CY8CKIT-037

PSoC® 4 Motor Control Evaluation Kit Guide

Doc. No. 001-92562 Rev.**

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134 USA
Phone (USA): +1.800.858.1810
Phone (Intnl): +1.408.943.2600
www.cypress.com

Copyrights

© Cypress Semiconductor Corporation, 2015. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Trademarks

PSoC is a registered trademark and PSoC Creator is a trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Source Code

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer

CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.

Contents

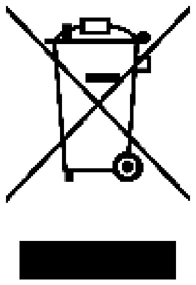
Safety Information

Regulatory Compliance

The CY8CKIT-037 PSoC® 4 Motor Control Evaluation Kit is intended for use as a motor control development platform for hardware or software in a laboratory environment. The board is an open-system design, which does not include a shielded enclosure. Therefore, the board may cause interference with other electrical or electronic devices in close proximity. In a domestic environment, this product may cause radio interference. In such cases, the user may be required to take adequate preventive measures. Also, this board should not be used near any medical equipment or RF devices.

Attaching additional wiring to this product or modifying the product operation from the factory default may affect its performance and cause interference with other apparatus in the immediate vicinity. If such interference is detected, suitable mitigating measures should be taken.

The CY8CKIT-037, as shipped from the factory, has been verified to meet with the requirements of CE as a Class A product.

	The CY8CKIT-037 contains ESD-sensitive devices. Electrostatic charges readily accumulate on the human body and any equipment, and can discharge without detection. Permanent damage may occur to devices subjected to high-energy discharges. Proper ESD precautions are recommended to avoid performance degradation or loss of functionality. Store unused CY8CKIT-037 boards in the protective shipping package.
	End-of-Life/Product Recycling The end of life for this kit is five years from the date of manufacture mentioned on the back of the box. Contact your nearest recycler to discard the kit.

General Safety Instructions

ESD Protection

ESD can damage boards and associated components. Cypress recommends that the user perform procedures only at an ESD workstation. If an ESD workstation is not available, use appropriate ESD protection by wearing an antistatic wrist strap attached to the chassis ground (any unpainted metal surface) on the board when handling parts.

Handling Boards

CY8CKIT-037 boards are sensitive to ESD. Hold the board only by its edges. After removing the board from its box, place it on a grounded, static-free surface. Use a conductive foam pad if available. Do not slide the board over any surface.

1. Introduction

Thanks for your interest in the CY8CKIT-037 PSoC® 4 Motor Control Evaluation Kit (EVK). This kit enables engineers to evaluate Cypress's PSoC family of devices for motor control applications. Based on this kit, customers can create control solutions for three dominant motor types: permanent magnet synchronous motor (PMSM), stepper, and brushless DC (BLDC).

The Cypress PSoC family of devices integrates abundant internal resources for motor control applications, such as Timer Counter Pulse Width Modulator (TCPWM), SAR-ADC, comparators, opamps, and universal digital blocks (UDBs). In addition, an ARM® Cortex™ core enables high-performance motor control solutions on PSoC devices. Headers provided on the EVK board allow you to connect it to the CY8CKIT-042 PSoC 4 Pioneer Kit board to create a complete motor control system.

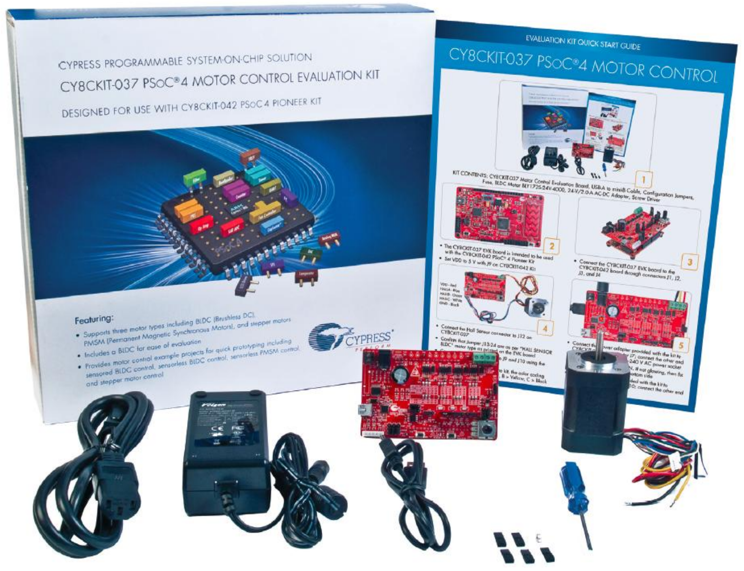
This kit guide documents the circuit structure of the Motor Control EVK board and explains how to configure it to create a solution for different motor types. It provides five example projects that cover sensored and sensorless BLDC control, PMSM sensorless field-oriented control (FOC), and stepper motor microstepping control. It also introduces the Bridge Control Panel (BCP) as a debugging tool in the motor control development process.

1.1 Kit Contents

The CY8CKIT-037 PSoC 4 Motor Control EVK includes the following contents, as shown in Figure 1-1:

- CY8CKIT-037 PSoC 4 Motor Control EVK main board
- AC-DC 24 V/2.0 A power adapter
- BLY172S-24V-4000 BLDC motor with sinusoidal back electromotive force
- USB-A to mini-B cable
- Configuration Jumpers
- Fuse
- Screwdriver
- Related documents:
  - Kit guide (this document)
  - Quick start guide
  - Release notes

Figure 1-1. CY8CKIT-037 PSoC 4 Motor Control EVK Contents



Visit www.cypress.com/shop for more information. Inspect the contents of the kit; if any parts are missing, contact your nearest Cypress sales office for help.

1.2 Additional Learning Resources

Visit www.cypress.com for additional learning resources in the form of datasheets, technical reference manuals, and application notes.

- *CY8CKIT-042 PSoC 4 Pioneer Kit _Schematic.pdf*
- *CY8CKIT-042 PSoC 4 Pioneer Kit_Layout.brd*

1.3 Documentation Conventions

Convention	**Usage**
`Courier New`	Displays file locations, user-entered text, and source code: `C:\...cd\icc\`
Italics	Displays file names and reference documentation: Read about the *sourcefile.hex* file in the *PSoC Designer User Guide*.
[**Bracketed, Bold**]	Displays keyboard commands in procedures: [**Enter**] or [**Ctrl**] [**C**]
File > Open	Represents menu paths: File > Open > New Project
Bold	Displays commands, menu paths, and icon names in procedures: Click the **File** icon and then click **Open**.
Times New Roman	Displays an equation: $2 + 2 = 4$
Text in gray boxes	Describes cautions or unique functionality of the product.

2. Kit Installation

This section describes the installation of the CY8CKIT-037 PSoC® 4 Motor Control EVK software and prerequisites.

2.1 Prerequisite Software

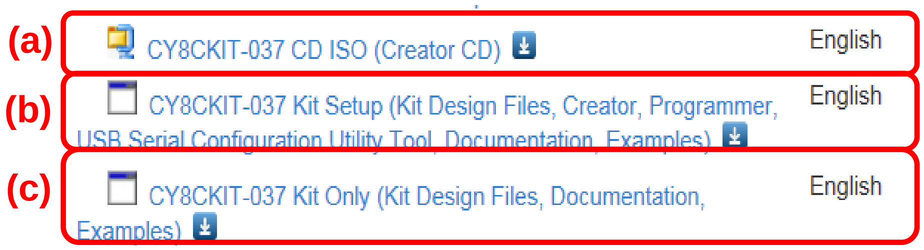
You must install the PSoC Creator™ Integrated Design Environment (IDE) and USB-Serial Configuration Utility software before the kit can be used. All Cypress software installations require administrator privileges. Administrator privileges are not required to execute the software after installation. Close all other Cypress software that is currently running before you install the kit software.

2.2 Install Kit Software

Follow these steps to install the CY8CKIT-037 PSoC 4 Motor Control EVK software:

1. Download the CY8CKIT-037 software from the Cypress web page: www.cypress.com/CY8CKIT-037. The CY8CKIT-037 software is available in three different formats for download (see Figure 2-1):

Figure 2-1. Available Formats for Downloading EVK Software



(a) CY8CKIT-037 CD ISO (Creator CD) English

(b) CY8CKIT-037 Kit Setup (Kit Design Files, Creator, Programmer, USB Serial Configuration Utility Tool, Documentation, Examples) English

(c) CY8CKIT-037 Kit Only (Kit Design Files, Documentation, Examples) English

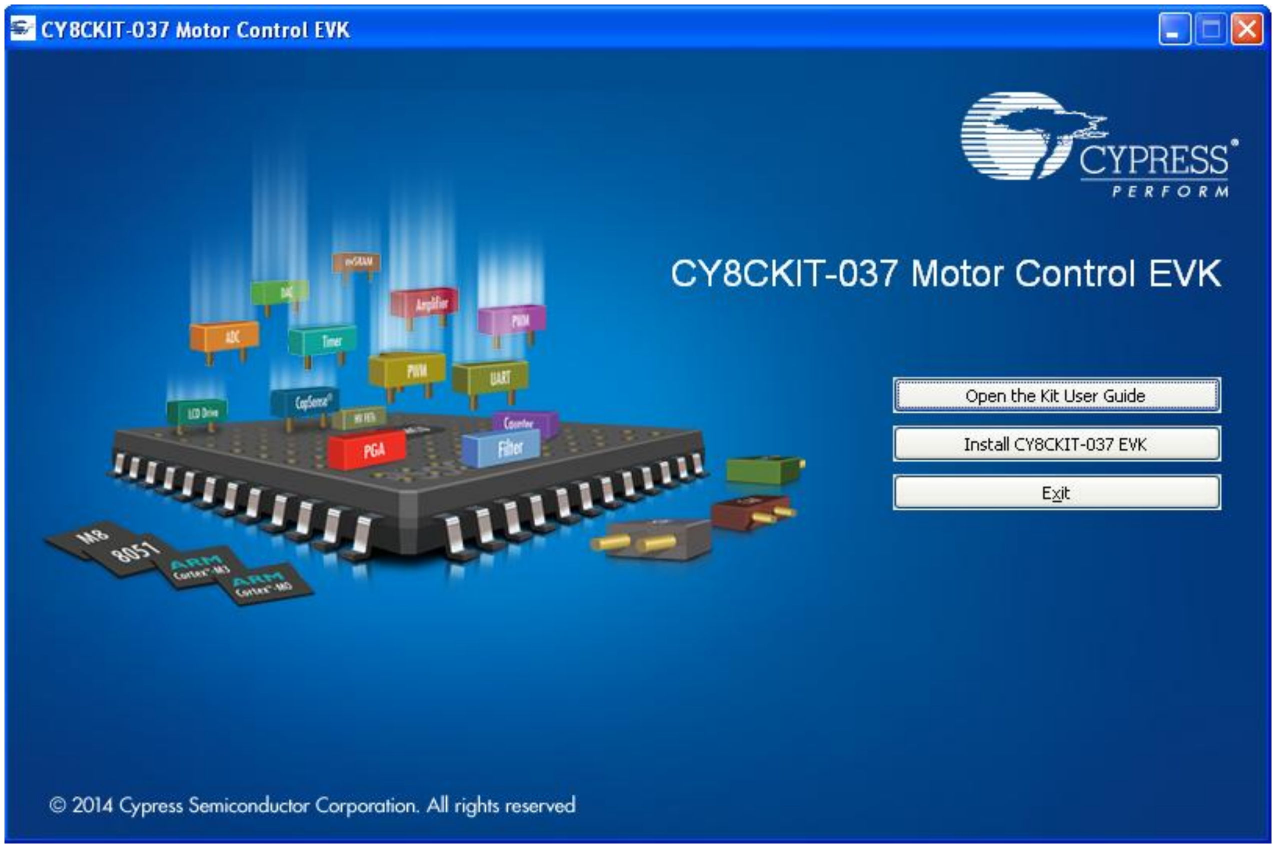
- **CY8CKIT-037 CD ISO**: This file is a complete package, stored in a CD-ROM image format, that you can use to create a CD or extract using ISO extraction programs, such as WinZip or WinRAR. The file can also be mounted like a virtual CD using virtual drive programs such as Virtual CloneDrive and MagicISO. This file includes all the required software, utilities, drivers, hardware files, and user documents.
- **CY8CKIT-037 Kit Setup**: This installation package contains all files related to the kit. However, it does not include the Windows Installer or Microsoft .NET framework packages, Internet Explorer, and Adobe Reader. If these packages are not on your computer, the installer directs you to download and install them from the Internet.
- **CY8CKIT-037 Kit Only**: This executable file installs only the kit contents, which include kit code examples, hardware files, and user documents. Use this package if the required software listed in step 4 is installed on your PC.

2. If you have downloaded the CD ISO file, mount it on a virtual drive. Extract the ISO contents if you do not have a virtual drive on which to mount. Double-click *cyautorun.exe* in the root directory of the extracted content or mounted ISO if “Autorun from CD/DVD” is not enabled on the PC. The installation window shown in Figure 2-2 will appear automatically.

   Note: If you are using the Kit Setup or Kit Only file, then go to step 3 for installation.

3. Click **Install CY8CKIT-037 EVK** to start the kit installation, as shown in Figure 2-2.

Figure 2-2. Kit Installer Startup Screen


CY8CKIT-037 Motor Control EVK
CYPRESS PERFORM
CY8CKIT-037 Motor Control EVK
Open the Kit User Guide
Install CY8CKIT-037 EVK
Exit
© 2014 Cypress Semiconductor Corporation. All rights reserved

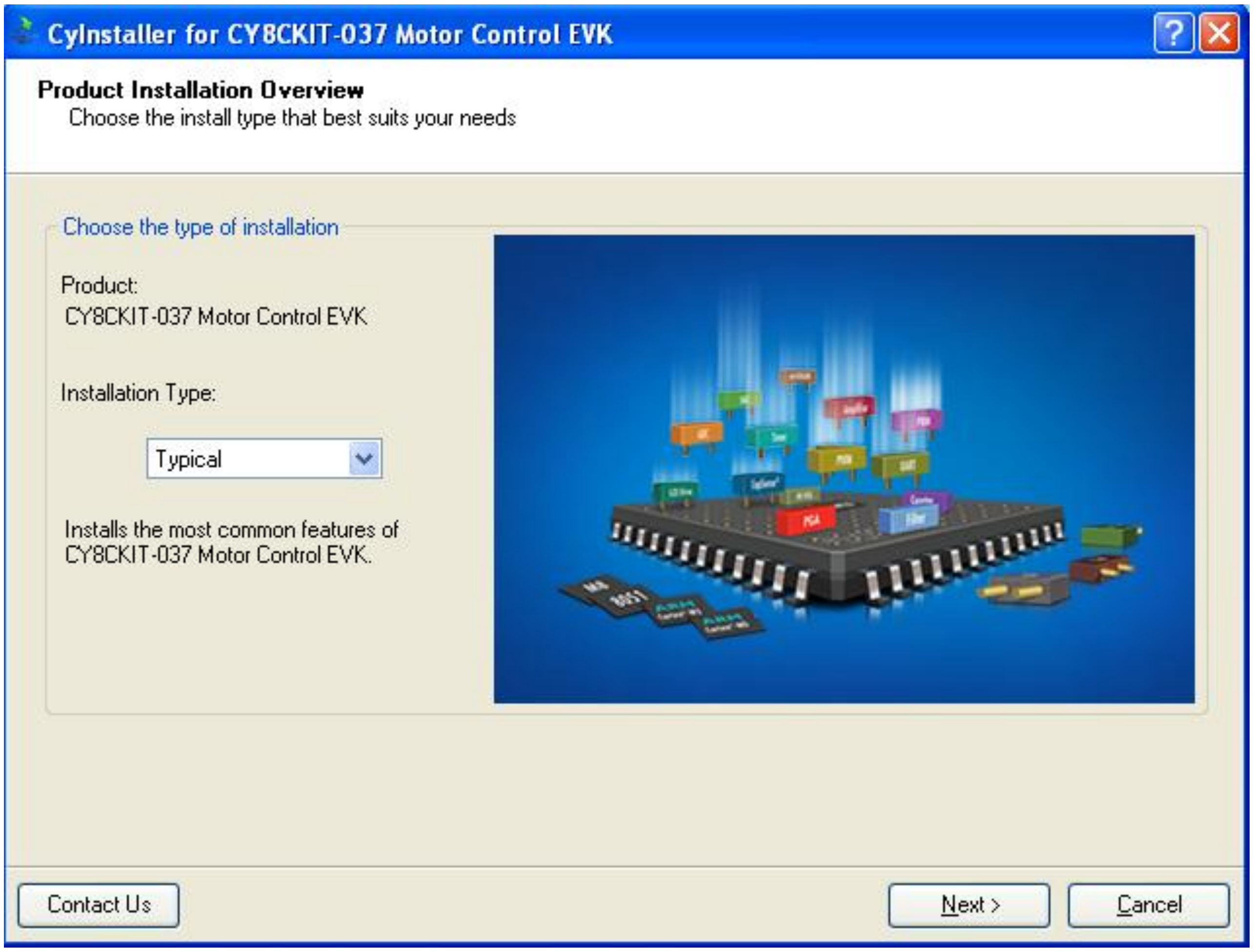
4. Select the folder in which you want to install the CY8CKIT-037 kit-related files. Choose the directory and click **Next**.

   When you click **Next** button, the CY8CKIT-037 ISO installer automatically installs the required software, if it is not present on your computer. Following is the required software:

   - PSoC Creator 3.2 or later version: Download the latest version from www.cypress.com/psoccreator.
   - PSoC Programmer 3.23 or later version: Download the latest version from www.cypress.com/programmer.

5. Select the **Typical installation** type in the **Product Installation Overview** window, as shown in Figure 2-3. Click **Next**.

Figure 2-3. Product Installation Overview Window


CyInstaller for CY8CKIT-037 Motor Control EVK

Product Installation Overview
Choose the install type that best suits your needs

Choose the type of installation

Product:
CY8CKIT-037 Motor Control EVK

Installation Type: Typical

Installs the most common features of CY8CKIT-037 Motor Control EVK.

When the installation begins, a list of packages appears on the installation page. A green check mark appears next to each package after successful installation.

6. Enter your contact information or select the option **Continue without Contact Information**. Click **Finish** to complete the CY8CKIT-037 kit installation.

   After the installation is complete, the kit contents are available at `<Install_Directory>\CY8CKIT-037 Motor Control EVK\1.0`. The default locations are as follows:

   - Windows 7 (64-bit): `C:\Program Files (x86)\Cypress\CY8CKIT-037 Motor Control EVK\1.0`
   - Windows 7 (32-bit): `C:\Program Files\Cypress\CY8CKIT-037 Motor Control EVK\1.0`

   Note: For Windows 7/8/8.1 users, the installed files and the folder are read-only. To change the property, right-click the folder and choose **Properties** > **Attributes** and then disable the **Read-only** option. Click **Apply** and **OK** to close the window.

   After the installation is complete, the following are installed on your computer:

   - PSoC Creator 3.2 or later version
   - PSoC Programmer 3.23 or later version
   - USB-Serial Configuration Utility

- Kit documents
  - Quick start guide
  - Kit guide
  - Release notes
- Firmware
  - Sensored BLDC Motor Control example project
  - Sensorless BLDC Motor Control example project
  - Sensorless Foc Motor Control example project
  - SingleShunt Foc Motor Control example project
  - Stepper Motor Control example project
- Hardware
  - Schematic
  - Layout
  - Gerber
  - PCB assembly drawing
  - Bill of materials (BOM)

2.3 Uninstall Software

The software can be uninstalled using one of the following methods:

- Go to **Start** > **Control Panel** > **Programs and Features**; select the appropriate software package and click **Uninstall**.
- Go to **Start** > **All Programs** > **Cypress** > **Cypress Update Manager** > **Cypress Update Manager**; click **Uninstall** for the appropriate software package.
- Select the “CY8CKIT-037 Motor Control EVK 1.0 Rev **” row and click **Uninstall**. In the **Product Installation Overview** window, select **Remove** from the **Installation Type** drop-down menu. Follow the instructions to uninstall.

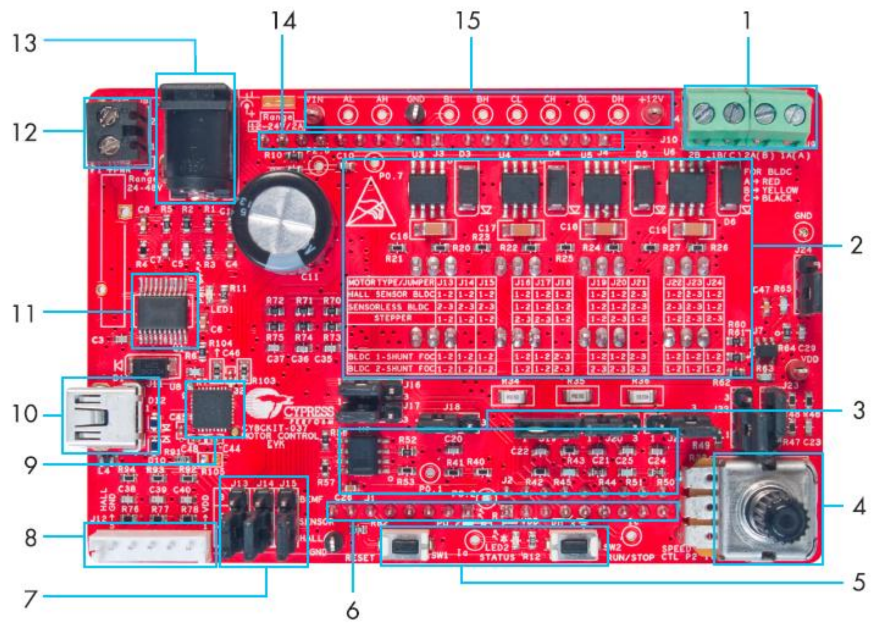
  Note: This method will uninstall only the kit software and not all the other software that may have been installed along with the kit software.

3. Kit Overview

3.1 CY8CKIT-037 Evaluation Kit Overview

The motor control system can be separated into two parts: the driver board and the controller board. The CY8CKIT-037 Motor Control EVK is the driver board, which contains the DC/DC power circuit, dual H-bridge circuit, motor current and bus voltage sampling and processing circuit, protection circuit, user configuration circuit, and connectors to the controller board. The controller board receives the signals, implements the proper algorithm to process them, and then generates control signals to the driver board to run the motor. Figure 3-1 shows the EVK board and its general description. CY8CKIT-037 EVK is the driver board; the CY8CKIT-042 kit works as controller board. They are interfaced with Arduino-compatible connectors.

Figure 3-1. General Description of EVK Board



1	Motor Winding Connector
2	PWM Driver for Dual H-Bridge Circuit
3	Current Chopper and Processing Circuit
4	Potentiometer to Change Motor Speed
5	Reset Button, Status LED and Run/Stop Button
6	Arduino™ Connectors to CY8CKIT-042
7	Hall and BEMF Selection Jumpers
8	Hall Sensor Interface
9	USB Serial Chip
10	USB Connector to PC
11	12 V DC/DC Switching Regulator
12	24-48 V DC Power Supply Connector
13	12-24 V/2 A Power Jack
14	Arduino Connectors to CY8CKIT-042
15	Test Points

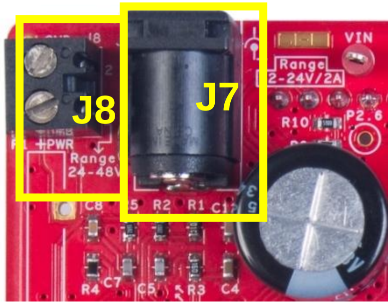
3.2 Kit Operation and Configuration Guide

3.2.1 DC Power Supply Connector

There are two connectors for DC power input, J7 and J8, as Figure 3-2 shows. J7 is the standard power jack for DC adapters, with 12-V or 24-V DC output. J8 enables the kit to get input power from a general variable DC power supply with 24- to 48-V DC output voltage. The input current of both connectors can be up to 2 A. For schematic details, see Input Protection Circuit. CY8CKIT-042 gets 12 V from the DC/DC converter on CY8CKIT-037. The LDO on CY8CKIT-042 generates the 3.3-V supply for PSoC if the VDD on CY8CKIT-042 is configured as 3.3 V. If the VDD on CY8CKIT-042 is configured as 5 V, CY8CKIT-042 needs 5 V from the USB cable.

CAUTION	**Do not connect power to both J7 and J8 simultaneously. Also, when using J7, connect the power cable to it before loading the AC supply at the AC/DC adapter to avoid a potential spark.**

Figure 3-2. Connectors for DC Power Input



3.2.2 Motor Winding Connectors

J9 and J10 provide four pins for motor windings, as shown in Figure 3-3. The BLDC and PMSM motors use three pins, while the stepper motor application needs all four pins. Connection details are as follows:

For BLDC and PMSM:

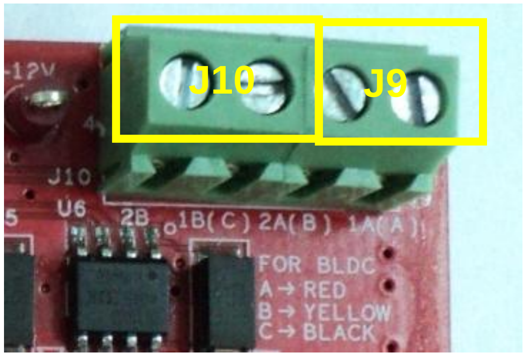
- Pin 1–1A (A) $\rightarrow$ Motor winding A – Red
- Pin 2–2A (B) $\rightarrow$ Motor winding B –Yellow
- Pin 3–1B (C) $\rightarrow$ Motor winding C – Black

For the stepper motor:

- Pin 1–1A (A) $\rightarrow$ Motor winding 1 terminal A – Blue
- Pin 2–2A (B) $\rightarrow$ Motor winding 1 terminal B – Yellow
- Pin 3–1B (C) $\rightarrow$ Motor winding 2 terminal A – Red
- Pin 4–2B $\rightarrow$ Motor winding 2 terminal B – Green

For schematic details, see Connectors.

Figure 3-3. Connectors to Motor Windings



3.2.3 Hall Sensors Connector

J12 is the Hall sensors connector for the Sensored BLDC Motor Control example project, as Figure 3-4 shows. It is a five-pin connector. The details of each pin are as follows.

- 1 – VDD $\rightarrow$ Hall sensor board VDD – Red
- 2 – HALL A $\rightarrow$ Hall sensor A output – Blue
- 3 – HALL B $\rightarrow$ Hall sensor B output – Green
- 4 – HALL C $\rightarrow$ Hall sensor C output – White
- 5 – GND $\rightarrow$ Hall sensor board ground – Black

For schematic details, refer to Hall Sensors and BEMF Sensing Circuit.

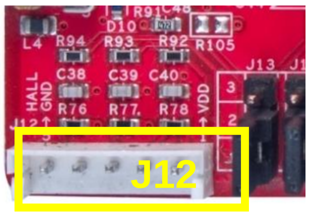


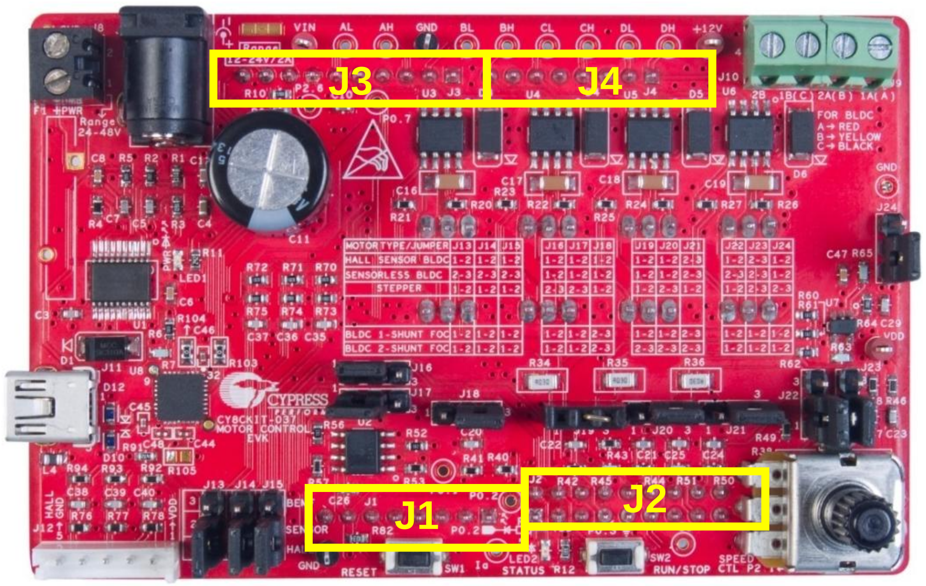
Figure 3-4. Hall Sensors Connector

3.2.4 Connectors to CY8CKIT-042 Board

Figure 3-5 shows the Arduino-compatible connectors to the CY8CKIT-042 board: J1, J2, J3, and J4. You can plug the EVK board into the CY8CKIT-042 board through these connectors. For schematic details, refer to Connectors.

Note: Pin connectors J1, J2, J3, and J4 may become bent inadvertently when plugging the EVK board into CY8CKIT-042.

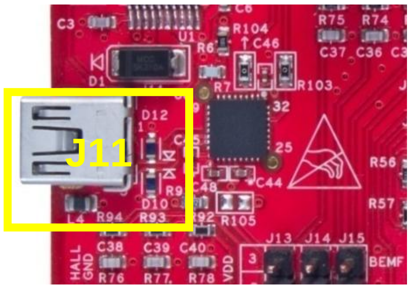
Figure 3-5. Connectors to CY8CKIT-042 Board



3.2.5 USB Connector

To monitor the real-time parameters while the motor is running, you can use the BCP (refer to Bridge Control Panel Monitor Tool Guide) to get data through the USB-to-UART bridge circuit on the EVK board during the debugging process. The USB-to-UART bridge circuit on the EVK board and PC can be connected by USB cable through the J11 connector, as Figure 3-6 shows. For schematic details of the USB-to-UART bridge circuit part, refer to USB-to-UART Bridge Controller Circuit.

Figure 3-6. USB-to-UART Bridge Controller Circuit



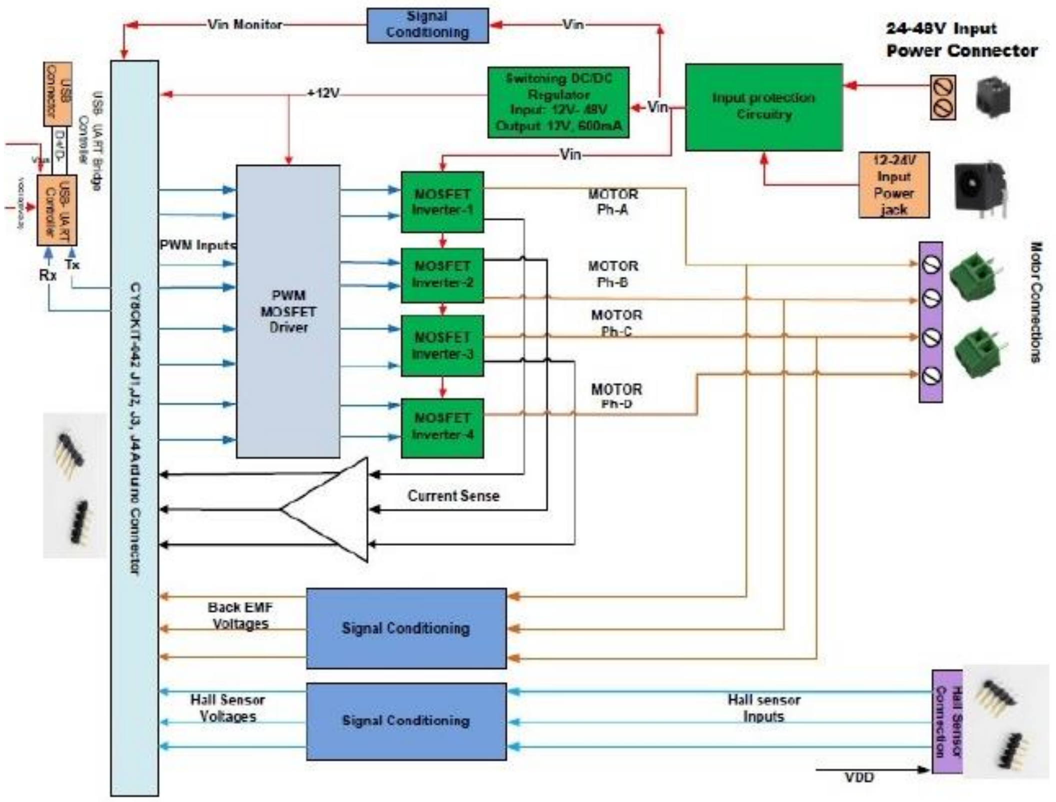
4. Kit Hardware Schematic Details

4.1 Block Diagram Overview

Figure 4-1 illustrates the CY8CKIT-037 hardware block diagram. The major features are as follows:

- Input protection circuit
- DC/DC switching regulator
- MOSFET dual H-bridge and dual H-bridge PWM drivers
- Phase current detecting and processing circuit
- Hall and back electromagnetic force (BEMF) sensing circuit
- USB-to-UART bridge controller circuit
- Connectors
- Test points and LEDs

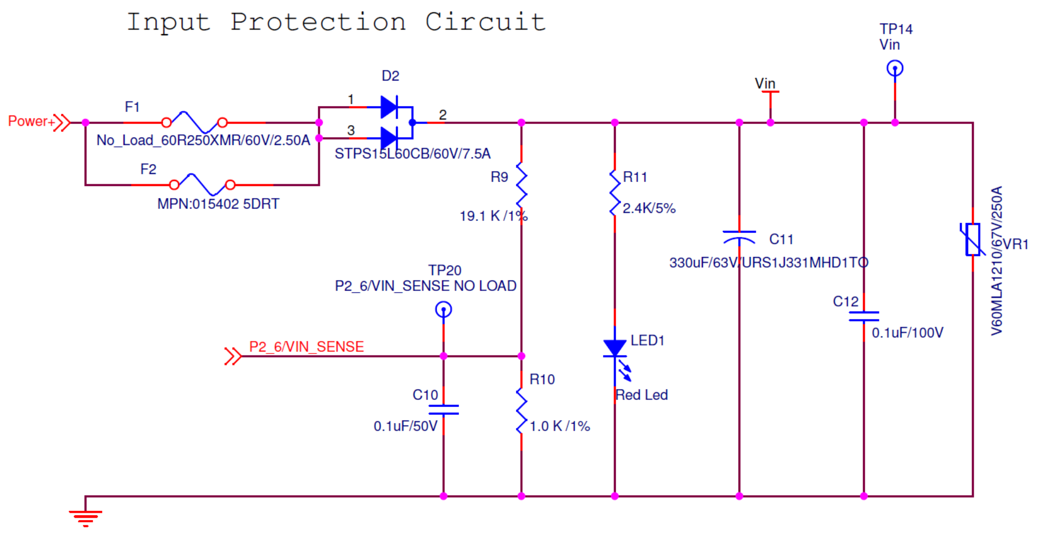
Figure 4-1. CY8CKIT-037 Hardware Block Diagram



4.2 Input Protection Circuit

Figure 4-2 shows the input protection circuit that consists of three parts: overcurrent protection by fuse F1 or F2, power supply reverse protection by Schottky diode D2, and input voltage inrush protection by varistor VR1. Notice that there are two parallel fuses: F1 is a PTC resettable fuse, and F2 is the common thermal fuse. F2 is populated onboard by default. You can also use PTC fuse F1 to replace F2 according to your requirements.

Figure 4-2. Input Protection Circuit

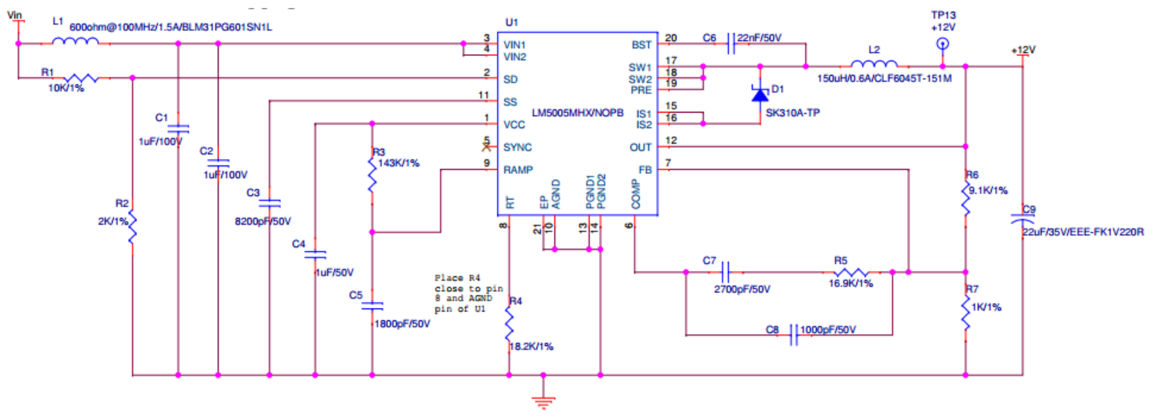


CAUTION	**Do not populate both F1 and F2 on the board. Otherwise, when an overcurrent condition occurs. F1 and F2 cannot cut off the circuit, which will lead to component damage on the board.**

4.3 DC/DC Switching Regulator

Figure 4-3 shows the DC/DC switching regulator circuit on the EVK board. It converts the voltage V_{in} (output of the input protection circuit, 0.5 V to 1 V lower than the input voltage) to +12 V. This switching converter is a buck regulator using LM5005. LM5005 has an internal 75-V N-channel buck switch with an output current capability of 2.5 A. It is designed to take an input in the range of 7–75 V and provides 12-V output with 1.5-A current capability when the input voltage is higher than 13 V.

Figure 4-3. Switching DC/DC Converter



When the voltage at the SD pin of LM5005 is less than 1.225 V, the IC will go into an inactive state. The external R1 and R2 divider with 10 kΩ and 2 kΩ will give exactly 1.225 at 7.35-V input. If the input goes below 7.35 V, LM5005 will be in an inactive state.

The +12V output of the regulator provides power to the dual H-bridge driver chips and the PSoC 4 Pioneer Kit. The other circuits and chips on the EVK board also need a power supply in the range of 2.7 V to 5.5 V, which is VDD. But the EVK board does not contain this converter. It gets this power from the controller board through Arduino headers.

4.4 MOSFET Dual H-Bridge and Dual H-Bridge PWM Drivers

This part contains four half-bridge circuits and four driver circuits. For BLDC and PMSM control, you can use three half-bridge circuits to form a standard three-phase inverter. For the stepper motor, you can control two windings of the stepper motor separately via the dual H-bridge.

Figure 4-4 shows the driver circuit, containing four ASIC driver chips (IR2101) and relative peripheral components. IR2101 can source 100-mA or sink 210-mA current for MOSFETs. Each chip can drive two MOSFETs of a half-bridge, and its high-side PWM operates with a floating supply by a peripheral bootstrap circuit.

Figure 4-4. Dual H-Bridge Driver Circuit

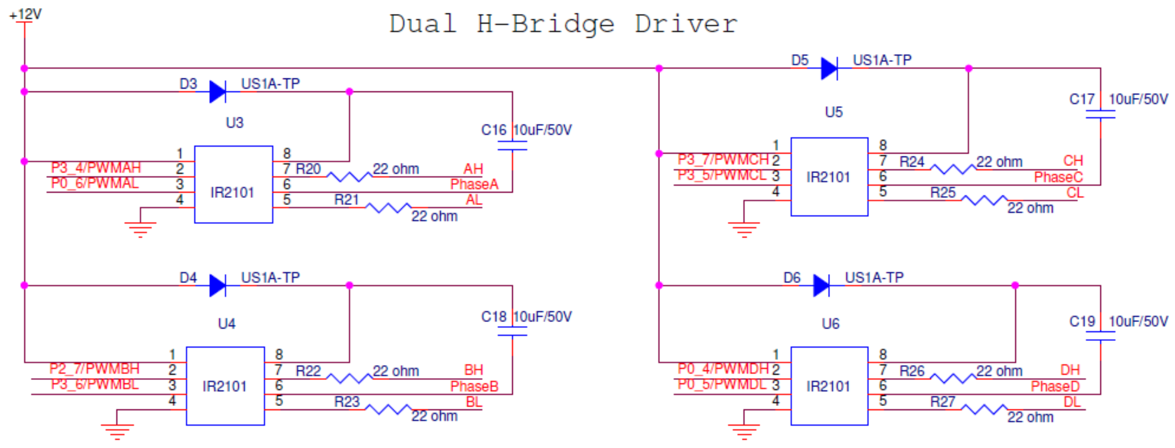
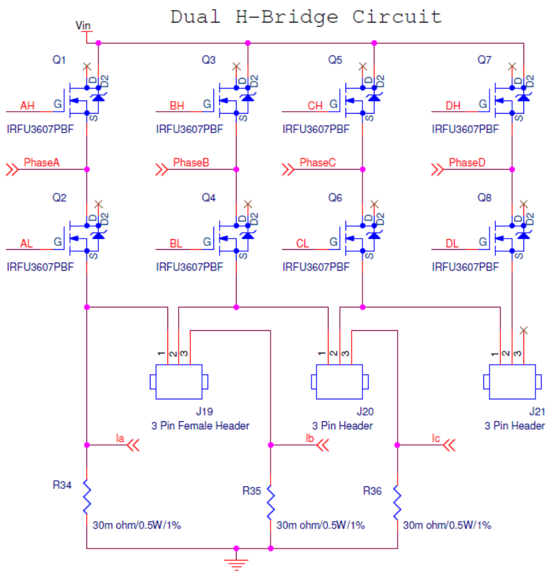


Figure 4-5 shows the MOSFET dual H-bridge. There are four half bridges in total, each one serving a motor phase or winding. Considering the compatibility for different motor control types, the population of components J19, J20, and J21 needs to be adjusted to fit different motor type applications.

Figure 4-5. MOSFET Dual H-Bridge



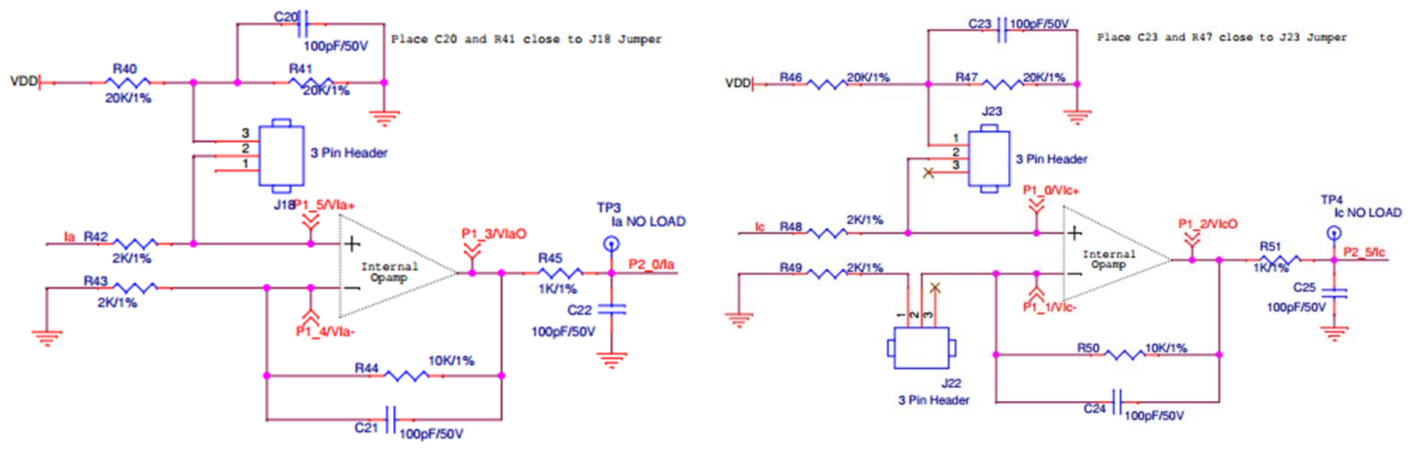
4.5 Phase Current Detecting and Processing Circuit

As Figure 4-5 shows, three sensing resistors, R34, R35 and R36, are inserted into three half-bridges. When the motor is running, the winding current floats through these resistors. So, detecting the voltage on the sensing resistors can get the phase current of the motor windings. Because of the sensing resistors' low value and the noise of the running motor, you need to amplify and filter the voltage signal. Figure 4-6 shows the current detecting and processing circuit for winding A and winding C.

The core of this part is the amplifier. You use PSoC internal opamps on the controller board. J18 and J23 are used only for PMSM FOC control because in the FOC application, the winding current can be both positive and negative. J18 and J23 add 1/2 VDD bias voltage to the sensing signal to make it always positive.

J22 is used only for the sensorless BLDC application, because the sensorless BLDC firmware uses an inside low-power comparator (LPComp). P1_1 is dedicated to its negative terminal, which needs to be connected to the Vin divider circuit (P2_6/Vin). R49 will change the divider proportion. You need to short pin 2 and pin 3 of J22 to remove R49 from the circuit in the Sensorless BLDC Motor Control example project.

Figure 4-6. Current Detecting and Processing Circuit



(Winding A) (Winding C)

Figure 4-7 shows the external two-phase winding current comparing circuit, which is used only in the stepper motor application. The positive terminal input is the current sensing signal after amplifying and filtering, and the negative terminal input is the current reference set by the internal IDAC. The comparator output is routed to the PWM kill terminal to shut down the PWM output, forcing the winding current to follow the IDAC current reference.

Figure 4-7. Two-Phase Winding Current Comparing Circuit

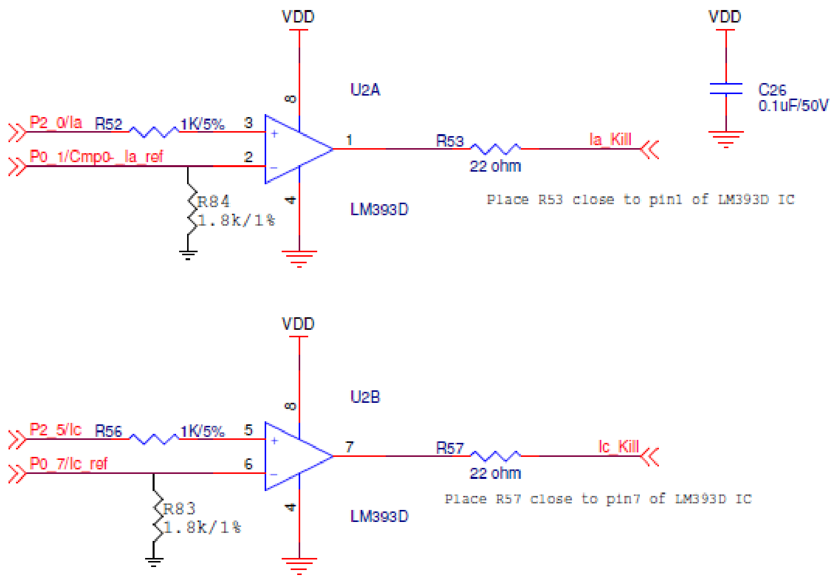
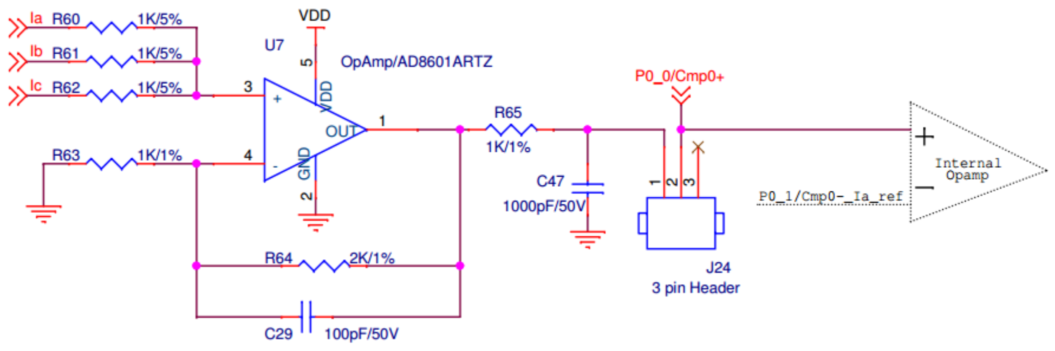


Figure 4-8 shows the overcurrent protection circuit. R60, R61, and R62 add the three-phase winding current, and the external opamp U7 amplifies and filters the sum of the currents. Its output is routed to the positive terminal of the internal LPComp to be compared with the overcurrent threshold set by the IDAC.

Figure 4-8. Overcurrent Protection Circuit



4.6 Hall Sensors and BEMF Sensing Circuit

Both sensored and sensorless BLDC motor can be rotated with the EVK board. The sensored BLDC gets its rotor position from the Hall sensors, and the sensorless BLDC gets its rotor position by detecting the BEMF signal from the stator winding. Figure 4-9 shows the Hall sensor interface. The Hall sensor output is in an open-collector structure, so the interface adds pull-up resistors to make the PSoC device receive the correct Hall sensor signal.

Figure 4-9. Hall Sensor Interface

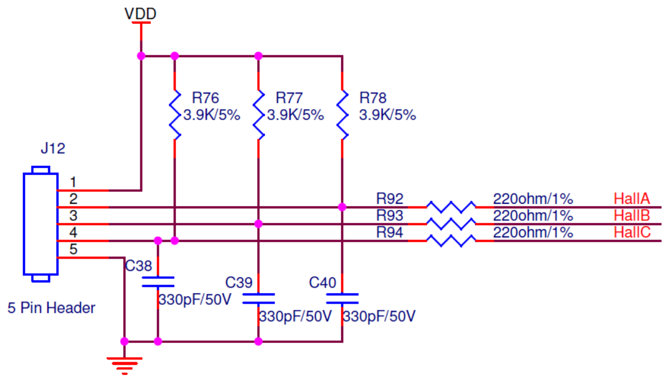


Figure 4-10 shows the voltage divider and filter circuit for BEMF detecting. Three phases of the Hall sensor and BEMF signals are routed to jumpers J13, J14, and J15, as shown in Figure 4-11. Users can select sensored or sensorless configuration via the three jumpers.

Figure 4-10. Sensorless BLDC BEMF Detecting Circuit

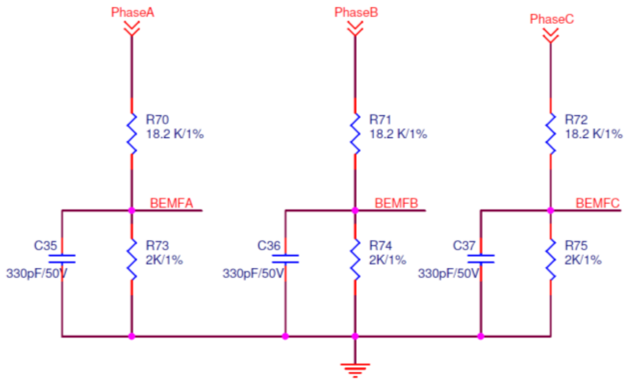
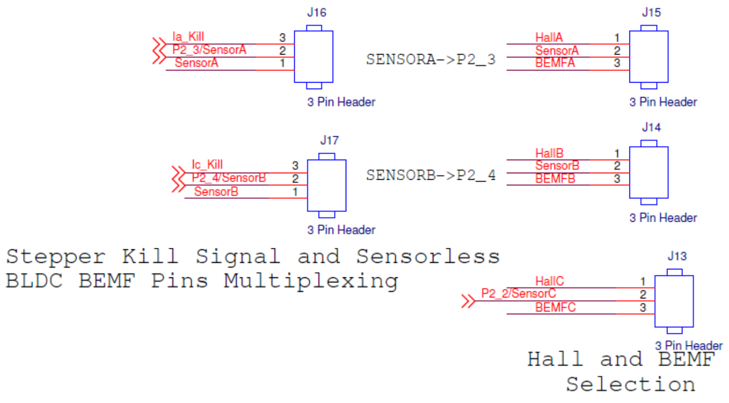


Figure 4-11 shows that the two Hall sensor and BEMF signals, as well as the two external comparator output kill signals of the stepper motor, are routed to J16 and J17. Since the GPIO pin numbers of PSoC 4 are limited, these signals need to share two GPIO pins. Be sure to configure J16 and J17 correctly for the BLDC and stepper motor applications.

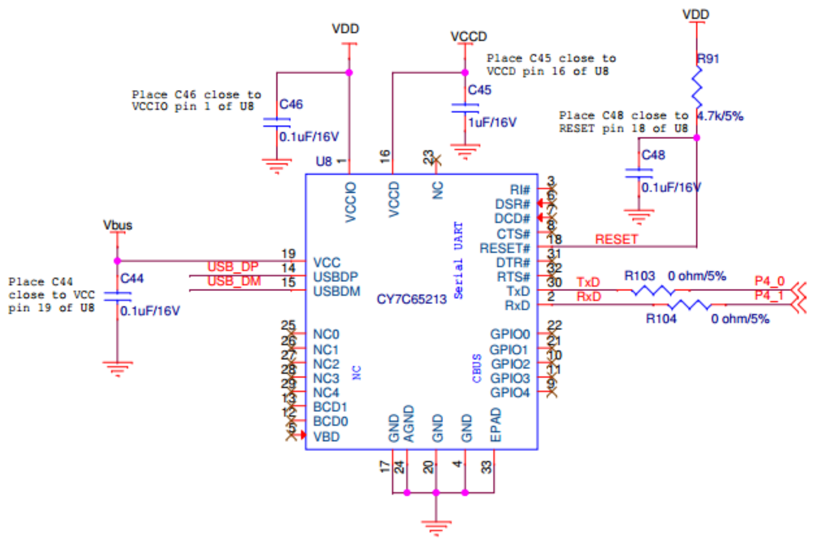
Figure 4-11. Signal Configuration Jumpers



4.7 USB-to-UART Bridge Controller Circuit

Figure 4-12 shows the USB-to-UART bridge controller used for transferring data, such as speed and winding current, that you want to monitor on the BCP. The bridge controller receives the data from PSoC 4 on CY8CKIT-042 over UART and transmits it over USB to be monitored using BCP software.

Figure 4-12. USB-to-UART Bridge Controller



4.8 Connectors

There are several connectors on the EVK board. Figure 4-13 shows the input power connectors. J7 is the standard power jack for DC adapters, with an output from 12 V to 24 V. J8 can accept an input voltage from 24 V to 48 V through raw wire from a general variable DC power supply. The input current of both connectors can be as high as 2 A.

Figure 4-13. Input Power Supply Connectors

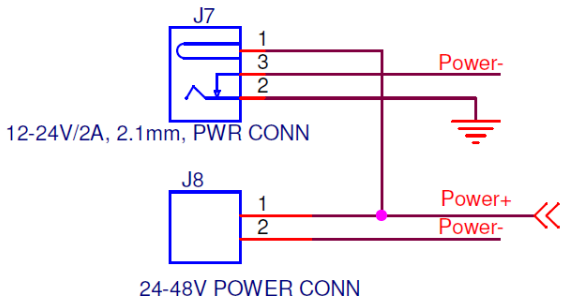


Figure 4-14 shows the motor winding connectors. The two connectors provide four pins in total. The BLDC motor contains three windings with a common neutral point inside the motor. It uses only three pins. The stepper motor contains two independent windings, so it needs all four pins.

Figure 4-14. Motor Winding Connectors

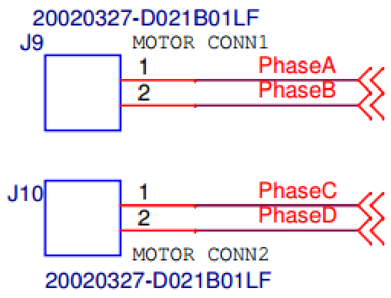
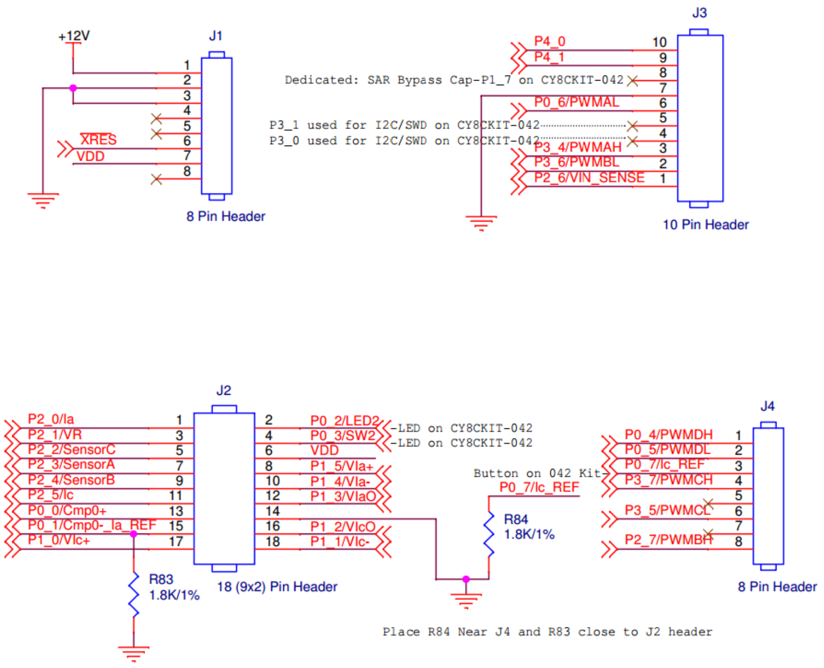


Figure 4-15 shows the connectors to the CY8CKIT-042 board. The layout of J1, J2, J3, and J4 on the EVK board is designed pin-to-pin with J1, J2, J3, and J4 on the Pioneer Kit, so users can plug the EVK board directly into the Pioneer Kit.

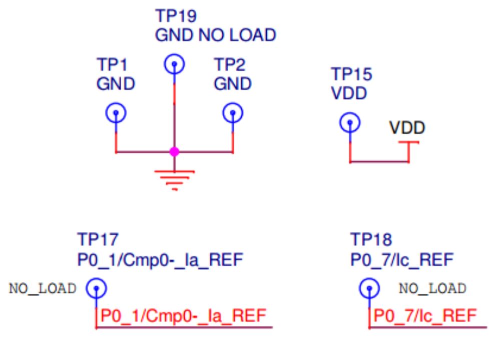
Figure 4-15. Connectors to CY8CKIT-042



4.9 Test Points

The EVK board provides many test points of critical signals, making it convenient for users to debug in the development process. Test points cover driver PWMs, power output and ground, winding current, and IDAC reference output. You can observe and monitor these signals using probes. Figure 4-16 show some instances of test points.

Figure 4-16. Test Points



CAUTION	**Do not power the kit through these test points to avoid damage to the board.**

5. Example Projects

Cypress provides five example projects with the Motor Control EVK board package, which can help you accelerate the motor control development process based on PSoC 4.

5.1 Configuration Jumpers for Different Motor Types

CY8CKIT-037 supports three motor types: BLDC, PMSM, and stepper. When switching between different motor types or algorithms, the EVK board needs to be reconfigured via jumpers J13–J24. The jumper configuration table is printed on the top layer of the EVK board, as shown in Figure 5-1.

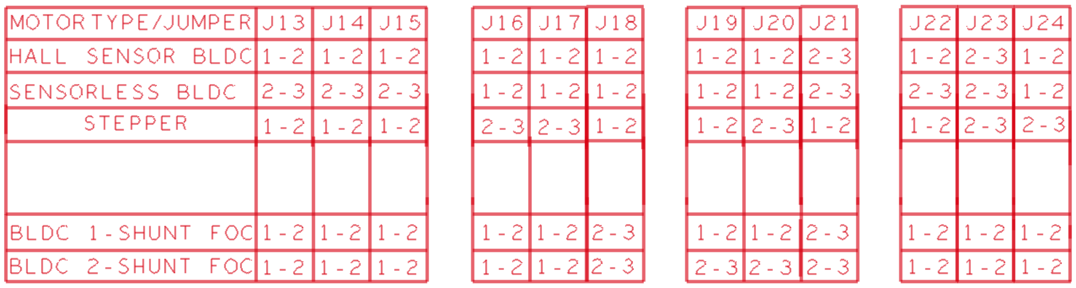


Figure 5-1. J13–J24 Configuration Table for Different Motors

MOTOR TYPE/JUMPER	J13	J14	J15	J16	J17	J18	J19	J20	J21	J22	J23	J24
HALL SENSOR BLDC	1-2	1-2	1-2	1-2	1-2	1-2	1-2	1-2	2-3	1-2	2-3	1-2
SENSORLESS BLDC	2-3	2-3	2-3	1-2	1-2	1-2	1-2	1-2	2-3	2-3	2-3	1-2
STEPPER	1-2	1-2	1-2	2-3	2-3	1-2	1-2	2-3	1-2	1-2	2-3	2-3
BLDC 1-SHUNT FOC	1-2	1-2	1-2	1-2	1-2	2-3	1-2	1-2	2-3	1-2	1-2	1-2
BLDC 2-SHUNT FOC	1-2	1-2	1-2	1-2	1-2	2-3	2-3	2-3	2-3	1-2	1-2	1-2

These jumpers are located in several different circuit parts. To learn about the principle of configuration, refer to Kit Hardware Schematic Details.

Note: When you prepare to change the configuration jumper settings for different motor types, ensure that all power sources are turned off; otherwise, the kits may be damaged.

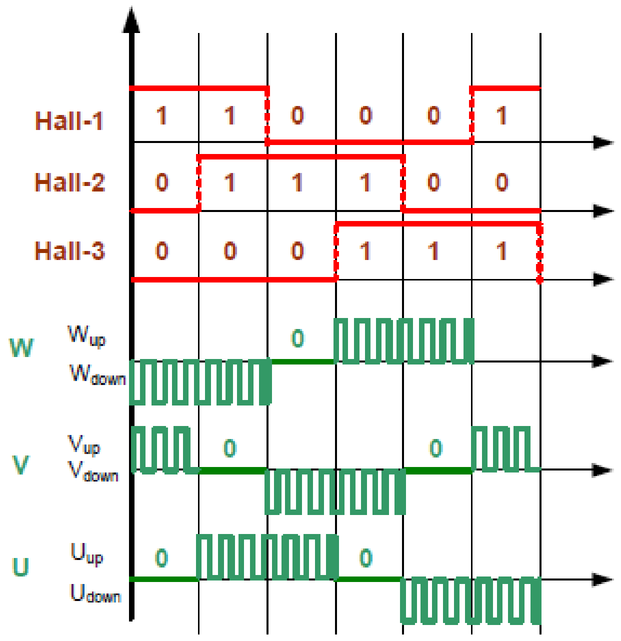
5.2 Sensored BLDC Motor Control Example Project

5.2.1 Sensored BLDC Background

BLDC motors are widely used in industrial applications, home appliances, and vehicle systems. Such motors consist of a multi-pole permanent magnet placed on the rotor and several windings. In the BLDC motor, the commutation is controlled electronically. The motor requires the stator windings to be energized in a particular sequence. To implement this sequence, it is important to know the rotor position. The simplest way is to use rotor position sensors. The sensors can be optical, magnetic (Hall or magneto-resistance effect based), or inductive. The Hall sensor is selected in most applications for its high accuracy and low cost.

Hall sensors are embedded in the stator. When the rotor magnetic poles pass near the Hall sensors, they supply a high or low signal, indicating that the north or south poles are passing nearby. The rotor position is derived from the exact combination of the three Hall sensor signals. Three position sensors can provide six effective states (except 000 and 111) and separate a whole electrical space into six parts, each having a 60-degree electrical angle. Figure 5-2 gives a timing diagram of the sensor output and the required motor driving voltage. The optional use of a PWM provides speed or torque control as shown in phases A, B, and C. The duty cycle of the modulated output control signal (PWM) is varied to change the speed and torque of the motor.

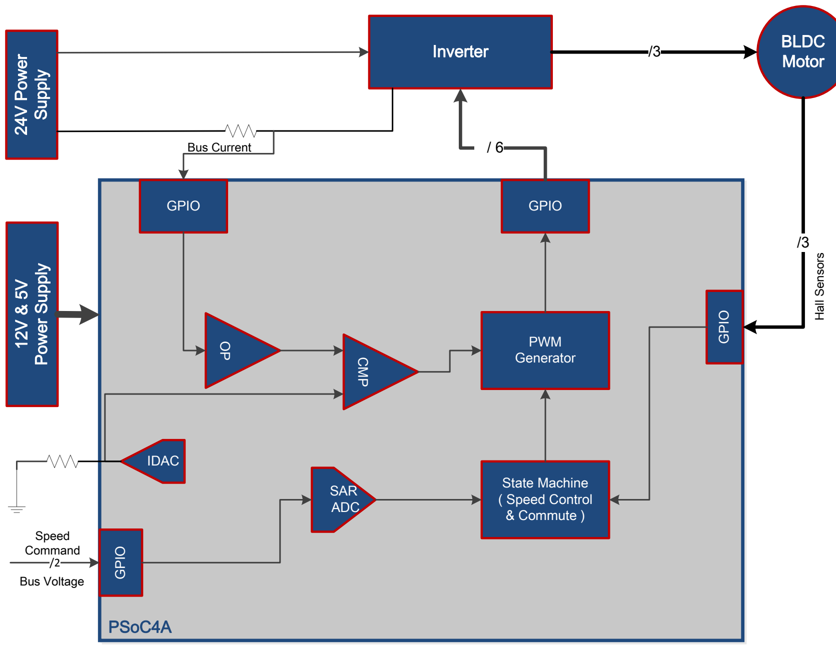
Figure 5-2. BLDC Sensor Output Versus Commutation Timing



5.2.2 Sensored BLDC Motor Control Example Project Overview

Figure 5-3 illustrates the block diagram of the Sensored BLDC Motor Control example project based on PSoC 4A.

Figure 5-3. Block Diagram of Sensored BLDC Motor Control Example Project Based on PSoC 4A



The input control signals to the PSoC 4 device are as follows.

- **Speed command:** Analog input pin that measures the voltage across a potentiometer to set the desired speed of rotation (one analog input pin).
- **Motor current detection:** Analog input pin that detects and cuts off the power device driver to protect the motor when an overcurrent condition is detected (see Firmware Introduction) (one analog input pin).

- **Hall sensors:** Three digital input pins connected to the outputs of the Hall sensors from the motor. These sensor inputs provide the position of the motor and are used to control the commutation by varying the PWM output signals to the power driver (three digital input pins).
- **Start/stop control:** Digital input connected to a switch to start and stop motor rotation (one digital input pin).

Outputs from the PSoC 4 device are power device driver signals.

- PWM signals to the high side of the MOSFET driver (three digital output pins)
- PWM signals to the low side of the MOSFET driver (three digital output pins)

5.2.3 Control Schematic Overview

The Sensored BLDC Motor Control example project firmware was developed in PSoC Creator 3.2 or later version. Its schematic was separated into two pages according to their function. Figure 5-4 shows the “PWM Drive and Commutate” page. The three Hall sensor signals are imported by pins “Hall1,” “Hall2,” and “Hall3.” Then “Lut_Cmut” outputs the PWM signal to the motor windings according to the Hall signals and its internal commutation table. “PWM_Drive” generates the real-time duty cycle to follow the user’s RPM request.

Figure 5-4. PWM Drive and Commutate Schematic

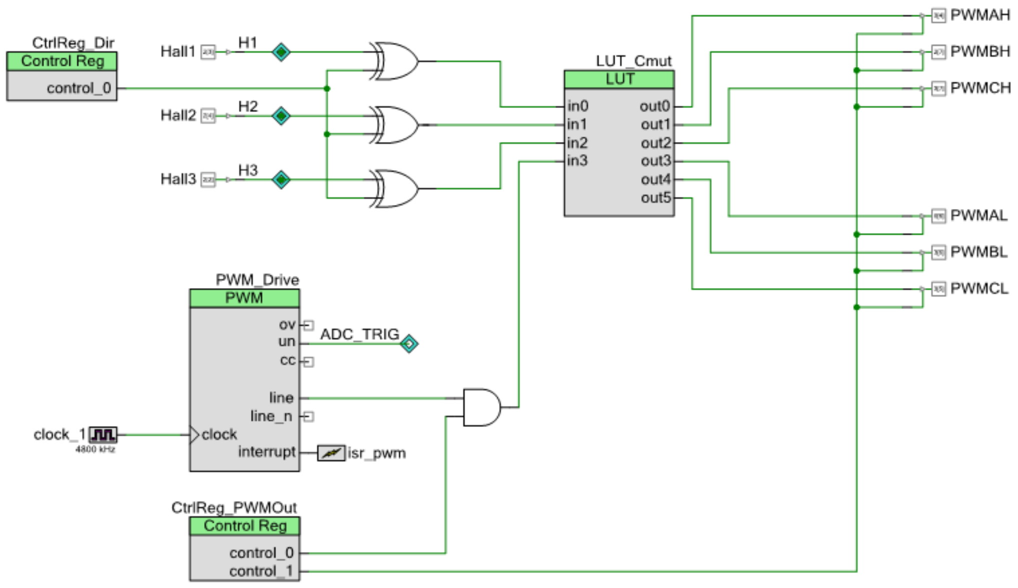


Figure 5-5 shows the “ADC sampling” schematic. “ADC_SAR_Seq_1” is the SAR ADC to detect and measure the bus voltage and potentiometer voltage.

Figure 5-5. ADC Sampling Schematic

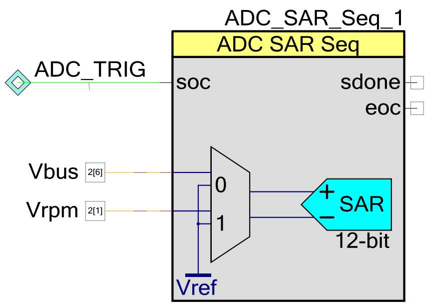


Figure 5-6 shows the “Speed Measurement” schematic. “Counter_Spd” uses one Hall signal to measure the RPM of the motor. It can work as real-time feedback of the closed-loop speed control.

Figure 5-6. Speed Measurement Schematic

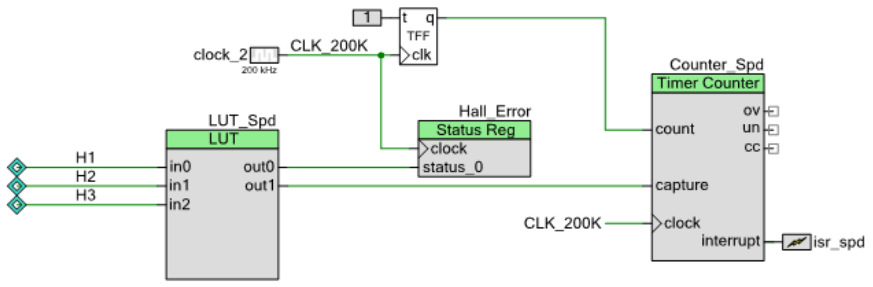
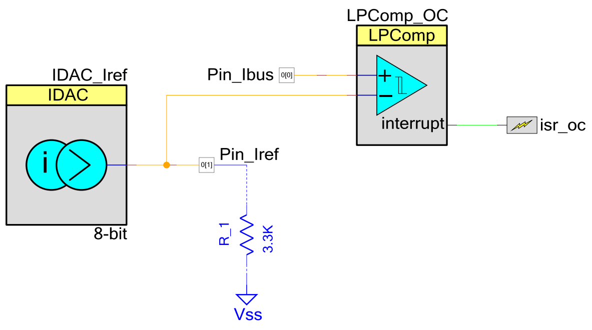


Figure 5-7 shows the "Overcurrent Protection" schematic. The motor current is detected by the sensing resistor and amplified by the external or internal opamp. The current signal is routed to the positive terminal of the internal low-power comparator "LPComp_OC." The negative terminal of the comparator is the overcurrent threshold set by the internal IDAC "IDAC_Iref." If overcurrent happens, "LPComp_OC" will trigger an interrupt, and an overcurrent action will be executed in its ISR.

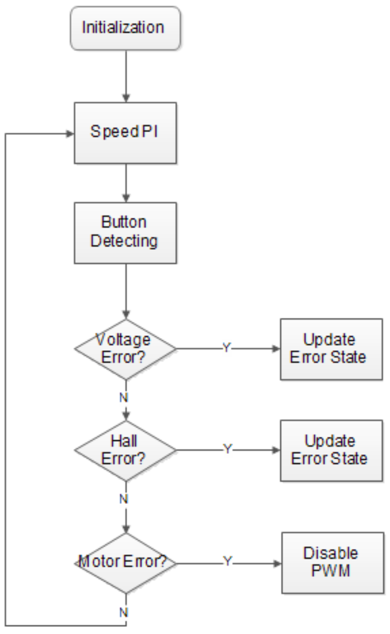
Figure 5-7. Overcurrent Protection Schematic



5.2.4 Firmware Introduction

Figure 5-8 shows the flow chart of the main loop function. The project schematic in Figure 5-4 shows that hardware performs the commutation. So, the firmware processes only the closed-loop speed control, button detecting, and protective action.

Figure 5-8. Main Loop Function Flow Chart

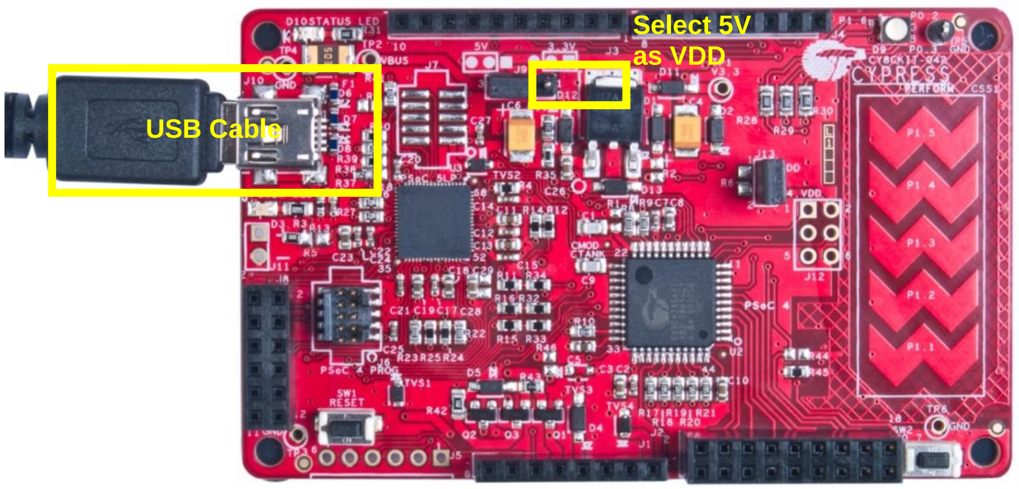


5.2.5 Running the Sensored BLDC Motor Control Example Project

5.2.5.1 Step 1 – Configure CY8CKIT-042

Select 5 V as the VDD power at jumper J9 on the Pioneer Kit, and always keep the USB cable connected to the Pioneer Kit because the Pioneer Kit does not provide a 5-V converter. It gets 5-V power directly from the USB port of the PC. Figure 5-9 shows the CY8CKIT-042 configuration.

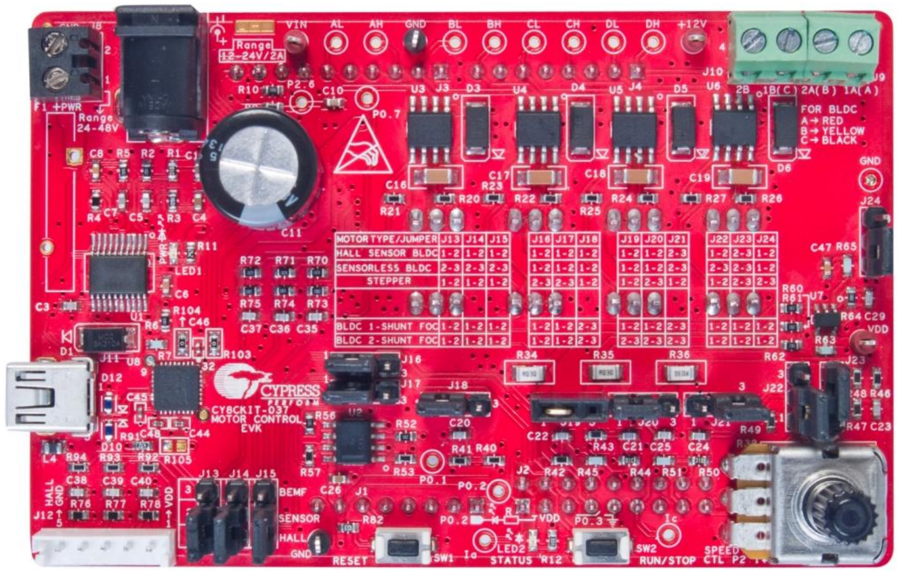
Figure 5-9. CY8CKIT-042 Configuration for Sensored BLDC Motor Control Example Project



5.2.5.2 Step 2 – Configure CY8CKIT-037

Configure the board via jumpers J13–J24 for the Sensored BLDC Motor Control example project (“HALL SENSOR BLDC” row in Figure 5-1). The CY8CKIT-037 is configured for sensored BLDC control by default. If you have not made any changes since receiving it from Cypress, bypass this step and go to Step 3 – Plug CY8CKIT-037 into CY8CKIT-042. Figure 5-10 shows the jumper overview after configuration.

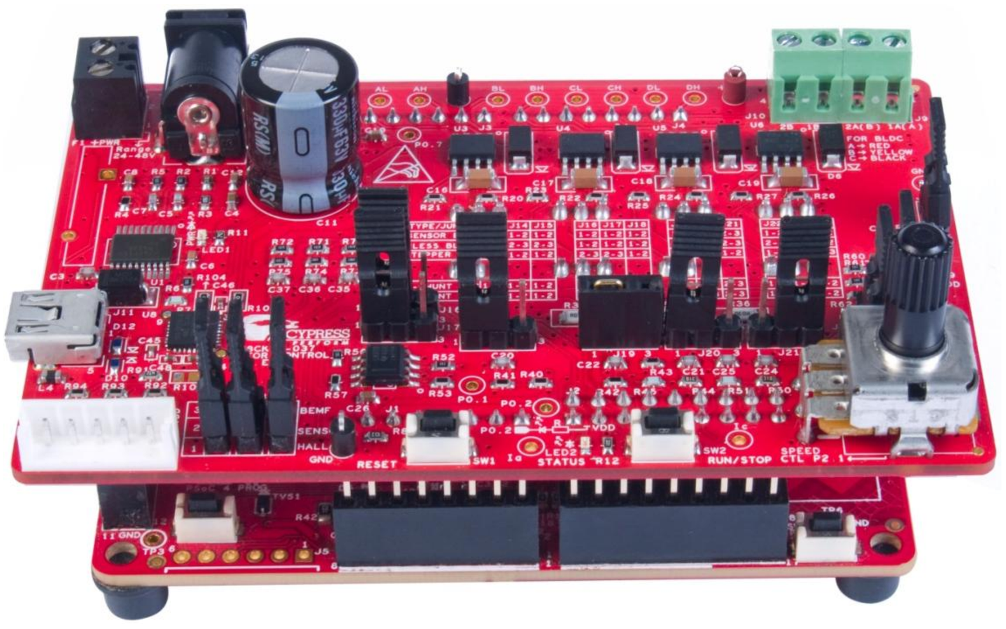
Figure 5-10. Configuration for Sensored BLDC Motor Control Example Project



5.2.5.3 Step 3 – Plug CY8CKIT-037 into CY8CKIT-042

Plug the EVK board into the Pioneer Kit via connectors J1–J4, as shown in Figure 5-11.

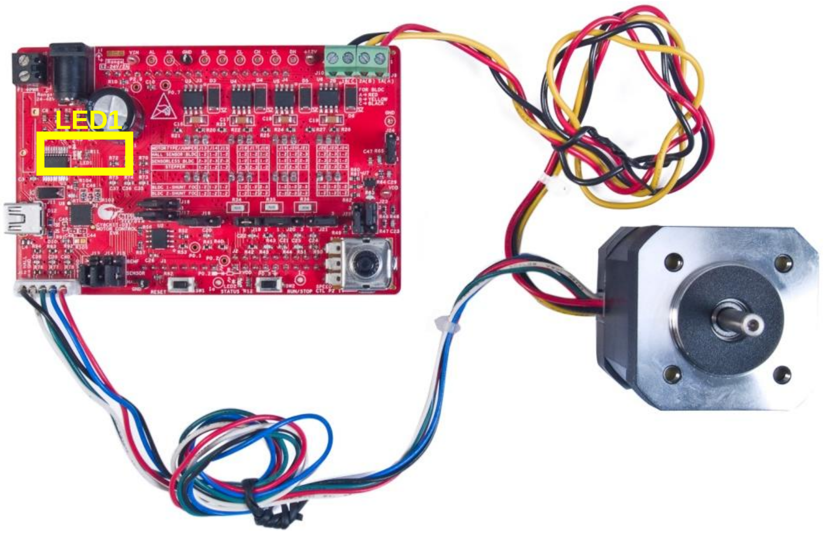
Figure 5-11. Plug CY8CKIT-037 EVK into CY8CKIT-042 Pioneer Kit



5.2.5.4 Step 4 – Connect the Power Supply and Motor

Connect the BLDC motor to the EVK board, including windings to J9/J10 and Hall sensor to J12. The wire sequence and color for the motor windings and Hall sensor should be exactly as shown in Figure 5-12. Refer to Motor Winding Connectors and Hall Sensors Connector for instructions. Then connect a 24-V power adapter to J7, and connect the other end of the USB cable to the PC. When LED1 is red, it indicates power is on. If it is not glowing, the fuse F2 may be broken. Please change the F2 with the provided backup fuse.

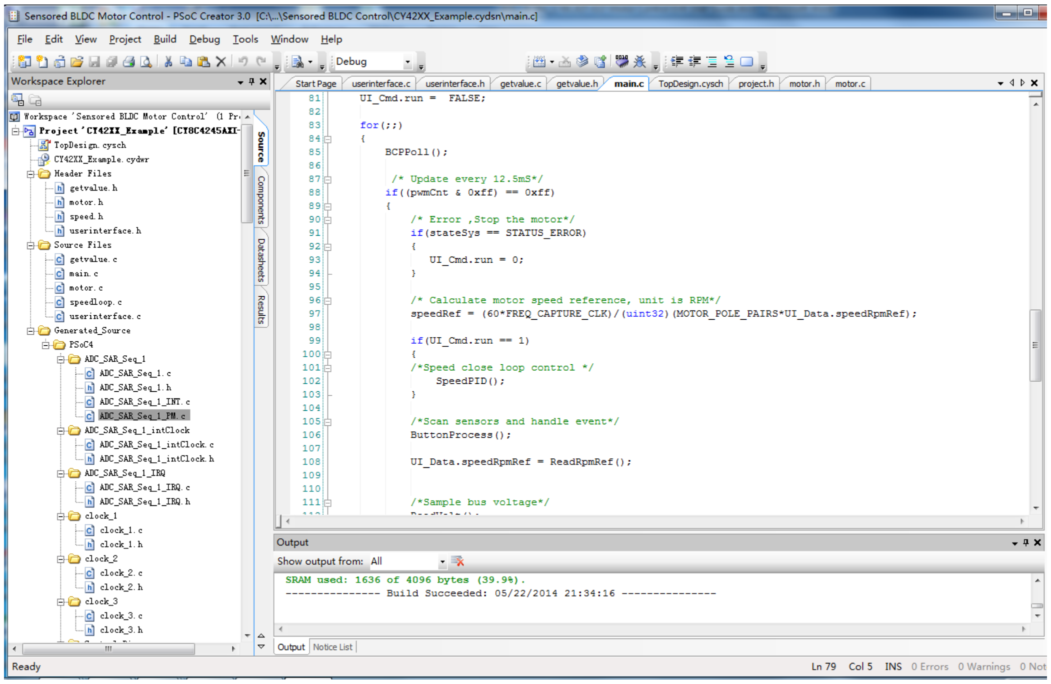
Figure 5-12. Connect Motor and Power Supply



5.2.5.5 Step 5 – Build the Project and Program PSoC 4

Open the Sensored BLDC Motor Control example project in PSoC Creator 3.2 or later version, as shown in Figure 5-13. Choose **Build** > **Build Sensored BLDC Motor Control** to start building the project, and then choose **Debug** > **Program** to program the chip.

Figure 5-13. Open the Sensored BLDC Motor Control Example Project



Sensored BLDC Motor Control - PSoC Creator 3.0 [C:\...\Sensored BLDC Control\CY42XX_Example.cydsn\main.c]

```
Workspace 'Sensored BLDC Motor Control' (1 Pr
Project 'CY42XX_Example' [CY8C4245AXI-
    TopDesign.cysch
    CY42XX_Example.cydwr
    Header Files
        getvalue.h
        motor.h
        speed.h
        userinterface.h
    Source Files
        getvalue.c
        main.c
        motor.c
        speedloop.c
        userinterface.c
    Generated_Source
        PSoC4
            ADC_SAR_Seq_1
                ADC_SAR_Seq_1.c
                ADC_SAR_Seq_1.h
                ADC_SAR_Seq_1_INT.c
                ADC_SAR_Seq_1_PM.c
            ADC_SAR_Seq_1_intClock
                ADC_SAR_Seq_1_intClock.c
                ADC_SAR_Seq_1_intClock.h
            ADC_SAR_Seq_1_IRQ
                ADC_SAR_Seq_1_IRQ.c
                ADC_SAR_Seq_1_IRQ.h
            clock_1
                clock_1.c
                clock_1.h
            clock_2
                clock_2.c
                clock_2.h
            clock_3
                clock_3.c
                clock_3.h
```

```c
    UI_Cmd.run =  FALSE;

    for(;;)
    {
        BCPPoll();

         /* Update every 12.5mS*/
        if((pwmCnt & 0xff) == 0xff)
        {
            /* Error ,Stop the motor*/
            if(stateSys == STATUS_ERROR)
            {
               UI_Cmd.run = 0;
            }

            /* Calculate motor speed reference, unit is RPM*/
            speedRef = (60*FREQ_CAPTURE_CLK)/(uint32)(MOTOR_POLE_PAIRS*UI_Data.speedRpmRef);

            if(UI_Cmd.run == 1)
            {
            /*Speed close loop control */
                SpeedPID();
            }

            /*Scan sensors and handle event*/
            ButtonProcess();

            UI_Data.speedRpmRef = ReadRpmRef();


            /*Sample bus voltage*/
```

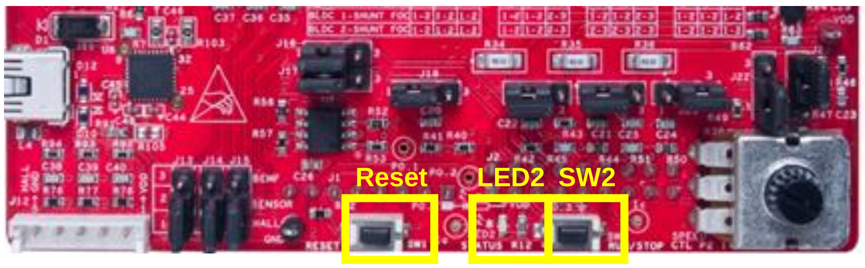
```
SRAM used: 1636 of 4096 bytes (39.9%).
--------------- Build Succeeded: 05/22/2014 21:34:16 ---------------
```

CAUTION	**During the debugging process, if you modify the code in the firmware, make sure that the changes do not turn on both the high-side and low-side MOSFETs.**

5.2.5.6 Step 6 – Press the SW2 Button to Start Motor Rotation

Ensure that the other end of the USB cable is connected to the PC in this step; otherwise, the motor will not start rotating. Press the SW2 button to start motor rotation (refer to Figure 5-14). If the motor does not rotate and you observe LED2 blinking, an error has occurred. If so, first check that step 1 through step 5 have executed correctly. Then press the Reset button and press the SW2 button again. If LED2 still blinks, there must be a problem in the hardware or software. You have to debug it or contact Cypress for technical support.

Figure 5-14. Buttons and Status LED



CAUTION	**Do not remove jumpers while the kit is powered. This may damage the kit.**

5.2.6 Adapting the Example Project to another Motor

This example project can be changed to rotate other sensored BLDC motors. Some parameters in the project firmware need to be modified according to your specific motor characteristic and functional requirements. These parameters are defined as a `struct` in *motor.h*.

```
typedef struct
{
   uint8    Dir; /*Direction*/

   uint16   speedRpm; /* Actural motor speed*/
   uint16   speedRpmRef; /* motor speed command value*/

   uint16   kp;    /* Proportional coeffient of PID*/
   uint16   ki;    /* Integral coeffient of PID*/

   uint16   maxSpeedRpm;   /* Motor parameters*/
   uint16   minSpeedRpm;   /* Motor parameters*/
   uint8    polePairs;         /*POLE PAIRS*/
   uint8    maxCurr;     /*Over current threshold*/
}UI_DATA;
```

The `struct` variable is in *motor.c* and is initialized in *motor.c* by function `Init_UI_FW`.

```
UI_DATA UI_Data;
```

```
void Init_UI_FW(void)
{
    /* Setting UI Initial parameter*/
  UI_Data.Dir = CLOCK_WISE;
  UI_Data.maxSpeedRpm = 4000;
  UI_Data.minSpeedRpm = 500;
  UI_Data.speedRpmRef = 1000;
  UI_Data.polePairs = 4;
  UI_Data.maxCurr = MAX_CURR_MEDIUM;
  UI_Data.kp = 500;
  UI_Data.ki = 50;

}
```

This function initializes the parameters before the motor begins running, and the initializing value is dedicated for the motor shipped from Cypress in the kit package. If you want to use your own motor and have different functional requirements such as RPM or direction, change the initializing value in this function.

5.3 Sensorless BLDC Motor Control Example Project

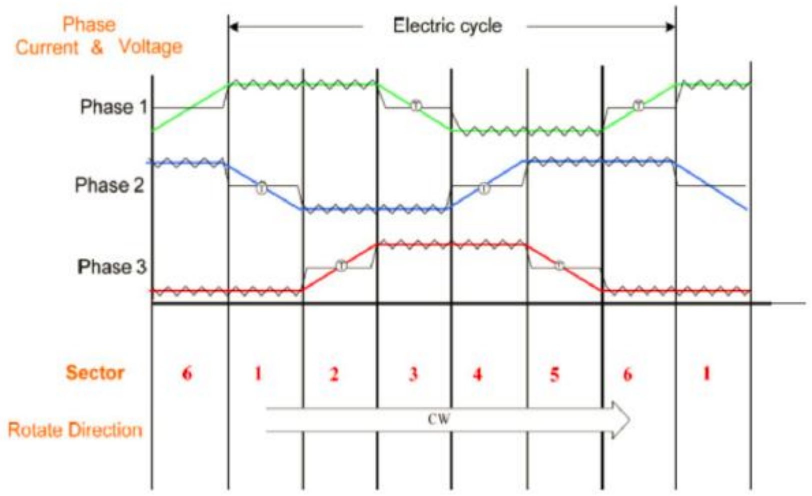
5.3.1 Sensorless BLDC Background

Various methods can be employed to control the BLDC motor. The simplest way is to use Hall position sensors. However, sensors increase cost and add reliability problems in motors operating in harsh environments where demands for sensor robustness are high. The increasing power of embedded computing, coupled with lower prices for power semiconductors and microcontrollers, has allowed more sophisticated methods of motor control. One popular technique is to use a BEMF signal, which is induced by revolving the rotor permanent magnet around the drive coils.

Most BLDC motors have a three-phase winding topology with a star connection. It is driven by energizing two phases simultaneously, while the other phase is kept afloat. The key to BLDC commutation is to sense the rotor position and then energize the phases that produce the maximum amount of torque. The rotor travels 60 electrical degrees at every commutation step. The appropriate stator current path is activated when the rotor is 120 degrees away from alignment with the corresponding stator magnetic field. It is then deactivated when the rotor is 60 degrees from alignment. The next circuit is activated and the process repeats.

Figure 5-15 shows that every commutation sequence has one winding energized positive, the second is negative, and the third winding is left open. The voltage polarity of BEMF crosses from positive to negative or from negative to positive (zero-crossing) between two commutations. Ideally, the zero-crossing of BEMF occurs at 30 electrical degrees after the last commutation and 30 electrical degrees prior to the next commutation. By measuring the zero-crossing of BEMF and the 30-degree time interval, the controller can perform the commutation without a position sensor.

Figure 5-15. General Operating Waveforms of Phase Voltages and Currents



5.3.2 Sensorless BLDC Motor Control Example Project Overview

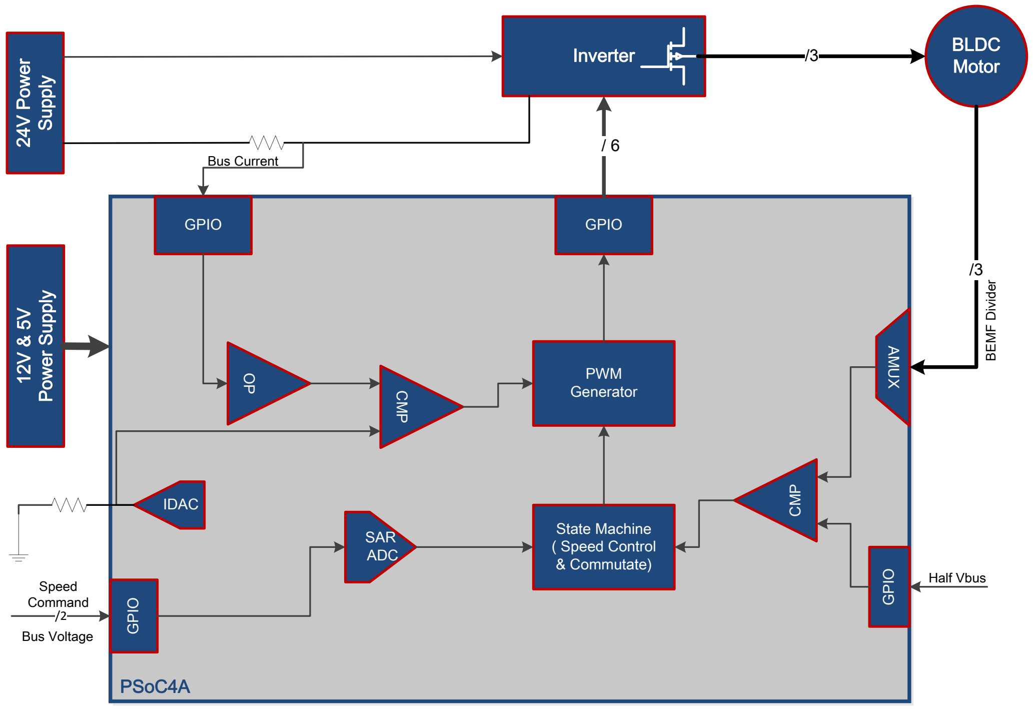
Figure 5-16 illustrates the block diagram of the Sensorless BLDC Motor Control example project based on PSoC 4A. The input control signals to the PSoC 4 device are the following.

- **Speed command:** An analog input pin that measures the voltage across a potentiometer to set the desired speed of rotation (one analog input pin).
- **Motor current detection:** An analog input pin to detect and cut off the power device driver to protect the motor when an overcurrent condition is detected (see Control Schematic Overview) (one analog input pin).
- **Half Vbus:** An analog input pin that routes half-bus voltage to the negative terminal of the internal low-power comparator.
- **BEMF divider voltage:** Three analog input pins connected to the outputs of the BEMF divider circuit. These BEMF inputs are routed to the internal AMUX, and then the AMUX selects the proper BEMF signal sequentially to compare with the half Vbus to get the zero-crossing point of the non-energized winding.
- **Start/stop control:** A digital input connected to a switch to start and stop rotation of the motor (one digital input pin).

Outputs from PSoC 4 are power device driver signals.

- PWM signals to the high side of the MOSFET driver (three digital output pins)
- PWM signals to the low side of the MOSFET driver (three digital output pins)

Figure 5-16. Sensorless BLDC Motor Control Example Project Block Diagram



5.3.3 Control Schematic Overview

The Sensorless BLDC Motor Control example project firmware was developed in PSoC Creator 3.2. Its schematic was separated into two pages according to their function. Figure 5-17 shows the “BEMF Checking” page. Three BEMF signals are routed to the internal AMUX “AMux_1.” It will select a nonenergized AMUX and compare it with the half Vbus. If zero-crossing output happens, the internal low-power comparator “BEMF_Comp” will toggle its output. The logic gating circuit on the right side of “BEMF_Comp” generates a pulse when “BEMF_Comp” toggles its output. So PSoC 4 can get a pulse at each zero-crossing point. The pulse triggers an interrupt, and the relevant code will be executed in its ISR.

Figure 5-17. BEMF Checking Schematic

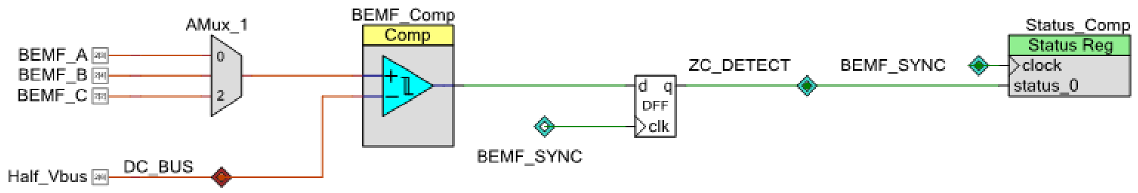
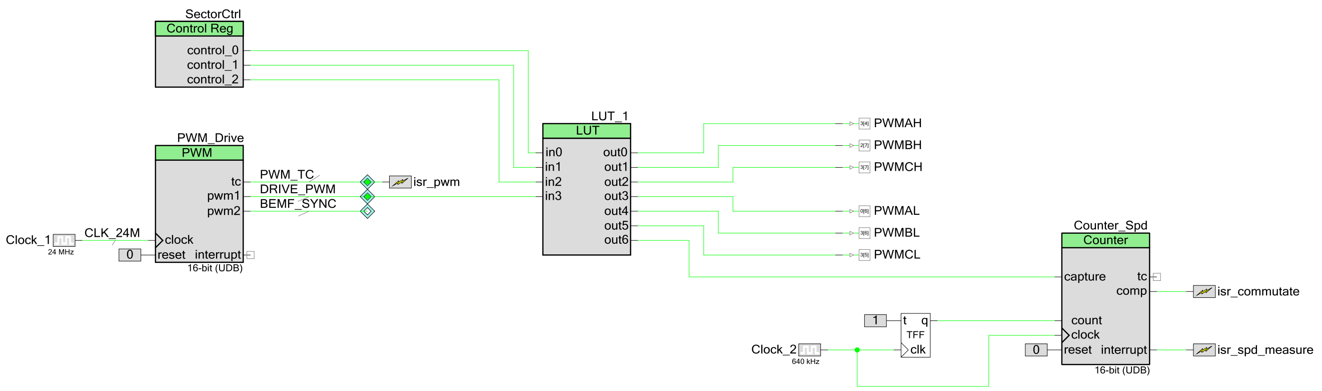


Figure 5-18 shows the “PWM Drive and Commutation” page. “SectorCtrl” indicates the current rotor position. It is dynamically changed in firmware by the zero-crossing detecting ISR. “Lut_Cmut” outputs the PWM signal to the motor windings according to the “SectorCtrl” output and its internal commutation table. “PWM_Drive” generates the right duty cycle to follow the user’s RPM request. “Counter_Spd” is used to measure motor speed; it can work as real-time feedback of the closed-loop speed control.

Figure 5-18. PWM Drive and Commutation Schematic



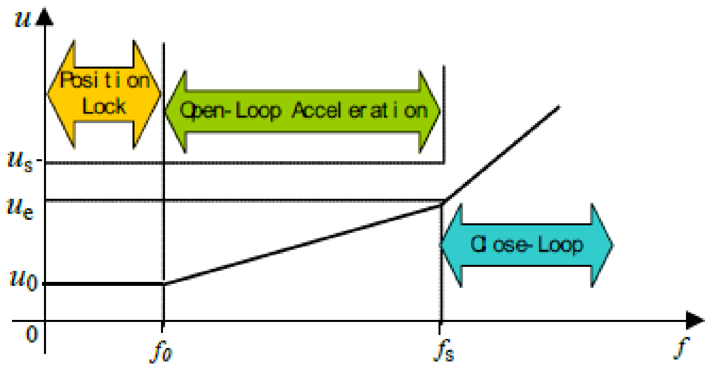
5.3.4 Firmware Introduction

The Sensorless BLDC Motor Control example project employs the typical three-step startup algorithm. The BEMF amplitude of BLDC is proportional to the motor speed. If the motor stops, there is no BEMF. When the motor rotates at a low speed, the BEMF is too weak to be detected. Therefore, before the BEMF can be measured by the firmware, there is a “free-running” stage in which the BLDC motor rotates step by step to acquire the initial BEMF zero-crossing signal before running sensorless control.

When rotating at a certain speed with open-loop stepping stimulation, the rotor position is approximately 90 electrical degrees ahead of that when run correctly under sensorless control. As a result, the BEMF zero-crossings cannot be sensed. To observe the zero-crossings, it is necessary to accelerate the motor at a certain rate.

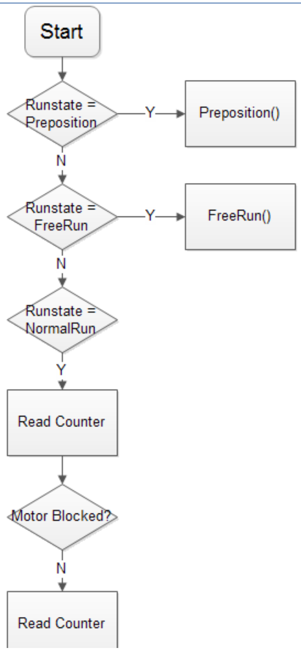
In this stage, the PWM duty cycle increases gradually, and the commutating period is longer than usual. A ramp-up table in F/W defines the timeout of every commutating period. The value of the table content is a result of experiments based on different motor types. When sufficient valid zero-crossing events are detected, F/W enters the normal synchronous running stage. If F/W cannot detect sufficient valid zero-crossing events during a period of time, an error occurs and BLDC must stop and wait for the next round of the free-running stage. Figure 5-19 shows the starting sequence.

Figure 5-19. Starting Sequence for Sensorless BLDC Motor



The main part of the sensorless startup and normal running control algorithm is in the PWM ISR. Figure 5-20 shows a flow chart of the PWM ISR.

Figure 5-20. PWM ISR Flow Chart



5.3.5 Running the Sensorless BLDC Motor Control Example Project

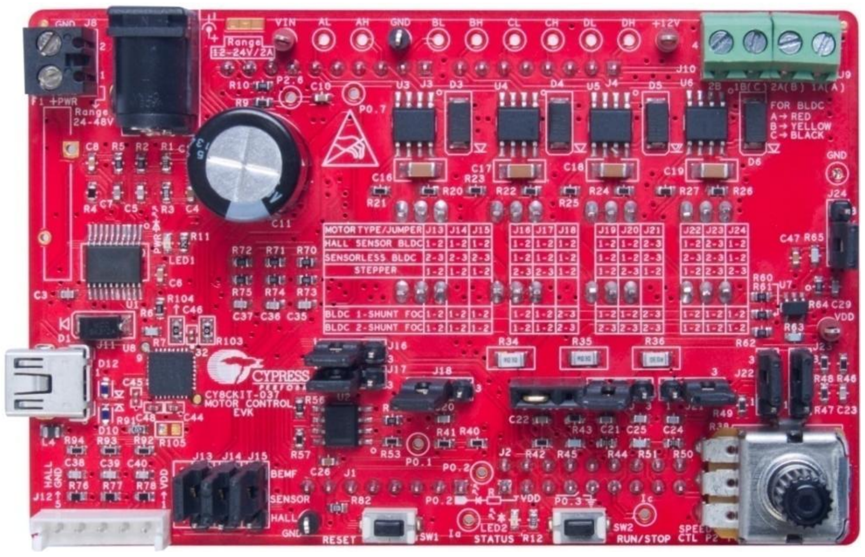
5.3.5.1 Step 1 – Configure CY8CKIT-042

Select 5 V as the VDD power at jumper J9 on the Pioneer Kit, and always keep the USB cable connected to the Pioneer Kit because the Pioneer Kit does not provide a 5-V converter. It gets 5-V power directly from the USB port of the PC. Figure 5-9 shows the configuration.

5.3.5.2 Step 2 – Configure CY8CKIT-037

Configure the board via jumpers J13–J24 for the Sensorless BLDC Motor Control example project (“SENSORLESS BLDC” row in Figure 5-1). Figure 5-21 shows the EVK board configured for the Sensorless BLDC Motor Control example project.

Figure 5-21. EVK Board Configuration for Sensorless BLDC Motor Control Example Project



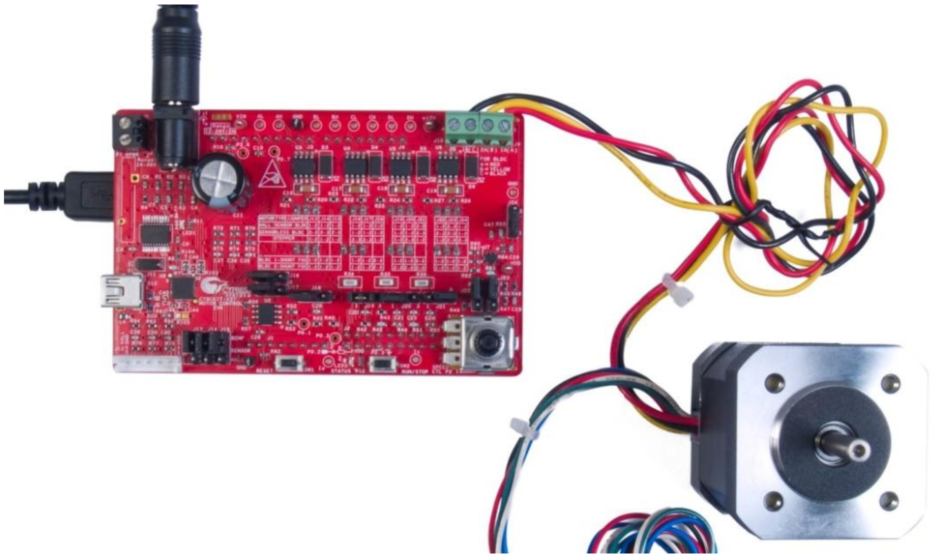
5.3.5.3 Step 3 – Plug CY8CKIT-037 into CY8CKIT-042

Plug the EVK board into the Pioneer Kit via connectors J1–J4, as shown in Figure 5-11.

5.3.5.4 Step 4 – Connect the Power Supply and Motor

Connect the BLDC windings to J9 and J10 on the EVK board. Then connect the 24-V power adapter to J7, and connect the other end of the USB cable to the PC, as shown in Figure 5-22. When LED1 is red, it indicates power is on. If it is not glowing, the fuse F2 may be broken. Please change the F2 with the provided backup fuse.

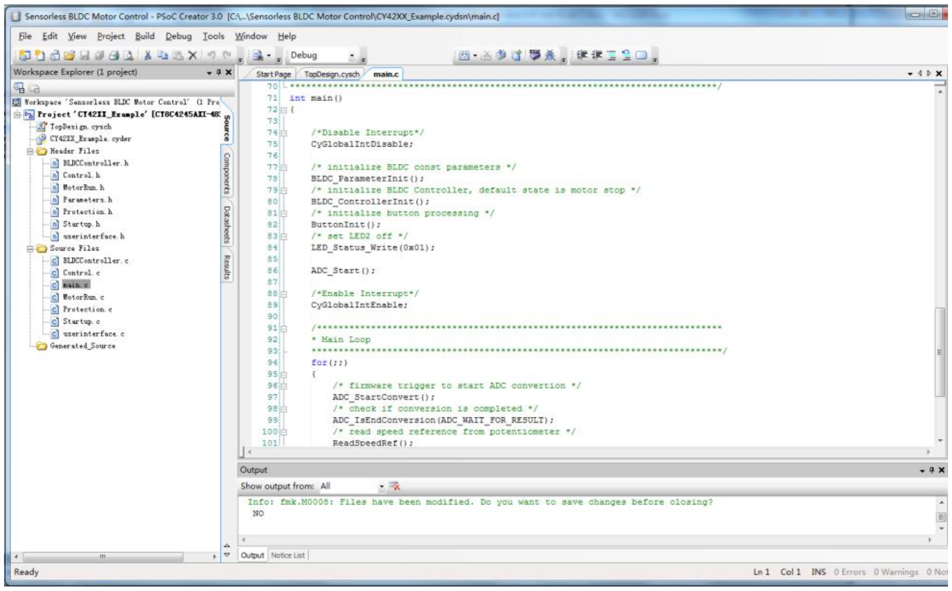
Figure 5-22. Connect Motor and Power Supply



5.3.5.5 Step 5 – Build the Project and Program PSoC 4

Open the Sensorless BLDC Motor Control example project in PSoC Creator 3.2 or later version, as shown in Figure 5-23. Choose **Build** > **Build Sensorless BLDC Motor Control** to start building the project, and then choose **Debug** > **Program** to program the chip.

Figure 5-23. Open the Sensorless BLDC Motor Control Example Project



```
*****************************************************************************/
int main()
{

    /*Disable Interrupt*/
    CyGlobalIntDisable;

    /* initialize BLDC const parameters */
    BLDC_ParameterInit();
    /* initialize BLDC Controller, default state is motor stop */
    BLDC_ControllerInit();
    /* initialize button processing */
    ButtonInit();
    /* set LED2 off */
    LED_Status_Write(0x01);

    ADC_Start();

    /*Enable Interrupt*/
    CyGlobalIntEnable;

    /*************************************************************************
    * Main Loop
    *************************************************************************/
    for(;;)
    {
        /* firmware trigger to start ADC convertion */
        ADC_StartConvert();
        /* check if conversion is completed */
        ADC_IsEndConversion(ADC_WAIT_FOR_RESULT);
        /* read speed reference from potentiometer */
        ReadSpeedRef();
```

```
Info: fmk.M0008: Files have been modified. Do you want to save changes before closing?
 NO
```

CAUTION	**During the debugging process, if you modify the code in the firmware, make sure that the changes do not turn on both the high-side and low-side MOSFETs.**

5.3.5.6 Step 6 – Press the SW2 Button to Start Motor Rotation

Ensure that the other end of the USB cable is connected to the PC in this step; otherwise, the motor will not start rotating. Press the SW2 button to start motor rotation (refer to Figure 5-14). If the motor does not rotate and you observe LED2 blinking, it indicates that an error has occurred. If so, first check that step 1 through step 5 have executed correctly. Then press the Reset button and press the SW2 button again. If LED2 still blinks, there must be a problem in the hardware or software. You have to debug it or contact Cypress for technical support.

CAUTION **Do not remove jumpers while the kit is powered.**

5.3.6 Adapting the Example Project to another Motor

This example project can be changed to rotate other sensorless BLDC motors. Some parameters in the project firmware need to be modified according to your specific motor characteristic and functional requirements.

5.3.6.1 Tuning Parameters Overview

This example project provides two structure definitions for tuning purposes. One structure is “BLDC_Config_T,” which is defined in the *parameters.h* file. It contains all the constant parameters used in BLDC rotation. The following code illustrates the members of the “BLDC_Config_T” structure.

```
typedef struct _BLDC_Config
{
 /*-------------------------------------------------------------*
  * motor parameters                                            *
  *-------------------------------------------------------------*/
    uint8   polePairNumber;
 /*-------------------------------------------------------------*
  * general parameters                                          *
  *-------------------------------------------------------------*/
    Direction_T   direction;
    uint16  initSpeedRefRpm;
 /*-------------------------------------------------------------*
  * PID parameters                                              *
  *-------------------------------------------------------------*/
   Uint32  kp;
   Uint32  ki;
 /*-------------------------------------------------------------*
  * preposition parameters                                      *
  *-------------------------------------------------------------*/
   uint16  prepositionTime;
   uint16  prepositionDuty;
 /*-------------------------------------------------------------*
  * startup parameters                                          *
  *-------------------------------------------------------------*/
   uint16  startDuty;
```

```
    uint16  freerunDuty;
    uint16  startCheckPeriod;

    uint8   startStageWait;
    uint8   decStageInterval;

    uint8   freerunStageWait;
    uint8   accStageInterval;

    uint8   accStageWait;
    uint8   accDutyStep;

    uint8   accTimeStep;
/*--------------------------------------------------------------*
 * normal run parameters                                        *
 *--------------------------------------------------------------*/
    uint8   speedCloseLoopWait;
    uint8   zcCheckSkipCount;
}BLDC_Config_T;
```

A variable of “BLDC_Config_T” is defined in the *BLDCController.c* file as follows.

```
BLDC_Config_T    BLDC_Config;
```

The parameters in “BLDC_Config_T” are initialized in the `BLDC_ParameterInit` function (see the following code), which is executed only once when the system powers up. Generally, these parameters should not be changed during motor rotation, and a number of macros are also defined in the *parameters.h* file to provide the initial value. You can modify the macro values for tuning based on application requirements or different motors. Note that macro values should be modified before the code example is compiled. Once the project is compiled for the binary hex file, changing macro values has no effect.

However, if some additional debugging code is developed to support run-time tuning via the communication interface, such as RS232, SPI, or I^2C, real-time updating of the parameters in the “BLDC_Config” variables is possible. Note that the Cypress code example does not include this kind of debugging code. Kit users must develop it themselves.

```
void BLDC_ParameterInit(void)
{
    /* motor pole-pair number */
    BLDC_Config.polePairNumber = MOTOR_POLE_PAIR_NUM;

    /* motor rotation direction, options are CLOCK and COUNTER_CLOCK */
    BLDC_Config.direction = CLOCK;
    /* speed reference in RPM unit */
    BLDC_Config.initSpeedRefRpm = INIT_SPEED_REF_RPM;
    /* kp for PID control */
    BLDC_Config.kp = KP_INITIAL_VALUE;
    /* ki for PID control */
    BLDC_Config.ki = KI_INITIAL_VALUE;

    BLDC_Config.startDuty = START_PWM_DUTY;
    BLDC_Config.freerunDuty = FREERUN_PWM_DUTY;
    BLDC_Config.startCheckPeriod = START_ZC_CHECK_PERIOD;
    BLDC_Config.startStageWait = START_STAGE_WAIT_COUNT;
    BLDC_Config.decStageInterval = DEC_STAGE_INTERVAL;
    BLDC_Config.freerunStageWait = FREERUN_STAGE_WAIT_COUNT;
    BLDC_Config.accStageInterval = ACC_STAGE_INTERVAL;
    BLDC_Config.accStageWait = ACC_STAGE_EXEC_COUNT;
    BLDC_Config.accDutyStep = ACC_STAGE_DUTY_STEP;
    BLDC_Config.accTimeStep = ACC_STAGE_TIME_STEP;

    BLDC_Config.prepositionTime = PREPOSITION_WAIT_TIME;
    BLDC_Config.prepositionDuty = PREPOSITION_PWM_DUTY;

    BLDC_Config.speedCloseLoopWait = SPEED_CLOSE_LOOP_DELAY;
    BLDC_Config.zcCheckSkipCount = ZC_CHECK_SKIP_COUNT;
}
```

Another structure is “BLDC_Control_T,” which is defined in the *BLDCController.h* file. It contains the system running status, such as the current running speed in RPM, the output of PID control, and so on. This structure is updated in real time during motor rotation. Monitoring these values is helpful in understanding system performance and troubleshooting. The following code illustrates the members of the “BLDC_Control _T” structure.

```
typedef struct _Run_Control
{
    uint8           runFlag;
    Error_Code_T    errorCode;

    uint8           sector;
    uint8           checkFallingEdge;
    uint8           inNormalRun;

    uint16          speedMeasuredRpm;
    uint16          speedRefRpm;
    uint16          speedGivenRpm;
    uint16          commutateStamp;
    uint16          zeroCrossStamp;
    uint16          zeroCrossPeriod;
    uint16          delayTime;

    int32           pidOutput;
    int32           pidSpeedErr;
}BLDC_Control_T;
```

A variable of “BLDC_Control_T” is defined in the *BLDCController.c* file as follows.

```
BLDC_Control_T   BLDC_Control;
```

A macro (see the following code) is defined to reset the values of “BLDC_Control” in the *BLDCController.c* file. This macro is executed in the `BLDC_ControllerInit` function when the system powers up and in the `BLDC_Start` function before the motor starts rotation every time.

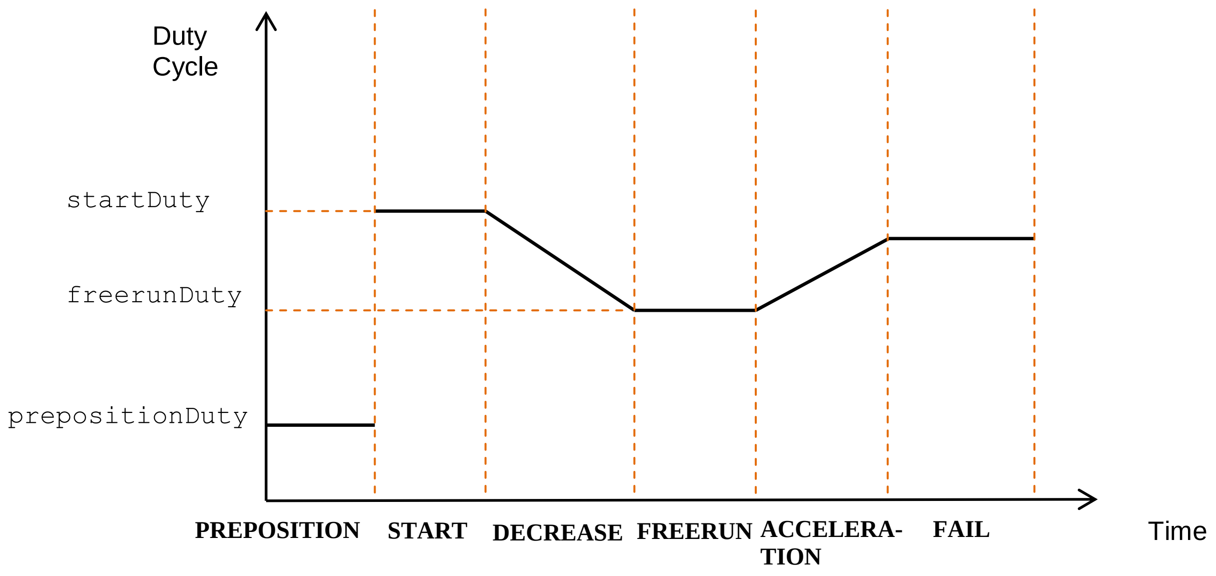
```
#define BLDC_CONTROL_INIT_CODE              {\
    BLDC_Control.runFlag = FALSE;\
    BLDC_Control.errorCode = NO_ERROR;\
    BLDC_Control.sector = 1;\
    BLDC_Control.checkFallingEdge = FALSE;\
    BLDC_Control.inNormalRun = FALSE;\
    BLDC_Control.speedMeasuredRpm = 0;\
    BLDC_Control.speedRefRpm = 0;\
    BLDC_Control.speedGivenRpm = 0;\
    BLDC_Control.commutateStamp = 0;\
    BLDC_Control.zeroCrossStamp = 0;\
    BLDC_Control.zeroCrossPeriod = 0;\
    BLDC_Control.delayTime = 0;\
    BLDC_Control.pidOutput = 0;\
    BLDC_Control.pidSpeedErr = 0;\
}
```

5.3.6.2 Tuning Startup Procedure

The startup procedure is important for the sensorless BLDC application. At the beginning of startup, the BEMF is not high enough to detect a stable zero-crossing point, which is used in sensorless control for commutation. The BLDC motor must rotate at a certain speed to generate a high enough BEMF in open-loop mode for a specified time. Then the system switches to position closed-loop mode for zero-crossing point tracking and commutation on demand. After position closed-loop mode is stable, the system enables speed closed-loop mode to force motor rotation close to a given speed reference. Once speed closed-loop mode is stable, the BLDC motor works in normal run mode.

The tuning startup procedure is performed for many possible reasons. One is that a different startup load is applied to the motor; another is that a changing motor requires performance tuning to match the new motor characteristic. The code example provides a multistage startup procedure, as shown in Figure 5-24. This procedure contains six stages, and the PWM duty cycle varies for the different stages.

Figure 5-24. Multistage Startup Procedure



A firmware counter is used to detect the zero-crossing for each stage. Stage switching happens only when the firmware counter is timed out. “startCheckPeriod” sets the timeout period for the firmware counter.

```
uint16  startCheckPeriod;
```

Stage PREPOSITION aligns the motor rotor to a given position for a while and prepares for starting rotation. Two parameters are tunable for this stage. “prepositionTime” is a threshold for an internal counter that counts how many times stage PREPOSITION is executed. A larger “prepositionTime” value lengthens the duration of the PREPOSITION stage. “prepositionDuty” is the duty cycle for stage PREPOSITION. A larger “prepositionDuty” value generates a larger preposition torque.

```
uint16  prepositionTime;
uint16  prepositionDuty;
```

Stage START makes the motor rotate from the stop state. Change “startDuty” for a different load status. A larger value generates a larger startup torque for a heavy load. “startStageWait” indicates how long stage START is executed. It is a threshold for an internal counter that counts how many times stage START is executed. A larger “startStageWait” value lengthens the duration of stage START.

```
uint16  startDuty;
uint8   startStageWait;
```

Stage DECREASE follows stage START. In this stage, the duty cycle of PWM is decreased by 1 every time until the duty reaches the “freerunDuty,” which is the minimum duty cycle for the startup procedure. Then the motor maintains rotation in accordance with the “freerunDuty” duty cycle in stage FREERUN. “decStageInterval” sets the interval between two decrease operations. A larger value results in a slower slope of the decrease curve. ”freerunStageWait” indicates how long stage FREERUN is executed. It is a threshold for an internal counter that counts how many times stage FREERUN is executed. A larger “startStageWait” value lengthens the duration of stage FREERUN.

```
uint16  freerunDuty;
uint8   decStageInterval;
uint8   freerunStageWait;
```

After stage FREERUN, the motor starts to accelerate for a higher BEMF in stage ACCELERATION. “accStageInterval” sets the interval between two decrease operations. “accStageWait” indicates how long the stage ACCELERATION is executed. A larger “accStageWait” value lengthens the duration of stage ACCELERATION. “accDutyStep” sets the increase step for the duty cycle, while “accTimeStep” sets the decrease step for the period of commutation in open loop.

```
uint8   accStageInterval;
uint8   accStageWait;
uint8   accDutyStep;
uint8   accTimeStep;
```

If the system cannot detect a valid continuous zero-crossing point in the previous stages, the startup procedure is considered a failure. The system running stage is set as “FREERUNFAILED,” and errorCode is set as “FREERUN_ERROR.”

If a continuous zero-crossing is detected within a sequence of timeout events, the system switches to position closed-loop mode, no matter which stage of the procedure is occurring. The system then waits for a specified time to enable speed closed-loop mode. This waiting time is specified by “speedCloseLoopWait.” It is a threshold for an internal counter that counts how many times position closed-loop mode is executed. A larger “speedCloseLoopWait” value lengthens the duration of position closed-loop mode.

```
uint8   speedCloseLoopWait;
```

5.3.6.3 Tuning for Different BLDC Motors

When changing the motor, some motor-related parameters or macros need modification, as listed in Table 5-1.

Table 5-1. Motor-Related Parameters for Sensorless BLDC Motor Control Example Project

No.	Parameter	Location	Comment
1	`polePairNumber`	BLDC_Config_T	Pole-pair number for motor
2	`direction`	BLDC_Config_T	Desired rotation direction
3	`NORMAL_DC_VOLTAGE`	*Parameters.h*	DC bus voltage in volts
4	`MAX_SPEED_REF_RPM`	*Parameters.h*	Maximum rotation speed
5	`MIN_SPEED_REF_RPM`	*Parameters.h*	Minimum rotation speed
6	`START_PWM_DUTY`	*Parameters.h*	The duty cycle of startup. Its value depends on input voltage, motor characteristic, and load status. Generally, it is tuned based on the try-observe method. The more the load, the larger the duty cycle value.
7	`FREERUN_PWM_DUTY`	*Parameters.h*	Duty cycle for startup stage 2
8	`INIT_SPEED_REF_RPM`	*Parameters.h*	Generally, it should be 15 to 25 percent of the maximum speed if no value is specified.

5.3.6.4 Status LED Display and Error Code

Multiple error states are defined in the *BLDCController.h* file as follows.

```
typedef enum _Error_Code
{
    NO_ERROR,                   /* no error happens */
    ZC_CHECK_ERROR,             /* zero-crossing detection failure */
    OV_ERROR,                   /* overvoltage happens */
    UV_ERROR,                   /* undervoltage happens */
    FREERUN_ERROR,              /* freerun stage fails, fail to enter close 
loop */
    ANY_ERROR,                  /* for any unknown error */
    ERROR_SIZE                  /* variable to store count of error types */
}Error_Code_T;
```

A status LED is used to display the system status. When the motor is rotating normally, the LED is turned on. When the motor is stopped normally, the LED is turned off. When an error occurs, the status LED flashes. The number of flash times indicates the error code. For example, if the status LED flashes three times, the UV_ERROR has occurred.

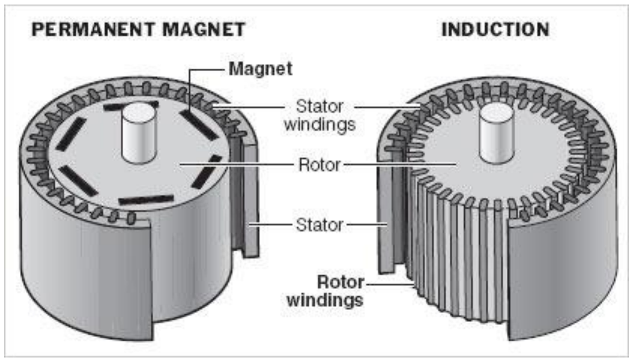
5.4 Sensorless Foc Motor Control Example Project

5.4.1 Sensorless FOC Background

5.4.1.1 PMSM Introduction

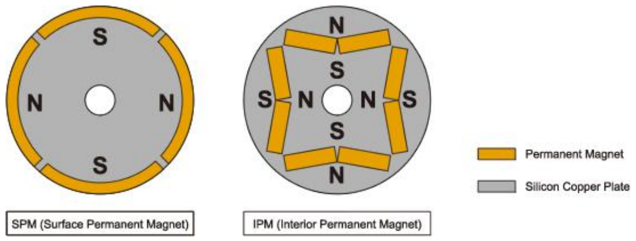
PMSM is the most popular motor type in industrial, home appliance, and automotive medium- and high-end applications. As Figure 5-25 shows, PMSM has a structure similar to that of the AC induction motor. PMSM has a wound stator and permanent magnet rotors that provide sinusoidal flux distribution in the air gap, making the BEMF inform a sinusoidal shape. The construction of the stator and rotor can provide lower rotor inertia and high power efficiency and reduce the motor size.

Figure 5-25. PMSM Structure



Depending on how magnets are attached to the rotor, PMSM motors can be classified into two types: surface PMSM (SPMSM) and interior PMSM (IPMSM). Figure 5-26 illustrates the differences between these two types. SPMSM mounts all magnet pieces on the surface, and IPMSM places magnets inside the rotor. SPMSM is widely used for simple manufacturing and assembling, while IPMSM optimizes performance by the specific placement of magnets. “PMSM” refers to “SPMSM” in the remainder of this guide, unless otherwise specified.

Figure 5-26. SPMSM and IPMSM Structural Differences


S
N
N
S
SPM (Surface Permanent Magnet)
N
S
S N N S
S
N
IPM (Interior Permanent Magnet)
Permanent Magnet
Silicon Copper Plate

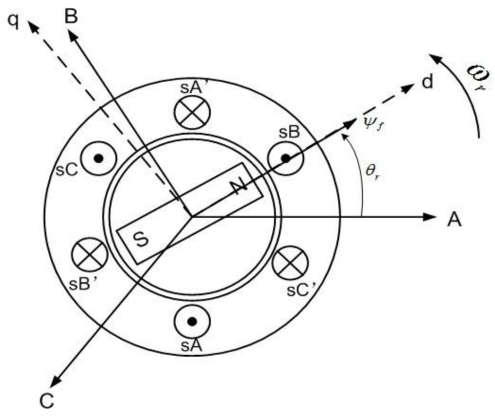
5.4.1.2 PMSM Mathematic Model

A mathematical model of PMSM motors is essential for control and performance analysis. Before discussing any model, some assumptions should be set.

- PMSM winding connection is in “Y” type; any delta connections need to convert to “Y” type.
- Magnetic saturation is neglected.
- Eddy currents and hysteresis losses are negligible.

Figure 5-27 illustrates a three-phase stator reference frame, which is the basic PMSM model. In this frame, the axis of ***A, B, C*** is aligned with the direction of the PMSM stator windings; $\boldsymbol{\Psi_f}$ is the flux linkage vector of the rotor magnet. The rotor rotates at angular speed $\boldsymbol{\omega_r}$ and generates an angle $\boldsymbol{\theta_r}$ between $\boldsymbol{\Psi_f}$ and phase ***A***.

Figure 5-27. Three-Phase Stator Reference Frame



The voltage on the stator winding can be presented as

$$\begin{bmatrix} u_a \\ u_b \\ u_c \end{bmatrix} = \begin{bmatrix} R_a & 0 & 0 \\ 0 & R_b & 0 \\ 0 & 0 & R_c \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \rho \begin{bmatrix} \Psi_a \\ \Psi_b \\ \Psi_c \end{bmatrix}$$

Where

$[u_a \quad u_b \quad u_c]^T$	Stator voltage vector
$diag[R_a \quad R_b \quad R_c]$	Stator resistance
$[i_a \quad i_b \quad i_c]^T$	Stator current vector
ρ	Differential operator d/dt
$[\Psi_a \quad \Psi_b \quad \Psi_c]^T$	Stator flux linkages

The stator winding flux linkages can be written as the sum of the flux linkages due to their own excitation, mutual flux linkages resulting from other winding currents, and flux linkages from magnet sources of the rotor. As stator windings have 120-degree angles in mechanical space, the stator flux linkages can be expressed as follows.

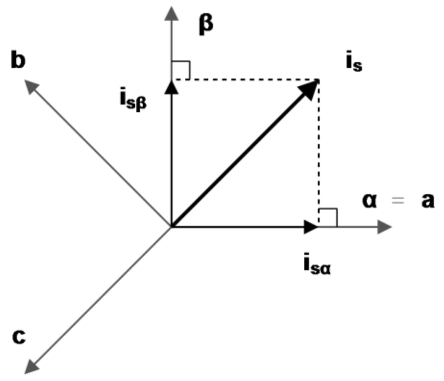
$$\begin{bmatrix} \Psi_a \\ \Psi_b \\ \Psi_c \end{bmatrix} = \begin{bmatrix} L_{aa}(\theta_r) & M_{ab}(\theta_r) & M_{ac}(\theta_r) \\ M_{ba}(\theta_r) & L_{bb}(\theta_r) & M_{bc}(\theta_r) \\ M_{ca}(\theta_r) & M_{cb}(\theta_r) & L_{cc}(\theta_r) \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \Psi_f \begin{bmatrix} \cos\theta_r \\ \cos(\theta_r - 120°) \\ \cos(\theta_r + 120°) \end{bmatrix}$$

Where

L_{aa} , L_{bb}, L_{cc}	Equivalence inductance of stator
M_{aa} ,M_{bb} ,M_{cc}	Mutual equivalence inductance of stator
Ψ_f	Amplitude of rotor flux linkages

This model is high order, strongly coupled, and nonlinear. It is difficult to analyze and control torque and flux; therefore, a two-phase d-q model is developed to simplify the three-phase model. The d-q model is based on a rotating two-phase reference frame that is aligned with the rotor flux. It first converts the three-phase reference frame *(*a, b, c*)* to a two-phase stationary reference frame *(α, β)* using the Clarke transformation (Figure 5-28).

Figure 5-28. Reference Frame Conversion Using Clarke Transformation



The current vector $[i_\alpha \quad i_\beta]^T$ in the (α, β) frame is

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \frac{2}{3}\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix}\begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix}$$

For “star” type winding connections, the sum of three-phase currents is zero.

$$i_a + i_b + i_c = 0$$

Therefore, the current vector on the α-axis and β-axis can be presented as

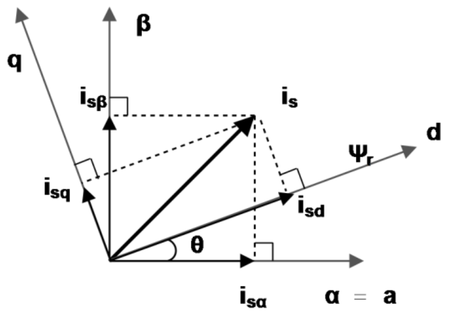
$$i_\alpha = i_a$$
$$i_\beta = \frac{1}{\sqrt{3}}(i_a + 2i_b)$$

Then a Park transformation converts the *(α, β)* frame to a rotating two-phase reference frame (d, q) aligned with the rotor flux (Figure 5-29). The current vectors in the (d, q) frame are

$$\mathbf{i_d = i_a \cos\theta + i_\beta \sin\theta}$$

$$\mathbf{i_q = -i_a \sin\theta + i_\beta \cos\theta}$$

Figure 5-29. Reference Frame Conversion Using Park Transformation



Now the voltage in the d-q frame can be calculated by i_d and i_q.

$$u_d = Ri_d + \frac{d\Psi_d}{dt} - \omega_r\Psi_q$$

$$u_q = Ri_q + \frac{d\Psi_q}{dt} + \omega_r\Psi_d$$

$$\Psi_d = L_d i_d + \Psi_f$$

$$\Psi_q = L_q i_q$$

The torque equation is

$$\begin{aligned} T_e &= \frac{3}{2} P_n(\Psi_d i_q - \Psi_q i_d) \\ &= \frac{3}{2} P_n[\Psi_f i_q - (L_q - L_d) i_d i_q] \end{aligned}$$

Where

P_n	Pole-pair number of rotor

Note that L_d and L_q have no relationship with electrical angle θ, and L_q is equal to L_d for SPMSM; therefore, the torque can be expressed as

$$T_e = \frac{3}{2} P_n \Psi_f i_q$$

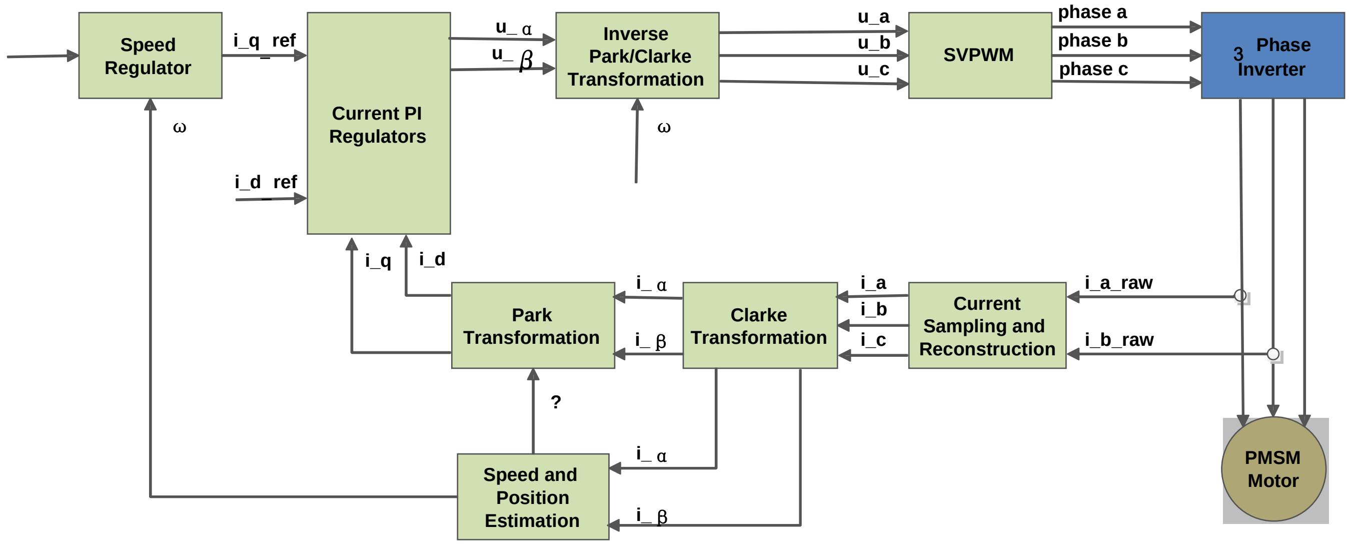
Both P_n and Ψ_f are related to motor characters, and they are not affected by motor operation. Therefore, torque is proportional only to the q-axis current i_q in the d-q model, which provides a simple method to control torque generation.

5.4.1.3 Field-Oriented Control

FOC, also called "vector control," employs the d-q model. It is a control technique used in PMSM and AC induction motors for high-performance motor applications. It can operate smoothly over a wide speed range and is capable of fast acceleration and deceleration. It can easily control the motor torque to reduce ripple and achieve smooth rotation with lower noise and vibration.

Figure 5-30 illustrates the control block diagram for FOC. Compared to the basic d-q model, it introduces four new blocks: Speed and Position Estimation, Speed Regulator, Current PI Regulators, and Space Vector PWM (SVPWM).

Figure 5-30. Sensorless FOC Block Diagram



Speed and Position Estimation samples the feedback signal from the motor and calculates the angular speed and position of the rotor. This information is employed in the Clarke and Park transformations to convert three-phase currents to i_d and i_q in the d-q model

Speed Regulator and Current PI Regulators employ control algorithms to ensure that rotation speed and d-q current remain close to the commanded values, which are set by the user or host system. The output of Current PI Regulators is transformed to three-phase voltages by inverse Clarke and Park transformation.

The SVPWM, also called "SVM," is used to generate a sinusoidal current waveform to feed to the stator coils. Suppose VT_1 through VT_6 are the six power transistors that shape the output, and VT_1, VT_3, VT_5 are the upper transistors. When an upper transistor is switched on, the corresponding lower transistor is switched off.

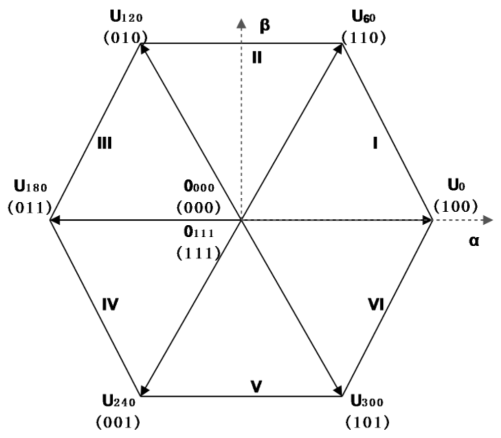
If recording the ON state of the upper transistor as"1" and the OFF state as "0", there are eight possible combinations of ON and OFF states. Corresponding to these different combinations are eight outputs, as shown in Table 5-2. u_a, u_b, u_c are the phase (line-to-neutral) voltages, while u_{ab}, u_{bc}, u_{ac} are the line-to-line voltages.

Table 5-2. Output Combination in Three-Phase Frame

ON/OFF State			Line-to-Line Voltage (Vdc)			Phase Voltage (Vdc)		
VT_1	VT_3	VT_5	u_{ab}	u_{bc}	u_{ac}	u_a	u_b	u_c
1	0	0	1	0	–1	2/3	–1/3	–1/3
1	1	0	0	1	–1	1/3	1/3	–2/3
0	1	0	–1	1	0	–1/3	2/3	–1/3
0	1	1	–1	0	1	–2/3	1/3	1/3
0	0	1	0	–1	1	–1/3	–1/3	2/3
1	0	1	1	–1	0	1/3	–2/3	1/3
0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0

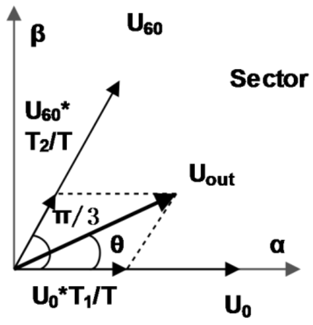
The eight combinations in Table 5-2 can be converted to six nonzero vectors and two zero vectors, as shown in Figure 5-31. The nonzero vectors form the axes of a hexagon, and the angle between any adjacent two nonzero vectors is 60 degrees. This divides the hexagon into six sectors: I through VI. The zero vectors are at the origin and apply zero voltage to a three-phase load. The eight vectors, called the “basic space vectors,” are denoted by $\boldsymbol{U_0}$, $\boldsymbol{U_{60}}$, $\boldsymbol{U_{120}}$, $\boldsymbol{U_{180}}$, $\boldsymbol{U_{240}}$, $\boldsymbol{U_{300}}$, $\boldsymbol{0_{000}}$, and $\boldsymbol{0_{111}}$.

Figure 5-31. Space Vectors



Assume $\boldsymbol{U_{out}}$ is the desired reference voltage in the (α, β) frame. The reference voltage $\boldsymbol{U_{out}}$ can be produced by a combination of basic space vectors using the SVPWM technique. Figure 5-32 shows an example. $\boldsymbol{U_{out}}$ is in sector I ($\boldsymbol{U_0} \sim \boldsymbol{U_{60}}$), and the period of PWM is $\boldsymbol{T}$. $\boldsymbol{T_1}$ is the respective duration for $\boldsymbol{U_0}$ appearance; $\boldsymbol{T_2}$ is the respective duration for $\boldsymbol{U_{60}}$ appearance; $\boldsymbol{T_0}$ is the duration of zero vectors.

Figure 5-32. Voltage Vector in Sector I



Then the $\boldsymbol{U_{out}}$ can be expressed as

$$\boldsymbol{T = T_1 + T_2 + T_0}$$

$$\boldsymbol{U_{out} = U_{60} \times \frac{T_2}{T} + U_0 \times \frac{T_1}{T}}$$

Therefore

$$|\boldsymbol{U_{out}}|\, \boldsymbol{cos\,\theta} = |\boldsymbol{U_{60}}| \times \frac{\boldsymbol{T_2}}{\boldsymbol{T}} \times \boldsymbol{cos}\frac{\boldsymbol{\pi}}{\boldsymbol{3}} + |\boldsymbol{U_0}| \times \frac{\boldsymbol{T_1}}{\boldsymbol{T}}$$
$$|\boldsymbol{U_{out}}|\, \boldsymbol{\sin\theta} = |\boldsymbol{U_{60}}| \times \frac{\boldsymbol{T_2}}{\boldsymbol{T}} \times \boldsymbol{sin}\frac{\boldsymbol{\pi}}{\boldsymbol{3}}$$

Then

$$\boldsymbol{T_2} = \boldsymbol{mT}\sin\boldsymbol{\theta}$$
$$\boldsymbol{T_1} = \boldsymbol{mT}\sin\left(\frac{\boldsymbol{\pi}}{\boldsymbol{3}} - \boldsymbol{\theta}\right)$$
$$\boldsymbol{m} = \sqrt{3} \times \frac{|\boldsymbol{U_{out}}|}{\boldsymbol{U_{dc}}}$$
$$|\boldsymbol{U_{out}}| = \sqrt{\boldsymbol{u_\alpha}^2 + \boldsymbol{u_\beta}^2}$$

According to the ON/OFF state of the upper transistor in $\boldsymbol{U_{60}}$, $\boldsymbol{U_0}$, and zero vector, the $\boldsymbol{U_{out}}$ can be generated by changing the duty cycles of PWMs applied to the inverter.

5.4.1.4 Sensorless FOC Technique

The Park transformation requires the rotor position information, which is presented as the angle θ between the rotor flux linkage and α-axis. This is calculated by the Speed and Position Estimation block based on the motor feedback signals.

In a sensored design, the feedback signal comes from specific sensors, such as Hall sensors or optical encoders. These sensors not only increase the total cost of the motor system, but they also require maintenance, as they may fail during the motor's lifetime. The sensorless technique removes the physical sensors and replaces them with an algorithm in a microcontroller.

The idea of the sensorless technique is to estimate position based on the BEMF due to its relationship with angle θ. The BEMF can be measured directly. However, the measurement circuitry has to handle high voltages; therefore, this technique is not encouraged.

The two-phase voltages in the *(α, β)* frame can be expressed as

$$\begin{bmatrix}\boldsymbol{u_\alpha}\\ \boldsymbol{u_\beta}\end{bmatrix} = \begin{bmatrix}\boldsymbol{R_s} + \boldsymbol{\rho L} & \boldsymbol{0}\\ 0 & \boldsymbol{R_s} + \boldsymbol{\rho L}\end{bmatrix}\begin{bmatrix}\boldsymbol{i_\alpha}\\ \boldsymbol{i_\beta}\end{bmatrix} + \begin{bmatrix}e_\alpha\\ e_\beta\end{bmatrix}$$

Where

$\boldsymbol{R_s}$	Resistance of winding phase
$\boldsymbol{L}$	$L_d = L_q = L$ for SPMWM
$\boldsymbol{\rho}$	Differential operator $\boldsymbol{{}^{d}/_{dt}}$
$\begin{bmatrix}e_\alpha\\ e_\beta\end{bmatrix}$	BEMF excited by rotor magnet

In the digital domain, the equation can be changed to

$$\frac{\boldsymbol{i_\alpha(n+1)} - \boldsymbol{i_\alpha(n)}}{\boldsymbol{T_s}} = \left(-\frac{\boldsymbol{R_s}}{\boldsymbol{L_s}}\right)\boldsymbol{i_\alpha(n)} + \frac{\boldsymbol{1}}{\boldsymbol{L_s}}[\boldsymbol{u_\alpha(n)} - \boldsymbol{e_\alpha(n)}]$$

Where

$\boldsymbol{T_s}$	Period of PWM on inverter

Solving for $\boldsymbol{i_\alpha}$ as

$$\boldsymbol{i_\alpha(n+1)} = (1 - \boldsymbol{T_s}\frac{\boldsymbol{R_s}}{\boldsymbol{L_s}})\boldsymbol{i_\alpha(n)} + \frac{\boldsymbol{T_s}}{\boldsymbol{L_s}}[\boldsymbol{u_\alpha(n)} - \boldsymbol{e_\alpha(n)}]$$

Note that $\boldsymbol{R_s}$ and $\boldsymbol{L_s}$ are the motor characteristics, which can be measured by equipment. $\boldsymbol{T_s}$ is a constant system parameter, $\boldsymbol{i_\alpha(n)}$is a sampling result in the last control cycle, and $\boldsymbol{u_\alpha(n)}$ is the calculation result in the last control cycle. Therefore, if $e_\alpha(n)$ is given as $\boldsymbol{e^*_\alpha(n)}$, an estimated current value $\boldsymbol{i^*_\alpha(n+1)}$ can be obtained. The "*" here indicates it is a value estimated by algorithm and not directly measured.

Comparing $\boldsymbol{i^*_\alpha(n+1)}$ with the actual current value $\boldsymbol{i_\alpha(n+1)}$ sampled by ADC, an error between these two can be fed to adjust the given value $\boldsymbol{e^*_\alpha(n)}$ for better estimation. This process is repeated until the error between $\boldsymbol{i^*_\alpha(n+1)}$ and $\boldsymbol{i_\alpha(n+1)}$ is as small as required. Then the given value $\boldsymbol{e^*_\alpha(n)}$ represents the actual BEMF $\boldsymbol{e_\alpha(n)}$.

The $\boldsymbol{e_\beta(n)}$ can be obtained in the same way, and the angle θ can be calculated as

$$\boldsymbol{\theta(n) = arc\,tan\frac{-e_\alpha(n)}{e_\beta(n)}}$$

Angular speed $\boldsymbol{\omega_r}$ is calculated by accumulating θ over $\boldsymbol{m}$ samples and is multiplied by speed constant $\boldsymbol{K}$.

$$\boldsymbol{\omega_r = \sum_{n=1}^{m}[\theta(n) - \theta(n-1)] * K}$$

Thus, the position and speed information are retrieved from the estimated BEMF.

5.4.2 Sensorless Foc Motor Control Example Project Overview

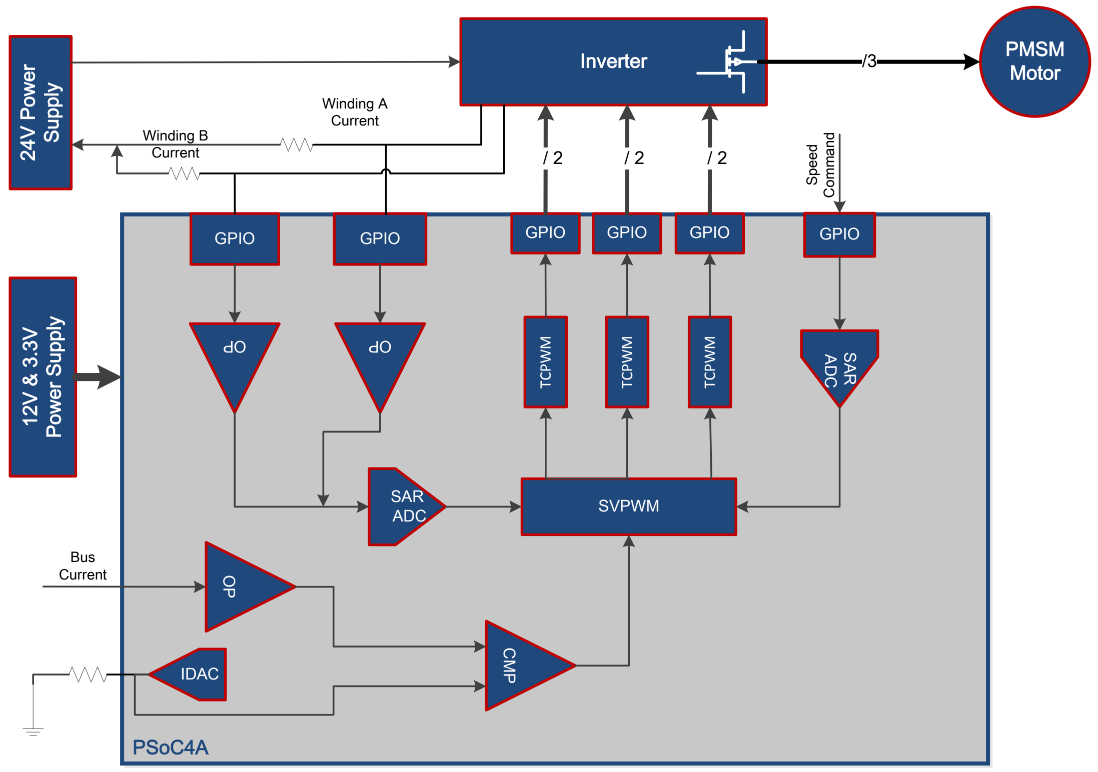
Figure 5-33 illustrates the block diagram of the Sensorless Foc Motor Control example project based on PSoC 4A. The input control signals to the PSoC 4 device are the following.

- **Speed command:** An analog input pin that measures the voltage across a potentiometer to set the desired speed of rotation (one analog input pin).
- **Motor current measurement:** Two analog input pins to detect the motor winding current by sensing resistors. Then the signal is routed to internal opamps to be amplified and filtered.
- **Start/Stop control:** Digital input connected to a switch to start and lock rotation of the motor (one digital input pin).

Outputs from the PSoC 4 device are power device driver signals.

- PWM signals to the high side of the MOSFET driver (three digital output pins)
- PWM signals to the low side of the MOSFET driver (three digital output pins)
- Overcurrent limit voltage set by the internal IDAC and an outside resistor

Figure 5-33. Sensorless Foc Motor Control Example Project Block Diagram



5.4.3 Control Schematic Overview

The Sensorless Foc Motor Control example project firmware was developed in PSoC Creator 3.2. Its schematic was separated into two pages according to the function. Figure 5-34 shows the PWM design in the “Sensorless FOC” page. The three TCPWM Components generate three couples of the center-aligned and complemented SVPWM signal according to the FOC algorithm.

Figure 5-34. PWM Schematic

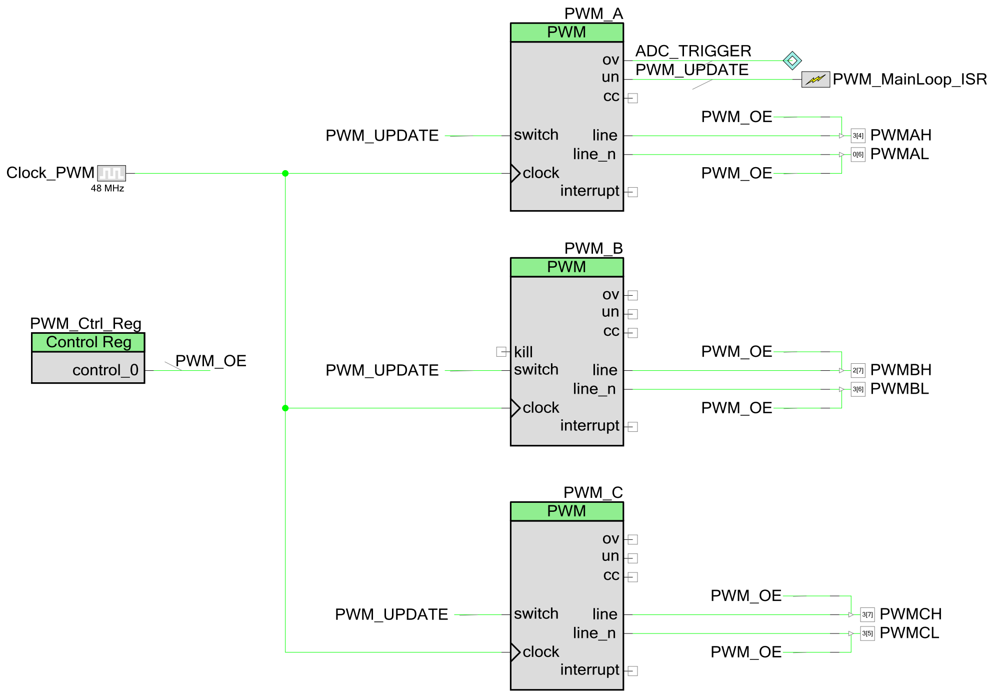


Figure 5-35 shows the ADC schematic in the “Sensorless FOC” page. SAR ADC is triggered in every PWM cycle by the “ov” signal, and it samples for the following signals: two-phase winding currents, bus voltage, and potentiometer divider voltage. “Opamp_A” and “Opamp_B” are internal opamps to amplify and filter the motor winding currents combining the external resistors and capacitors network.

Figure 5-35. ADC Schematic

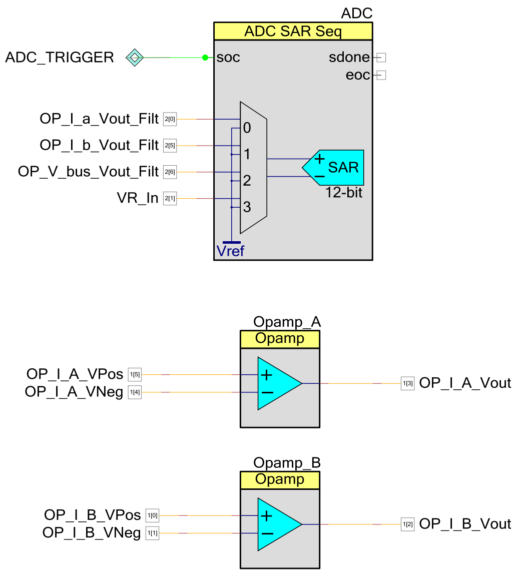
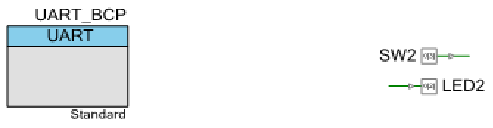


Figure 5-36 shows the "System" page. "SW2" is the external switch that starts and stops the motor. "UART_BCP" is an internal SCB Component configured as "UART," which is responsible for sending data to the BCP while the motor is running.

Figure 5-36. System Schematic

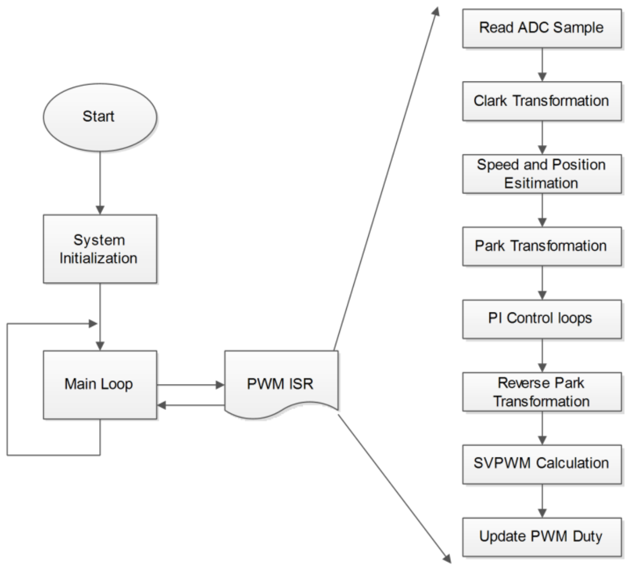


5.4.4 Firmware Introduction

Figure 5-37 gives the firmware execution flow. Most of the FOC calculation is done in an ISR triggered by the PWM UN event. As there is only one UN for each PWM period, the ISR is executed per 100 µs (10-kHz PWM on the inverter), which is also the control cycle period for the PSoC 4 FOC implementation.

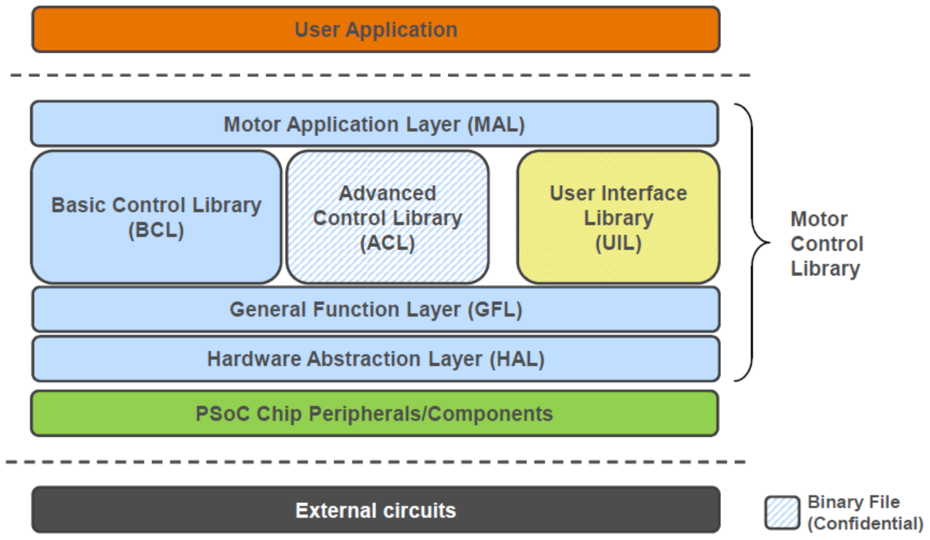
The FOC algorithm requires that the PWM duty cycle be updated for each control period, while other functions, such as communication with other systems, overvoltage detection, and so on do not require real-time processing. Therefore, these functions are executed in the main routine.

Figure 5-37. Firmware Execution Flow Chart



To create a multilayer, professional, and extensible library architecture for the Cypress motor control solution, the Sensorless Foc Motor Control example project utilizes a multilayer structure. Figure 5-38 shows the layer diagram of the example project.

Figure 5-38. Sensorless Foc Motor Control Example Project Layer Diagram


User Application
Motor Application Layer (MAL)
Basic Control Library (BCL)
Advanced Control Library (ACL)
User Interface Library (UIL)
General Function Layer (GFL)
Hardware Abstraction Layer (HAL)
PSoC Chip Peripherals/Components
Motor Control Library
External circuits
Binary File (Confidential)

A description of each layer follows.

Hardware Abstraction Layer (HAL)

- This layer separates the chip-related code from the motor control algorithm, which ensures the algorithm can cross multiple PSoC devices.
- Use the macro definition for operation instead of a different component name in a different user design.
- User binds the real component name to the macro definition.

General Function Layer (GFL)

- Q-Math (16-bit, 24-bit, 32-bit fixed-point math operation)
- Trigonometric and math function (sin, cos, arctan, sqrt, abs, and so on)
- Standard PID calculation
- General filters (FIR, 1-order IIR, 2-order IIR, median filter)
- General definition such as BOOL, FALSE, TRUE, and so on

Basic Control Layer (BCL)

- Fundamental control modules (Clarke/Park transformation and so on)
- Decoder (Hall effect, incremental encoder, rotary encoder, and so on)
- Other fundamental features like startup

Advanced Control Layer (ACL)

- Sensorless algorithm for FOC, high-frequency injection
- Other protected IPs

User Interface Layer (UIL)

- Firmware to support tuning GUI
- Firmware to operate IDAC for debugging
- This layer is useful for debugging, but it may be removed in the final product.

Motor Application Layer (MAL)

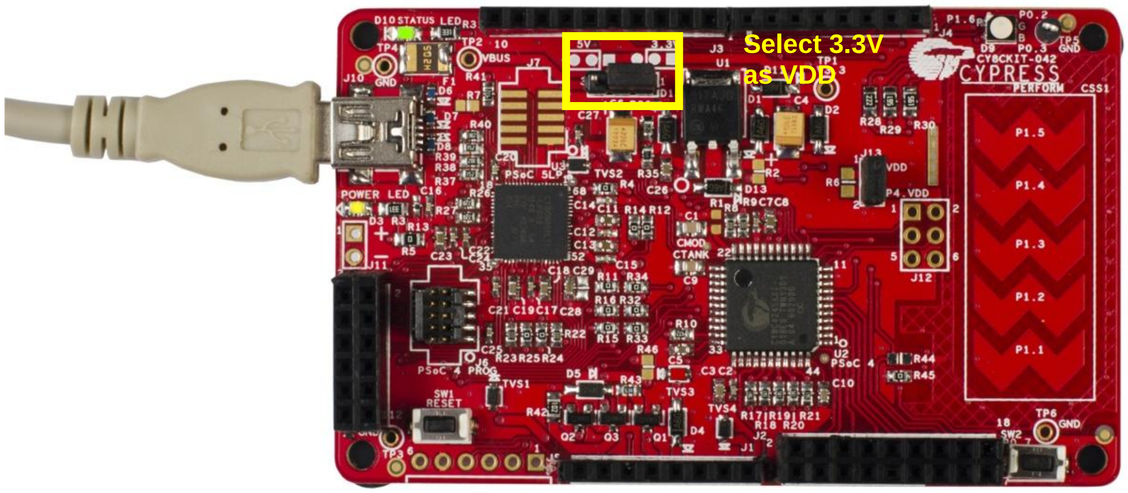
- General motor application functions (start motor, stop motor, speed ramp-up, speed slowdown, and so on).
- Basic fault detection (overcurrent, overvoltage, undervoltage, motor fault, and so on)

5.4.5 Running the Sensorless Foc Motor Control Example Project

5.4.5.1 Step 1 – Configure CY8CKIT-042

Select 3.3 V as the VDD power at jumper J9 on the Pioneer Kit. In this application, the USB cable is used only for burning firmware, so it can be removed from CY8CKIT-042 after programming. Figure 5-39 shows the configuration.

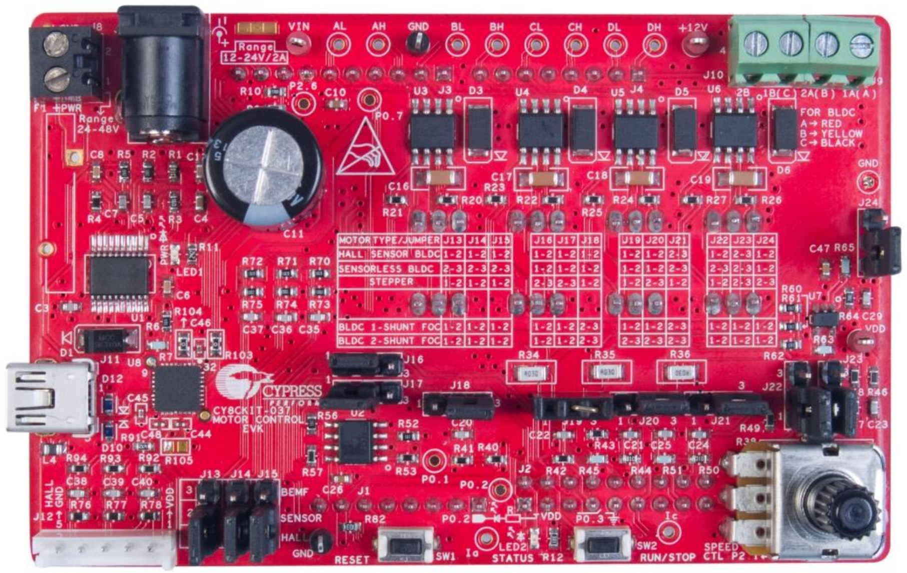
Figure 5-39. CY8CKIT-042 Configuration for Sensorless Foc Motor Control Example Project



5.4.5.2 Step 2 – Configure CY8CKIT-037

Configure the board via jumpers J13–J24 for the Sensorless Foc Motor Control example project (“BLDC 2-SHUNT FOC” row in Figure 5-1). Figure 5-40 shows the EVK board configured for the Sensorless Foc Motor Control example project.

Figure 5-40. CY8CKIT-037 Configuration for Sensorless Foc Motor Control Example Project



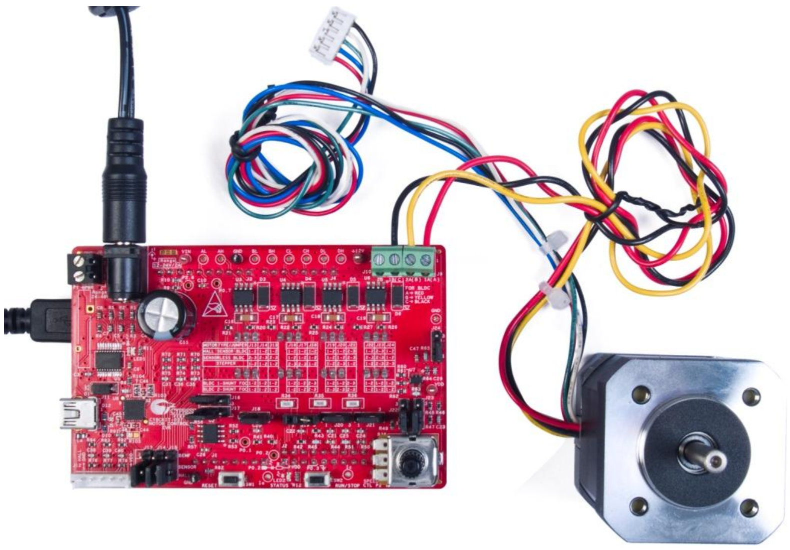
5.4.5.3 Step 3 – Plug CY8CKIT-037 into CY8CKIT-042

Plug the EVK board into the Pioneer Kit via connectors J1–J4, as shown in Figure 5-11.

5.4.5.4 Step 4 – Connect the Power Supply and Motor

Connect the BLDC motor windings to J9 and J10 on the EVK board, and then connect the 24-V power adapter to J7, as shown in Figure 5-41. When LED1 is red, it indicates power is on. If it is not glowing, the fuse F2 may be broken. Please change the F2 with the provided backup fuse..

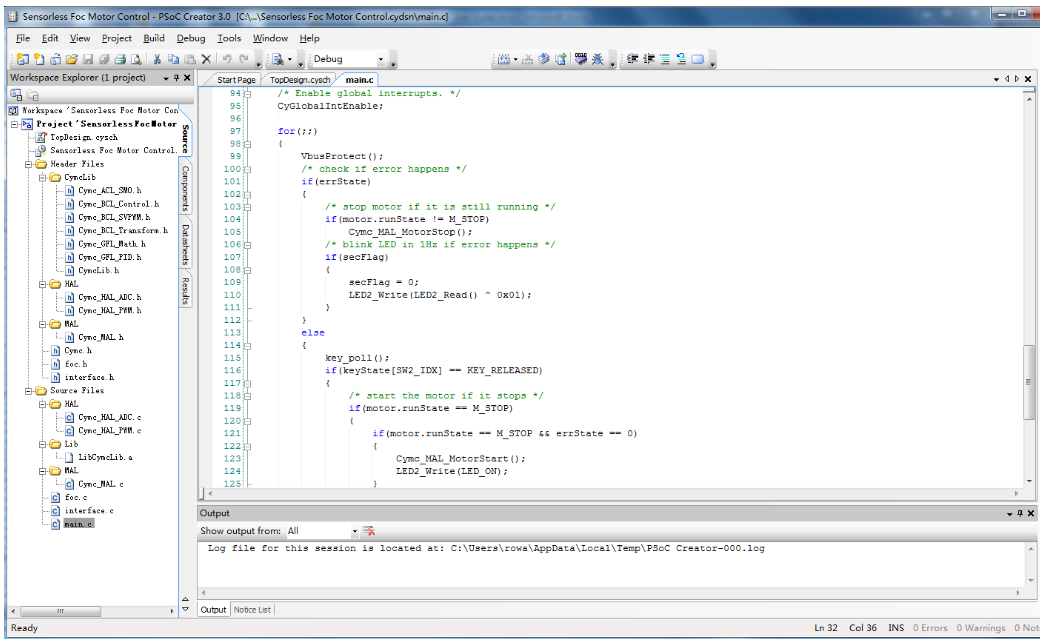
Figure 5-41. Connect Motor and Power Supply



5.4.5.5 Step 5 – Build the Project and Program PSoC 4

Open the Sensorless Foc Motor Control example project in PSoC Creator 3.2 or later version, as shown in Figure 5-42. Choose **Build** > **Build Sensorless Foc Motor Control** to start building the project, and then choose **Debug** > **Program** to program the chip.

Figure 5-42. Open the Sensorless Foc Motor Control Example Project



```
    /* Enable global interrupts. */
    CyGlobalIntEnable;

    for(;;)
    {
        VbusProtect();
        /* check if error happens */
        if(errState)
        {
            /* stop motor if it is still running */
            if(motor.runState != M_STOP)
                Cymc_MAL_MotorStop();
            /* blink LED in 1Hz if error happens */
            if(secFlag)
            {
                secFlag = 0;
                LED2_Write(LED2_Read() ^ 0x01);
            }
        }
        else
        {
            key_poll();
            if(keyState[SW2_IDX] == KEY_RELEASED)
            {
                /* start the motor if it stops */
                if(motor.runState == M_STOP)
                {
                    if(motor.runState == M_STOP && errState == 0)
                    {
                        Cymc_MAL_MotorStart();
                        LED2_Write(LED_ON);
                    }
```

```
Log file for this session is located at: C:\Users\rowa\AppData\Local\Temp\PSoC Creator-000.log
```

CAUTION **During the debugging process, if you modify the code in the firmware, make sure that the changes do not turn on both the high-side and low-side MOSFETs.**

5.4.5.6 Step 6 – Press the SW2 Button to Start Motor Rotation

Press the SW2 button to start motor rotation (refer to Figure 5-14). If the motor does not rotate and you observe LED2 blinking, it indicates that an error has occurred. If so, first check that step 1 through step 5 have executed correctly. Then press the Reset button and press the SW2 button again. If LED2 still blinks, there must be a problem in the hardware or software. You have to debug it or contact Cypress for technical support.

CAUTION **Do not remove jumpers while the kit is powered.**

5.4.6 Adapting the Example Project to another Motor

This example project can be used to control several different types of three-phase PMSM motors, and motor performance is closely related to motor parameters. So the motor-related parameters in the project firmware need to be modified according to your specific motor characteristics. The motor-related parameters are defined by a `struct` in *Cymc_MAL.h*. Table 5-3 shows the motor-related parameters.

Table 5-3. Motor-Related Parameters for Sensorless Foc Motor Control Example Project

Variable's Name in Project	Structure Member	Comments
Name: `motor` **Type:** `Struct MotorController` **Location:** `Cymc_MAL.h` **Comments**: motor parameters struct	`float Rs`	`motor Rs`
	`float Ls`	`motor Ls`
	`uint8 Poles`	`motor Poles`
	`float baseFrequency`	`base frequency`
	`float Ts`	`PWM period`
	`uint16 speedRPM`	`Speed in RPM format. Read only.`
	`uint16 ratedSpeedRPM`	`rated speed in RPM format`
	`uint8 runState`	`motor run state`
	`q15_t speedRef`	`speed reference`
	`q15_t vdRef`	`vd reference`
	`q15_t vqRef`	`vq reference`

Rs, Ls, Poles, baseFrequency, Ts, and ratedSpeedRPM are the motor's parameters. The related macro definitions are in *Cymc_MAL.h* as follows.

```
#define M_RS            0.8
#define M_LS            0.0012
#define M_POLES         8
#define M_TS            0.0001
#define M_BASEFREQUENCY 266.67
```

You can modify these parameters in the `void Cymc_MAL_MotorInit() function in` *Cymc_MAL.c.*

Table 5-4 shows the control-related parameters.

Table 5-4. Control-Related Parameters

Variable's Name in Project	Structure Member	Comments
Name: `rampUp` **Type:** `Struct RampUp` **Location:** `Cymc_MAL.h` **Comments**: motor parameters struct	`q15_t RefSpeed`	`reference speed`
	`q15_t delayPrescaler`	`delay prescaler`
	`q15_t stepSpeed`	`step speed`
	`q15_t output`	`output speed`
Name: `pidSpeed`	`q15_t ref`	`reference set point`

Variable's Name in Project	Structure Member	Comments
Type: `Struct PIController` **Location**:`Cymc_MAL.h` **Comments**: PID controller for speed	`q15_t fbk`	`feedback`
	`q15_t out`	`controller output`
	`q15_t kr`	`reference set-point weighting`
	`q15_t kp`	`proportional loop gain`
	`q15_t ki`	`integral gain`
	`q15_t outMax`	`upper output limit`
	`q15_t outMin`	`lower output limit`
Name: `pidId` **Type**: `Struct PIController` **Location**:`Cymc_MAL.h` **Comments**: PID controller for d-axis current	`q15_t ref`	`reference set-point`
	`q15_t fbk`	`feedback`
	`q15_t out`	`controller output`
	`q15_t kr`	`reference set-point weighting`
	`q15_t kp`	`proportional loop gain`
	`q15_t ki`	`integral gain`
	`q15_t outMax`	`upper output limit`
	`q15_t outMin`	`lower output limit`
Name: `pidIq` **Type**: `Struct PIController` **Location**:`Cymc_MAL.h` **Comments**: PID controller for q-axis	`q15_t ref`	`reference set-point`
	`q15_t fbk`	`feedback`
	`q15_t out`	`controller output`
	`q15_t kr`	`reference set-point weighting`
	`q15_t kp`	`proportional loop gain`
	`q15_t ki`	`integral gain`
	`q15_t outMax`	`upper output limit`
	`q15_t outMin`	`lower output limit`

rampUp is used for ramping up the speed reference to the given speed.

pidSpeed is the PID controller for speed. The related macro definitions are located *in Cymc_MAL.h* as follows.

```
#define PID_SPEED_KP    _FQ(0.8333)
#define PID_SPEED_KR    _FQ(1.0)
#define PID_SPEED_KI    _FQ(0.0033264)
#define PID_SPEED_OMAX  _FQ(0.55)
#define PID_SPEED_OMIN  _FQ(-0.55)
```

pidId is the PID controller for d-axis current. The related macro definitions are located in *Cymc_MAL.h* as follows.

```
#define PID_ID_KP       _FQ(1.0)
#define PID_ID_KR       _FQ(1.0)
#define PID_ID_KI       _FQ(0.2)
#define PID_ID_OMAX     _FQ(0.7)
#define PID_ID_OMIN     _FQ(-0.7)
```

pidIq is the PID controller for q-axis current. The related macro definitions are located in *Cymc_MAL.h* as follows.

```
#define PID_IQ_KP        _FQ(1.0)

#define PID_IQ_KR        _FQ(1.0)

#define PID_IQ_KI        _FQ(0.2)

#define PID_IQ_OMAX      _FQ(0.95)

#define PID_IQ_OMIN      _FQ(-0.95)
```

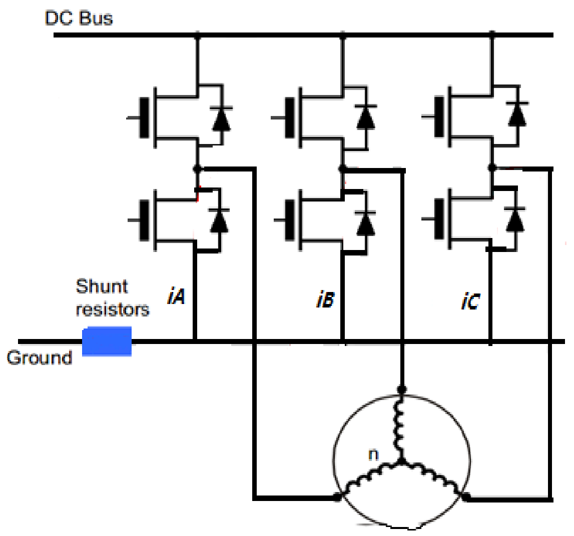
You can modify these parameters in the `Cymc_MAL_MotorInit()` and `Cymc_MAL_MotorStart()` functions in *Cymc_MAL.c*.

5.5 SingleShunt Foc Motor Control Example Project

5.5.1 SingleShunt Foc Background

The two-shunt sensorless Foc example project in the Sensorless Foc Motor Control Example Project gets the motor winding current and estimates the rotor position from two sensing resistors directly in series with two coil current paths. The single-shunt FOC technique uses only one sensing resistor to get the bus current and reconstruct the two-phase motor winding current. It then estimates the rotor position according to the reconstructed winding current. Compared with the two-shunt FOC technique, it saves a sensing resistor and a differential amplifier and so reduces the cost and PCB space. Figure 5-43 shows the current measurement by the single shunt.

Figure 5-43. Single-Shunt FOC Current Measurement

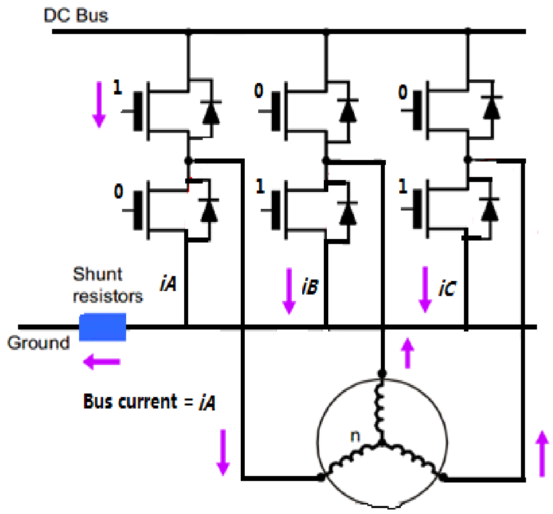


The disadvantages of single-shunt FOC are obvious. You need to reconstruct the three-phase motor winding current from the bus current, so the sinusoidal modulation pattern needs to be modified to allow the bus current to be measured. This makes the PWM signal generation algorithm more complicated than in the two-shunt sensorless Foc example project.

As Figure 5-44 shows, there are eight possible combinations of ON and OFF states of PWM. Each state has a different bus current. When the PWM state is zero vector (000 or 111), the bus current is zero. For the other six states, the following discussion uses 100 vector to explain the winding current reconstruction.

When PWMA is ON, PWMB and PWMC are OFF (100 state), and current flows from phase A into phases B and C. At this time, the bus current is i_A, as Figure 5-44 shows.

Figure 5-44. Bus Current on 100 PWM Vector



In the same way, you can get the bus current of other PWM states. Table 5-5 shows the relationship between the bus current and the PWM state.

Table 5-5. Relationship Between Bus Current and PWM State

1H	1L	2H	2L	3H	3L	iBus
1	0	0	1	0	1	iA
0	1	1	0	0	1	IB
0	1	0	1	1	0	IC
0	1	1	0	1	0	-iA
1	0	0	1	1	0	-IB
1	0	1	0	0	1	-IC

So, if you sample bus current twice in one PWM period in a different PWM state, you can get two-phase currents. Then you can reconstruct the third one.

5.5.2 SingleShunt Foc Motor Control Example Project Overview

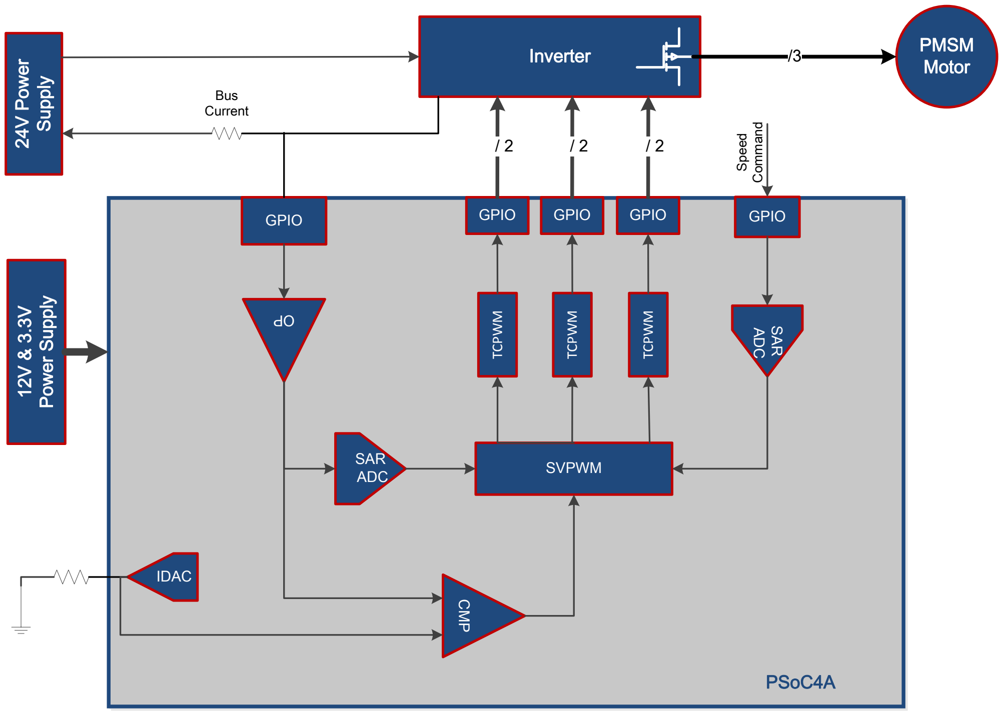
Figure 5-45 illustrates the block diagram of the SingleShunt Foc Motor Control example project based on PSoC 4A. The input control signals to the PSoC 4 device are the following.

- **Speed command:** Analog input pin that measures the voltage across a potentiometer to set the desired speed of rotation (one analog input pin)
- **Motor current detection:** Analog input pin to detect and cut off power to the device driver to protect the motor when an overcurrent condition is detected (see Control Schematic Overview)
- **Vbus:** Analog input pin that routes bus voltage to the SAR ADC to monitor the bus voltage
- **Start/stop control:** Digital input connected to a switch to start and stop the motor rotation (one digital input pin)

Outputs from the PSoC 4 device are power device driver signals.

- PWM signals to the high side of the MOSFET driver (three digital output pins)
- PWM signals to the low side of the MOSFET driver (three digital output pins)
- Overcurrent limit voltage set by the internal IDAC and an outside resistor

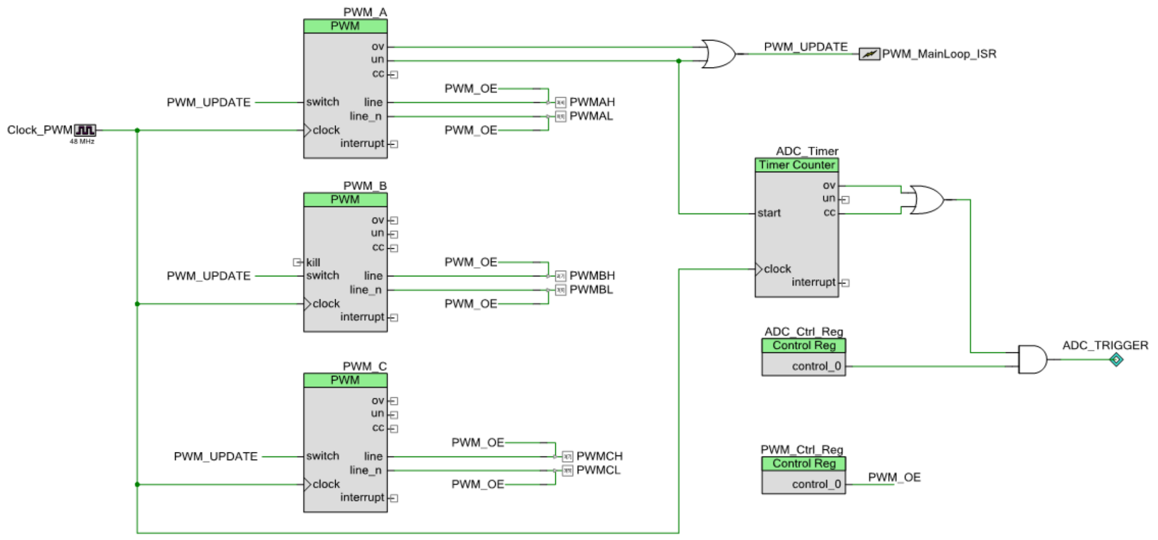
Figure 5-45. SingleShunt Sensorless Foc Motor Control Example Project Block Diagram



5.5.3 Control Schematic Overview

The SingleShunt Foc Motor Control example project firmware was developed in PSoC Creator 3.2. Its schematic was separated into two pages according to function. Figure 5-46 shows the “PWM Drive” schematic in the “FOC” page. The three TCPWM Components generate three couples of the center-aligned and complemented SVPWM signal according to the FOC algorithm.

Figure 5-46. PWM Drive Schematic



By comparing Figure 5-47 with Figure 5-34, you can see that the PWM duty cycle update and ADC trigger mechanism are different than in the two-shunt sensorless FOC because the current sensing and reconstructing method has been changed in the single-shunt sensorless FOC.

Figure 5-47 shows the ADC trigger timing. To avoid impacting the current pulse when the PWM state changes, ADC sampling is triggered in the middle of two PWM states. So the schematic shows the use of a PWM underflow signal (“un”) as a trigger source and then the use of a timer (“ADC_Timer”) to create a delay to ensure that ADC is triggered in the middle of ***T1*** and ***T2***. The compare value of “ADC_Timer” is updated in real time while the motor is running. In the PWM OV interrupt ISR, the ADC trigger delay should be set as ***T1/2***. Then after ADC is triggered, the ADC trigger delay should be set as ***T2/2*** in the ADC eoc interrupt ISR.

So, to change the compare value twice in one PWM period, you need two TC events. Then the PWM switch signal is triggered by “PWM_UPDATE” on both the OV and UN events. The PWM compare value will be updated twice in one period. The PWM ISR should also be triggered twice on the OV and UN events).

Figure 5-47. ADC Trigger Timing

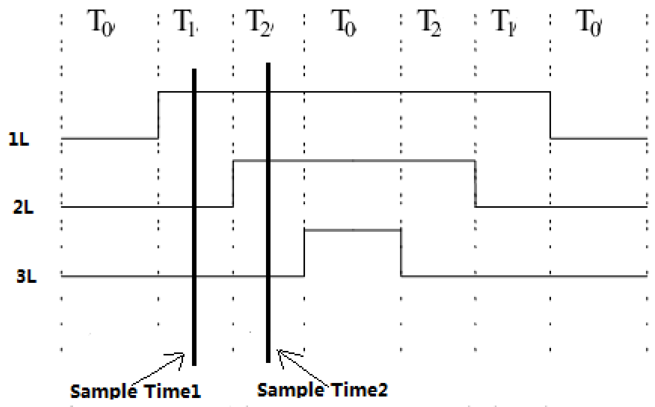


Figure 5-48 shows the “Phase Current Sampling” schematic in the “FOC” page. SAR ADC is triggered twice in every PWM cycle by the “ov” and “cc” signals, and it samples three signals: bus current, bus voltage, and potentiometer divider voltage. “Opamp_A” is internal opamp to amplify and filter the bus current combining the external resistors and capacitors network.

Figure 5-48. Phase Current Sampling Schematic

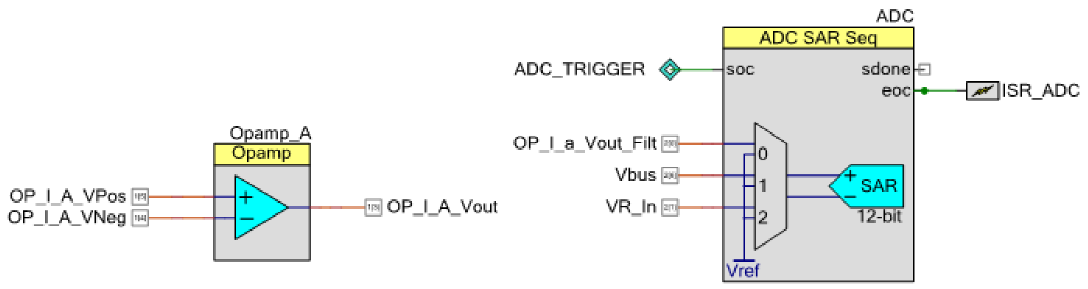
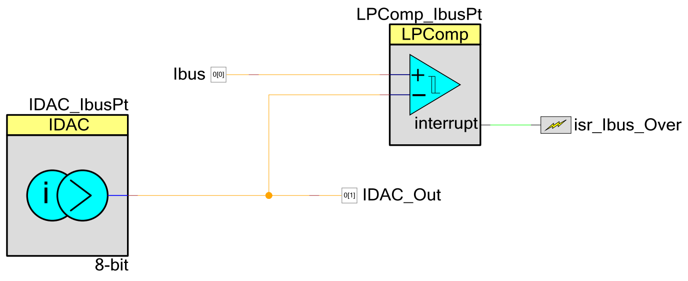


Figure 5-49 shows the “Overcurrent Protection” schematic. The bus current is measured by an external opamp and routed to the positive terminal of the internal low-power comparator ”LPComp_IbusPt.” The overcurrent threshold is set by an internal IDAC, “IDAC_IbusPt,” and routed to the negative terminal of ”LPComp_IbusPt.” When an overcurrent condition happens, ”LPComp_IbusPt” will generate an interrupt to disable the PWM output.

Figure 5-49. Overcurrent Protection Schematic



5.5.4 Firmware Introduction

Except for the motor current sensing and reconstruction mechanism introduced in Control Schematic Overview, the control firmware is similar to the Sensorless Foc Motor Control example project. Refer to Firmware Introduction to learn about the firmware of the SingleShunt Foc Motor Control example project.

5.5.5 Running the SingleShunt Foc Motor Control Example Project

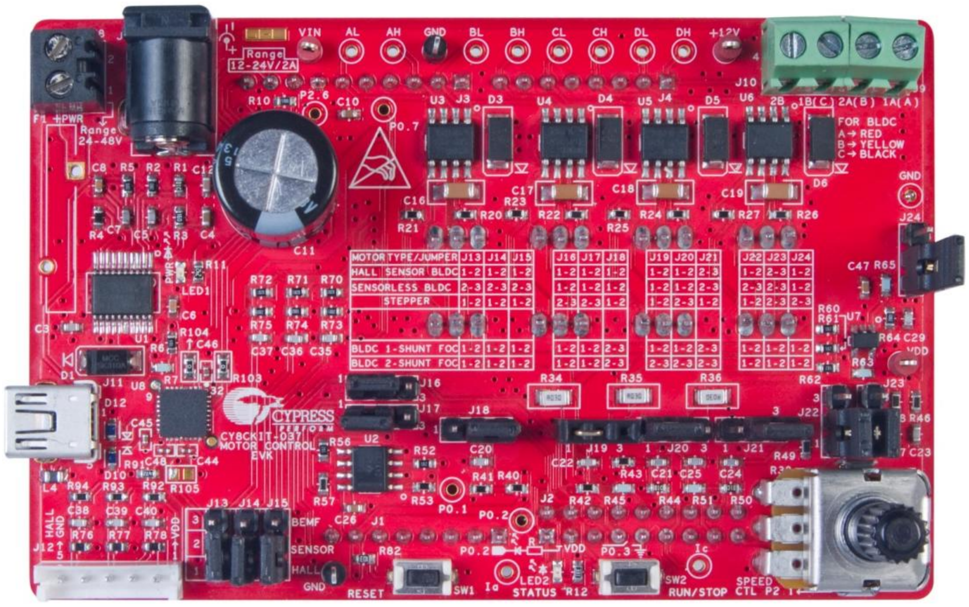
5.5.5.1 Step 1 – Configure CY8CKIT-042

Select 3.3 V as the VDD power at jumper J9 on the Pioneer Kit. In this application, the USB cable is used only for burning firmware, so it can be removed from CY8CKIT-042 after programming. Figure 5-39 shows the CY8CKIT-042 configuration.

5.5.5.2 Step 2 – Configure CY8CKIT-037

Configure the board via jumpers J13–J24 for the SingleShunt Foc Motor Control example project ("BLDC 1-SHUNT FOC" row in Figure 5-1). Figure 5-50 shows the EVK board configured for the SingleShunt Foc Motor Control example project.

Figure 5-50. CY8CKIT-037 Configuration for SingleShunt Foc Motor Control Example Project



5.5.5.3 Step 3 – Plug CY8CKIT-037 into CY8CKIT-042

Plug the EVK board into the Pioneer Kit via connectors J1–J4, as shown in Figure 5-11.

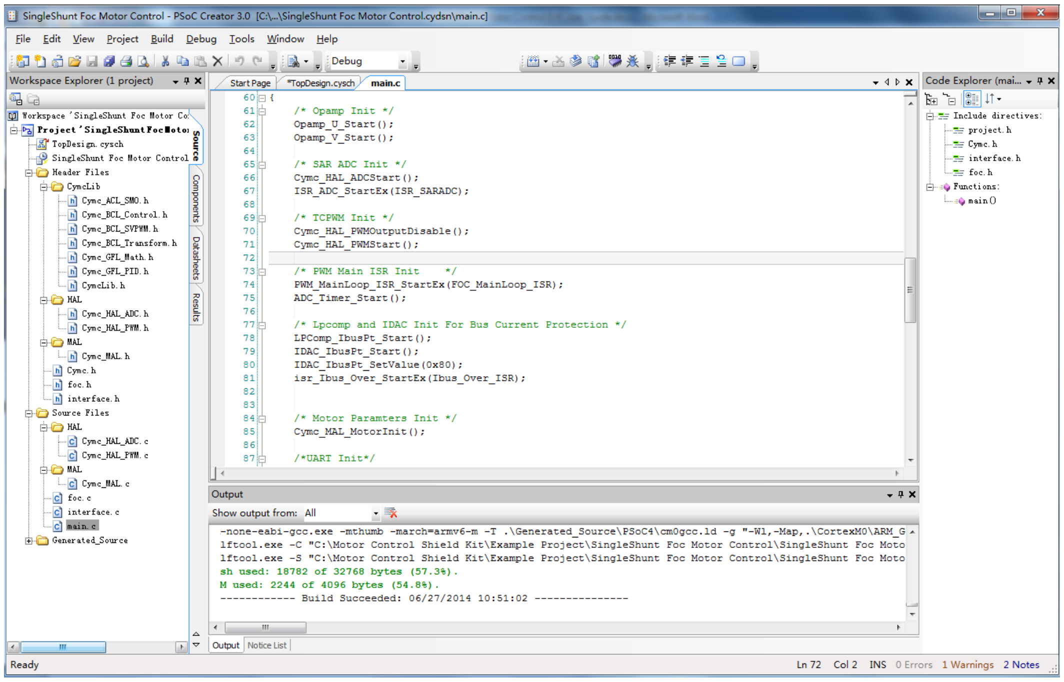
5.5.5.4 Step 4 – Connect the Power Supply and Motor

Connect the BLDC motor windings to J9 and J10 on the EVK board, and then connect the 24-V power adapter to J7, as shown in Figure 5-41. When LED1 is red, it indicates power is on. If it is not glowing, the fuse F2 may be broken. Please change the F2 with the provided backup fuse..

5.5.5.5 Step 5 – Build the Project and Program PSoC 4

Open the SingleShunt Foc Motor Control example project in PSoC Creator 3.2 or later version, as shown in Figure 5-51. Choose **Build** > **Build SingleShunt Foc Motor Control** to start building the project, and then choose **Debug** > **Program** to program the chip.

Figure 5-51. Open the SingleShunt Foc Motor Control Example Project



```
{
    /* Opamp Init */
    Opamp_U_Start();
    Opamp_V_Start();

    /* SAR ADC Init */
    Cymc_HAL_ADCStart();
    ISR_ADC_StartEx(ISR_SARADC);

    /* TCPWM Init */
    Cymc_HAL_PWMOutputDisable();
    Cymc_HAL_PWMStart();

    /* PWM Main ISR Init    */
    PWM_MainLoop_ISR_StartEx(FOC_MainLoop_ISR);
    ADC_Timer_Start();

    /* Lpcomp and IDAC Init For Bus Current Protection */
    LPComp_IbusPt_Start();
    IDAC_IbusPt_Start();
    IDAC_IbusPt_SetValue(0x80);
    isr_Ibus_Over_StartEx(Ibus_Over_ISR);


    /* Motor Paramters Init */
    Cymc_MAL_MotorInit();

    /*UART Init*/
```

```
-none-eabi-gcc.exe -mthumb -march=armv6-m -T .\Generated_Source\PSoC4\cm0gcc.ld -g "-Wl,-Map,.\CortexM0\ARM_G
lftool.exe -C "C:\Motor Control Shield Kit\Example Project\SingleShunt Foc Motor Control\SingleShunt Foc Moto
lftool.exe -S "C:\Motor Control Shield Kit\Example Project\SingleShunt Foc Motor Control\SingleShunt Foc Moto
sh used: 18782 of 32768 bytes (57.3%).
M used: 2244 of 4096 bytes (54.8%).
------------ Build Succeeded: 06/27/2014 10:51:02 ---------------
```

CAUTION	**During the debugging process, if you modify the code in the firmware, make sure that the changes do not turn on both the high-side and low-side MOSFETs.**

5.5.5.6 Step 6 – Press the SW2 Button to Start Motor Rotation

Press the SW2 button to start motor rotation (refer to Figure 5-14). If the motor does not rotate and you observe LED2 blinking, it indicates that an error has occurred. If so, first check that step 1 through step 5 have executed correctly. Then press the Reset button and press the SW2 button again. If LED2 still blinks, there must be a problem in the hardware or software. You have to debug it or contact Cypress for technical support.

CAUTION	**Do not remove jumpers while the kit is powered.**

5.5.6 Adapting the Example Project to another Motor

Refer to Adapting the Example Project to another Motor.

5.6 Stepper Motor Control Example Project

CY8CKIT-037 does not provide a stepper motor in the kit package. However, the kit does support a stepper motor controlled with microstep from hardware and firmware. A Stepper Motor Control example project is included in the kit installation directory. To demonstrate this project, you need to connect your stepper motor to the CY8CKIT-037 kit. The only requirement for the motor is that the power range should meet the kit spec.

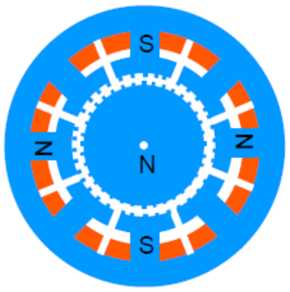
5.6.1 Stepper Motor Background

The stepper motor is an electromechanical device that converts electrical pulses into discrete mechanical movement. It can achieve accurate positioning and potential rotation speed control without a feedback sensor. So it is ideally suited for many measurement and control applications where positional accuracy is important and cost is very low.

The most frequently used stepper motor is the hybrid stepper motor. It has smaller steps, greater torque, and greater maximum speeds than a variable reluctance and permanent magnet stepper motor. Figure 5-52 shows a sectional view of the hybrid stepper motor. Both the rotor and stator have teeth, providing a smaller magnetic circuit resistance in some rotor positions, which improves static and dynamic torque.

Stepper motors have several operational modes, such as full-step mode, half-step mode, and microstep mode. Each mode controls stepper motor phases in different ways. The full-step and half-step modes have obvious torque rippling and resonance. Microstep mode changes the current by small steps that split each step into microsteps. When two phases are turned on and the current on each phase is not equal, the current phase ratio determines the rotor position. Changing the ratio produces a number of microsteps inside each full step.

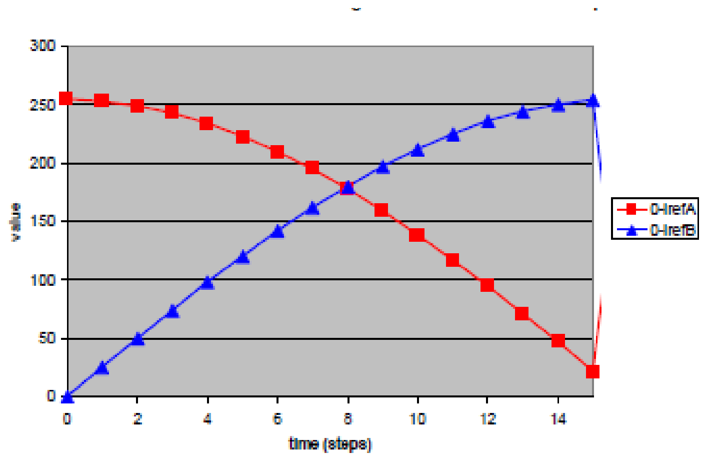
Figure 5-52. Hybrid Stepper Motor



Fractional steps are created by scaling torque contributions between windings. Because torque is proportional to the magnetic flux that is proportional to the current in the winding, the rotor's position can be controlled by controlling the current flowing in each winding. To create smooth microsteps between full steps, the current is varied sinusoidally with a 90-degree phase shift between the two windings, as shown in Figure 5-53.

The current is scaled by controlling the RMS current using a current-mode buck converter, commonly called a "chopper drive" when used with stepper motors. The phase current is converted into a voltage using a sensing resistor in each phase ground path. This voltage is routed to a comparator, which disables the output whenever the phase current rises above a reference. The comparator reference is provided by an IDAC. By changing the IDAC-supplied current limit for each microstep, the total motor torque remains approximately constant for each step of the sinusoidal current waveform.

Figure 5-53. VDAC Current Limit for Microstep Mode



5.6.2 Stepper Motor Control Example Project Overview

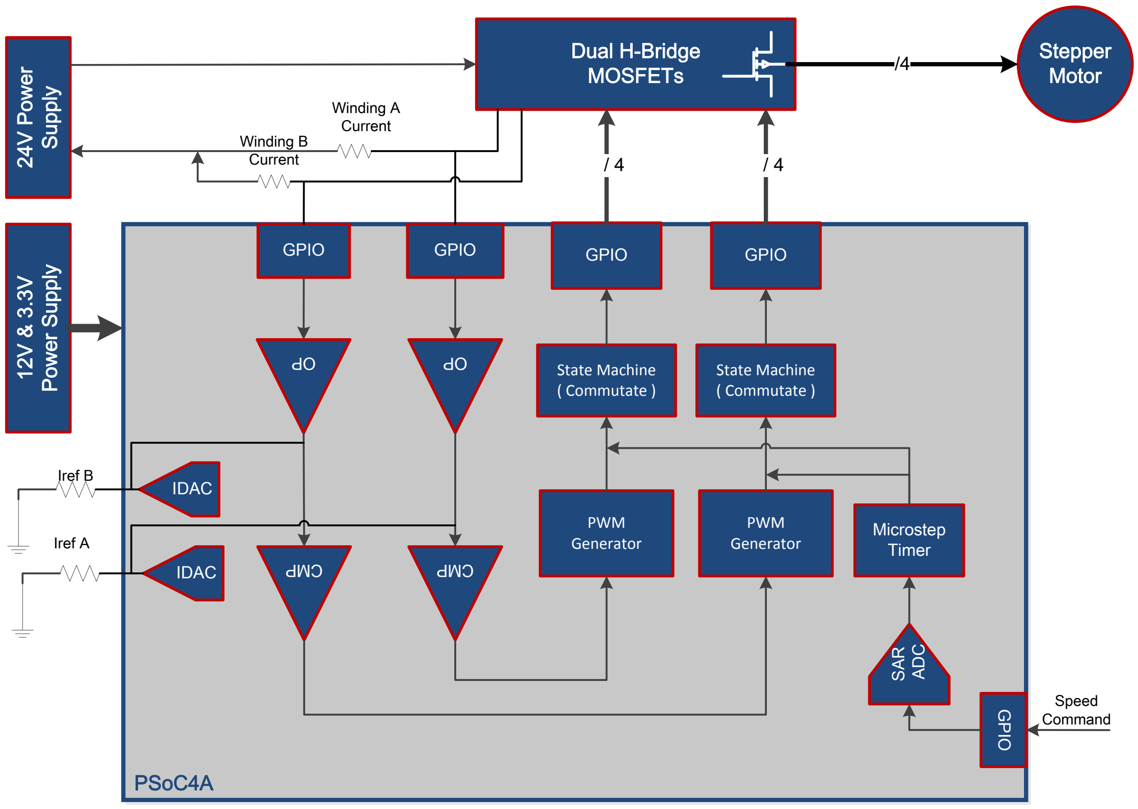
Figure 5-54 illustrates the block diagram of the Stepper Motor Control example project based on PSoC 4A. The input control signals to the PSoC 4 device are as follows.

- **Speed command:** An analog input pin that measures the voltage across a potentiometer to set the desired speed of rotation (one analog input pin).
- **Motor current detection:** Two analog input pins to detect the motor winding current by sensing resistors. Then the signal is routed to internal opamps to be amplified.
- **PWM kill:** Two digital input pins that are the output of external comparators. A high level of this signal will stop internal PWM output and force the winding current to decrease to the current reference set by the IDAC.
- **Start/lock control:** A digital input connected to a switch to start and lock rotation of the motor (one digital input pin).

Outputs from the PSoC 4 device are power device driver signals.

- PWM signals to the high side of the MOSFET driver (four digital output pins).
- PWM signals to the low side of the MOSFET driver (four digital output pins).
- A winding current signal amplified by internal opamps. The signal is routed to positive external comparators and is compared with the current reference set by the IDAC to construct the winding current chopper circuit.
- The sinusoidal current reference voltage set by the internal IDAC and outside resistors.

Figure 5-54. Stepper Motor Control Example Project Block Diagram



5.6.3 Control Schematic Overview

The Stepper Motor Control example project firmware was developed in PSoC Creator 3.2. Its schematic was separated into four pages according to function. Figure 5-55 shows the "PWM and Commutation" page. When the motor phase current is more than the sinusoidal reference current level, the chopper circuit will output a high level. This high level closes the PWM output and synchronizes the real motor phase current with the sinusoidal reference current level. "Control_Reg_1" indicates the current stage of the stepper motor. It is dynamically modified by firmware to commutate two phase windings independently through "LUT_A" and "LUT_B."

Figure 5-55. PWM and Commutation Schematic

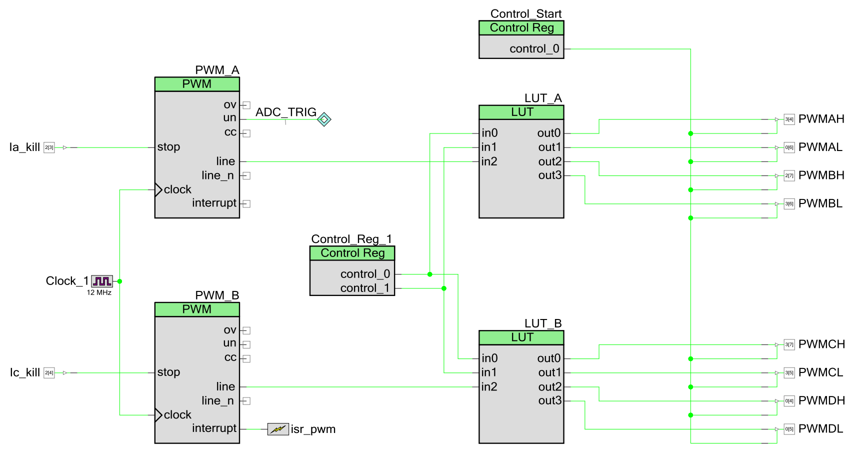


Figure 5-56 shows the “Current Chopping” page. Internal IDACs generate the sinusoidal reference, employing the internal opamps circuit sensing, and amplify the winding current. Then the output of the opamps circuit is compared with the sinusoidal reference current level for each phase in every microstep by external comparators. When the motor phase current is more than the sinusoidal reference current level, the comparator will output a high level.

Figure 5-56. Current Chopping Schematic

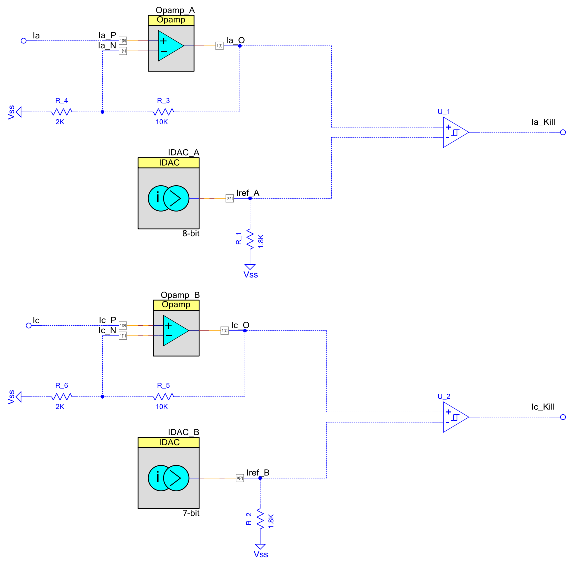
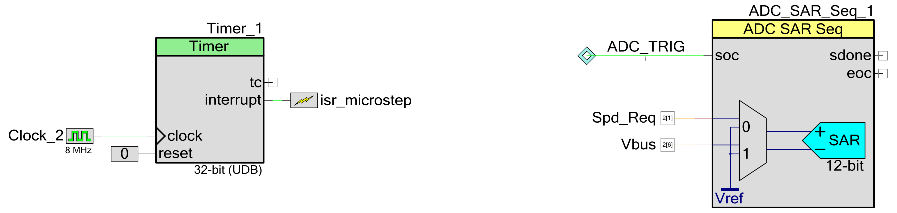


Figure 5-57 shows the “ADC,Button and Microstep Timer” page. “Timer_1” sets the period of the microsteps. Its overflow event triggers an interrupt, and winding commutating and sinusoidal current reference value updating are executed in its ISR. “ADC_SAR_Seq_1” is the SAR ADC to detect and measure bus voltage and potentiometer voltage.

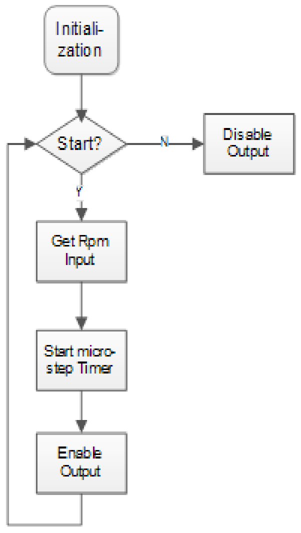
Figure 5-57. ADC,Button and Microstep Timer Schematic



5.6.4 Firmware Introduction

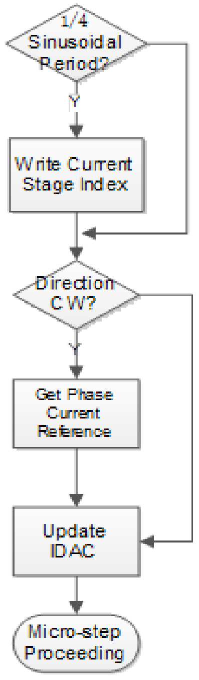
Figure 5-58 shows the flow chart of the main loop function. The program first initializes and configures the internal resources of PSoC 4, and then the main loop detects the user’s start-stop command and RPM reference.

Figure 5-58. Main Loop Function Flow Chart



The microstep drive algorithm is processed in the timer overflow ISR in which the microstep length was set. Figure 5-599 shows the flow chart of the timer overflow ISR.

Figure 5-59. Timer Overflow ISR Flow Chart



5.6.5 Running the Stepper Motor Control Example Project

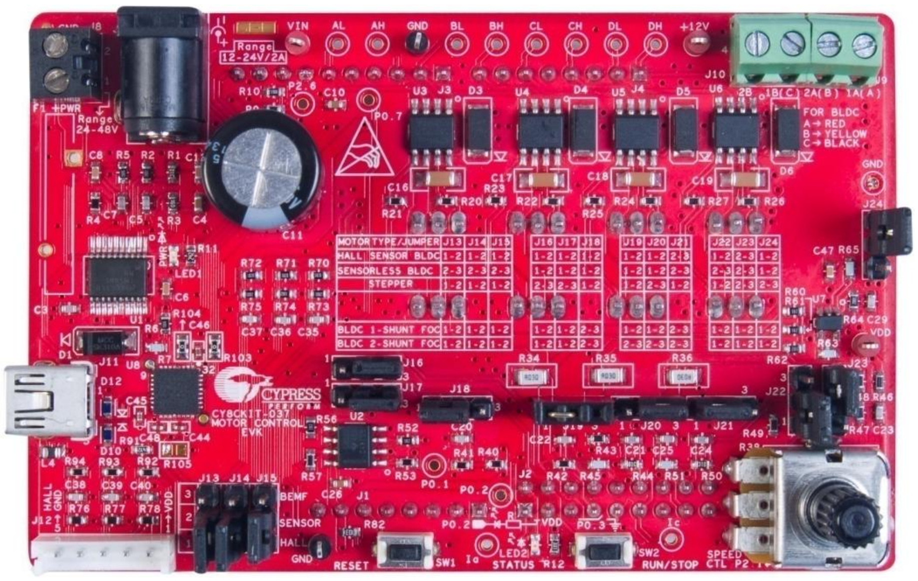
5.6.5.1 Step 1 – Configure CY8CKIT-042

Select 3.3 V as VDD at jumper J9 on the Pioneer Kit. In this application, the USB cable is used only for burning firmware, so it can be removed from CY8CKIT-042 after programming. Figure 5-39 shows the CY8CKIT-042 configuration.

5.6.5.2 Step 2 – Configure CY8CKIT-037

Configure the board via jumpers J13–J24 for the Stepper Motor Control example project ("STEPPER" row in Figure 5-1). Figure 5-60 shows the EVK board configured for the Stepper Motor Control example project.

Figure 5-60. CY8CKIT-037 Configuration for Stepper Motor Control Example Project



5.6.5.3 Step 3 – Plug CY8CKIT-037 into CY8CKIT-042

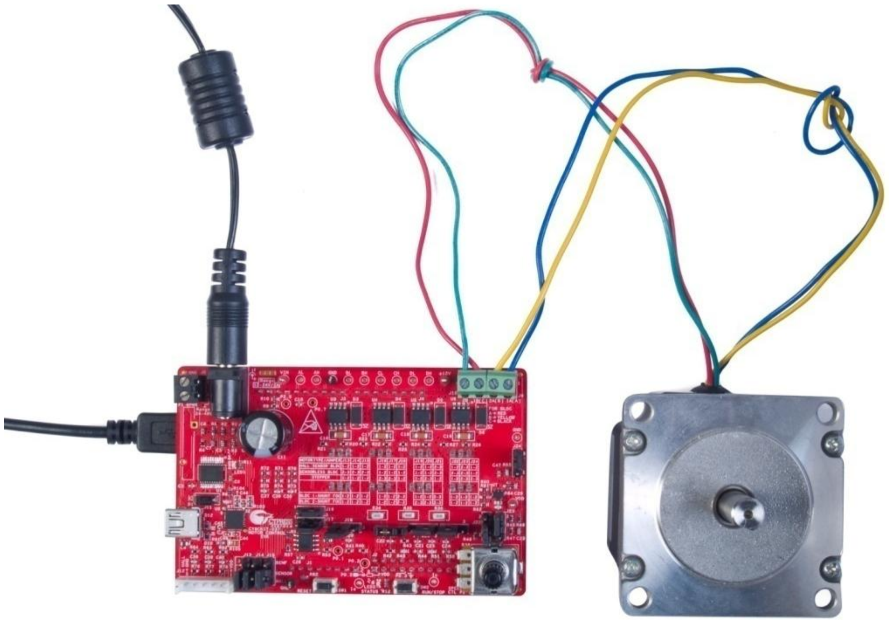
Plug the EVK board into the Pioneer Kit via connectors J1–J4, as shown in Figure 5-11.

5.6.5.4 Step 4 – Connect the Power Supply and Motor

Connect the stepper motor windings to J9 and J10 on the EVK board, as shown in Figure 5-61. The wire sequence and color for the motor windings should be exactly as shown in Figure 5-61. Refer to Motor Winding Connectors for instructions. Then connect the 24-V adapter to J7. When LED1 is red, it indicates power is on. If it is not glowing, the fuse F2 may be broken. Please change the F2 with the provided backup fuse.

CAUTION	**Note: The CY8CKIT-037 PSoC 4 Motor Control EVK package does not include a stepper motor. Cypress recommends that you use a 42-mm (diameter or side length) stepper motor: 42BYGH403AA (with 1.8-degree step angle and 1.65-A phase current). You can also find a stepper motor from other manufacturers with the same size and electrical parameters.**

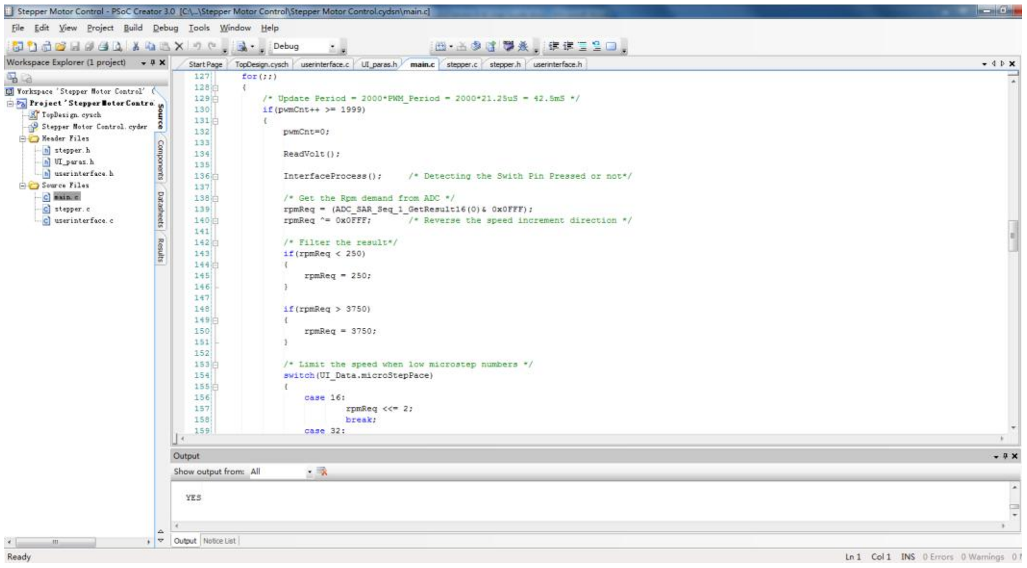
Figure 5-61. Connect Motor and Power Supply



5.6.5.5 Step 5 – Build the Project and Program PSoC 4

Open the Stepper Motor Control example project in PSoC Creator 3.2 or later version, as shown in Figure 5-62. Choose **Build** > **Build Stepper Motor Control** to start building the project, and then choose **Debug** > **Program** to program the chip.

Figure 5-62. Open the Stepper Motor Control Example Project


Stepper Motor Control - PSoC Creator 3.0 [C:\...\Stepper Motor Control\Stepper Motor Control.cydsn\main.c]
File Edit View Project Build Debug Tools Window Help
Debug
Workspace Explorer (1 project)
Workspace 'Stepper Motor Control'
Project 'Stepper Motor Contro
TopDesign.cysch
Stepper Motor Control.cydwr
Header Files
stepper.h
UI_paras.h
userinterface.h
Source Files
main.c
stepper.c
userinterface.c
Source
Components
Datasheets
Results
Start Page
TopDesign.cysch
userinterface.c
UI_paras.h
main.c
stepper.c
stepper.h
userinterface.h
Output
Show output from: All
Output
Notice List
Ready
Ln 1 Col 1 INS 0 Errors 0 Warnings

```
    for(;;)
    {
        /* Update Period = 2000*PWM_Period = 2000*21.25uS = 42.5mS */
        if(pwmCnt++ >= 1999)
        {
            pwmCnt=0;

            ReadVolt();

            InterfaceProcess();     /* Detecting the Swith Pin Pressed or not*/

            /* Get the Rpm demand from ADC */
            rpmReq = (ADC_SAR_Seq_1_GetResult16(0)& 0x0FFF);
            rpmReq ^= 0x0FFF;       /* Reverse the speed increment direction */

            /* Filter the result*/
            if(rpmReq < 250)
            {
                rpmReq = 250;
            }

            if(rpmReq > 3750)
            {
                rpmReq = 3750;
            }

            /* Limit the speed when low microstep numbers */
            switch(UI_Data.microStepPace)
            {
                case 16:
                        rpmReq <<= 2;
                        break;
                case 32:
```

```
YES
```

CAUTION	**During the debugging process, if you modify the code in the firmware, make sure that the changes do not turn on both the high-side and low-side MOSFETs.**

5.6.5.6 Step 6 – Press the SW2 Button to Start Motor Rotation

Press the SW2 button to start motor rotation (refer to Figure 5-14). If the motor does not rotate and you observe LED2 blinking, it indicates that an error has occurred. If so, first check that step 1 through step 5 have executed correctly. Then press the Reset button and press the SW2 button again. If LED2 still blinks, there must be a problem in the hardware or software. You have to debug it or contact Cypress for technical support.

CAUTION **Do not remove jumpers while the kit is powered.**

5.6.6 Adapting the Example Project to another Motor

This example project can be used to control two phases of the HB (hybrid) stepper motor with microstep algorithm. So you need to modify the motor-related parameters in the project firmware according to your specific motor characteristics. You can also modify parameters related to function, like microstep numbers, RPM, and current level, according to your requirements. The main parameters are defined by a `struct` in *UI_paras.h* as follows.

```
typedef struct
{
   uint8   dir;                /*Direction*/
   uint8   microStepPace;      /*For Stepper Motor*/
}UI_DATA;
```

The real `struct` variable is defined in *main.c.*

```
UI_DATA UI_Data;
```

This `struct` variable is initialized in *main.c* by a function as follows.

```
void MotorInit(void)
{
    Control_Start_Write(0);
      UI_Cmd.run = FALSE;
    UI_Data.microStepPace = 128;          /* Set the number of micro-step pace:
                                           microstep number = 128 /
                                           UI_Data.microStepPace */
      UI_Data.dir = CCW;          /* Set rotating direction of motor */
}
```

This function initializes the parameters before the motor begins running. The initializing value is dedicated for the motor shipped from Cypress in the kit package. If you want to use your own motor and have different functional requirements like microstep numbers or direction, change the initializing value in this function.

Note: The number of microsteps is equal to “128/UI_Data.microStepPace.” So, if you set UI_Data.microStepPace=128, there is only one microstep, which means no microstep pace. So the stepper motor generates high mechanical vibration, torque ripple, and noise and may stop abruptly at some mechanical resonance points when the potentiometer is varied. So it is recommended that you set “UI_Data.microStepPace=1, 2, 4, 8, 16, or 32” to enable the microstep algorithm to make the motor run more stably and smoothly.

5.7 Bridge Control Panel Monitor Tool Guide

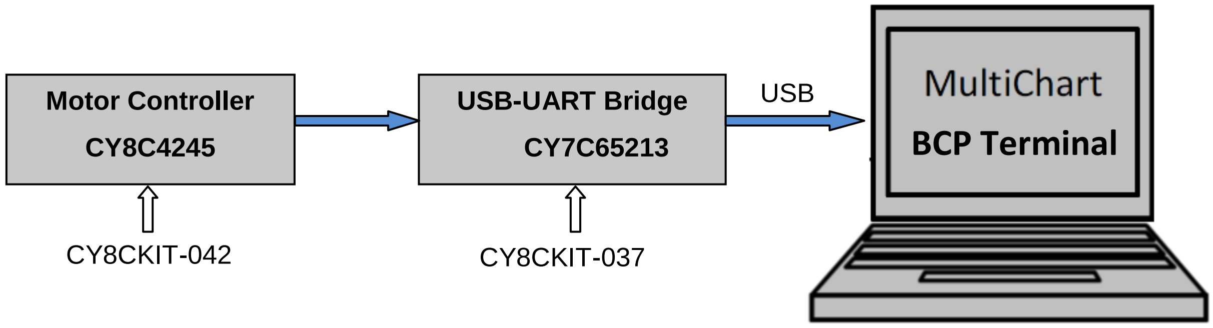
During the motor control debugging process, you cannot use the step debug because it prevents the CPU from running at breakpoint, which leads to blockage of the motor rotation and generates a very high current that burns the motor windings. Therefore, you need to use the real-time debugging tool to monitor the status of the running motor.

The BCP is a very useful and convenient real-time debugging tool provided by Cypress to monitor the motor state in the debugging process. This section introduces how to configure and use the BCP in real-time debugging for the motor control application.

5.7.1 BCP Monitoring Overview

Figure 5-63 shows the system structure of real-time debugging for the motor control application. The controller board (CY8CKIT-042) works as a lower terminal, and the BCP terminal on the PC works as an upper terminal. CY8CKIT-042 sends real-time motor state data through PSoC 4 dedicated UART pins (P4.0 and P4.1), and then data is pushed into the USB-to-UART bridge controller (CY7C65213) on the driver board (CY8CKIT-037). CY7C65213 repackages the data and sends it to the upper terminal through the USB cable, so you can see the curve or chart of the real-time motor state data.

Figure 5-63. Real-Time Debugging Structure

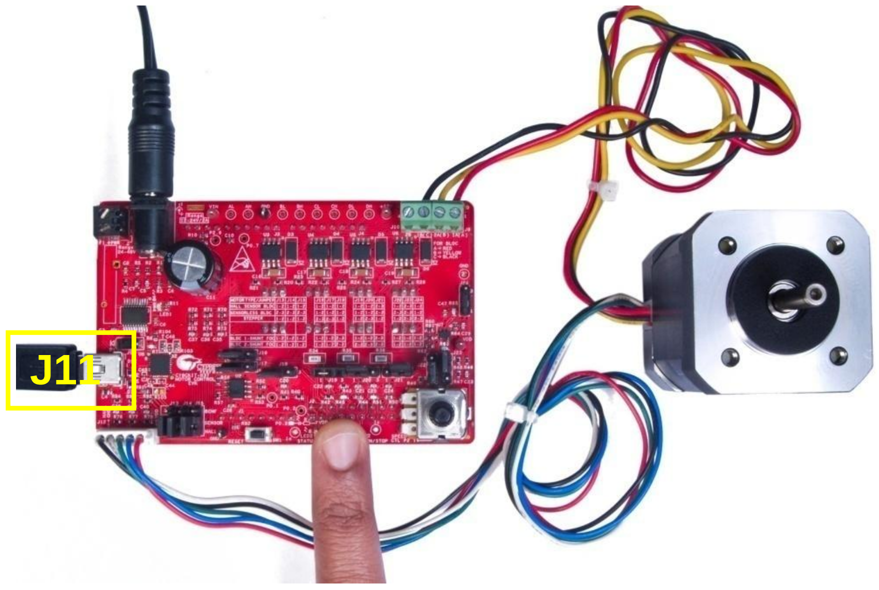


5.7.2 Installing the Driver for CY7C65213 USB-to-UART Bridge Controller

The first step in monitoring data on the BCP is to set up the hardware and install the driver program for the USB-to-UART bridge controller (CY7C65213) on CY8CKIT-037.

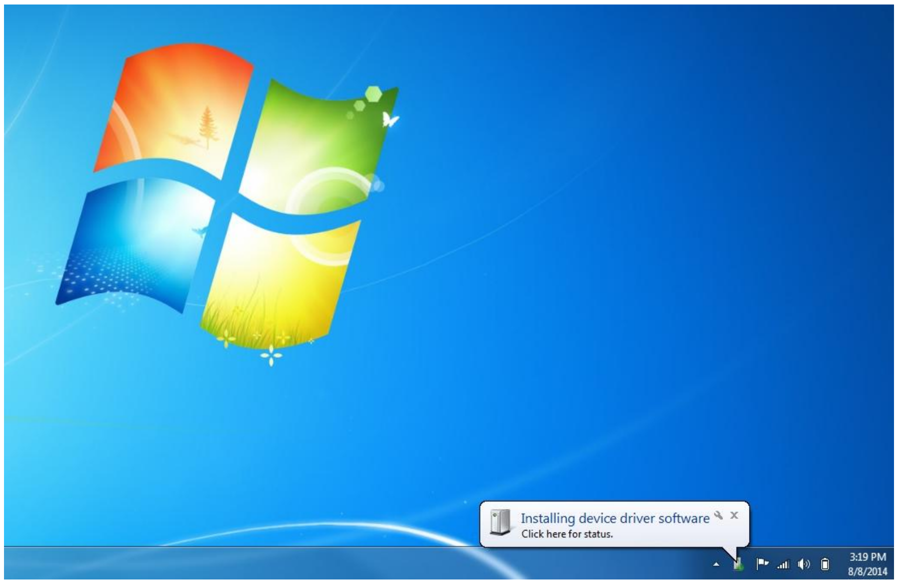
Connect the USB cable to CY8CKIT-037 at connector J11. Then configure the hardware for the Sensored BLDC Motor Control example project following step1 to step 5 in section 5.2.5. Figure 5-64 shows the hardware setup.

Figure 5-64. Hardware Setup for BCP



Connect the other end of J11 to your PC. The PC operating system (Windows 7 in this document) will automatically recognize the CY7C65213 USB-to-UART bridge controller and search the driver program for it, as shown in Figure 5-65.

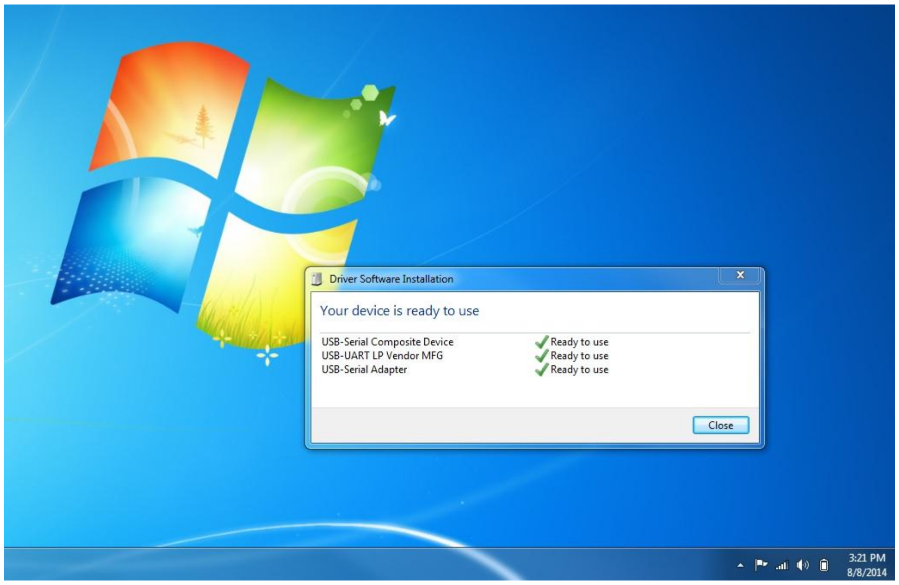
Figure 5-65. CY7C65213 USB-to-UART Bridge Controller Recognition



When the operating system gets the driver program, it will automatically install it for CY7C65213. Figure 5-66 shows the information window that indicates that the driver program installation for the CY7C65213 USB-to-UART Bridge Controller is finished.

CAUTION	**Do not press switch SW2 on the CY8CKIT-037 board during the installation process because the controller board (CY8CKIT-042) is unable to send any data to the USB-to-UART Bridge Controller (CY7C65213) during the installation process. Otherwise, the BCP will not work after the installation. All the example projects included in the kit send data only after the motor is started by pressing the start/stop switch, SW2.**

Figure 5-66. Driver Program Installation Completion



Your device is ready to use

USB-Serial Composite Device	✓ Ready to use
USB-UART LP Vendor MFG	✓ Ready to use
USB-Serial Adapter	✓ Ready to use

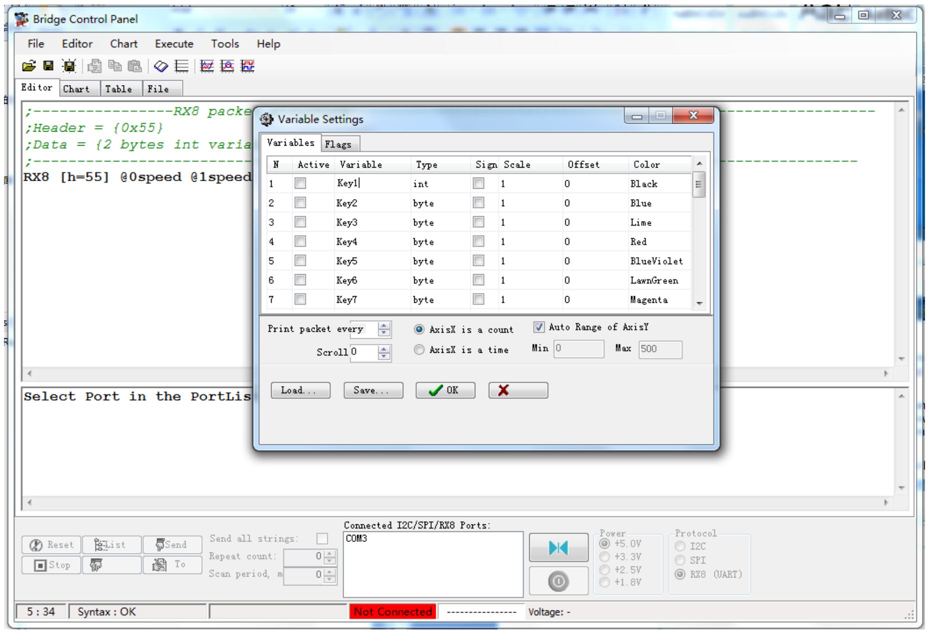
5.7.3 Upper Terminal Configuration Guide

Each example project contains the lower terminal firmware to send real-time data and attaches two upper terminal configuration files, *.ini* and *.iic*.

This section uses the Sensored BLDC Motor Control example project as an example. Other example projects are similar.

Enter the BCP software interface, and choose **Chart** > **Variable Settings** to open the **Variable Settings** window, as Figure 5-67 shows.

Figure 5-67. Variable Settings Window



```
;----------------RX8 packe
;Header = {0x55}
;Data = {2 bytes int varia
;--------------------------
RX8 [h=55] @0speed @1speed
```

```
Select Port in the PortLis
```

N	Active	Variable	Type	Sign	Scale	Offset	Color
1	☐	Key1	int	☐	1	0	Black
2	☐	Key2	byte	☐	1	0	Blue
3	☐	Key3	byte	☐	1	0	Lime
4	☐	Key4	byte	☐	1	0	Red
5	☐	Key5	byte	☐	1	0	BlueViolet
6	☐	Key6	byte	☐	1	0	LawnGreen
7	☐	Key7	byte	☐	1	0	Magenta

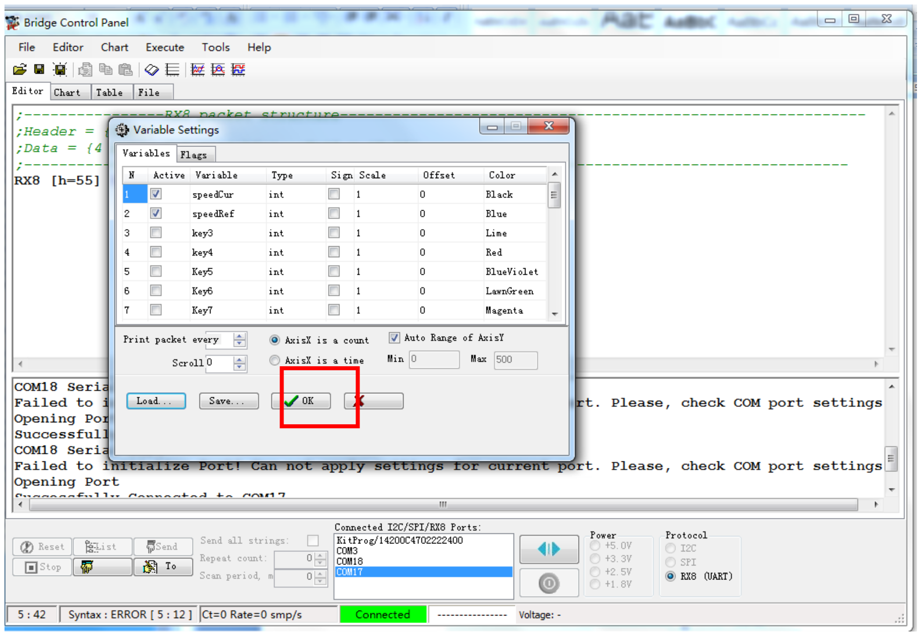
Print packet every; Scroll 0; [✓ AxisX is a count] AxisX is a time; ✓ Auto Range of AxisY; Min 0; Max 500

Connected I2C/SPI/RX8 Ports: COM3

5 : 34 | Syntax : OK | Not Connected | ---------------- | Voltage: -

Then click the **Load** button to import the *variables.ini* file into the Sensored BLDC Motor Control example project, as Figure 5-68 shows. Click the **OK** button to finish.

Figure 5-68. Import variables.ini File



```
;--------------RX8 packet structure-------------------------
;Header = {
;Data = {4
;--------------------------
RX8 [h=55]
```

N	Active	Variable	Type	Sign	Scale	Offset	Color
1	✓	speedCur	int	☐	1	0	Black
2	✓	speedRef	int	☐	1	0	Blue
3	☐	key3	int	☐	1	0	Lime
4	☐	key4	int	☐	1	0	Red
5	☐	Key5	int	☐	1	0	BlueViolet
6	☐	Key6	int	☐	1	0	LawnGreen
7	☐	Key7	int	☐	1	0	Magenta

Print packet every; Scroll 0; [✓ AxisX is a count] AxisX is a time; ✓ Auto Range of AxisY; Min 0; Max 500

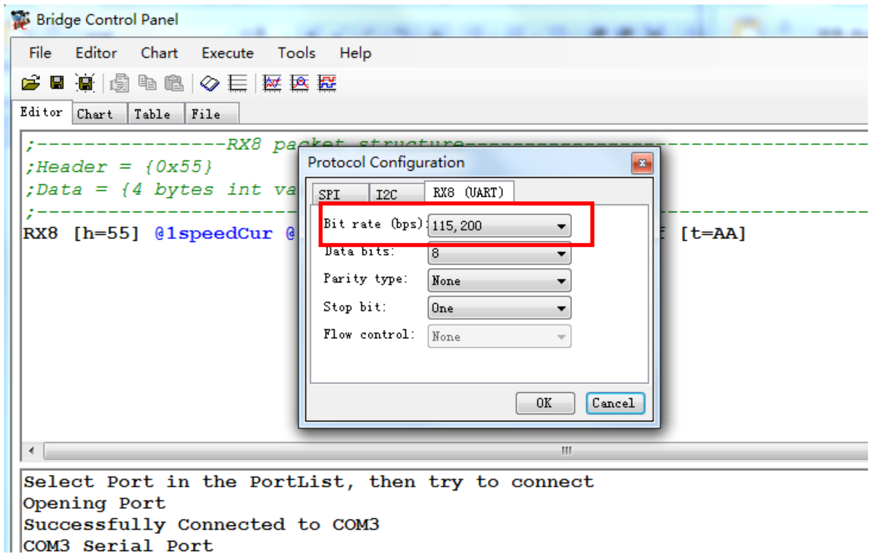
```
COM18 Seria
Failed to i                                          rt. Please, check COM port settings
Opening Por
Successfull
COM18 Seria
Failed to initialize Port! Can not apply settings for current port. Please, check COM port settings
Opening Port
Successfully Connected to COM17
```

Connected I2C/SPI/RX8 Ports:
KitProg/14200C4702222400
COM3
COM18
COM17

5 : 42 | Syntax : ERROR [5 : 12] | Ct=0 Rate=0 smp/s | Connected | ---------------- | Voltage: -

You also need to set the baud of BCP communication. Enter the BCP software interface, and choose **Tools** > **Protocol Configuration** to open the **Protocol Configuration** window, as Figure 5-69 shows. The recommended baud setting is 115,200 bps.

Figure 5-69. Baud (bps) Settings Window



```
;----------------RX8 packet structure----------------------------
;Header = {0x55}
;Data = {4 bytes int va
;-------------
RX8 [h=55] @1speedCur @                                  [t=AA]
```

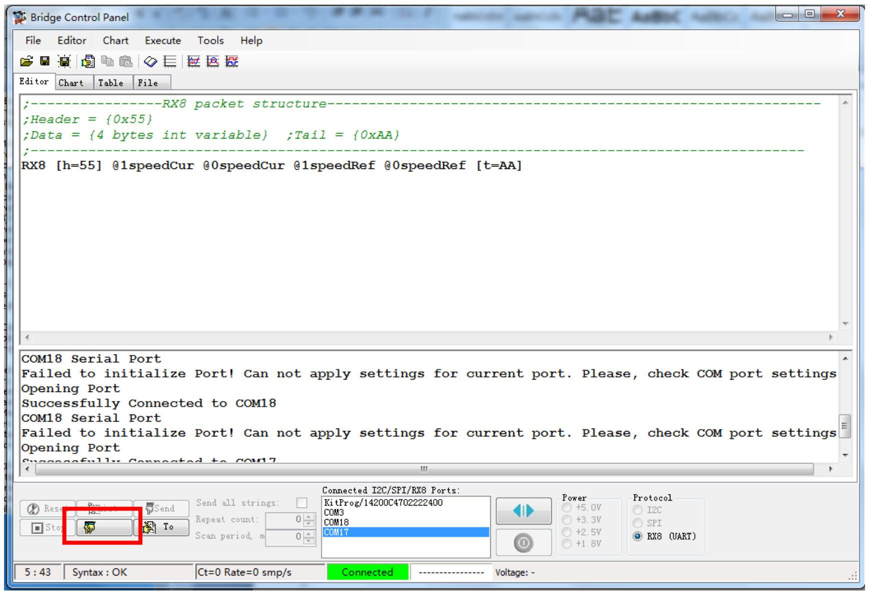
Bit rate (bps): 115, 200
Data bits: 8
Parity type: None
Stop bit: One
Flow control: None

```
Select Port in the PortList, then try to connect
Opening Port
Successfully Connected to COM3
COM3 Serial Port
```

Choose **File** > **Open File** to import the *Sensored BLDC Control.iic* configuration file into the Sensored BLDC Motor Control example project, as Figure 5-70 shows. Click the highlighted button to start monitoring data defined by the command in the **Editor** and **Variable Settings** windows.

The **Editor** window shows the BCP monitor 2 variables as a default. “SpeedCur” shows the real-time motor speed, and “SpeedRef” shows the reference motor speed. You can change or add monitoring variables by changing the command statement in the **Editor** window and keeping the **Variable Settings** window updated synchronously.

Figure 5-70. Bridge Control Panel Editor



```
;----------------RX8 packet structure-----------------------------------------------------------
;Header = {0x55}
;Data = {4 bytes int variable}  ;Tail = {0xAA}
;---------------------------------------------------------------------------------------------
RX8 [h=55] @1speedCur @0speedCur @1speedRef @0speedRef [t=AA]
```

```
COM18 Serial Port
Failed to initialize Port! Can not apply settings for current port. Please, check COM port settings
Opening Port
Successfully Connected to COM18
COM18 Serial Port
Failed to initialize Port! Can not apply settings for current port. Please, check COM port settings
Opening Port
Successfully Connected to COM17
```

Send all strings:
Repeat count: 0
Scan period, m 0

Connected I2C/SPI/RX8 Ports:
KitProg/14200C4702222400
COM3
COM18
COM17

5 : 43 | Syntax : OK | Ct=0 Rate=0 smp/s | Connected | ---------------- | Voltage: -

The command structure consists of four parts: start symbol “Rx8”, header, body, and tail as follows.

Rx8 [H=<byte0> <byte1> … <byteN>] <byte0> <byte1> … <byteK> [T=<byte0> <byte1> … <byteM>]

Where “Rx8” is a start symbol that indicates the beginning of the RX8 command.

[H=<byte0> <byte1> … <byteN>] is a header with a list of bytes. It is intended for synchronization of the BCP to an asynchronous input stream. The header part is optional. It can be present or absent in the script. But if it is absent, then the tail also should be absent. Otherwise, the ScriptEngine syntax analyzer will inform you about a syntax error and mark in red the incorrect part of the command.

<byte0> <byte1> … <byteK> is a body list of bytes. It is a useful packet load that is received by the host and shown in the Log and Chart subsystems, depending on the variables configuration in the command. The body is a mandatory part of the command.

[T=<byte0> <byte1> … <byteM>] is a tail with a list of bytes. It indicates the end of the RX8 command. The tail is an optional part of the command. The RX8 command can be present in the BCP script editor analyzer in three ways:

1) Without the header and tail, with only the body. In this scenario, the host does not synchronize with a stream of bytes. It simply prints bytes received from the MiniProg3 to the Log and on the Chart if variables were configured in the script, as shown in the following example.

RX8 X X @1Var1 @0Var1 X @3Var2 @2Var2 @1Var2 @0Var2 X X

2) With the header and body, but without the tail. In this scenario, the host synchronizes with a stream of bytes, finding header bytes in the stream, as shown in the following example.

RX8 [H = 0A 0E 0C] X X @1Var1 @0Var1 X @3Var2 @2Var2 @1Var2 @0Var2 X X

3) With the header, body, and tail. In this scenario, the host synchronizes with a stream of bytes, finding header, body, and tail bytes in the stream, as shown in the following example.

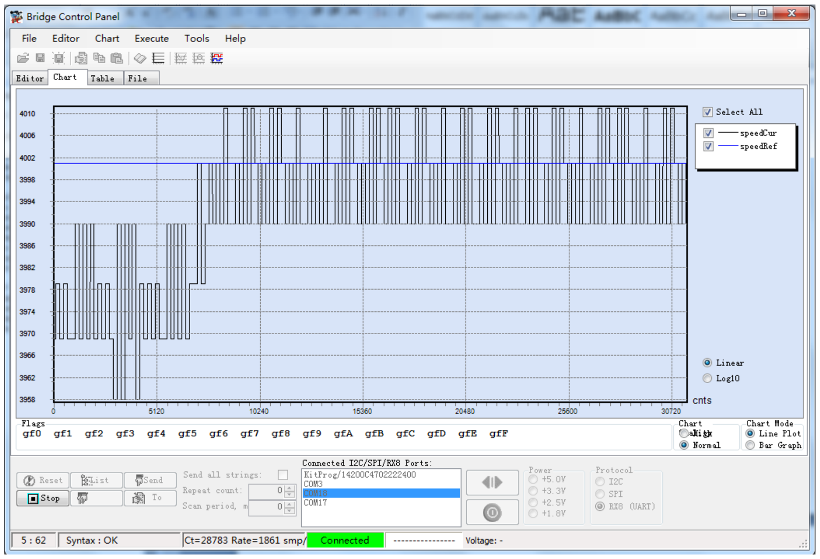
RX8 [H = 0A 0E 0C] X X @1Var1 @0Var1 X @3Var2 @2Var2 @1Var2 @0Var2 X X [T = 00 FF FF]

All five example projects use this command format: H = 0x55 and T =0XAA. Variables are 16 bits and always in “MSB-LSB” format.

You can define your own commands by following these rules.

Keep the default configuration, Press SW2 on CY8CKIT-037 to start the motor. You can see the real-time data in the **Chart** window, as Figure 5-71 shows. You can use a tachometer to measure the motor speed and check it on the BCP display.

Figure 5-71. BCP Data Display for Sensored BLDC Motor Control Example Project



Send all strings: ☐
Repeat count: 0
Scan period, m 0

Connected I2C/SPI/RX8 Ports:
KitProg/14200C4702222400
COM3
COM18
COM17

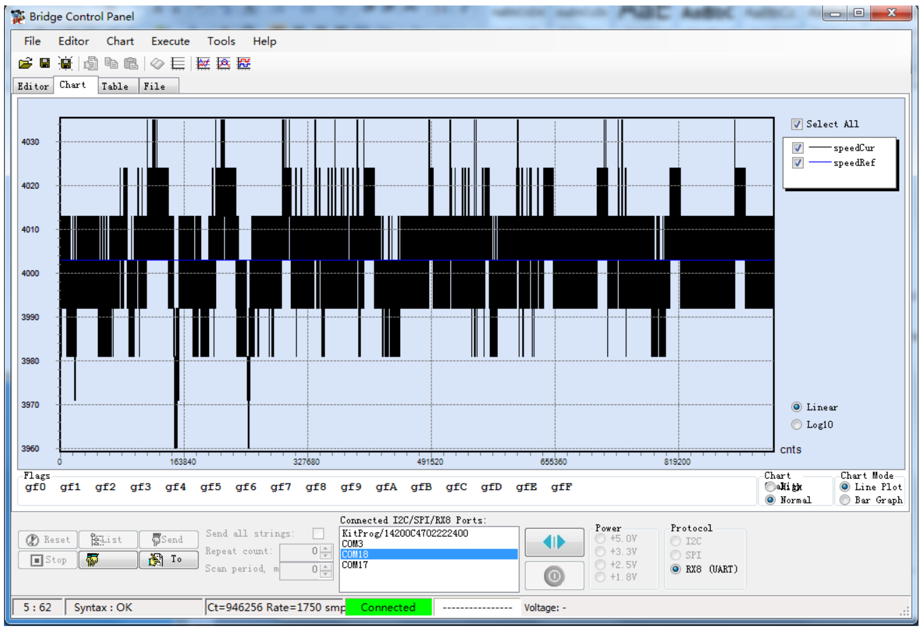
5 : 62 | Syntax : OK | Ct=28783 Rate=1861 smp/ | Connected | --------------- | Voltage: -

Note: Before you complete all the configuration steps in this section (from Figure 5-67 to Figure 5-70), ensure that switch SW2 on CY8CKIT-037 is not pressed. If you press it, you will not see data display on the BCP **Chart** window. However, if you do press SW2, press SW1 on CY8CKIT-037 to reset the motor, and then complete all the configuration steps shown in Figure 5-67 to Figure 5-70. You can then press SW2 to start the motor and observe the data on the BCP **Chart** window.

If you want to run the motor and monitor data for a long time (more than 8 minutes), the BCP can display only the original 81920 counts (about 7 minutes at 115,200 bps), as Figure 5-72 shows. The later data no longer appears because the

BCP puts all the received data for display in limited stack memory. So if the stack memory is full, the BCP cannot receive and display new data. If you want to monitor the real-time data for a long time, reconfigure the BCP to make it display only the latest data, not all the data from the motor or BCP start.

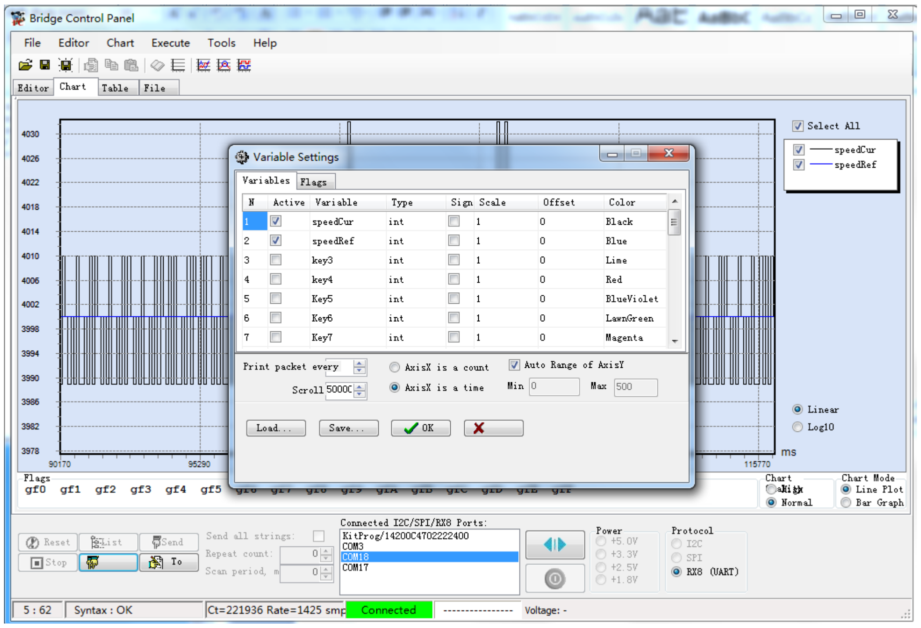
Figure 5-72. Maximum Data Counts on BCP Monitoring



5 : 62 | Syntax : OK | Ct=946256 Rate=1750 smp | Connected | Voltage: -

Choose **Chart** > **Variable Settings** to open the **Variable Settings** window. Then select the “AxisX is a time” option and write a number (minimum: 1, maximum: 500000, unit: ms) into the **Scroll** text box to define the latest time length to make the BCP display data, as Figure 5-73 shows. Then the stack memory will never be full, and the BCP can always show the latest data of the defined time length.

Figure 5-73. BCP Reconfiguration



N	Active	Variable	Type	Sign	Scale	Offset	Color
1	✓	speedCur	int		1	0	Black
2	✓	speedRef	int		1	0	Blue
3		key3	int		1	0	Lime
4		key4	int		1	0	Red
5		Key5	int		1	0	BlueViolet
6		Key6	int		1	0	LawnGreen
7		Key7	int		1	0	Magenta

Print packet every
Scroll 5000C
AxisX is a count [✓ AxisX is a time]
✓ Auto Range of AxisY
Min 0 Max 500

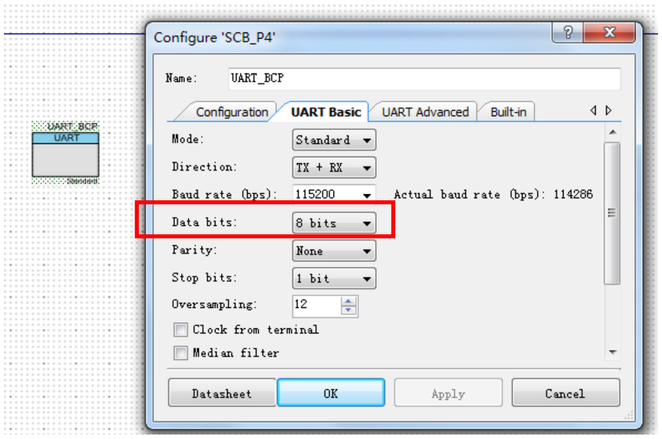
Connected I2C/SPI/RX8 Ports: KitProg/14200C4702222400, COM3, COM18, COM17

5 : 62 | Syntax : OK | Ct=221936 Rate=1425 smp | Connected | Voltage: -

5.7.4 Lower Terminal Configuration Guide

While the motor is running, MCU CY8C4245 on the controller board gets all motor state parameters and is responsible for sending them to the BCP upper terminal. Figure 5-74 shows the Serial Communication Block (SCB) Component and configuration window. It sends motor state data through two dedicated UART pins (P4.0 and P4.1). It is configured as "UART."

Figure 5-74. SCB Component Configuration



Configure 'SCB_P4'

Name: UART_BCP

Mode: Standard
Direction: TX + RX
Baud rate (bps): 115200 Actual baud rate (bps): 114286
Data bits: 8 bits
Parity: None
Stop bits: 1 bit
Oversampling: 12
Clock from terminal
Median filter

Data is sent by function `BCPPoll()`, located in the main control loop of the firmware. The function code follows.

```
void BCPPoll()
{
    uint8 index = 0;

    if(UART_BCP_SpiUartGetTxBufferSize())
       return;

    /* package header */
    bcpTxBuffer[index++] = 0x55;

    /* construct BCP data pacakge with speed value, MSB first */

    /* current measured speed */
    bcpTxBuffer[index++] = (uint8)((UI_Data.speedRpm & 0xFF00) >> 8);
    bcpTxBuffer[index++] = (uint8)(UI_Data.speedRpm & 0x00FF);
    /* speed reference */
    bcpTxBuffer[index++] = (uint8)((UI_Data.speedRpmRef & 0xFF00) >> 8);
    bcpTxBuffer[index++] = (uint8)(UI_Data.speedRpmRef & 0x00FF);

    /* package tail */
    bcpTxBuffer[index++] = 0xAA;

    UART_BCP_SpiUartPutArray(bcpTxBuffer, index);
}
```

This function sends two motor state variables: `UI_Data.speedRpm` and `UI_Data.speedRpmRef`. You can modify the variable name and add variables by copying the highlighted code and then changing the variable name from `UI_Data.speedRpm` to whatever you want.

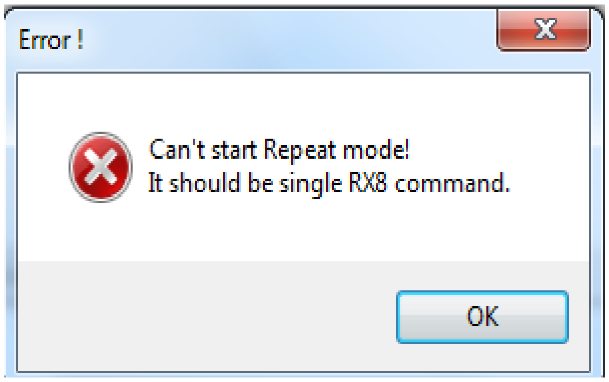
5.7.5 Reading the Motor Speed Using BCP Commands

The BCP commands are not functional if the tool is invoked after the motor starts rotating.

To see the proper motor rotation status or RPM readings, follow these steps.

1. Connect and configure CY8CKIT-037 on the CY8CKIT-042 Pioneer Kit per the Sensored BLDC project.(See Section 5.2.5
2. Program the HEX file.
3. Connect a USB cable at the J11 port of CY8CKIT-037 to enable the motor speed to be viewed in the BCP. (Wait until driver enumeration is complete.)
4. Open the BCP and import the UART commands related to the Sensored BLDC project. Refer section 5.7.3.
5. Select the appropriate COM port related to USB-to-UART on CY8CKIT-037 (example: COM17 in Figure 5-70. Bridge Control Panel Editor).
6. Move the cursor to the end of the BCP command "RX8 [h=55] @1speedCur @0speedCur @1speedRef @0speedRef [t=AA]" in the **Command** window, and press **Enter** to execute the BCP command once. Then click the **Repeat** button to repeat the command execution. (If this operation is missing, an error message window pops up as shown in Figure 5-75)

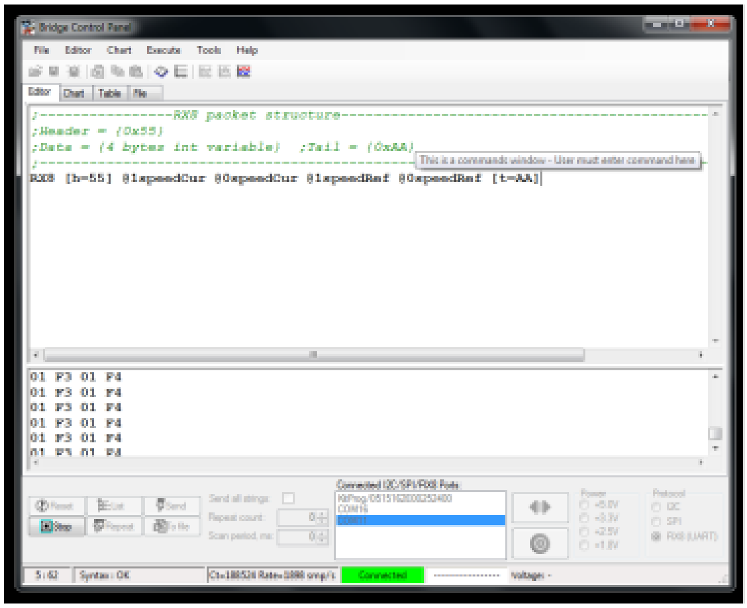
Figure 5-75.Error Message that Appears when Step 6 is Missed while Configuring BCP


Error !

Can't start Repeat mode!
It should be single RX8 command.

7. Press button SW2 on CY8CKIT-037 to start the BLDC motor rotation.
8. Observe the motor speed on the BCP Editor window (see Figure 5-76) and in Chart Window (see Figure 5-77).

Figure 5-76. Motor Speed in Editor Window

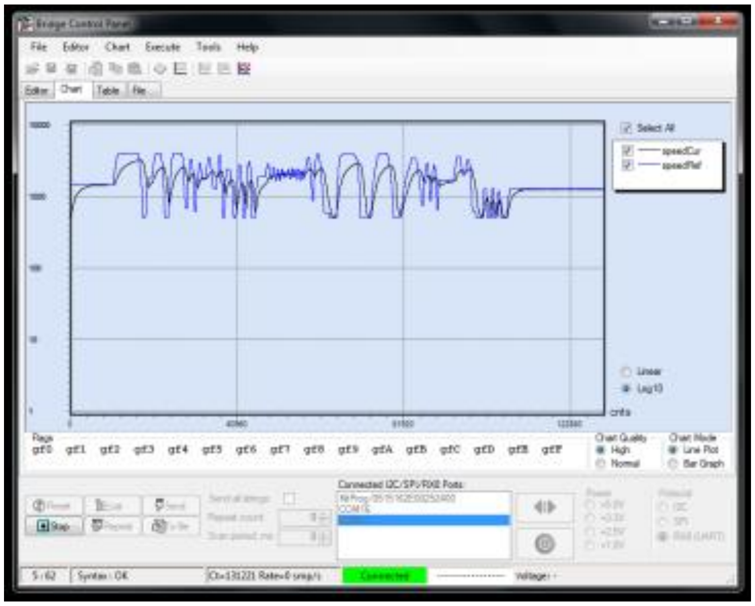


```
;----------------RX8 packet structure--------------------------------------------
;Header = (0x55)
;Data = (4 bytes int variable)  ;Tail = (0xAA)
;--------------------------------------------------------------------------------
RX8 [h=55] @1speedCur @0speedCur @1speedRef @0speedRef [t=AA]
```

```
01 F3 01 F4
01 F3 01 F4
01 F3 01 F4
01 F3 01 F4
01 F3 01 F4
01 F3 01 F4
```

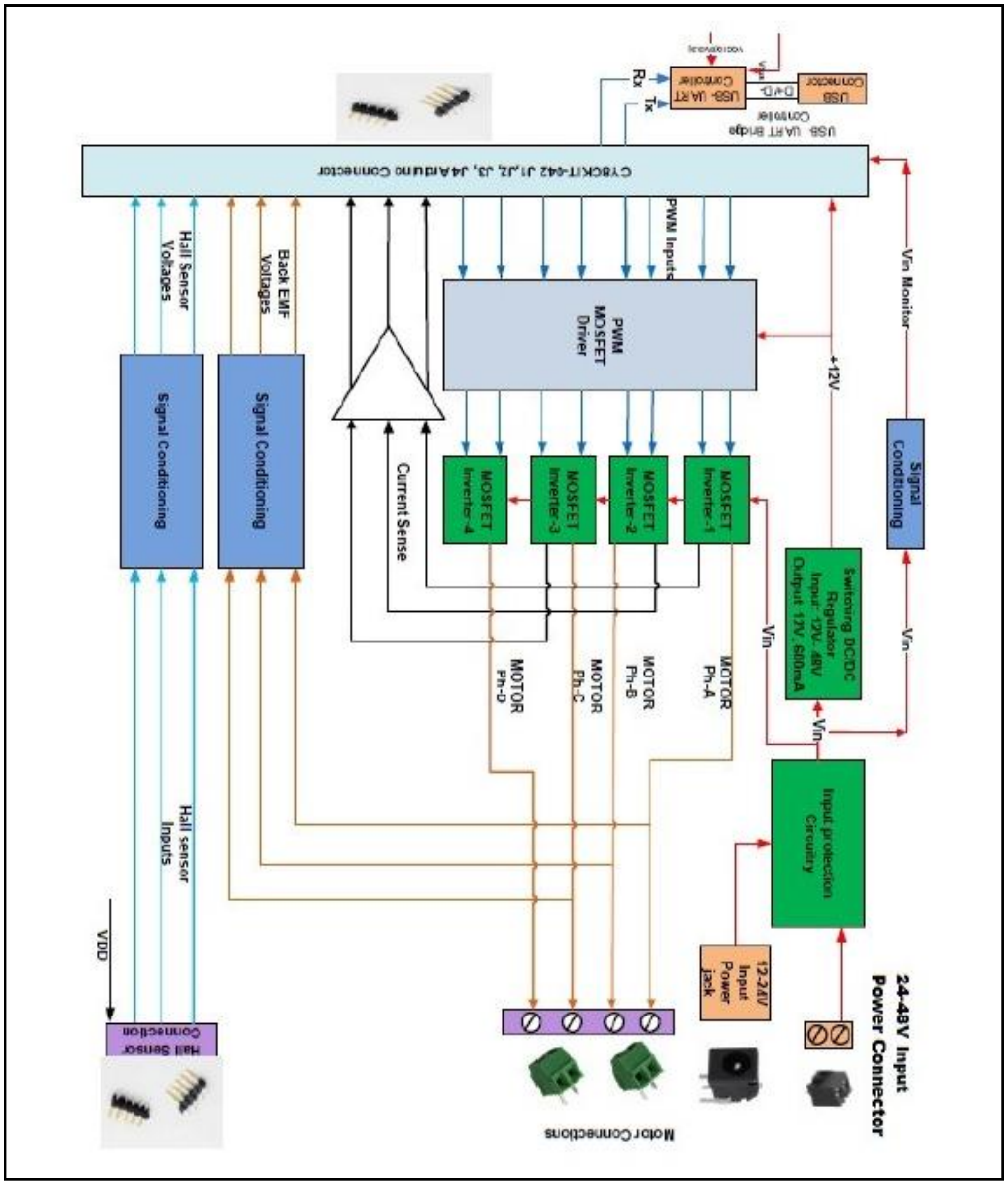
Bridge Control Panel
File Editor Chart Execute Tools Help
Editor Chart Table File
This is a commands window - User must enter command here
Connected I2C/SPI/RX8 Ports
COM16
Reset List Send Stop Repeat To file
Send all strings
Repeat count
Scan period, ms
Power
Protocol
I2C
SPI
RX8 (UART)
Syntax : OK
Connected
Voltage: -

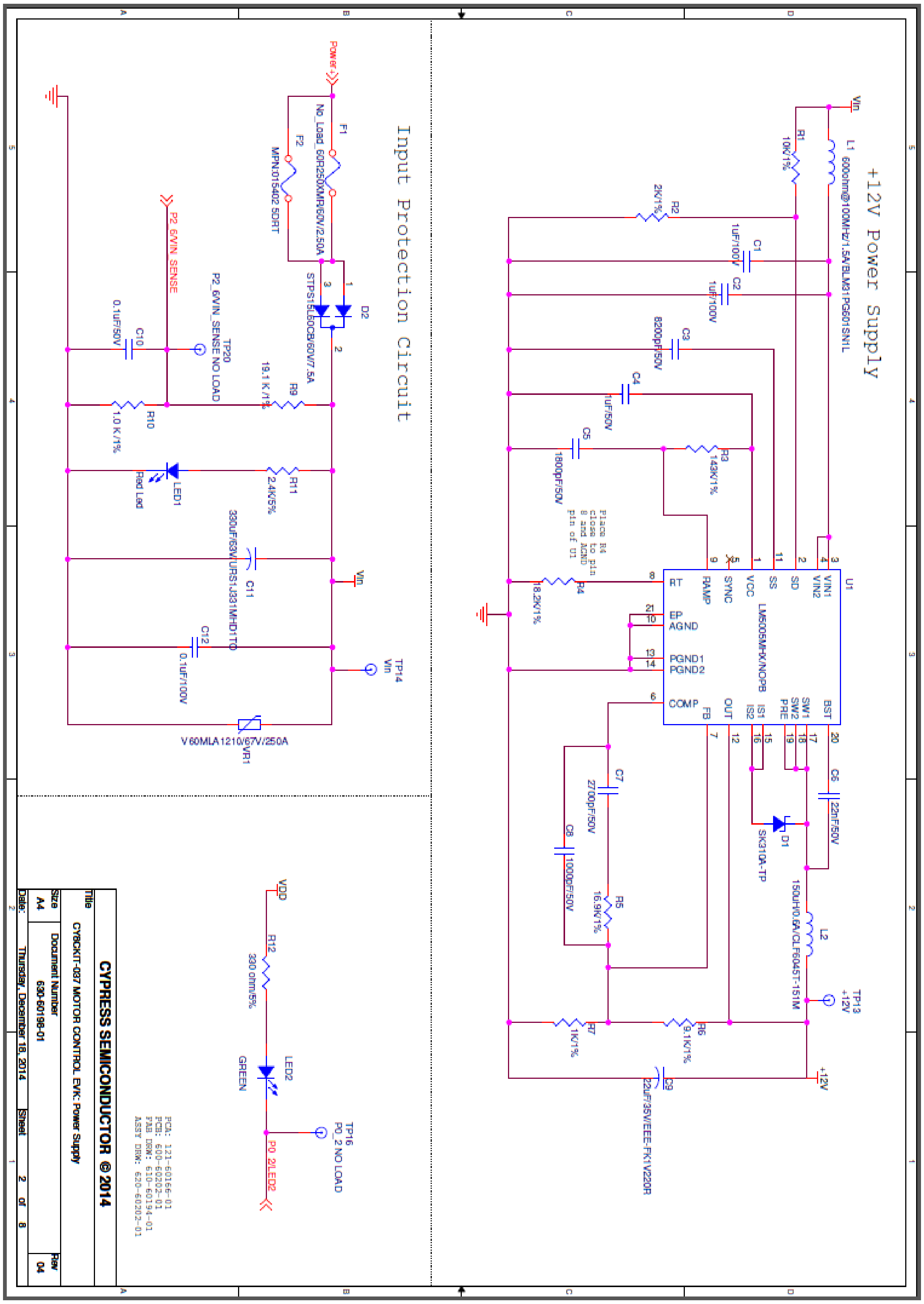
Figure 5-77.Motor Speed in Chart Window

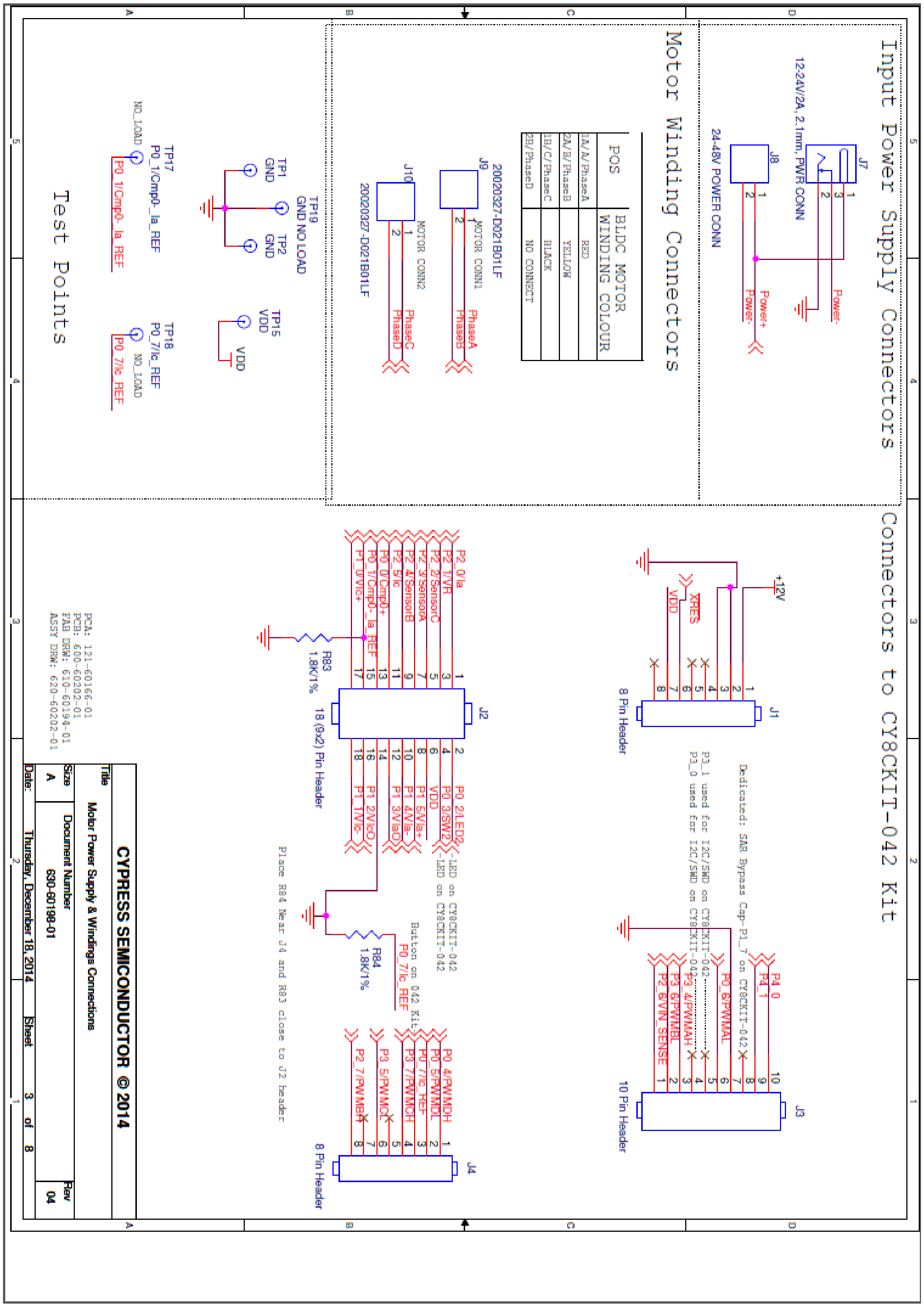


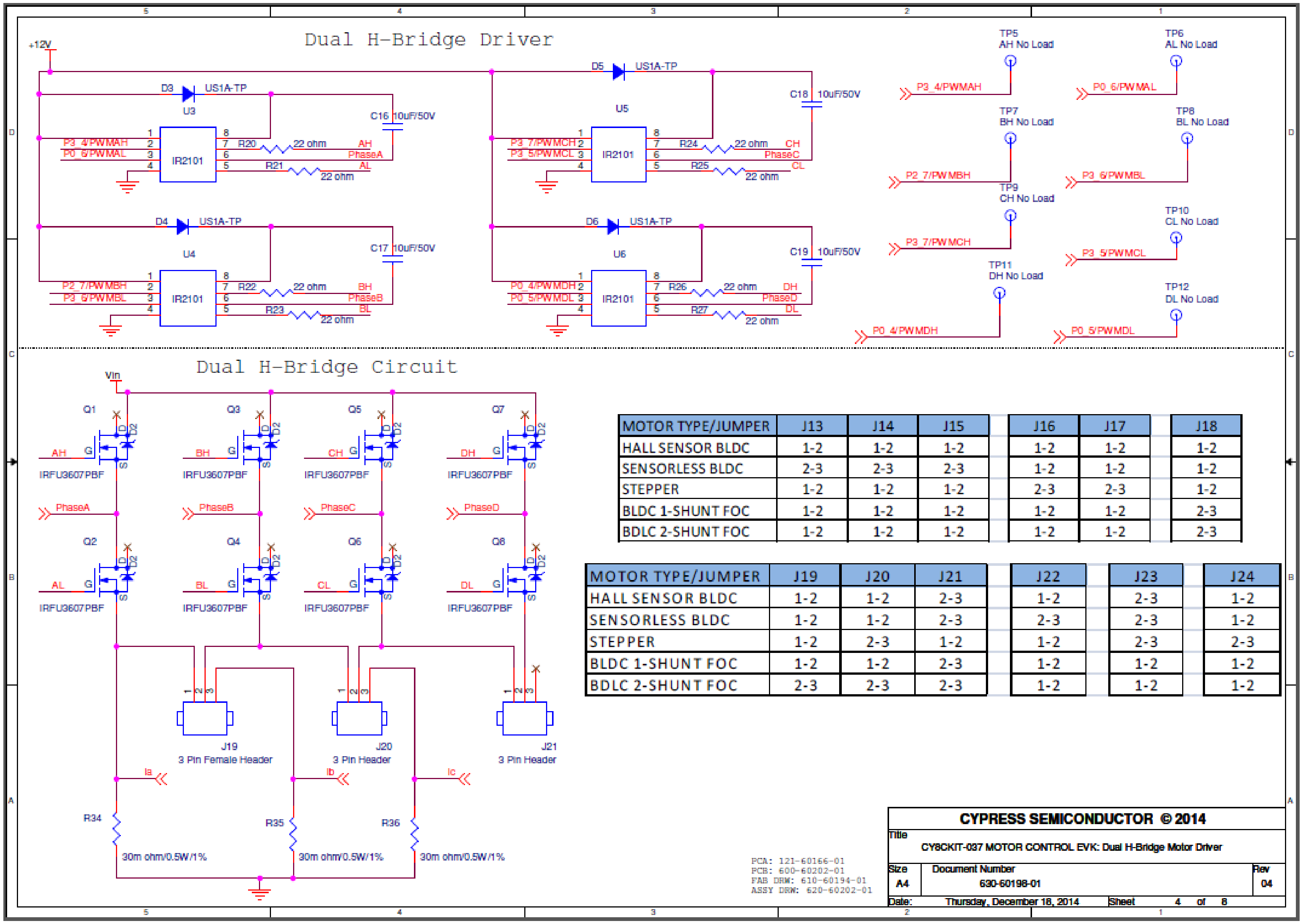
Appendix A: Board Schematics, Board Layout, and BOM

A.1 Board Schematics




+12V Power Supply
Vin
L1 600ohm@100MHz/1.5A/BLM31PG601SN1L
R1 10K/1%
R2 2K/1%
C1 1uF/100V
C2 1uF/100V
C3 8200pF/50V
C4 1uF/50V
C5 1800pF/50V
R3 143K/1%
Place R4 close to pin 8 and AGND pin of U1
R4 18.2K/1%
U1 LM5005MH-X/NOPB
VIN1 VIN2 SD SS VCC SYNC RAMP RT EP AGND PGND1 PGND2 COMP FB OUT IS1 IS2 PRE SW2 SW1 BST
C6 22nF/50V
D1 SK310A-TP
L2 150uH/0.6A/CLF6045T-151M
TP13 +12V
C7 2700pF/50V
C8 1000pF/50V
R5 16.9K/1%
R6 9.1K/1%
R7 1K/1%
C9 22uF/35V/EEE-FK1V220R
+12V
Input Protection Circuit
Power
F1 No_Load_60R250XMR/60V/2.50A
F2 MPN015402 5DRT
D2 STPS15L60CB/60V/7.5A
R9 19.1 K /1%
R10 1.0 K /1%
P2_6/VIN_SENSE
TP20 P2_6/VIN_SENSE NO LOAD
C10 0.1uF/50V
R11 2.4K/5%
LED1 Red Led
C11 330uF/63V/URS1J331MHD1TD
C12 0.1uF/100V
Vin
TP14 Vin
VR1 V60MLA1210/67V/250A
VDD
R12 330 ohm/5%
LED2 GREEN
TP16 P0_2 NO LOAD
P0_2/LED2
PCN: 121-60166-01
PCB: 600-60202-01
FAB DRW: 610-60194-01
ASSY DRW: 620-60202-01
CYPRESS SEMICONDUCTOR © 2014
Title CY8CKIT-037 MOTOR CONTROL EVK: Power Supply
Size A4
Document Number 630-60198-01
Rev 04
Date: Thursday, December 18, 2014
Sheet 2 of 8


Input Power Supply Connectors
J7
12-24V/2A, 2.1mm, PWR CONN
J8
24-48V POWER CONN
Power+
Power-
Motor Winding Connectors
POS
BLDC MOTOR WINDING COLOUR
1A/A/PhaseA
RED
2A/B/PhaseB
YELLOW
1B/C/PhaseC
BLACK
2B/PhaseD
NO CONNECT
J9
20020327-D021B01LF
MOTOR CONN1
PhaseA
PhaseB
J10
20020327-D021B01LF
MOTOR CONN2
PhaseC
PhaseD
Test Points
TP1 GND
TP2 GND
TP19 GND NO LOAD
TP15 VDD
VDD
TP17 P0_1/Cmp0-_Ia_REF
NO_LOAD
P0_1/Cmp0-_Ia_REF
TP18 P0_7/Ic_REF
NO_LOAD
P0_7/Ic_REF
Connectors to CY8CKIT-042 Kit
+12V
XRES
VDD
J1
8 Pin Header
Dedicated: SAR Bypass Cap-P1_7 on CY8CKIT-042
P3_1 used for I2C/SWD on CY8CKIT-042
P3_0 used for I2C/SWD on CY8CKIT-042
P4_0
P4_1
P0_6/PWMAL
P3_4/PWMAH
P3_6/PWMBL
P2_6/VIN_SENSE
J3
10 Pin Header
P2_0/Ia
P2_1/VR
P2_2/SensorC
P2_3/SensorA
P2_4/SensorB
P2_5/Ic
P0_0/Cmp0+
P0_1/Cmp0-_Ia_REF
P1_0/Vic+
R83
1.8K/1%
J2
18 (9x2) Pin Header
P0_2/LED2
P0_3/SW2
VDD
P1_5/Ia+
P1_4/Ia-
P1_3/IaO
P1_2/IcO
P1_1/Vic-
LED on CY8CKIT-042
LED on CY8CKIT-042
Button on 042 Kit
P0_7/Ic_REF
R84
1.8K/1%
Place R84 Near J4 and R83 close to J2 header
P0_4/PWMDH
P0_5/PWMDL
P0_7/Ic_REF
P3_7/PWMCH
P3_5/PWMCL
P2_7/PWMBH
J4
8 Pin Header
PCA: 121-60166-01
PCB: 600-60202-01
FAB DRW: 610-60194-01
ASSY DRW: 620-60202-01
CYPRESS SEMICONDUCTOR © 2014
Title
Motor Power Supply & Windings Connections
Size A
Document Number 630-60198-01
Rev 04
Date: Thursday, December 18, 2014
Sheet 3 of 8



MOTOR TYPE/JUMPER	J13	J14	J15	J16	J17	J18
HALL SENSOR BLDC	1-2	1-2	1-2	1-2	1-2	1-2
SENSORLESS BLDC	2-3	2-3	2-3	1-2	1-2	1-2
STEPPER	1-2	1-2	1-2	2-3	2-3	1-2
BLDC 1-SHUNT FOC	1-2	1-2	1-2	1-2	1-2	2-3
BDLC 2-SHUNT FOC	1-2	1-2	1-2	1-2	1-2	2-3

MOTOR TYPE/JUMPER	J19	J20	J21	J22	J23	J24
HALL SENSOR BLDC	1-2	1-2	2-3	1-2	2-3	1-2
SENSORLESS BLDC	1-2	1-2	2-3	2-3	2-3	1-2
STEPPER	1-2	2-3	1-2	1-2	2-3	2-3
BLDC 1-SHUNT FOC	1-2	1-2	2-3	1-2	1-2	1-2
BDLC 2-SHUNT FOC	2-3	2-3	2-3	1-2	1-2	1-2

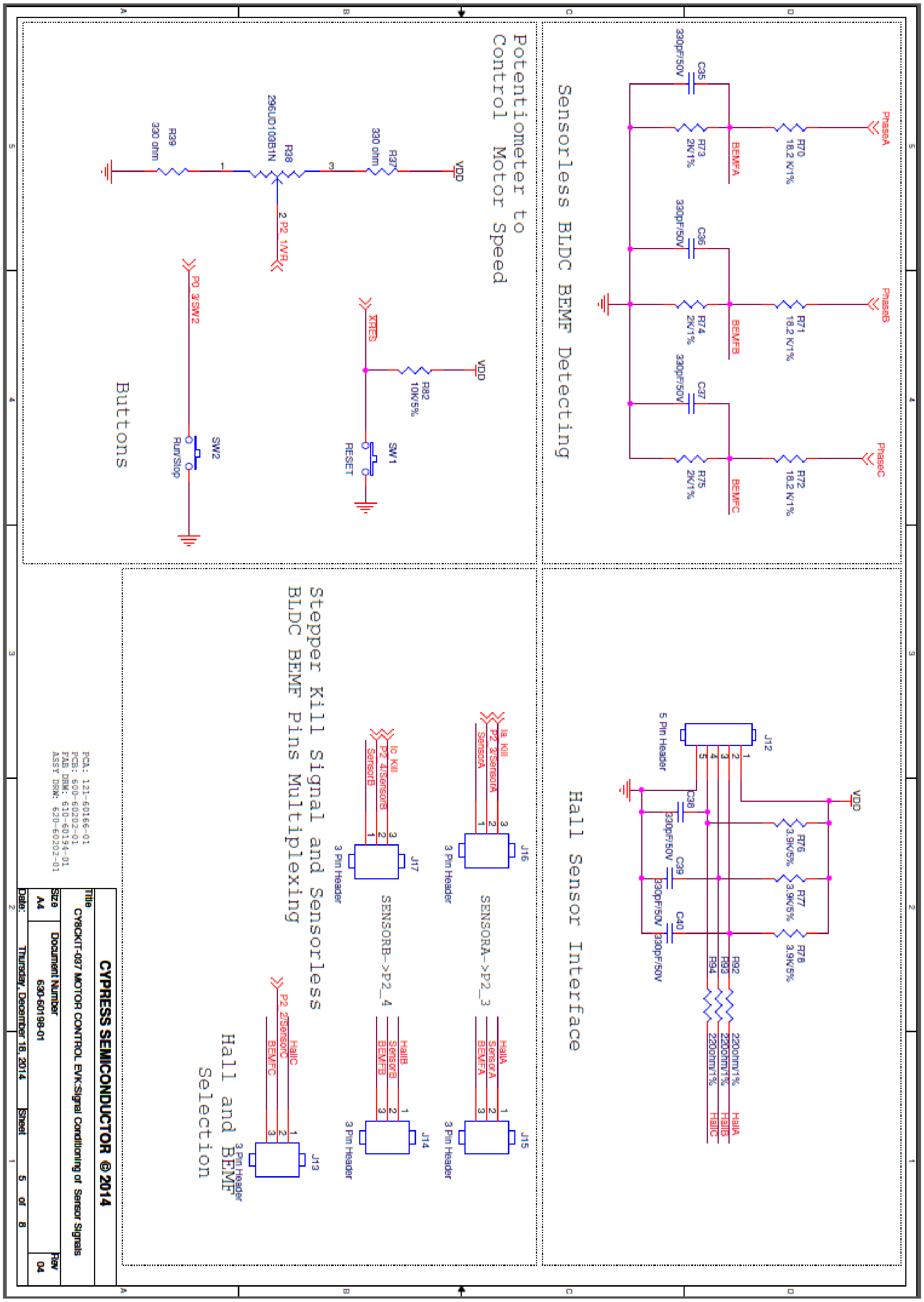
PCA: 121-60166-01
PCB: 600-60202-01
FAB DRW: 610-60194-01
ASSY DRW: 620-60202-01

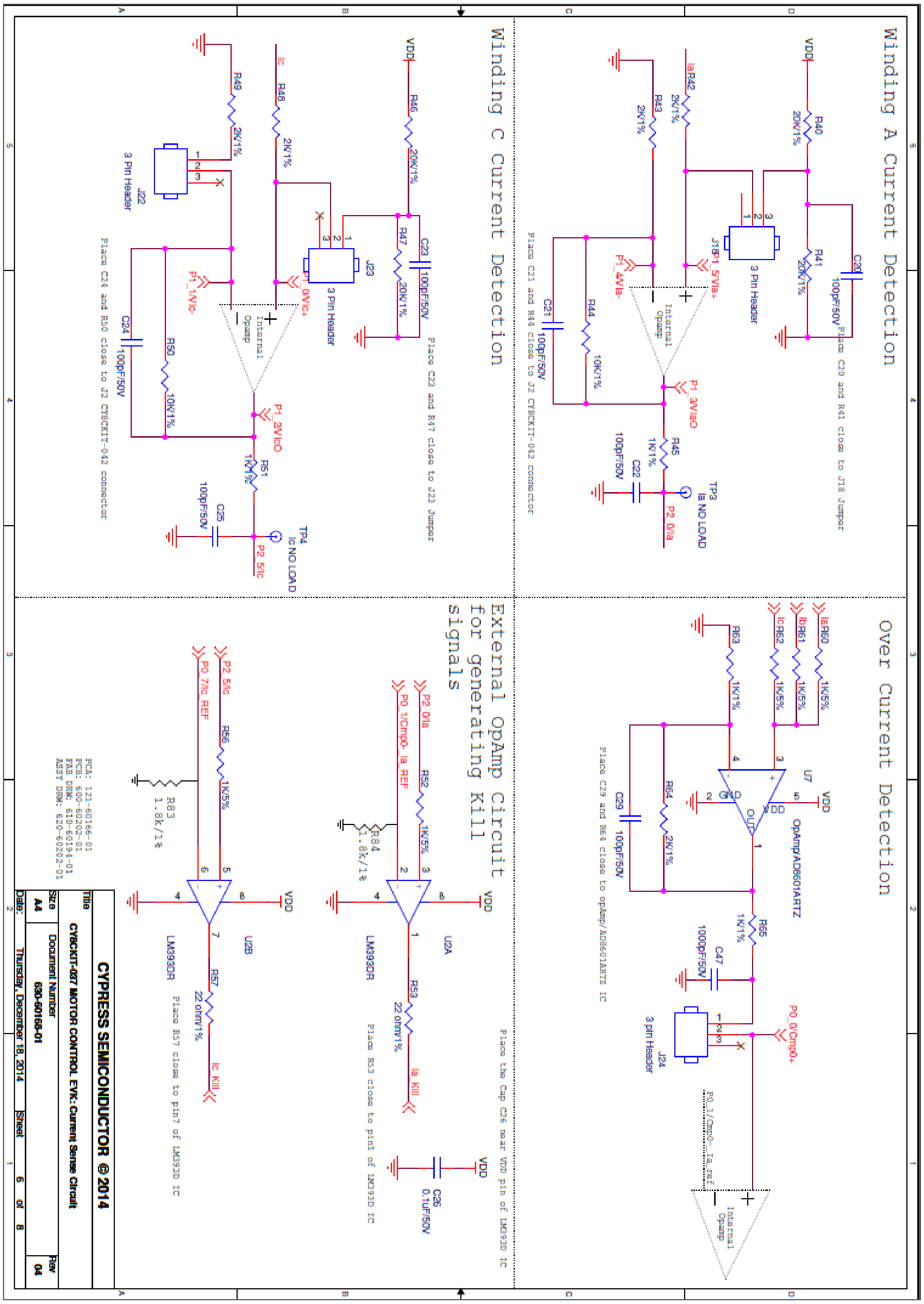
CYPRESS SEMICONDUCTOR © 2014

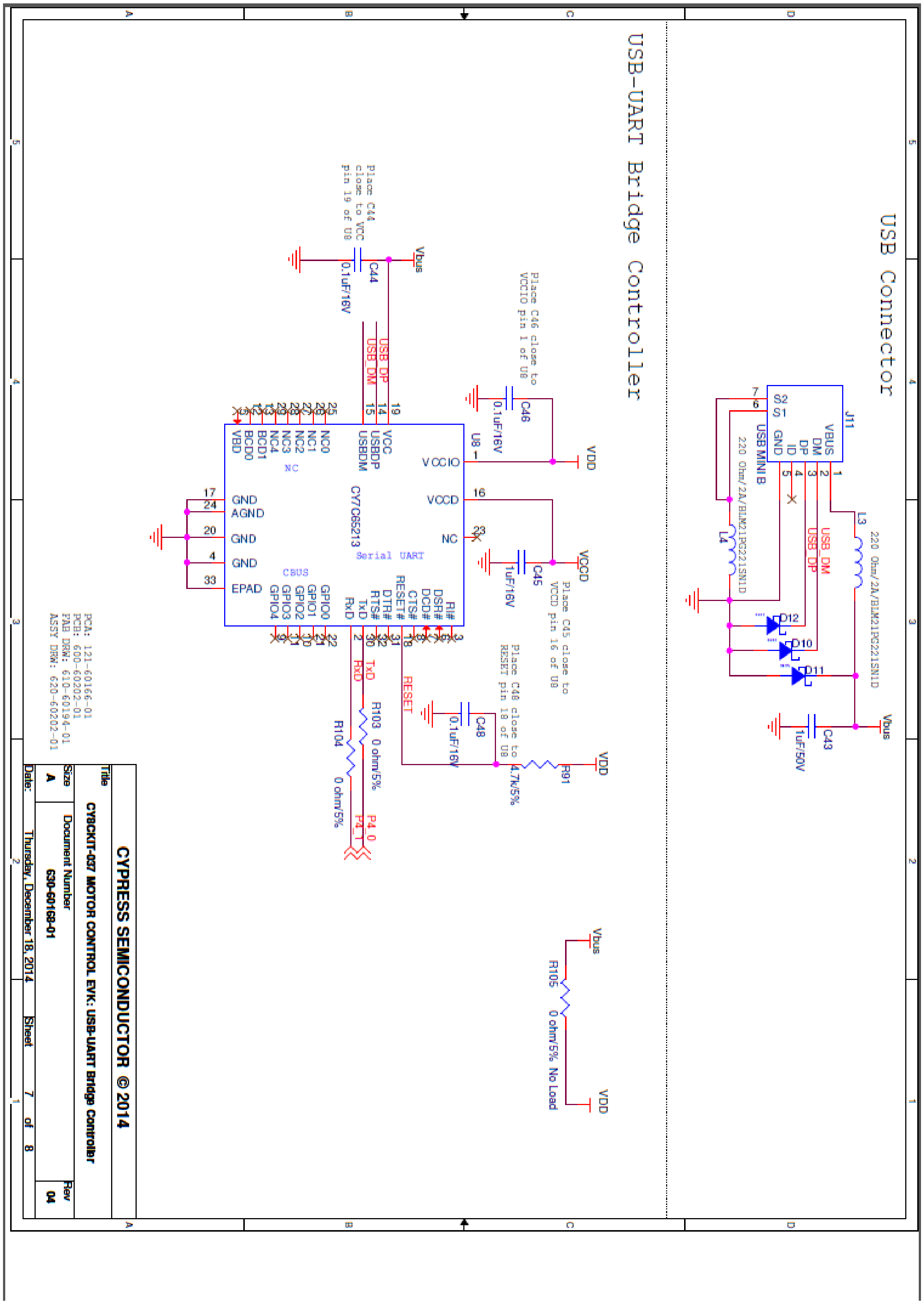
Title: CY8CKIT-037 MOTOR CONTROL EVK: Dual H-Bridge Motor Driver

Size	Document Number	Rev
A4	630-60198-01	04

Date: Thursday, December 18, 2014 Sheet 4 of 8

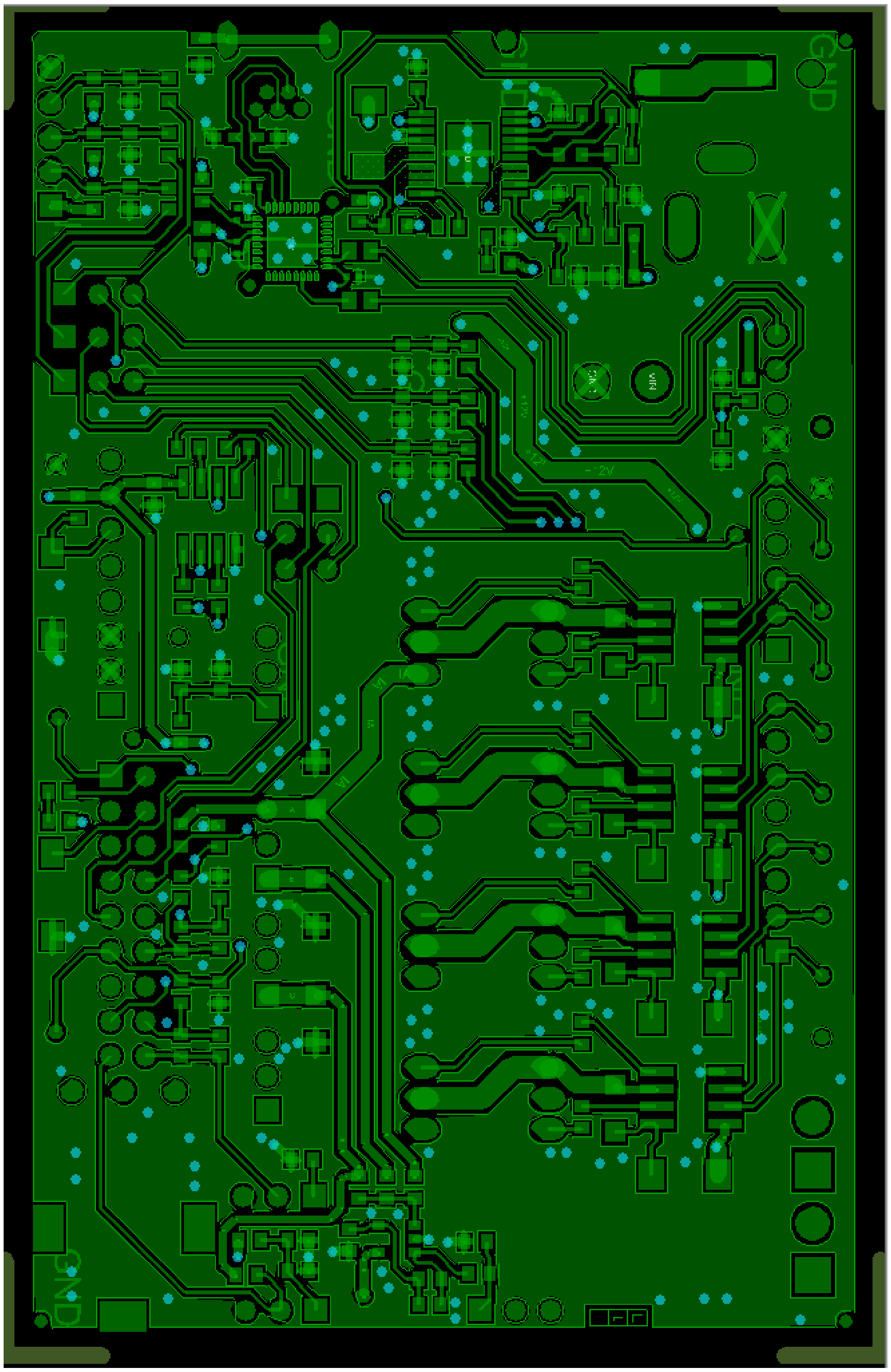

Sensorless BLDC BEMF Detecting
PhaseA
PhaseB
PhaseC
R70 18.2 K/1%
R71 18.2 K/1%
R72 18.2 K/1%
R73 2K/1%
R74 2K/1%
R75 2K/1%
BEMFA
BEMFB
BEMFC
C35 330pF/50V
C36 330pF/50V
C37 330pF/50V
Potentiometer to Control Motor Speed
VDD
R37 330 ohm
R38 296UD103B1N
R39 330 ohm
2 P2_1/VR
Buttons
R82 10K/5%
XRES
SW1 RESET
P0_3/SW2
SW2 Run/Stop
Hall Sensor Interface
J12 5 Pin Header
R76 3.9K/5%
R77 3.9K/5%
R78 3.9K/5%
C38 330pF/50V
C39 330pF/50V
C40 330pF/50V
R92 220ohm/1%
R93 220ohm/1%
R94 220ohm/1%
HallA
HallB
HallC
Stepper Kill Signal and Sensorless BLDC BEMF Pins Multiplexing
Ia_Kill
P2_3/SensorA
SensorA
J16 3 Pin Header
SENSORA->P2_3
Ic_Kill
P2_4/SensorB
SensorB
J17 3 Pin Header
SENSORB->P2_4
Hall and BEMF Selection
J15 3 Pin Header
J14 3 Pin Header
J13 3 Pin Header
P2_2/SensorC
PCA: 121-60166-01
PCB: 600-60202-01
FAB DRW: 610-60194-01
ASSY DRW: 620-60202-01
CYPRESS SEMICONDUCTOR © 2014
Title
CY8CKIT-037 MOTOR CONTROL EVK-Signal Conditioning of Sensor Signals
Size A4
Document Number 630-60198-01
Rev 04
Date: Thursday, December 18, 2014
Sheet 5 of 8


Winding A Current Detection
Winding C Current Detection
Over Current Detection
External OpAmp circuit for generating Kill signals
Place C20 and R41 close to J18 Jumper
Place C21 and R44 close to J2 CY8CKIT-042 connector
Place C23 and R47 close to J23 Jumper
Place C24 and R50 close to J2 CY8CKIT-042 connector
Place C29 and R64 close to opAmp/AD8601ARTZ IC
Place the Cap C26 near VDD pin of LM393D IC
Place R53 close to pin1 of LM393D IC
Place R57 close to pin7 of LM393D IC
Internal Opamp
3 Pin Header
OpAmp/AD8601ARTZ
LM393DR
U7
U2A
U2B
J18
J22
J23
J24
TP3 Ia NO LOAD
TP4 Ic NO LOAD
P1_5/Ia+
P1_4/Ia-
P1_3/IaO
P2_0/Ia
P1_0/Ic+
P1_1/Ic-
P1_2/IcO
P2_5/Ic
P0_0/CmpO+
P0_1/CmpO-_Ia_ref
P0_1/CmpO-_Ia_REF
P0_7/Ic_REF
Ia Kill
Ic Kill
Ia
Ib
Ic
VDD
GND
OUT
PCA: 121-60165-01
PCB: 600-60202-01
FAB DRW: 610-60134-01
ASSY DRW: 620-60202-01
CYPRESS SEMICONDUCTOR © 2014
Title
CY8CKIT-037 MOTOR CONTROL EVK: Current Sense Circuit
Size A4
Document Number 630-60168-01
Rev 04
Date: Thursday, December 18, 2014
Sheet 6 of 8

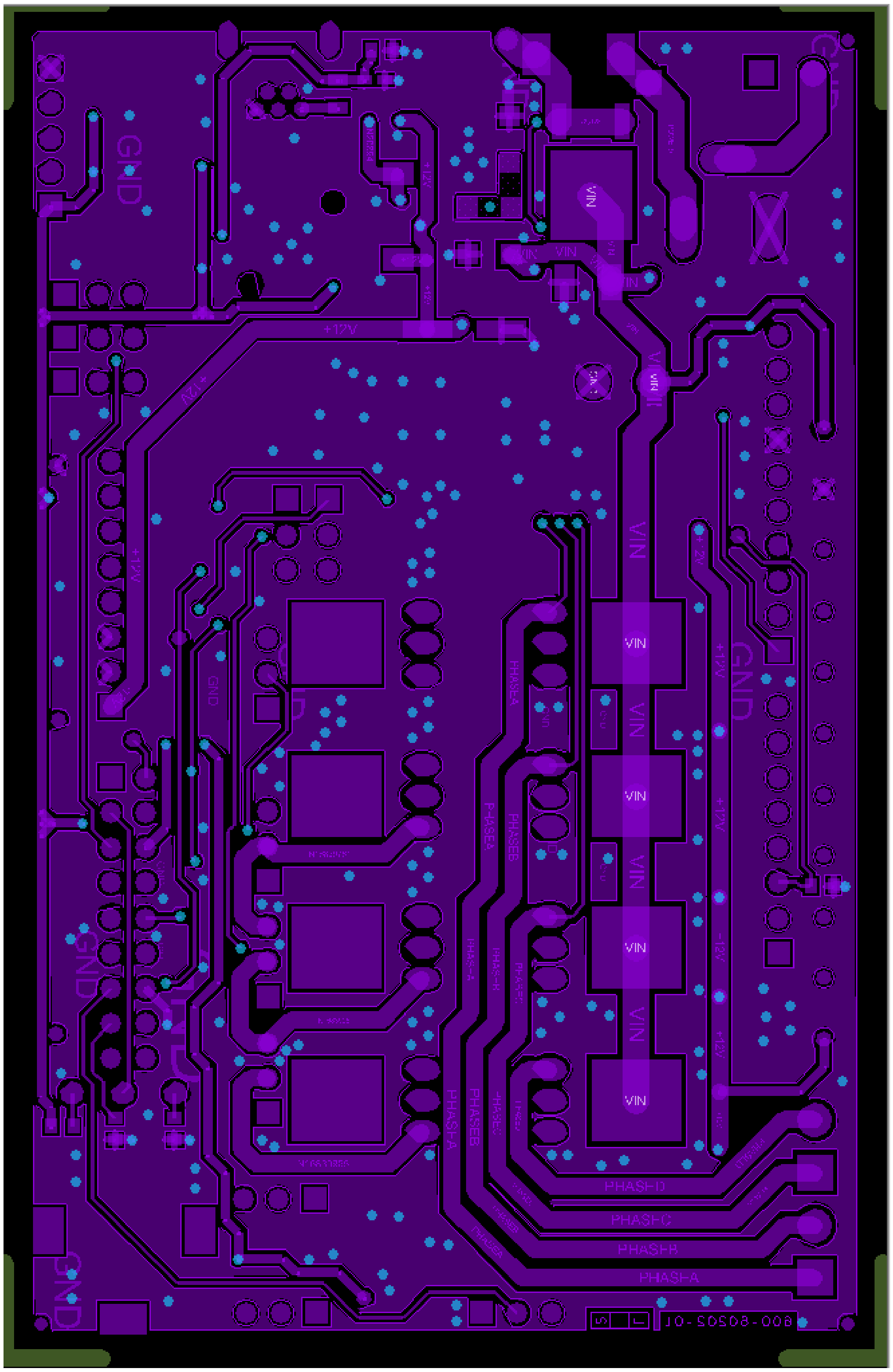

USB Connector
J11
VBUS
DM
DP
ID
GND
S1
S2
USB MINI B
L3
L4
220 Ohm/2A/BLM21PG221SN1D
USB_DM
USB_DP
D10
D11
D12
C43
1uF/50V
Vbus
USB-UART Bridge Controller
U8
CY7C65213
Serial UART
VCC
USBDP
USBDM
NC0
NC1
NC2
NC3
NC4
BCD1
BCD0
VBD
VCCIO
VCCD
NC
GND
AGND
EPAD
CBUS
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
RI#
DSR#
DCD#
CTS#
RESET#
DTR#
RTS#
TxD
RxD
C44
0.1uF/16V
Place C44 close to VCC pin 19 of U8
C46
0.1uF/16V
Place C46 close to VCCIO pin 1 of U8
C45
1uF/16V
Place C45 close to VCCD pin 16 of U8
C48
0.1uF/16V
Place C48 close to RESET pin 18 of U8
R91
4.7k/5%
RESET
VDD
VCCD
R103
0 ohm/5%
R104
0 ohm/5%
TxD
RxD
P4_0
P4_1
R105
0 ohm/5% No Load
PCA: 121-60166-01
PCB: 600-60202-01
FAB DRW: 610-60194-01
ASSY DRW: 620-60202-01
CYPRESS SEMICONDUCTOR © 2014
Title
CY8CKIT-037 MOTOR CONTROL EVK: USB-UART Bridge Controller
Size
A
Document Number
630-60169-01
Rev
04
Date:
Thursday, December 18, 2014
Sheet
7
of
8

A.2 Board Layout

Top Layer



Bottom Layer



A.3 Bill of Materials

Item	Qty	Reference	Description	Manufacturer	Mfr Part Number
1	2	C1, C2	CAP CER 1UF 100V 10% X7R 1206	TDK Corporation	C3216X7R2A105K160AA
2	1	C3	CAP CER 8200PF 50V 10% NP0 0603	TDK Corporation	C1608C0G1H822K080AA
3	2	C4, C43	CAP CER 1UF 50V 10% X7R 0603	Taiyo Yuden	UMK107AB7105KA-T
4	1	C5	CAP CER 1800PF 50V 10% NP0 0603	TDK Corporation	C1608C0G1H182K080AA
5	1	C6	CAP CER 0.022UF 50V 10% X8R 0603	TDK Corporation	C1608X8R1H223K080AA
6	1	C7	CAP CER 2700PF 50V 10% NP0 0603	TDK Corporation	C1608C0G1H272K080AA
7	2	C8, C47	CAP CER 1000PF 50V 10% X7R 0603	TDK Corporation	CGA3E2X7R1H102K080AA
8	1	C9	CAP ALUM 22UF 35V 20% SMD	Panasonic Electronic Components	EEE-FK1V220R
9	2	C10,C26	CAP CER 0.1UF 50V 10% X7R 0603	TDK Corporation	C1608X7R1H104K080AA
10	1	C11	CAP ALUM 330UF 100V 20% RADIAL	Nichicon	URS1J331MHD1TO
11	1	C12	CAP CER 0.1UF 100V 10% X7S 0603	TDK Corporation	CGA3E3X7S2A104K080AB
12	4	C16,C17,C18,C19	CAP CER 10UF 50V 10% X5R 1206	TDK Corporation	C3216X5R1H106K160AB
13	7	C20, C21, C22, C23, C24,C25,C29	CAP CER 100PF 50V 10% CH 0603	TDK Corporation	C1608CH1H101K080AA
14	6	C35, C36, C37, C38, C39, C40	CAP CER 330PF 50V 10% NP0 0603	TDK Corporation	C1608C0G1H331K080AA
15	3	C44,C46,C48	CAP CER 0.1UF 16V 10% X7R 0402	TDK Corporation	C1005X7R1C104K050BC
16	1	C45	CAP CER 1UF 16V 10% X5R 0402	TDK Corporation	C1005X5R1C105K050BC
17	1	D1	DIODE SCHOTTKY 100V 3A SMA	Micro Commercial Co	SK310A-TP
18	1	D2	DIODE SCHOTTKY 60V 7.5A DPAK	STMicroelectronics	STPS15L60CB-TR
19	4	D3, D4, D5, D6	DIODE FAST REC 50V 1A SMA	Micro Commercial	US1A-TP

Item	Qty	Reference	Description	Manufacturer	Mfr Part Number
				Co	
20	3	D10,D11,D12	DIO, SUPPRESSOR ESD 5VDC 0603 SMD	Bourns Inc	CG0603MLC-05LE
21	1	F2	FUSE BRD MNT 2.5A 125VAC/VDC SMD	Littelfuse Inc	015402 5DRT
22	2	J1,J4	2.54mm PITCH SINGLE ROW 8POS HEADER, 13mm MATING LENGTH	PROTECTRON	P9101-08-D32-1
23	1	J2	2.54mm PITCH SINGLE ROW 18POS HEADER, 13mm MATING LENGTH	PROTECTRON	P9103-18-D32-1
24	1	J3	2.54mm PITCH SINGLE ROW 10POS HEADER, 13mm MATING LENGTH	PROTECTRON	P9101-10-D32-1
25	1	J7	CONN, JACK, POWER,2.1mm , MALE, 3PIN, PCB RA, TH	CUI Inc	PJ-102A
26	1	J8	BLACK COLOUR, TERM BLOCK 2POS SIDE ENT 3.81MM	TE Connectivity	1776113-2
27	2	J9,J10	GREEN COLOUR, TERM BLOCK 2POS 3.81MM PCB HORIZ	FCI	20020327-D021B01LF
28	1	J11	CONN, MINI USB, RCPT R/A, DIP, B TYPE	TE Connectivity	1734510-1
29	1	J12	ConWTB Wire to Board Connector 2.50mm pitch , “J” Type Vertical Male Shrouded Header, 5 pin	Wurth Elektronik	688 005 116 22
30	11	J13, J14, J15, J16, J17, J18, 20, J21, J22, J23, J24	CONN, HEADER, VERT, SGL, 3POS, GOLD	PROTECTRON	P9101-03-12-1
31	1	J19	CONN, HEADER, FEMALE, 3POS, .1", TIN, TH	Sullins Connector Solutions	PPTC031LFBN-RC
32	1	LED1	LED RED CLEAR 0603 SMD	Lite-On Inc	LTST-C190CKT
33	1	LED2	LED GREEN CLEAR THIN 0603 SMD	Lite-On Inc	LTST-C190GKT
34	1	L1	FERRITE CHIP 600 OHM 1500MA 1206	Murata Electronics North America	BLM31PG601SN1L
35	1	L2	INDUCTOR POWER 150UH 20% SMD	TDK Corporation	CLF6045T-151M-H
36	2	L3,L4	IND, FERRITE, 220 OHM, 2A, 25%, 0805	Murata Electronics North America	BLM21PG221SN1D
37	8	Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8	MOSFET N-CH 75V 56A I-PAK	International Rectifier	IRFU3607PbF
38	3	R1,R44,R50	RES 10.0K OHM 1/16W 1% 0603	TE Connectivity	5-1879337-9
39	5	R2,R64,R73,R74,R75	RES 2.00K OHM 1/10W 1% 0603 SMD	Vishay Dale	CRCW06032K00FKEA
40	1	R3	RES 143K OHM 1/10W 1% 0603 SMD	Yageo	RC0603FR-07143KL

Item	Qty	Reference	Description	Manufacturer	Mfr Part Number
41	1	R4	RES 18.2K OHM 1/10W 1% 0603 SMD	Yageo	RC0603FR-0718K2L
42	3	R70,R71,R72	RES 8.20K OHM 1/10W 1% 0603 SMD	Vishay Dale	CRCW06038K20FKEA
43	1	R5	RES 16.9K OHM 1/10W 1% 0603 SMD	Vishay Dale	CRCW060316K9FKEA
44	1	R6	RES 9.10K OHM 1/10W 1% 0603 SMD	Vishay Dale	CRCW06039K10FKEA
45	5	R7,R45,R51,R63,R65	RES 1.00K OHM 1/8W 1% SMD 0603	Vishay Beyschlag	MCT06030C1001FP500
46	1	R10	RES 590 OHM 1/10W 1% 0603 SMD	Vishay Dale	CRCW0603590RFKEA
47	1	R9	RES, 5.49K OHM, 1%, 1/10W, 0603, SMD	Panasonic-ECG	ERJ-3EKF5491V
48	1	R11	RES 2.4K OHM 1/10W 5% 0603 SMD	Yageo	RC0603JR-072K4L
49	3	R12, R37, R39	RES, 330 OHM, 5%, 200PPM, 1/10W, 0603	Panasonic-ECG	ERJ-3GEYJ331V
50	10	R20,R21,R22,R23, R24,R25,R26,R27, R53,R57	RES, 22 OHM, 1%, 200PPM, 1/10W, 0603, SMD	Panasonic-ECG	ERJ-3EKF22R0V
51	3	R34, R35, R36	RES 0.03 OHM 1/2W 1% 1206 SMD	Rohm Semiconductor	UCR18EVHFSR030
52	1	R38	POT 10K OHM 1/8W CARB VERTICAL	CTS Corporation	296UE103B1C
53	4	R40, R41, R46, R47	RES, 20K OHM, 1%, 100PPM, 1/10W, 0603	Panasonic Electronic Components	ERJ-3EKF2002V
54	4	R42,R43,R48,R49	RES 2.40K OHM 1/10W 1% 0603 SMD	Panasonic Electronic Components	ERJ-3EKF2401V
55	5	R52,R56,R60,R61, R62	RES 1.0K OHM 1/10W 5% 0603 SMD	Vishay Dale	CRCW06031K00JNTA
56	3	R76, R77, R78	RES 3.9K OHM 1/10W 5% 0603 SMD	Yageo	RC0603JR-073K9L
57	1	R82	RES 10.0K OHM 1/10W 5% 0603	Yageo	RC0603JR-0710KL
58	2	R83,R84	RES 1.80K OHM 1/10W 1% 0603 SMD	Vishay Dale	CRCW06031K80FKEA
59	1	R91	RES 4.7K OHM 1/10W 5% 0603 SMD	Panasonic Electronic Components	ERJ-3GEYJ472V
60	3	R92,R93,R94	RES 220 OHM 1/10W 1% 0603 SMD	Panasonic Electronic Components	ERJ-3EKF2200V
61	2	R103, R104	RES, 0.0 OHM, 1/10W, 5%, 0603, SMD	Panasonic Electronic Components	ERJ-3GEY0R00V
62	2	SW1,SW2	SWITCH TACTILE SPST-NO 0.05A 12V	Panasonic	EVQ-PE105K

Ite m	Qty	Reference	Description	Manufacturer	Mfr Part Number
				Electronic Components	
63	2	TP1(GND),TP2(GN D) ,	TP, TEST POINT, BLACK, TH	Keystone Electronics	5006
64	3	TP13(+12V), TP14(Vin),TP15(V DD)	TP, TEST POINT, RED, 0.063"D, TH	Keystone Electronics	5005
65	1	U1	IC REG BUCK ADJ 2.5A 20TSSOP	TI	LM5005MHX/NOPB
66	1	U2	IC, DUAL, DIFF, COMP, 8PIN, SOIC	Rohm Semiconductor	LM393DR
67	4	U3,U4,U5,U6	IC DRIVER HIGH/LOW SIDE 8SOIC	International Rectifier	IR2101STRPBF
68	1	U7	IC OPAMP GP 8.4MHZ RRO SOT23-5	Analog Devices	AD8601ARTZ-R2
69	1	U8	IC, CYPRESS, USB-UART LP, 512-BYTE FLASH, QFN-32	Cypress Semiconductor	CY7C65213-32LTXI
70	1	VR1	VARISTOR 67V 250A 1210	Littelfuse Inc	V60MLA1210H
71	1	PCB	CY8CKIT-037 MOTOR CONTROL SHIELD PCB (Size: 3.8 inch x 2.44 inch No of Layers: 2 Surface Finish: ENIG Solder Mask color: Red Board thickness: 1.6 mm)		

NoLoad Components

Item	Qty	Reference	Description	Manufacturer	Mfr Part Number
1	1	R105	RES, 0.0 OHM, 1/10W, 5%, 0603, SMD	Panasonic Electronic Components	ERJ-3GEY0R00V
2	15	TP3(Ia),TP4(Ic),P5(AH), TP6(AL),TP7(BH),TP8(BL),TP9(CH),TP10(CL),TP11(DH),TP12(DL), TP16(P0_2),TP17(P0_1),TP18(P0_7),TP19(GND), TP20(P2_6/VIN_SENSE)	TP, TEST POINT, RED, 0.063"D, TH	Keystone Electronics	5005
3	1	F1	PTC RESETTABLE 60V 2.50A KINKAMO	Littelfuse Inc	60R250XMR

Special Installation Instructions

Item	Qty	Reference	Description	Manufacturer	Mfr Part Number
1	11	J13,J14,J15,J16, J17,J18,J20,J21, J22,J23,J24	MINI JUMPER GF 13.5 CLOSE TYPE BLACK WITH HANDLE	Kobiconn	151-8030-E
2	1	SHUNT_J19	JUMPER SHUNT NON-INSULATED MALE .100"	Mill-Max Manufacturing Corp.	999-11-110-10-000000
3	1	J12	ConWTB Wire To board Connector 2.50mm pitch, "J" type Female Terminal Housing, 5 Pins	Wurth Elektronik	688 005 113 322
4	5	J12 Hall sensor wire Crimping	2.50mm “J” Type Female Crimp Contact WR-WTB	Wurth Elektronik	688 001 137 22

Item	Qty	Reference	Description	Manufacturer
1	1	N/A	LBL, Kit Product Identification Label, Vendor Code, Datecode, Serial Number CY8CKIT-037 Motor Control EVK Rev**(YYWWVVXXXXX)	Cypress Semiconductor
2	1	N/A	LBL, QR Code, CY8CKIT-037 Printed Circuit Assembly, 13mm X 13mm	Cypress Semiconductor

Revision History

Document History

Document Title: CY8CKIT-037 PSoC® 4 Motor Control Evaluation Kit Guide

Document Number: 001-92562

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	4784695	ROWA	06/02/2015	New document