TI Designs
4.4 to 30 V, 15 A, High Performance Brushless DC Propeller Controller Reference Design

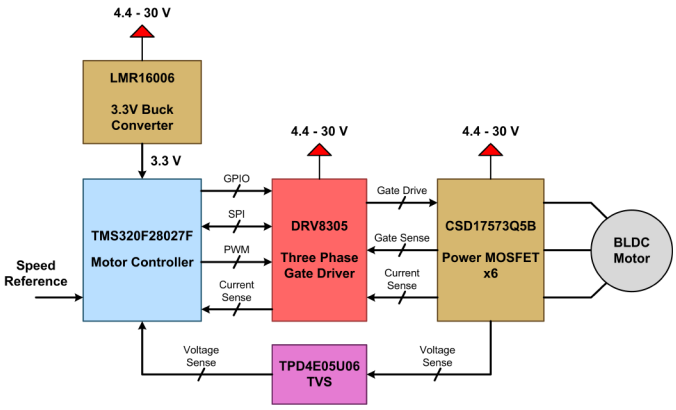
Design Overview

The TIDA-00643 reference design is a 4.4 to 30 V brushless DC motor controller for high power propeller, fan, and pump applications. It uses the Texas Instrument's DRV8305 brushless DC motor gate driver, CSD17573Q5B 30V NexFET™ power MOSFETs, TPD4E05U06 TVS protection IC, C2000 motor control MCU, and LMR16006 3.3 V buck converter. It utilizes InstaSPIN™-FOC for sensorless field oriented motor control and commands the motor speed through an external reference signal from a central controller. This design is focused on demonstrating a highly efficient and high power BLDC motor system.

Design Resources

TIDA-00643	Design Folder
DRV8305	Product Folder
TMS320F28027F	Product Folder
LMR16006	Product Folder
TPD4E05U06	Product Folder
CSD17573Q5B	Product Folder
InstaSPIN™	Product Folder
MotorWare™	Tool Folder
BOOSTXL-DRV8305EVM	Tool Folder

Block Diagram



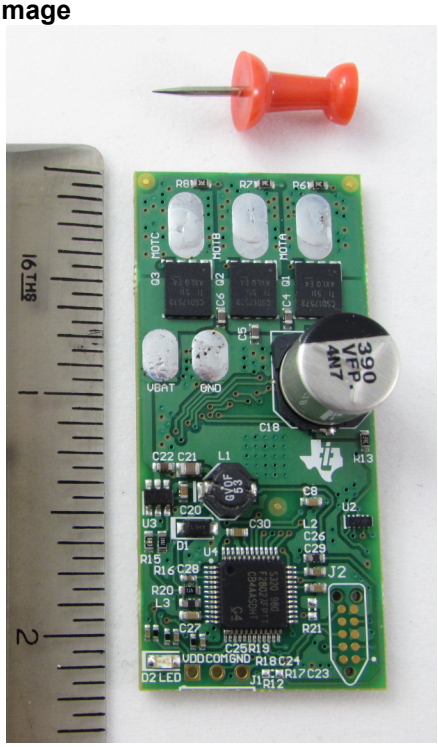
Design Features

- 4.4 V to 30 V input voltage range
- 15 A RMS, 23 A peak output current capability
- Small form factor (L x W): 2.2" x 1.0"
- Speed control with single reference signal
- Onboard 3.3 V, 0.6 A buck converter
- Motor control through InstaSPIN-FOC™ sensorless field oriented control
- Wide array of system protection features including MOSFET V_{DS} overcurrent and supply undervoltage protection

Featured Applications

- Drone Propeller Electronic Speed Controller
- Unmanned Air Vehicles
- Remote Control Applications with Speed Control Motor Drive
- High Power Fans and Pumps

Board Image



1 System Description

TIDA-00643 is compact motor controller designed for high power, brushless DC (BLDC) propellers. It mimics the function of an off the shelf electronic speed controller (ESC) often used in RC and hobby applications. The motor controller decodes a speed reference from a central controller and processes this into the appropriate drive signals for the BLDC motor. This allows the central controller to off load the processor functions related to BLDC motor control and focus on functions central to the system. This type of motor controller can also be used for high power fan or pump applications.

The motor controller is composed of two main components. The first component is the MCU, which decodes the speed reference signal from the central controller, measures the motor's back-EMF and current signals, and sends the appropriate control signals for the power stage. The second component is the power stage which consists of the gate driver and power MOSFETs. The power stage amplifies the control signals from the MCU to the motor.

The motor controller uses InstaSPIN-FOC, a sensorless field oriented control algorithm for brushless DC motors. Field oriented control allows for optimal efficiency and noise performance from the motor that is being driven by the controller. InstaSPIN^{TM}-FOC uses the signals from the motor back-EMF and phases currents to interpolate where the motor rotor is located and send the correct drive patterns. Power is supplied to the motor controller from the main power input through a switching buck converter.

1.1 DRV8305 Three Phase Motor Gate Driver

The DRV8305 is used as a gate driver for the external N-channel power MOSFETs. The MOSFETs are connected in an inverter or triple half-bridge configuration to drive each winding of the three phase BLDC motor. The gate driver receives logic level inputs from the microcontroller and amplifies these to the battery voltage rail for the motor. The DRV8305 is a highly integrated device that runs off of a single power supply with minimal external components. It has accurate slew rate control for the external power MOSFET to manage the system switching performance.

The DRV8305 provides system protection through V_{DS} voltage monitors for the external power MOSFETs. The V_{DS} monitors can be configured to detect an overcurrent event in the external MOSFETs that might result from a MOSFET or motor short circuit condition. The DRV8305 has power supply and gate supply undervoltage detection to prevent an under drive condition as well as multilevel temperature flag and fault reporting.

The DRV8305 also provides three, high performance low-side current shunt amplifiers to monitor the motor phase currents. The current shunt amplifiers receive that voltage across a sense resistor and amplify it to the microcontroller. The amplifiers have several configurable settings including gain, blanking time, and reference offset.

1.2 TMS320F28027F Piccolo Microcontroller with InstaSPIN-FOC

The TMS320F28027F is the brain of the motor controller. Its uses the feedback signals from the motor and the speed reference signals to determine the proper signal pattern for the brushless DC motor.

The MCU supports InstaSPIN-FOC, a sensorless field oriented control algorithm for high efficiency and performance applications. Sensorless algorithms remove the need for a mechanical motor rotor sensor in order to reduce system costs and weight. InstaSPIN™-FOC uses TI's FAST™ (flux, angle, speed, and torque) software encoder (sensorless observer) to obtain the rotor position.

Through the MCU ADCs the system obtains the motor back-EMF and phase currents. The high-performance EPWM modules provide the PWM modulation signals to the power stage, and the ECAP module receives the speed reference signal from the central controller.

1.3 LMR16006 Step-Down Buck Converter

The LMR16006 provides an efficient, regulated 3.3 V power supply from the unregulated battery input. The 3.3 V rail supplies the C2000 MCU for the motor controller. The LMR16006 incorporates a 700 kHz switching frequency, internal soft-start, and compensation circuits to minimize external footprints and board space.

1.4 CSD17573Q5B NexFET™ N-Channel Power MOSFET

The motor controller uses six CSD17573Q5B to form the inverter for the brushless DC motor. This power MOSFET is an ultra-low $R_{DS(on)}$ device designed to minimize losses in power conversion, switching applications. It comes in a compact, 8 pin SON 5 x 6 mm package with an $R_{DS(on)}$ of 0.84 mΩ at a V_{GS} of 10 V to minimize board space required and limit thermal dissipation.

T_A = 25°C		TYPICAL VALUE		UNIT
V_{DS}	Drain-to-Source Voltage	30		V
Q_g	Gate Charge Total (4.5 V)	49		nC
Q_{gd}	Gate Charge Gate-to-Drain	11.9		nC
$R_{DS(on)}$	Drain-to-Source On-Resistance	V_{GS} = 4.5 V	1.19	mΩ
		V_{GS} = 10 V	0.84	mΩ
$V_{GS(th)}$	Threshold Voltage	1.4		V

Figure 1: CSD17573Q5B Summary

1.5 TPD4E05U06 Transient Voltage Suppressor

The TPD4E05U06 is a unidirectional Transient Voltage Suppressor (TVS) based electrostatic discharge (ESD) protection diodes with ultra-low capacitance. Each device can dissipate ESD strikes above the maximum level specified by the IEC 61000-4-2 international standard.

2 Block Diagram

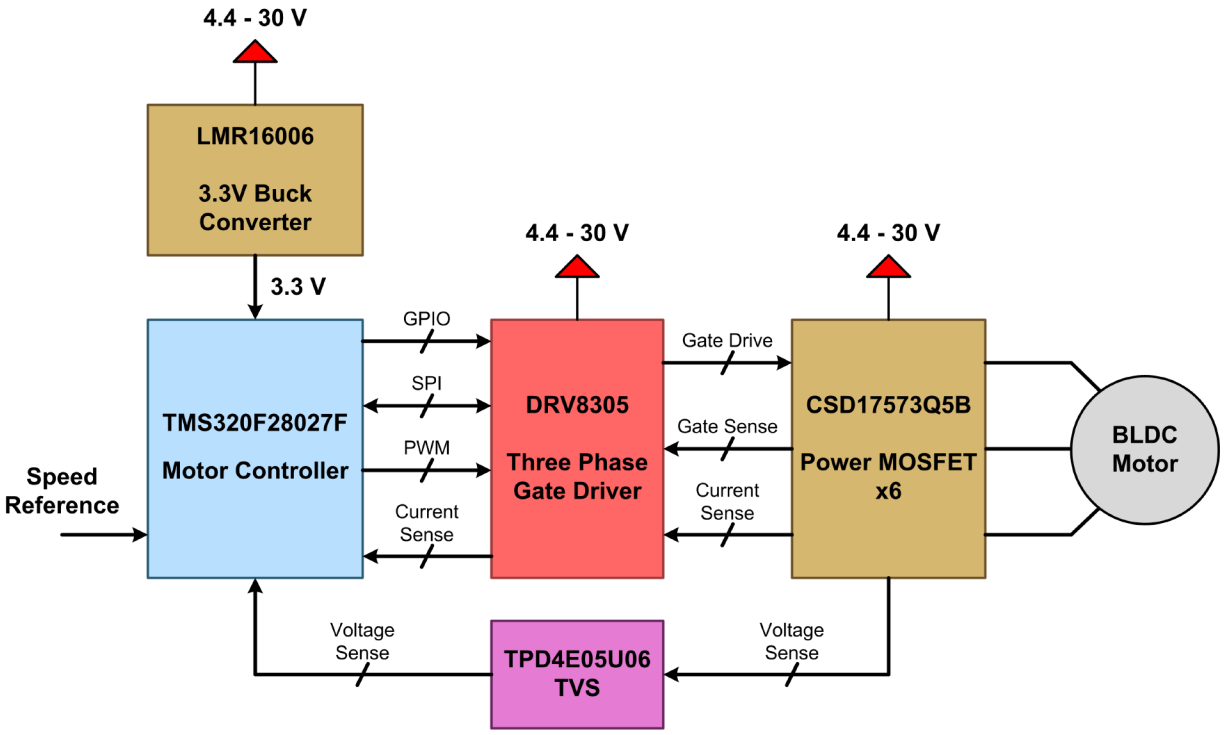


Figure 2: TIDA-00643 Block Diagram

2.1 Highlighted Products

The critical devices to this design are outlined below.

- DRV8305: Three phase motor gate driver with current shunt amplifiers
- TMS320F28027F: 60 MHz Piccolo microcontroller with InstaSPIN-FOC

2.1.1 DRV8305

The DRV8305 is a gate driver IC for three phase motor drive applications. It provides three highly accurate, trimmed, and temperature compensated half bridge drivers, each capable of driving a high-side and low-side N-channel power MOSFET. A charge pump driver supports 100% duty cycle and low voltage operation. The gate driver can also handle load dumps up to 45 V.

The gate driver uses automatic handshaking when switching to prevent current shoot through in the external power MOSFETs. Accurate VDS sensing is utilized for the high-side and low-side MOSFET to detect and overcurrent condition.

The DRV8305 includes three bidirectional current shunt amplifiers with adjustable gain levels and blanking times for accurate low-side current measurements. The SPI interface and device registers provide detailed fault reporting and flexible parameter settings such as the current shunt amplifier configurations, gate driver slew rate control, and numerous protection features.

A block diagram for the gate driver is shown below.

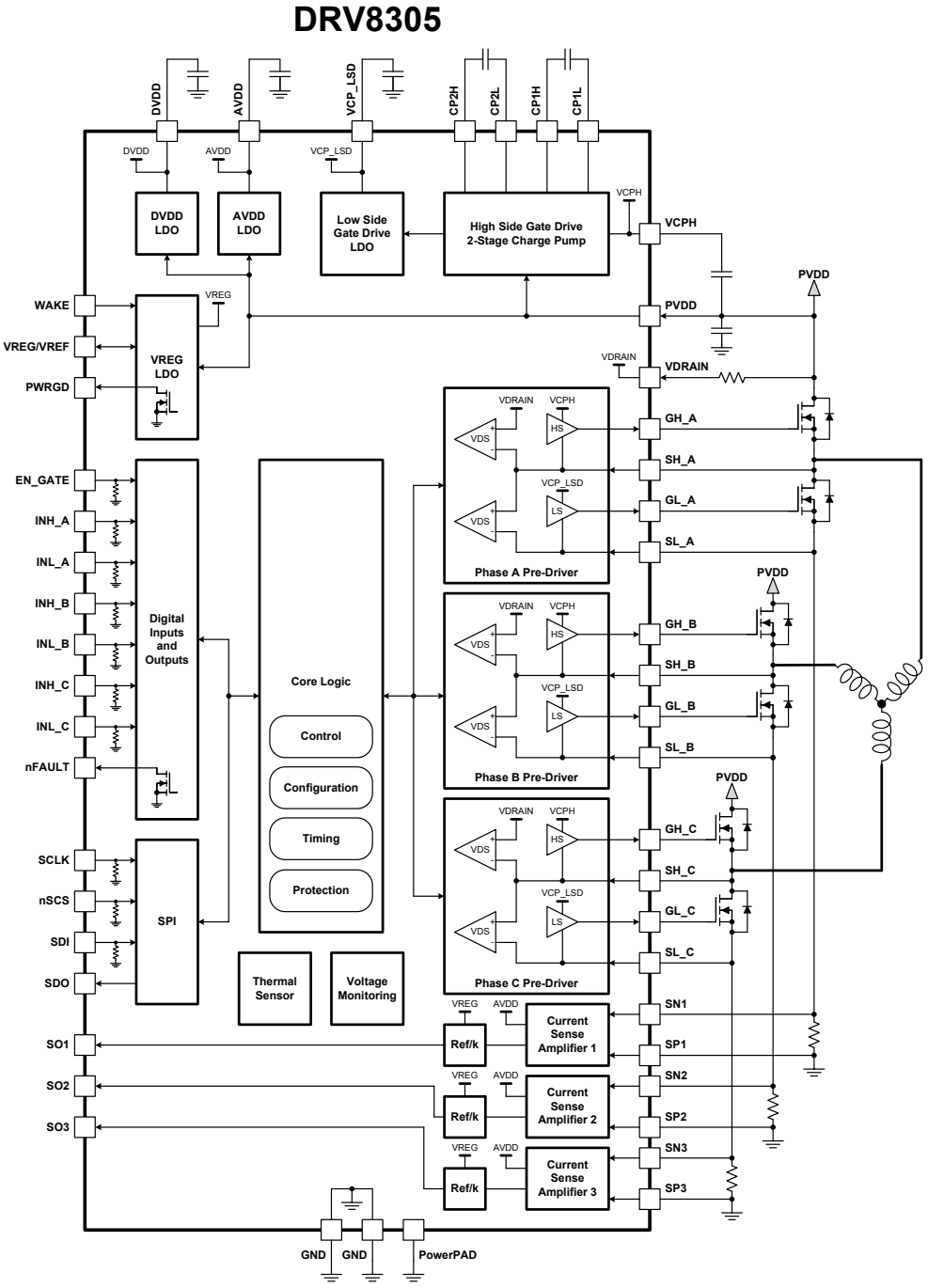


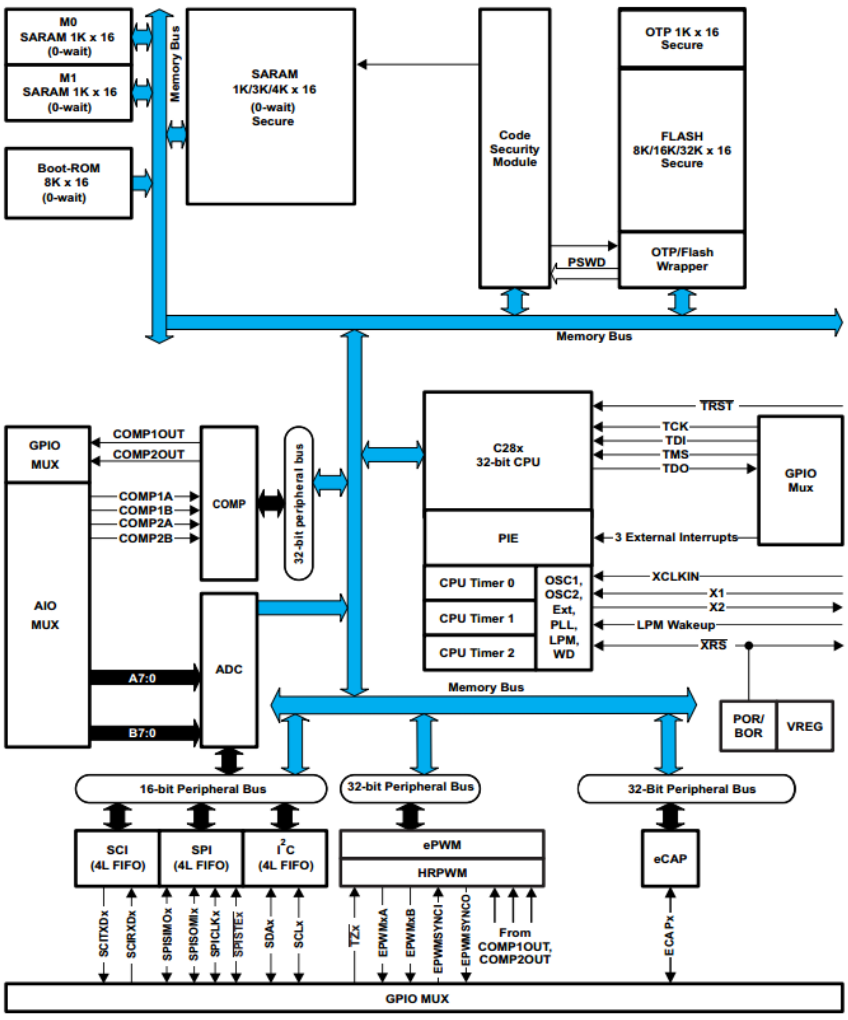
Figure 3: DRV8305 Block Diagram

2.1.2 TMS320F28027F

The F2802x Piccolo family of microcontrollers provides the power of the C28xx core coupled with highly integrated control peripherals in low pin-count devices. The family is code-compatible with previous C28x-based code, as well as providing a high level of analog integration.

An internal voltage regulator allows for single-rail operation. Enhancements have been made to the HRPWM module to allow for dual-edge control (frequency modulation). Analog comparators with internal 10-bit references have been added and can be routed directly to control the PWM outputs. The ADC converts from 0 to 3.3-V fixed full-scale range and supports ratio-metric V_{REFHI}/V_{REFLO} references. The ADC interface has been optimized for low overhead and latency.

These devices have special motor control software in execute only ROM to enable InstaSPIN-FOC or InstaSPIN-MOTION solutions, with system software support through MotorWare. While standard C2000 controlSUITE software can be used with these devices please note that this special ROM replaces the standard ROM, meaning certain software functions that controlSUITE projects expect to be in ROM will need to be linked into the project. Please see the InstaSPIN-FOC and InstaSPIN-MOTION Memory Considerations section of the InstaSPIN User's Guide, or the Memory section of the datasheet for more details.



A. Not all peripheral pins are available at the same time due to multiplexing.

Figure 4: TMS430F28027F Block Diagram

3 System Design Theory

The 4.4 to 30 V, 15 A, High Performance Brushless DC Propeller Controller Reference Design demonstrates a high performance three phase BLDC motor controller using the DRV8305 motor gate driver and TMS320F28027F microcontroller.

The motor controller receives speed commands from an external reference signal that is fed to the TMS320F28027F MCU ECAP module. Using the speed reference and feedback signals from the motor the MCU determines the correct signals to send to the power stage composed of the DRV8305 and CSD17573Q5B.

Section 4.1 will describe the hardware design theory and section 4.2 will describe the software theory of the reference design.

3.1 Hardware System Design Theory

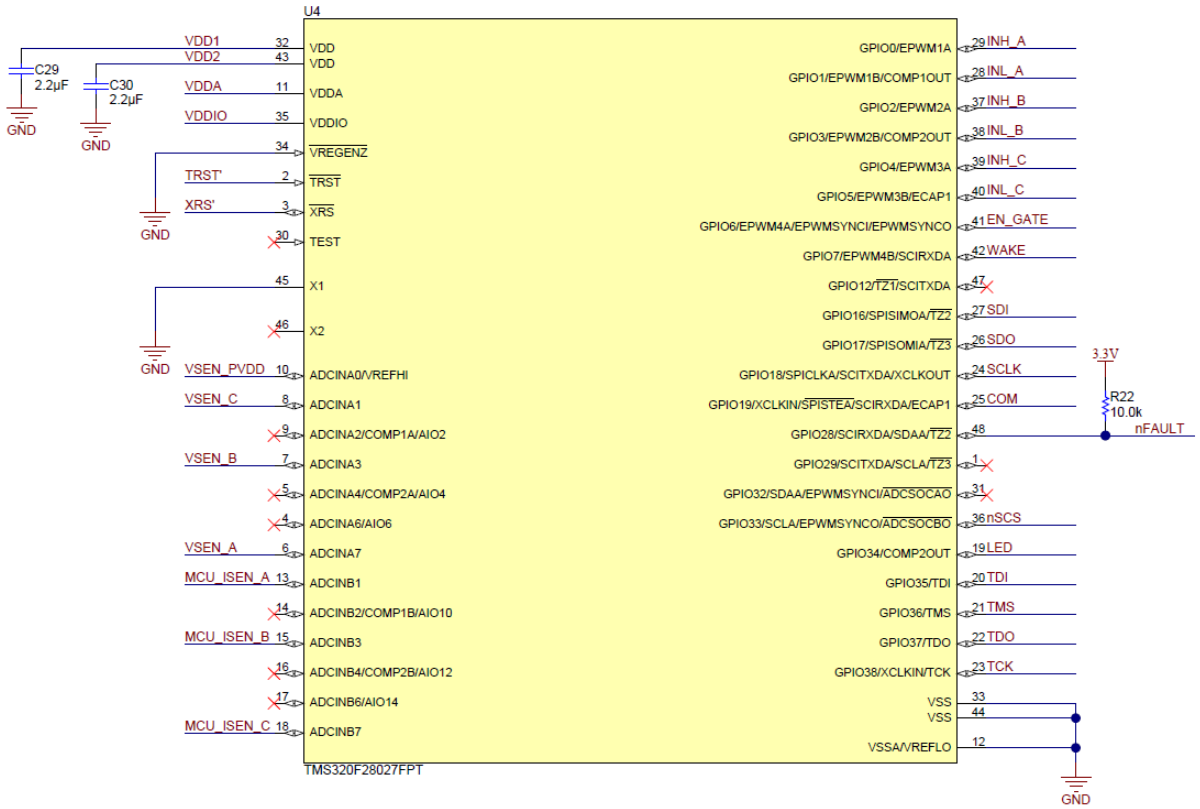


Figure 5: TMS320F28027F Block

The TMS320F28027F has been configured to require minimal external components and use its internal oscillator for the 60 MHz clock. The 3.3 V power supply is supplied from the external buck regulator.

The EPWM1, EPWM2, and EPWM3 enhanced PWM modules are used to generate the pulse width signals for the switching power stage. Multiple ADC channels are utilized to read the motor back-EMF currents and phase currents. A JTAG connector is utilized to program the memory. Several GPIO and an SPI interface communicate with the DRV8305 to set the gate driver modes, configurations, and read back status information.

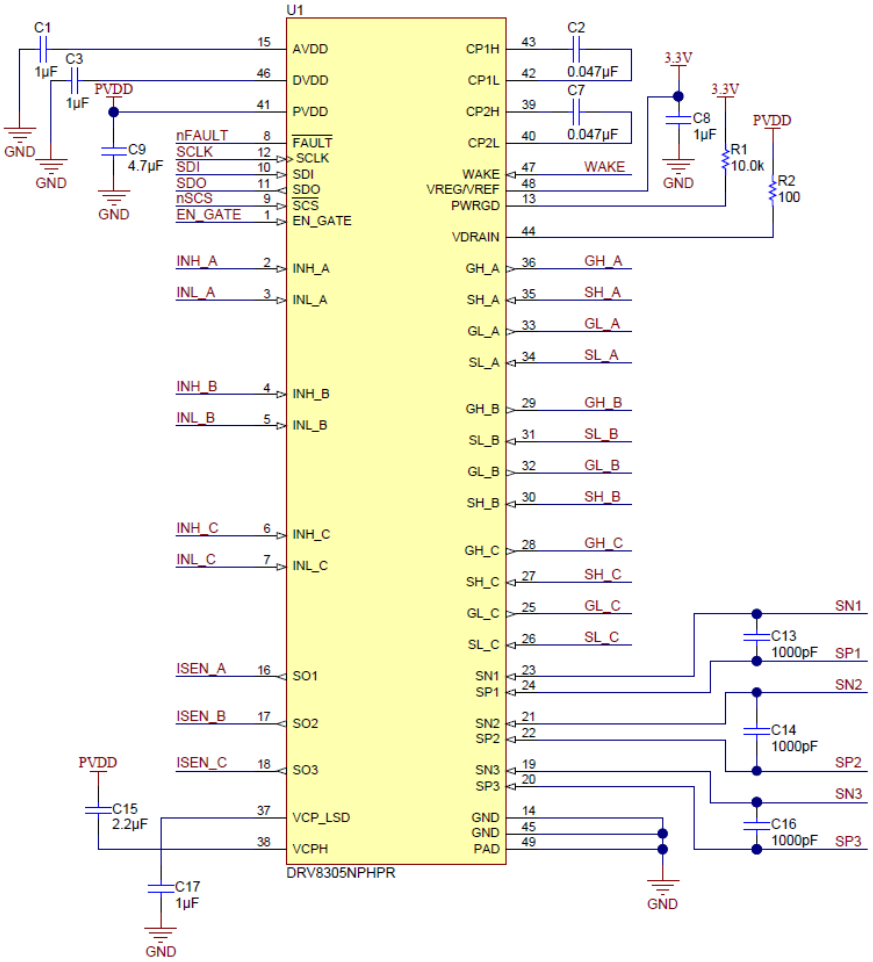


Figure 6: DRV8305 Block

The DRV8305 is a highly integrated three phase gate driver with features specifically for motor drive. It has a single power supply (PVDD) that is bypassed with a 4.7 µF ceramic capacitor. Two internal regulators (AVDD and DVDD) are externally bypassed with 1 µF ceramic capacitors. A tripler charge pump (VCPH, CP1, and CP2) is utilized to generate the voltage supply (PVDD + 10 V) for the high-side N-channel power MOSFETs and a linear regulator (VCP_LSD) provides the 10 V supply for the low-side MOSFETs.

The internal current shunt amplifiers are referenced to the VREG/VREF pin which is supplied from the MCU 3.3 V supply and there settings are configured over the SPI interface. The INH_X and INL_X pins control the state of the half-bridge drivers with GH_X driving the high-side MOSFET gate and GL_X driving the low-side MOSFET gate. The gate driver amplifies the logic level inputs from the MCU to the battery supply voltage (PVDD). The dedicated VDRAIN pin is used to sense the drain voltage for the high-side MOSFETs for use with the V_{DS} overcurrent monitors. The WAKE pin is used to wake the device from its low power sleep mode.

The DRV8305 provides configurations for managing the MOSFET slew rate and switching performance along with protection features such as automatic switching handshaking, overcurrent, undervoltage, and overtemperature protection.

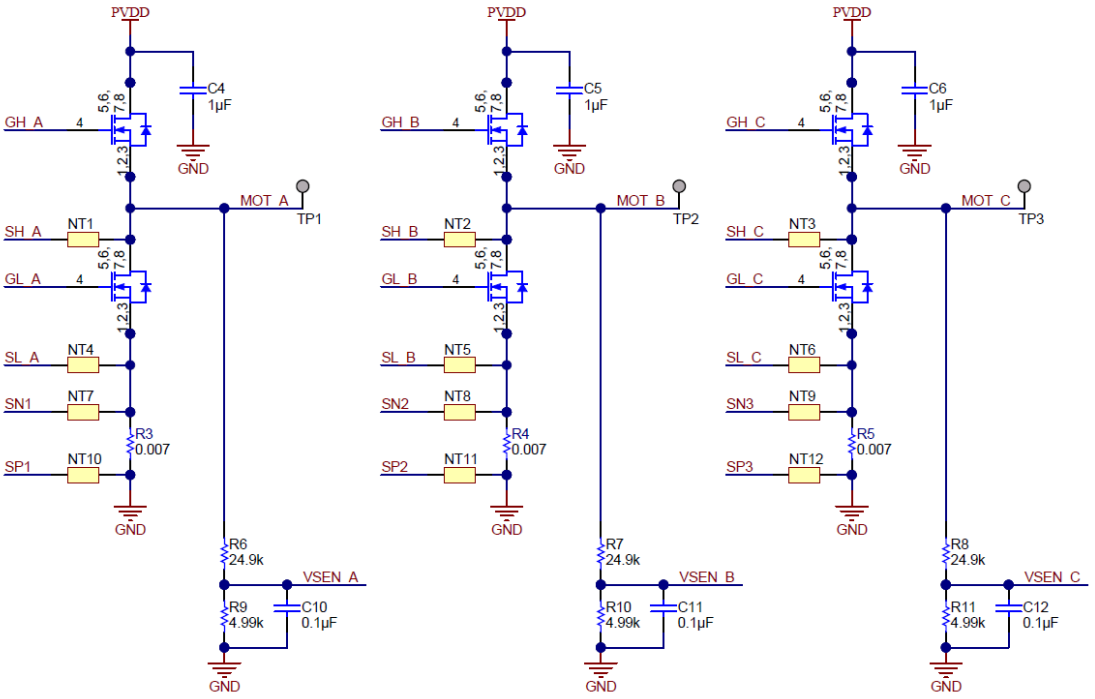


Figure 7: Inverter Block

Six CSD17573Q5B N-channel power MOSFETs form the inverter for the brushless DC motor. The inverter allows the controller to manage the voltage (and proportionally the current) across the motor windings. Applying the correct voltage/current pattern allows the motor to rotate. To improve the efficiency the design, the MOSFET are driven with the DRV8305 gate driver. This allows for minimal conduction and switching losses. To understand more about motor gate drivers and MOSFETs see this app note.

In addition to the MOSFETs, a voltage divider is placed on each motor phase in order to measure the motor back-EMF. The voltage divider is fed directly into the MCU ADC. Back-EMF is voltage that the motor generates when it is spinning. It is used to help determine the rotor position. Three current shunt resistors are placed on the low-side of each half-bridge in order to determine the current through each leg. The current shunt resistor voltage is fed into the DRV8305 current shunt amplifiers and then to the MCU ADCs. This is also used to help determine the rotor position.

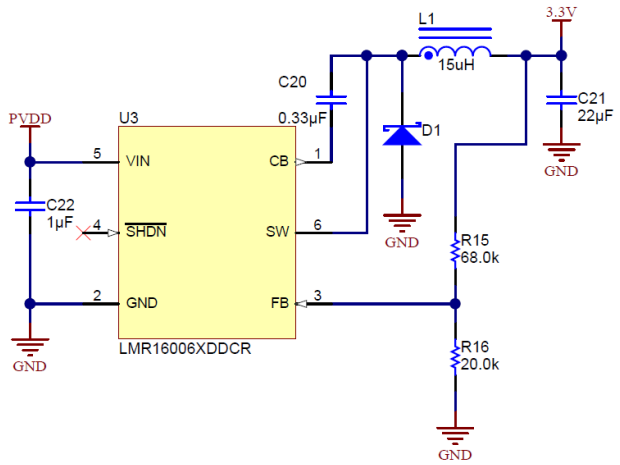


Figure 8: 3.3 V Block

The LMR16006 is used as the basis for the 3.3 V supply. It is a regulated buck converter that takes the battery input voltage and steps it down to 3.3 V for the MCU. The input supply is bypassed with a 1 µF ceramic capacitor and the output is filter with a 15 µH power inductor and a 22 µF ceramic capacitor. A feedback loop enables the converter to regulate to 3.3 V.

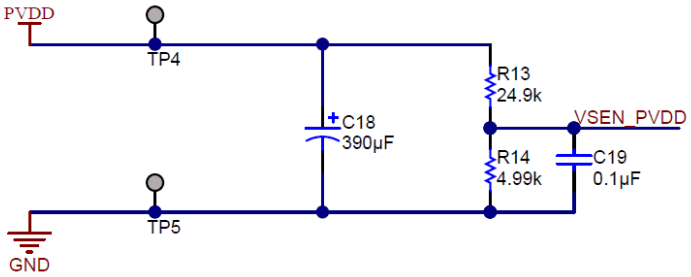


Figure 9: Power Input Block

The battery supply is connected directly to the board and filtered with a 390 µF bulk electrolytic capacitor. A voltage divider allows the MCU to sense the supply voltage directly.

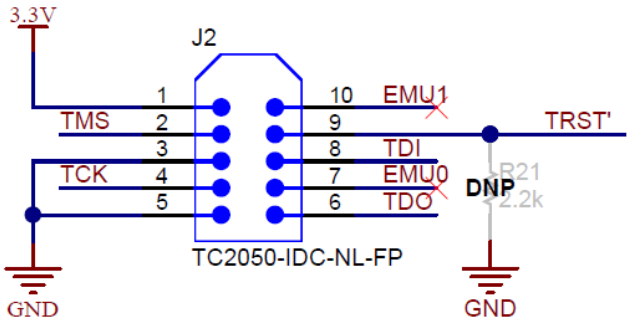


Figure 10: JTAG Block

Programming and debugging are enabled by a JTAG XDS100v2 emulator. A compact Tag-Connect adapter was utilized to minimize board footprint for the connector.

3.2 Software System Design Theory

The motor controller software is based on the InstaSPIN-FOC technology and MotorWare infrastructure. The final system is based around Lab 5b from the MotorWare projects which utilizes InstaSPIN-FOC with a PI speed controller.

Lab 5b uses two cascaded proportional-integral (PI) controllers, one for the current controller and one for the speed controller. In this system, the speed controller is wrapped around the current controller. A PI controller is a common control loop feedback mechanism that is used in a variety of system. The PI controller calculates an error value as the difference between a measured variable and the desired set point. It then attempts to correct for this error with a specified response system.

In the TIDA-00463 reference design, the motor is the brushless DC propeller motor and the load inertia is from the propeller. The entire control system is instantiated on the TMS320F28027F and the power stage serves as the interface between the controller and the motor.

The block diagram below shows the entire system with the FAST observer, field oriented control (FOC) algorithm, ADC feedback, space vector modulation (SVM), and PI controllers. To learn more about FOC control you can watch this video or read application report.

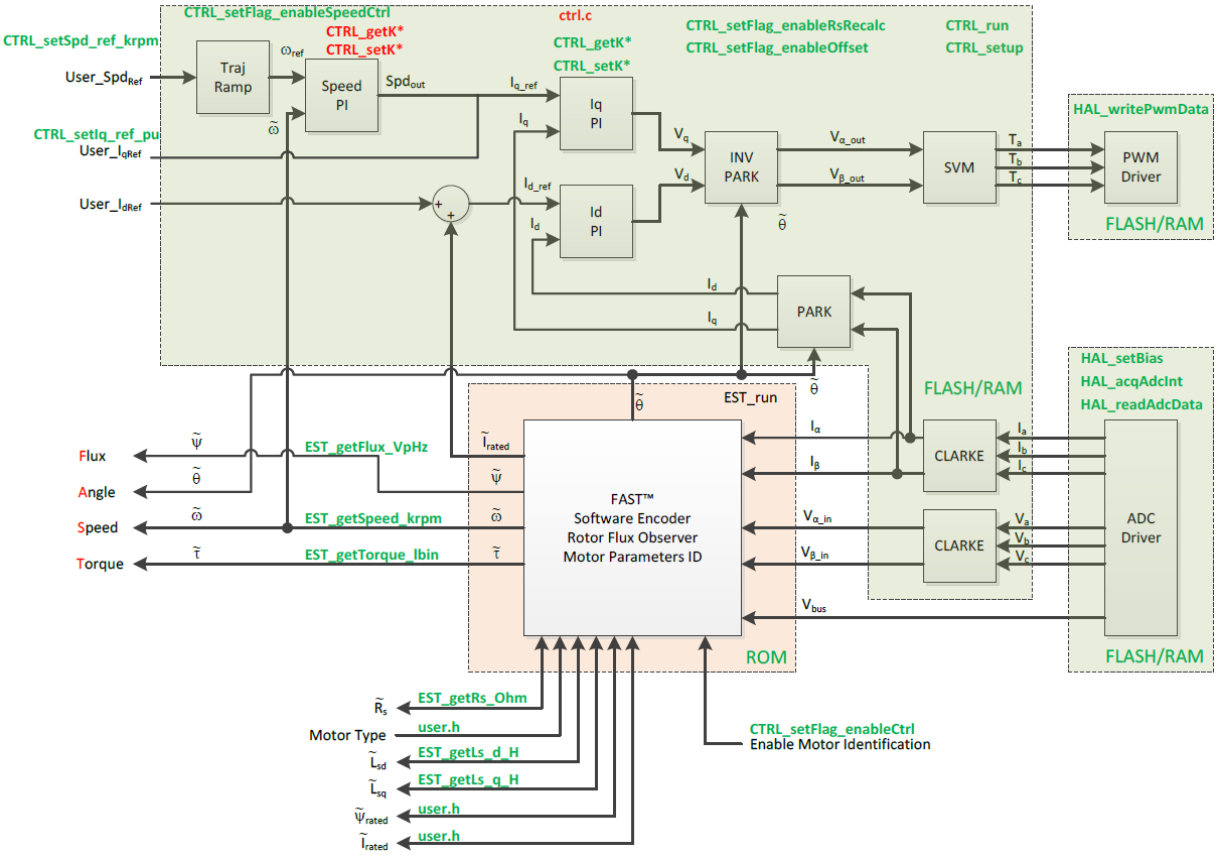


Figure 11: Software System Block Diagram

The get the speed reference from the central controller and send it into the PI controller, an ECAP module has been configured to receive a reference signal. The reference signal sets the speed command through a servo control method. It can accept a frequency from 50 to 500 Hz with a high period of 1 ms corresponding to 0 % speed and a period 2 ms corresponding to 100 % speed. The periods in between these operate with a linear speed curve.

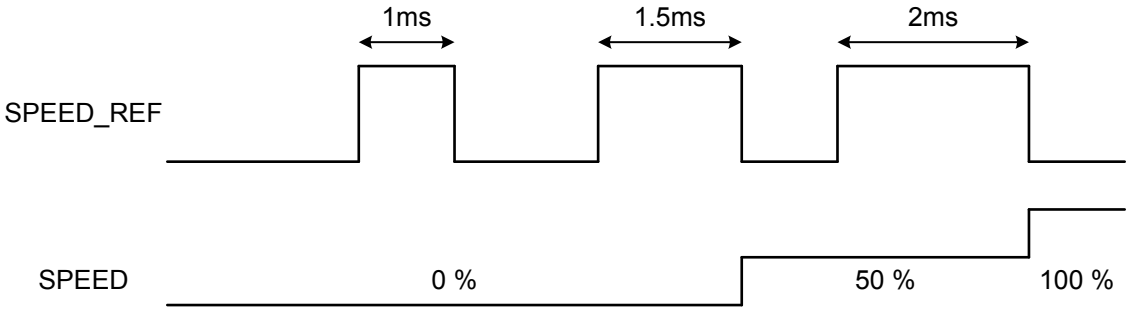


Figure 12: Speed Reference Signal

4 Getting Started Hardware

4.1 Connections

The TIDA-00643 reference design can be powered from a compatible battery or power supply from 4.4 to 30 V. The supply is connected to the motor controller through the two solder pads labeled VBAT and GND. The motor controller can be configured for most brushless DC motors through the firmware, but is targeted at high power propeller motors that are often utilized in drone and RC applications. The three motor phases can be connected to the three solder pads labeled MOTA, MOTB, and MOTC.

The speed reference signal from the central controller is connected to the motor controller through a three pin header. The header is labeled with VDD (optional 3.3V power supply), COM (speed reference signal), and GND.

The XDS100v2 JTAG connection allows for programming and debugging of the TMS320F28027F motor controller. A Tag-Connect adapter was utilized to minimize the board space require for the JTAG connector.

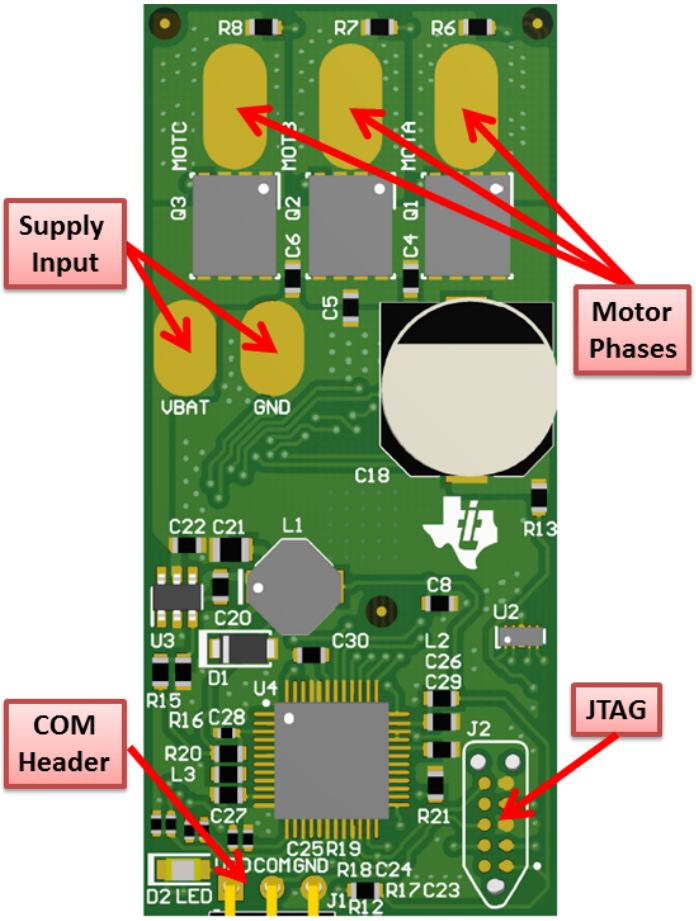


Figure 13: Connections

The XDS100v2 JTAG debugger can be sourced from ti.com at the following link

- http://www.ti.com/tool/TMDSEMU100V2U-14T

The Tag-Connect adapter and cable can be found at their site under the following part numbers.

- TC2050-IDC-NL: “10-pin, no-leg, Tag-Connect cable with ribbon connector”
- TC-C2000-M: “Male adapter for TI C2000”
- TC2050-CLIP: “Retaining clip board for use with TC2050-NL family”

The motor controller supports a wide variety of brushless motors through the MotorWare tuning process. For this design, a DJI E300 920 KV brushless motor was utilized.

4.2 Procedure

See the steps below to get started with the reference design hardware.

1. Connect the power supply or battery to the design through the VBAT and GND solder pads.
2. Connect the motor phase wires to the design through the MOTA, MOTB, and MOC solder pads.
3. Attach the JTAG debugger, enable the VBAT supply, and program the onboard MCU. The debugger can remain connected if you wish to interface to the design through JTAG.
4. Remove the debugger and send the appropriate control signal through the COM header.

5 Getting Started Firmware

The TIDA-00643 uses MotorWare for the brushless motor control and system controllers. To get started, go and download the latest version from ti.com. MotorWare walks through a series of labs to properly ID and tune the system for the specific brushless motor being utilized. This document will walk through the process with the DJI E300 920 KV motor, but can be repeated for almost any brushless motor within the capabilities of the hardware.

5.1 Adding Custom Hardware to MotorWare

The TIDA-00643 custom hardware will need to be added into MotorWare in order to begin working. There are several steps that should be completed in order to properly complete this process.

5.1.1 Create New HAL Directory

First, a new hardware abstraction layer (HAL) will need to be created.

The HAL directory is located at:

- MW_INSTALL_DIR\sw\modules\hal

It is recommended to create the new HAL directory from an existing design that is similar to the custom hardware. The boostxldrv8305_revA hardware will be used in this case. It uses the same driver (DRV8305) and controller (TMS320F28028F). Create a copy of the existing design and rename to the custom hardware.

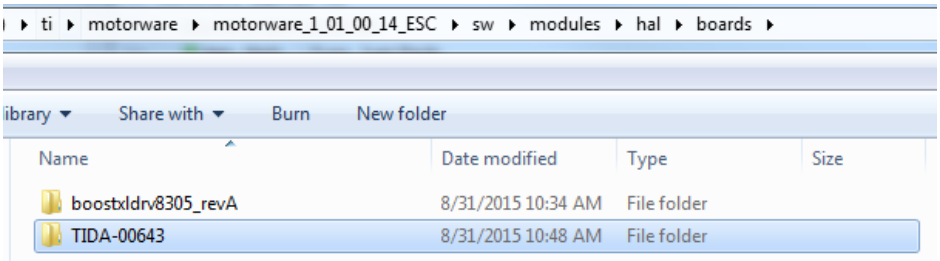


Figure 14: Custom HAL Directory

The main modification to the HAL in this design is that the SCS pin for the DRV8305 has been moved to GPIO33 and an LED has been added to GPIO34.

5.1.2 Create New PROJECT Directory

In order to operate with an independent set of labs a new PROJECT directory will need to be created.

The PROJECT directory is located at:

- MW_INSTALL_DIR\sw\solutions\instaspin_foc\boards

It is recommended to create the new PROJECT directory from an existing design that is similar to the custom hardware. This design closely matches the boostxldrv8305_revA hardware. Create a copy of the existing design and rename to the custom hardware.

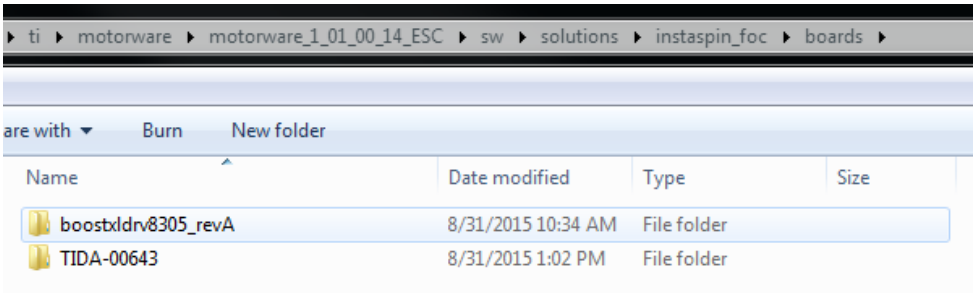


Figure 15: Custom PROJECT Directory

After creating the new PROJECT directory you will need to update the HAL link for each lab project in order to point at the new HAL.

The link can me modified in the .project file under each lab project located at:

- MW_INSTALL_DIR\sw\solutions\instaspin_foc\boards\TIDA-00643\f28x\f2802xF\projects\ccs5\proj_labXX

```
<link>
    <name>ctrl.c</name>
    <type>1</type>
    <locationURI>MW_INSTALL_DIR/sw/modules/ctrl/src/32b/ctrl.c</locationURI>
</link>
<link>
    <name>hal.c</name>
    <type>1</type>
    <locationURI>MW_INSTALL_DIR/sw/modules/hal/boards/TIDA-00643/f28x/f2802x/src/hal.c</locationURI>
</link>
<link>
```

Figure 16: .project HAL link

The include link for the HAL directory will also need to be updated and is located in the .cproject file:

- MW_INSTALL_DIR\sw\solutions\instaspin_foc\boards\TIDA-00643\f28x\f2802xF\projects\ccs5\proj_labXX

```
<tool id="com.ti.ccstudio.buildDefinitions.C2000_6.2.exe.compilerDebug.651464893" name="C2000 Compiler" superClass="com.ti.ccstudio.
    <option id="com.ti.ccstudio.buildDefinitions.C2000_6.2.compilerID.SILICON_VERSION.1955403241" name="Processor version (--silicon
    <option id="com.ti.ccstudio.buildDefinitions.C2000_6.2.compilerID.LARGE_MEMORY_MODEL.1623421441" name="Use large memory model (-
    <option id="com.ti.ccstudio.buildDefinitions.C2000_6.2.compilerID.UNIFIED_MEMORY.752513104" name="Unified memory (--unified_memc
    <option id="com.ti.ccstudio.buildDefinitions.C2000_6.2.compilerID.OPT_LEVEL.941695247" name="Optimization level (--opt_level, -O
    <option id="com.ti.ccstudio.buildDefinitions.C2000_6.2.compilerID.DEBUGGING_MODEL.1404681160" name="Debugging model" superClass=
    <option id="com.ti.ccstudio.buildDefinitions.C2000_6.2.compilerID.INCLUDE_PATH.97525854" name="Add dir to #include search path (
        <listOptionValue builtIn="false" value="&quot;${MW_INSTALL_DIR}/sw/modules/hal/boards/TIDA-00643/f28x/f2802x/src/&quot;"/>
        <listOptionValue builtIn="false" value="&quot;${CG_TOOL_ROOT}/include&quot;"/>
        <listOptionValue builtIn="false" value="&quot;${PROJECT_ROOT}/../../../src&quot;"/>
        <listOptionValue builtIn="false" value="${MW_INSTALL_DIR}"/>
```

Figure 17: .cproject INCLUDE link

At this point, you should be able to import the project into Code Composer Studio and compile successfully.

5.1.3 Update User.h File

After verifying that the new projects successfully compile in Code Composer Studio, the next step is to update the MotorWare user.h with the custom hardware parameters. The parameters to update are shown below.

```
#define USER_IQ_FULL_SCALE_VOLTAGE_V      (20.0)    // TIDA-00643
```

```
#define USER_ADC_FULL_SCALE_VOLTAGE_V     (19.73)  // TIDA-00643
#define USER_IQ_FULL_SCALE_CURRENT_A      (23.57)  // TIDA-00643
#define USER_ADC_FULL_SCALE_CURRENT_A     (47.14)  // TIDA-00643
#define USER_VOLTAGE_FILTER_POLE_Hz       (382.64) // TIDA-00643
```

5.2 MotorWare Labs

MotorWare is a cohesive set of software and technical resources designed to minimize motor control system development time. It utilizes a series of simple step by step lab projects that walk through setting up a motor control system. For this project, Lab02c, Lab3a, Lab3b, Lab4, Lab5a, and Lab5b were utilized. The next sections will briefly walk through each lab and the role they play in the TIDA-00643 brushless motor controller.

5.2.1 Lab2c: Motor ID for Low Inductance PMSM

Lab2 walks through identifying the parameters of the specific brushless DC motor that will be utilized in the system. For the TIDA-00643 motor controller, this will be the DJI E300 920 KV brushless DC motor. Due to the design of high-speed drone propeller motors, the winding inductance is often very low compared to typical brushless DC motors. Lab2c utilizes some additional functions to account for these types of motors.

The first step is to create motor profile in the User.h file. The DJI_E300 motor has 7 pole pairs. To account for the lower winding inductance the USER_MOTOR_FLUX_EST_PREQ_Hz has been increased to 100.0.

```
#define DJI_E300                    119
```

```
#define USER_MOTOR DJI_E300
```

```
#elif (USER_MOTOR == DJI_E300)
#define USER_MOTOR_TYPE                 MOTOR_Type_Pm
#define USER_MOTOR_NUM_POLE_PAIRS       (7)
#define USER_MOTOR_Rr                   (NULL)
#define USER_MOTOR_Rs                   (0.08385667)
#define USER_MOTOR_Ls_d                 (1.002232e-05)
#define USER_MOTOR_Ls_q                 (1.002232e-05)
#define USER_MOTOR_RATED_FLUX           (0.005794205)
#define USER_MOTOR_MAGNETIZING_CURRENT  (NULL)
#define USER_MOTOR_RES_EST_CURRENT      (1.5)
#define USER_MOTOR_IND_EST_CURRENT      (-1.5)
#define USER_MOTOR_MAX_CURRENT          (15.0)
#define USER_MOTOR_FLUX_EST_FREQ_Hz     (100.0)
```

5.2.2 Lab3a: Using Motor Parameters without Offsets

After running lab2c, you should now have the motor parameters. The user.h file can be updated with these to remember the motor. Lab3a shows how to run the motor with the parameters from Lab2c in a basic control loop.

In our setup, the motor actually runs quite poorly. This is related to the default parameters being utilized for the control loops. These will be optimized in the next labs.

5.2.3 Lab3b: Using Motor Parameters with Offsets

Lab3b shows how to save the offsets for the specific hardware that is being utilized and then run the motor with them. The offsets from the TIDA-00643 motor controller are shown below. The voltage and current offsets will vary from board to board and for best results should be updated for every hardware set.

```
//! \brief ADC current offsets for A, B, and C phases
#define   I_A_offset (1.019250691)  // TIDA-00643
#define   I_B_offset (1.022722363)  // TIDA-00643
#define   I_C_offset (1.021086812)  // TIDA-00643

//! \brief ADC voltage offsets for A, B, and C phases
#define   V_A_offset (0.2965255976) // TIDA-00643
#define   V_B_offset (0.2964715958) // TIDA-00643
#define   V_C_offset (0.2963429689) // TIDA-00643
```

5.2.4 Lab4: Torque Mode

Since by its nature field oriented control is a current control method, Lab4 walks through spinning the motor with only a torque controller.

5.2.5 Lab5a: Torque Mode and Tuning Id/Iq PI

Lab5a deals with tuning the torque controller. Since InstaSPIN-FOC utilizes a speed controller wrapped around a torque controller it is critical that the torque controller be properly tuned. For this system, the defaults calculated were sufficient.

5.2.6 Lab5b: Speed Mode and Tuning Speed PI

The last lab utilized, Lab5b, deals with tuning the speed controller. This lab will also be utilized for the final version of the motor controller control system. After determining the correct Kp_spd and Ki_spd, they can be programmed to load by default.

```
// initialize the watch window kp and ki values with pre-calculated values
gMotorVars.Kp_spd = CTRL_getKp(ctrlHandle,CTRL_Type_PID_spd);
gMotorVars.Ki_spd = CTRL_getKi(ctrlHandle,CTRL_Type_PID_spd);

// TIDA-00643 custom values for speed controller
gMotorVars.Kp_spd = _IQ(2.000);
gMotorVars.Ki_spd = _IQ(0.059);
```

Since InstaSPIN-FOC is a sinewave modulation technique it should be noted that the peak output is limited to approximately 77% of the full bus voltage. To achieve the full velocity range of the motor an over-modulation technique may be used by following lab10. This is not covered in this design.

5.3 eCAP Speed Reference Receiver

In order for the motor controller to receive a speed reference from the central controller, a front end receiver is required. This receiver can be implemented with the eCAP module of the TMS320F28027F. The eCAP module receives a variable duty cycle PWM from the central controller and translates it to a speed command for the PI controller.

5.3.1 Pin Configurations

In order to utilize the eCAP module, the pin must be correctly configured. This is done in the hal.c file.

```
// ECAP
GPIO_setMode(obj->gpioHandle, GPIO_Number_19, GPIO_19_Mode_ECAP1);
```

It is also required to update the SPI SCS pin to use the GPIO resource.

```
// SPI_SCS
GPIO_setMode(obj->gpioHandle,GPIO_Number_33,GPIO_33_Mode_GeneralPurpose);
GPIO_setDirection(obj->gpioHandle, GPIO_Number_33, GPIO_Direction_Output);
GPIO_setHigh(obj->gpioHandle, GPIO_Number_33);
```

Since the eCAP is utilizing the dedicated SPIS CS resource from the C2000 it is required to make a few slight modifications to the existing SPI routines in the firmware. These changes are made in the drv8305.c file.

DRV8305_writeSpi function:

```
// Added: SCS High->Low
GPIO_setLow(obj->gpioHandle, GPIO_Number_33);

// write the command
SPI_write(obj->spiHandle,ctrlWord);

// wait for registers to update
for(n=0;n<0xf;n++)
  asm(" NOP");

// Added: SCS Low->High
GPIO_setHigh(obj->gpioHandle, GPIO_Number_33);
```

DRV8305_readSpi function:

```
  // Added: SCS High->Low
  GPIO_setLow(obj->gpioHandle, GPIO_Number_33);

  // write the command
  SPI_write(obj->spiHandle,ctrlWord);

  // wait for the response to populate the RX fifo, else a wait timeout will
occur
  while((RxFifoCnt < SPI_FifoStatus_1_Word) && (WaitTimeOut < 0xffff))
  {
    RxFifoCnt = SPI_getRxFifoStatus(obj->spiHandle);
      if(++WaitTimeOut > 0xfffe)
      {
            obj->RxTimeOut = true;
      }
  }

  // Added: SCS Low->High
  GPIO_setHigh(obj->gpioHandle, GPIO_Number_33);
```

5.3.2 eCAP Handle

In MotorWare, peripherals are accessed through the Hardware Abstraction Layer (HAL). The peripherals are accessed through their dedicated handle. In order to utilize the eCAP it must also be added to the HAL.

The first step is to add the eCAP library into the MotorWare project. The link to the library can be added in the .project file under the lab5b project located at:

- MW_INSTALL_DIR/sw/solutions/instaspin_foc/boards/TIDA-00643/f28x/f2802xF/projects/ccs5/proj_lab5b

```
<link>
    <name>adc.c</name>
    <type>1</type>
    <locationURI>MW_INSTALL_DIR/sw/drivers/adc/src/32b/f28x/f2802x/adc.c</locationURI>
</link>
<link>
    <name>cap.c</name>
    <type>1</type>
    <locationURI>MW_INSTALL_DIR/sw/drivers/cap/src/32b/f28x/f2802x/cap.c</locationURI>
</link>
<link>
```

Figure 18: eCAP Library Link

Add the cap library to the hal_obj.h file.

```
#include "sw/drivers/adc/src/32b/f28x/f2802x/adc.h"
#include "sw/drivers/cap/src/32b/f28x/f2802x/cap.h"
#include "sw/drivers/clk/src/32b/f28x/f2802x/clk.h"
```

Add the cap handle to the _HAL_Obj_ in the hal_obj.h file.

```
typedef struct _HAL_Obj_
{
  ADC_Handle    adcHandle;        //!< the ADC handle

  CAP_Handle    capHandle;        //!< the CAP handle
```

Instantiate the handle with the other peripherals handles.

```
// initialize the ADC
obj->adcHandle = ADC_init((void *)ADC_BASE_ADDR,sizeof(ADC_Obj));

// initialize the eCAP
obj->capHandle = CAP_init((void *)CAP1_BASE_ADDR,sizeof(CAP_Obj));

// initialize the clock handle
obj->clkHandle = CLK_init((void *)CLK_BASE_ADDR,sizeof(CLK_Obj));
```

5.3.3 eCAP ISR

After creating the eCAP handle, the next step is to create the interrupt service routine (ISR) that will process the speed reference signal. First, create a setup function for the eCAP ISR. You will add the declaration to the hal.h file.

```
//! \brief Setup the ECAP
//! \param[in] handle         The hardware abstraction layer (HAL) handle
void HAL_setupeCAP(HAL_Handle handle);
```

Then add the definition into the hal.c. The eCAP has been setup to measure the time between a rising and falling edge.

```
// ECAP
void HAL_setupeCAP(HAL_Handle handle)
{
    HAL_Obj *obj = (HAL_Obj *) handle;

    CAP_setModeCap(obj->capHandle); // set mode to CAP

    //Disables counter synchronization
    CAP_disableSyncIn(obj->capHandle);

    //Sets the capture event polarity
    CAP_setCapEvtPolarity(obj->capHandle, CAP_Event_1, CAP_Polarity_Rising);

    //Sets the capture event polarity
    CAP_setCapEvtPolarity(obj->capHandle, CAP_Event_2, CAP_Polarity_Falling);

    //Sets the capture event counter reset configuration
    CAP_setCapEvtReset(obj->capHandle, CAP_Event_1, CAP_Reset_Disable);

    //Sets the capture event counter reset configuration (reset counting here)
    CAP_setCapEvtReset(obj->capHandle, CAP_Event_2, CAP_Reset_Enable);

    // continous timer
    CAP_setCapContinuous(obj->capHandle);

    //Set the stop/wrap mode to 2 events
    CAP_setStopWrap(obj->capHandle, CAP_Stop_Wrap_CEVT2);

    //Enables loading of CAP1-4 on capture event
    CAP_enableCaptureLoad(obj->capHandle);

    // Enables Time Stamp counter to running
    CAP_enableTimestampCounter(obj->capHandle);

    //Enables capture (CAP) interrupt source
    CAP_enableInt(obj->capHandle, CAP_Int_Type_CEVT2);

    // enable eCAP interrupt
    PIE_enableInt(obj->pieHandle, PIE_GroupNumber_4, PIE_InterruptSource_ECAP1);

    // enable CPU ECAP Group interrupts
    CPU_enableInt(obj->cpuHandle, CPU_IntNumber_4);

    return;
} // end of HAL_setupCAP() function
```

Lastly, call the setupeCAP function in the hal.c HAL_setParams() function.

Ensure that the eCAP clock has been setup in the hal.c

.

```
void HAL_setupPeripheralClks(HAL_Handle handle)
{
  HAL_Obj *obj = (HAL_Obj *)handle;


  CLK_enableAdcClock(obj->clkHandle);

  CLK_enableCompClock(obj->clkHandle,CLK_CompNumber_1);
  CLK_enableCompClock(obj->clkHandle,CLK_CompNumber_2);
  CLK_enableCompClock(obj->clkHandle,CLK_CompNumber_3);

  CLK_enableEcap1Clock(obj->clkHandle);
```

After creating the setup function, the ECAP ISR must be declared in the hal.h.

```
// the globals


extern interrupt void mainISR(void);

// ECAP
extern interrupt void ecapISR(void);
```

Then it is added to the interrupt vector table in order to be properly addressed and pointed to. The pie is a handle found within the HAL handle object and it is the C2000's Peripheral Interrupt Expansion (PIE) block that supports all peripheral interrupts.

```
static inline void HAL_initIntVectorTable(HAL_Handle handle)
 {
  HAL_Obj *obj = (HAL_Obj *)handle;
  PIE_Obj *pie = (PIE_Obj *)obj->pieHandle;


  ENABLE_PROTECTED_REGISTER_WRITE_MODE;

  pie->ADCINT1 = &mainISR;

    // ECAP
  pie->ECAP1_INT = &ecapISR;

  DISABLE_PROTECTED_REGISTER_WRITE_MODE;
```

With the eCAP ISR created, the last step is to define the actual eCAP ISR. This is added to the proj_lab05b.c file which contains the main() function. The eCAP ISR clears the interrupt, calculates the period the speed reference signal is high, and then calculates the appropriate speed command for the motor.

```
__interrupt void ecapISR(void)
{
    // Clear capture (CAP) interrupt flags
    CAP_clearInt(halHandle->capHandle, CAP_Int_Type_All);

    // Compute the PWM high period (rising edge to falling edge)
    uint32_t PwmDuty = (uint32_t) CAP_getCap2(halHandle->capHandle) - (uint32_t)
CAP_getCap1(halHandle->capHandle);

    // Assign the appropriate speed command, combine 0-5% and 95-100%
    // 0-100% speed is proportional to 1-2ms high period
    // 60MHz * 2ms = 120000

    // 0-1%
    if (PwmDuty <= 61000)
    {
        gSpeedRef_Ok = 0;
        gSpeedRef_duty = _IQ(0);
        gMotorVars.Flag_Run_Identify = 0;
    }
    // 1-99%
    if ((PwmDuty > 61000) && (PwmDuty < 119000))
    {
        gSpeedRef_Ok = 0;
        gSpeedRef_duty = _IQdiv(PwmDuty - 60000, 60000);
        gMotorVars.Flag_Run_Identify = 1;
    }
    // 99-100%
    else if (PwmDuty >= 119000)
    {
        gSpeedRef_Ok = 0;
        gSpeedRef_duty = _IQ(1.0);
        gMotorVars.Flag_Run_Identify = 1;
    }
    // Catch all
    else
    {
        gSpeedRef_duty = _IQ(0);
        gMotorVars.Flag_Run_Identify = 0;
    }

    // Clears an interrupt defined by group number
    PIE_clearInt(halHandle->pieHandle, PIE_GroupNumber_4);
}  // end of ecapISR() function
```

The last step is to feed the motor speed command into the motor controller. This is done in the main ISR in the updateGlobalVariables_motor() function.

```
void updateGlobalVariables_motor(CTRL_Handle handle)
{
  CTRL_Obj *obj = (CTRL_Obj *)handle;

  // get the speed estimate
  gMotorVars.Speed_krpm = EST_getSpeed_krpm(obj->estHandle);

  // set motor speed command dependent
  gMotorVars.SpeedRef_krpm = gSpeedRef_duty * SPEED_BASE_KRPM;
```

A timeout function has also been added to prevent losing control of the motor if the speed reference signal is lost. In the main ISR for the motor control, a timer has been added for the speed reference signal. Every time a new speed reference signal is received, the timer is reset. If the timer times out (signal is lost) then the motor is disabled.

```
interrupt void mainISR(void)
{
  // toggle status LED
  if(gLEDcnt++ > (uint_least32_t)(USER_ISR_FREQ_Hz / LED_BLINK_FREQ_Hz))
  {
    HAL_toggleLed(halHandle,(GPIO_Number_e)HAL_Gpio_LED2);
    gLEDcnt = 0;
  }

  // Check if speed reference signal is active
  // If more than 2000 service routine cycles pass without signal, disable motor
  if (gSpeedRef_Ok++ > 2000)
  {
      gSpeedRef_duty = _IQ(0);
      gMotorVars.Flag_Run_Identify = 0;
      gSpeedRef_Ok = 0;
  }
```

6 Test Setup

Equipment	Name
DC Power Supply	Chroma 620012P-100-50
Multimeter	Tektronix DMM 4040
Oscilloscope	Tektronix DPO 7054
Function Generator	Agilent 33522A
Thermometer	Fluke 54

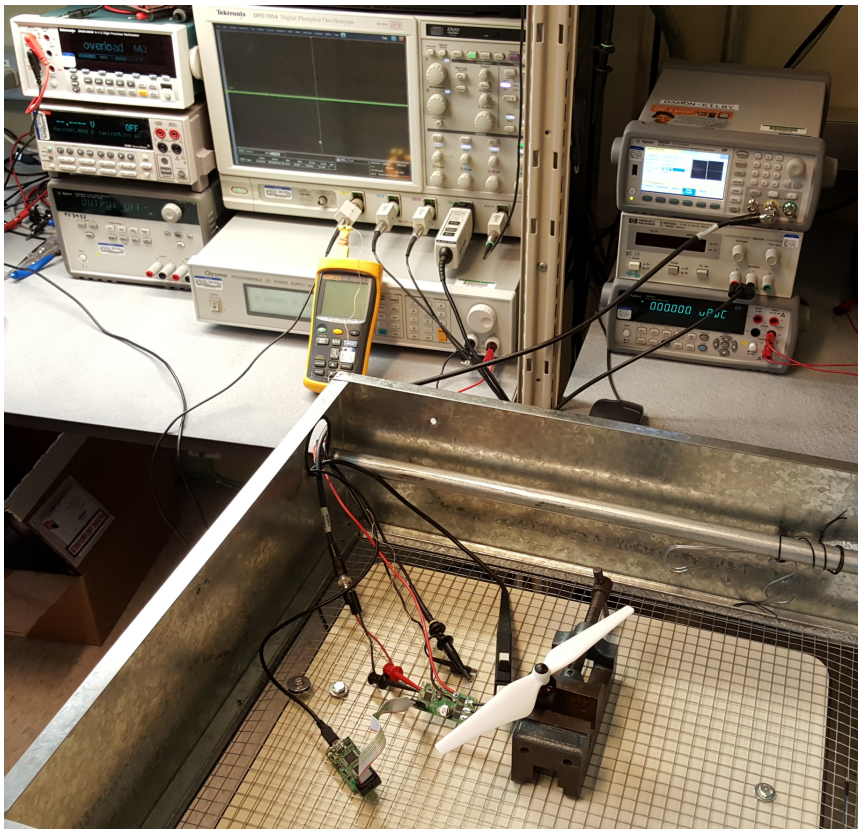


Figure 19: Test Setup

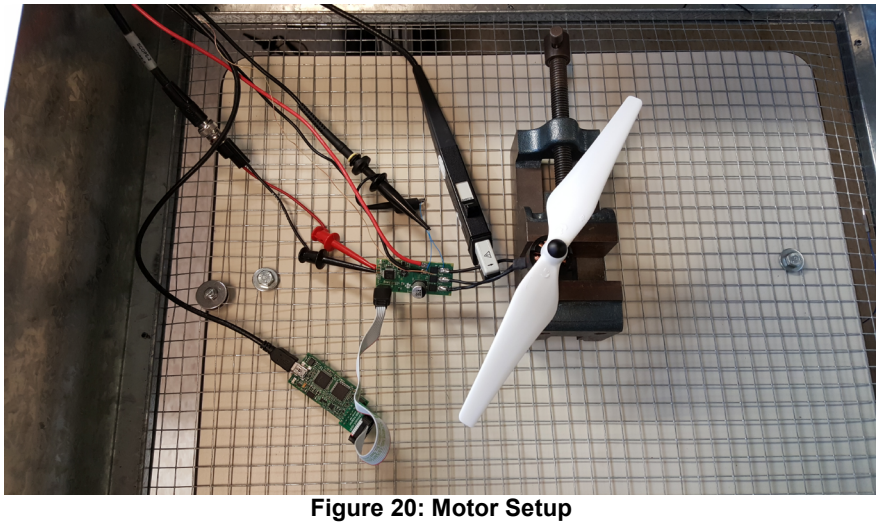


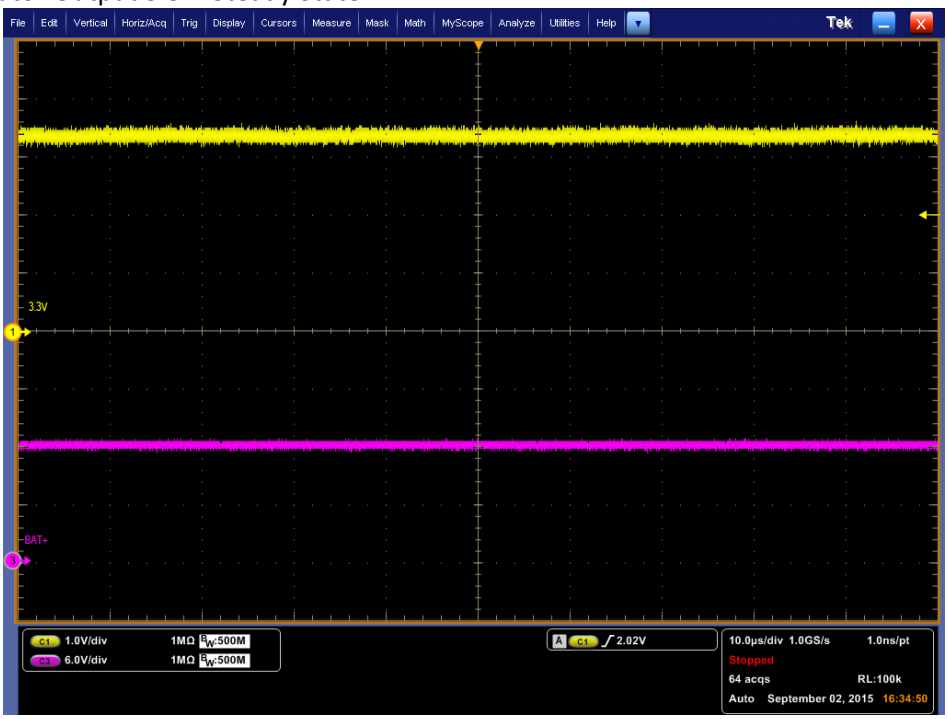
Figure 20: Motor Setup

7 Test Data

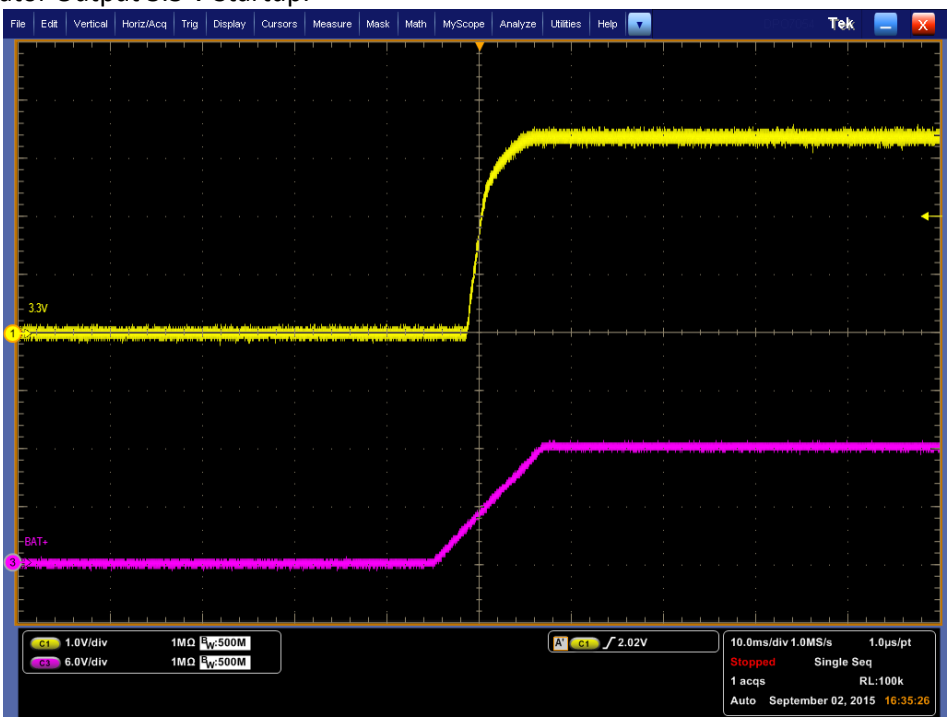
7.1 Functional Tests

7.1.1 Buck Regulator

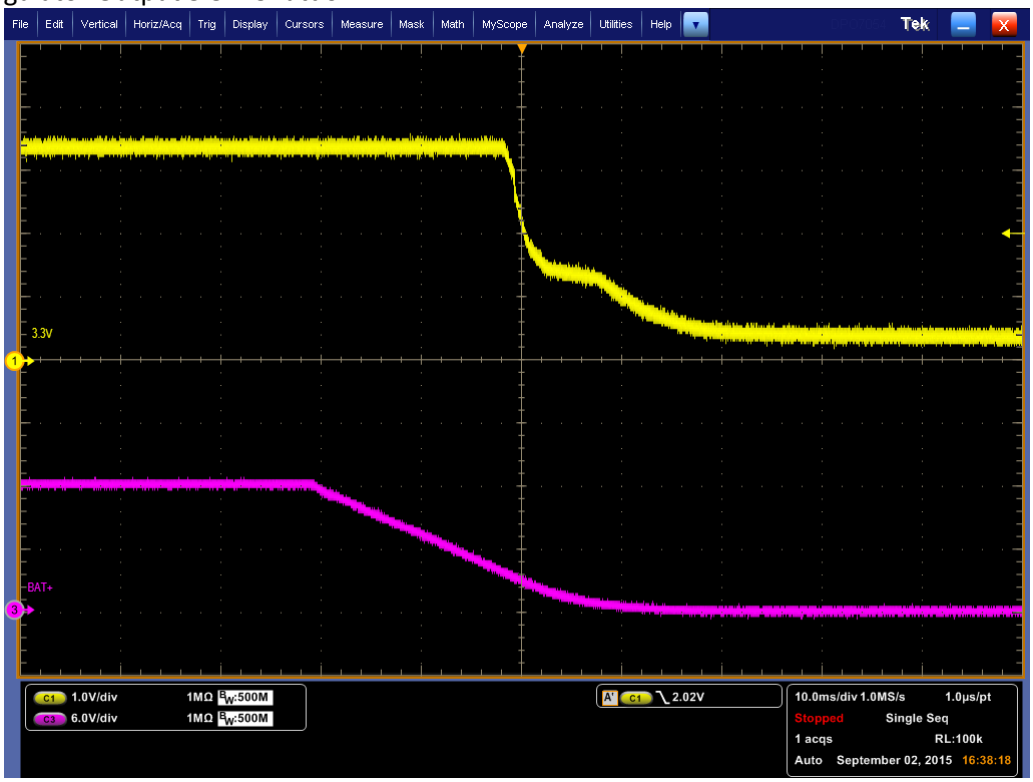
Test to examine the buck regulator input (VBAT) vs. output (3.3 V).
Buck Regulator Output 3.3 V Steady State:



Buck Regulator Output 3.3 V Startup:



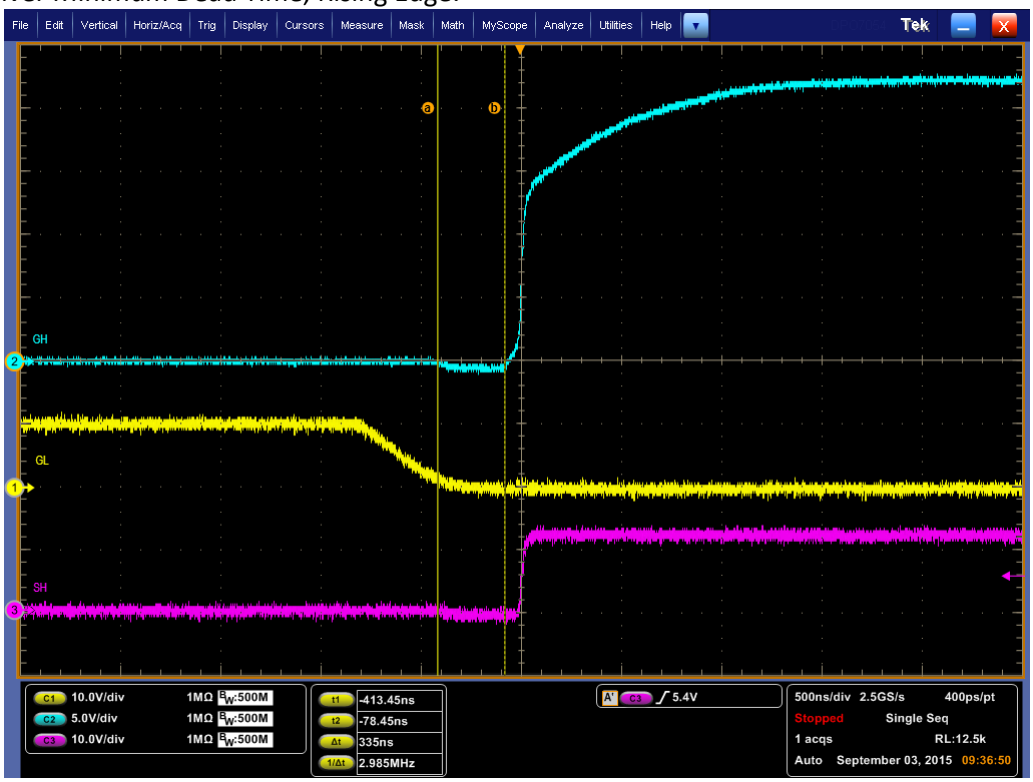
Buck Regulator Output 3.3 V Shutdown:



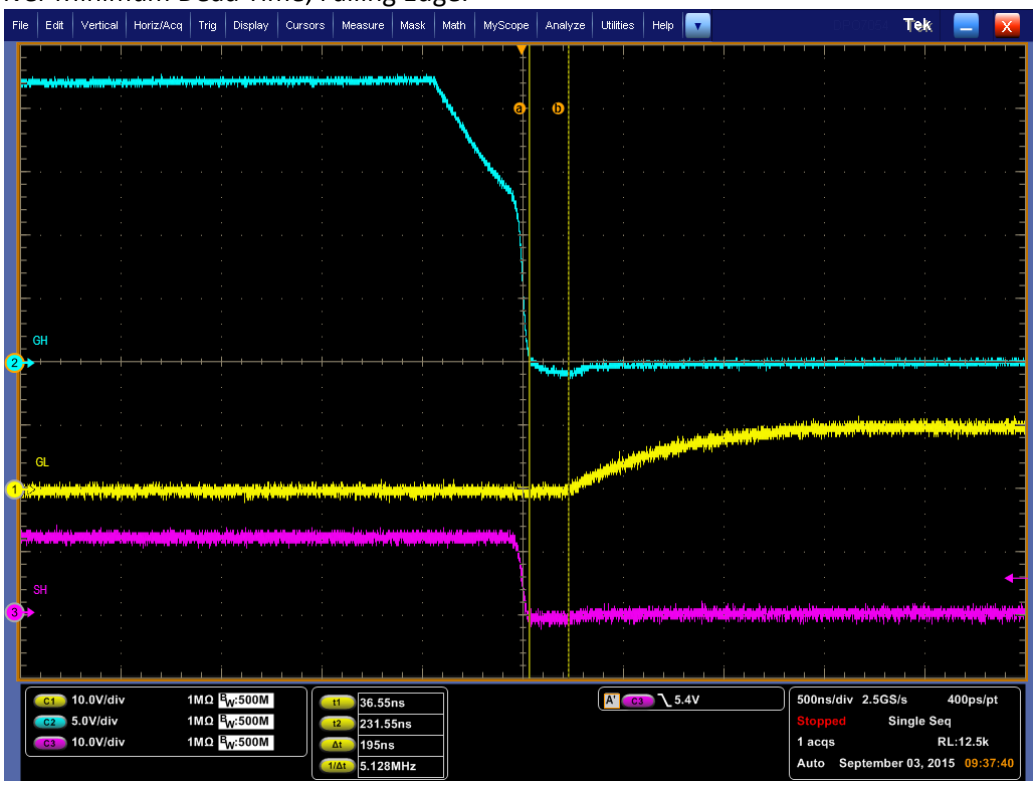
7.1.2 Gate Driver Dead Time

Test to examine the gate driver outputs (GL, GH, and SH) dead time. Dead time is configurable through the DRV8305.

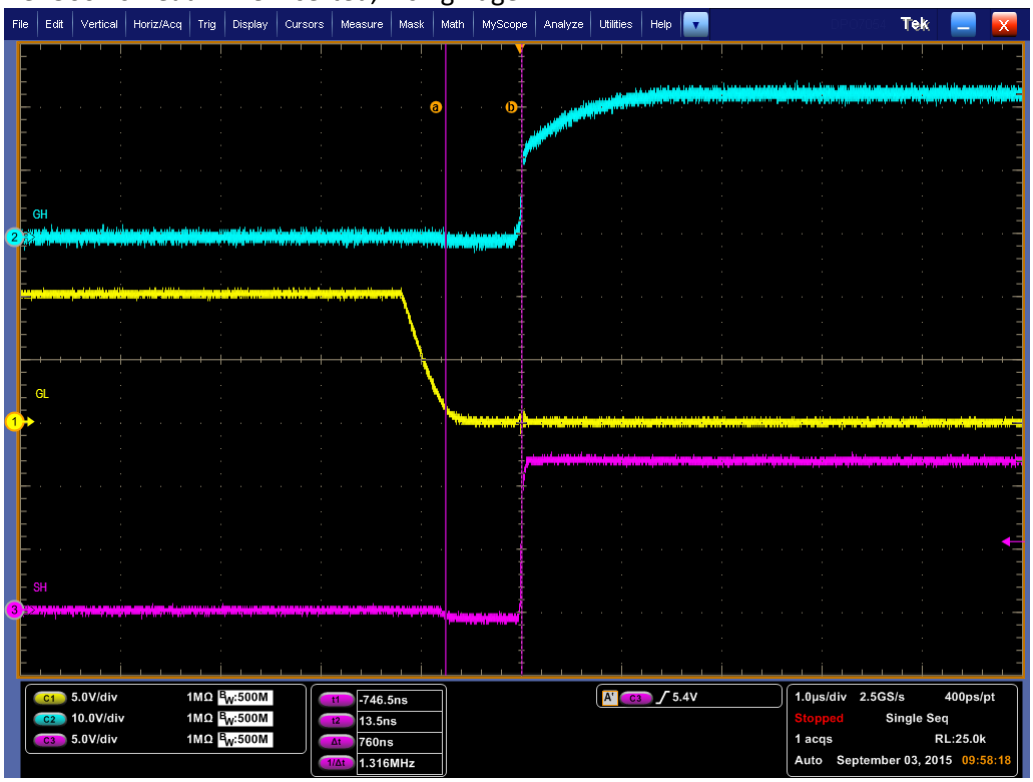
Gate Driver Minimum Dead Time, Rising Edge:



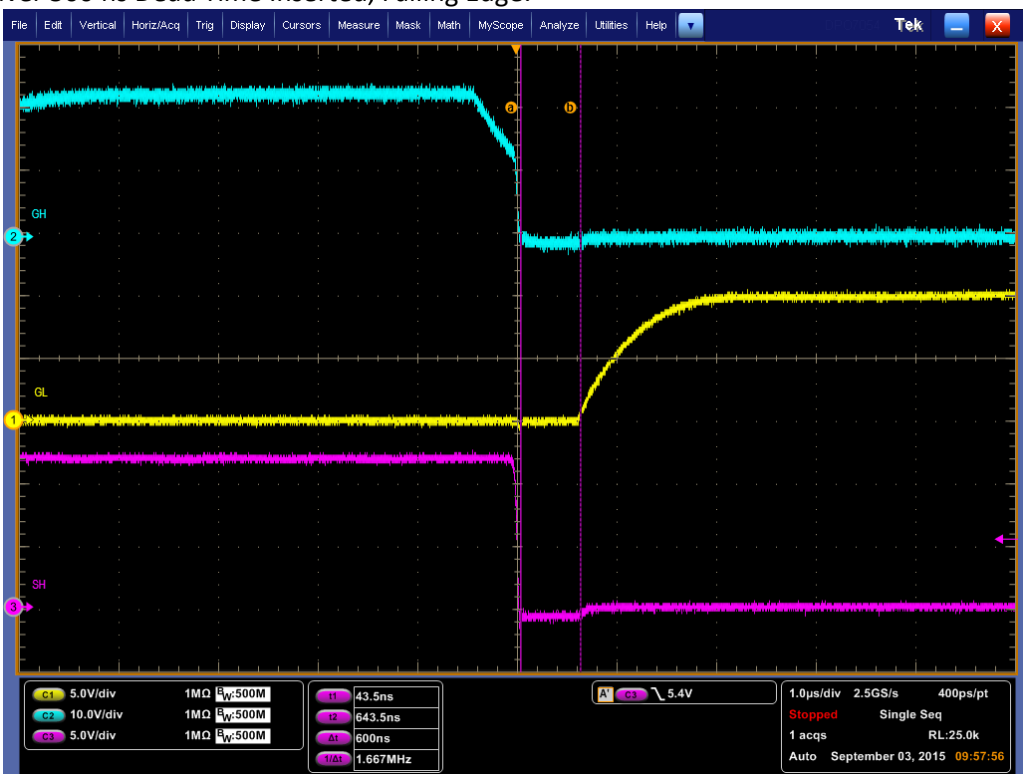
Gate Driver Minimum Dead Time, Falling Edge:



Gate Driver 500 ns Dead Time Inserted, Rising Edge:



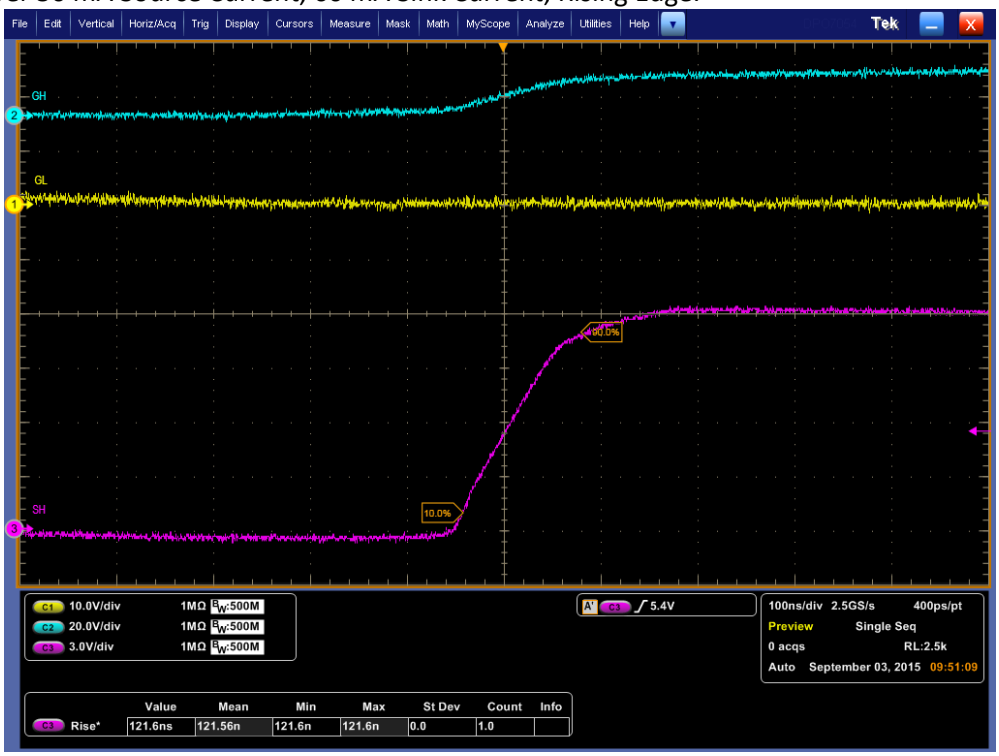
Gate Driver 500 ns Dead Time Inserted, Falling Edge:



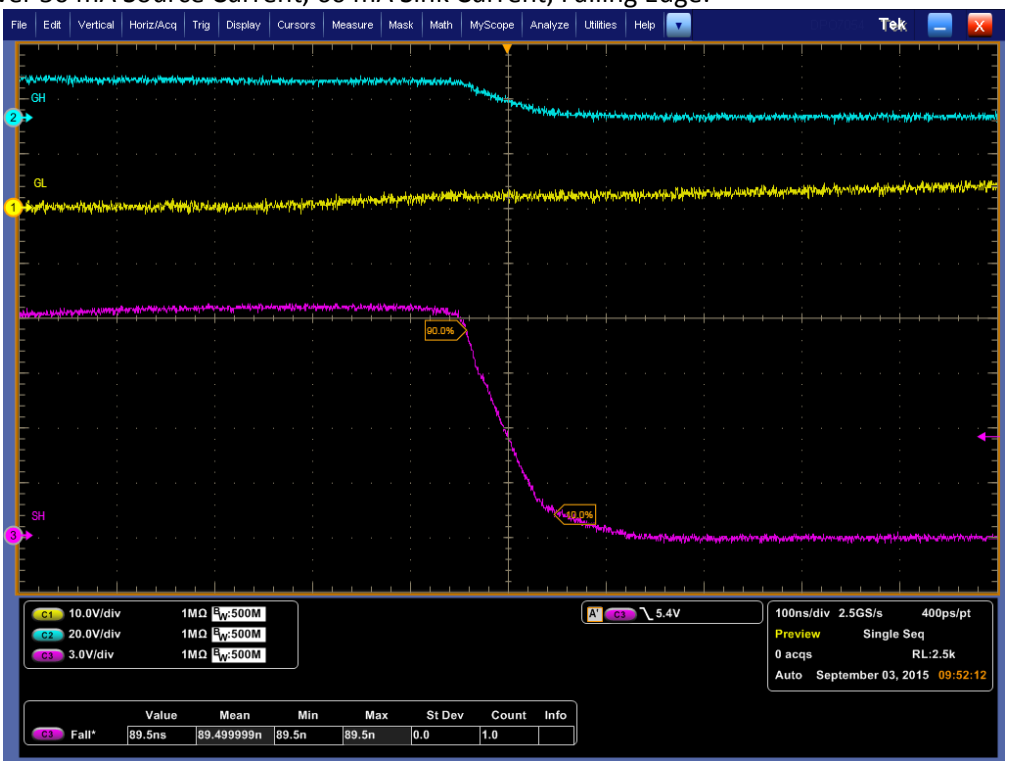
7.1.3 Gate Driver Current Levels

Test to examine the gate driver outputs (GL, GH, and SH) drive strength. Drive strength is configurable through the DRV8305.

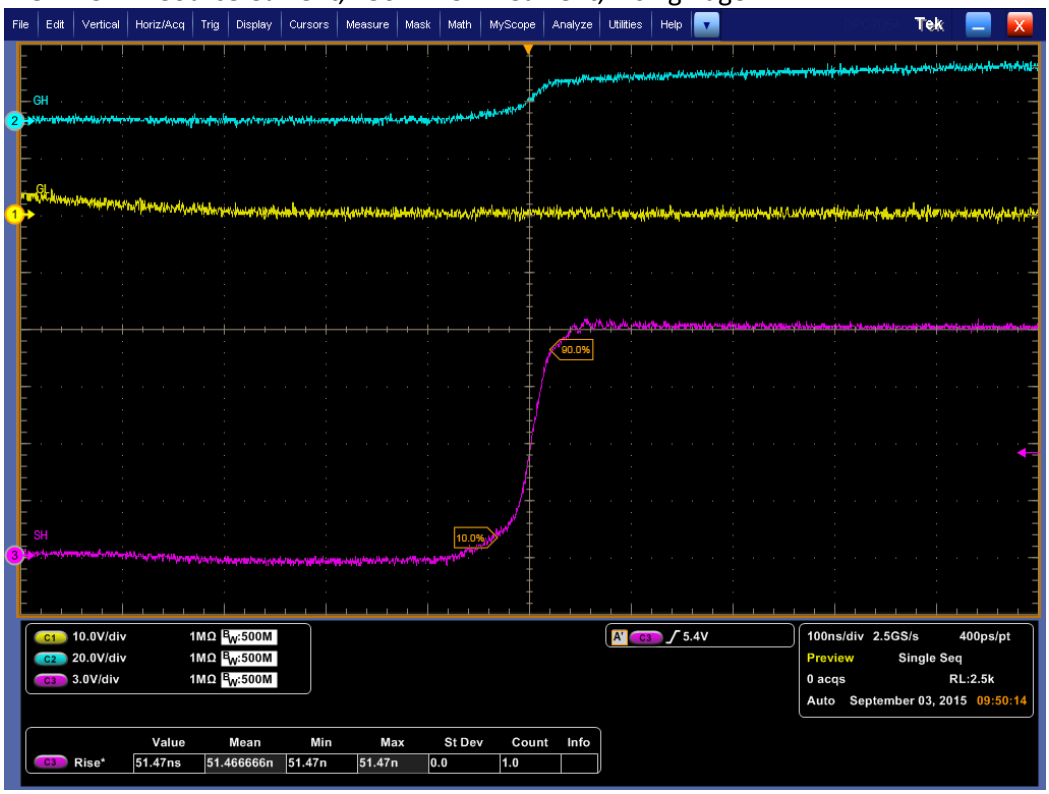
Gate Driver 50 mA Source Current, 60 mA Sink Current, Rising Edge:



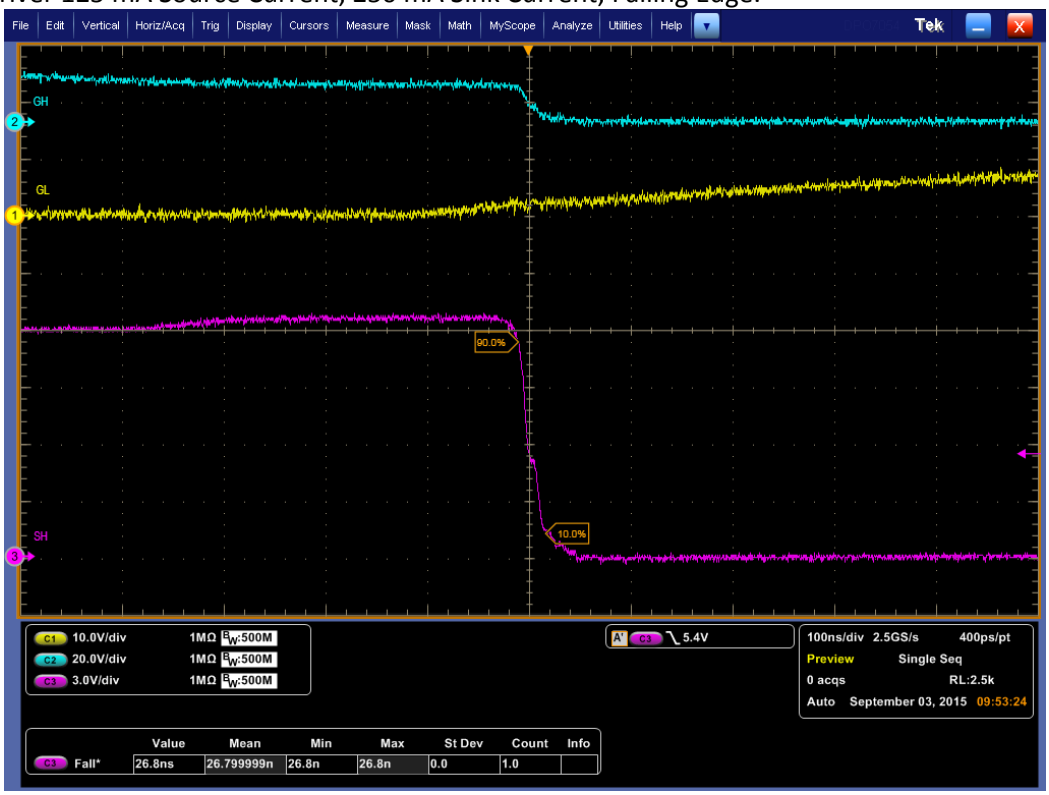
Gate Driver 50 mA Source Current, 60 mA Sink Current, Falling Edge:



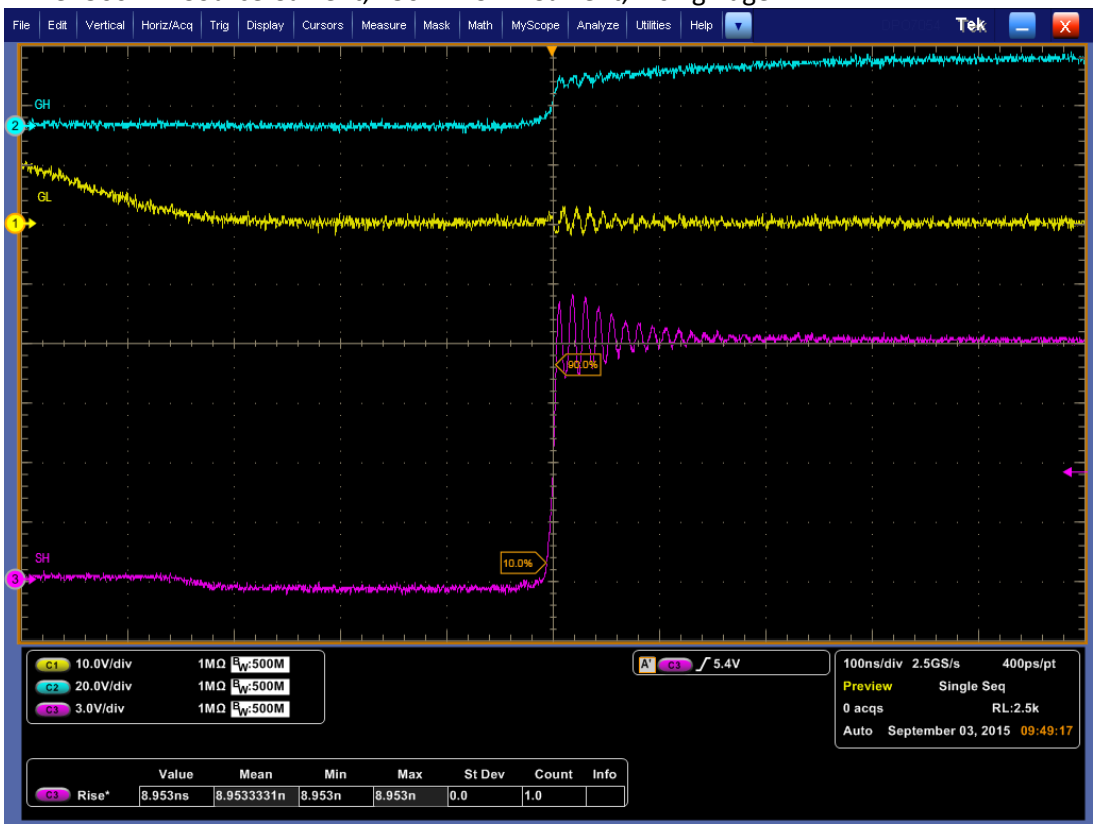
Gate Driver 125 mA Source Current, 250 mA Sink Current, Rising Edge:



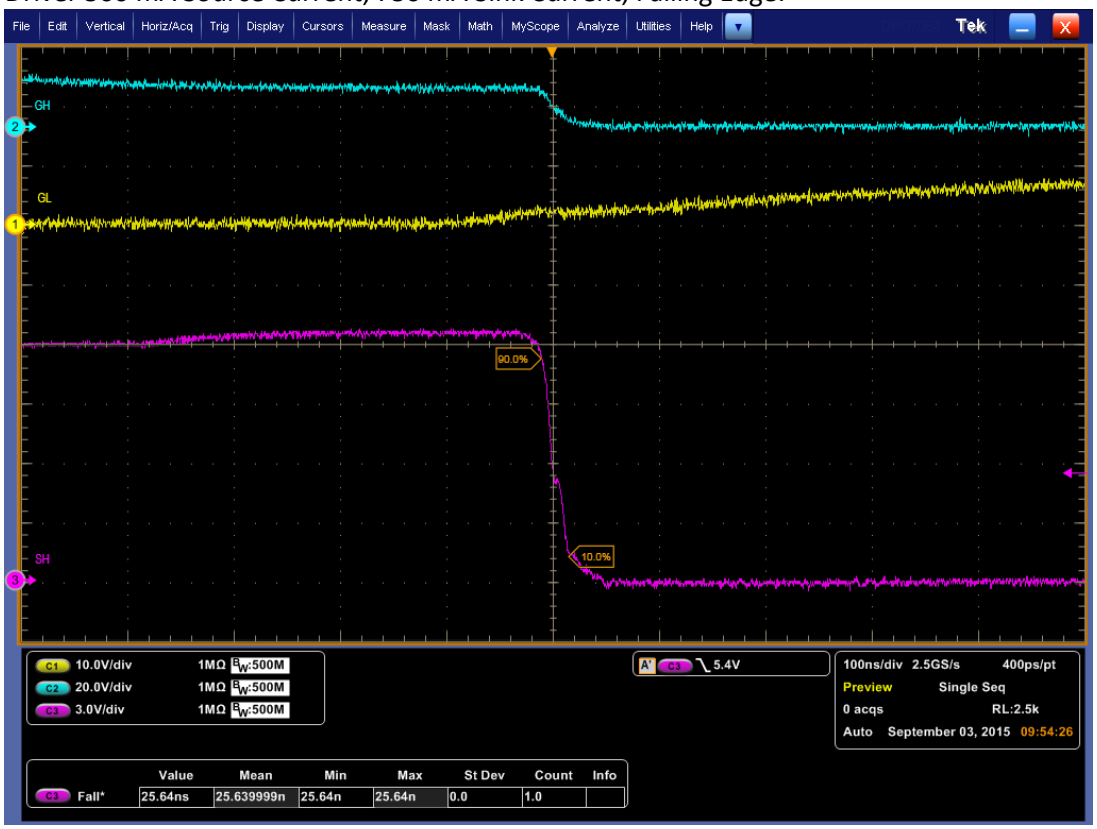
Gate Driver 125 mA Source Current, 250 mA Sink Current, Falling Edge:



Gate Driver 500 mA Source Current, 750 mA Sink Current, Rising Edge:



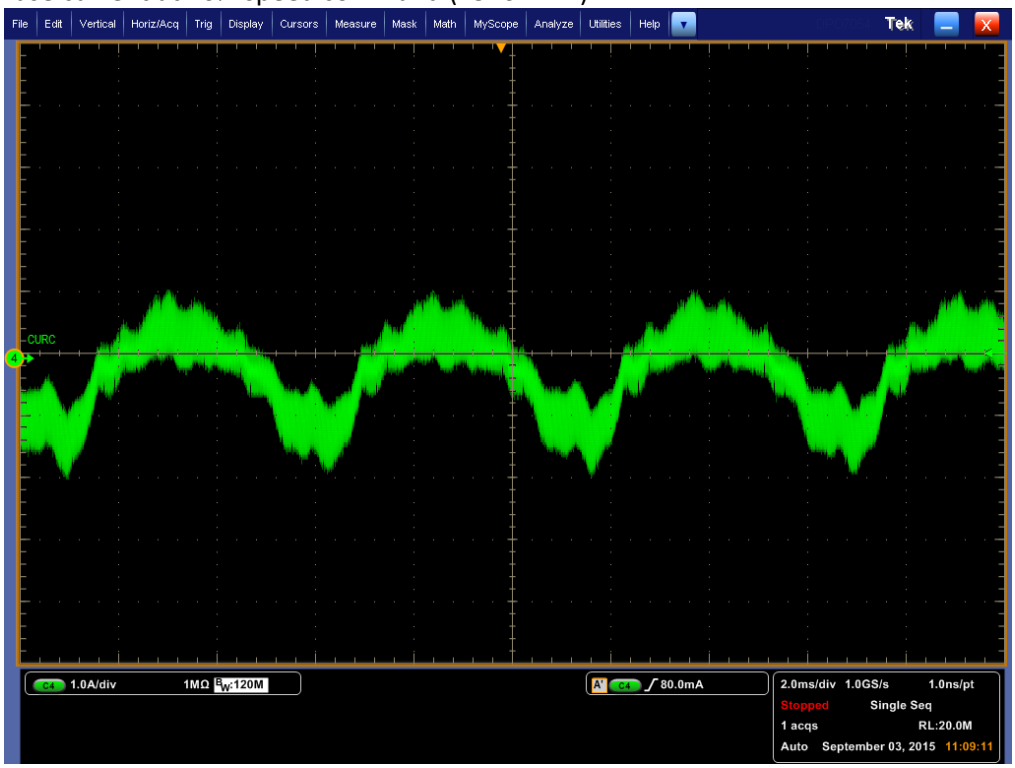
Gate Driver 500 mA Source Current, 750 mA Sink Current, Falling Edge:



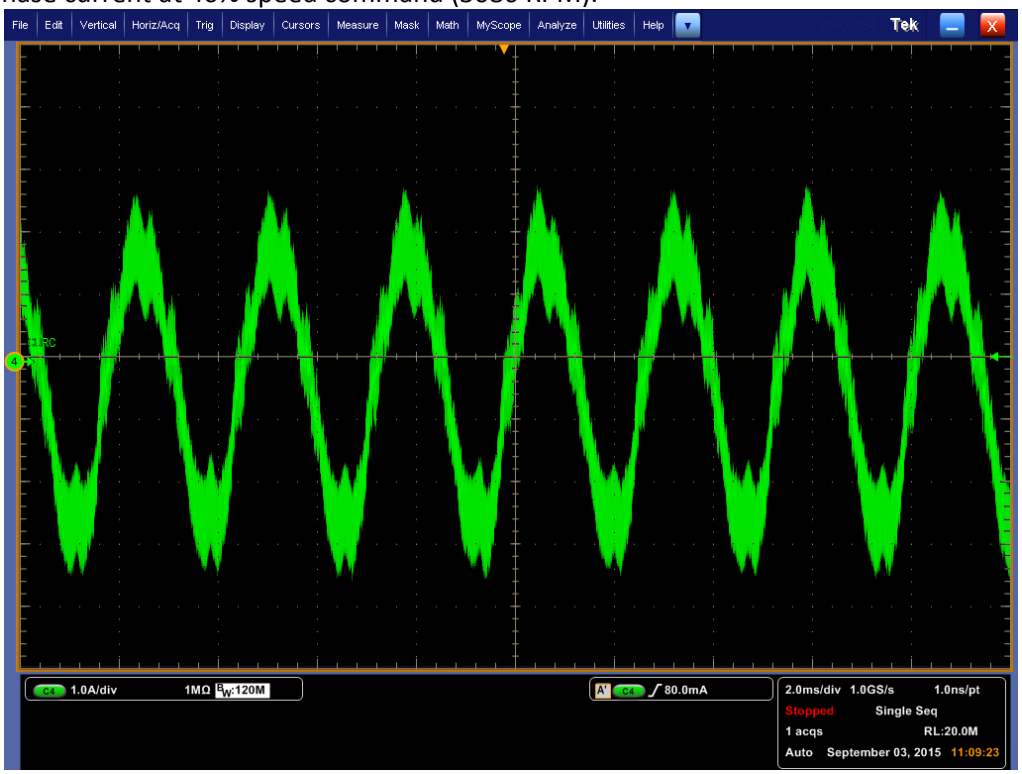
7.2 Load Tests

7.2.1 Motor Phase Current

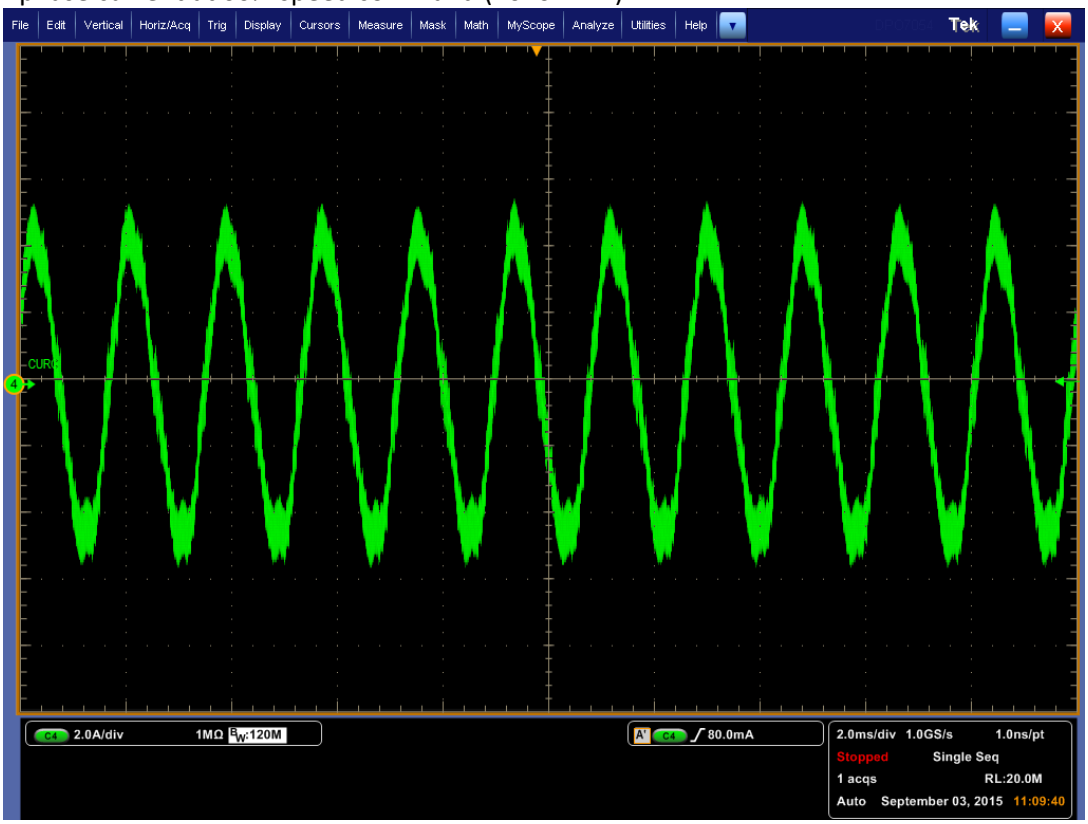
Test to examine the motor phase current vs. motor speed command.
Motor phase current at 20% speed command (1540 RPM).



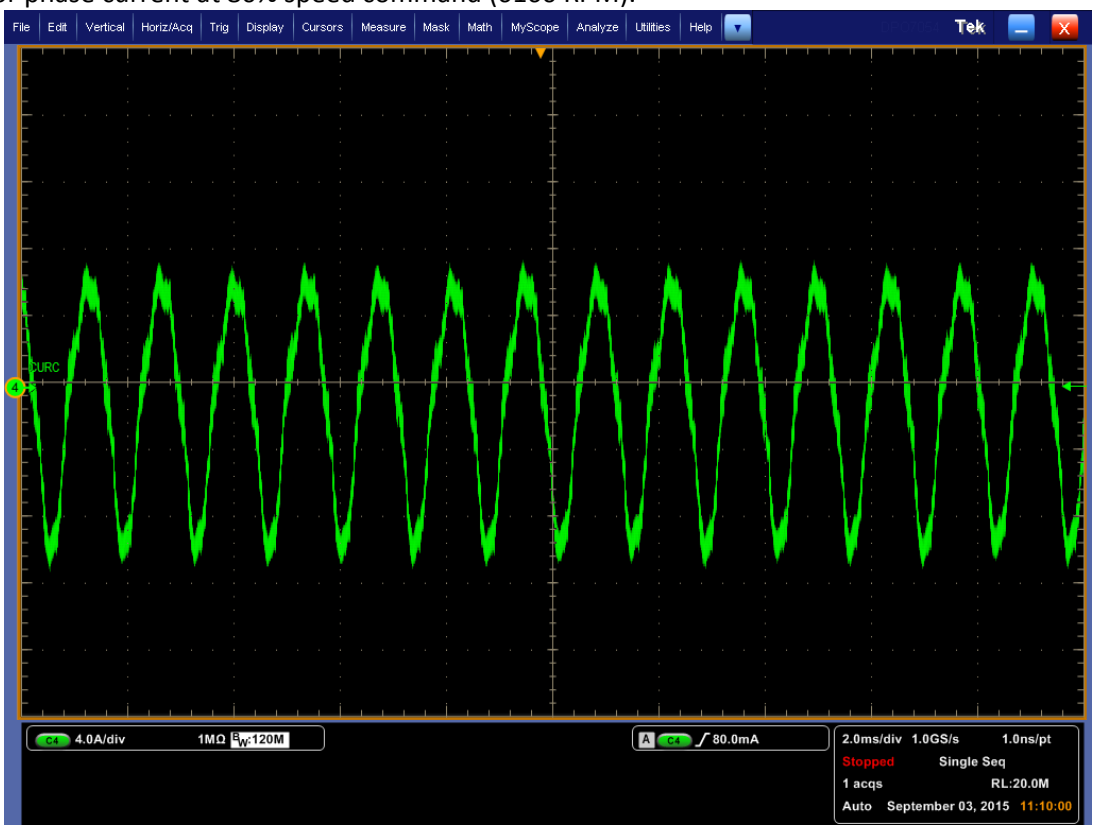
Motor phase current at 40% speed command (3080 RPM).



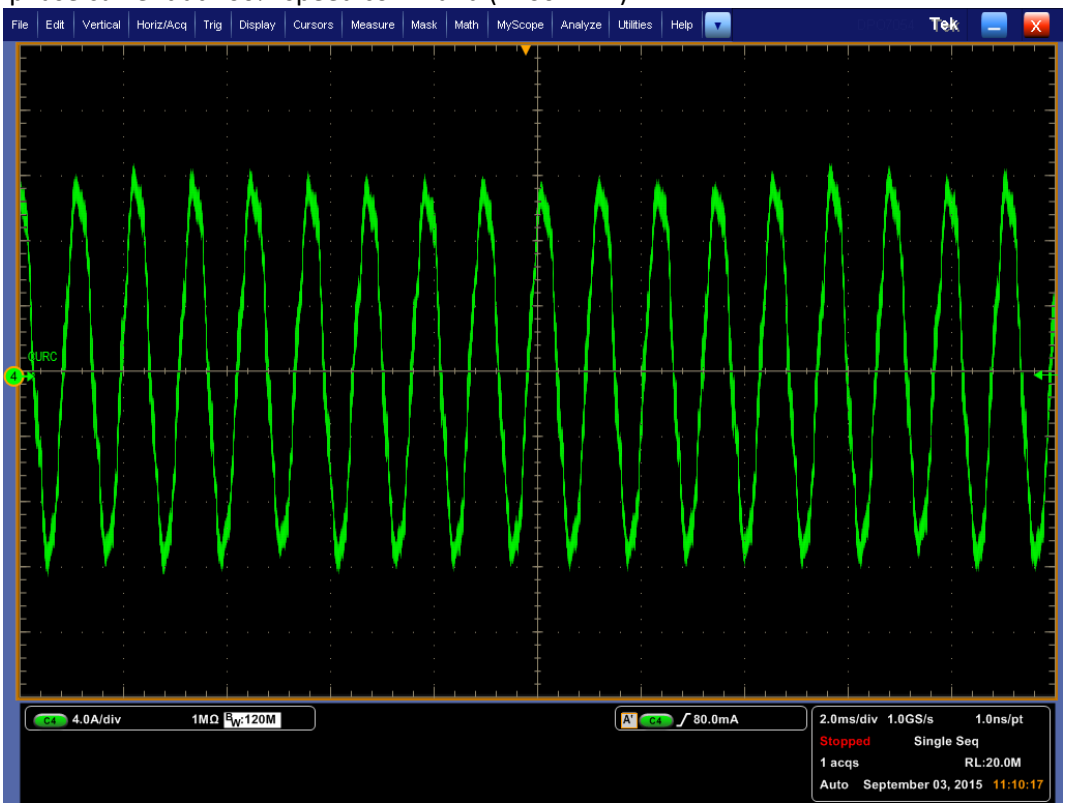
Motor phase current at 60% speed command (4620 RPM).



Motor phase current at 80% speed command (6160 RPM).

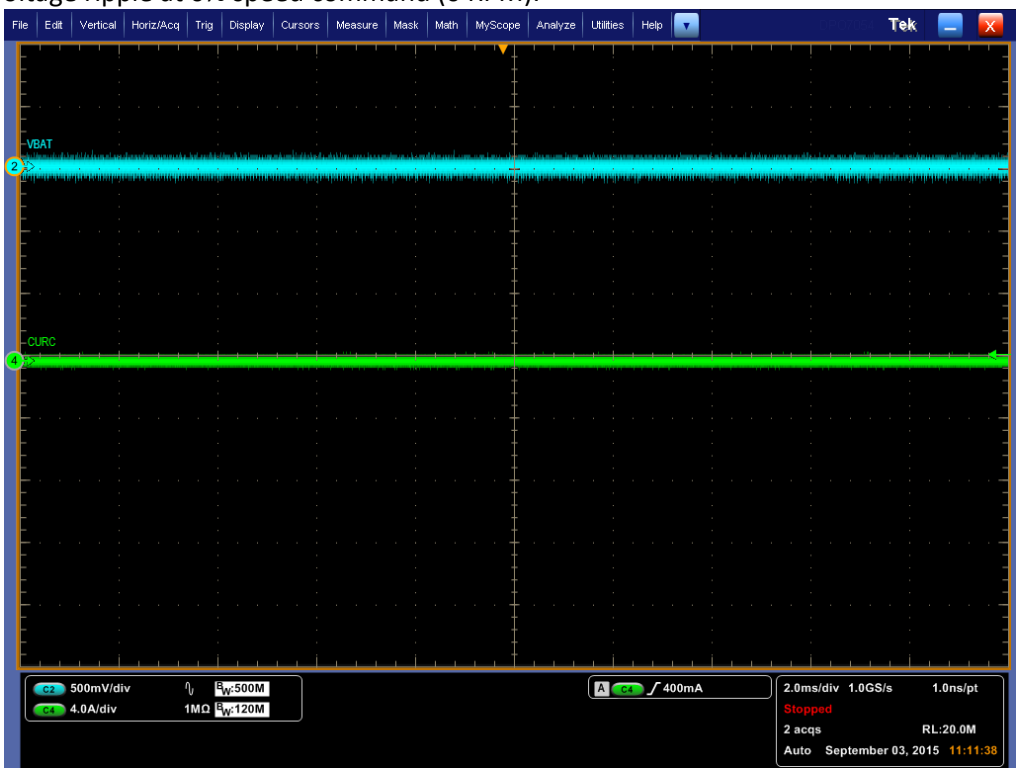


Motor phase current at 100% speed command (7700 RPM).

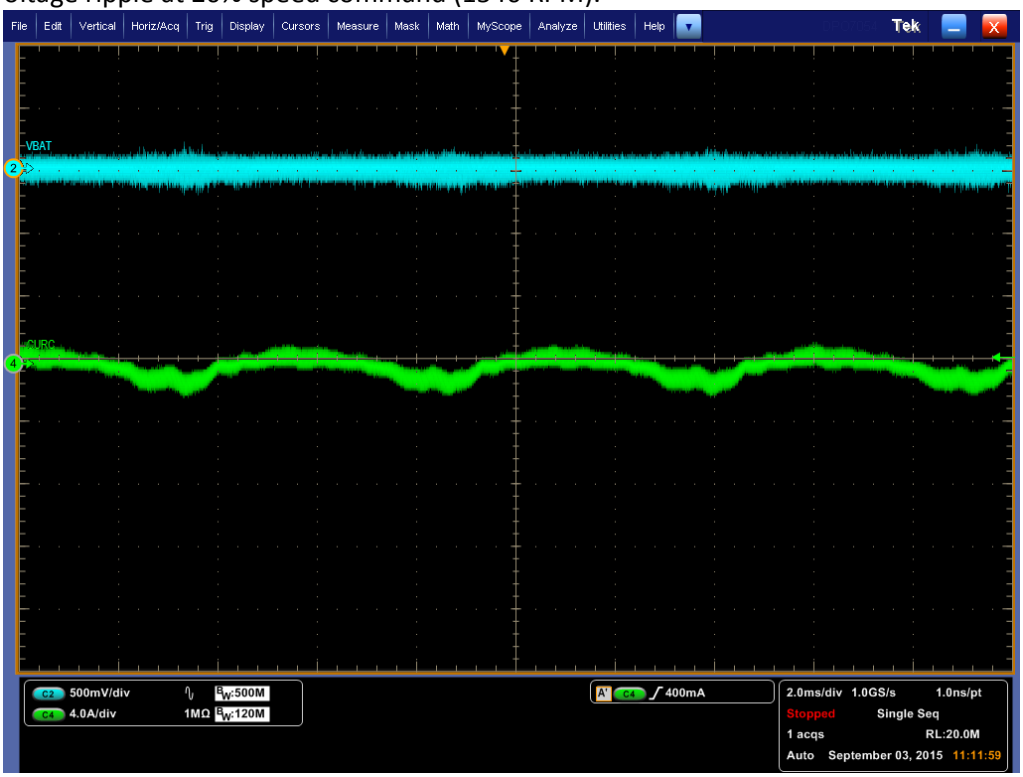


7.2.2 Supply Voltage Ripple

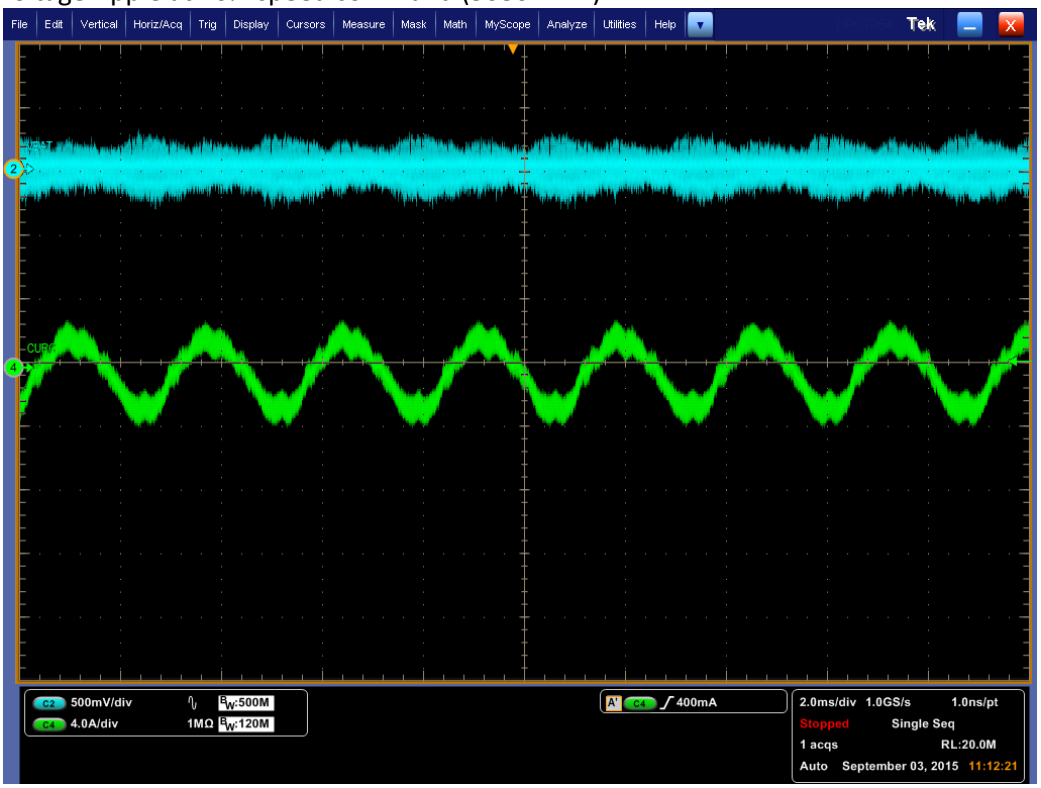
Test to examine the supply voltage (VBAT) ripple vs motor phase current.
Supply voltage ripple at 0% speed command (0 RPM).



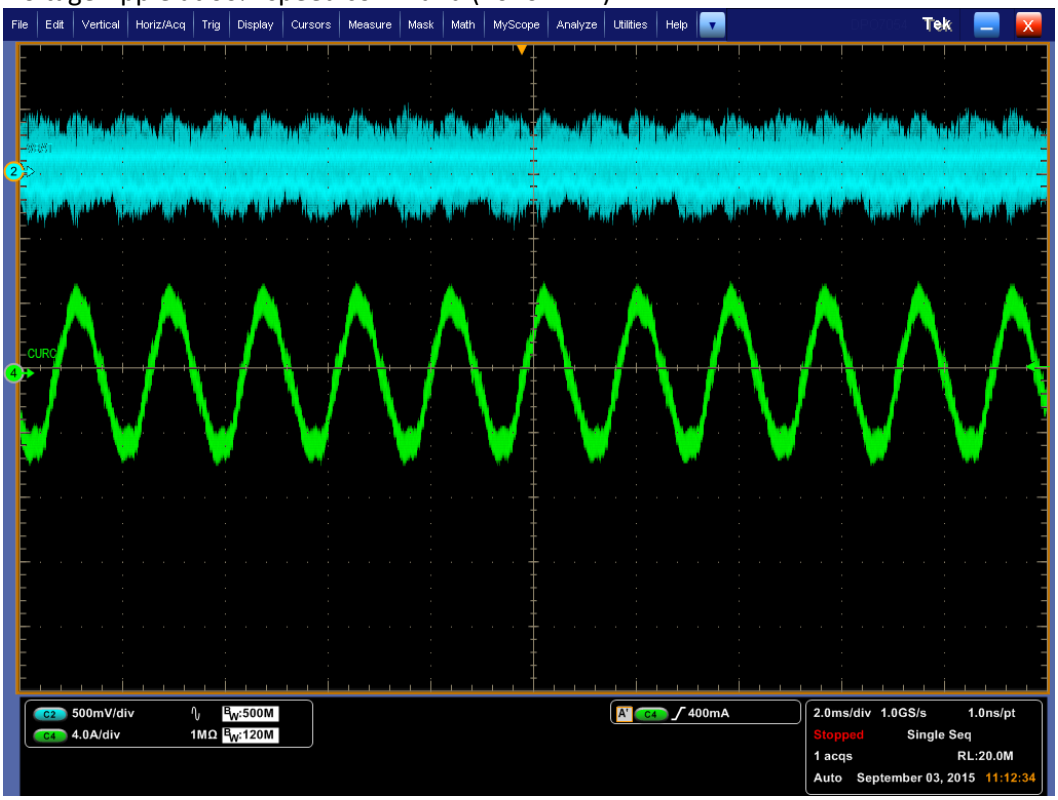
Supply voltage ripple at 20% speed command (1540 RPM).



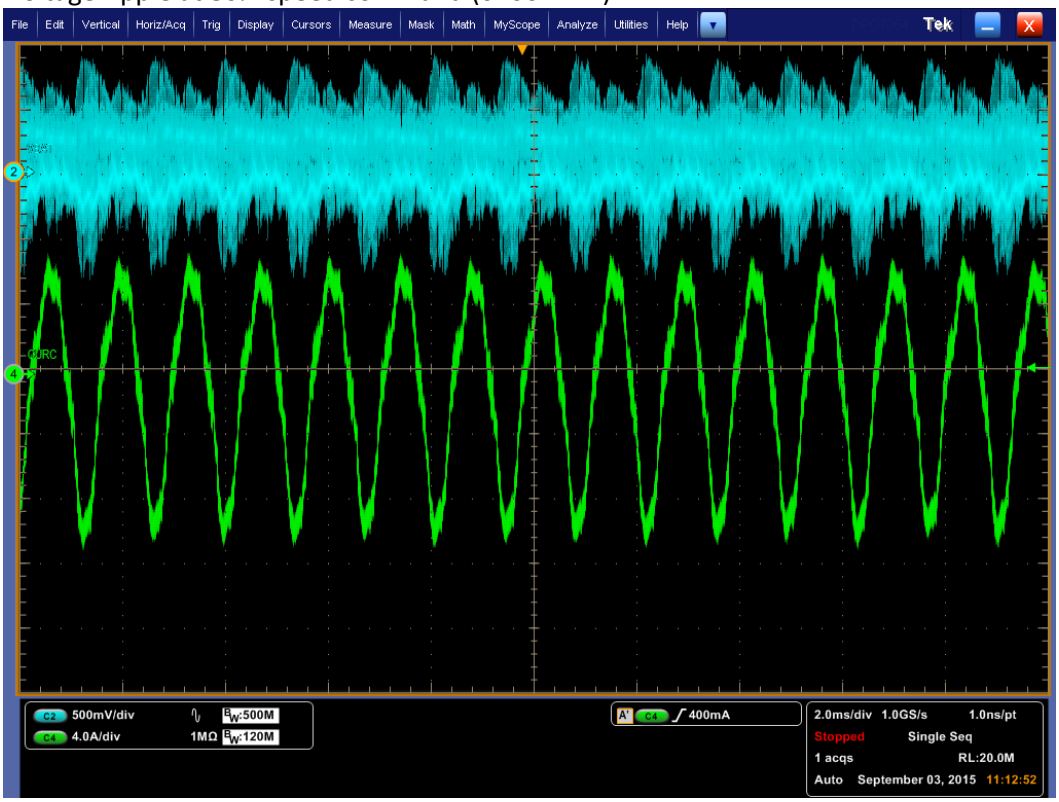
Supply voltage ripple at 40% speed command (3080 RPM).



Supply voltage ripple at 60% speed command (4620 RPM).



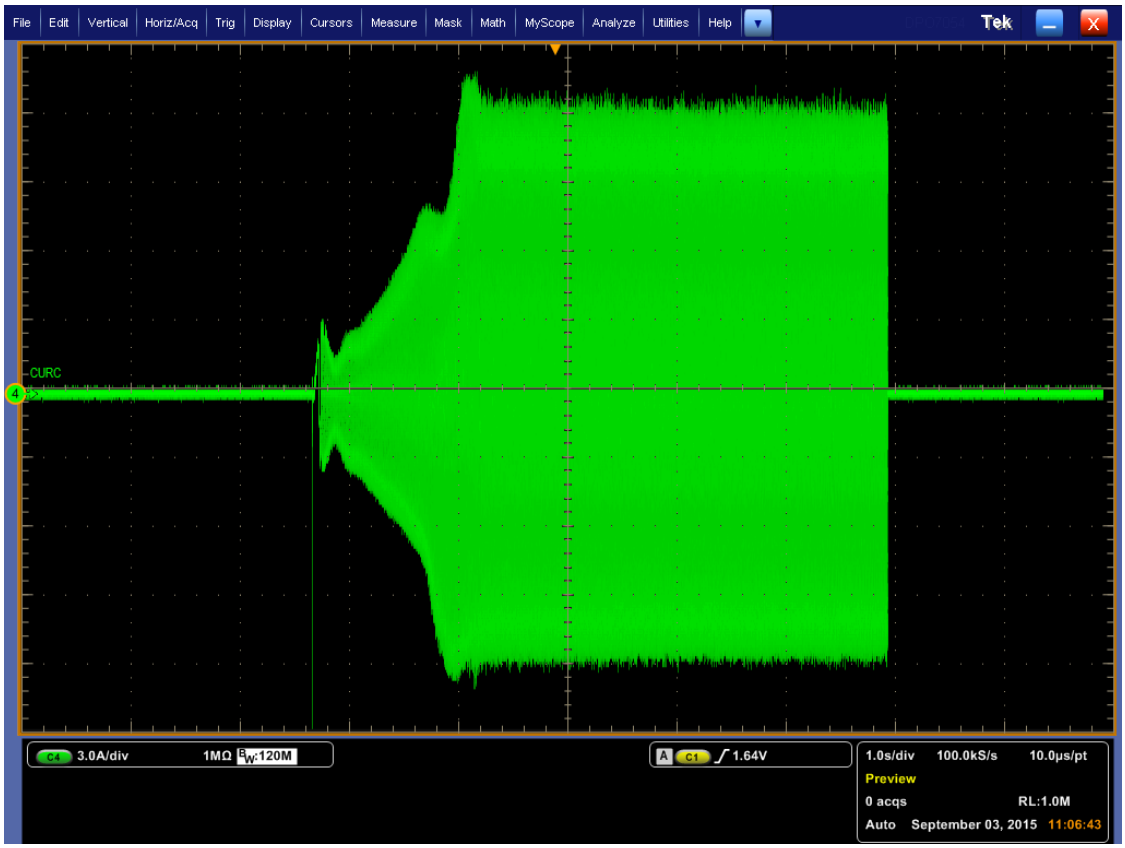
Supply voltage ripple at 80% speed command (6160 RPM).



Supply voltage ripple at 100% speed command (7700 RPM).

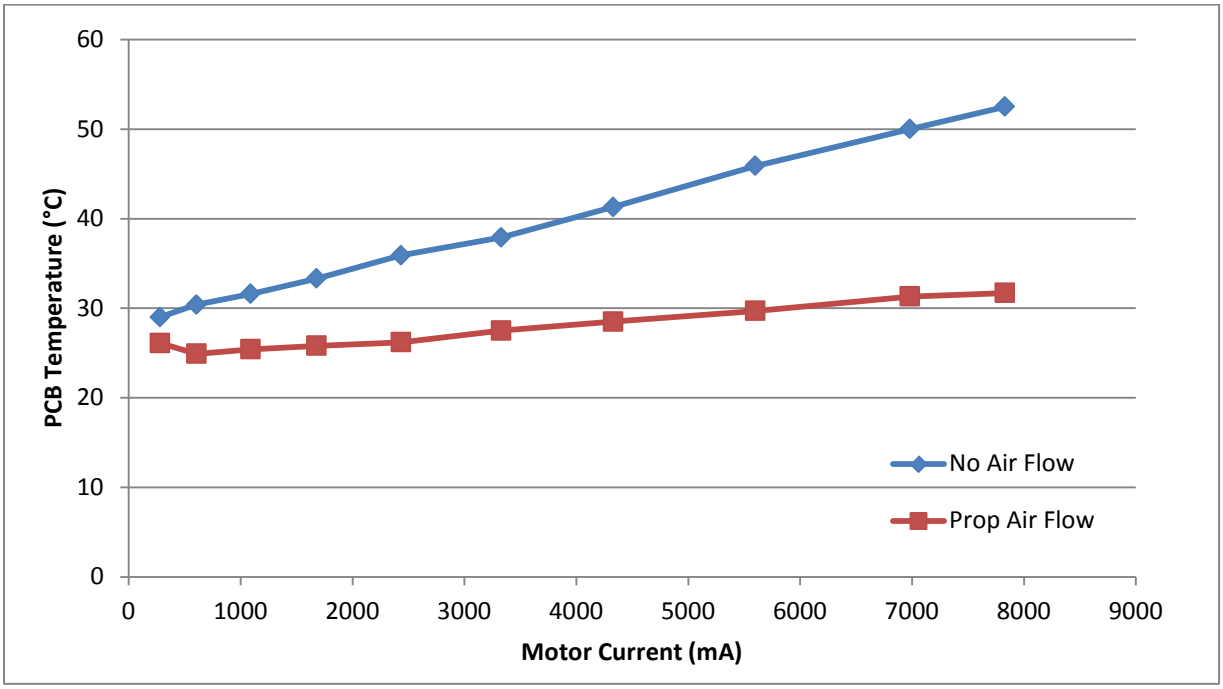
7.2.3 Motor Ramp Up

Test to examine motor ramping from stop (0 RPM) to full speed (7700 RPM).



7.2.4 Board Temperature

Test to examine the PCB temperature at different motor current levels. The motor phase current is measured with a current probe and the RMS value is recorded. The temperature is recorded with a thermocouple on the power stage. To mimic an application scenario the test is also run with the motor controller underneath the motor propeller (10”) it is driving. This propeller provides air flow for active cooling.



8 Design Files

The complete design files can be downloaded at the reference design home page at TIDA-00643.

9 Software Files

The updated MotorWare HAL and PROJECT files can be downloaded at the reference design home page TIDA-00643.

10 References

1. Texas Instruments InstaSPIN Solutions, www.ti.com/c2x-instaspin-b
2. Texas Instruments Motor Drive Solutions, http://www.ti.com/lsds/ti/analog/motordrivers/overview.page
3. Texas Instruments WEBENCH® Design Center, http://www.ti.com/webench
4. Texas Instruments E2E Community, http://e2e.ti.com/
5. Texas Instruments TMS320F28027F C2000 MCU, http://www.ti.com/product/tms320f28027f
6. Texas Instruments DRV8305 BLDC Gate Driver, http://www.ti.com/product/drv8305
7. Texas Instruments SIMPLE SWITCHER® 60 V 0.6 A Buck Regulator, http://www.ti.com/product/lmr16006
8. Texas Instruments CSD17573Q5B 30 V N-Channel NexFET™ Power MOSFETs, http://www.ti.com/product/csd17573q5b
9. Texas Instruments eCAP Module Reference Guide, http://www.ti.com/lit/ug/sprufz8a/sprufz8a.pdf
10. Texas Instruments Code Composer Studio Integrated Development Environment, http://www.ti.com/tool/ccstudio
11. Texas Instruments MotorWare, http://www.ti.com/tool/motorware
12. DRV8305 Evaluation Board, http://www.ti.com/tool/boostxl-drv8305evm
13. TMSF282027F Evaluation Board, http://www.ti.com/tool/launchxl-f28027

11 About the Author

Nicholas Oborny is an Applications Engineer for Texas Instrument's motor drive business, where he is responsible for supporting TI's motor drive portfolio, developing evaluation tools, product demonstrations, motor drive training, and motor drive IC development. Nicholas has several years of experience with hard-switching power stages and brushless motor control system development. Nicholas graduated from Texas A&M University College Station with a Bachelor's of Science in Computer Engineering.

Hector Hernandez Luque is an Applications Engineer Intern at Texas Instruments, where he is responsible for developing reference design solutions for the motor drive business. Hector brings to his role, experience in microprocessor interfacing, software development, parallel computing, schematic capture, and PCB layout design. Hector is currently earning his Bachelors of Science in Computer Engineering from University of Puerto Rico in Mayaguez, PR. Hector is a member of the Institute of Electrical and Electronics Engineers (IEEE).

IMPORTANT NOTICE FOR TI REFERENCE DESIGNS

Texas Instruments Incorporated ("TI") reference designs are solely intended to assist designers (“Buyers”) who are developing systems that incorporate TI semiconductor products (also referred to herein as “components”). Buyer understands and agrees that Buyer remains responsible for using its independent analysis, evaluation and judgment in designing Buyer’s systems and products.

TI reference designs have been created using standard laboratory conditions and engineering practices. **TI has not conducted any testing other than that specifically described in the published documentation for a particular reference design.** TI may make corrections, enhancements, improvements and other changes to its reference designs.

Buyers are authorized to use TI reference designs with the TI component(s) identified in each particular reference design and to modify the reference design in the development of their end products. HOWEVER, NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY THIRD PARTY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT, IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI components or services are used. Information published by TI regarding third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI REFERENCE DESIGNS ARE PROVIDED "AS IS". TI MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO THE REFERENCE DESIGNS OR USE OF THE REFERENCE DESIGNS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING ACCURACY OR COMPLETENESS. TI DISCLAIMS ANY WARRANTY OF TITLE AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, QUIET ENJOYMENT, QUIET POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO TI REFERENCE DESIGNS OR USE THEREOF. TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY BUYERS AGAINST ANY THIRD PARTY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON A COMBINATION OF COMPONENTS PROVIDED IN A TI REFERENCE DESIGN. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT OF TI REFERENCE DESIGNS OR BUYER’S USE OF TI REFERENCE DESIGNS.

TI reserves the right to make corrections, enhancements, improvements and other changes to its semiconductor products and services per JESD46, latest issue, and to discontinue any product or service per JESD48, latest issue. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All semiconductor products are sold subject to TI’s terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its components to the specifications applicable at the time of sale, in accordance with the warranty in TI’s terms and conditions of sale of semiconductor products. Testing and other quality control techniques for TI components are used to the extent TI deems necessary to support this warranty. Except where mandated by applicable law, testing of all parameters of each component is not necessarily performed.

TI assumes no liability for applications assistance or the design of Buyers’ products. Buyers are responsible for their products and applications using TI components. To minimize the risks associated with Buyers’ products and applications, Buyers should provide adequate design and operating safeguards.

Reproduction of significant portions of TI information in TI data books, data sheets or reference designs is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Buyer acknowledges and agrees that it is solely responsible for compliance with all legal, regulatory and safety-related requirements concerning its products, and any use of TI components in its applications, notwithstanding any applications-related information or support that may be provided by TI. Buyer represents and agrees that it has all the necessary expertise to create and implement safeguards that anticipate dangerous failures, monitor failures and their consequences, lessen the likelihood of dangerous failures and take appropriate remedial actions. Buyer will fully indemnify TI and its representatives against any damages arising out of the use of any TI components in Buyer’s safety-critical applications.

In some cases, TI components may be promoted specifically to facilitate safety-related applications. With such components, TI’s goal is to help enable customers to design and create their own end-product solutions that meet applicable functional safety standards and requirements. Nonetheless, such components are subject to these terms.

No TI components are authorized for use in FDA Class III (or similar life-critical medical equipment) unless authorized officers of the parties have executed an agreement specifically governing such use.

Only those TI components that TI has specifically designated as military grade or “enhanced plastic” are designed and intended for use in military/aerospace applications or environments. Buyer acknowledges and agrees that any military or aerospace use of TI components that have ***not*** been so designated is solely at Buyer's risk, and Buyer is solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI has specifically designated certain components as meeting ISO/TS16949 requirements, mainly for automotive use. In any case of use of non-designated products, TI will not be responsible for any failure to meet ISO/TS16949.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265