virtually free of PWM contamination.

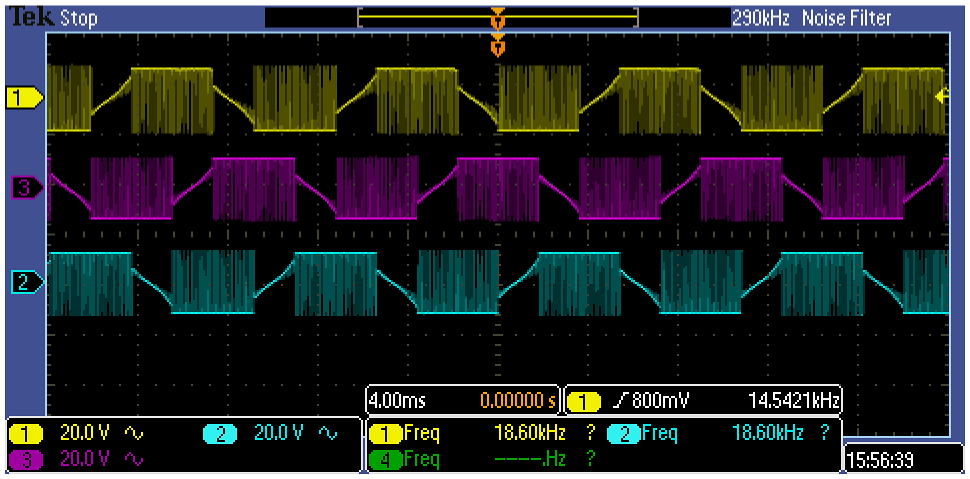


**Figure 6. three 120 Degree Shift Phase Voltages with Complementary PWM Modulation**

## BEMF Detection and Switching

After the BEMF is detected, the BEMF zero-crossings occur midway between commutation switching, as depicted in Figure 7.

The BEMF detection method discussed in this application note uses the Phase-locked loop algorithm by implementing Timer0 such that each rollover period is used alternately to either adjust the BEMF against the zero-cross reference voltage, or to advance the motor into the next commutation state with the calculated switching frequency. These alternate Timer0 rollover intervals occur every 30 degrees, as depicted in Figure 7.

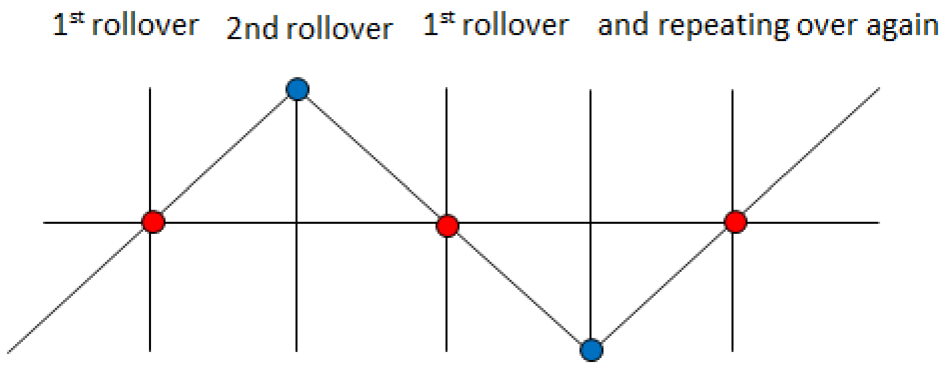


**Figure 7. Red Dots Indicate Zero Cross Detection and Blue Dots Indicate Switching Events**

## First Timer0 Rollover

Timer0 rolls over at the red circle where the BEMF magnitude at the selected phase is compared against the zero-crossing reference voltage and its error is fed into the PI loop (Phase Detector). The output of the PI loop will have some magnitude to be used for the switching frequency calculation in the second Timer0 rollover.

## Second Timer0 Rollover

Timer0 rolls over at the blue circle where the error-output of the PI loop of the previous pass of Timer0 rollover was used to calculate the next commutation step period.

The output of the PLL produces switching periods and is used to calculate the angular frequency using Equation 3.

**Equation 3.**

Switching frequency calculation based on period in terms of timer ticks to be loaded into commutationTimer0:

$$\text{Frequency} = \frac{1}{\text{Period}}$$

The process of locking the PLL PI loop is called *acquisition* and is accomplished when the error between the sensed BEMF and reference zero-crossing voltage is zero, or near-zero. Once the PLLis locked, the output of the PLL PI loop has produced a value to generate the commutation switching of the motor phases at the correct phase-angle (commutation-angle). This algorithm for back-EMF sensing is shown in Figure 8.

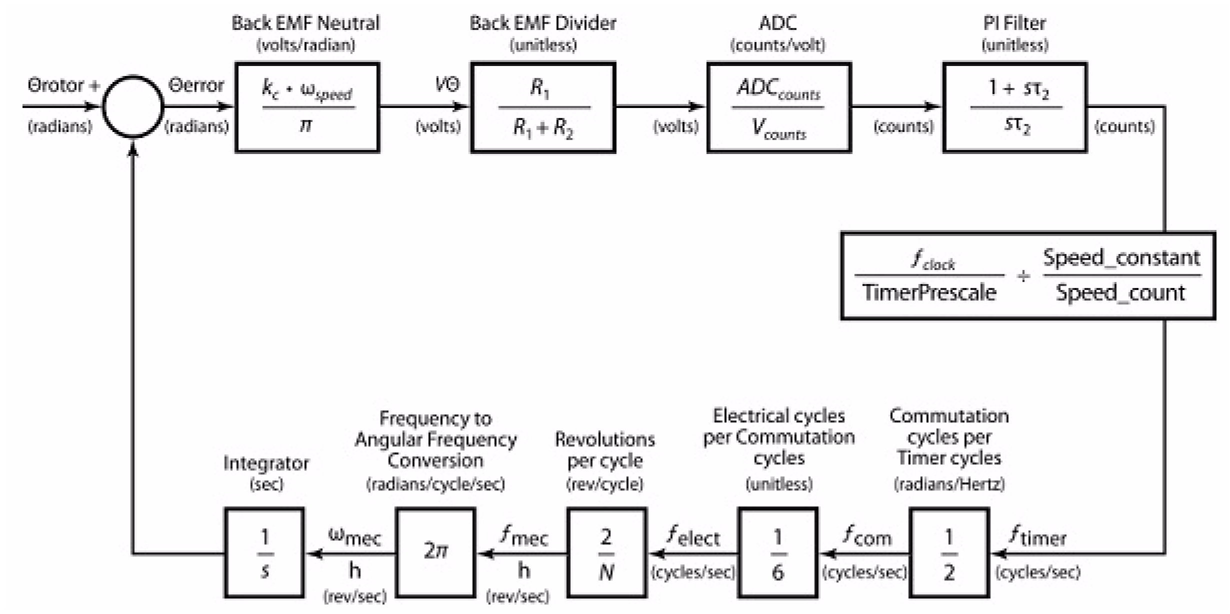

Back EMF Neutral (volts/radian)
Back EMF Divider (unitless)
ADC (counts/volt)
PI Filter (unitless)
Θrotor + (radians)
Θerror (radians)
$\frac{k_c \cdot \omega_{speed}}{\pi}$
VΘ (volts)
$\frac{R_1}{R_1 + R_2}$
(volts)
$\frac{ADC_{counts}}{V_{counts}}$
(counts)
$\frac{1 + s\tau_2}{s\tau_2}$
(counts)
$\frac{f_{clock}}{TimerPrescale} \div \frac{Speed\_constant}{Speed\_count}$
Integrator (sec)
Frequency to Angular Frequency Conversion (radians/cycle/sec)
Revolutions per cycle (rev/cycle)
Electrical cycles per Commutation cycles (unitless)
Commutation cycles per Timer cycles (radians/Hertz)
$\frac{1}{s}$
ωmec h (rev/sec)
$2\pi$
fmec h (rev/sec)
$\frac{2}{N}$
felect (cycles/sec)
$\frac{1}{6}$
fcom (cycles/sec)
$\frac{1}{2}$
ftimer (cycles/sec)

**Figure 8. Back-EMF sensing using the Phase-locked loop (PLL)**

The Phase-locked loop PI used for the BEMF sensing is shown in Figure 9.

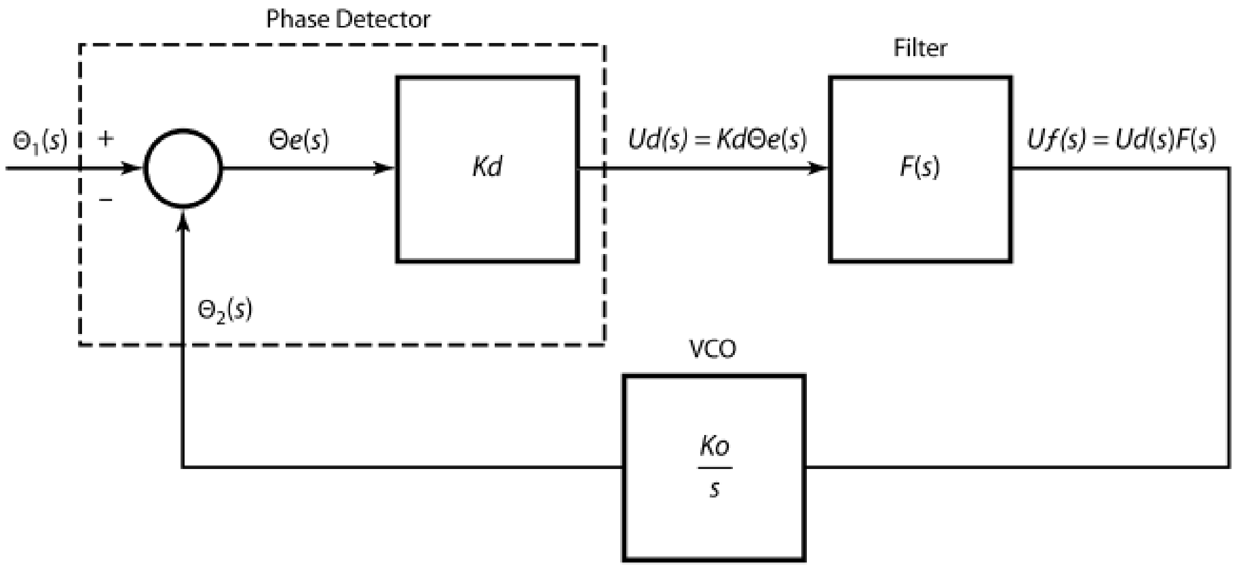


**Figure 9. The Block Diagram of the Proportional-Integral (PI) Filter for Back-EMF Sensing**

## Speed Control

This application note discusses two types of speed control:

- Closed-loop speed control– There is always a feedback between demanded speed and actual speed so that load changes can be compensated for, while maintaining constant power.
- Open-loop speed control – Runs the motor without knowing if the motor is running at the demanded speed and any load changes will not maintain constant power.

Some applications, notably fan and blower applications, may require a specific speed control method. This application allows for the use of either method by commenting or uncommenting the macro `CLOSED_LOOP` in the `main.h` header file.

## Closed Loop PI (Proportional Integral) Loop

The PI loop implemented here does not implement the *D* term, only the Proportional error-term. Integral error terms are used.

As already discussed, the commutation periods are captured by Timer0 in the commutation function. The commutation period is the actual commutation period which is fed into the PI loop and compared against the demand commutation period to produce the PI output to maintain the demanded speed.

To control the speed with this PI loop, the input of the PI loop is driven toward zero while the output of the PI will take on varying magnitude to maintain the speed.

The demand period is derived from Equation 4.

**Equation 4.**

The period is a reciprocal function, a fact used to linearize potential non-linear responses of some BLDC type motors:

$$\text{Period} = \frac{1}{\text{Frequency}}$$

The set period in terms of timer ticks is calculated using Equation 1 and the clock frequency of the MCU is shown in Equation 5.

**Equation 5.**

Timer-ticks as a function of MCU clock frequency and motor frequency:

$$\text{TimerTicks} = \frac{(\text{MCUclock})/(\text{TimerPrescaler})}{(\text{RPM} \times 6 \times \text{N})/60}$$

In the program, the factor *RPM* is taken out of Equation 5 to produce Equation 6, which makes it a constant number so that *RPM* is substituted with the variable *RPM* resulting from the line-equation output as depicted in Equation 6 to be divided into Equation 6 during program execution.

**Equation 6.**

Simplified Equation 5 without the RPM factor and further simplified:

$$\text{TimerTicks} = \frac{(\text{MCUclock})/(\text{TimerPrescaler})}{\text{N}/10}$$

Some BLDC machines, especially fan and blower type applications, can respond with varying degrees of non-linear speed responses to a linear speed demand due to the varying complex impedances among those machines, as depicted in Figure 10.

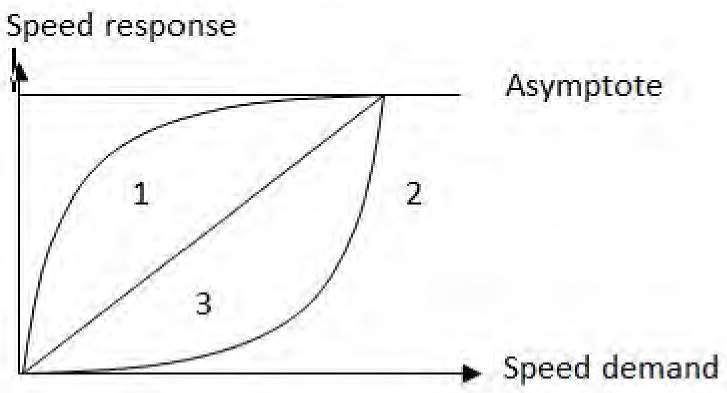


**Figure 10. Non-linear Speed Response to a Linear Speed Demand**

Upon startup, the speed is low but the current is high. As the motor speeds up, the complex impedances combined with the rising BEMF oppose the armature current as the speed reaches its terminal speed, producing an exponential non-linear concave down speed response, as depicted by line 1. However, if an inverse function is applied as a non-linear speed demand, as depicted by line 2, then the speed response is linear, as depicted by line 3. Since the rotational demand periods are of exponential nature, potential non-linearities of some motor types are now compensated.

Figure 11 shows the actual motor response produced in this application by using the linearization method.

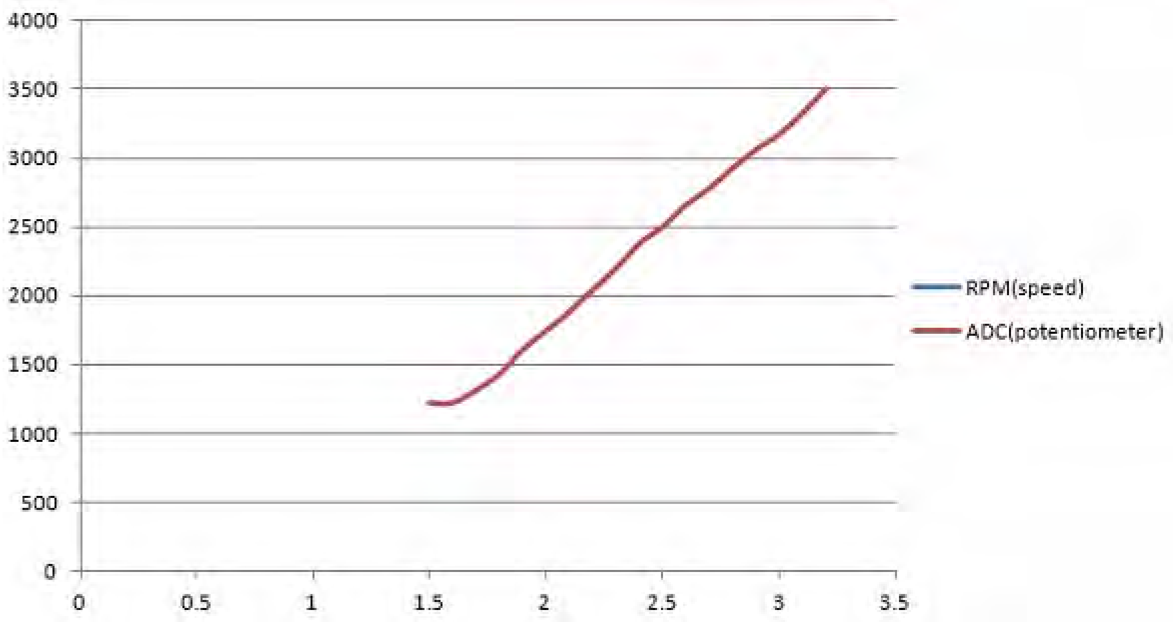


**Figure 11. Speed Linearization**

Equations 1, 3, 4, and 5 produce a non-linear speed demand which results in a linear speed response in closed-loop speed control.

## Open Loop Speed Control

In open loop speed control, the ADC value obtained from the speed control potentiometer becomes the duty cycle value to be applied to the phase windings to change the speed. As implied, no feedback is provided in terms of actual speed to compensate for possible load changes.

## Locked Rotor Protection

In this application note, locked rotor is defined as a condition in which the motor is not able to startup due to the motor shaft being locked. However, the locked rotor function is also invoked when the motor was running but has stalled for some reason. The restart intervals are user definable in the `main.h` file with the macro `LOCKED_ROTOR_RESTART_DLY`. The configurability of this parameter is important because the restart attempt can overheat the switching devices and consequently the motor windings.

## Motor Spin Direction

The motor must come to a stop before applying a direction change to the motor because if the motor is under load and the spin direction is changed at a high speed, the motor will immediately apply the voltages to the phases to follow the new direction. However, because the load has some mechanical inertia, the motor will apply as much current as it can to force the load into the new direction, potentially damaging the load, and possibly the motor. The time it takes for the motor to stop upon a speed direction change is user definable with the `MOTOR_STOP_DETECT` macro in the `main.h` header file.

## Motor Control Parameters

The following macros in the `main.h` header file can be changed to alter the start up or running behavior of the motor:

| | | |
|---|---|---|
| #define CLOSED_LOOP | 1µ | uncommenting this line allows open loop speed control |
| #define P_LOOP | 1µ | uncommenting this line allows Proportional loop only. Otherwise PI loop is selected |
| #define STARTUP | 60µ | this value determines for how long the motor undergoes the startup time |
| #define MIN_RAMP_SPEED | 115µ | This value runs the motor at the minimum duty cycle during the startup time |
| #define KP | 0x0485 | PLL proportional coefficient |
| #define KI | 0x0100 | PLL integral coefficient |
| #define PI_MIN | 180 | PLL output clamping |
| #define PI_MAX | 4000 | PLL output clamping |
| #define SKP | 0x0002 | Speed PI loop proportional coefficient |
| #define SKI | 0x0001 | Speed PI loop integral coefficient |
| #define INTEGRAL_LIM | 25000 | Clamps the integral value to avoid saturation of the PI loop |
| #define INTERNAL_CLCK | 1µ | Uses internal clock with PLL |
| #define UART | 1µ | Enable this line for UART control |
| #define DIR | 1u | Enter 1 for CW and 0 for CCW direction |
| #define COMPLEMENT | 1µ | Uncomment this line for commentary PWM operation |
| #define SETPERIOD | 2.3E6 | Used for closed loop demand period calculations in main.c |
| #define SPEEDCONST | 60E3 | Used for PLL commutation frequency calculations |
| #define MOTOR_STOP_DETECT | 1000µ | This value determines how long the motor is stopped before resuming a change in direction |
| #define COM_MAG_MIN | 127µ | Minimum duty cycle register value |
| #define SPEED_CMD_MIN | 210µ | Minimum speed command duty cycle |
| #define SPEED_RAMP | 80µ | Used in the ADC speed control function as current ramp |
| #define MIN_PWM | 40µ | Used in the ADC speed control function for minimum PWM duty cycle |

## Equipment Used

The following equipment is used for the setup to demonstrate the sensor-less BEMF commutation technique. The first four items are included in the MultiMotor Development Kit (ZMULTIMC100ZCOG).

- MultiMotor Development Board (99C1358-0001G)
- 24V AC/DC power supply
- LINIX 3-phase 24VDC, 30W, 3200RPM BLDC motor (45ZWN24-30)
- Opto-isolated UART-to-USB adapter (99C1359-001G)
- Z32F128 MultiMotor MCU Module (99C1461-001G) – Order separately
- Keil ULINK2 debugger – Order separately
- Digital oscilloscope
- PC with Internet access and at least two open USB ports

## Hardware Setup

Figure 12 illustrates the application hardware connections required to operate the motor with block commutation.

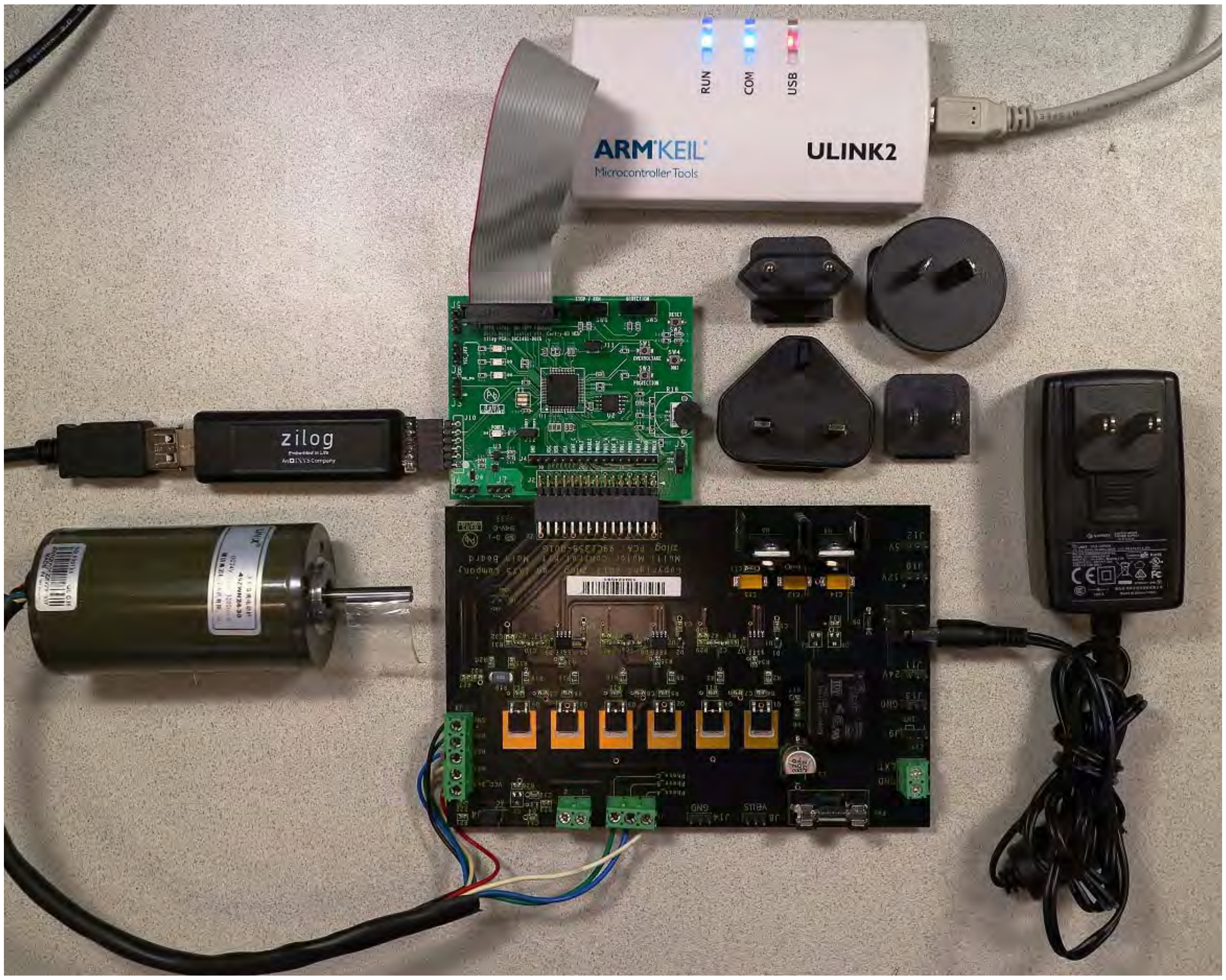
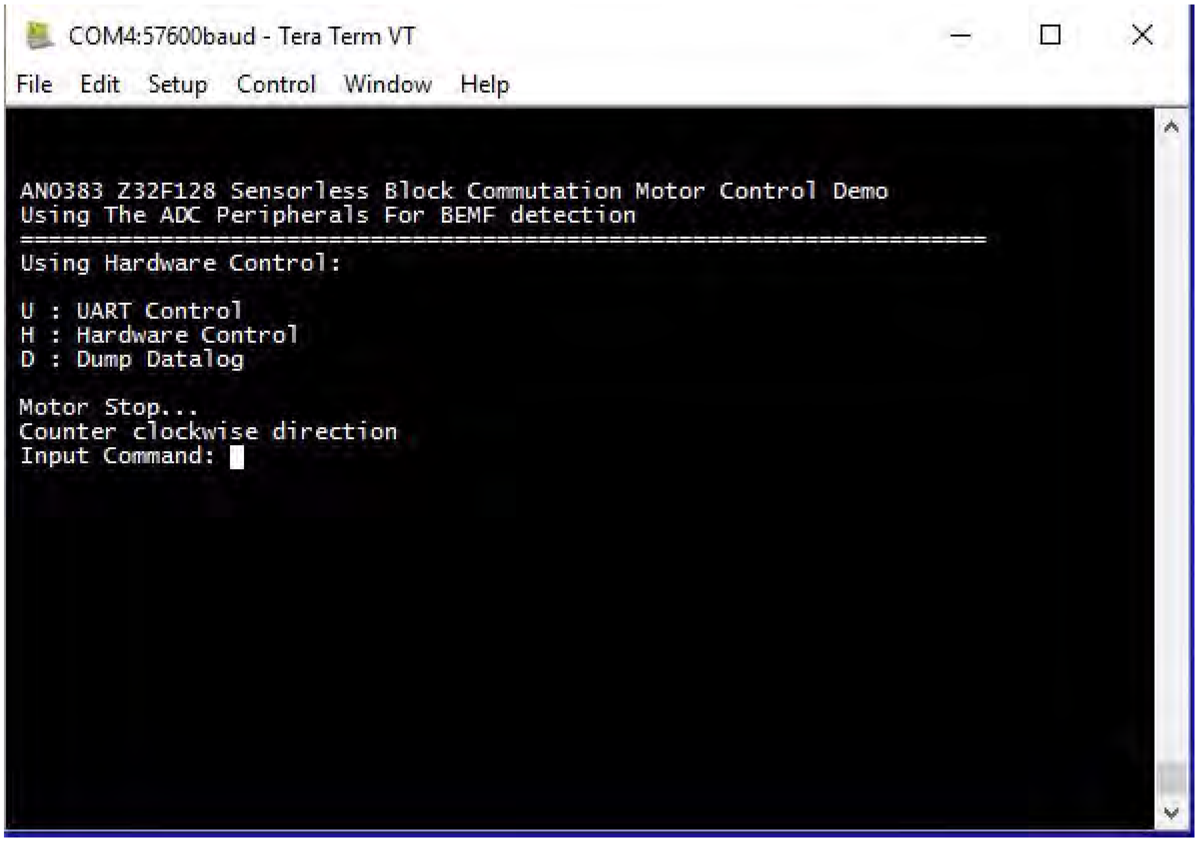


**Figure 12. The MultiMotor Development Kit**

## Testing Procedure

Observe the following procedure to test the motor in sensorless block commutation on the Z32F128 MCU Module.

1. Download and install Keil MDK-ARM µVision IDE version 5.18.0.0 (or newer) on your PC from the Keil website.
2. Download the AN0383-SC01.zip source code file from the Zilog website and unzip it to an appropriate location on your PC.
3. Connect the hardware as shown in Figure 12.
    a. The cables from the Keil ULINK2 debugger and the UART-to-USB adapter must be connected to two of the PC's USB ports.
    b. Download and install the drivers for the Keil ULINK2 and the UART-to-USB adapter, if required.
4. Power up the MultiMotor Series Development Board using the 24V DC adapter that is included in the Kit.
5. Using a serial terminal emulation program such as HyperTerminal, TeraTerm, or Real-Term, configure the serial port to 57600-8-N-1-N. A console screen should appear on the PC which will show the status of the motor and allow changes to the motor's operation.
6. Launch Keil µVision version 5.18.0.0 (or newer), select **Open Project** from the **Project** menu, browse to the directory on your PC to which the AN0383-SC01 source code was downloaded, locate the `AN0383_SC01` µVision 5 file, highlight it, and select **Open**.
7. Ensure that the RUN/STOP switch on the Z32F128 MCU Module is in the STOP position.
8. In Keil µVision, compile and flash the firmware to the Z32F128 MCU Module by selecting **Rebuild All target files** from the **Project** menu. Next, select **Debug** → **Start/Stop Debug Session**, followed by **Debug** → **Run**.
9. Set the RUN/STOP switch on the Z32F128 MCU Module to RUN. The motor should begin turning.
10. In the GUI terminal console, enter the letter `U` to switch to UART control; a menu similar to the example shown in Figure 13 should appear. As a result, commands can now be entered using the console to change the motor's operation.


COM4:57600baud - Tera Term VT
File Edit Setup Control Window Help

```
AN0383 Z32F128 Sensorless Block Commutation Motor Control Demo
Using The ADC Peripherals For BEMF detection
=====================================================================
Using Hardware Control:

U : UART Control
H : Hardware Control
D : Dump Datalog

Motor Stop...
Counter clockwise direction
Input Command: █
```

**Figure 13. GUI Terminal Console Menu**

**Note:** A data logger is implemented in this application as an example how to track and store into non-volatile memory motor specific parameters such as:
– Speed
– Direction (CW/CCW)
– Total motor run time (seconds, minutes, days and years)
– Example function to measure temperature
– Example function to measure the bus voltage

The data stored in this data logger can be obtained by entering **D** in the hyper terminal window when the motor is operated with UART.

## Summary

The sensor-less block commutation of a three phase BLDC motor was realized with the implementation of Zilog's advanced Z32F128 Cortex M3 architecture, which is specifically designed for motor control applications. Together with the inverter bridge consisting of 64N55 IXYS N-channel MOSFETS, this application forms the synergy of hardware, firmware, and microcontroller for motor control.

This application note discusses the sensorless commutation of a three phase BLDC motor using three ADC channels for BEMF sensing with the Z32F128 ARM Cortex M3 architecture. The functional theory of utilizing the ADC for BEMF sensing using a Phase-locked loop and the setup of this architecture's PWM module to generate PWM duty cycles in complementary fashion is discussed to explain how the phases of the motor are energized using the inverter bridge consisting of six 64N55 IXYS N-channel MOSFETS, while the ADC modules monitor the three phases for BEMF events.

A Proportional Integral speed control loop (PI loop) is implemented in this application to control the speed in closed loop. Alternatively, the user can select the open-loop speed control method. Calculations for computing the demand speed of the motor are shown. A brief description of the characteristics of some motors' response to the speed control based on their electro-mechanical composition and motor type is also included in this application note.

In addition to the display of current motor conditions such as speed and direction, a data-logger is implemented for users interested in historical records of motor control parameters which are stored on an external Flash Spansion IC.

## References

Documents associated with the Z32F128 Series of products are listed below. Each of these documents can be obtained from the Zilog website by clicking the link associated with its document number where indicated

- Z32F128 MCU Product Specification (PS0345)
- Z32F128 Evaluation Kit User Manual (UM0277)
- MultiMotor Series Development Kit Quick Start Guide (QS0091)
- MultiMotor Series Development Kit User Manual (UM0262)

# Appendix A. Schematic Diagrams

Figures 14 and 15 present the schematic diagrams for the Z32F128 MCU Module.

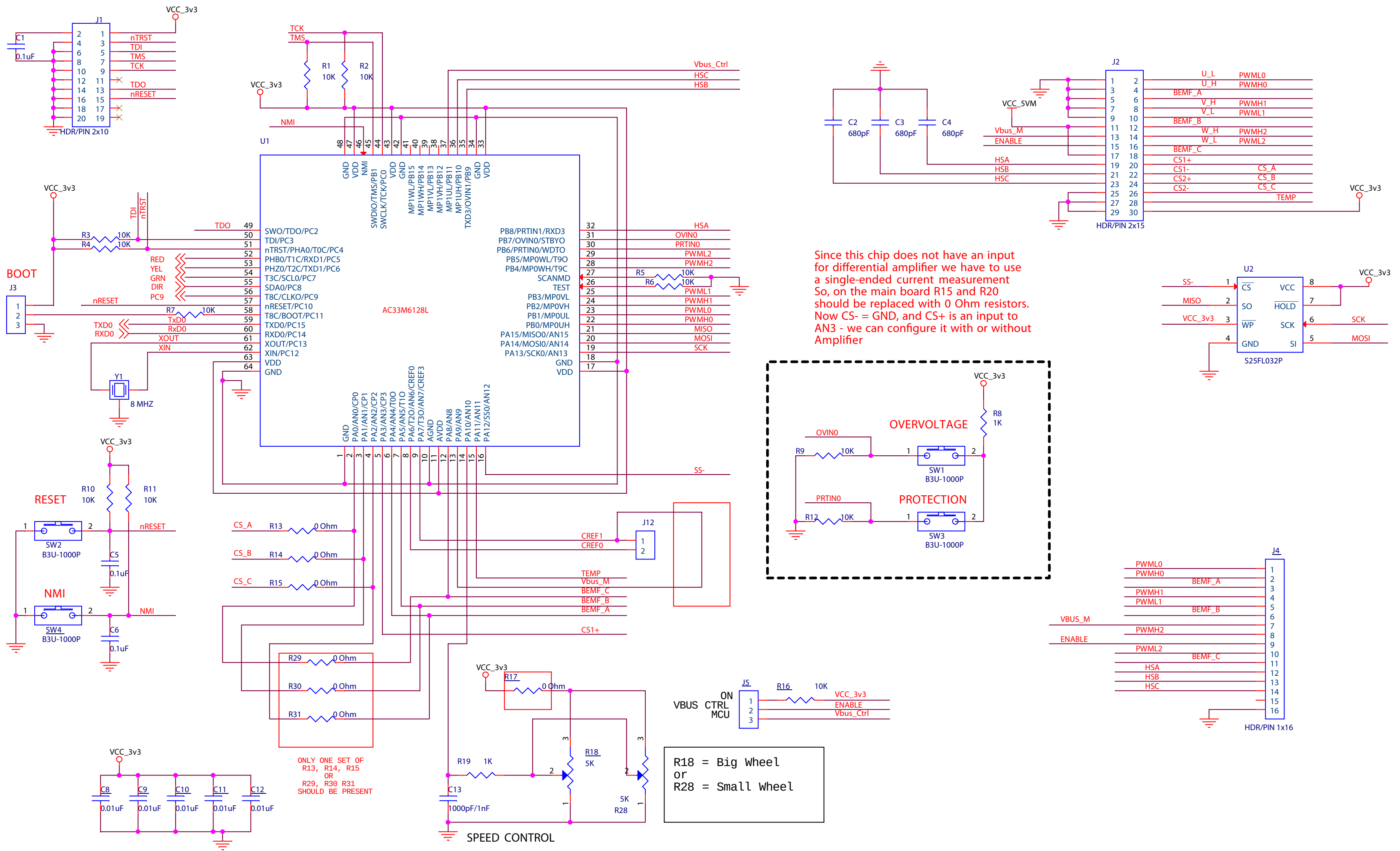

Since this chip does not have an input for differential amplifier we have to use a single-ended current measurement So, on the main board R15 and R20 should be replaced with 0 Ohm resistors. Now CS- = GND, and CS+ is an input to AN3 - we can configure it with or without Amplifier
AC33M6128L
BOOT
RESET
NMI
OVERVOLTAGE
PROTECTION
SPEED CONTROL
ONLY ONE SET OF R13, R14, R15 OR R29, R30 R31 SHOULD BE PRESENT
R18 = Big Wheel or R28 = Small Wheel
ON VBUS CTRL MCU
S25FL032P
HDR/PIN 2x10
HDR/PIN 2x15
HDR/PIN 1x16

**Figure 14. Z32F128 MultiMotor MCU Module, #1 of 2**

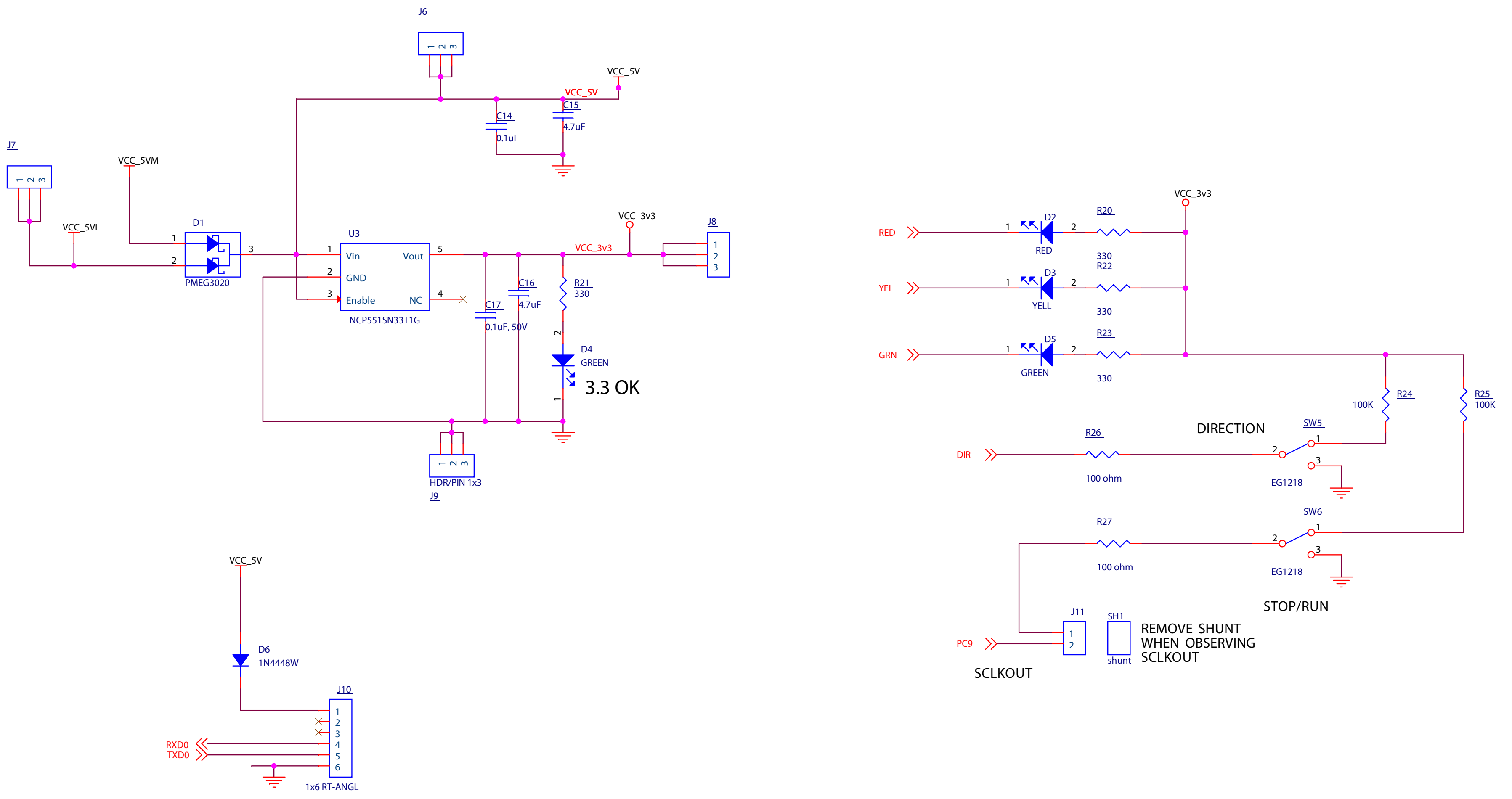


**Figure 15. Z32F128 MultiMotor MCU Module, #2 of 2**

Figures 16 and 17 present the schematic diagrams for the MultiMotor Main Board.

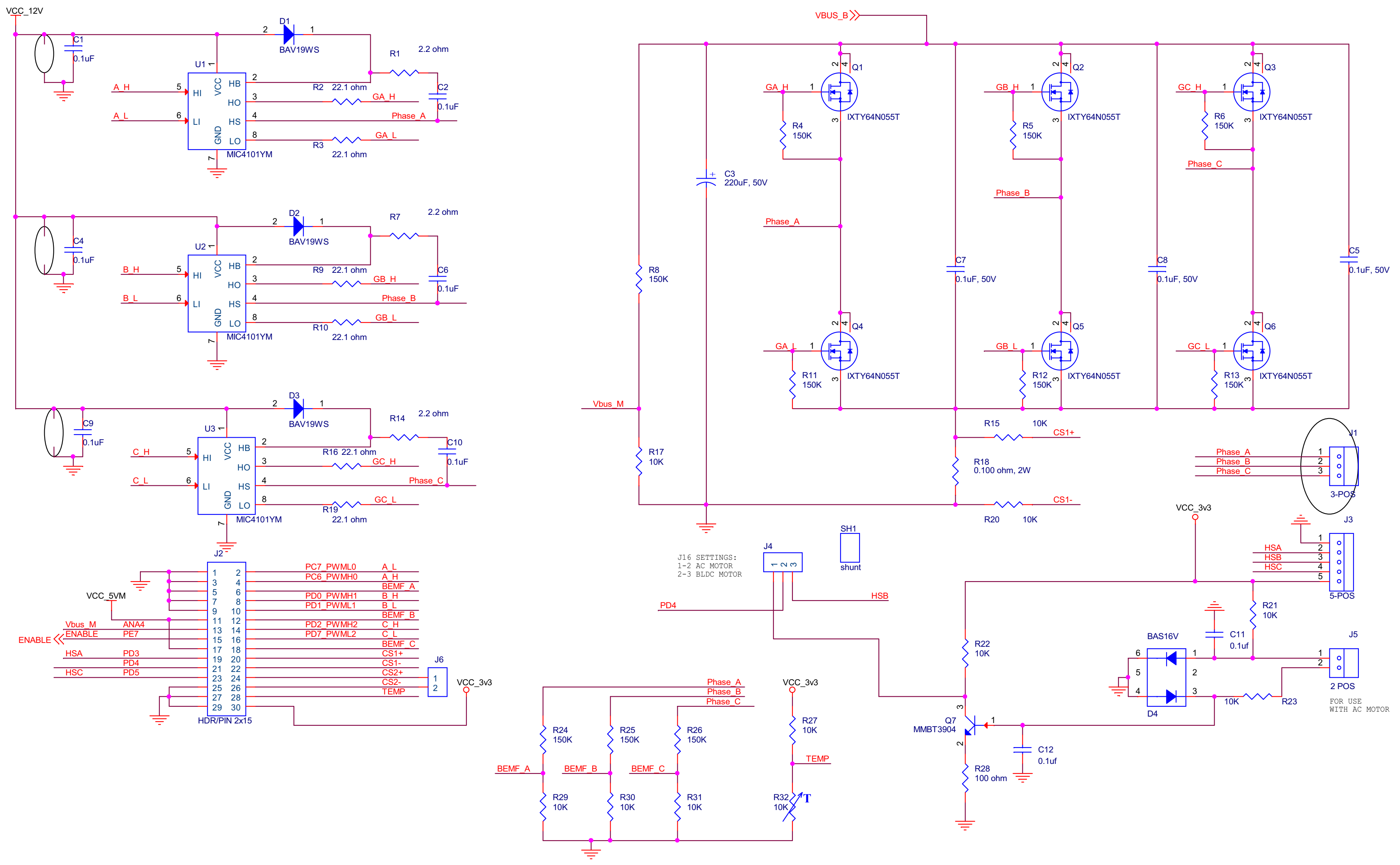


**Figure 16. MultiMotor Development Board, #1 of 2**

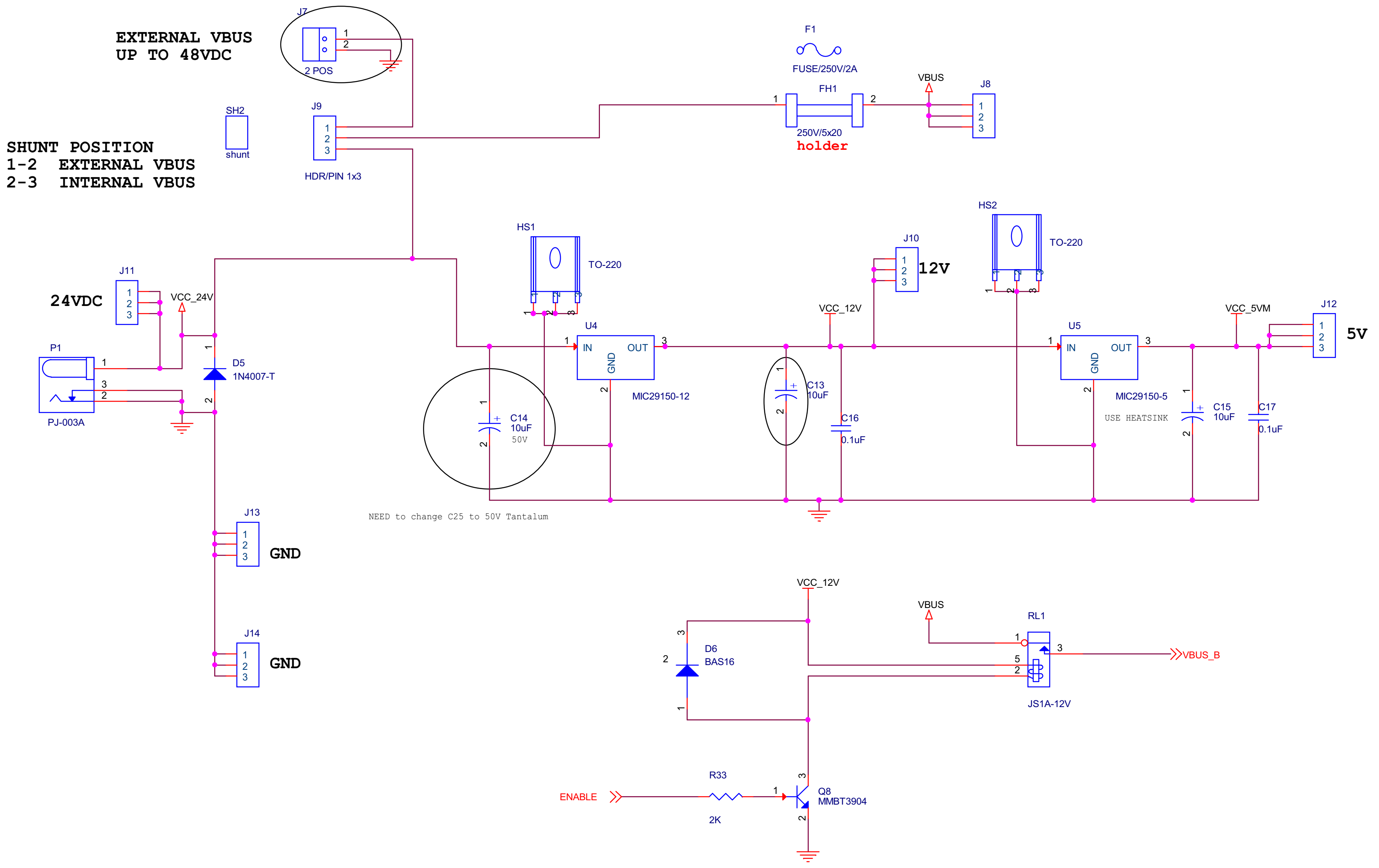


**Figure 17. MultiMotor Development Board, #2 of 2**

# Appendix B. Flow Chart

Figure 18 presents the typical flow of the space vector control routine.

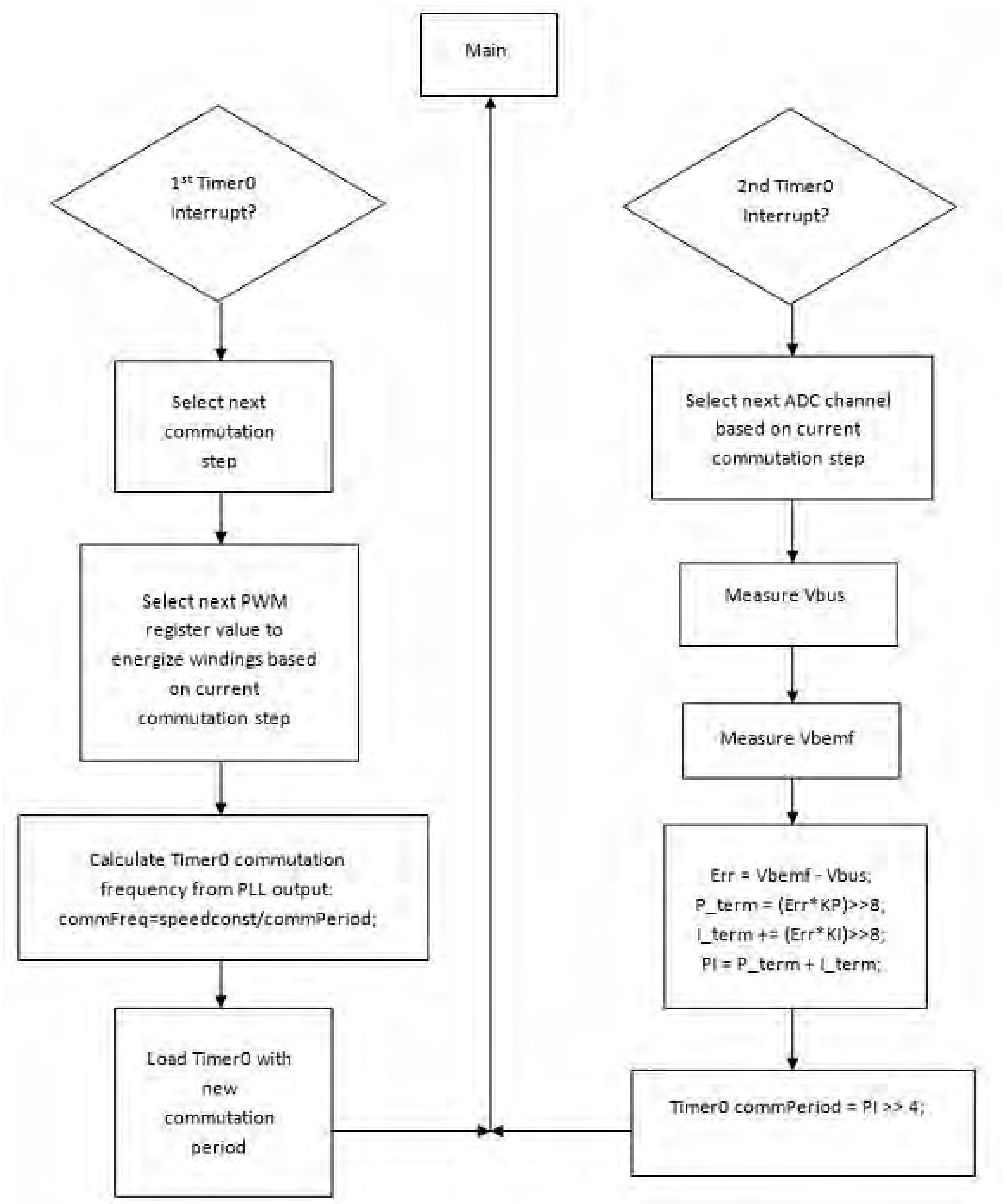

Main
1st Timer0 Interrupt?
Select next commutation step
Select next PWM register value to energize windings based on current commutation step
Calculate Timer0 commutation frequency from PLL output: commFreq=speedconst/commPeriod;
Load Timer0 with new commutation period
2nd Timer0 Interrupt?
Select next ADC channel based on current commutation step
Measure Vbus
Measure Vbemf
Err = Vbemf - Vbus; P_term = (Err*KP)>>8; I_term += (Err*KI)>>8; PI = P_term + I_term;
Timer0 commPeriod = PI >> 4;

**Figure 18. Commutation Timer0 Flowchart**

## Customer Support

To share comments, get your technical questions answered, or report issues you may be experiencing with our products, please visit Zilog's Technical Support page at http://support.zilog.com.

To learn more about this product, find additional documentation, or to discover other facets about Zilog product offerings, please visit the Zilog Knowledge Base at http://zilog.com/kb or consider participating in the Zilog Forum at http://zilog.com/forum.

This publication is subject to replacement by a later edition. To determine whether a later edition exists, please visit the Zilog website at http://www.zilog.com.

**Warning:** DO NOT USE THIS PRODUCT IN LIFE SUPPORT SYSTEMS.