THESIS

OFFLINE DETECTION OF BROKEN ROTOR BARS IN AC INDUCTION MOTORS

Submitted by

Craig Stephen Powers

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Fall 2015

Master's Committee:

Advisor: George J. Collins

Steven C. Reising
Hiroshi Sakurai

ABSTRACT

OFFLINE DETECTION OF BROKEN ROTOR BARS IN AC INDUCTION MOTORS

The detection of the broken rotor bar defect in medium- and large-sized AC induction machines is currently one of the most difficult tasks for the motor condition and monitoring industry. If a broken rotor bar defect goes undetected, it can cause a catastrophic failure of an expensive machine. If a broken rotor bar defect is falsely determined, it wastes time and money to physically tear down and inspect the machine only to find an incorrect diagnosis. Previous work in 2009 at Baker/SKF-USA in collaboration with the Korea University has developed a prototype instrument that has been highly successful in correctly detecting the broken rotor bar defect in ACIMs where other methods have failed. Dr. Sang Bin and his students at the Korea University have been using this prototype instrument to help the industry save money in the successful detection of the BRB defect.

A review of the current state of motor conditioning and monitoring technology for detecting the broken rotor bar defect in ACIMs shows improved detection of this fault is still relevant. An analysis of previous work in the creation of this prototype instrument leads into the refactoring of the software and hardware into something more deployable, cost effective and commercially viable.

ACKNOWLEDGMENTS

I thank my wonderful family, who have been ever so patient in my multi-year journey while seeking my master's degree and also working full time. They've put up with many years of their dad isolating himself evenings and weekends to study, to finish his assignments, and to work on his thesis.

I also thank my wonderful professor Dr. George Collins for his wise, witty, and always inspirational lectures in my undergraduate days. He later took me on as a graduate student and allowed me to work on the things pertaining to my interests. He teaches that the value of engineering is not just solving engineering problems, but more importantly on doing things cheaper, faster and better for the needs of society. I also thank my master's committee members who graciously gave their time reviewing this work.

I thank the late Dr. Ernesto Wiedenbrug, who was wonderfully knowledgeable in all things motors. He was a dear co-worker and friend whom I always looked forward to visiting each and every day to share his many inspirational ideas on improving condition monitoring testing. The wonderful memories of him helping me destroy motor bearings for defect testing, lecturing me on motor torque calculations, horsepower, DQ versus DQZ transforms, etc., will never be forgotten.

I thank my young co-worker Adam Bierman for graciously loaning me his Textronix 2235 oscilloscope to use at home to see the PWM signals during my thesis project. I thank my former employer Advanced Energy Inc. for paying for my master's classes, and my boss Joel Blackburn for his continued support in furthering my education. Finally, I thank my current employer SKF USA for providing development kits, motors and a thesis topic.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1 Introduction

AC induction motors (ACIMs) are used throughout the world to perform work by converting electricity into mechanical power. ACIMs range from very small fractional horsepower to the very large of thousands of horsepower. They operate in severe and hazardous conditions and are expected to perform with high reliability. They are used in applications of pumping, chopping, crushing, machining, centrifuging, and more.

As with most things, failures are unavoidable even for the ACIM. These complex electro-mechanical devices suffer from bearing, cooling, winding, and rotor failures [1]. According to published surveys [2] [3], induction motor failures can be grouped by bearing, stator, rotor and other failures.

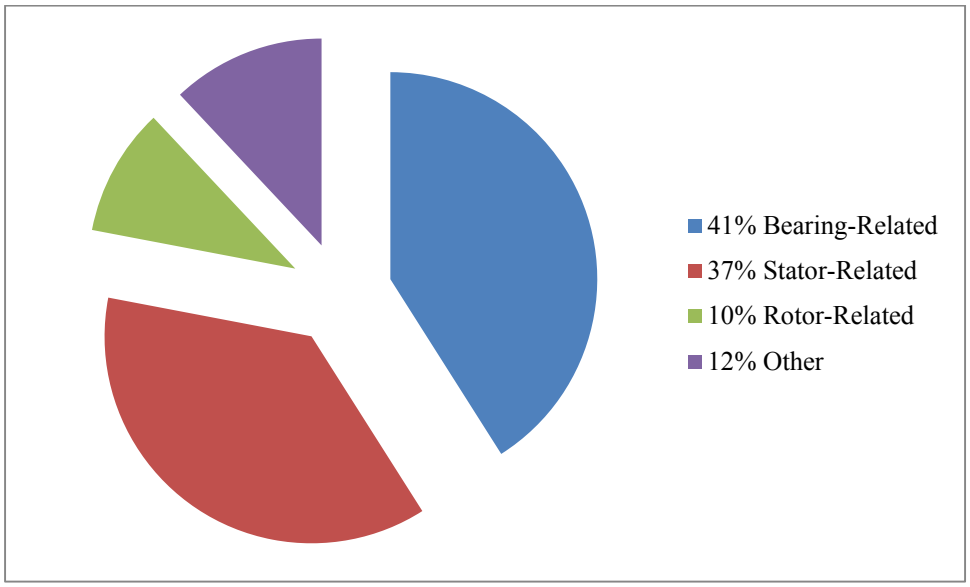


Figure 1: IEEE-IAS Percentage of failure by component

One of the most difficult motor faults to diagnose accurately in the AC induction motor is the broken rotor bar fault. The studies [2] [3] show that approximately 9-10% of induction motor failures are rotor-related. Broken rotor bars can lead to catastrophic and severe damage to a

motor if not repaired early [4], so accurate diagnosis is a necessity. Tearing down a motor for physical inspection is a costly and time-consuming affair which takes the motor offline for an extended period of time and must be avoided unless absolutely necessary. Plant operators need a simple, inexpensive and reliable method to determine whether the rotor bars in their motors are 'healthy' or 'faulty'.

Previous work [5] created a prototype offline broken bar tester. This previous work also has successfully shown the merits of this prototype offline tester in detecting broken rotor bars where other methods have failed.

The goal of this thesis is to improve on the original prototype design in the following areas:

1. Capability to operate off of a standard 15-20A, 110V wall outlet
2. Motor bump capability to allow subtraction of core hysteresis effects
3. Reducing the bill of material cost by replacing National Instrument FPGA test instruments with an inexpensive microprocessor reference design
4. Replacing MATLAB software for post analysis of data with production C/C++/C# code to integrate with an existing SKF motor tester

In the summer of 2009, the Korea University and Baker Instruments (now SKF-USA) collaborated to build a prototype offline broken bar tester. The term 'Magic Box' was coined by a customer for the prototype box which seemed to magically detect broken rotor bars in an AC induction motor without even spinning or disconnecting the motor from its mechanical load. The 'Magic Box' concept is to provide a magnetic angle to the stator and then continually reverse it at a test frequency from 1 to 60 Hz. These stator signals are influenced by the magnetic coupling

to the rotor of the motor. The voltage and current signals from the stator are simultaneously sampled and recorded. Using the Park transform [5], the reference frame of the voltages and currents are transformed into the d-q axis. Then using the fast Fourier transform, each of the d-q axis voltage and current time-domain signals are converted into their frequency-domain spectra. The impedance versus frequency for the standard motor model can be computed by dividing the voltage spectrum by the current spectrum. The magnetic stimulus angle is then advanced and impedance measurements are again made until the complete set of magnetic angles has been tested. The reversing test frequency affects the probing depth of the magnetic field into the rotor core. A slower test frequency has a larger skin depth and will probe deeper into the rotor's magnetic core. A motor with defective rotor bars shows variations in the impedance versus the applied magnetic field angle. A motor with good rotors bars shows constant impedance versus the applied magnetic field angle. The details are covered in US Patent Application US20110191034.

Chapter 2 Literature review

Methods for detecting broken rotor bars can be grouped into the categories of online and offline testing. Online testing is where the motor is connected to an AC line or VFD drive while monitoring the running motor to make measurements. Offline testing involves turning off and disconnecting the power source from the motor before proceeding to make measurements on the motor.

Online testing offers the benefits of not having to take the motor offline while testing is taking place. Current online testing drawbacks include problems in separating the electrical drive signal from the test signals, load variations imposing signals on the stator's electrical signals and requirements of a minimum load [30-100%] applied to the motor [1]. In addition, there can be safety concerns when connecting and sampling running motors due to the high voltages and currents if provisions have not already been put in place such as potential and current transformers to step down the voltages and currents to a safe level for measurement. One online method for detecting broken rotor bars involves detecting the twice-slip frequency components in the vibration or current spectra on the motor under test. Determination of broken rotor bars using twice-slip frequency suffers from air-ducted false positive indications [6] where the number of axial air ducts is an integer multiple of the number of poles in the motor. Several methods exist to analyze the startup current to test for BRBs [7] [8] [9]. A method using wavelet analysis of the startup transient currents improves the reliability of detecting BRBs [10] in situations where the axial air ducts influence the current spectra.

Offline testing for BRBs can be grouped into tests that involve disassembly of the motor and ones that do not. Offline testing offers advantages that additional tests for insulation quality,

resistance and inductance symmetry between phases and other motor health-determining tests can also be run. The single phase rotation test (SPRT) is an offline method for detecting BRBs without disassembly of the motor. A single-phase low-voltage AC source is connected to two of the three phases of the motor. The motor is manually rotated while the current is monitored. As the rotor is rotated, a change in the AC current will occur due to rotor impedance variations caused by BRBs. The technique has also been extended to detecting static and dynamic eccentricity faults [11] . Two additional offline tests that involve the disassembly of the motor are the growler test and the rated rotor flux test [12].

Chapter 3 Realization of refactored system

Three-phase AC induction motors are driven by three sine wave voltages each shifted by 120 degrees from each other. The stimuli from each leg drive coils offset rotationally around the stator. These sine waves produce a bi-polar magnetic field with an angle defined by the rotational position of the coil around the rotor. Each stimulus can be represented by a vector with magnitude and direction proportional to the magnitude and phase of the voltage applied to that leg. The individual magnetic fields produced by each leg vectorially add together to form a single rotating magnetic field with a constant magnitude. The graphic below shows the magnetic field produced in the stator for each of the three phases and how they add together to form a single rotating field in a two-pole motor.

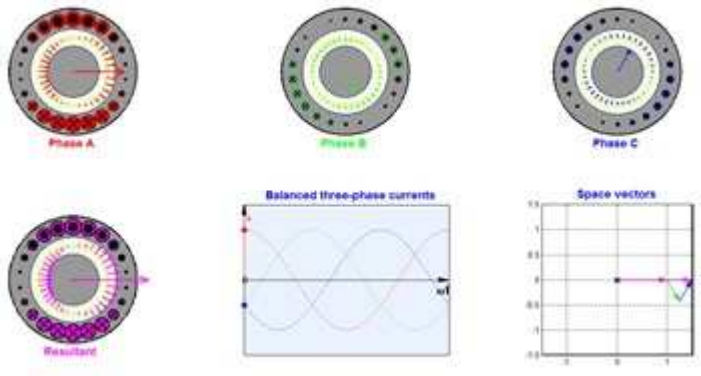


Figure 2: Diagram showing magnetic field vector addition of each coil of a 2-pole motor into a single magnetic vector of angle 0 degrees

An animated version of the above figure was prepared by Riaz [13].

If the magnitudes of the three AC phases are frozen in time, a constant magnetic field will be induced through the stator at a given angle. Simply by multiplying the three magnitudes by negative one, the magnetic field can be flipped 180 degrees in the stator. Repeating this process at some frequency produces a (reversing) pulsating magnetic field at the given stator

angle. Since the stator's magnetic field is only reversing but not rotating, no rotational torque is induced into the rotor to cause the motor's rotor to rotate.

The pulsating magnetic vector at θ can be produced with pulse width modulation with the three-phase voltage references set using the mathematical equations below [5]:

$$Vas(\phi,\acute{\omega})=v\cos(\phi)squ(\acute{\omega}t) \quad (1)$$

$$Vbs(\phi,\acute{\omega})=v\cos(\phi+120°)squ(\acute{\omega}t) \quad (2)$$

$$Vcs(\phi,\acute{\omega})=v\cos(\phi+240°)squ(\acute{\omega}t) \quad (3)$$

where V is the excitation voltage magnitude and squ(ώt) represents the square wave pulsating at the excitation frequency ώ. The pulsating field induces voltage in the rotor bars resulting in rotor current flow, but the motor does not rotate since the average induced torque is zero.

As the square wave is applied to each leg of the motor, the three voltage and three current waveforms can be acquired. Using a variant of the Park transform [5], the voltages and the currents can be transformed into a time-invariant d-q coordinate system. The impedance versus frequency can be computed by dividing the FFT of Vq by the FFT of Iq. Finding the value at the fundamental frequency will yield the standard motor model's equivalent impedance. The real part represents the resistance of the motor model and the imaginary portion represents the inductive part of the motor model. The graph data below shows an example of data generated using the technique described above.

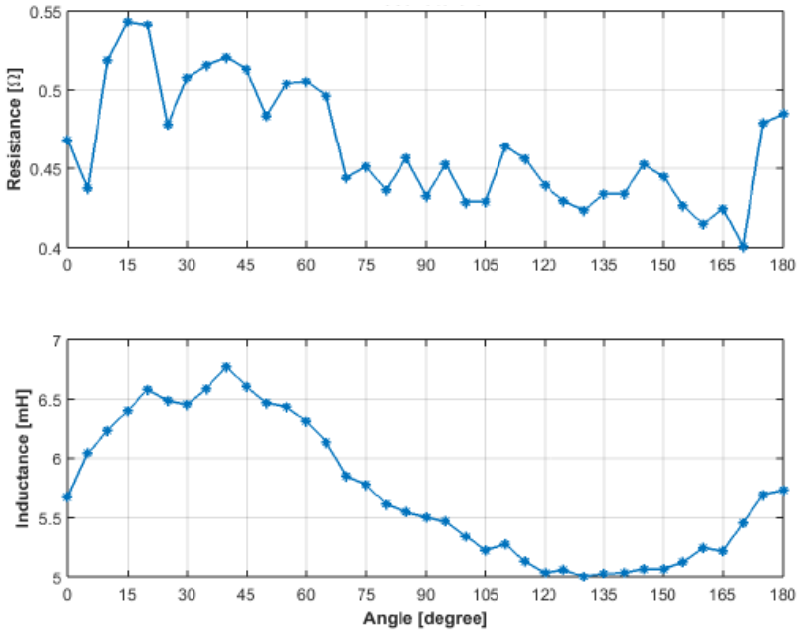


Figure 3: Resistance and inductance measurements produced by the MB

The 'magic box' target market is to test medium-voltage industrial AC induction motors. Previous research [5] shows that 5-20% of motor nameplate current must be used for effective broken rotor bar detection. To calculate the corner points for the voltage and current drive the following range of medium voltage class motors will be considered.

Nominal Line Voltage	VFD Cont. Amps	Nominal Motor kW	Nominal Motor hp
2400	46	150	200
2400	430	1500	2000
3300	46	187	250
3300	430	2050	2750
4160	46	261	350
4160	430	2600	3500
6600	40	400	500
6600	430	4000	5500

Figure 4: Example medium-voltage motor voltage, current and power levels derived from PowerFlex 7000 AC drive ratings

If lumped element model data were readily available from the manufacturers of medium-voltage motors, then it would fairly simple to determine the maximum voltage needed when testing at 10% of name plate current for standstill conditions. Since the lumped element data is not easily obtainable from most motor manufacturers, another method will be presented to approximate the voltage at a given current utilizing the nameplate current and NEMA locked rotor code presented on the nameplate of the motor. The name plate data for an example 5 HP motor to calculate LR voltage is presented below:

Dayton Industrial Motor				
Model: 3kW376	Type: ASGANE	ANB: 40C		
Output: 5HP 3.7kW	Frame: 184T	LR kVA code: J	Rating: CONT	
Poles: 4	Design B	INS: F		
Volts: 208-230/460	ENCODP			
Amps: 14.5-13.1/6.6	RPM: 1740			
SerNo: CT B3083010198				
BRG: 6206ZZC3-6205ZZC3				
Weight: 72/33 Lb/kg	NEMA NOM Eff: 87.5			
	NEMA Min Eff: 85.5			
50 Hz Data 5 HP	190/380 Volts	15.9/7.9 Amps	1420 RPM	1.0 SF

Figure 5: Nameplate data for a 5HP AC induction motor

This motor shows that when wired for 208VAC the full load run current is 14.5 amps; and that when wired for 460 VAC, the full-load run current is 6.6 amps. The nameplate NEMA LR kVa code for this motor is “J”. NEMA provides a table (shown below) which shows the ratio of locked rotor bar current to full load current as indicated by the code letter J.

Code Letter	kVA/hp	Code Letter	kVA/hp
A	0.0-3.15	K	8.0-9.0
B	3.15-3.55	L	9.0-10.0
C	3.55-4.0	M	10.0-11.2
D	4.0-4.5	N	11.2-12.5
E	4.5-5.0	P	12.5-14.0
F	5.0-5.6	R	14.0-16.0
G	5.6-6.3	S	16.0-18.0
H	6.3-7.1	T	18.0-20.0
J	7.1-8.0	U	20.0-22.4

Figure 6: NEMA locked rotor kVA/hp

If an assumption is made that under locked rotor conditions (standstill) the motor behaves as pure impedance since none of the energy is converted into mechanical energy, then the ratio of voltage to current must remain constant. Using the pu (per unit) method, then 1*puV would be 460V and 1*puI would be 6.6Amps for the motor above when wired as a 460V motor. Using code J, the locked rotor current is 7.1 puI at the nominal line voltage of 1*puV. To calculate the voltage for 10% of current (0.1puI), then following ratio equation can be set up.

$$7.1\text{puI} / 0.1\text{puI} = 1\text{puV} / x\text{puV} \quad (4)$$

Re-arranging the equation to solve for ‘x’ yields:

$$x\text{puV} = 0.1 / 7.1 * 1\text{puV} \quad (5)$$

So xpuV = 0.1 / 7.1 * 460V = 6.47 V to develop 0.1puI (10 percent name plate current). The calculated voltage and current drive requirements for medium-voltage motor testing (shown below) drive the requirements for the stimulus and measurement sections of the refactored tester.

Nominal Line Voltage	VFD Cont. Amps	Nominal Motor kW	Nominal Motor hp	Code "F" LRR kvA/hp	0.1pul	xpuV @ 0.1pul
2400	46	150	200	5	4.6	48
2400	430	1500	2000	5	43	48
3300	46	187	250	5	4.6	66
3300	430	2050	2750	5	43	66
4160	46	261	350	5	4.6	83.2
4160	430	2600	3500	5	43	83.2
6600	40	400	500	5	4	132
6600	430	4000	5500	5	43	132

Figure 7: Example drive voltages needed to test medium-voltage motors at ten percent nameplate current for NEMA code "F" in locked rotor condition

The required stimulus hardware for medium-voltage motors would be:

- AC-DC section
- Large 400V electrolytic caps for energy storage
- Six PWM outputs with deadband for high and low side drive of 3 ½ H switches
- IGBT driver with isolation, Vce sat protection, crowbar protection
- Intelligent power module (V>=400V, I>=50A)
- Programmable interrupt for 10-60Hz reversal
- A/D with LP filter for readback of BUS voltage for PWM output adjustment of BUS V

The required measurement hardware for medium-voltage motor measurements:

- Three 50A CTs
- Three voltage dividers with max V of 400V
- Six 12-24 bit simultaneous A/Ds
- Six PGAs if needed, depending upon A/D resolution
- Six LP (anti-aliasing) filters

The figure below shows a simplified schematic showing the PWM drive portion of the tester connected to a test motor. The measurement portion of the tester will be implemented by a separate piece of equipment discussed later in the chapter.

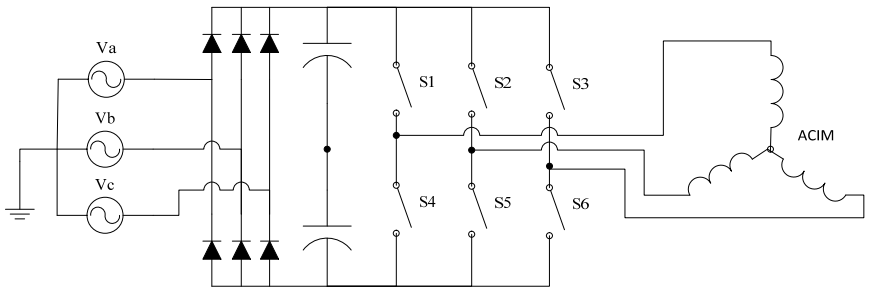


Figure 8: Schematic of prototype MB tester's PWM drive

For this project, the ST STEVAL-IHM025V1 1kW 3-phase motor control demonstration board was chosen for its availability and features. The power module can be fed by AC mains of 90-285 VAC which is rectified for a DC bus or can be fed by a separate DC supply voltage of 125-400 VDC. The intelligent power module (IPM) provides +15V and 3.3V auxiliary power from the DC link voltage derived with a buck converter which can be used to power a microprocessor. It uses the STGIPL14K60 IPM which provides a 3-phase inverter based on 600V IGBTs. The IPM provides high-side drivers, cross-conduction protection and op amps for current sensing in one rugged short-circuit IGPT package. The 'interlocking' feature prevents both the high and low switches from ever simultaneously turning on even if both are enabled by the PWM drive. During initial PWM drive signal development, this feature can prevent accidental destruction of the demonstration board by preventing cross-conduction when incorrect PWM signals are applied. When run on 110VAC, the rectified DC bus is approximately 165VDC and 330VDC when powered by 220VAC. It can test a large set of medium-voltage

motors with its 6.5A and 330VDC output maximum. Larger modules can be purchased from ST and numerous other manufacturers to easily scale up the voltage and current drive requirements.

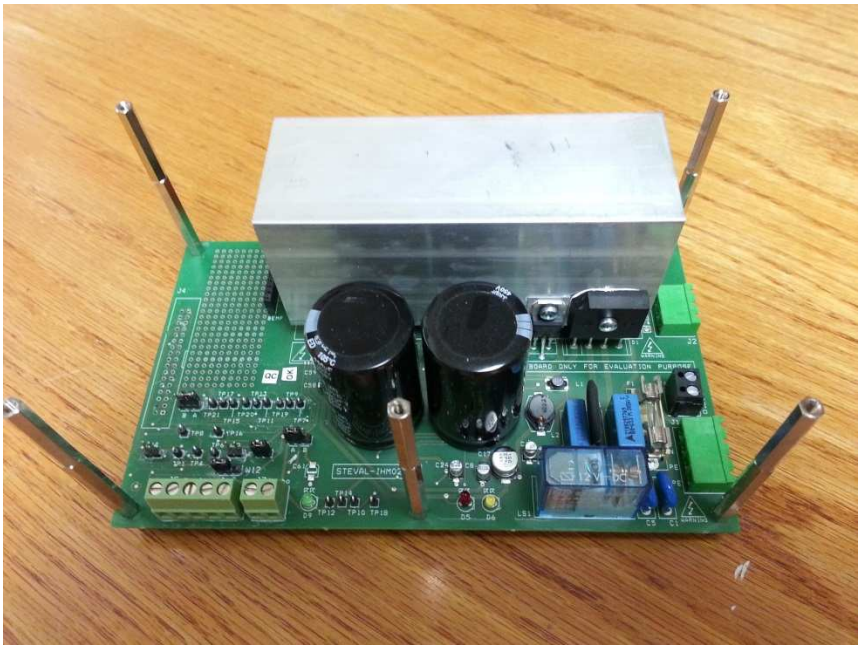


Figure 9: ST STEVAL-IHM025V1 1kW 3-phase motor control demonstration board

To drive the 3-phase intelligent power module, three pairs of PWM drive signals need to be generated. Each pair of PWM drive signals will drive one of the three ½ H switches in the IPM. To avoid cross-conduction of each ½ H switch, a programmable dead-band is programmed in the PWM drive generator or implemented in hardware by the IGBT drivers driving the ½ H switches. A sine-wave lookup table is used to determine the magnitude for phase U given the test angle. The magnitude of phase V is calculated from the sine of the test angle plus 120 degrees, while phase W is calculated from the sine of the test angle plus 240 degrees. A test frequency ranging from 1 to 60 Hz will drive the magnitude reversals of the U, V and W phases to generate the reversing stator magnetic field.

Initial PWM drive signal development began with the BeagleBone development kit which utilizes the 600MHz AM335x system on a chip (SOC) [14] [15]. The TI StarterWare bare

metal C library has basic driver support for all of the peripherals on the AM335x processor including the PWM, LCD, USB and A/D peripherals needed to build a very elegant embedded motor test system. PWM code was easily written to output three PWM signals with their complementary signals including deadband generation. Code was also written to interface the ADS1278 eight-channel 144kHz simultaneously sampling 24-bit ADC chip through the ASP port of the AM335x processor to allow sampling of three voltage and three current channels of a motor with high dynamic range. Using an A/D converter with high dynamic range can negate the need to use a programmable gain amplifier and cumbersome range changes. The AM335x processor was ideal in many aspects except for the following drawbacks: no motor control libraries are available, its BGA package is difficult to layout for PCBs and there is no high voltage isolation on the USB port. An attempt to develop the PWM drive using the DE0-nano utilizing the Altera Cyclone IV FPGA was successful except that the floating point multiplication scaling and sine functions consumed substantial amounts of the FPGA resources. Microprocessors are vastly superior in efficiently implementing floating point operations!

The final selection to provide the PWM stimulus was the inexpensive LAUNCHXL-F28069M development kit [16] [17] which offered the C2000 series F28069M microprocessor with floating point arithmetic, a motor control library, and most importantly included voltage isolation chips between the USB port and the microprocessor. In addition, the F28069M processor contains a ROM programmed to support TI's InstaSpin and InstaMotion proprietary libraries [18] [19]. The libraries allow motor parameter identification as well as supporting sensorless field-oriented control (FOC) of many motor types. Though the source code is not currently published, these ROM programmed libraries can be called to immediately spin motors in speed and torque control applications.

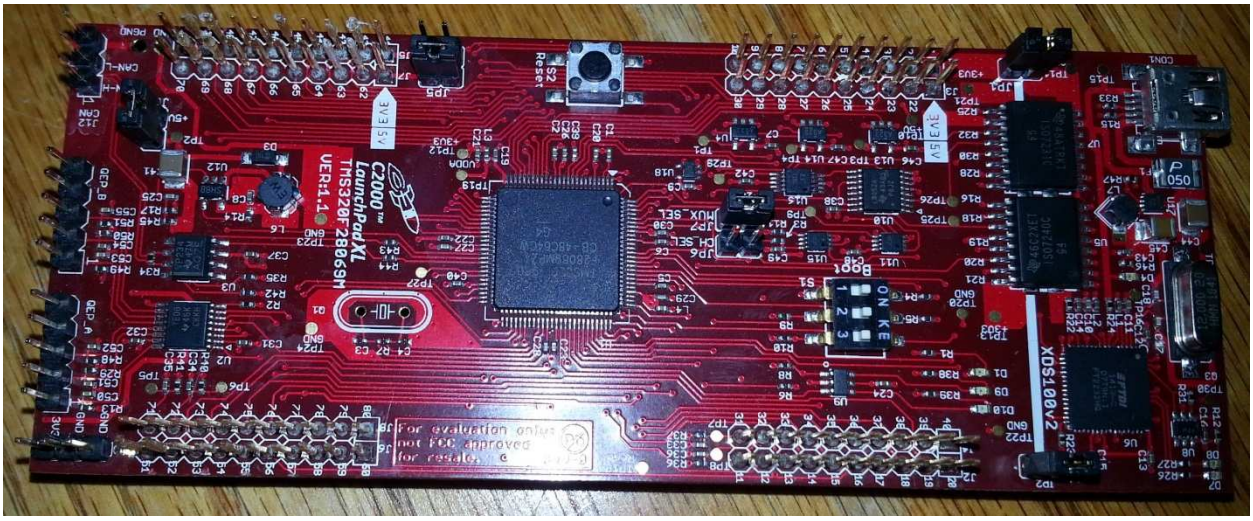


Figure 10: The LAUNCHXL-F28069M C2000 microprocessor development kit

The USB voltage isolation feature of the LAUNCHXL-F28069M is very important because the ST STEVAL-IHM025V1 demonstration kit directly rectifies the input AC and its ground reference is at half of the rectified bus voltage. When the STEVAL-IHM025V1 is powered by 110VAC and supplies +3.3V power to the F28069M microprocessor, the microprocessor is around +70 volts higher than the USB ground coming from the computer used to program the microprocessor. Jumpers JP1 and JP2 must be removed to enable the USB voltage isolation on the LAUNCHXL-F28069M development kit when powered from the ST STEVAL-IHM025V1 demonstration kit. According to published specifications, the ISO7240 and ISO7231 USB isolation chips have a rating of 2500Vrms for 1 minute and provide a working isolation voltage of 500V for up to 25 years [20]. Powering the STEVAL-IHM025V1 using a sufficiently sized isolation transformer is another solution that was not tried that could add a secondary voltage isolation barrier.

The DRV-8301 booster pack board is a compatible add-on to the LAUNCHXL-F28069M development kit and offers a safe low-voltage (+24V) 3-phase motor driver for ACIM or brushless DC (BLDC) motors up to 10A [21] [22]. The free MotorWare package from TI offers

numerous example motor control labs using the LAUNCHXL-F28069M and DRV-8301 booster pack combination.

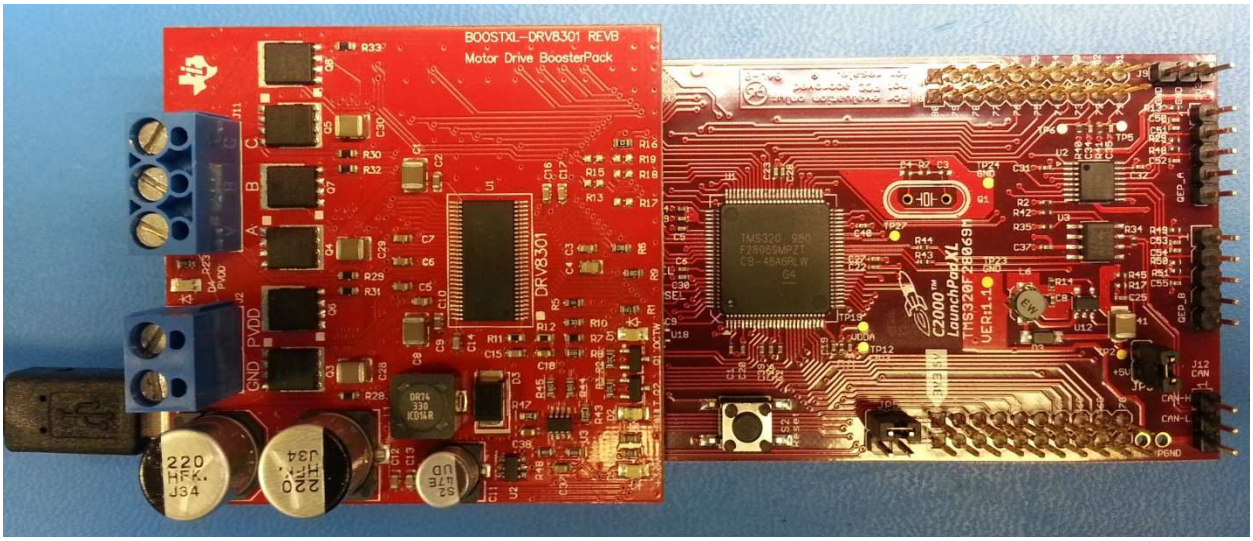


Figure 11: The LAUNCHXL-F28069M with DRV-8301 booster board attached

An adapter board was built to connect the STEVAL-IHM025V1 module to the LAUNCHXL-F28069M development board using the function pin mapping used by the DRV-8301 booster pack. With minor modifications to the InstaSpin code for voltage/current scaling and polarity and PWM polarity, the F28069M development board should be able to spin high-voltage motors using the STEVAL-IHM025V1 as it does with the low-voltage DRV-8301 motor drive board.

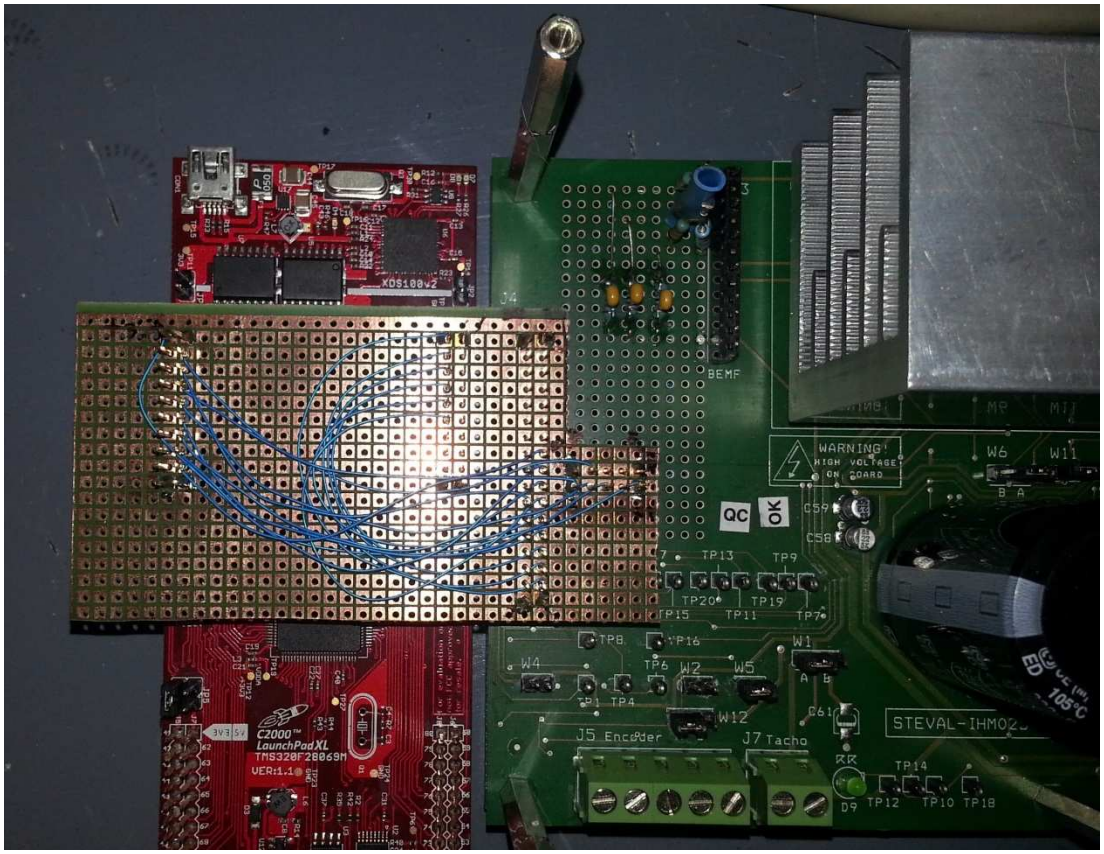


Figure 12: Adapter card built to interface LAUNCHXL-F28069M to STEVAL-IHM025V1

Numerous code examples are included for each of the TI C2000 processors supported in TI's control suite library [23]. For this project, code was pulled from the ControlSuite f2806x examples for PWM deadband, PWM tripzone, CPU timer, A/D and interrupt examples as detailed in the comments of the software listed in Appendix A. The F28069M code shown in Appendix A was built to load into the flash memory of the microprocessor but did not run until the code was modified to copy time-critical functions from the flash memory to RAM.

Though the F28069M processor cannot drive a LCD display, the TI GUI Composer can be used to build a GUI display running on a host computer to show the target processor values through the USB interface. In addition, the code composer studio supports an 'expression' window which can display the values of variables on the target system in real time.

The measurement of the motor signals will be performed by the SKF EXP4000 online motor tester. The EXP4000 tester uses a National Instruments NI-6212 data acquisition card for measuring the three voltages and the three currents supplied to the motor under test. The NI-6212 DAQ card can measure 16 channels each with 16 bits of resolution. Individually, each channel can be set to a voltage range of +/- [0.2, 0.5, 5.0, 10.0] volts. The acquisition rate is set to 25 kHz for each channel. The channels are sequentially scanned at a 400 kHz rate. Though the NI-6212 acquisition card has limitations in simultaneous sampling, cost and sample continuity during auto ranging to another voltage range, the easy USB interface and availability of software drivers makes it better than the lower resolution 12-bit A/Ds available on the LAUNCHXL-F28069M development board.

Figure 13: SKF EXP4000 used to make measurements of motor signals

Chapter 4 Performance of system

For the initial test of the system, a Baldor 2-pole 3-phase 480V 1A 0.5hp motor was connected. The microprocessor was set to run at 10 percent of its PWM drive with a reversal frequency of 15Hz. The SKF EXP4000 motor analyzer was attached using its 10A current clamps and voltage inputs for waveform recording purposes. The EXP4000 transient analyzer tool was set to record the three voltages and currents when the current went above .05A.

Figure 14: Trigger settings for EXP4000 transient analysis tool

The LAUNCHXL-F28069M/STEVAL-IHM025V1 combination was then plugged in to start the testing. As programmed, the test ran for approximately 90 seconds as indicated by the flashing blue and red LEDs on the LAUNCHXL-F28069M for each magnetic angle tested. The EXP4000 recorded the data (shown below) which shows each magnetic angle test with an off period followed by the next magnetic angle.

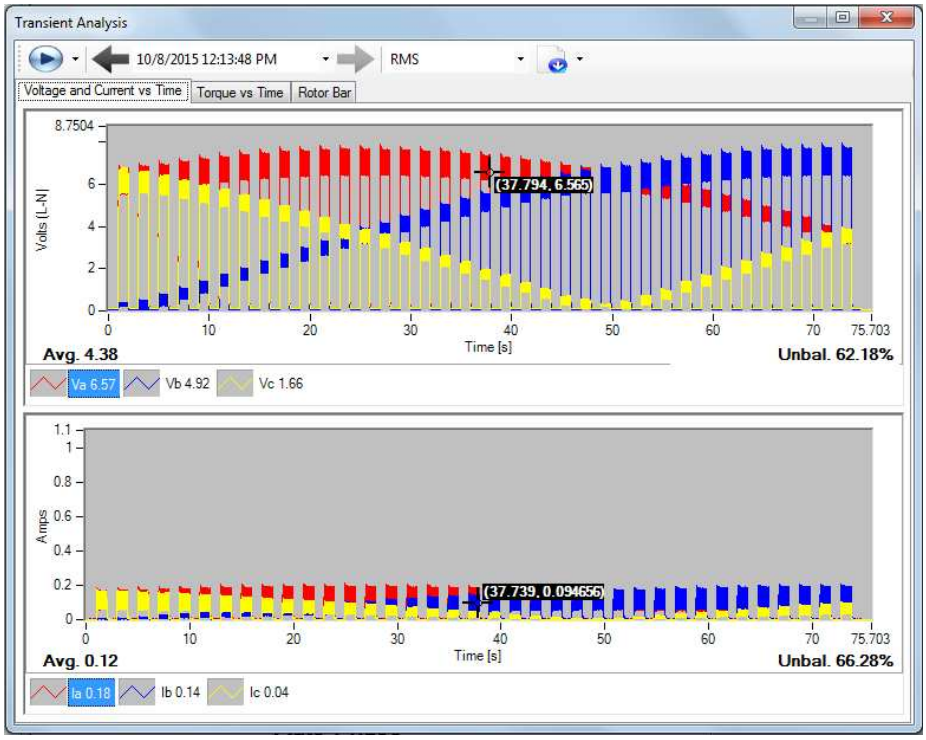


Figure 15: Recording by EXP4000 of 0.5hp 2-pole motor stepped through 36 magnetic angles 0 to 180 degrees

The recorded data was exported and then imported into the MATLAB script that previously written for the prototype MB built in 2009. The MATLAB script sorts each magnetic angle test section and then finds a pivot value to center the voltages and currents as shown below.

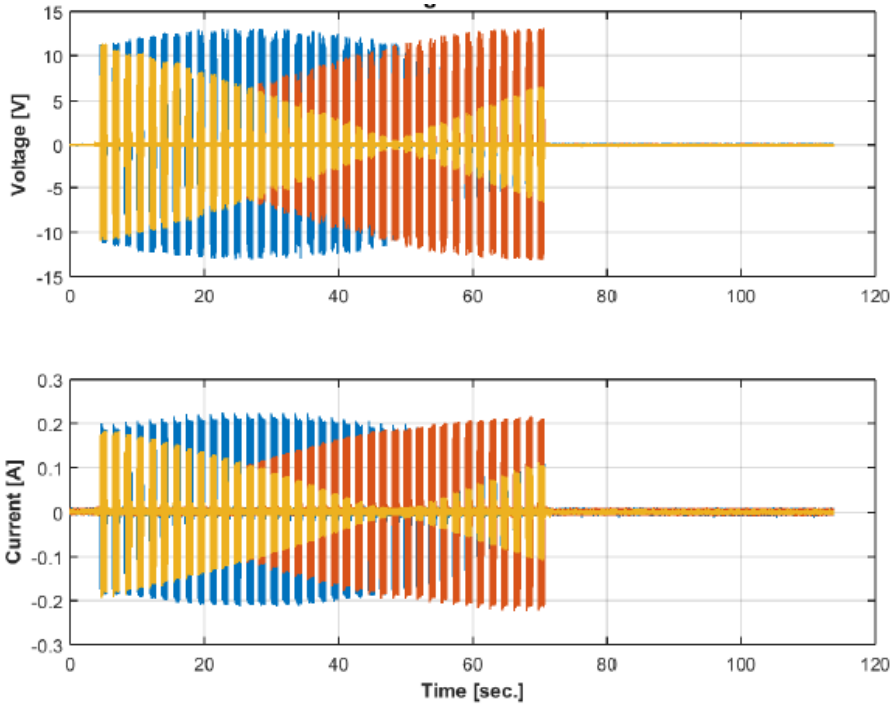


Figure 16: Voltages and currents after pivot by original MB MATLAB analysis script

The set of voltages and currents are then put through a customized Park transform which uses the magnetic angle that was used to generate each test section. Then the FFT spectra are computed for the d-q voltage and current, which are then divided to produce an impedance spectrum. The real part of the spectrum represents the resistance of the motor and the imaginary part represents the inductance of the motor. Plotting the peak value for each magnetic angle yields the impedance versus magnetic angle plots shown below.

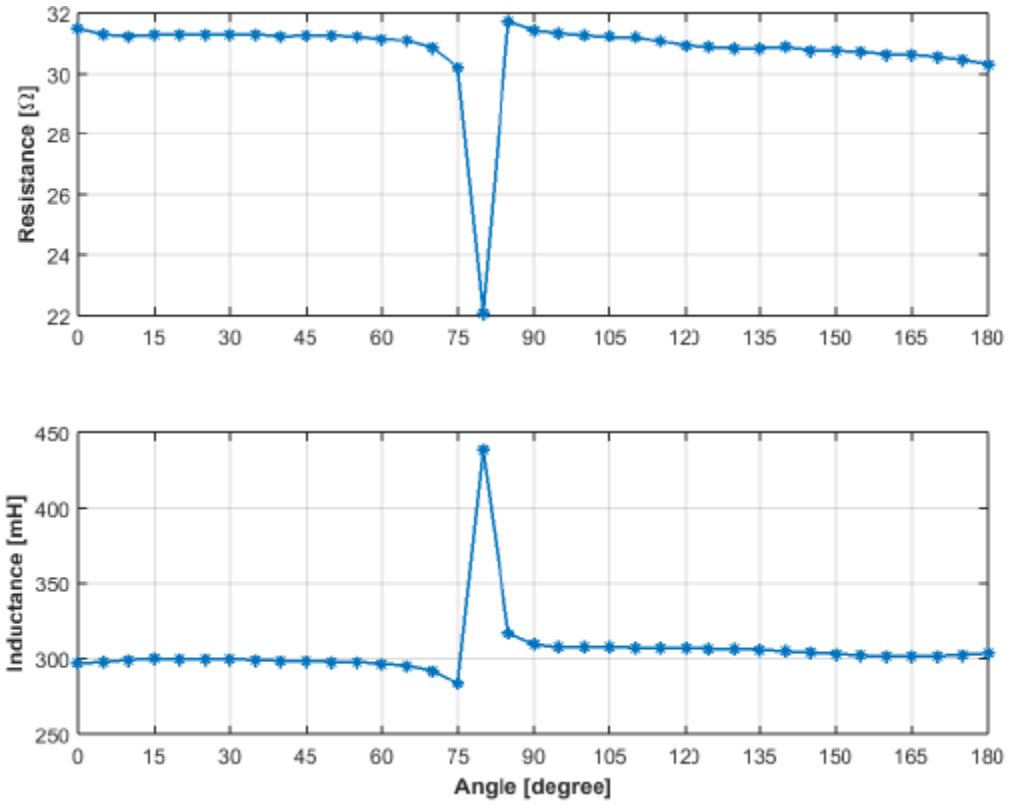


Figure 17: Data taken with refactored MB

Though the calculated resistance and inductance values looked reasonable for this particular motor, the data above shows a spike from magnetic angle 75-90 degrees. To determine if the discontinuity in resistance and inductance between angles 75 and 90 degrees was due to problems with the PWM generation or a bad motor, the motor shaft was rotated by 45 degrees and the test run again.

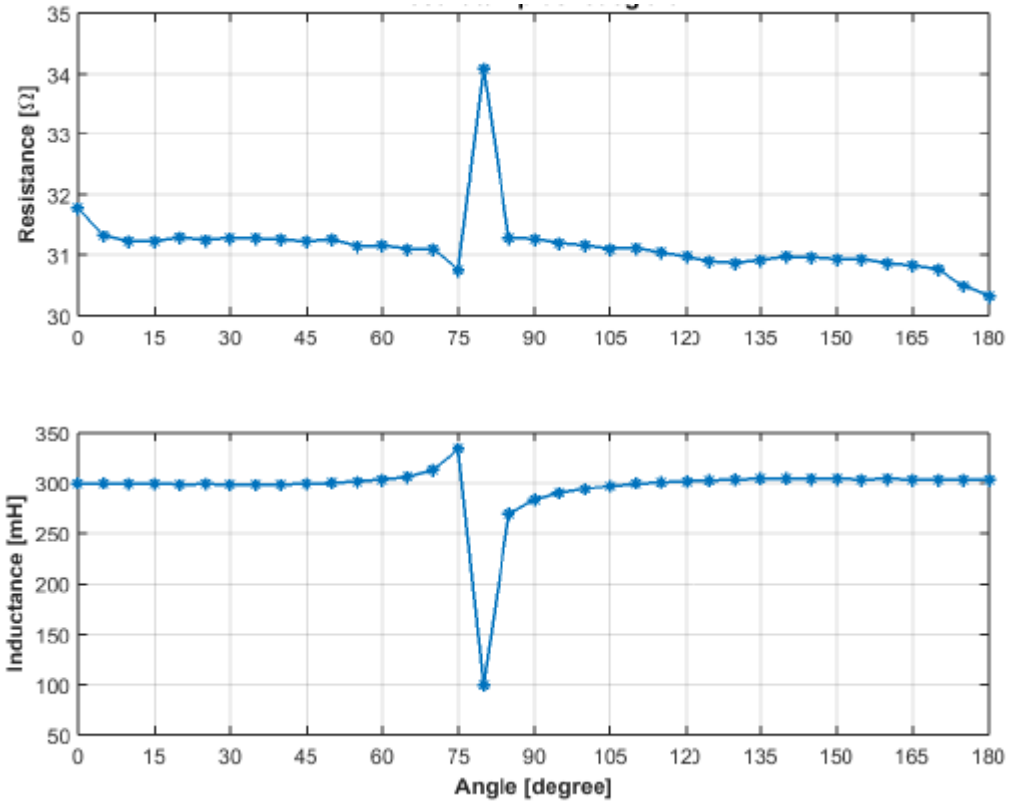


Figure 18: Data re-taken after rotating motor shaft by 45 degrees

The data discontinuity did not move by 45 degrees, so there may be a problem in the PWM generation at those particular angles or with the measured phases not matching up for the supplied DQ angle. Comparing the peaking of each phase for the data set taken with the prototype tester built in 2009 to the peaking of each phase for the new data set showing the discontinuity revealed that phases A and C were swapped on the new data set. After swapping the A and C voltage and current phase data and re-running the MATLAB script, the discontinuity went away indicating that the magnetic angle used for the DQ calculations did not line up to the angle used in the data set.

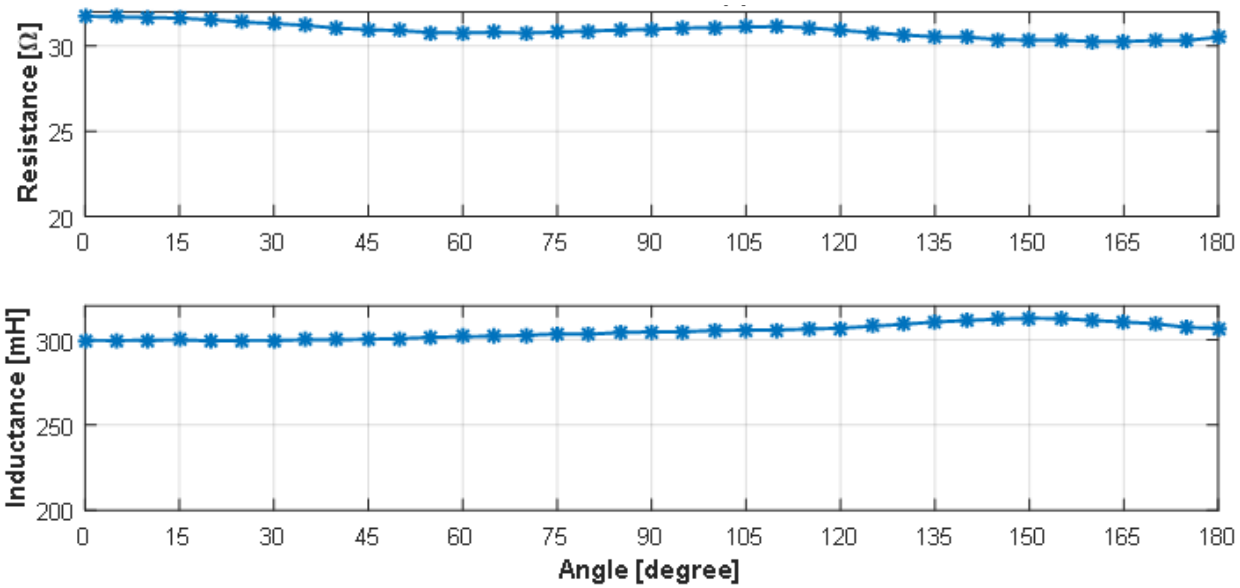


Figure 19: Re-calculated results with phases A and C swapped in data set showing proper resistance and inductance measurements on a good motor

Data was then taken on a motor of the same model but with one of the rotor bars drilled through to induce a single-bar rotor defect. The graph shows a slight dip in the resistance around 60 degrees indicating the presence of the induced single-bar rotor defect.

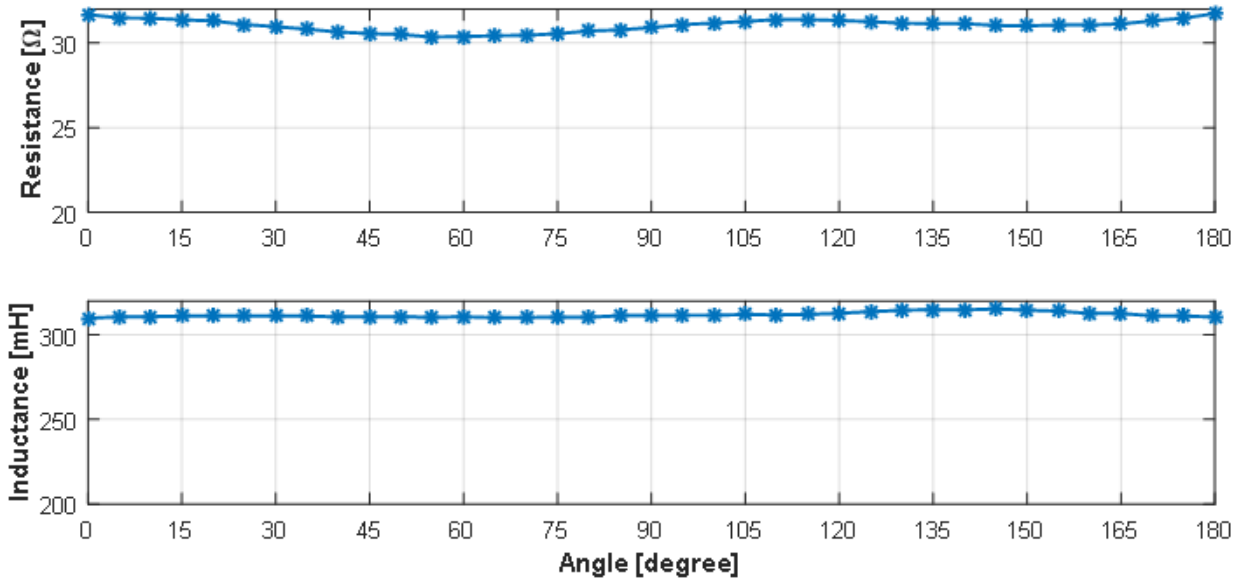


Figure 20: Results for a motor with a single-bar rotor defect

Chapter 5 Future work

The refactored tester has demonstrated that it can successfully measure the resistance and inductance versus magnetic angle of a motor just as the original prototype built in 2009. By detecting deviations in the resistance and inductance versus magnetic angle, the refactored tester should be able to detect the BRB defect in motors. Tests now need to be run on many motors with and without the BRB defect to ensure the refactored test system can find the BRB defect as well as the original prototype built in 2009. The refactored tester was wired to work with the TI InstaSpin libraries to add the new motor bump capability and still needs to be tested. Only minor modifications of TI's supplied DRV-8301 software module for voltage and current scaling along with PWM polarity for the STEVAL-IHM025V1 module should be needed to test if the refactored tester can bump and spin motors. In addition to testing for the BRB defect in motors, the refactored tester should also be able to detect missing magnetic wedges in a motor along with detecting faulty cores in motors [24] [25].

A VFD drive contains all of the hardware needed to implement the MB except possibly for the reversing-angle PWM drive and analysis algorithms. It would be beneficial for VFD drive manufacturers to incorporate the algorithms of MB into their drives to provide customers constant diagnostics of their motor's health. Whenever a VFD drive is requested to stop, the VFD drive could perform a broken rotor bar measurement. Previous stored measurements could be subtracted from the measurement to remove influences from the stator core itself and results for both the rotor bar health and core health could be determined by the VFD drive. If the motor is attached to a load which continues to rotate even when the motor is not energized, the VFD drive could determine the speed of the motor from the electrical signals, and then make rotor bar

measurements at given angles while adjusting for the additional angle movement due to the rotating load.

The MB computes the impedance of the attached load at a given frequency. So in addition to the detection of broken rotor bars, it can be used to measure the impedance of the attached load versus frequency. The fundamental PWM frequency can be shifted to measure the impedance for a range of frequencies much like a network analyzer. Once the characteristic impedance versus frequency is known for a given motor, the information can be used to synthesize a realistic lumped element model of the motor. The refined lumped element model could then be incorporated by a VFD drive to improve its motor control loop. The TI Insta-Spin motor control libraries are already using a similar technique to improve motor control.

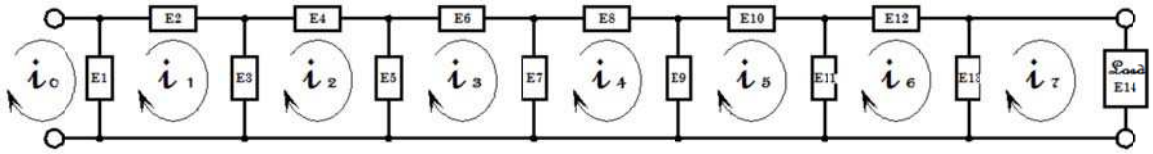
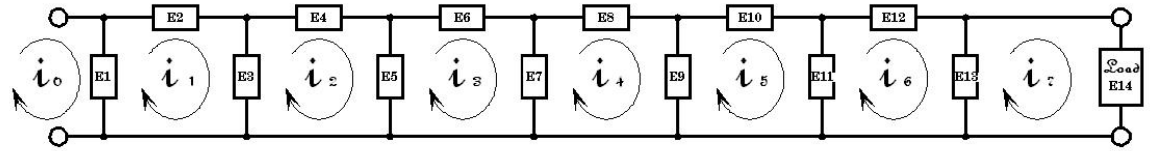


Figure 21: Generalized ladder impedance network for modeling AC induction motors



$$\begin{bmatrix} E_1 & -E_1 & 0\Omega & 0\Omega & 0\Omega & 0\Omega & 0\Omega & 0\Omega \\ -E_1 & (E_1+E_2+E_3) & -E_3 & 0\Omega & 0\Omega & 0\Omega & 0\Omega & 0\Omega \\ 0\Omega & -E_3 & (E_3+E_4+E_5) & -E_5 & 0\Omega & 0\Omega & 0\Omega & 0\Omega \\ 0\Omega & 0\Omega & -E_5 & (E_5+E_6+E_7) & -E_7 & 0\Omega & 0\Omega & 0\Omega \\ 0\Omega & 0\Omega & 0\Omega & -E_7 & (E_7+E_8+E_9) & -E_9 & 0\Omega & 0\Omega \\ 0\Omega & 0\Omega & 0\Omega & 0\Omega & -E_9 & (E_9+E_{10}+E_{11}) & -E_{11} & 0\Omega \\ 0\Omega & 0\Omega & 0\Omega & 0\Omega & 0\Omega & -E_{11} & (E_{11}+E_{12}+E_{13}) & -E_{13} \\ 0\Omega & 0\Omega & 0\Omega & 0\Omega & 0\Omega & 0\Omega & -E_{13} & E_{13}+E_{14} \end{bmatrix}$$

Figure 22: Matrix representation for ladder network for synthesis of ACIM lumped element model

To enable quick PWM drive development, the prototype MB was built using a expensive National Instruments FPGA module. To provide robust voltage and current drive, it used three 1200V 75A dual IGBT modules with three hybrid IGBT driver modules.

Manufacturer	Part Number	Description	$/ea	Quantity	Sum
Semikron	SKHI 22B H4 R	Hybrid Dual IGBT Driver	$ 89.22	3	$ 267.66
Semikron	SKM100GB125DN	1200V 75A Dual IGBT	$ 95.09	3	$ 285.27
National Instruments	NI sbRIO-9651 System on Module	FPGA PWM driver	$ 559.01	1	$ 559.01
			Total		**$ 826.67**

Figure 23: Cost for PWM drive, IGBTs and IGBT drivers in prototype MB

To reduce the cost of the refactored MB tester but keep the same performance, several key expensive components can be replaced with components more optimized for the final design requirements. The calculated maximum 165V voltage drive requirement (Figure 7) shows that the refactored MB could use 600V-rated IGBTs instead of the 1200V-rated IGBTs used in the prototype MB. A single intelligent power module such as the 600V 75A Mitsubishi PS21A7A provides all six IGBT drivers and IGBTs in a single package with substantial savings. The easy-to-program but expensive National Instruments FPGA module providing the PWM drive signals can now be replaced by the LAUNCHXL-F28069M.

Manufacturer	Part Number	Description	$/ea	Quantity	Sum
Mitsubishi	PS21A7A	600V 75A Intelligent Power Module	$ 107.82	1	$ 107.82
Texas Instruments	LAUNCHXL-F28069M	C2000 PWM driver	$ 29.00	1	$ 29.00
			Total		**$ 136.82**

Figure 24: Cost for PWM drive, IGBTs and IGBT drivers in refactored MB

Making these component changes will reduce the bill of material cost of the refactored MB tester by $689.85 over the bill of material cost for the prototype MB tester. To cover the overhead costs of marketing, engineering and sales, a low-volume product must typically be sold at about 5 times the cost of the bill of material. By reducing the MB bill of materials by $689.85,

the sales price of the MB tester can be $3449.25 lower allowing the product to be more competitive in the condition-monitoring tester market.

Bibliography

[1] S. Bindu and V. V. Thomas, "Diagnosis of internal faults of three phase squirrel cage induction motor - A review," in *International Conference on Advances in Energy Conversion Technologies*, 2014, pp. 48-54.

[2] P. F. Albrecht et al., "Assessment of the reliability of motors in utility applications-Updated," *IEEE Trans. on Energy Convers.*, vol. 1, pp. 39-46, 1986.

[3] P. Donnell, "Report of large motor reliability survey of industrial and commercial installation, Part I and Part II," *IEEE Trans. Ind. Appl.*, vol. 21, no. 4, pp. 853-872, 1985.

[4] G. B Kilman et al., "Noninvasive detection of broken rotor bars in operating induction motors," *IEEE Trans. Energy Convers.*, vol. 3, no. 4, pp. 873 - 879, Oct. 1988.

[5] B. Kim et al., "Automated Detection of Rotor Faults for Inverter-fed Induction Machines under Standstill Conditions," in *IEEE Energy Conversion Congress*, 2009, pp. 2277 - 2284.

[6] S. Lee et al., "Evaluation of the Influence of Rotor Axial Air Ducts on Condition Monitoring of Induction Motors," in *IEEE Energy Conversion Congress*, 2012, pp. 3016 - 3023.

[7] A. Garcia-Perez et al., "Startup Current Analysis of Incipient Broken Rotor Bar in Induction Motors using High-Resolution Spectral Analysis," in *IEEE International Symposium on Diagnostics for Electric Machines, Powers Electronics and Drives*, 2011, pp. 657 - 663.

[8] Y. Wei et al., "Broken Rotor Bar Detection in Induction Motors via Wavelet Ridge," in *International Conference on Measuring Technology and Mechatronics Automation*, 2009, pp. 625 - 628.

[9] K. M. Siddiqui and V. K Giri, "Broken rotor bar fault detection in induction motors using Wavelet Transform," in *International Conference on Computing , E-Learning and Emerging Technology*, 2012, pp. 1-6.

[10] C. Yang et al., "Reliable Detection of Induction Motor Rotor Faults Under the Rotor Axial Air Duct Influence," *IEEE Trans. Ind. Appl.*, vol. 50, no. 4, pp. 2493 - 2502, 2014.

[11] D. Hyun et al., "Detection of airgap eccentricity for induction motors using the single-phase rotation test," *IEEE Trans. Energy Convers.*, pp. 689 - 696, 2012.

[12] I. Kathir et al., "Detection of rotor fault in an induction motor under standstill condition," in *International Conference on Advances in Engineering, Science and Management*, 2012, pp. 547 - 552.

[13] M. Riaz. (2015, October) SIMULATION OF ELECTRIC MACHINE AND DRIVE SYSTEMS USING MATLAB AND SIMULINK [Online]. http://www.ece.umn.edu/users/riaz/animations/abcvec.html

[14] Texas Instruments. (2011, October) AM335x ARM® Cortex™-A8 Microprocessors (MPUs) [Online]. http://www.ti.com/lit/ds/sprs717h/sprs717h.pdf

[15] Gerald Coley. (2012, May) BeagleBone System Reference [Online]. http://beagleboard.org/static/beaglebone/latest/Docs/Hardware/BONE_SRM.pdf

[16] Texas Instruments. (2014) Meet the TMS320F28069M LaunchPad Development Kit [Online]. http://www.ti.com/lit/ml/sprui02/sprui02.pdf

[17] Texas Instruments. (2015, January) LAUNCHXL-F28069M Overview User's Guide [Online]. http://www.ti.com/lit/ug/sprui11/sprui11.pdf

[18] Texas Instruments. (2013, January) InstaSPIN-FOC™ and InstaSPIN-MOTION™ User's

Guide [Online]. http://www.ti.com/lit/ug/spruhj1f/spruhj1f.pdf

[19] Texas Instruments. (2013, April) TMS320F28069M, TMS320F28068M InstaSPIN™-MOTION Software Technical Reference Manual [Online]. http://www.ti.com/lit/ug/spruhj0b/spruhj0b.pdf

[20] Texas Instruments. (2007, September) ISO724x High-Speed, Quad-Channel Digital Isolators [Online]. http://www.ti.com/lit/ds/symlink/iso7240m.pdf

[21] Texas Instruments. (2013, October) BOOSTXL-DRV8301 Hardware User's Guide [Online]. http://www.ti.com/lit/ug/slvu974/slvu974.pdf

[22] Texas Instruments. (2013) Motor Drive BoosterPack Quick Start Guide: BOOSTXL-DRV8301 [Online]. http://www.ti.com/lit/sg/sldc006/sldc006.pdf

[23] Texas Instruments. (2010, January) ControlSUITE™ Getting Started Guide [Online]. http://www.ti.com/lit/ml/sprugu2c/sprugu2c.pdf

[24] K.W. Lee et al., "Detection of stator slot magnetic wedge failure for induction motors without disassembly," in *IEEE International Symposium on Diagnostics for Electric Machines, Powers Electronics and Drives*, 2013, pp. 183-191.

[25] K. Lee et al., "A Stator-Core Quality-Assessment Technique for Inverter-Fed Induction Machines," *IEEE Trans. Ind. Applicat.*, vol. 46, no. 1, pp. 213 - 221, Jan./Feb. 2010.

[26] M. F. Cabanas et al., "A New Portable, Self-Powered, and Wireless Instrument for the Early Detection of Broken Rotor Bars in Induction Motors," *IEEE Trans. Ind. Electron.*, vol. 58, no. 10, Oct. 2011.

[27] C. Demian et al., "Detection of Induction Machines Rotor Faults at Standstill Using Signals Injection," *IEEE Trans. Ind. Applicat.*, vol. 40, no. 6, Nov./Dec. 2004.

[28] B. Mirafzal and N. A. O. Demerdash, "Effects of Load Magnitude on Diagnosing Broken Bar Faults in Induction Motors Using the Pendulous Oscillation of the Rotor Magnetic Field Orientation," *IEEE Trans. Ind. Appl.*, vol. 41, pp. 771-783, 2005.

[29] *IEEE Recommended Practice: Definition of Basic Per-Unit Quantities for AC Rotating Machines*, IEEE Standard 86, 1987.

[30] A. M. da Silva, "Induction motor fault diagnostic and monitoring methods," M.S. thesis, Dept. Elect. Comp. Eng., Marquette Unv., Milwaukee, WI, 2006.

[31] M. E. H. Benbouzid, "A review of induction motors signature analysis as a medium for faults detection," *IEEE Trans. Ind. Electron.*, vol. 47, no. 5, pp. 984-993, 2000.

[32] Texas Instruments. (2011, January) TMS320x2806x Piccolo Technical Reference Manual [Online]. http://www.ti.com/lit/ug/spruh18e/spruh18e.pdf

[33] Texas Instruments. (2005, November) Programming TMS320x28xx and 28xxx Peripherals in C/C++ [Online]. http://www.ti.com/lit/an/spraa85d/spraa85d.pdf

[34] K. Sudha, "Implementation of FPGA based controller for induction motor drives," in *International Conference on Computation of Power, Energy, Information and Communication*, 2013, pp. 37 - 42.

Appendix A Source code for generating PWM signals

```
//###########################################################################
// Description:
//! \addtogroup f2806x_example_list
//! <h1>ePWM Deadband Generation (epwm_deadband)</h1>
//!
//! This example configures ePWM1, ePWM2 and ePWM3 for:
//!   - Count up/down
//!   - Deadband
//! 3 Examples are included:
//!   - ePWM1: Active low PWMs
//!   - ePWM2: Active low complementary PWMs
//!   - ePWM3: Active high complementary PWMs
//!
//! Each ePWM is configured to interrupt on the 3rd zero event
//! when this happens the deadband is modified such that
//! 0 <= DB <= DB_MAX.  That is, the deadband will move up and
//! down between 0 and the maximum value.
//!
//! \b External \b Connections \n
//!  - EPWM1A is on GPIO0
//!  - EPWM1B is on GPIO1
//!  - EPWM2A is on GPIO2
//!  - EPWM2B is on GPIO3
//!  - EPWM3A is on GPIO4
//!  - EPWM3B is on GPIO5
//

//###########################################################################
// $TI Release: F2806x C/C++ Header Files and Peripheral Examples V150 $
// $Release Date: June 16, 2015 $
// $Copyright: Copyright (C) 2011-2015 Texas Instruments Incorporated -
//             http://www.ti.com/ ALL RIGHTS RESERVED $
//###########################################################################

//###########################################################################
// Code adapted by Craig S. Powers for providing PWM stimulus to motors to measure broken
rotor bars
//
// Snippets of code were spliced together from the many examples provided by Texas
Instruments
// in their Control Suite library for the F28069 processor
// .\controlSUITE\device_support\f2806x\v150\F2806x_examples_ccsv5
```

```
//
// Code adapted to drive the ST STEVAL-IHM025V1 IPM module

// Added code from C:\ti\controlSUITE\development_kits\LAUNCHXL-
F28069M\LaunchPadDemo
//    to flash Blue and Red LEDs on launchpad
//
// Change all PWMs to Active high mode for STGIPL14K60 IPM drive requirements
// --> Changed EPwm1Regs.DBCTL.bit.POLSEL = /*DB_ACTV_LO*/ DB_ACTV_HI;
//
// Pulled PWM Freq. calc. code from Example_EPwmSetup.c located in project
Example_2806xEqep_freqcal
//
// Pulled code from project Example_2806xAdcSoc to implement ADCs on PWM1 trigger
//
// Pulled code from project Example_2806xLaunchPad to implement CPU timer for magnetic
angle reversals
//
// Removed divide by 16 for PWM clocks, so PWMs go faster and frequency matches
// calculations pulled from Example_EPwmSetup.c
//   EPwm1Regs.TBCTL.bit.HSPCLKDIV  = TB_DIV1 /*TB_DIV4*/;   // Clock ratio to
SYSCLKOUT
//   EPwm1Regs.TBCTL.bit.CLKDIV          = TB_DIV1 /*TB_DIV4*/;
//
// Added code to calculate magnitudes for each three phase output for a given angle
//
// sin(x) was returning incorrect values
//         Deleted linker libraries
//         "IQmath_fpu32.lib"
//                  "rts2800_fpu32_fast_supplement.lib"
//                  "libc.a"
//
//  Added CPU timer2 from Example_2806xCPUTimer.c for test angle on/off state timer
//
//  Added Trip Zone 5 & 6 PWM single-shot shutdown based on Example_2806xEPwmTripZone
//    and notes from TI's document Spruh18e pg. 244, pg. 293 so PWMs go to safe state
//    on debugger 'pause'. Safe PWM state for debugger 'Halt' is not yet working
//          So it's best to 'pause', then 'Halt' debugger! to stop PWMs
//    Many sad cases on TI's forums of guys blowing up their HV IPMs
//
// Adapting to run from FLASH by adding function to copy key timing sections to RAM
//
//###########################################################################

#include <stdint.h> //Include new clarified types such as int16_t
#include <math.h>
```

```
#include "DSP28x_Project.h"     // Device Headerfile and Examples Include File

//Prototypes for initializing PWMs
void InitEPwm1Example(void);
void InitEPwm2Example(void);
void InitEPwm3Example(void);

//Prototypes for Interrupt Service Routines to take care of PWMs
__interrupt void epwm1_isr(void);
__interrupt void epwm2_isr(void);
__interrupt void epwm3_isr(void);

// Interrupt Service Routines to take care of PWMs during a Trip Zone event
__interrupt void epwm1_tzint_isr(void);
__interrupt void epwm2_tzint_isr(void);
__interrupt void epwm3_tzint_isr(void);

//Prototypes for the 'magnetic polarity reversing' CPU_Timer0
__interrupt void cpu_timer0_isr(void);

//Prototypes for the 'on/off' CPU_Timer1 for each magnetic angle step
__interrupt void cpu_timer1_isr(void);

//Proto types for ADC
__interrupt void adc_isr(void);
void Adc_Config(void);

void MemCopy(Uint16 *SourceAddr, Uint16* SourceEndAddr, Uint16* DestAddr);

// Global variables used in this example
Uint32  EPwm1TimerIntCount;
Uint32  EPwm2TimerIntCount;
Uint32  EPwm3TimerIntCount;
Uint16  EPwm1_DB_Direction;
Uint16  EPwm2_DB_Direction;
Uint16  EPwm3_DB_Direction;

//May ditch variables below which are only useful if cycle-cycle trip zone counting versus
current on-shot use
Uint32  EPwm1TZIntCount;
Uint32  EPwm2TZIntCount;
Uint32  EPwm3TZIntCount;

// Global A/D variables used in this example:
Uint16 LoopCount;
Uint16 ConversionCount;
```

```
//ADC channels to sample
Uint16 Vbus[10];
Uint16 Va[10];
Uint16 Vb[10];
Uint16 Vc[10];
Uint16 Ia[10];
Uint16 Ib[10];
Uint16 Ic[10];

// Global Variables for GUI Composer/SCIA(C# interface) to access
int polarity = 1; //The magnetic angle polarity
int onOffCount = 0;
float ReversalTime_Hz = 15.0 /*0.333*/; //The magnetic reversal test frequency
double TestAngle_deg = 0.0; // Magnetic test angle to build from the three phase angle vector
summation

// 2 pole motors test from 0 to 180 degrees, magnetic reversals covers other half
// 4 pole motors test from 0 to 90 degrees
// N pole motors test from 0 to 180/(pole pairs)
double motorPoles = 4;
double numberOfTestAngles = 36.0;
double maxTestAngle;
double angleStepSize;

double sinPHa = 0.0;
double sinPHb = 0.0;
double sinPHc = 0.0;

int TestStart = 0;
int angleOn = 0; //If zero, then all motor terminal PWM voltages equal, otherwise magnetic
angle voltages

double sinZero;
double sinPIover2;
double sinPIover4;

float Current_Pct; //PWM output voltage/current from 0 to 100%
float I_pct;

//uint16_t on_off_Count = 0;

//PWM (Bi-polar) magnitude adjustments to CMPA from midpoint
        int16 PHa_mag;
        int16 PHb_mag;
        int16 PHc_mag;
```

```
//PWM CMPA values
        Uint16 PWM1_CMPA = 0;
        Uint16 PWM2_CMPA = 0;
        Uint16 PWM3_CMPA = 0;

// Maximum Dead Band values
#define EPWM1_MAX_DB   0x03FF
#define EPWM2_MAX_DB   0x03FF
#define EPWM3_MAX_DB   0x03FF

#define EPWM1_MIN_DB   0
#define EPWM2_MIN_DB   0
#define EPWM3_MIN_DB   0

// To keep track of which way the Dead Band is moving
#define DB_UP   1
#define DB_DOWN 0

//To Calculate the PWM Frequwncy
#define CPU_CLK   90e6
#define PWM_CLK   5e3
#define SP       CPU_CLK/(2*PWM_CLK) //TBPRD is set to SP and CMPA is setto SP/2

#define PI 3.1415926
#define HALF_PI 1.57079632
#define DEG_2_RAD 0.017453292519943295769236907684889

#define FLASH //Comment out if running in RAM

void main(void)
{
        //Variables purposely not visible to GUI Composer
        float Timer_uS;

        // Only used if running from FLASH
        // Note that the variable FLASH is defined by the compiler (-d FLASH)
        #ifdef FLASH
        // Copy time critical code and Flash setup code to RAM
        // The  RamfuncsLoadStart, RamfuncsLoadEnd, and RamfuncsRunStart
        // symbols are created by the linker. Refer to the linker files.
                MemCopy(&RamfuncsLoadStart, &RamfuncsLoadEnd, &RamfuncsRunStart);

        // Call Flash Initialization to setup flash waitstates
        // This function must reside in RAM
                InitFlash();    // Call the flash wrapper init function
```

```
        #endif //(FLASH)

        //Global variables visible to GUI Composer
        Current_Pct = 10.0; //PWM output percentage for a motor test
    maxTestAngle = 360/motorPoles;
        angleStepSize = maxTestAngle/numberOfTestAngles;

// Step 1. Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
// This example function is found in the F2806x_SysCtrl.c file.
   InitSysCtrl();

// Step 2. Initalize GPIO:
// This example function is found in the F2806x_Gpio.c file and
// illustrates how to set the GPIO to it's default state.
// InitGpio();  // Skipped for this example

// For this case just init GPIO pins for ePWM1, ePWM2, ePWM3
// These functions are in the F2806x_EPwm.c file
   InitEPwm1Gpio();
   InitEPwm2Gpio();
   InitEPwm3Gpio();
   //InitTzGpio(); //Call if TZ1 or TZ2 pins are going to be used

// Step 3. Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
   DINT;

// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
// This function is found in the F2806x_PieCtrl.c file.
   InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
   IER = 0x0000;
   IFR = 0x0000;

// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
// This will populate the entire table, even if the interrupt
// is not used in this example.  This is useful for debug purposes.
// The shell ISR routines are found in F2806x_DefaultIsr.c.
// This function is found in F2806x_PieVect.c.
   InitPieVectTable();
```

```
// Interrupts that are used in this example are re-mapped to
// ISR functions found within this file.
   EALLOW;  // This is needed to write to EALLOW protected registers
   PieVectTable.EPWM1_INT = &epwm1_isr;
   PieVectTable.EPWM2_INT = &epwm2_isr;
   PieVectTable.EPWM3_INT = &epwm3_isr;
   PieVectTable.EPWM1_TZINT = &epwm1_tzint_isr; //Setup PWM1's Trip zone ISR
   PieVectTable.EPWM2_TZINT = &epwm2_tzint_isr; //Setup PWM2's Trip zone ISR
   PieVectTable.EPWM3_TZINT = &epwm3_tzint_isr; //Setup PWM3's Trip zone ISR
   PieVectTable.TINT0       = &cpu_timer0_isr; //Initialize the CPU timer0 interrupt
   PieVectTable.TINT1       = &cpu_timer1_isr; //Initialize the CPU timer1 interrupt
   PieVectTable.ADCINT1 = &adc_isr; //Initialize the ADC interrupt
   EDIS;    // This is needed to disable write to EALLOW protected registers


// Step 4. Initialize all the Device Peripherals:
// This function is found in F2806x_InitPeripherals.c
// InitPeripherals();  // Not required for this example

   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 0;
   EDIS;

   //Initialize the PWM modules
   InitEPwm1Example();
   InitEPwm2Example();
   InitEPwm3Example();

   EALLOW;
   SysCtrlRegs.PCLKCR0.bit.TBCLKSYNC = 1;
   EDIS;

   //Initialize GPIOs for the LEDs and turn them off
   EALLOW;
   GpioCtrlRegs.GPBDIR.bit.GPIO34 = 1;
   GpioCtrlRegs.GPBDIR.bit.GPIO39 = 1;
   GpioDataRegs.GPBDAT.bit.GPIO34 = 1;
   GpioDataRegs.GPBDAT.bit.GPIO39 = 1;
   EDIS;

   InitCpuTimers();   // For this example, only initialize the Cpu Timers

   //Set up the CPU timer 0 for the magnetizing vector reversals
   Timer_uS = 1.0/(2.0*ReversalTime_Hz) * 1E6;
   ConfigCpuTimer(&CpuTimer0, 90, Timer_uS /*1000*/);
   CpuTimer0Regs.TCR.all = 0x4010; // Use write-only instruction to set TSS bit = 1
```

```
   //Set up the CPU timer 1 for on/off time for each test angle
   ConfigCpuTimer(&CpuTimer1, 90 /*uP Clk*/, 1000000 /*uS*/); //Set for 1 second period
   CpuTimer1Regs.TCR.all = 0x4010; // Use write-only instruction to set TSS bit = 1

   //Setup the ADCs
   InitAdc();  // For this example, init the ADC
   AdcOffsetSelfCal();

// Step 5. User specific code, enable interrupts
// Initialize counters:
   EPwm1TimerIntCount = 0;
   EPwm2TimerIntCount = 0;
   EPwm3TimerIntCount = 0;

   EPwm1TZIntCount = 0; //Counters for each trip zone event (only useful in cycle-by-cycle trip
sources)
   EPwm2TZIntCount = 0;
   EPwm3TZIntCount = 0;

// Enable CPU INT3 which is connected to EPWM1-3 INT:
   IER |= M_INT3;
// Enable CPU INT2 which is connected to EPWMX_TZINT: where X = 1-N
   IER |= M_INT2; //Group 2 interrupts for TZ?

// Enable CPU int1 which is connected to CPU-Timer 0, CPU int13
// which is connected to CPU-Timer 1, and CPU int 14, which is connected
// to CPU-Timer 2:
   IER |= M_INT1;
   IER |= M_INT13;
   //IER |= M_INT14; //Not yet using CPU-Timer3

// Enable EPWM INTn in the PIE: Group 3 interrupt 1-3
   PieCtrlRegs.PIEIER3.bit.INTx1 = 1;
   PieCtrlRegs.PIEIER3.bit.INTx2 = 1;
   PieCtrlRegs.PIEIER3.bit.INTx3 = 1;

// Enable EPWM TZ INTn in the PIE: Group 2 interrupt 1-3 (EPWM1_TZINT)
   //PieCtrlRegs.PIEIER2.bit.INTx1 = 1;
   //PieCtrlRegs.PIEIER2.bit.INTx2 = 1;
   //PieCtrlRegs.PIEIER2.bit.INTx3 = 1;

   PieCtrlRegs.PIEIER1.bit.INTx7 = 1; //Enable CPU timer interrupt

// ?? Can both PIEIER1 and PIEIER3 for PWM be set to INTx1 ???
//Apparently Yes
```

```
   // Enable ADCINT1 in PIE
   PieCtrlRegs.PIEIER1.bit.INTx1 = 1;         // Enable INT 1.1 in the PIE

// Enable global Interrupts and higher priority real-time debug events:
   EINT;   // Enable Global interrupt INTM
   ERTM;   // Enable Global realtime interrupt DBGM

   //ADC count variables
   LoopCount = 0;
   ConversionCount = 0;

   //Setup initial LED state for toggling (Red LED on, blue off)
   GpioDataRegs.GPBDAT.bit.GPIO34 = 0;
   GpioDataRegs.GPBDAT.bit.GPIO39 = 1;

   // Configure ADCs
         EALLOW;
      AdcRegs.ADCCTL2.bit.ADCNONOVERLAP = 1;      // Enable non-overlap mode
      AdcRegs.ADCCTL1.bit.INTPULSEPOS          = 1;    // ADCINT1 trips after AdcResults
latch
      AdcRegs.INTSEL1N2.bit.INT1E      = 1;         // Enabled ADCINT1
      AdcRegs.INTSEL1N2.bit.INT1CONT    = 0;    // Disable ADCINT1 Continuous mode
      AdcRegs.INTSEL1N2.bit.INT1SEL      = 1;   // setup EOC1 to trigger ADCINT1 to fire


      AdcRegs.ADCSOC0CTL.bit.CHSEL    = 7;    // set SOC0 channel select to ADCINA7 -
Vbus

      AdcRegs.ADCSOC1CTL.bit.CHSEL    = 9;    // set SOC1 channel select to ADCINB1 - Va
      AdcRegs.ADCSOC2CTL.bit.CHSEL    = 2;    // set SOC2 channel select to ADCINA2 - Vb
      AdcRegs.ADCSOC3CTL.bit.CHSEL    = 10;   // set SOC3 channel select to ADCINB2 - Vc

      AdcRegs.ADCSOC4CTL.bit.CHSEL    = 0;    // set SOC4 channel select to ADCINA0 - Ia
      AdcRegs.ADCSOC5CTL.bit.CHSEL    = 8;    // set SOC5 channel select to ADCINB0 - Ib
      AdcRegs.ADCSOC6CTL.bit.CHSEL    = 1;    // set SOC6 channel select to ADCINA1 - Ic
      //May in future add SOC7 to measure heat sink temp readback of STEVAL-IHM025V1
module

      AdcRegs.ADCSOC0CTL.bit.TRIGSEL          = 5;    // set SOC0 start trigger on
EPWM1A, due to round-robin SOC0 converts first then SOC1
      AdcRegs.ADCSOC1CTL.bit.TRIGSEL          = 5;    // set SOC1 start trigger on
EPWM1A, due to round-robin SOC0 converts first then SOC1
      AdcRegs.ADCSOC2CTL.bit.TRIGSEL          = 5;    // set SOC2 start trigger on EPWM1A

      //May in the future switch this to start trigger on EPWM2A
```

```
    AdcRegs.ADCSOC3CTL.bit.TRIGSEL      = 5;   // set SOC3 start trigger on EPWM1A
    AdcRegs.ADCSOC4CTL.bit.TRIGSEL      = 5;   // set SOC4 start trigger on EPWM1A

    //May in the future switch this to start trigger on EPWM3A
    AdcRegs.ADCSOC5CTL.bit.TRIGSEL      = 5;   // set SOC5 start trigger on EPWM1A
    AdcRegs.ADCSOC6CTL.bit.TRIGSEL      = 5;   // set SOC6 start trigger on EPWM1A

    AdcRegs.ADCSOC0CTL.bit.ACQPS    = 6;    // set SOC0 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    AdcRegs.ADCSOC1CTL.bit.ACQPS    = 6;    // set SOC1 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    AdcRegs.ADCSOC2CTL.bit.ACQPS    = 6;    // set SOC2 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    AdcRegs.ADCSOC3CTL.bit.ACQPS    = 6;    // set SOC3 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    AdcRegs.ADCSOC4CTL.bit.ACQPS    = 6;    // set SOC4 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    AdcRegs.ADCSOC5CTL.bit.ACQPS    = 6;    // set SOC5 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
    AdcRegs.ADCSOC6CTL.bit.ACQPS    = 6;    // set SOC6 S/H Window to 7 ADC Clock
Cycles, (6 ACQPS plus 1)
        EDIS;

  //Enable CPU timer
  CpuTimer0Regs.TCR.all = 0x4000; // Use write-only instruction to set TSS bit = 1
  CpuTimer1Regs.TCR.all = 0x4000; // Use write-only instruction to set TSS bit = 1

  // Assumes ePWM1 clock is already enabled in InitSysCtrl();
    EPwm1Regs.ETSEL.bit.SOCAEN      = 1;            // Enable SOC on A group
    EPwm1Regs.ETSEL.bit.SOCASEL     = 4;            // Select SOC from CMPA on
upcount
    EPwm1Regs.ETPS.bit.SOCAPRD      = 1;            // Generate pulse on 1st event

    //Check if linker libraries are correct and sin(x) is working properly
    // DO NOT LINK "IQmath_fpu32.lib"
    // DO NOT LINK "rts2800_fpu32_fast_supplement.lib"
        sinZero = sin(0.0); // Result should be 0.0
        sinPIover4 = sin(0.78539816339744830961566084581988); //Result should be
0.7071068
        sinPIover2 = sin(1.57079632679489661923132169163975); //Result should be 1.0

 if (sin(0.0) == 0.0) //Correct linker library check to ensure sin(x) is return correct values
        {

        TestStart = 1; //Start Magic Box test!!!
```

```
                // Step 6. IDLE loop. Just sit and loop forever (optional):
            for(;;)
            {
                    __asm("         NOP");
            }
           }
 //Exit due to bad linker libraries
}

//===========================================================================
=======
// Main() end. Function definitions follow
//===========================================================================
=======

__interrupt void  adc_isr(void)
{

  Vbus[ConversionCount] = AdcResult.ADCRESULT0;
    Va[ConversionCount] = AdcResult.ADCRESULT1;
    Vb[ConversionCount] = AdcResult.ADCRESULT2;
    Vc[ConversionCount] = AdcResult.ADCRESULT3;
    Ia[ConversionCount] = AdcResult.ADCRESULT4;
    Ib[ConversionCount] = AdcResult.ADCRESULT5;
    Ic[ConversionCount] = AdcResult.ADCRESULT6;

  // If 20 conversions have been logged, start over
  if(ConversionCount == 9)
  {
     ConversionCount = 0;
  }
  else ConversionCount++;

  AdcRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;             //Clear ADCINT1 flag reinitialize for
next SOC
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;   // Acknowledge interrupt to PIE

  return;
}

__interrupt void epwm1_isr(void)
{

          // Setup PWM level
            EPwm1Regs.CMPA.half.CMPA = PWM1_CMPA;
```

```
#if 0

  if(EPwm1_DB_Direction == DB_UP)
  {
     if(EPwm1Regs.DBFED < EPWM1_MAX_DB)
     {
       EPwm1Regs.DBFED++;
       EPwm1Regs.DBRED++;
     }
     else
     {
       EPwm1_DB_Direction = DB_DOWN;
       EPwm1Regs.DBFED--;
       EPwm1Regs.DBRED--;
     }
  }
  else
  {
     if(EPwm1Regs.DBFED == EPWM1_MIN_DB)
     {
       EPwm1_DB_Direction = DB_UP;
       EPwm1Regs.DBFED++;
       EPwm1Regs.DBRED++;
     }
     else
     {
       EPwm1Regs.DBFED--;
       EPwm1Regs.DBRED--;
     }
  }
  EPwm1TimerIntCount++;

#endif

  // Clear INT flag for this timer
  EPwm1Regs.ETCLR.bit.INT = 1;

  // Acknowledge this interrupt to receive more interrupts from group 3
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

__interrupt void epwm2_isr(void)
{
        // Setup PWM level
        EPwm2Regs.CMPA.half.CMPA = PWM2_CMPA;
```

```
#if 0

  if(EPwm2_DB_Direction == DB_UP)
  {
    if(EPwm2Regs.DBFED < EPWM2_MAX_DB)
    {
      EPwm2Regs.DBFED++;
      EPwm2Regs.DBRED++;
    }
    else
    {
      EPwm2_DB_Direction = DB_DOWN;
      EPwm2Regs.DBFED--;
      EPwm2Regs.DBRED--;
    }
  }
  else
  {
    if(EPwm2Regs.DBFED == EPWM2_MIN_DB)
    {
      EPwm2_DB_Direction = DB_UP;
      EPwm2Regs.DBFED++;
      EPwm2Regs.DBRED++;
    }
    else
    {
      EPwm2Regs.DBFED--;
      EPwm2Regs.DBRED--;
    }
  }

#endif

  EPwm2TimerIntCount++;

  // Clear INT flag for this timer
  EPwm2Regs.ETCLR.bit.INT = 1;

  // Acknowledge this interrupt to receive more interrupts from group 3
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

__interrupt void epwm3_isr(void)
{
        // Setup PWM level
        EPwm3Regs.CMPA.half.CMPA = PWM3_CMPA;
```

```
#if 0
  if(EPwm3_DB_Direction == DB_UP)
  {
     if(EPwm3Regs.DBFED < EPWM3_MAX_DB)
     {
       EPwm3Regs.DBFED++;
       EPwm3Regs.DBRED++;
     }
     else
     {
       EPwm3_DB_Direction = DB_DOWN;
       EPwm3Regs.DBFED--;
       EPwm3Regs.DBRED--;
     }
  }
  else
  {
     if(EPwm3Regs.DBFED == EPWM3_MIN_DB)
     {
       EPwm3_DB_Direction = DB_UP;
       EPwm3Regs.DBFED++;
       EPwm3Regs.DBRED++;
     }
     else
     {
       EPwm3Regs.DBFED--;
       EPwm3Regs.DBRED--;
     }
  }

#endif

  EPwm3TimerIntCount++;

  // Clear INT flag for this timer
  EPwm3Regs.ETCLR.bit.INT = 1;

  // Acknowledge this interrupt to receive more interrupts from group 3
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP3;
}

void InitEPwm1Example()
{
        //Enable TZ5 and TZ6 as one shot trip sources, so PWMs go to safe state on CPU clock
fail or CPU halt
```

```
        EALLOW;
        //EPwm1Regs.TZSEL.bit.OSHT1 = 1; // 8 One-shot TZ1 select
        //EPwm1Regs.TZSEL.bit.OSHT2 = 1; // 9 One-shot TZ2 select
        EPwm1Regs.TZSEL.bit.OSHT5 = 1; // 12 One-shot TZ5 select
        EPwm1Regs.TZSEL.bit.OSHT6 = 1; // 13 One-shot TZ6 select
        EPwm1Regs.TZSEL.bit.CBC6 = 1; // 13   One-shot TZ6 select

        // What do we want the trip zones to do?
        // Table 13. of STGIPL14K60 IPM needs HIN=low, LIN=high to set to "logic state 0"
        EPwm1Regs.TZCTL.bit.TZA = TZ_FORCE_LO; //EPWM1A connected to HIN
        EPwm1Regs.TZCTL.bit.TZB = TZ_FORCE_HI; //EPWM1B connected to LIN

        // Enable TZ interrupt
        EPwm1Regs.TZEINT.bit.OST = 1;
        EPwm1Regs.TZEINT.bit.CBC = 1;
        EDIS;

  EPwm1Regs.TBPRD = /*6000*/ SP;              // Set timer period (PWM Freq)
  EPwm1Regs.TBPHS.half.TBPHS = 0x0000;        // Phase is 0
  EPwm1Regs.TBCTR = 0x0000;                   // Clear counter

  // Setup TBCLK
  EPwm1Regs.TBCTL.bit.CTRMODE    = TB_COUNT_UPDOWN;   // Count up
  EPwm1Regs.TBCTL.bit.PHSEN      = TB_DISABLE;        // Disable phase loading
  EPwm1Regs.TBCTL.bit.HSPCLKDIV  = TB_DIV1 /*TB_DIV4*/;   // Clock ratio to
SYSCLKOUT
  EPwm1Regs.TBCTL.bit.CLKDIV   = TB_DIV1 /*TB_DIV4*/;

  EPwm1Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;    // Load registers every ZERO
  EPwm1Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
  EPwm1Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO;
  EPwm1Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;

  // Setup compare
  EPwm1Regs.CMPA.half.CMPA = /*3000*/ SP/2; // Initialize to 50% duty

  // Set actions
  EPwm1Regs.AQCTLA.bit.CAU = AQ_SET;            // Set PWM1A on CAU
  EPwm1Regs.AQCTLA.bit.CAD = AQ_CLEAR;                    // Clear PWM1A on CAD

  EPwm1Regs.AQCTLB.bit.CAU = AQ_CLEAR;        // Clear PWM1B on CAU
  EPwm1Regs.AQCTLB.bit.CAD = AQ_SET;          // Set PWM1B on CAD

  // Active Low PWMs - Setup Deadband
  EPwm1Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
  EPwm1Regs.DBCTL.bit.POLSEL = /*DB_ACTV_LO*/ DB_ACTV_HI;
```

```
  EPwm1Regs.DBCTL.bit.IN_MODE = DBA_ALL;
  EPwm1Regs.DBRED = EPWM1_MIN_DB;
  EPwm1Regs.DBFED = EPWM1_MIN_DB;
  EPwm1_DB_Direction = DB_UP;

  // Interrupt where we will change the Deadband
  EPwm1Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;     // Select INT on Zero event
  EPwm1Regs.ETSEL.bit.INTEN = 1;                // Enable INT
  EPwm1Regs.ETPS.bit.INTPRD = ET_3RD;           // Generate INT on 3rd event
}

void InitEPwm2Example()
{
        //Enable TZ5 and TZ6 as one shot trip sources, so PWMs go to safe state on CPU clock
fail or CPU halt
                EALLOW;
                //EPwm2Regs.TZSEL.bit.OSHT1 = 1; // 8 One-shot TZ1 select
                //EPwm2Regs.TZSEL.bit.OSHT2 = 1; // 9 One-shot TZ2 select
                EPwm2Regs.TZSEL.bit.OSHT5 = 1; // 12 One-shot TZ5 select
                EPwm2Regs.TZSEL.bit.OSHT6 = 1; // 13 One-shot TZ6 select

                // What do we want the trip zones to do?
                // Table 13. of STGIPL14K60 IPM needs HIN=low, LIN=high to set to "logic
state 0"
                EPwm2Regs.TZCTL.bit.TZA = TZ_FORCE_LO; //EPWM1A connected to
HIN
                EPwm2Regs.TZCTL.bit.TZB = TZ_FORCE_HI; //EPWM1B connected to LIN

                // Enable TZ interrupt
                EPwm2Regs.TZEINT.bit.OST = 1;
                EDIS;

  EPwm2Regs.TBPRD = /*6000*/ SP;                // Set timer period
  EPwm2Regs.TBPHS.half.TBPHS = 0x0000;          // Phase is 0
  EPwm2Regs.TBCTR = 0x0000;                     // Clear counter

  // Setup TBCLK
  EPwm2Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up
  EPwm2Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Disable phase loading
  EPwm2Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1 /*TB_DIV4*/;      // Clock ratio to
SYSCLKOUT
  EPwm2Regs.TBCTL.bit.CLKDIV = TB_DIV1 /*TB_DIV4*/;         // Slow just to observe on
the scope

  // Setup compare
  EPwm2Regs.CMPA.half.CMPA = /*3000*/ SP/2; // Initialize to 50% duty
```

```
  // Set actions
  EPwm2Regs.AQCTLA.bit.CAU = AQ_SET;            // Set PWM2A on CAU
  EPwm2Regs.AQCTLA.bit.CAD = AQ_CLEAR;          // Clear PWM2A on CAD

  EPwm2Regs.AQCTLB.bit.CAU = AQ_CLEAR;          // Clear PWM2B on CAU
  EPwm2Regs.AQCTLB.bit.CAD = AQ_SET;            // Set PWM2B on CAD

  // Active Low complementary PWMs - setup the deadband
  EPwm2Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
  EPwm2Regs.DBCTL.bit.POLSEL = /*DB_ACTV_LOC*/DB_ACTV_HI;
  EPwm2Regs.DBCTL.bit.IN_MODE = DBA_ALL;
  EPwm2Regs.DBRED = EPWM2_MIN_DB;
  EPwm2Regs.DBFED = EPWM2_MIN_DB;
  EPwm2_DB_Direction = DB_UP;

  // Interrupt where we will modify the deadband
  EPwm2Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;     // Select INT on Zero event
  EPwm2Regs.ETSEL.bit.INTEN = 1;                // Enable INT
  EPwm2Regs.ETPS.bit.INTPRD = ET_3RD;           // Generate INT on 3rd event
}

void InitEPwm3Example()
{
        //Enable TZ5 and TZ6 as one shot trip sources, so PWMs go to safe state on CPU clock
fail or CPU halt
                EALLOW;
                //EPwm3Regs.TZSEL.bit.OSHT1 = 1; // 8 One-shot TZ1 select
                //EPwm3Regs.TZSEL.bit.OSHT2 = 1; // 9 One-shot TZ2 select
                EPwm3Regs.TZSEL.bit.OSHT5 = 1; // 12 One-shot TZ5 select
                EPwm3Regs.TZSEL.bit.OSHT6 = 1; // 13 One-shot TZ6 select

                // What do we want the trip zones to do?
                // Table 13. of STGIPL14K60 IPM needs HIN=low, LIN=high to set to "logic
state 0"
                EPwm3Regs.TZCTL.bit.TZA = TZ_FORCE_LO; //EPWM1A connected to
HIN
                EPwm3Regs.TZCTL.bit.TZB = TZ_FORCE_HI; //EPWM1B connected to LIN

                // Enable TZ interrupt
                EPwm3Regs.TZEINT.bit.OST = 1;
                EDIS;

  EPwm3Regs.TBPRD = /*6000*/ SP;                // Set timer period
  EPwm3Regs.TBPHS.half.TBPHS = 0x0000;          // Phase is 0
  EPwm3Regs.TBCTR = 0x0000;                     // Clear counter
```

```
  // Setup TBCLK
  EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN; // Count up
  EPwm3Regs.TBCTL.bit.PHSEN = TB_DISABLE;       // Disable phase loading
  EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1 /*TB_DIV4*/;       // Clock ratio to
SYSCLKOUT
  EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1 /*TB_DIV4*/;         // Slow so we can observe
on the scope

  // Setup compare
  EPwm3Regs.CMPA.half.CMPA = /*3000*/ SP/2; // Initialize to 50% duty

  // Set actions
  EPwm3Regs.AQCTLA.bit.CAU = AQ_SET;            // Set PWM3A on CAU
  EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;          // Clear PWM3A on CAD

  EPwm3Regs.AQCTLB.bit.CAU = AQ_CLEAR;          // Clear PWM3B on CAU
  EPwm3Regs.AQCTLB.bit.CAD = AQ_SET;            // Set PWM3B on CAD

  // Active high complementary PWMs - Setup the deadband
  EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE;
  EPwm3Regs.DBCTL.bit.POLSEL = /*DB_ACTV_HIC*/ DB_ACTV_HI;
  EPwm3Regs.DBCTL.bit.IN_MODE = DBA_ALL;
  EPwm3Regs.DBRED = EPWM3_MIN_DB;
  EPwm3Regs.DBFED = EPWM3_MIN_DB;
  EPwm3_DB_Direction = DB_UP;

  // Interrupt where we will change the deadband
  EPwm3Regs.ETSEL.bit.INTSEL = ET_CTR_ZERO;      // Select INT on Zero event
  EPwm3Regs.ETSEL.bit.INTEN = 1;                // Enable INT
  EPwm3Regs.ETPS.bit.INTPRD = ET_3RD;           // Generate INT on 3rd event
}

__interrupt void cpu_timer0_isr(void)
{
        //CpuTimer0.InterruptCount++;
        static uint16_t cycleCount = 0;

        //Flip the magnetic polarity
        polarity *= -1;

        //Bound current drive to between 0-100 [%]
                        if (Current_Pct < 0.0)
                                Current_Pct = 0.0;
                        if (Current_Pct > 100.0)
                                Current_Pct = 100.0;
```

```
        I_pct = Current_Pct / 100.0; //Calculated percent

        // When CMPA is zeroed, PWM mis-behaves. For now give a little headroom, until
problem solved
        I_pct *= 0.95; //limit to 95% max

        //Calculate magnitude and reversal polarity for PWMs for test angle
        sinPHa = sin((double)(TestAngle_deg*DEG_2_RAD)) * polarity * I_pct;
        sinPHb = sin((double)((TestAngle_deg+120)*DEG_2_RAD)) * polarity * I_pct;
        sinPHc = sin((double)((TestAngle_deg+240)*DEG_2_RAD)) * polarity * I_pct;

        //Calculate CMPA offsets to create vector magnitude for each phase
        PHa_mag = (int16)(sinPHa * SP/2);
        PHb_mag = (int16)(sinPHb * SP/2);
        PHc_mag = (int16)(sinPHc * SP/2);

        if (angleOn)
        {
                //Set all PWMs to levels needed for magnetic test angle
                PWM1_CMPA = SP/2 - PHa_mag;
                PWM2_CMPA = SP/2 - PHb_mag;
                PWM3_CMPA = SP/2 - PHc_mag;

                //Toggle the Red and Blue LEDs to show an angle test in progress
                GpioDataRegs.GPBTOGGLE.bit.GPIO34 = 1;
                GpioDataRegs.GPBTOGGLE.bit.GPIO39 = 1;
        }
        else
        {
                //Set all PWMs to 50% which turns off current through motor since all voltages
equal
                PWM1_CMPA = SP/2;
                PWM2_CMPA = SP/2;
                PWM3_CMPA = SP/2;
        }



#if 0
        if(tempDelta > 10){
                //Red LED on, blue off
           GpioDataRegs.GPBDAT.bit.GPIO34 = 0;
           GpioDataRegs.GPBDAT.bit.GPIO39 = 1;
        }else if(tempDelta < -10){
```

```
            GpioDataRegs.GPBDAT.bit.GPIO34 = 1;
            GpioDataRegs.GPBDAT.bit.GPIO39 = 0;
        }else{
                if(tempDelta > 0){
                    GpioDataRegs.GPBDAT.bit.GPIO39 = 1;
                    if(tempDelta > cycleCount){
                    GpioDataRegs.GPBDAT.bit.GPIO34 = 0;
                    }else{
                            GpioDataRegs.GPBDAT.bit.GPIO34 = 1;
                    }

                }else{
                    GpioDataRegs.GPBDAT.bit.GPIO34 = 1;
                    if(abs(tempDelta) > cycleCount){
                    GpioDataRegs.GPBDAT.bit.GPIO39 = 0;
                    }else{
                            GpioDataRegs.GPBDAT.bit.GPIO39 = 1;
                    }
                }

        }

#endif

        cycleCount++;

        if(cycleCount == 10)
                cycleCount = 0;

  // Acknowledge this interrupt to receive more interrupts from group 1
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
}

// PWM on/off timer and angle advanced
__interrupt void cpu_timer1_isr(void)
{
            CpuTimer1.InterruptCount++;
            // The CPU acknowledges the interrupt.
            EDIS;

            if (TestAngle_deg > maxTestAngle )
            {
                    TestStart = 0; //Stop testing
                    angleOn = 0;   //Turn off test angle

                    //Force a software 'Trip Zone' event to put PWM outputs into safe off mode
```

```
            EALLOW;
            EPwm1Regs.TZFRC.bit.OST = 1;
            EPwm2Regs.TZFRC.bit.OST = 1;
            EPwm3Regs.TZFRC.bit.OST = 1;
            EDIS;
        }
        if(TestStart)
        {
            if (onOffCount%2)
            {
                angleOn = 0; //Turn off motor drive by setting all PWM levels to 50%
                TestAngle_deg += angleStepSize;
            }
            else
            {
                angleOn = 1; //Turn on motor drive for motor angle
            }
            onOffCount++;
        }
}

__interrupt void epwm1_tzint_isr(void)
{
  EPwm1TZIntCount++;

// Leave these flags set so we only take this
// interrupt once
//
// EALLOW;
// EPwm1Regs.TZCLR.bit.OST = 1;
// EPwm1Regs.TZCLR.bit.INT = 1;
// EDIS;

  // Acknowledge this interrupt to receive more interrupts from group 2
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;
}

__interrupt void epwm2_tzint_isr(void)
{

  EPwm2TZIntCount++;

// Clear the flags - we will continue to take
// this interrupt until the TZ pin goes high
//
  EALLOW;
```

```
  EPwm2Regs.TZCLR.bit.CBC = 1;
  EPwm2Regs.TZCLR.bit.INT = 1;
  EDIS;

  // Acknowledge this interrupt to receive more interrupts from group 2
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;
}

__interrupt void epwm3_tzint_isr(void)
{

  EPwm3TZIntCount++;

// Clear the flags - we will continue to take
// this interrupt until the TZ pin goes high
//
  EALLOW;
  EPwm3Regs.TZCLR.bit.CBC = 1;
  EPwm3Regs.TZCLR.bit.INT = 1;
  EDIS;

  // Acknowledge this interrupt to receive more interrupts from group 2
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;
}

// This function will copy the specified memory contents from
// one location to another.
//
//        Uint16 *SourceAddr       Pointer to the first word to be moved
//                        SourceAddr < SourceEndAddr
//        Uint16* SourceEndAddr    Pointer to the last word to be moved
//        Uint16* DestAddr         Pointer to the first destination word
//
// No checks are made for invalid memory locations or that the
// end address is > then the first start address.

void MemCopy(Uint16 *SourceAddr, Uint16* SourceEndAddr, Uint16* DestAddr)
{
   while(SourceAddr < SourceEndAddr)
   {
     *DestAddr++ = *SourceAddr++;
   }
   return;
}
```

```
//=================================================================================
========
// No more.
//=================================================================================
========
```

List of Abbreviations

ACIM – AC Induction motor

A/D - Analog to Digital converter

BEMF – Back Electro-Motive Force

BLDC – Brushless DC (motor)

BRBs – Broken Rotor Bars

CTs – Current Transformers

CTFS – Continuous-Time Fourier Series

CTFT – Continuous-Time Fourier Transform

DAQ Card – Data Acquisition Card

DFT – Discrete Fourier Transform

DTFT – Discrete-Time Fourier Transform

DTFS – Discrete-Time Fourier Series

DTFSC – Discrete-Time Fourier Series Coefficients

DQ – Direct-Quadrature

DQZ – Direct-Quadrature-Zero

EHRPWM – Enhanced High Resolution Pulse Width Modulator

FFT – Fast Fourier Transform

FS – Fourier Series

FSC - Fourier Series Coefficients

H - Two sets of high and low side switches (in the shape of the letter H)

IPM – Intelligent Power Module

J – Joule

LP – Low Pass

LPF - Low Pass Filter

LR – Locked Rotor

MB – Magic Box, also known as the ‘Offline broken rotor bar tester’

NEMA – National Electrical Manufacturers Association

PCB – Printed Circuit Board

PWM – Pulse Width Modulation

PTs – Potential Transformers

SD Card – Secure Digital Card

SOC – System on a Chip (eg. AM335x)

SPRT – Single-Phase Rotation Test

TEFC – Totally enclosed, fan cooled

TI - Texas Instruments

TRM - Technical Reference Manual

USB - Universal Serial Bus