

PSoC® 4 Sensorless Field-Oriented Control (FOC)

About this document

Scope and purpose

AN93637 shows how to implement sensorless field-oriented control (FOC) for a permanent magnet synchronous motor (PMSM) with a PSoC 4500S device. A code example using the CY8CKIT-037 Motor Control Evaluation Kit is included to demonstrate sensorless FOC.

Requirements

Tool: [PSoC Creator](#) 4.4

Associated Parts: PSoC 4500S

More code examples? We heard you.

To access an ever-growing list of hundreds of PSoC code examples, please visit our [code examples web page](#). You can also explore the video training library [here](#).

Table of contents

About this document.....	1
Requirements	1
Table of contents.....	1
1 Introduction	3
2 Motor Control Terms and Definitions	4
3 Sensorless FOC Basics.....	5
4 Code Example	9
4.1 Features	9
4.2 Design Overview	9
4.3 Firmware.....	11
4.4 CY8CKIT-037 Kit	13
4.5 Operation.....	14
4.5.1 Step 1 – Configure CY8CKIT-045S	14
4.5.2 Step 2 – Configure CY8CKIT-037	15
4.5.3 Step 3 – Plug CY8CKIT-037 into CY8CKIT-045S.....	15
4.5.4 Step 4 – Connect the Power Supply and Motor	15
4.5.5 Step 5 – Build the Project and Program the PSoC 4 Device.....	16
4.5.6 Step 6 – Press the SW2 Button to Start Motor Rotation.....	16
4.6 Performance	17
5 Design Details	19
5.1 Current Sampling	19
5.1.1 PSoC Creator Schematic	19
5.1.2 Current Sampling Firmware	23

Table of contents

5.2	Transformations.....	24
5.3	Slide Mode Observer (SMO)	25
5.4	PI Controllers.....	27
5.5	SVPWM Generation.....	29
5.5.1	PSoC Creator Schematic Design.....	29
5.5.2	Firmware Implementation.....	32
6	Summary	34
7	Resources.....	35
8	Appendix A: PMSM Model	36
9	Appendix B: Slide Mode Observer (SMO)	40
10	Appendix C: SVPWM Theory	42
11	Appendix D: Q Number Format (Fixed-Point Number)	45
11.1	Characteristics.....	45
11.2	Conversion.....	46
11.2.1	Float to Q	46
11.2.2	Q to float	46
11.3	Math Operations.....	46
11.3.1	Addition	47
11.3.2	Subtraction.....	47
11.3.3	Multiplication	47
11.3.4	Division	48
12	Appendix E: Adapting the Design to Other Motors	49
12.1	Operation Guide	49
12.1.1	Check the power range and motor type.....	49
12.1.1.1	Power range	49
12.1.1.2	Motor type	49
12.1.2	Change the parameters in the example project.	49
12.1.2.1	Change the parameters for the motor.	49
12.1.2.2	Change the macro definitions for the system parameters.....	50
12.1.2.3	Change the parameters for the PI controllers.	50
12.1.3	Set up the hardware.....	50
12.1.4	Program the CY8CKIT-045S and observe the performance.....	51
12.1.5	Tune the parameters if the motor does not rotate.....	51
13	Appendix F: Tunable Parameters List	52
13.1	Hardware Parameter Setting.....	52
13.2	Firmware Parameter Setting.....	53
13.3	Motor Parameters.....	54
13.4	ADC Sampling Parameters.....	54
13.5	PI Regulator Parameters.....	55
13.6	Start-up Parameters.....	55
13.7	Close Loop Running Parameters	56
13.8	Protection Parameters.....	57
13.9	Other global Parameters.....	57
	Revision history.....	60

Introduction

1 Introduction

This application note shows how to control a permanent magnet synchronous motor (PMSM) with the sensorless field-oriented control (FOC) algorithm, using an ARM® Cortex®-M0+-based PSoC 4 device.

The FOC algorithm is frequently used in motor control applications because it allows motors to operate with less noise and more stable torque output than other algorithms. Sensorless FOC adds the advantage of reducing the cost due to the absence of rotor position sensors. Sensorless FOC is used in many applications including consumer (air conditioner, refrigerator), industrial (blower, pump), and commercial (elevator, escalator) products.

Sensorless FOC is calculation-intensive, and thus has been traditionally implemented with expensive digital signal processing (DSP) devices. However, with 32-bit ARM Cortex-M cores, it is possible to implement sensorless FOC with more cost-effective 32-bit MCUs.

The Sensorless FOC motor control algorithm and codes in this AN also can be applied to other PSoC 4 device, but the Pins assignment should change according to the schematics.

This application note assumes that you are familiar with PSoC 4 and the PSoC Creator IDE. If you are new to PSoC 4, see [AN79953 – Getting Started with PSoC 4](#) and the [PSoC 4500S datasheet](#). If you are new to PSoC Creator, see the [PSoC Creator home page](#).

You should also understand motor control fundamentals; start with “[electric motor](#)” on Wikipedia.

Finally, this application note includes a code example to be used with the Cypress [CY8CKIT-037 Motor Control Evaluation Kit](#). This kit includes a 24-V 53-W PMSM motor.

Note: The CY8CKIT-037 kit board can operate at voltages as high as 48 V DC, and some components may operate at high temperatures. Use this kit with caution to avoid personal injury or equipment damage.

Motor Control Terms and Definitions

2 Motor Control Terms and Definitions

BEMF: Back electromotive force. Commonly refers to the voltage that occurs in electric motors where there is relative motion between the armature of the motor and the magnetic field from the motor's field magnets or windings. See [Appendix B](#).

BLDC: Brushless DC motor. It has a trapezoidal BEMF; see [Appendix E](#) and [Figure 42](#).

PMSM: Permanent-magnet synchronous motor, or permanent-magnet motor (PMM). A synchronous motor that uses permanent magnets rather than windings in the rotor. It has a sinusoidal BEMF; see [Appendix E](#) and [Figure 42](#).

Sensorless FOC Basics

3 Sensorless FOC Basics

This section introduces the hardware structure of a typical sensorless FOC system as well as a firmware overview of the FOC algorithm. If you are familiar with these concepts, you can skip this section and go to the [Code Example](#) section.

Figure 1 shows the diagrams of the two types of PMSM motor; they differ in how magnets are placed in the rotor:

- Surface PMSM (SPMSM) – Left
- Interior PMSM (IPMSM) – Right

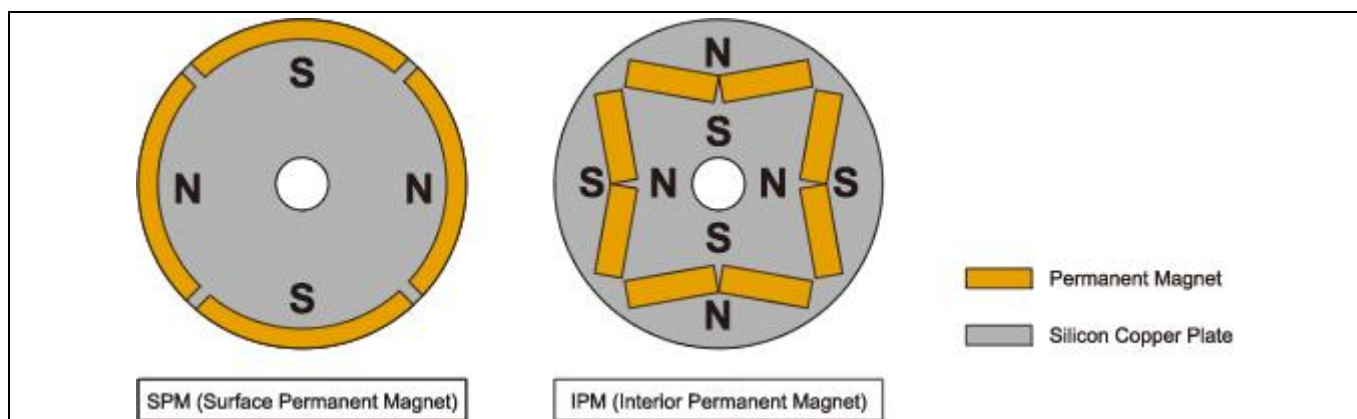


Figure 1 Rotor Structure for SPMSM and IPMSM (Source: <http://www.hamaco.jp>)

SPMSM is widely used due to the ease of manufacture and assembly, while IPMSM has a larger torque output with the same sized motor. The sensorless FOC algorithm varies depending on the motor type; this application note uses SPMSM, referred to as just “PMSM”.

Figure 2 shows the hardware block diagram of a typical sensorless FOC system. It consists of:

- MCU
- Inverter
- PMSM
- Current sampling and signal conditioning circuit to determine the rotor position
- Communication interface

These components can be on the same controller board or separated in the system such as on an MCU board and an inverter board.

The inverter is composed of gate drivers and six MOSFETs (two for each motor phase). Turning different MOSFETs ON or OFF changes the current direction through the motor’s stator windings or phases.

For example, turning ON Q1 and Q4 generates a current from phase A to phase B, while turning ON Q3 and Q2 reverses the current direction in those phases. Changing the current direction changes the stator flux direction and makes the rotor rotate.

Sensorless FOC Basics

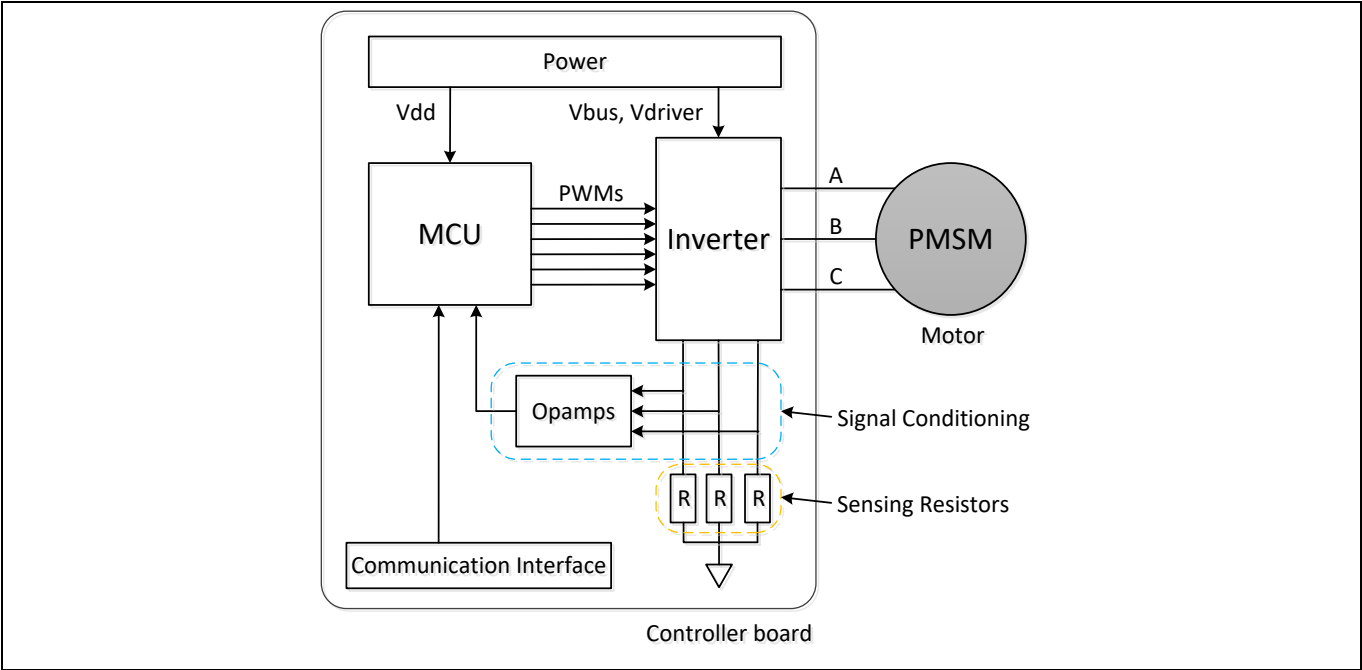


Figure 2 Overview of a Typical Sensorless FOC System

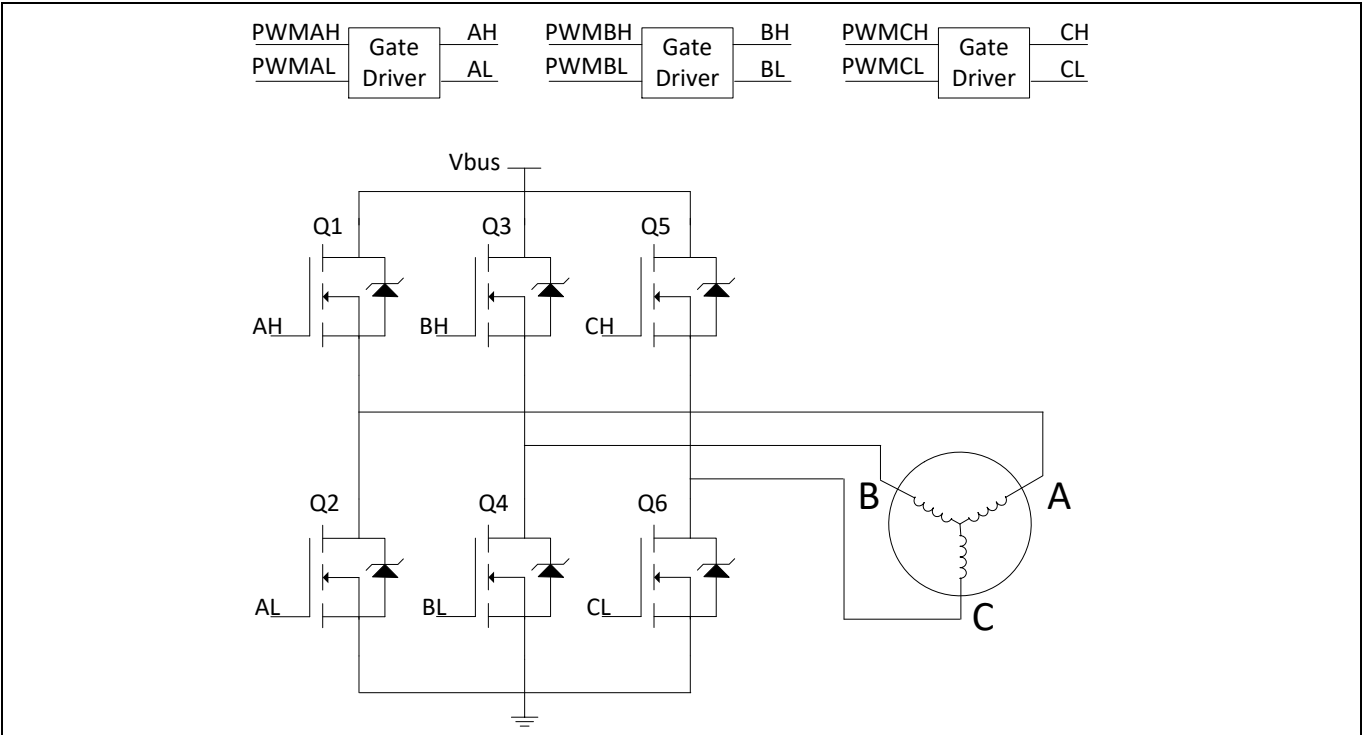


Figure 3 Details of Inverter Block

Sensorless FOC Basics

Vbus is a higher-voltage DC supply to power the motor. For example, it is 24 V in the **CY8CKIT-037** kit.

Note that the pairs of MOSFETs on the same phase (for example, Q1 and Q2) must not be turned ON at the same time – the resultant low resistance causes high currents that can damage the MOSFETs.

Figure 4 and **Figure 5** show diagrams of the sensorless FOC algorithm and its calculation flow. The algorithm controls either the motor speed or motor torque using a proportional-integral (PI) controller based on a mathematical model of the PMSM (**Appendix A** describes the PMSM model). The control result is sent to a Space Vector Pulse Width Modulation (SVPWM) block (**Appendix C**). The SVPWM block generates three-phase voltages that change the stator currents.

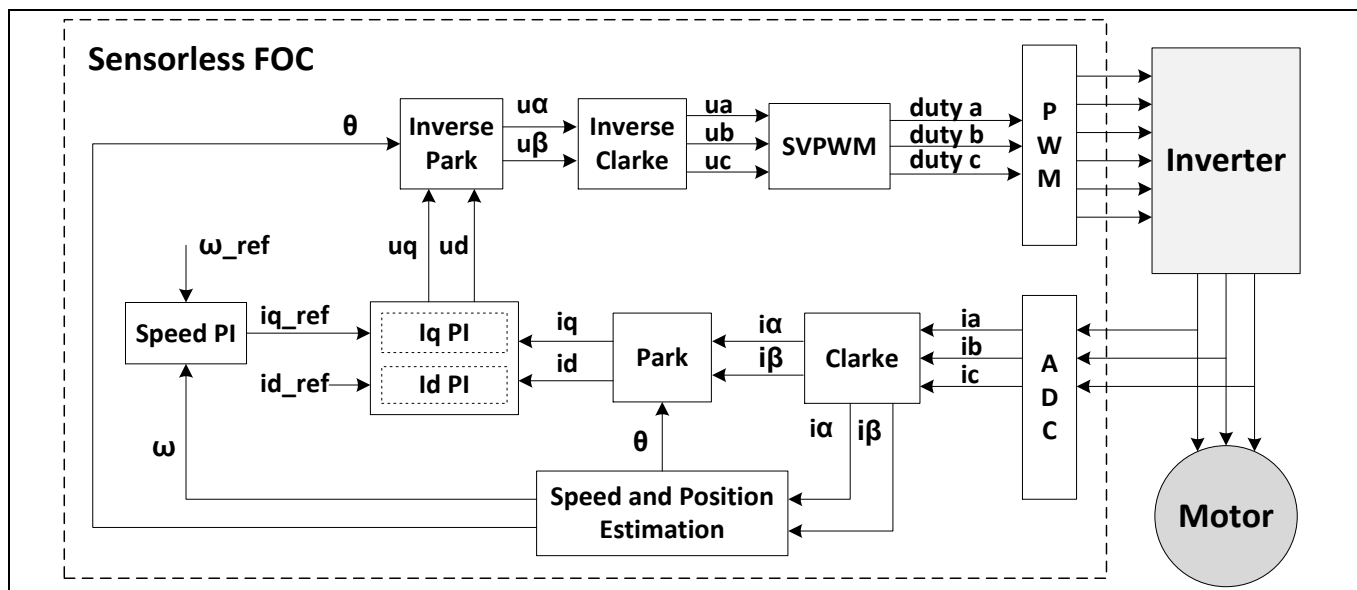


Figure 4 Sensorless FOC Control Block Diagram

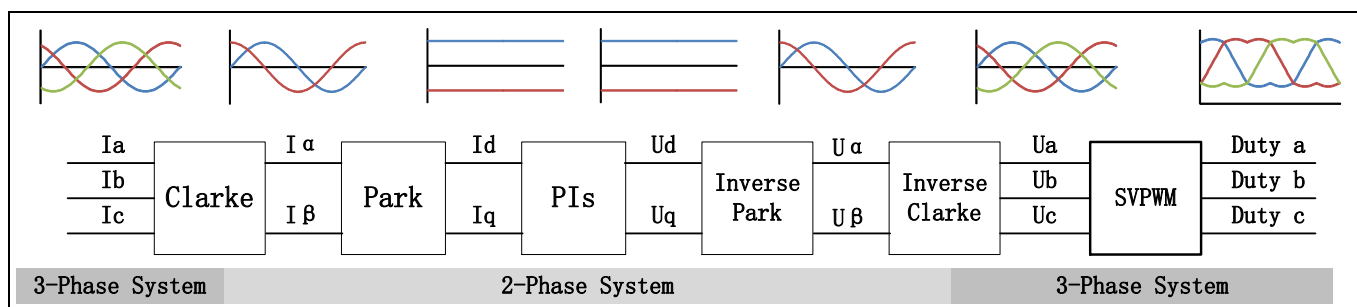


Figure 5 Sensorless FOC Calculation Flow

The Clarke and Park transformation calculations convert these three sampled motor phase currents into two values that are used by the PI controller. The Inverse Clarke and Inverse Park transformations are the opposites of the Clarke and Park transformations, respectively.

Figure 6 shows the Clarke transformation, where the three motor phase currents (i_a , i_b , i_c) are converted to i_α and i_β . The (**a, b, c**) frame is a three-phase stator reference frame, where the axes are 120° apart from each other. The transformation method is to project (i_a , i_b , i_c) onto the (**α , β**) axes, which produce the outputs i_α and i_β .

Sensorless FOC Basics

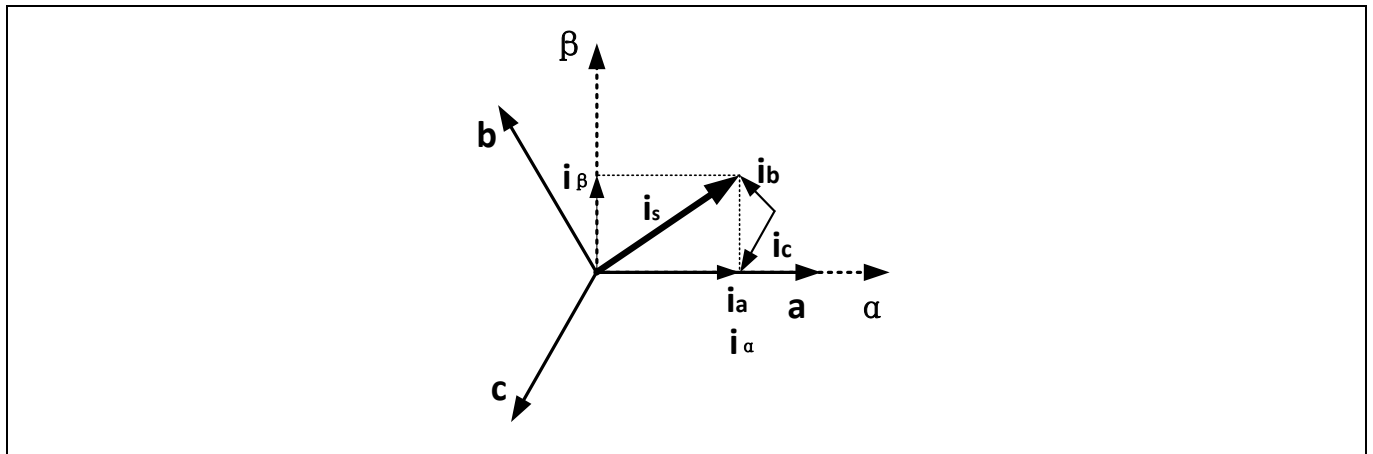


Figure 6 Clarke Transformation

Figure 7 shows the details of the Park transformation. This transformation converts the current vectors from the Clarke transformation, i_α and i_β , to a frame on the rotating part of the motor. The axes of the rotating frame are called (**d**, **q**). The current vectors on these axes are called i_d and i_q .

Ψ_f is the flux linkage vector of the rotor magnet. The **d** axis is always aligned with Ψ_f , and the **q** axis is at 90° to the **d** axis. The rotor rotates at an angular speed ω_r , and θ_r is the angle between the α and **d** axes.

In the (**d**, **q**) frame, the motor torque is proportional to i_q . We can control i_q to achieve the desired torque by using the **proportional-integral (PI) controller**. For details on the Clarke and Park transformations as well as the torque output, see **Appendix A: PMSM Model**.

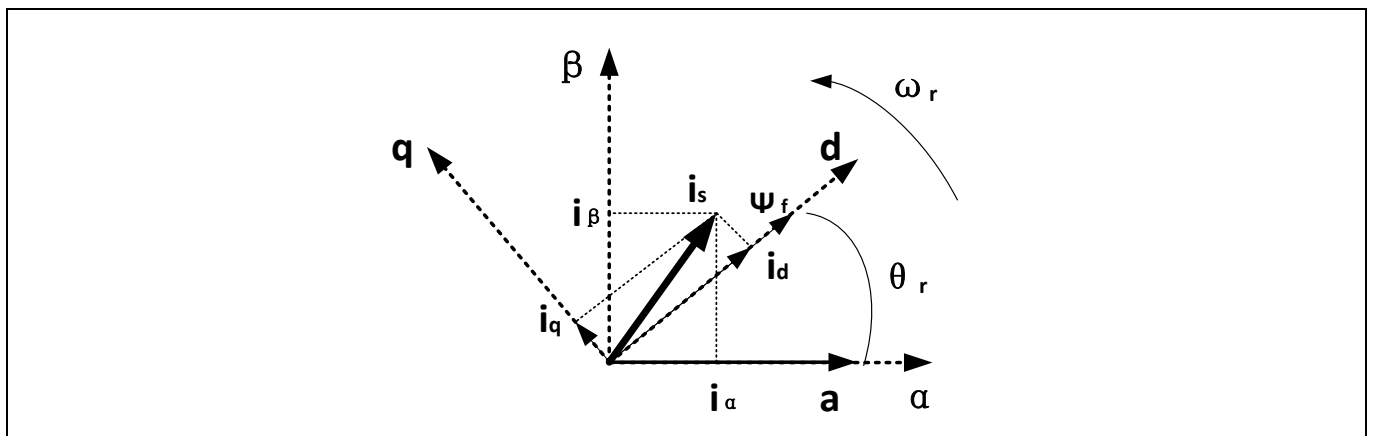


Figure 7 Park Transformation

The angle θ_r used in the Park transformation is derived from the speed and position estimation. An algorithm called “Slide Mode Observer” (SMO) uses i_α and i_β to derive the θ_r value. Then, the angular speed ω_r is calculated based on θ_r . For more information, see **Appendix B: Slide Mode Observer (SMO)**.

Code Example

4 Code Example

This application note includes an example project to demonstrate sensorless FOC algorithm implemented with CY8C42xx devices. The code example is a complete design with all the code already written. It is intended to be used directly with the Cypress's motor kit, [CY8CKIT-037](#).

This kit can be used with other motors if the motor's power level is in the range supported by the [CY8CKIT-037](#) kit. See [Appendix E: Adapting the Design to Other Motors](#). [Appendix F: Tunable Parameters List](#) provides a quick reference for tuning the parameters for other motors.

If the motor's power level is greater than the range supported by this kit, you must design your custom control board with higher-power MOSFETS, power supply, and other components. The code example can still be used.

4.1 Features

- Implements the sensorless FOC algorithm and closed-loop speed control in a multilayer, extensible, binary library architecture
- Estimates the rotor position with the Slide Mode Observer (SMO) algorithm
- Uses the PSoC 4 internal opamps and the 888Ksps successive approximation register (SAR) ADC for signal conditioning and measuring the motor phase current
- Employs open-loop control at startup, which is changed to closed-loop control after the rotor position is determined
- Supports motor speeds from 500 to 4000 RPM by default. Can support higher speeds in other motors by modifying the tuning parameters in the code example
- Can adjust the motor speed by using the potentiometer on the kit
- Provides control accuracy 5% over the default speed range. Using high-resolution sensing resistors and advanced control algorithms can improve the accuracy; this topic is outside the scope of this application note.

4.2 Design Overview

[Figure 8](#) illustrates sensorless FOC implementation in PSoC 4. A 12-bit SAR ADC and two opamps are used to sample the motor phase currents (only two phase currents need be sampled; the third phase can be calculated from the other two.) The three TCPWMs generate six PWM outputs applied to the Inverter. A serial communication block (SCB) implements a UART to communicate with the host. [Figure 9](#) is the corresponding implementation in PSoC Creator. For details, see [Design Details](#).

Code Example

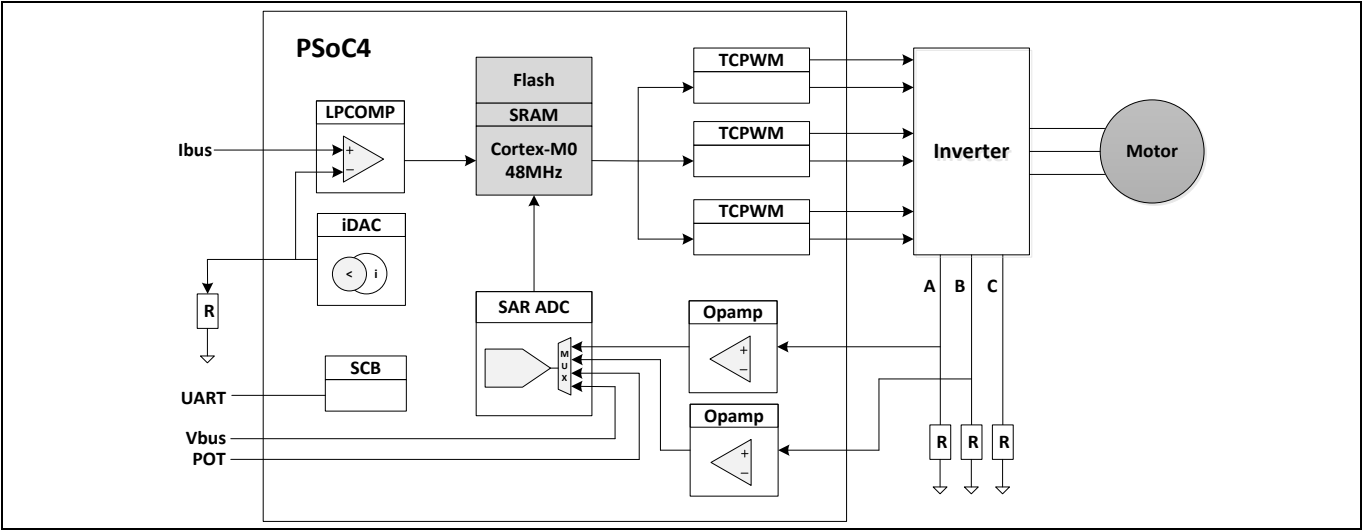


Figure 8 PSoC4 Sensorless FOC Implementation Block Diagram

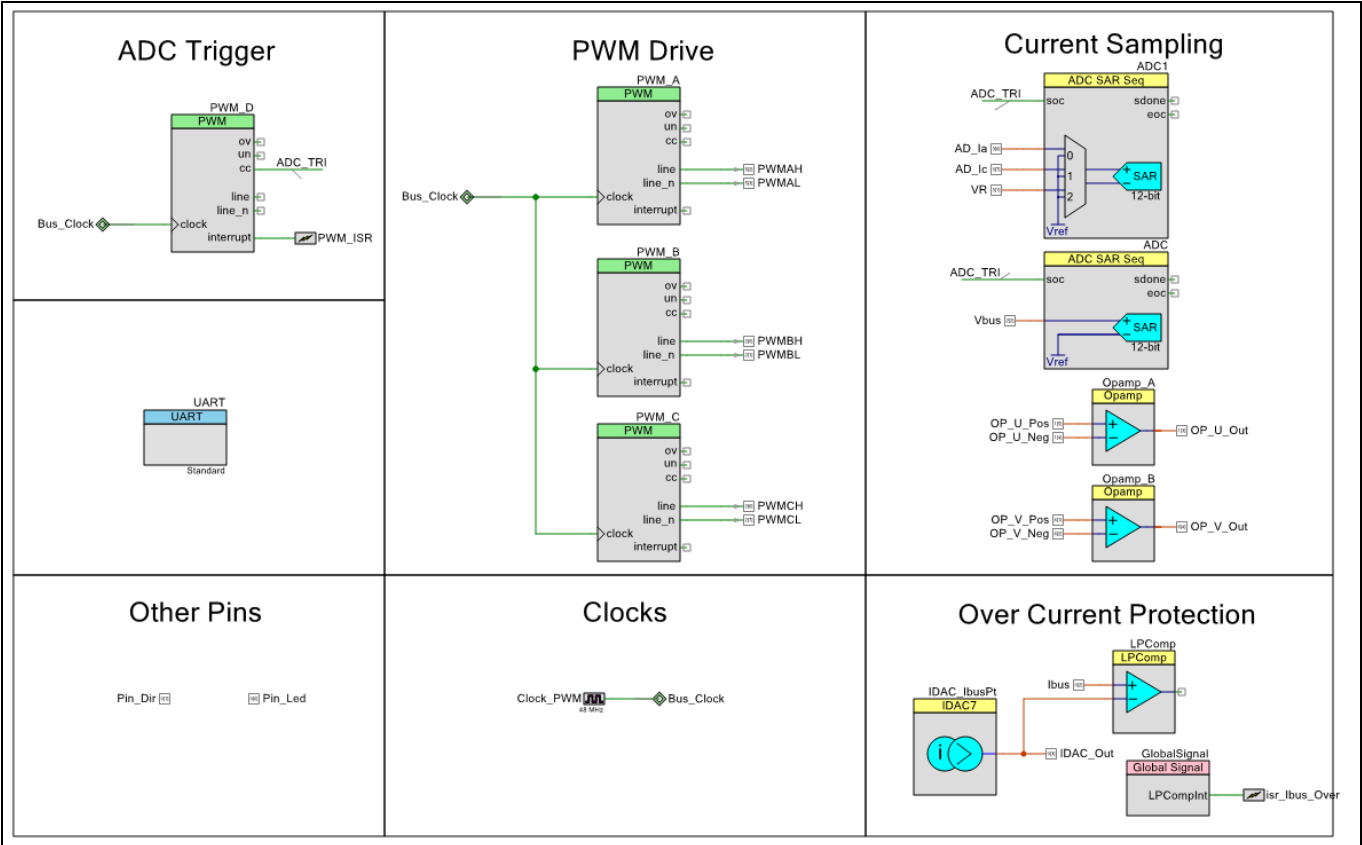


Figure 9 PSoC Creator Schematic Overview

Code Example

Table 1 shows the PSoC 4 resources that are used by this code example:

Table 1 Resource Usage Summary

Item	Used	Available	Usages
CPU Frequency	48MHz	48MHz	Internal system clock
PWM Frequency	10KHz	5KHz~20KHz	NA
Flash	23216 bytes	256KB	NA
SRAM	3628 bytes	32KB	NA
Interrupts	3	32	Generate interrupts that system needed
TCPWM Blocks	4	8	Three TCPWM are used to generate 3 phase signals to control PMSM drive. Another TCPWM used to generate trigger pulse for ADC to current measurement.
OPAMP	2	2	Used to amplify voltage from current sense resistors prior to feeding the voltages to ADC inputs.
UART	1	5	Reserved for communication with the host
Low-Power Comparators	1	2	Used for over current protection
8-bit current DAC((IDAC)	1	1	Generate the source current for over current protection
12-bit SAR ADC	2	2	Used to transfer the phase current sampled value to digital signals
Other Pins	2	27	Pin_Led: the CPIO to control LED; Pin_Dir: the GPIO to control the motor running direction.

4.3 Firmware

Figure 10 shows the firmware execution flow. The FOC algorithm requires the PWM duty cycle to be updated every control cycle. Therefore, FOC calculations must be done in a periodic interrupt service routine (ISR). The ISR is triggered by the PWM every 100 μ s (10-kHz PWM) – this is the control cycle period. Note that this interrupt should be high priority for effective duty cycle control – see [SVPWM Generation](#).

The cycle period can be decreased by increasing the PWM frequency. A shorter control period results in a higher-bandwidth control system with two benefits:

- Motor can be run faster
- Better response to load changes

The communication and other functions that do not require real-time processing are executed in the main loop.

Code Example

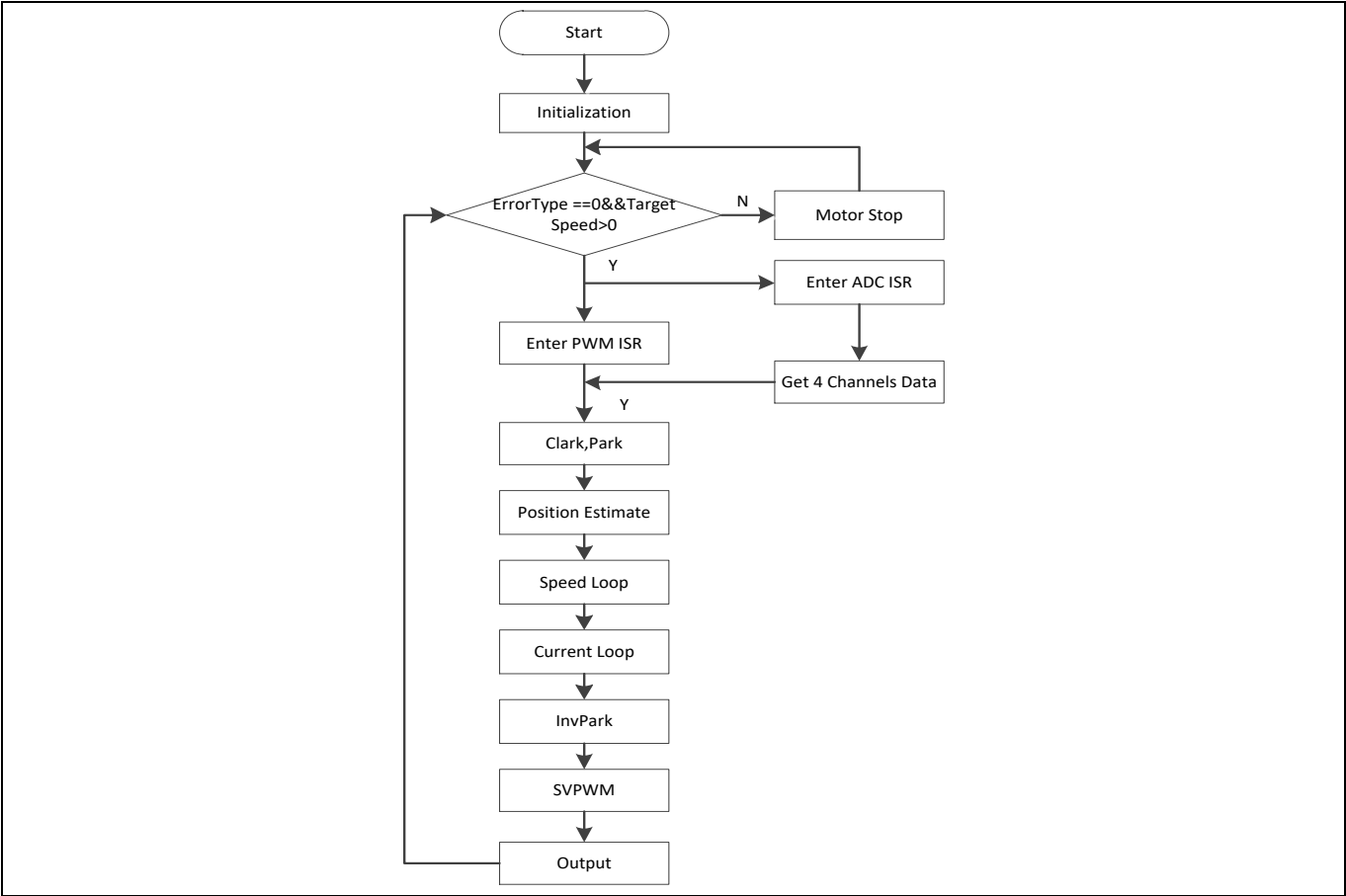


Figure 10 Firmware Execution Flow

In this project, each layer is independent. If the performance and independence is not affected, the files in the lower layer can call the files in the upper layer. In most cases, the files in upper layer call the files in the lower layer. [Figure 11](#) shows the relationship between each layer.

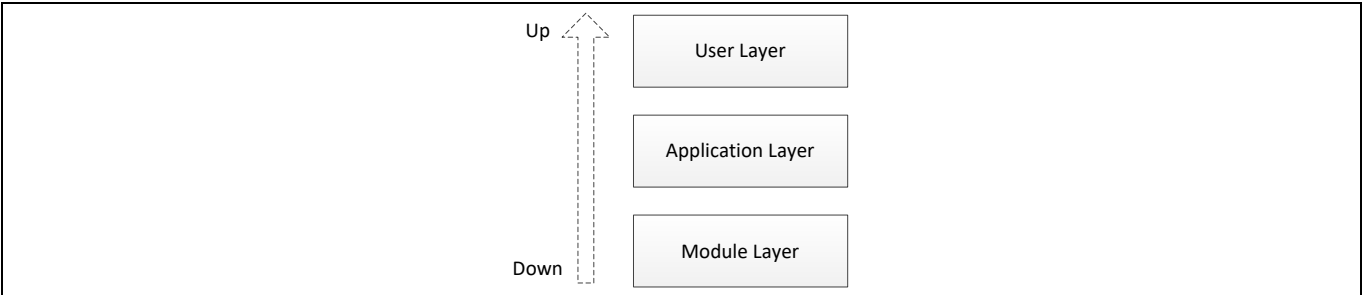


Figure 11 Relationship of each layer

Code Example

Table 2 Firmware Description

Folder Name	Description
h01_module	Event function declaration and variables definition folder
h02_app	Application function declaration and variables definition folder
h03_user	Hardware parameters, software parameters and MCU configuration header file folder
define.h	Global function and data format definition
s01_module	Event function folder
s02_app	Application function folder
s03_user	Hardware parameters and software parameters and MCU configuration source file folder

4.4 CY8CKIT-037 Kit

The Cypress kit CY8CKIT-037 is a motor-driver board designed to support three control algorithms: trapezoidal, FOC, and microstepping control for stepper motors. It has no MCU; it is a peripheral board to be used with the **CY8CKIT-045S (Figure 12)**, through the Arduino-compatible interface. For more information, refer to the **CY8CKIT-037 User Guide**.

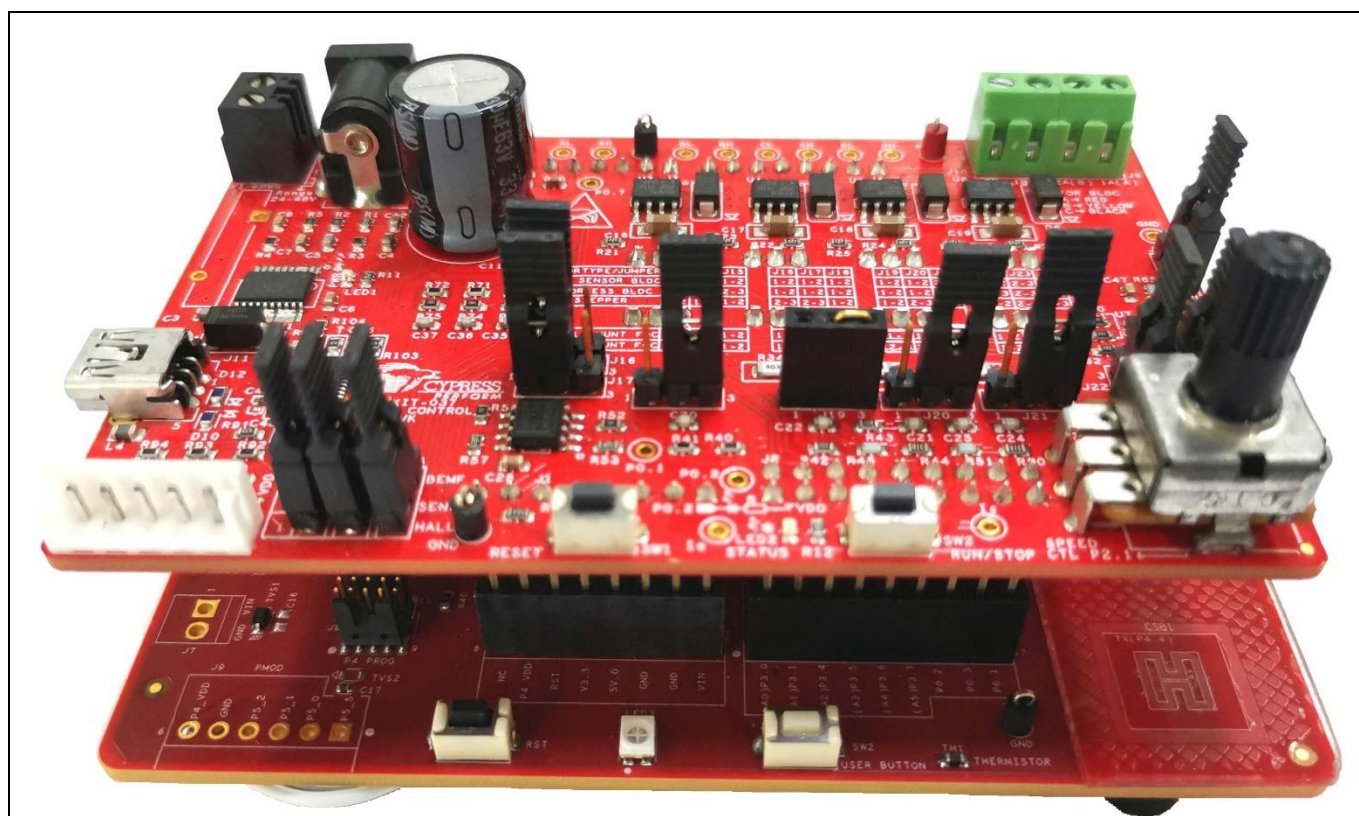


Figure 12 CY8CKIT-037 (Top) and CY8CKIT-045S (Underneath)

A PMSM, manufactured by **Anaheim Automation**, is included in this kit. **Table 3** lists the motor parameters. See **Appendix E: Adapting the Design to Other Motors** for information on how to change the code example by changing the motor parameters listed in this table.

Code Example

Table 3 Parameters for CY8CKIT-037 Motor

Item	Parameter
Part Number	BLY172S-24V-4000
Rated Torque (oz-in)	18.0
Rated Voltage (V)	24
Rated Power (watts)	53
Rated Speed (RPM)	4000
Torque Constant (oz-in/A)	5.03
Back EMF Voltage (V/kRPM)	4.14
Line-to-Line Resistance (ohm)	0.8
Line-to-Line Inductance (mH)	1.2
Rotor Inertia (oz-in-sec ²)	0.000680
"L" Length (in)	2.37
Shaft	Single

4.5 Operation

4.5.1 Step 1 – Configure CY8CKIT-045S

Select 5.0V as the VDD power at jumper J6 on the CY8CKIT-045S, as [Figure 13](#) shows.

Select 2-3 at jumper J14 and J15 when using 037Kit for motor control.

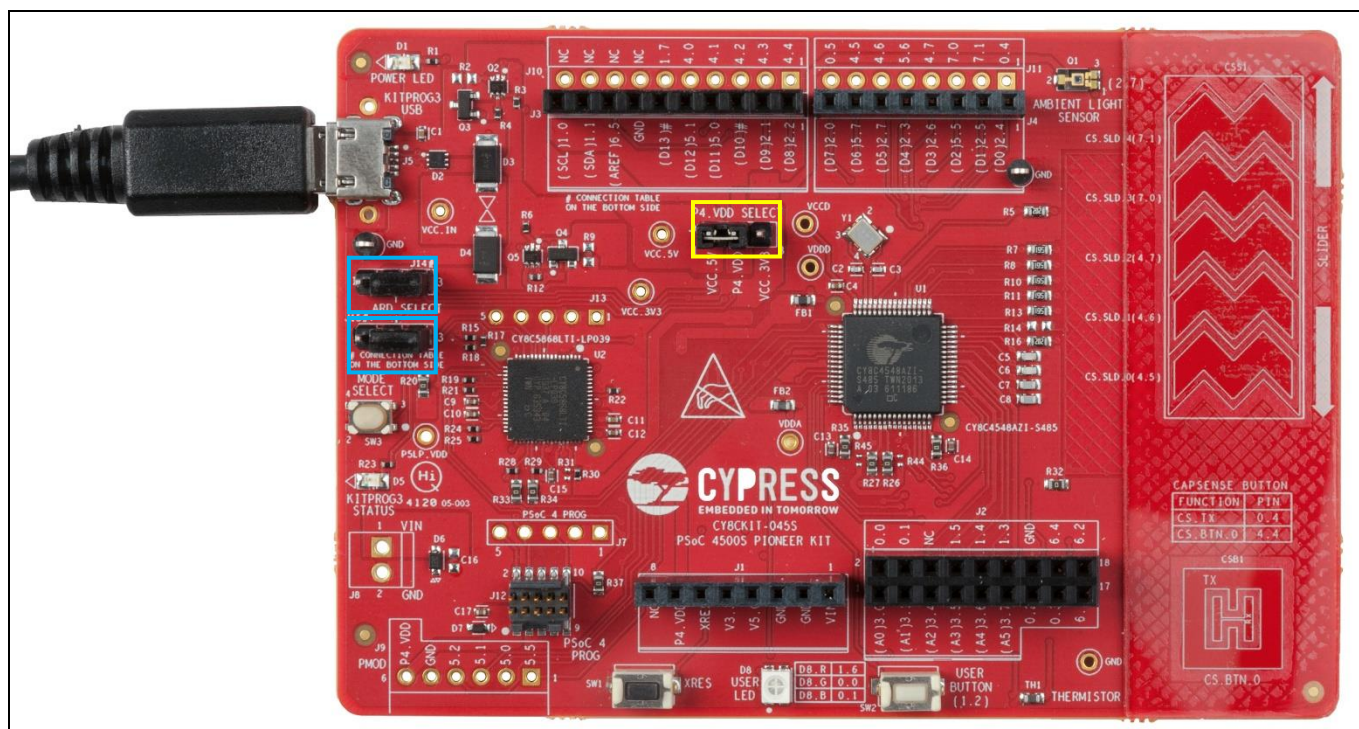


Figure 13 CY8CKIT-045S Configuration

Note: Please ensure to keep the USB cable connected to the Pioneer Kit because the Pioneer Kit does not provide a 5V converter. It gets 5V power directly from the USB port of the PC.

Code Example

4.5.2 Step 2 – Configure CY8CKIT-037

Configure the board via jumpers J13-J24 as listed in the row “BLDC 2-SHUNT FOC” printed on the board. See [Figure 14](#) and [Figure 15](#).

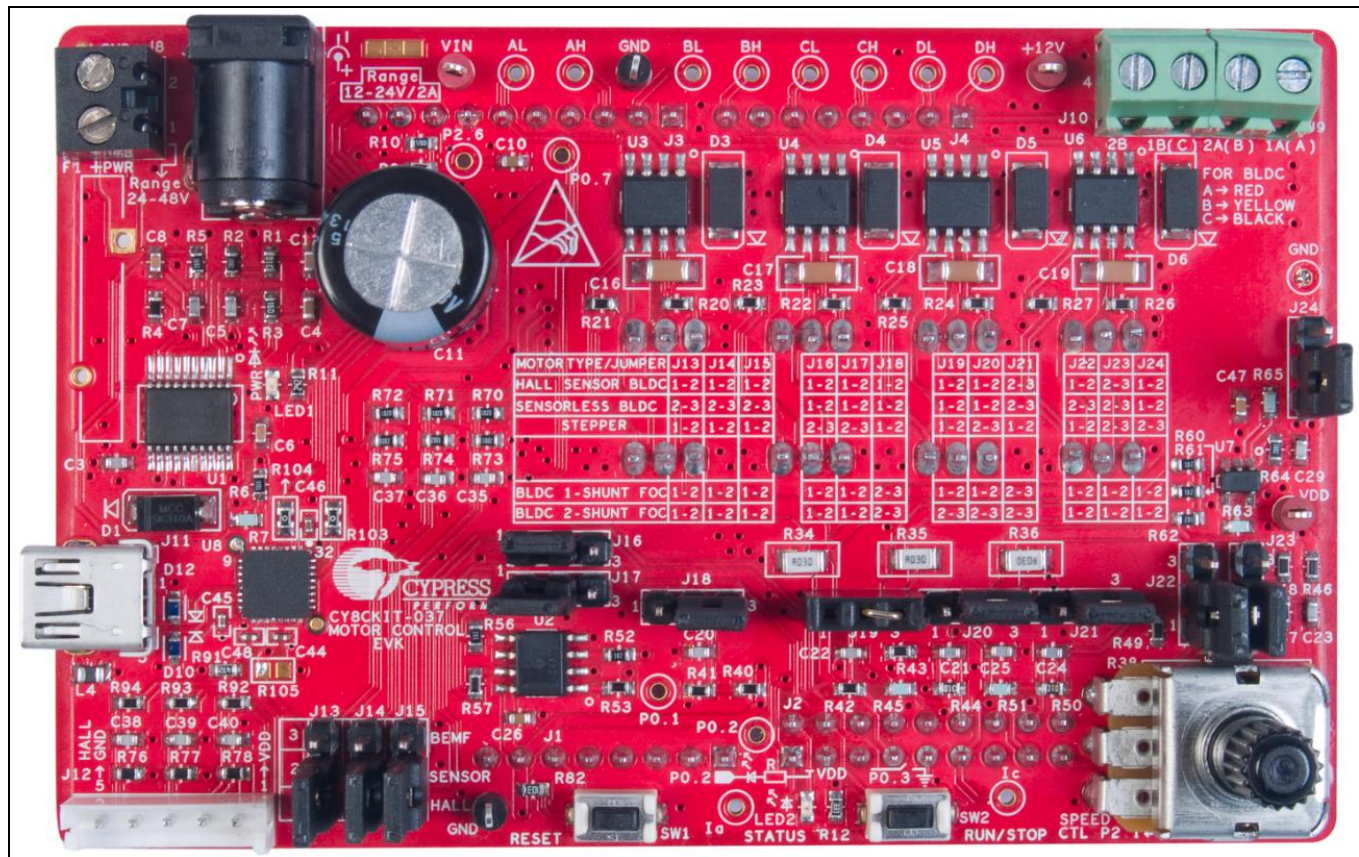


Figure 14 CY8CKIT-037 Configuration for Sensorless FOC Motor Control

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	1-2	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

Figure 15 Jumper Table for CY8CKIT-037

4.5.3 Step 3 – Plug CY8CKIT-037 into CY8CKIT-045S

Plug the CY8CKIT-037 into the CY8CKIT-045S via Arduino connectors J1-J4, as [Figure 12](#) shows.

4.5.4 Step 4 – Connect the Power Supply and Motor

Connect the BLDC motor to J9 and J10 on CY8CKIT-037. The other motor cable routes the signals from the sensors inside the motor. (The kit hardware supports sensed BLDC motors and sensed FOC.) Because this is a sensorless example, we do not need to connect this cable. Connect the 24-V power adapter to J7. See [Figure 16](#).

Code Example

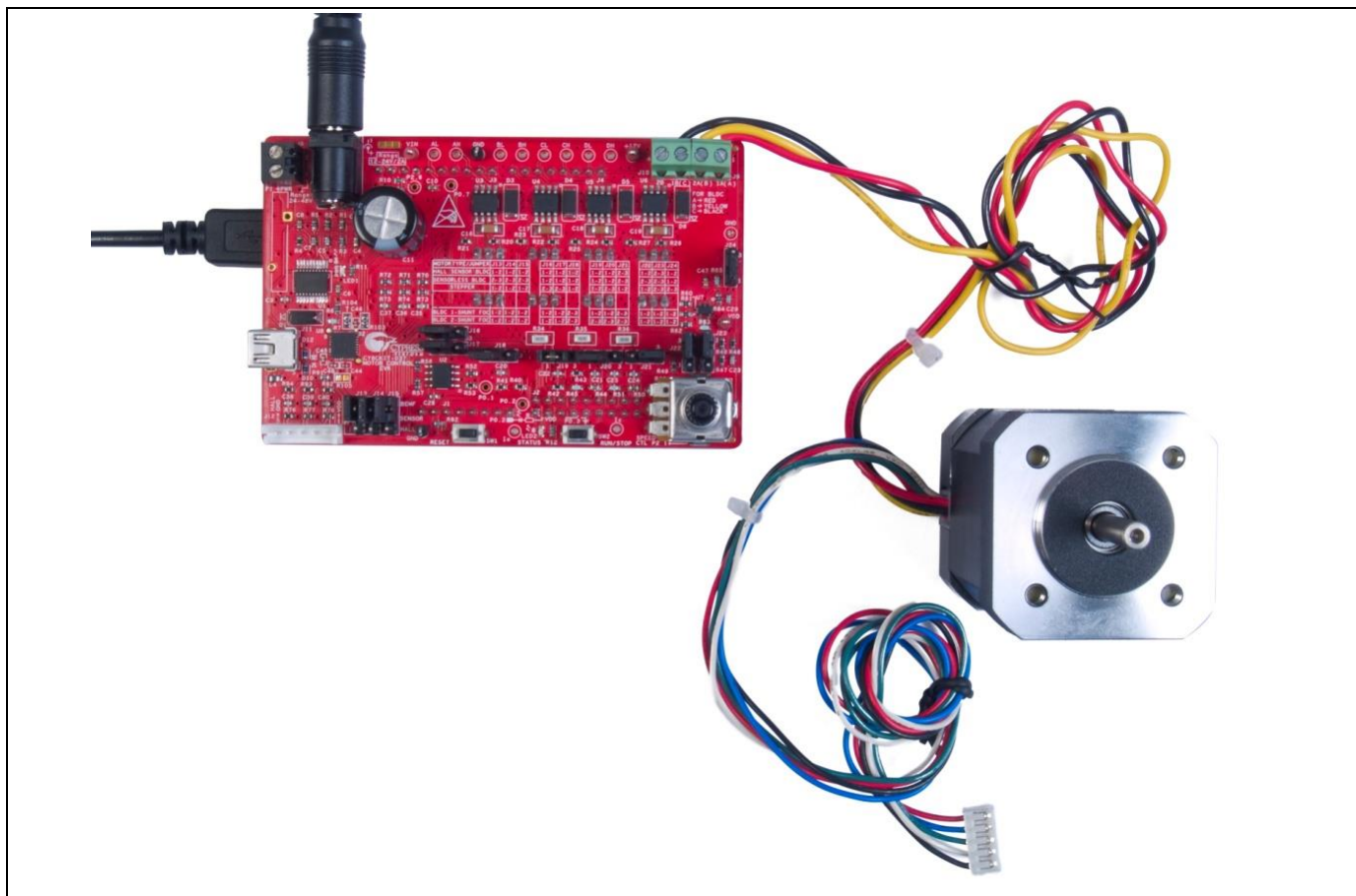


Figure 16 Connect Motor and Power Supply

4.5.5 Step 5 – Build the Project and Program the PSoC 4 Device

Open the sensorless FOC motor control code example project provided with this application note in PSoC Creator 4.4 or later. Select **Build > Build Sensorless FOC Motor Control**. When the build is complete, select **Debug > Program** to program the PSoC 4 device.

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

Rotate the potentiometer R38 to change the motor rotation speed (see [Figure 17](#)). Press the SW2 button to set the motor running direction. If the motor does not rotate, it indicates that an error has occurred. If so, first ensure that step 1 through step 5 have been executed correctly. Then press the **Reset** button and rotate the potentiometer R38 again. If the motor still does not rotate, there must be a problem in the hardware or software. You can debug it using a multimeter or oscilloscope to observe the signals or set breakpoints to monitor variables. You can also contact Cypress for [technical support](#).



Figure 17 Buttons and Status LED

Code Example

4.6 Performance

Figure 18 to Figure 20 show one of the phase currents for different motor speeds using the motor provided in the kit. Figure 21 shows the phase current during startup.

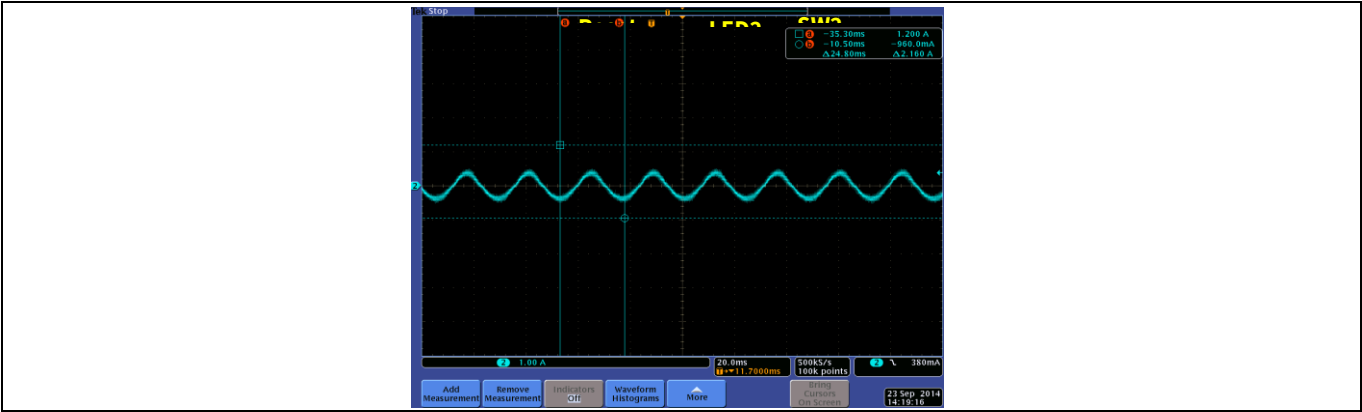


Figure 18 Phase Current – 600 RPM

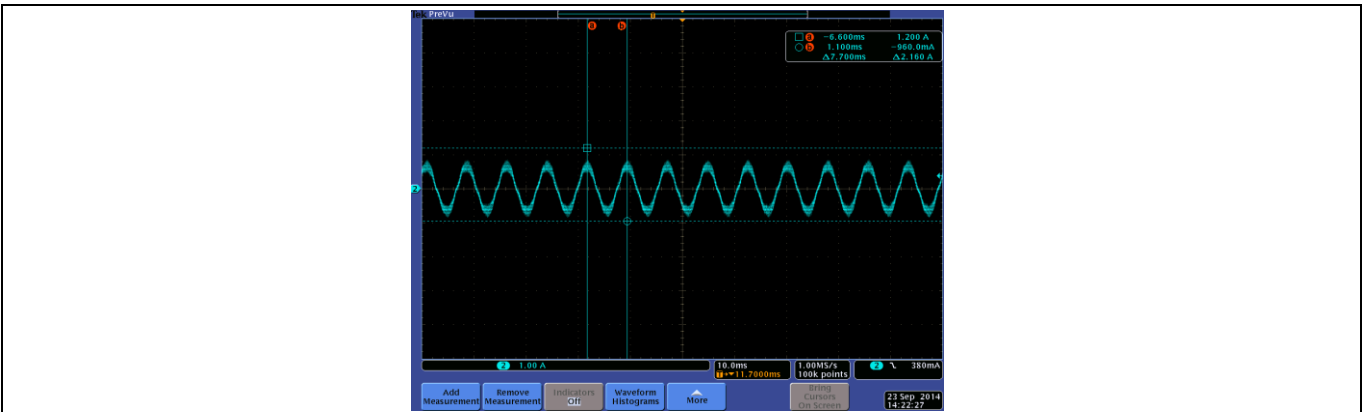


Figure 19 Phase Current– 2000 RPM

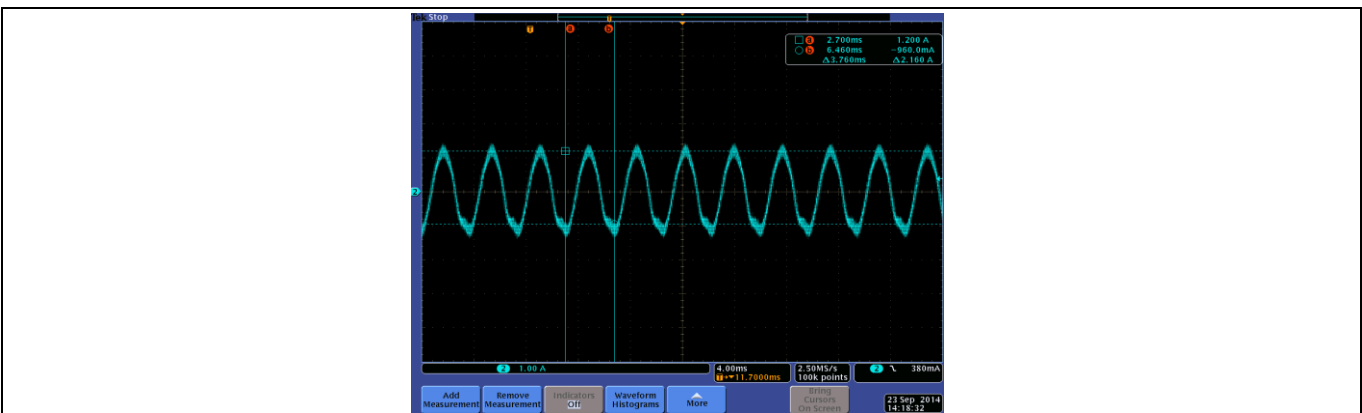


Figure 20 Phase Current– 4000 RPM

Code Example

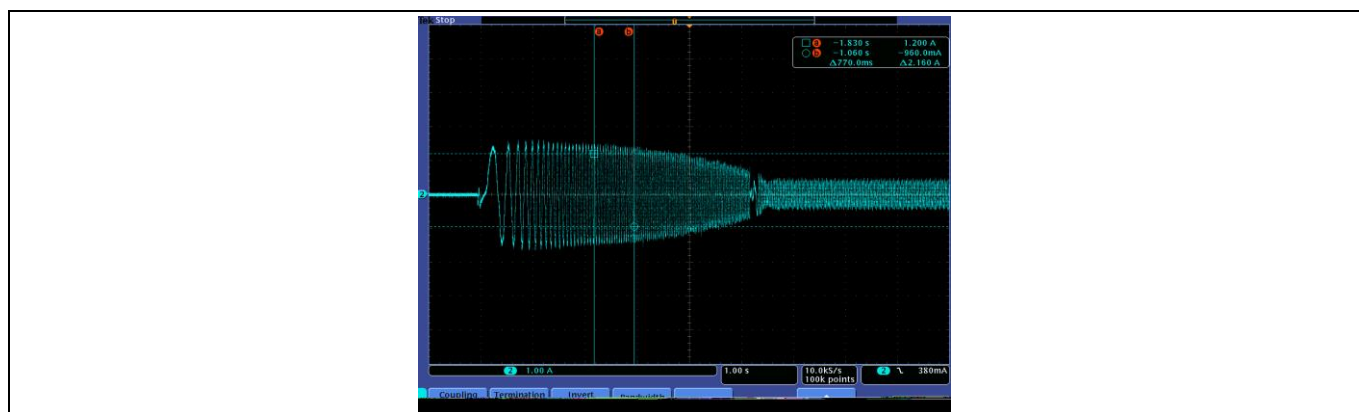


Figure 21 Phase Current at Startup

Design Details

5 Design Details

This section presents implementation details for each stage of the sensorless FOC processing listed in the PWM ISR (Figure 10), including current sampling, Clarke and Park transformations, SMO, PI controller, and SVPWM.

5.1 Current Sampling

5.1.1 PSoC Creator Schematic

Figure 22 shows the ADC schematic in the project. The “CC” of PWM_D is used to generate Start-of-Conversion signal for SAR ADC, so ADC could sample current when MOSFET is turned on, and it samples for the following signals:

Phase winding currents: AD_Ia and AD_Ib

Bus voltage (Vbus). This is the Inverter power supply (default 24 V) divided by resistors.

- Voltage input from variable resistor (potentiometer): VR.

An external sense resistor is used convert sense winding current into voltage, which is amplified using internal “Opamp_A” and “Opamp_B” Opamps. The amplified voltage is filtered using external R-C filter before connected to ADC inputs.

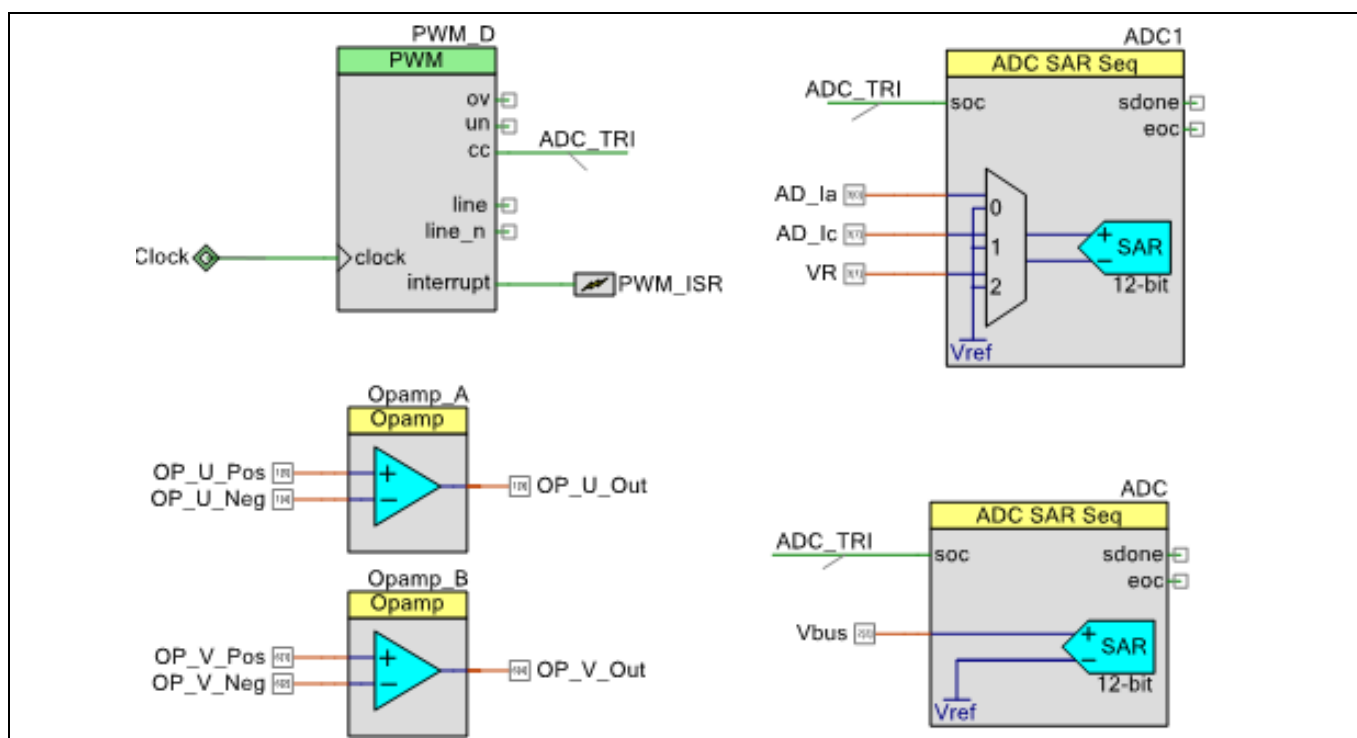


Figure 22 PSoC Creator Schematic – Current Sampling

Figure 23 and Figure 24 show the SAR ADC configuration with the following features:

There are two ADCs used for motor phase current and other ADC (such as VBus voltage and potentiometer) items’ sampling on CY8CKIT-045S because some pins need to be multiplexed for other functions. By the way, these two ADCs have same configurations.

- Sampling clock is 12 MHz for an 888-kcps sampling rate. The actual sample rate shown in Figure 23 indicates that four channels are being sampled ($888 / 4 = 222$).

Design Details

- Voltage reference is $V_{DDA}/2$ to obtain a 0-to- V_{DDA} input range.
- All channels are single-ended.
- The sampling result is unsigned.
- A hardware trigger starts sampling. After four channels are sampled, the ADC stops and waits for the next trigger signal. The trigger frequency is 10 kHz. The PWMs provide a common timing for ADC sampling, CPU interrupt, and MOSFET control – see [Figure 9](#), [Figure 10](#), and the [SVPWM Generation](#) section.

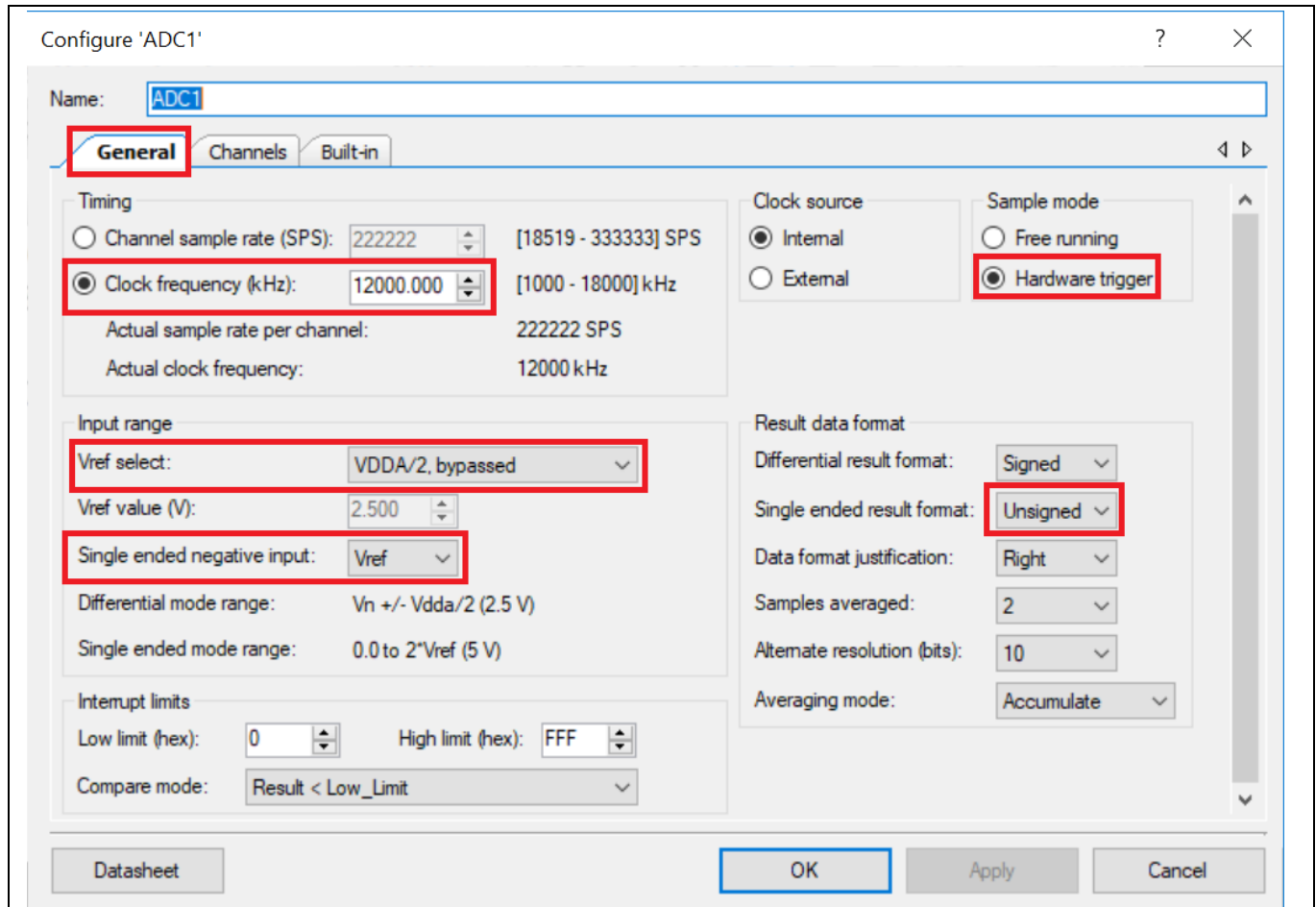


Figure 23 SAR ADC Configuration, General Tab

Design Details

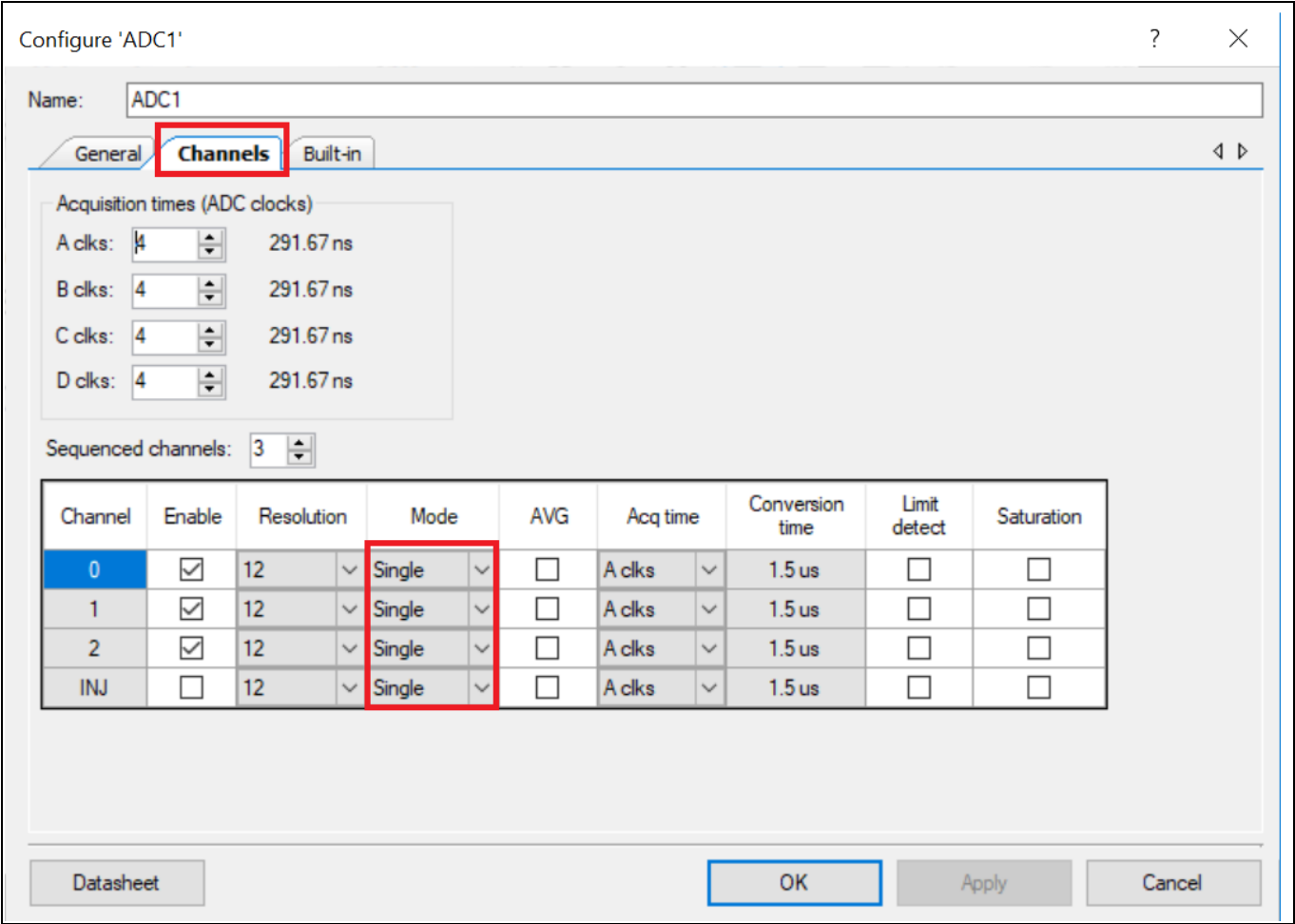


Figure 24 SAR ADC Configuration, Channels Tab

The motor phase current is converted to a voltage by the sensing resistors, as [Figure 25](#) shows. The figure also shows that, because the sum of the three currents must be zero at the sampling point, we can sample just two of the currents and calculate the third.

The opamp gains and the sensing resistor values are selected so that:

- The voltage stays in the ADC input range when the current is at the rated maximum. Sensing resistors are typically on the order of milliohms.
- The measurement of low currents is accurate. The sensing resistors have a tolerance of 1%.

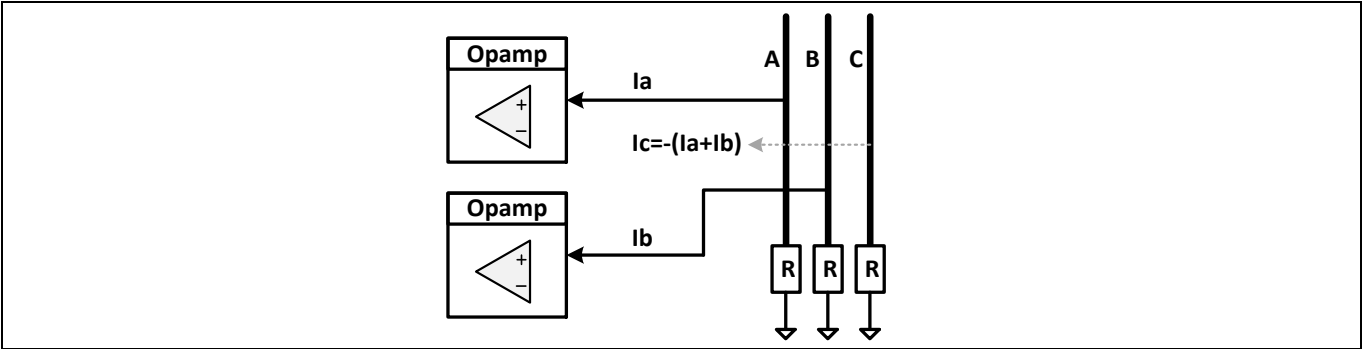


Figure 25 Dual-Shunt Current Sampling

Design Details

Figure 26 shows the schematic design for the CY8CKIT-037 kit. The kit board has 30-mΩ sensing resistors (not shown) and a 2.1-A rated current. Bias resistors (R40, R41) are included to handle positive and negative currents.

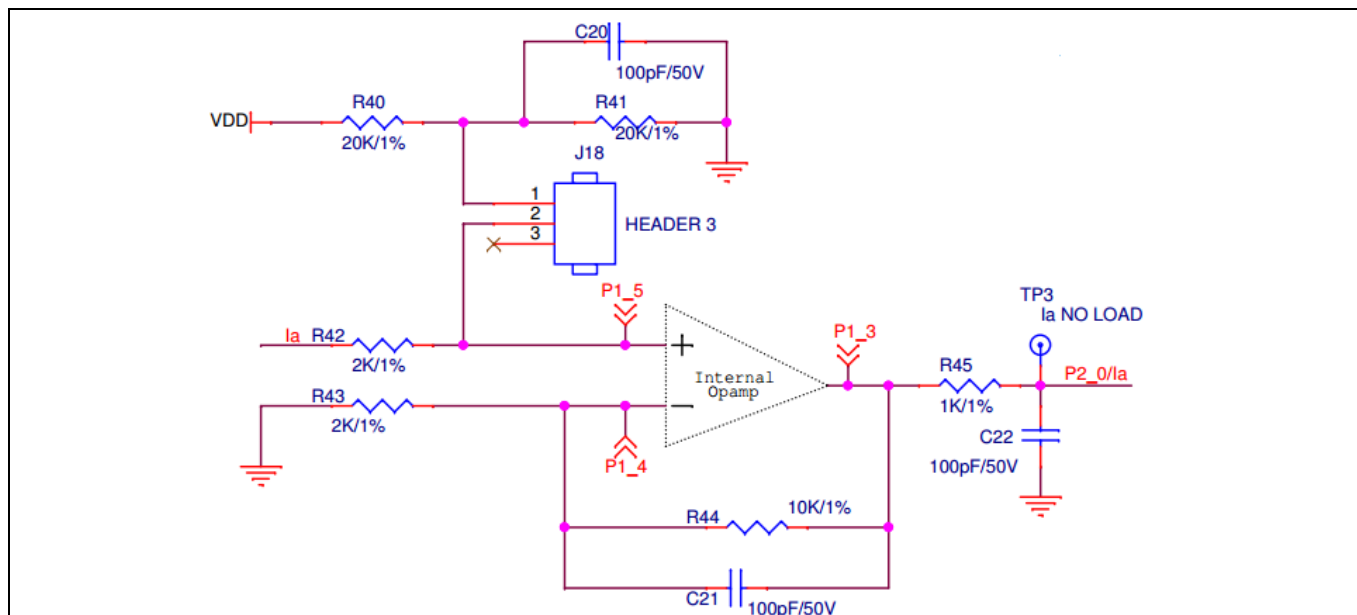


Figure 26 CY8CKIT-037 Schematic: Signal Conditioning for Phase-A Current

The PSoC 4 internal opamps are used. **Figure 27** shows the configuration of each Opamp Component.

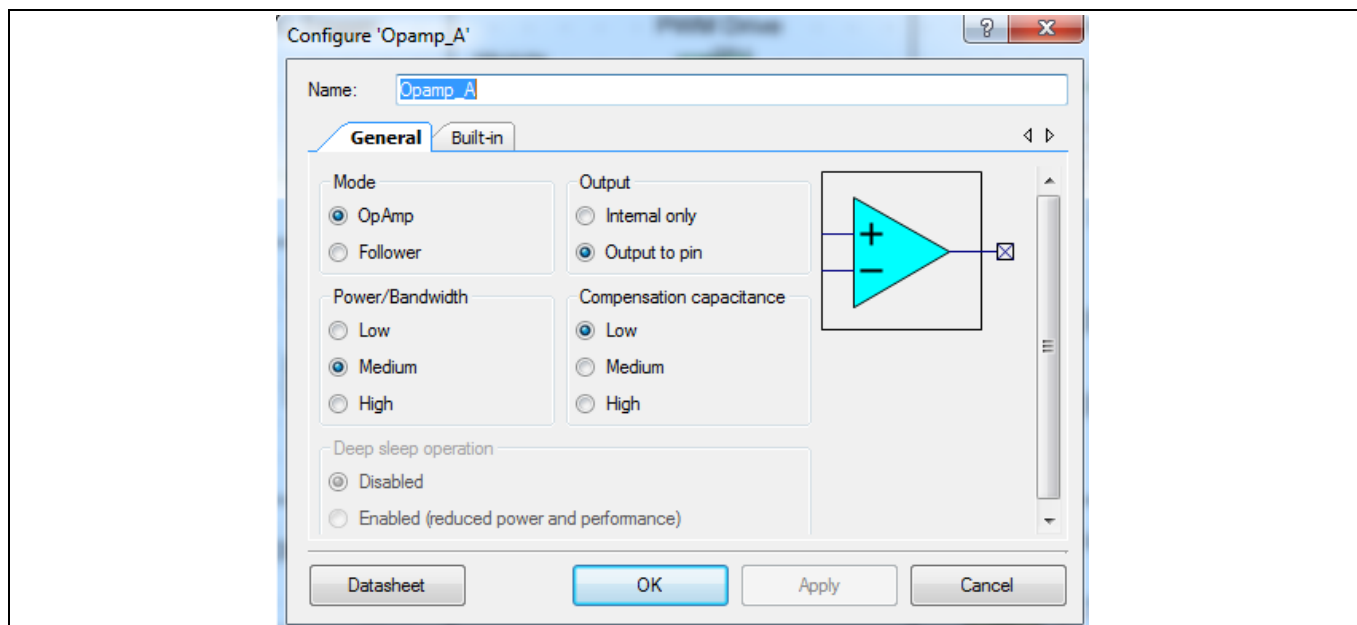


Figure 27 PSoC 4 Internal Opamp Configuration

Figure 28 shows the pin assignments for the SAR ADC and opamps. The opamp pins are direct connected to the internal opamps. Using these pins reduces the usage of analog routing resources as well as resistances between the pins and the opamp terminals.

Design Details










	AD_Ia	P3[0]	▼	18	▼	✓
	AD_Ic	P3[7]	▼	25	▼	✓
	Ibus	P0[2]	▼	41	▼	✓
	OP_U_Neg	P1[4]	▼	62	▼	✓
	OP_U_Out	P1[3]	▼	61	▼	✓
	OP_U_Pos	P1[5]	▼	63	▼	✓
	OP_V_Neg	P6[2]	▼	14	▼	✓
	OP_V_Out	P6[4]	▼	15	▼	✓
	OP_V_Pos	P6[1]	▼	13	▼	✓

Figure 28 Pin Assignments for Opamp and SAR ADC

5.1.2 Current Sampling Firmware

The firmware for current sampling consists of initialization function calls and functions to read and use the sample data. The code example associated with the kit and this application note includes a Cypress-proprietary motor control library. This library manages the sensorless FOC procedure, as described in the [Firmware](#) section. Some system parameters such as DC bus voltage and sensing resistors value are defined as macros in the motor control library. For more information, see [Appendix E: Adapting the Design to Other Motors](#).

For system initialization, the Opamp Component API has a Start() function, as [Code Listing 1](#) shows. The SAR ADC is initialized by a motor control library function (see [Figure 11](#)).

Code Listing 1 Current Sampling Initialization

```

/* main.c */

void main()
{
    /* Initialize Opamps */
    Opamp_A_Start();
    Opamp_B_Start();
    /* Initialize SAR ADC */
    Adc_Start();
}

/*****
/* Cymc_HAL_ADC.c */

void Adc_Start();
{
    /* start SAR ADC */
    SADC_Start();
}

```

Current sampling is done in the ISR, as [Code Listing 2](#) shows. A motor control library function (see [Figure 11](#)) is called to read the ADC raw data.

Design Details

Code Listing 2 Current Sampling in the ISR

```

/* isr.c */
CY_ISR(FOC_MainLoop_ISR)
{
    . . .
    /* Read ADC and get phase current*/
    Adc_ReadSample();
    . . .
    MotorCtrl_stcRunPar.i32Q8_VR =
    (Adc_u16RsltOfAdc[MOTOR_SPEED_VR]) >> 5;

}

/*****
/* adc_sample.c */
void Adc_ReadSample()
{
    uint8 i = 0;
    for(i = 0; i < ADC_CH_AMOUNT; i++)
    {
        Adc_u16RsltOfAdc[i] = SADC_GetResult16(i) & (0xFFFF);
    }
}

```

5.2 Transformations

Four structures are defined in the module layer in motor control library (**Figure 11**) for Clarke and Park transformations and the inverse transformations, as **Code Listing 3** shows. Four functions are defined to do the transformations. The structures and function prototypes are declared in the motor control library file *coordinate_transform.h*:

Code Listing 3 Clarke and Park Transformation Structures and Function Prototypes

```

/* coordinate_transform.h*/
/* struct definition for coordinate transformation*/
typedef struct
{
    int32_t i32Q8_Xu;    /*Phase U variable*/
    int32_t i32Q8_Xv;    /*Phase V variable*/
    int32_t i32Q8_Xw;    /*Phase W variable*/
}stc_uvw_t;

typedef struct
{
    int32_t i32Q8_Xa;    /*Alpha axis variable*/
    int32_t i32Q8_Xb;    /*Beta axis variable*/
}stc_ab_t;

```


Design Details

```
typedef struct
{
    int32_t i32Q8_Xd;    /*D-axis variable*/
    int32_t i32Q8_Xq;    /*Q-axis variable*/
    int32_t i32Q12_Cos; /*Angle sin variable*/
    int32_t i32Q12_Sin; /*Angle cos variable*/
}stc_dq_t;

extern void Clark(stc_uvw_t *pstc_uvw, stc_ab_t *pstc_ab);
extern void InvClark(stc_ab_t *pstc_ab, stc_uvw_t *pstc_uvw);
extern void Park(stc_ab_t *pstc_ab, stc_dq_t *pstc_dq);
extern void InvPark(stc_dq_t *pstc_dq, stc_ab_t *pstc_ab);
```

Code Listing 4 shows how to use these functions:

Code Listing 4 Using Clarke and Park Transformation Functions

```
/* motor_ctrl.c */

MotorCtrl_Process
{
    /* Clarke Transformation uvw -> αβ */
    Clark(&MotorCtrl_stcIuvwSensed, &MotorCtrl_stcIabSensed);

    /* Park Transformation αβ -> dq */
    Park(&MotorCtrl_stcIabSensed, &MotorCtrl_stcIdqSensed);

    /* InvPark Transformation dq-> αβ */
    InvPark(&MotorCtrl_stcVdqRef, &MotorCtrl_stcVabRef);

    /* InvClark Transformation αβ -> uvw */
    InvClark(&_2sC_Ref, &pstcPar->_3sC_Ref);
}
```

5.3 Slide Mode Observer (SMO)

An introduction to SMO theory is in [Appendix B: Slide Mode Observer \(SMO\)](#). The structure and function prototypes for SMO calculation ([Code Listing 5](#)) are defined in the module layer of motor control library (see [Figure 11](#)).

Code Listing 5 SMO Calculation Structure and Function Prototypes

```
/*smo_calculate.h*/

typedef struct stc_SMO_Estimator
{
    int32_t i32Q8_Res;    /*the phase resistance*/
    int32_t i32Q8_Lddt;   /*q axis inductance digital factor*/
    int32_t i32Q12_LdLq; /*dq Axis Mutual Inductance*/
}
```

Design Details

```

int32_t i32Q8_IalphaPre; /*stationary alpha-axis stator current*/
int32_t i32Q8_IbetaPre; /*stationary beta-axis stator current*/
int32_t i32Q8_ValphaPre; /*stationary alpha-axis stator voltage*/
int32_t i32Q8_VbetaPre; /*stationary beta-axis stator voltage */
int32_t i32Q8_ValphaBemf; /*eestimated alpha Back EMF*/
int32_t i32Q8_VbetaBemf; /*eestimated beta Back EMF*/
int32_t i32Q8_ValphaBemfLpf; /*filtered alpha Back EMF for angle calculate*/
int32_t i32Q8_VbetaBemfLpf; /*filtered beta Back EMF for angle calculate*/
stc_one_order_lpf_t ValphaBemLpfK; /*LPF calculate factor*/
stc_one_order_lpf_t VbetaBemLpfK; /*LPF calculate factor*/
int32_t i32Q22_EstimWmHz; /*estimated rotor speed Q22 format*/
int32_t i32Q8_EstimWmHz; /*estimated rotor speed Q8 format*/
int32_t i32Q8_EstimWmHzf; /*filtered estimated rotor speed Q8 format*/
stc_one_order_lpf_t stcWmLpf; /*LPF calculate factor*/
int32_t i32Q12_Cos;
int32_t i32Q12_Sin;
int32_t i32Q12_CosPre;
int32_t i32Q12_SinPre;
int32_t i32Q22_Theta; /*estimated rotor angle*/
int32_t i32Q22_ThetaOld; /*estimated rotor angle old*/
int32_t i32Q22_Dtheta; /*delta theta of rotor angle for speed calculate*/
uint16_t u16_1msCount; /*counter used to calculate motor speed*/
int32_t i32Q12_MaxLPFK; /*BackEMF voltage's max filter parameter*/
int32_t i32Q12_MinLPFK; /*BackEMF voltage's min filter parameter*/
int32_t i32Q15_LPFKTS; /*BackEMF filter's calculation factor*/
uint16_t u161msTimer; /*1ms timer count*/
int32_t i32SpdCalKts; /*speed calculate factor*/
uint8_t u8closeLoopFlg; /*closed loop flag*/
}stc_SMO_Estimator_t;

extern void Smo_Estimate(stc_SMO_Estimator_t *pstcEstimPar, stc_ab_t *pstc2sVol, stc_ab_t
*pstc2sCurrent);

extern void Smo_Init(stc_SMO_Estimator_t *SMO_Eistimator_t);

```

A global variable of type SMO structure ([Code Listing 6](#)) is also defined in the motor control library (see [Figure 11](#)).

Code Listing 6 SMO Structure Variable

```

/* smo_calculate.h */
extern stc_SMO_Estimator_t Motor_stcSMO; /* slide mode
observer struct */

```

Design Details

To use the SMO module, you must first call the motor control library function `MotorCtrl_Start()` in your initialization code. **Code Listing 7** shows initialization and usage of SMO:

Code Listing 7 SMO Initialization and Usage

```
/* motor_ctrl.c */
void MotorCtrl_Start(uint16_t u16SampleFreq)
{
    . . .
    /* Initialize Motor Parameters */
    Smo_Init(&Motor_stcSMO);
    . . .
}
```

Code Listing 8 shows how to use the SMO routines to obtain and use the angle information that is stored in the variable 'Motor_stcSMO':

Code Listing 8 Using the SMO Module in the ISR

```
/* motor_ctrl.c */
void MotorCtrl_Process(void)
{
    . . .
    /* Position Calculation*/
    Smo_Estimate(&Motor_stcSMO, &MotorCtrl_stcVabReal, &MotorCtrl_stcIabSensed);
    . . .
}

static void MotorCtrl_ThetaGen(void)
{
    . . .
    MotorCtrl_stcRunPar.i32Q8_EstimWmHz = Motor_stcSMO.i32Q8_EstimWmHz;
    MotorCtrl_stcRunPar.i32Q22_ElecAngle = Motor_stcSMO.i32Q22_Theta;
    MotorCtrl_stcRunPar.i32Q22_DeltaThetaTs = Motor_stcSMO.i32Q22_Dtheta;
    . . .
}
```

5.4 PI Controllers

The PI regulator keeps the output follow the expected output by a comparing error between the expected output and the real output. The P-value is to make a fast output response to the comparing error, and the I-value is to decrease the stable output errors. The transfer function can be expressed as shown in **Figure 29**.

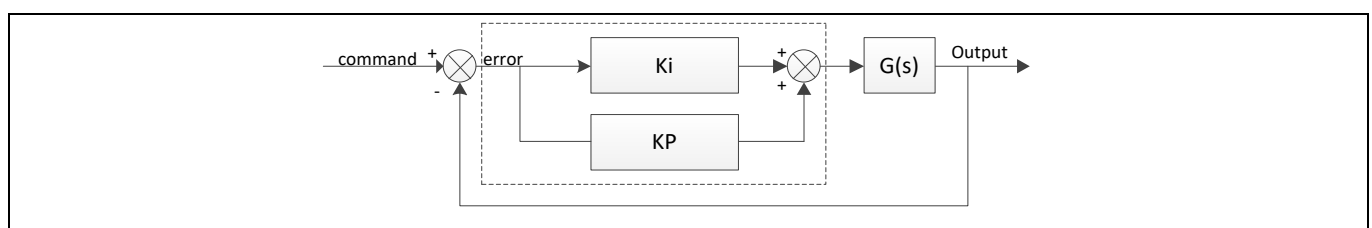


Figure 29 PI-regulator Controller

Design Details

PI regulator causes a fluctuating output. The fluctuating amplitude decreases, and after regulating period, the output follows the expected output with a very small fluctuation around the expected output value.

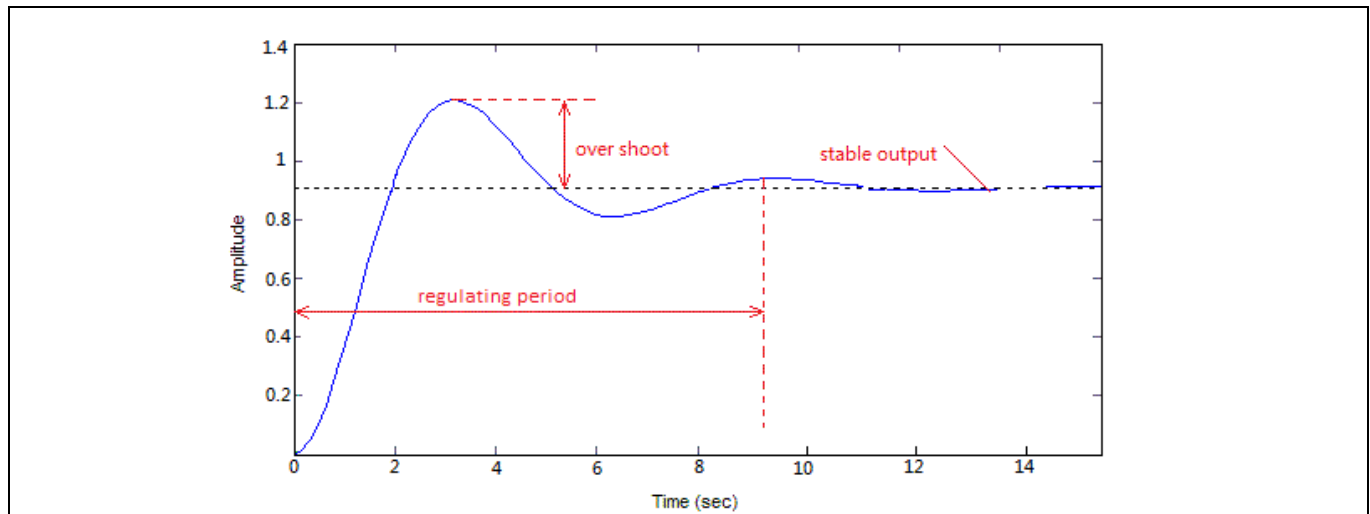


Figure 30 PI Regulator Output

PI regulator formula:

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau) d\tau \quad \text{Equation 1}$$

Incremental algorithm:

$$\Delta u(k) = k_p [e(k) - e(k-1)] + k_i e(k) \quad \text{Equation 2}$$

$$u(k) = u(k-1) + \Delta u(k) \quad \text{Equation 3}$$

Where,

k_p : Proportional factor

k_i : Integration factor

$e(k)$: error between actual and reference

$e(k-1)$: last error

$u(k)$: output value of PI regulator

$u(k-1)$: last output value of PI regulator

$\Delta u(k)$: differential value between two output value

PI output limitation:

This is to limit the PI output to a regular range:

Design Details

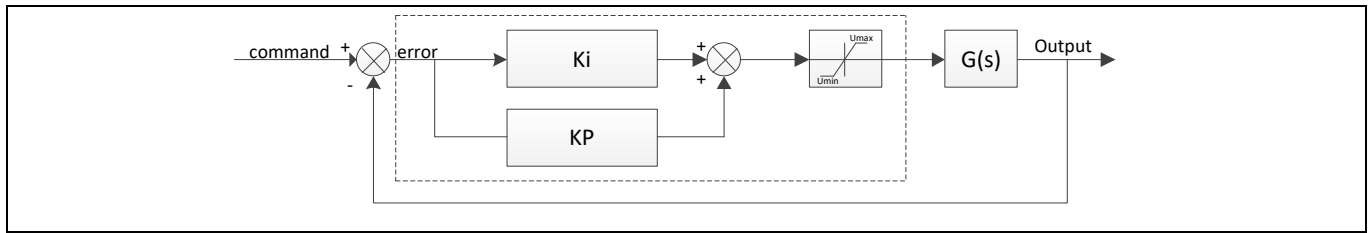


Figure 31 PI Regulator with Limitation

Three parameters – motor speed, i_q , and i_d – are controlled by separate PI controllers. The speed PI controller uses the error between the calculated rotation speed and a given speed reference to calculate the control output, which in turn is the reference for the i_q PI controller. The i_q and i_d PI controllers control u_q and u_d , respectively, using the errors for i_q and i_d . See [Figure 4](#).

[Code Listing 9](#) shows how to implement PI control.

Code Listing 9 PI Controllers

```
/* foc.c */
CY_ISR(FOC_MainLoop_ISR)
{
    . . .

    /* Speed PI Controller */
    Pid_Pos(&MotorCtrl_stcWmPidReg, MotorCtrl_stcRunPar.i32Q8_Ta
rgetSpeedWmHz - MotorCtrl_stcRunPar.i32Q8_EstimWmHzf);

    /* Iq PI Controller */
    Pid_Pos(&MotorCtrl_stcIqPidReg, MotorCtrl_stcIdqRef.i32Q8_Xq
- MotorCtrl_stcIdqSensed.i32Q8_Xq);

    /* Id PI Controller */
    Pid_Pos(&MotorCtrl_stcIdPidReg, MotorCtrl_stcIdqRef.i32Q8_Xd
- MotorCtrl_stcIdqSensed.i32Q8_Xd);
}
```

5.5 SVPWM Generation

5.5.1 PSoC Creator Schematic Design

The SVPWM subsystem produces sinusoidal currents on the motor phases by changing the output duty cycles of the three PWMs (for details, see [Appendix C: SVPWM Theory](#)). The PWM outputs – two complementary outputs for each motor phase – turn the MOSFETs ON or OFF (see [Figure 3](#)).

[Figure 32](#) shows the SVPWM implementation in PSoC Creator. A common 48-MHz clock synchronizes the PWM outputs.

Design Details

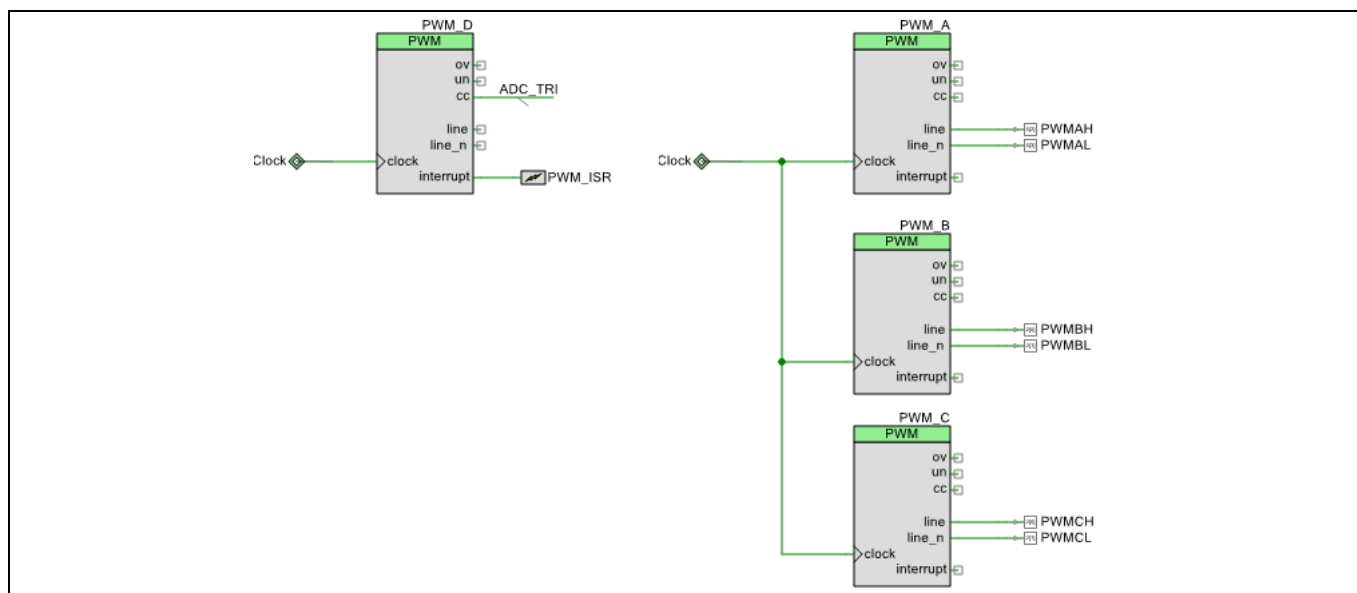


Figure 32 PSoc Creator Schematic: SVPWM and Timing

Figure 33 shows the timing for all three PWMs as well as the details of PWM_A. In addition to the PWM signals, PWM_D generates the trigger signals for PWM interrupt(FOC_MainLoop_ISR) – see the **Firmware** section.

The PWM interrupt is triggered on the terminal count of PWM_D. The result is that the ISR controls the PWM duty cycle on every cycle by updating the PWM compare buffer register (**Figure 34**). The register must be updated before the next underflow event occurs, or the duty cycle will be incorrect, which in turn causes an increased motor noise.

Note that each PWM has a different duty cycle.

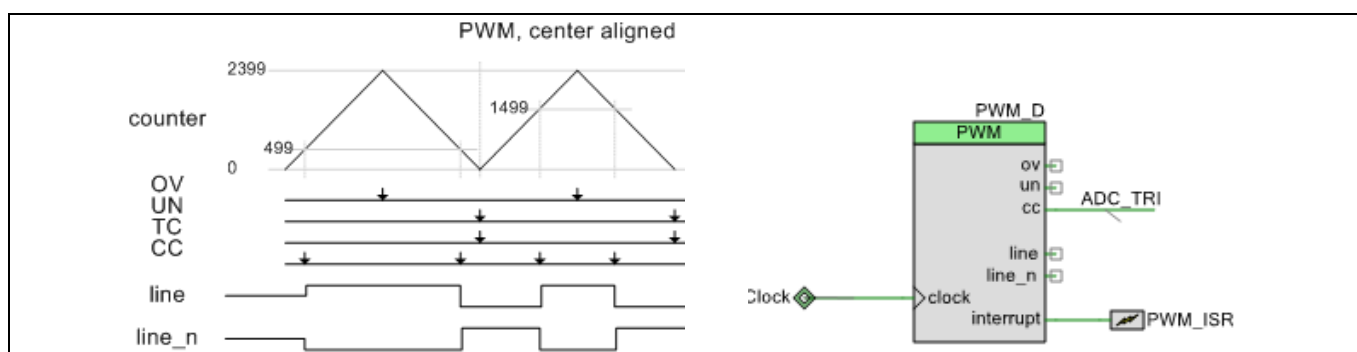


Figure 33 PWM Timing and PWM_D detail

Figure 34 shows the configuration for PMW_A (phase A); it applies to all three PWM Components:

- The alignment mode is “Center align”. This produces the complementary PWM outputs ‘line’ and ‘line_n’. The outputs turn the MOSFETs of one of the motor phases ON and OFF (such as Q1 and Q2 in **Figure 3**).
- A deadband time is inserted to avoid turning ON both MOSFETs at the same time, which can damage the MOSFETs. In this code example, 38 cycles of a 48-MHz clock results in a dead time of 0.8 μ s.
- The period value is the clock frequency divided by twice the desired PWM frequency minus 1. We double the desired PWM frequency because the count mode is up-down (see **Figure 33**). For a 48-MHz clock and a desired PWM frequency of 10 kHz, the period is $(48,000,000 / (2 * 10,000)) - 1$, or 2,399.
- To implement the duty cycle control technique described **previously**, we do the following:

Design Details

- route the PWM_A ‘un’ output back to the ‘switch’ input of all PWMs (signal ‘PWM_UPDATE’ in [Figure 32](#))
- check the “switch on rising edge” and “swap” boxes ([Figure 34](#))
- Update the PWM compare buffer register in firmware. On each cycle, the PWM hardware copies (by swapping) the compare buffer register into the compare register, which changes the duty cycle.

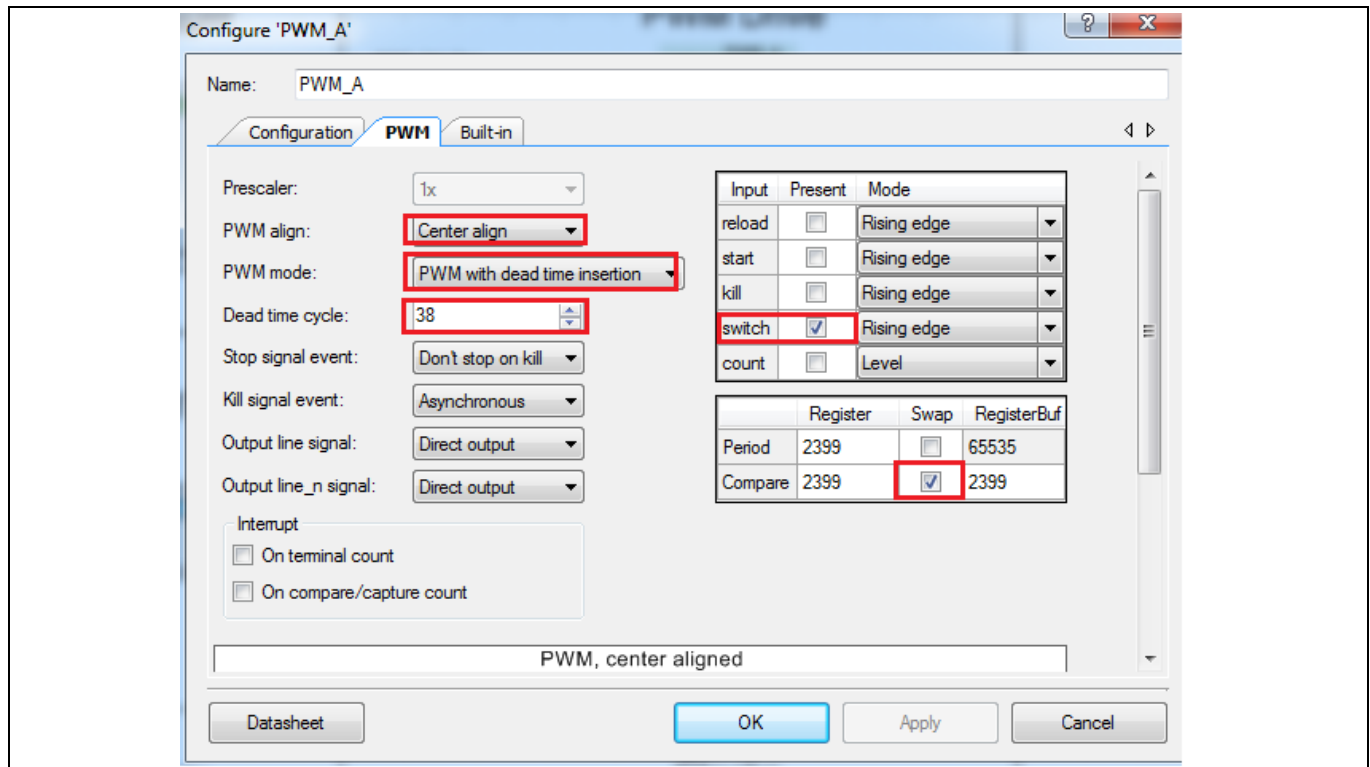


Figure 34 PWM_A Configuration Window

[Figure 35](#) shows the configuration for PMW_A (phase A); it applies to all three PWM Components:

- The alignment mode is “Left align”
- The period is two times longer than PWM_A. For a 48-MHz clock and a desired PWM frequency of 10 kHz, the period is $\ast(48,000,000 / (2 \ast 10,000)) - 1 \ast 2$, or 4,798.

Design Details

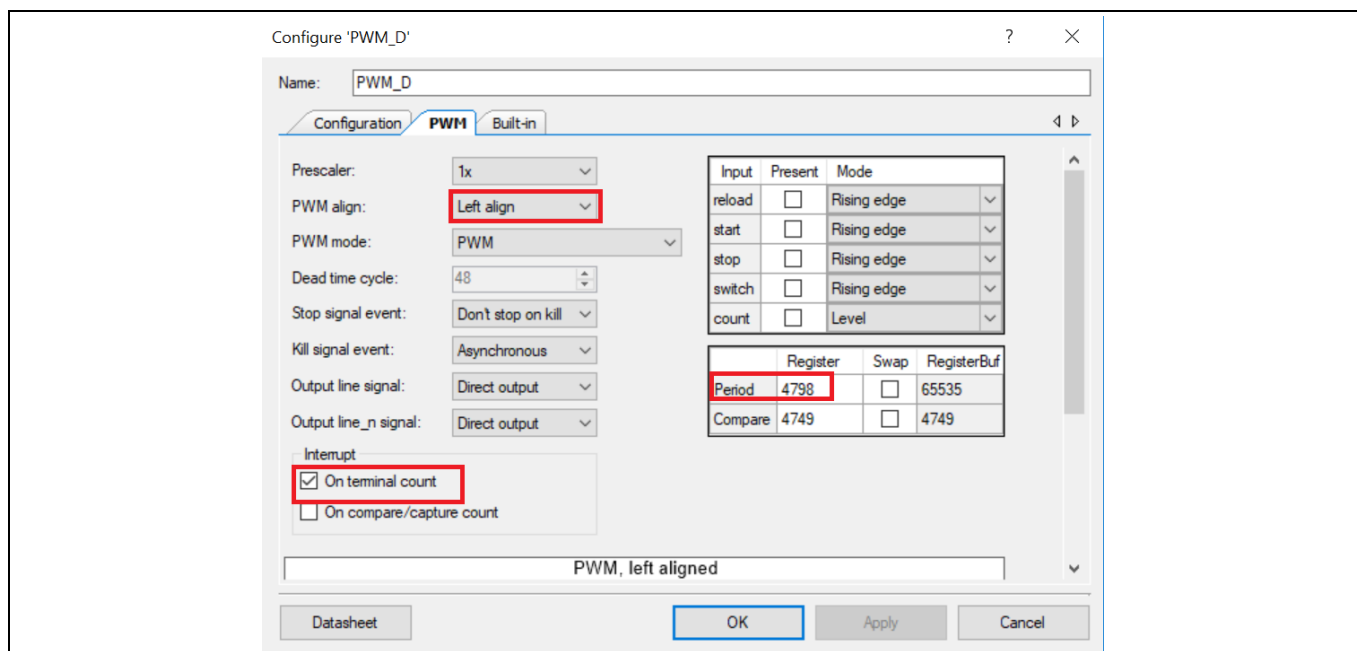


Figure 35 PWM_D Configuration Window

5.5.2 Firmware Implementation

Code Listing 10 shows how to initialize the PWMs and update them in the FOC ISR.

Code Listing 10 SVPWM Control

```
/* main.c */
void main()
{
    . . .
    /* TCPWM Init */
    PWM_Start();
    /* Motor Parameters Init */
    MotorCtrl_InitPar(Motor_ul6CarrierFreq);
    /* PWM Main ISR Init */
    PWM_MainLoop_ISR_StartEx(FOC_MainLoop_ISR);
    . . .
}

/***** isr.c *****/
CY_ISR(FOC_MainLoop_ISR)
{
    . . .
    uint32_t Clear_ISR;
    /*read interrupt status register*/
    Clear_ISR = PWM_D_GetInterruptSourceMasked();
    MotorCtrl_Process();
    Led_CountTime++;
}
```

Design Details

```
/*clear handled interrupt*/  
PWM_D_ClearInterrupt(Clear_ISR);  
. . .  
}
```

Summary

6 Summary

This application note has shown how to design a sensorless FOC control algorithm for a permanent magnet synchronous motor (PMSM), using a PSoC 4500S device. PSoC 4's resources and flexible internal routing make it an excellent fit for motor control applications. For more information on motor control using PSoC 4, go to the [Cypress](#) website.

Resources

7 Resources

- [1] [AN88619 – PSoC4 4100/4200 Hardware Design Considerations](#)
- [2] [AN90799 – PSoC4 Interrupts](#)
- [3] [PSoC® 4500S Pioneer Kit \(CY8CKIT-045S\) User Guide and Schematic](#)
- [4] [AN80994 – PSoC3, PSoC4, and PSoC 5LP EMC Best Practices and Recommendations](#)

Appendix A: PMSM Model

8 Appendix A: PMSM Model

This appendix presents the mathematical model of a permanent magnet synchronous motor (PMSM). To simplify the model, some assumptions are made:

- The PMSM motor winding connection is the “star” type. “Delta” type connections must be converted to the “star” type.
- Magnetic saturation is neglected.
- Eddy currents and hysteresis losses are negligible.

Figure 36 illustrates the PMSM motor model in a three-phase stator reference frame, also called the (a, b, c) frame. In this frame, the a , b , and c axes are aligned with the currents i_a , i_b , i_c in the three phases of the PMSM stator, and are 120° apart from each other. Ψ_f is the flux linkage vector of the rotor magnet. The rotor rotates with an angular speed ω_r , and θ_r is the angle between Ψ_f and phase a .

The a , b , and c phases are each called “line”. The connection point of a , b , and c is called the neutral point. See **Appendix B** for information on usage of these terms.

The voltages on the stator windings are represented as:

where:

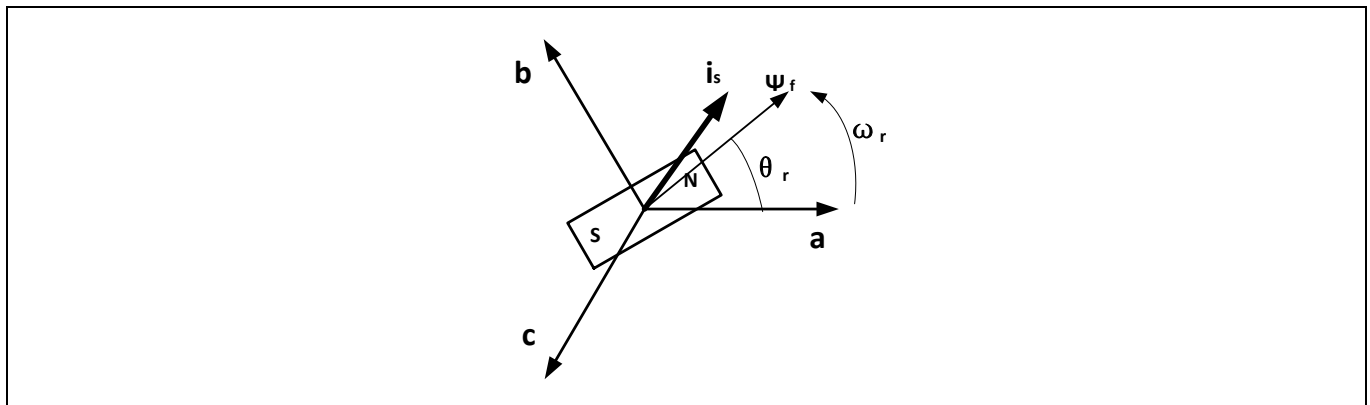


Figure 36 Three-Phase Stator Reference Frame

$$\begin{cases} u_a = R_a \times i_a + \frac{d\Psi_a}{dt} \\ u_b = R_b \times i_b + \frac{d\Psi_b}{dt} \\ u_c = R_c \times i_c + \frac{d\Psi_c}{dt} \end{cases}$$

u_a, u_b, u_c Stator voltage vector

R_a, R_b, R_c Stator resistance

i_a, i_b, i_c Stator current vector

Ψ_a, Ψ_b, Ψ_c Stator flux linkages

The stator winding flux linkage is the sum of the flux linkages from their own excitation, mutual flux linkages from other winding currents, and flux linkages from the rotor magnet. Because the current phases on the stator windings are 120° apart, the stator flux linkages are written as:

Appendix A: PMSM Model

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

where: L_{aa} , L_{bb} , L_{cc} Equivalent inductances of stator phases

M_{ab} , M_{ac} , M_{ba} , M_{bc} , M_{ca} , M_{cb} Mutual equivalent inductances of stator phases

Ψ_f Amplitude of rotor flux linkage

θ_r Angle between Ψ_f and phase a

The above model is of a high order, is strongly coupled, and has nonlinearity; analyzing it and controlling the torque and flux based on it is difficult. Therefore, the (d, q) frame is used to simplify the three-phase model. The (d, q) frame defines a rotating two-phase reference frame where the d axis is aligned with the rotor flux direction.

There are two transformations to convert the (a, b, c) frame to the (d, q) frame. The first one is a Clarke transformation – it converts the (a, b, c) frame to a two-phase stationary reference frame (α , β) (Figure 37).

The current vectors in the (α , β) frame are:

For “star” type winding connections, the sum of the currents in the three phases is zero:

Therefore, the current vectors in the (a, b, c) frame are transformed to the (α , β) frame as:

The Park transformation then converts the (α , β) frame to the (d, q) frame. The (d, q) frame has two axes – direct and quadrature – that rotate with the same angle speed ω_r as the current vector. The direct axis is aligned with the rotor flux Ψ_f (Figure 38). The angle between the d axis and the α axis is θ_r .

The current vectors in the (d, q) frame are calculated as:

The voltages in the (d, q) frame are calculated from i_d and i_q , as:

Appendix A: PMSM Model

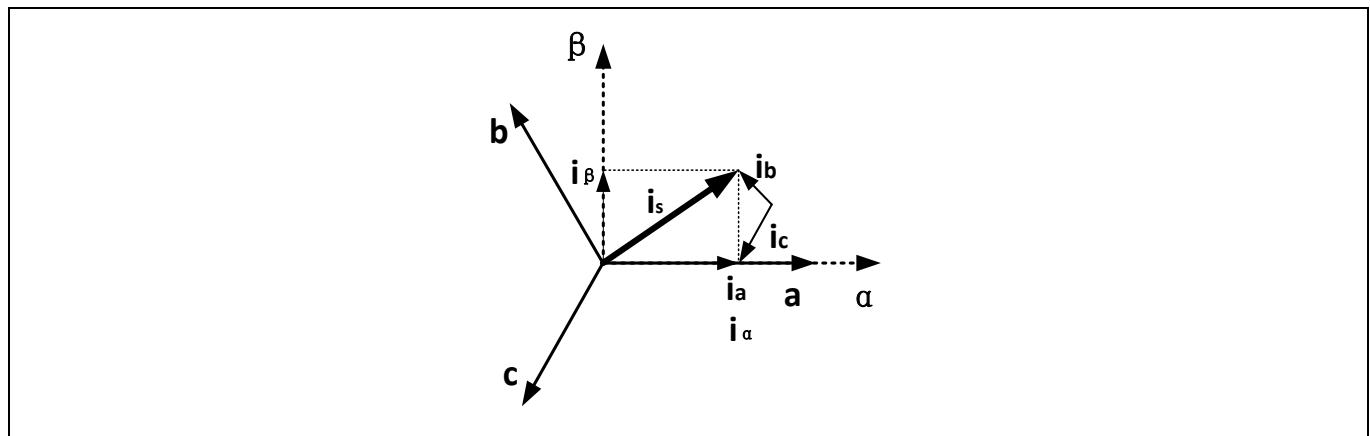


Figure 37 Clarke Transformation

$$\begin{cases} i_{\alpha} = \frac{2}{3} \times i_a - \frac{1}{3} \times i_b - \frac{1}{3} \times i_c \\ i_{\beta} = \frac{\sqrt{3}}{3} \times i_b - \frac{\sqrt{3}}{3} \times i_c \end{cases}$$

$$i_a + i_b + i_c = 0$$

$$\begin{cases} i_{\alpha} = i_a \\ i_{\beta} = \frac{\sqrt{3}}{3} \times i_a + \frac{2\sqrt{3}}{3} \times i_b \end{cases}$$

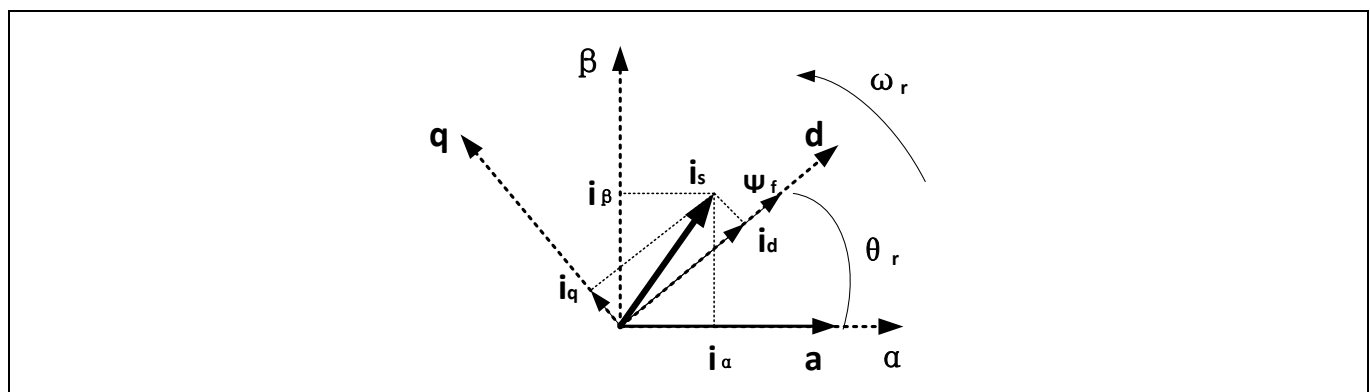


Figure 38 Park Transformation

$$\begin{cases} i_d = i_{\beta} \times \sin \theta_r + i_{\alpha} \times \cos \theta_r \\ i_q = i_{\beta} \times \cos \theta_r - i_{\alpha} \times \sin \theta_r \end{cases}$$

Appendix A: PMSM Model

$$\begin{cases} u_d = R \times i_d + \frac{d\Psi_d}{dt} - \omega_r \times \Psi_q \\ u_q = R \times i_q + \frac{d\Psi_q}{dt} + \omega_r \times \Psi_d \end{cases}$$

and:

The torque equation is expressed as:

where:

Note that for a SPMSM ([Figure 1](#)), L_q and L_d are independent of θ_r , and L_q is equal to L_d . Thus, the torque equation is simplified for SPMSM as:

$$\begin{cases} \Psi_d = L_d \times i_d + \Psi_f \\ \Psi_q = L_q \times i_q \end{cases}$$

$$T_e = \frac{3}{2} P_n [\Psi_f i_q - (L_q - L_d) i_d i_q] - T_L$$

L_d , L_q Inductances of direct and quadrature axes

P_n Number of pole pairs in rotor

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

P_n and Ψ_f are motor characteristics that are not affected by the motor rotation. Compared to the three-phase model, the torque is proportional only to the q-axis current i_q , which is easier to control.

Appendix B: Slide Mode Observer (SMO)

9 Appendix B: Slide Mode Observer (SMO)

Obtaining the position of a rotating rotor is critical for FOC. The Park transformation (see [Appendix A](#)) requires the rotor position angle θ_r between the rotor flux linkage Ψ_f and the α axis. Originally, this information came from physical sensors, such as Hall-effect sensors, optical encoders, and so on. These sensors not only increase the system cost, but they also require maintenance. Later, the sensorless technique was developed to remove the need for sensors. Some high-precision applications such as robotics still require encoders.

The idea of the sensorless technique is to estimate the angle θ_r based on the BEMF value in the (α, β) frame. The typical algorithm to do this is called a slide mode observer (SMO). In this algorithm, the two-phase voltages in the (α, β) frame is expressed as:

where:

$$\begin{cases} u_\alpha = R_s \times i_\alpha + L_s \times \frac{di_\alpha}{dt} + e_\alpha \\ u_\beta = R_s \times i_\beta + L_s \times \frac{di_\beta}{dt} + e_\beta \end{cases}$$

R_s Line-to-neutral resistance

L_s Line-to-neutral inductance

e_α, e_β BEMF on (α, β) axes

In the digital domain, the u_α equation is changed to:

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

where: T_s Period of PWM on inverter

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

We now define two new parameters that are related to motor parameters:

$$F = 1 - T_s \frac{R_s}{L_s}$$

$$G = \frac{T_s}{L_s}$$

Note that R_s and L_s are motor characteristics that can be measured. T_s is a constant system parameter, $i_\alpha(n)$ is the sampled result from the last control cycle, and $u_\alpha(n)$ is the calculation result of the last control cycle. Therefore, if given an estimated $e_\alpha^*(n)$, an estimated current value $i_\alpha^*(n+1)$ can be calculated ("*" indicates an estimated value).

Comparing $i_\alpha^*(n+1)$ with the actual current value $i_\alpha(n+1)$ sampled by the ADC, the error between these two values is used to adjust $e_\alpha^*(n)$ for a better estimation. Repeat this process until the error between $i_\alpha^*(n+1)$ and $i_\alpha(n+1)$ is small enough to meet the design requirements. Then, the estimated $e_\alpha^*(n)$ can represent the actual BEMF $e_\alpha(n)$. The $e_\beta(n)$ is obtained in the same manner.

$$\text{Because } e_\alpha(n) \text{ and } e_\beta(n) \text{ are expressed as: } \begin{cases} e_\alpha(n) = -\Psi_f \times \omega \times \sin \theta \\ e_\beta(n) = \Psi_f \times \omega \times \cos \theta \end{cases}$$

Appendix B: Slide Mode Observer (SMO)

The angle θ is calculated as: $\theta(n) = \arctan \frac{-e_{\alpha}(n)}{e_{\beta}(n)}$

The angular speed ω_r is calculated by accumulating θ over m samples and multiplied by the speed constant K :

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

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

Appendix C: SVPWM Theory

10 Appendix C: SVPWM Theory

In **Figure 3**, Q1, Q3, and Q5 are the upper MOSFETs of the inverter. If you consider the MOSFET ON state as “1” and the OFF state as “0”, there are eight combinations of ON/OFF states, which lead to eight inverter outputs.

Table 4 lists the ON/OFF state combinations and their corresponding inverter outputs. u_a , u_b , and u_c are the phase (line-to-neutral) voltages (see **Appendix A**), while u_{ab} , u_{bc} , and u_{ac} are the line-to-line voltages. The values in each cell indicate the voltage as a percentage of the bus voltage, V_{bus} . For example, 2/3 means 2/3 of V_{bus} .

Table 4 Output Combination in Three-Phase Frame

Q1 (A)	Q3 (B)	Q5 (C)	u_a	u_b	u_c	u_{ab}	u_{bc}	u_{ca}	
1	0	0	2/3	-1/3	-1/3	1	0	-1	U_0
1	1	0	1/3	1/3	-2/3	0	1	-1	U_{60}
0	1	0	-1/3	2/3	-1/3	-1	1	0	U_{120}
0	1	1	-2/3	1/3	1/3	-1	0	1	U_{180}
0	0	1	-1/3	-1/3	2/3	0	-1	1	U_{240}
1	0	1	1/3	-2/3	1/3	1	-1	0	U_{300}
0	0	0	0	0	0	0	0	0	0_{000}
1	1	1	0	0	0	0	0	0	0_{111}

The eight combinations can be considered as six non-zero vectors and two zero vectors (000 and 111). As **Figure 39** shows, the non-zero vectors are the axes of a hexagon; the angle between any two adjacent axes is 60 degrees. This divides the hexagon into six sectors (Roman numerals I to VI). The zero vectors are at the origin, and they generate zero voltage on the three phases. These eight vectors, called “basic space vectors,” are called U_0 , U_{60} , U_{120} , U_{180} , U_{240} , U_{300} , 0_{000} , and 0_{111} . A voltage vector is synthesized by one or two of the six nonzero basic space vectors.

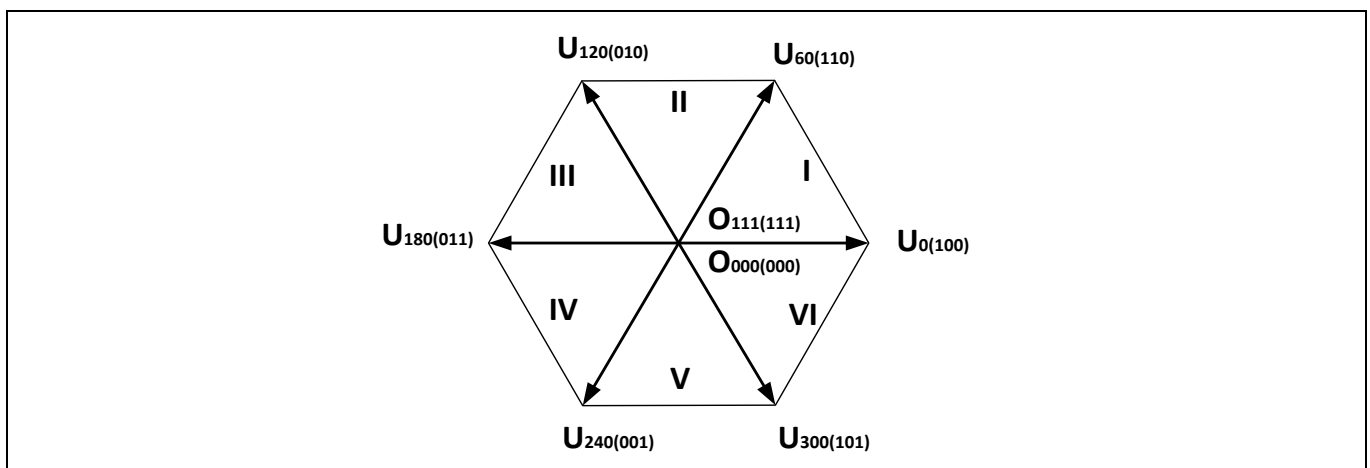


Figure 39 Basic Space Vectors

For example, as **Figure 40** shows, the voltage vector $\vec{U_s}$ is in sector I, and the period of the PWM is T . T_1 is the duration of U_0 ; T_2 is the duration of U_{60} ; and T_0 is the duration of the two zero vectors. The vectors $\vec{u_\alpha}$ and $\vec{u_\beta}$ compose a voltage vector, $\vec{U_s}$, that can also be composed by basic space vectors U_0 and U_{60} .

Appendix C: SVPWM Theory

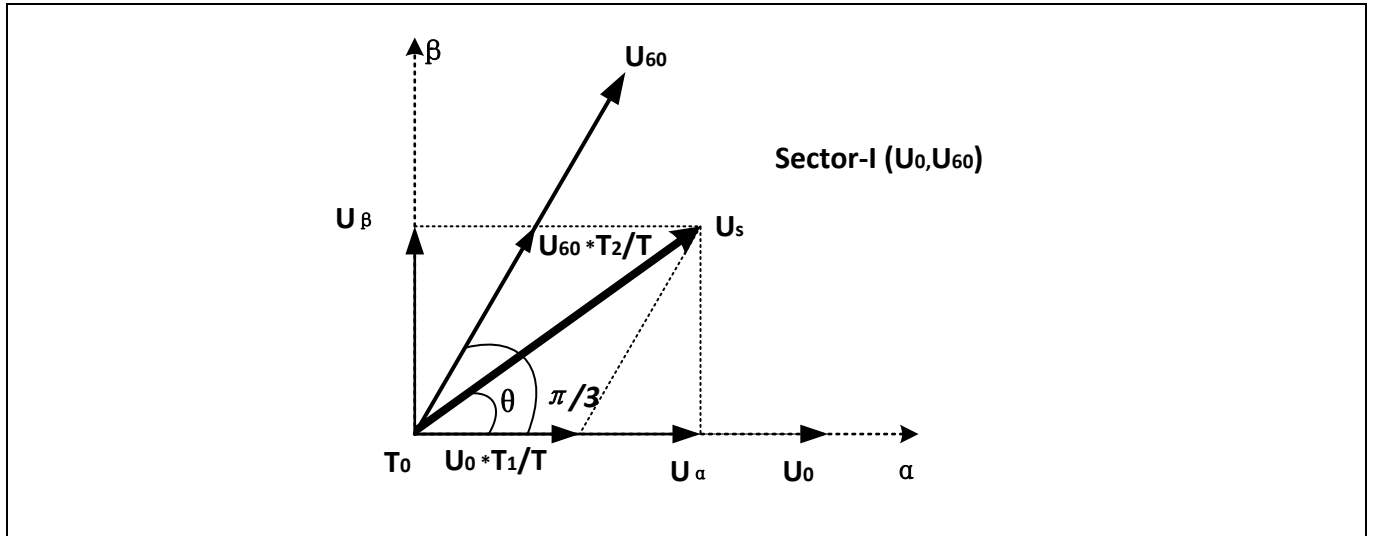


Figure 40 Voltage Vector in Sector I

\vec{U}_s can be expressed as:

$$T = T_1 + T_2 + T_0$$

$$\vec{U}_s = \vec{U}_{60} \times \frac{T_2}{T} + \vec{U}_0 \times \frac{T_1}{T}$$

Therefore:

$$|U_s| \cos \theta = |U_{60}| \times \frac{T_2}{T} \times \cos \frac{\pi}{3} + |U_0| \times \frac{T_1}{T}$$

$$|U_s| \sin \theta = |U_{60}| \times \frac{T_2}{T} \times \sin \frac{\pi}{3}$$

Then:

$$T_1 = mT \sin \left(\frac{\pi}{3} - \theta \right)$$

$$T_2 = mT \sin \theta$$

$$T_0 = T - T_1 - T_2 \quad (T_0 \geq 0)$$

Where:

$$m = \sqrt{3} \times \frac{|U_{out}|}{U_{dc}}$$

$$|U_{out}| = \sqrt{|u_\alpha|^2 + |u_\beta|^2}$$

Note that all basic space vectors are generated with a specific ON/OFF state of upper MOSFETs; the duration is actually the time of the PWM being high, or the duty cycle. Thus, generating a \vec{U}_s is related to a change in duty cycle of the PWMs applied to the inverter. In this example, both \vec{U}_0 and \vec{U}_{60} require phase A to be turned ON, and \vec{U}_{60} requires phase B to be turned ON. Therefore:

$$Duty_A = \frac{T_1 + T_2}{T}, \quad T_1 + T_2 \leq T$$

Appendix C: SVPWM Theory

$$Duty_B = \frac{T_2}{T}, \quad T_2 \leq T$$

$$Duty_0 = \frac{T - T_1 - T_2}{T}$$

Depending on how you use zero vectors, the SVPWM has two output patterns: a five-phase pattern and a seven-phase pattern. The five-phase pattern uses only 0_{000} or 0_{111} . The seven-phase pattern uses both 0_{000} and 0_{111} , and their durations are equal. [Figure 41](#) illustrates these two patterns. Note that in 5-phase SVPWM, phase A is always on or always off.

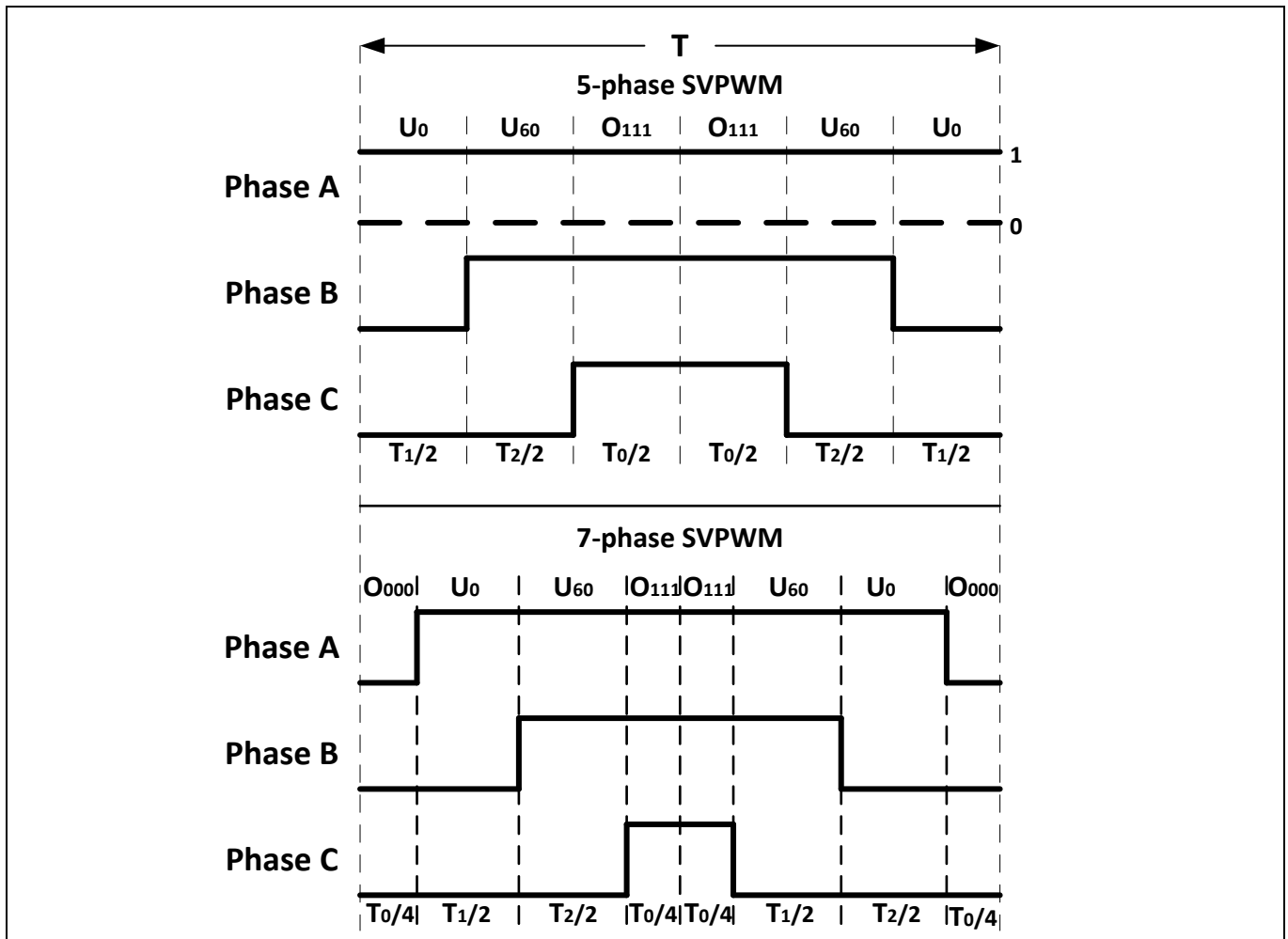


Figure 41 Five- and Seven-Phase SVPWM in Sector I

There is no difference in the synthesized voltage vector generated by these two methods. However, the five-phase pattern reduces the number of MOSFETs that are switching. This can reduce the switching losses in the power components, but it creates more harmonics than the seven-phase pattern.

Appendix D: Q Number Format (Fixed-Point Number)

11 Appendix D: Q Number Format (Fixed-Point Number)

The Q number format is a well-known method to store and process floating-point numbers. It enables faster floating-point operations done by the CPU, so that a separate floating-point unit is not needed. However, some accuracy may be lost by using floating-point.

The example project provided in this application note uses the Q number format. Although understanding the Q number format is not mandatory, gaining a fundamental knowledge of it will help you master the example code faster.

An introduction to the Q number format can be found in Wikipedia. This appendix contains a copy of the “Q (number format)” page from the Wikipedia site, if you are not able to connect to the Internet but need to know about the Q number format when reading this application note.

Note: The following content is from Wikipedia. Cypress does not maintain this content for accuracy, nor guarantee that it is up to date. If you have access to the Internet, go to the Wikipedia website to read the latest version by clicking the following link or entering it in your browser.

This material from Wikipedia is reproduced under the Creative Commons Attribution-ShareAlike 3.0 Unported License, which you can view at the following URL: <http://creativecommons.org/licenses/by-sa/3.0/>. For more information, please see Wikipedia’s licensing statement at http://en.wikipedia.org/wiki/Wikipedia:Text_of_Creative_Commons_Attribution-ShareAlike_3.0_Unported_License. Your rights to this Wikipedia material are governed by the foregoing license.

From Wikipedia, the free encyclopedia:

Q (number format) on Wikipedia: http://en.wikipedia.org/wiki/Q_%28number_format%29

Q is a **fixed point** number format where the number of **fractional bits** (and optionally the number of **integer bits**) is specified. For example, a Q15 number has 15 fractional bits; a Q1.14 number has 1 integer bit and 14 fractional bits. Q format is often used in hardware that does not have a floating-point unit and in applications that require **constant resolution**.

11.1 Characteristics

Q format numbers are (*notionally*) fixed point numbers (but not actually a number itself); that is, they are stored and operated upon as regular binary numbers (i.e. signed integers), thus allowing standard integer hardware/**ALU** to perform **rational number** calculations. The number of integer bits, fractional bits and the underlying word size are to be chosen by the programmer on an application-specific basis—the programmer's choices of the foregoing will depend on the range and resolution needed for the numbers.

Some DSP architectures offer native support for common formats, such as Q1.15. In this case, the processor can support arithmetic in one step, offering saturation (for addition and subtraction) and renormalization (for multiplication) in a single instruction. Most standard CPUs do not. If the architecture does not directly support the particular fixed point format chosen, the programmer will need to handle saturation and renormalization explicitly with bounds checking and bit shifting.

There are 2 conflicting notations for fixed point. Both notations are written as $Q_{m,n}$, where:

- Q designates that the number is in the Q format notation—the “Q” being reminiscent of the standard symbol for the set of **rational numbers**.

Appendix D: Q Number Format (Fixed-Point Number)

- m . (optional, assumed to be zero or one) is the number of bits set aside to designate the two's complement integer portion of the number, exclusive or inclusive of the sign bit (therefore if m is not specified it is taken as zero or one).
- n is the number of bits used to designate the fractional portion of the number, i.e. the number of bits to the right of the binary point. (If $n = 0$, the Q numbers are integers—the degenerate case).

One convention includes the sign bit in the value of m , and the other convention does not. The choice of convention can be determined by summing $m+n$. If the value is equal to the register size, then the sign bit is included in the value of m . If it is one less than the register size, the sign bit is not included in the value of m .

In addition, the letter U can be prefixed to the Q to indicate an unsigned value, such as UQ1.15, indicating values from 0.0 to +1.99997.

Signed Q values are stored in 2's complement format, just like signed integer values on most processors. In 2's complement, the sign bit is extended to the register size.

For a given $Qm.n$ format, using an $m+n+1$ bit signed integer container with n fractional bits:

- its range is $[-(2^m), 2^m - 2^{-n}]$
- its resolution is 2^{-n}

For a given UQ $m.n$ format, using an $m+n$ bit unsigned integer container with n fractional bits:

- its range is $[0, 2^m - 2^{-n}]$
- its resolution is 2^{-n}

For example, a Q14.1 format number:

- requires $14+1+1 = 16$ bits
- its range is $[-2^{14}, 2^{14} - 2^{-1}] = [-16384.0, +16383.5] = [0x8000, 0x8001 \dots 0xFFFF, 0x0000, 0x0001 \dots 0x7FFE, 0x7FFF]$
- its resolution is $2^{-1} = 0.5$

Unlike **floating point** numbers, the resolution of Q numbers will remain constant over the entire range.

11.2 Conversion

11.2.1 Float to Q

To convert a number from **floating point** to $Qm.n$ format:

1. Multiply the floating point number by 2^n
2. Round to the nearest integer

11.2.2 Q to float

To convert a number from $Qm.n$ format to floating point:

1. Convert the number to floating point as if it were an integer
2. Multiply by 2^{-n}

11.3 Math Operations

Q numbers are a ratio of two integers: the numerator is kept in storage, the denominator is equal to 2^n .

Appendix D: Q Number Format (Fixed-Point Number)

Consider the following example:

The Q8 denominator equals $2^8 = 256$

1.5 equals $384/256$

384 is stored, 256 is inferred because it is a Q8 number.

If the Q number's base is to be maintained (n remains constant) the Q number math operations must keep the denominator constant. The following formulas shows math operations on the general Q numbers N_1 and N_2 .

$$\begin{aligned}\frac{N_1}{d} + \frac{N_2}{d} &= \frac{N_1 + N_2}{d} \\ \frac{N_1}{d} - \frac{N_2}{d} &= \frac{N_1 - N_2}{d} \\ \left(\frac{N_1}{d} \times \frac{N_2}{d}\right) \times d &= \frac{N_1 \times N_2}{d} \\ \left(\frac{N_1}{d} / \frac{N_2}{d}\right) / d &= \frac{N_1 / N_2}{d}\end{aligned}$$

Because the denominator is a power of two the multiplication can be implemented as an arithmetic shift to the left and the division as an arithmetic shift to the right; on many processors shifts are faster than multiplication and division.

To maintain accuracy the intermediate multiplication and division results must be double precision and care must be taken in rounding the intermediate result before converting back to the desired Q number.

Using C the operations are (note that here, Q refers to the fractional part's number of bits):

11.3.1 Addition

```
signed int a, b, result;
result = a + b;
```

With saturation

```
signed int a, b, result;
signed long int tmp;
tmp = a + b;
if (tmp > 0x7FFFFFFF) tmp = 0x7FFFFFFF;
if (tmp < -1 * 0x7FFFFFFF) tmp = -1 * 0x7FFFFFFF;
result = (signed int) tmp;
```

11.3.2 Subtraction

```
signed int a, b, result;
result = a - b;
```

11.3.3 Multiplication

```
// precomputed value:
#define K (1 << (Q-1))

signed int      a, b, result;
signed long int temp;
```

Appendix D: Q Number Format (Fixed-Point Number)

```
temp = (long int)a * (long int)b; // result type is operand's type
// Rounding; mid values are rounded up
temp += K;
// Correct by dividing by base
result = temp >> Q;
```

11.3.4 Division

```
signed int a, b, result;
signed long int temp;
// pre-multiply by the base (Upscale to Q16 so that the result will be in
Q8 format)
temp = (long int)a << Q;
// So the result will be rounded ; mid values are rounded up.
temp += b/2;
result = temp/b;
```

Text is available under the [Creative Commons Attribution-ShareAlike License](https://creativecommons.org/licenses/by-sa/4.0/).

Appendix E: Adapting the Design to Other Motors

12 Appendix E: Adapting the Design to Other Motors

This appendix helps you to drive other motors with the code example provided with this application note. You should follow the operation guide step by step. A **bold** font indicates a **mandatory** action or **critical** information that requires more attention.

Hardware: CY8CKIT-037 or your own motor driver board

Firmware: Sensorless FOC project from the latest version of this application note

Equipment: Oscilloscope, multimeter, PC, USB cable for CY8CKIT-045S or MiniProg3 for programming your own board

12.1 Operation Guide

12.1.1 Check the power range and motor type.

12.1.1.1 Power range

CY8CKIT-037 supports a **12-V to 48-VDC** supply voltage with up to **2 A** input DC current. You should use the kit **only** in this power range; using the kit out of this power range may damage it.

12.1.1.2 Motor type

A motor with **sinusoidal back electromotive force (BEMF)** is recommended. A motor with trapezoidal BEMF may not rotate or achieve the desired performance with the sensorless FOC project. [Figure 42](#) illustrates these two BEMF types. To measure BEMF, connect the ground of the oscilloscope probe to one motor phase and the probe to another motor phase. Leave the other motor phases floating. Rotate the motor either by hand or by using another motor. You should see the BEMF waveform on the oscilloscope.

The sinusoidal BEMF contains the complete angle information, which can be calculated with the SMO algorithm. The trapezoidal BEMF is almost flat at the wave crest and trough and therefore is missing sufficient angle information. As a result, the SMO algorithm cannot reliably retrieve the angle from this waveform, which may halt the motor rotation.

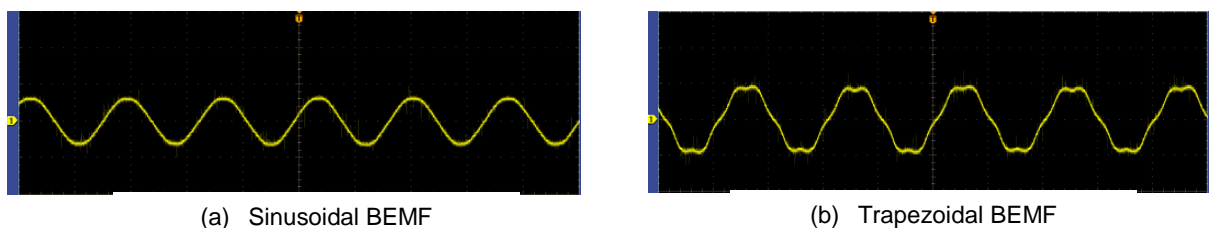


Figure 42 Sinusoidal BEMF Versus Trapezoidal BEMF

12.1.2 Change the parameters in the example project.

12.1.2.1 Change the parameters for the motor.

These parameters are defined as global variables in `h03_user\customer_interface.c`. You should change them based on your motor specifications.

```
int32_t Motor_i32PolePairs = 4; // the pole pairs of rotor
```

Appendix E: Adapting the Design to Other Motors

```
float32_t Motor_f32Ld          = 0.06;    // the d axis reductance,unit:mh
float32_t Motor_f32Lq          = 0.06;    // the q axis reductance,unit:mh
float32_t Motor_f32Res          = 1.1;     // the resistance between two
phases
```

12.1.2.2 Change the macro definitions for the system parameters.

These macro definitions are related to system parameters, such as the sampling resistor and so on. You should change them (*h03_user\hardware_config.h*) if the default values are different from your system.

```
#define SYS_VDC_FACTOR          20.1      //DC voltage sample resistor factor
#define MOTOR_SHUNT_NUM         2         // The number of shunt used to sense
current
#define Motor_f32IuvwSampleResistor 0.03 // Iuvw sample resistor (ohm)
#define Motor_i32IuvwAmplifierFactor 4.16 // Iuvw calculation factor
#define ADC_VOLT_REF            5.0f      // Reference voltage for ADC
#define ADC_VALUE_MAX           4096.0f   // 12-bits ADC max value
```

12.1.2.3 Change the parameters for the PI controllers.

You may need to change the PI coefficient parameters in the PI controller if the PI controller does not work well with your motor. You can change the parameters in *h03_user\customer_interface.c*. For more details, see [Appendix F: Tunable Parameters List](#).

12.1.3 Set up the hardware.

If you are using the CY8CKIT-037 kit, you can use the adapter provided with the kit for any motor whose maximum power is 24 V DC / 2.1 A. If a different voltage (such as 48 V) or current (such as 3 A) is required, connect the DC voltage source to the J8 connector (yellow marker in [Figure 43](#)) instead of the supplied power adapter.

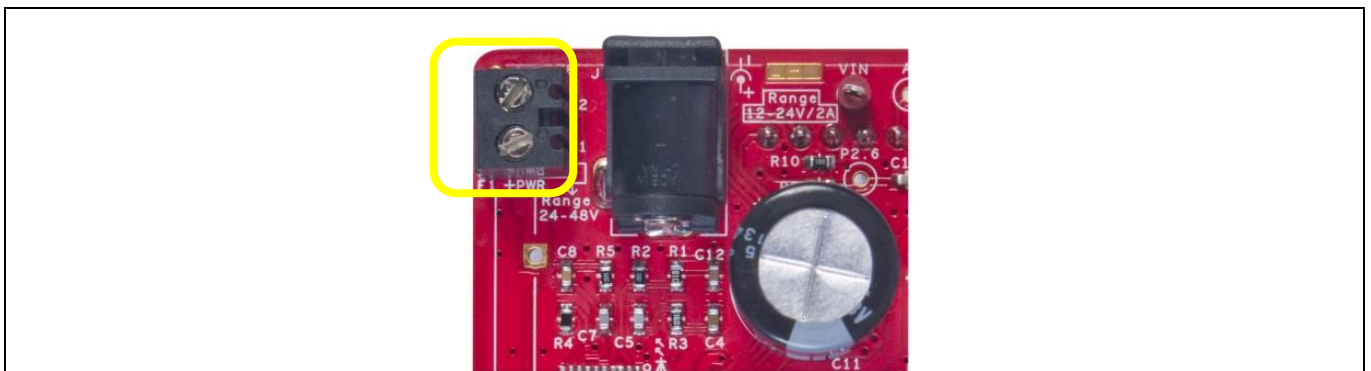


Figure 43 Connector for DC Power Input

Appendix E: Adapting the Design to Other Motors

12.1.4 Program the CY8CKIT-045S and observe the performance.

12.1.5 Tune the parameters if the motor does not rotate

The motor starts up in open-loop control and then switches to closed-loop speed control later. If switching to the closed loop control fails (motor halts very soon after the rotation starts), you may need to tune the below parameters. Try the following methods:

- Confirm that the motor parameters are set correctly in [12.1.2.1](#).
- Change the parameters switch from open loop to closed-loop in *h03_user\customer_interface.c*.

```
uint16_t Motor_u16OpenLoopSpdInitHz    = 5;      //open loop start speed
uint16_t Motor_u16OpenLoopSpdEndHz     = 10;     //open loop end speed
uint16_t Motor_u16OpenLoopSpdIncHz     = 10;     //acceleration speed of open
loop
uint16_t Motor_u16ColseLoopTargetSpdHz = 10;     //target speed when
switching to close loop
```

- Debug with the Back-EMF low pass filter factors in SMO structure *Motor_stcSMO*.
- If the error occurs when motor is running, the motor will stop immediately. Check the variable *MotorCtrl_stcRunPar.u32ErroType* to find the error. If the error is over/under voltage, confirm the parameters set in [12.1.2.2](#). You can clear the error by rotating the potentiometer to a smallest value. If the error occurs more than 10 times, it cannot be cleared, and you should reset the board.
- When motor is running, the LED2 will blink according to the motor's speed. If motor's speed goes high, the LED2 will blink more frequent.
- When motor is running, press SW2 will change the running direction of motor.

If the motor rotates with a vibration, try tuning the Kp and Ki parameters in the PI controller. The larger the Kp value, the faster the system closes in on the reference value; however, it may make the system unstable. The Ki value can reduce the static error and make the system stable; however, a larger Ki may make the integration value saturate.

Appendix F: Tunable Parameters List

13 Appendix F: Tunable Parameters List

13.1 Hardware Parameter Setting

The hardware parameters should be set according to the kit. If you have your own inverter board, change the parameters mentioned in [Table 5](#) in the `h03_user\hardware_config.h` file.

Table 5 Hardware Parameter Setting

No.	Macro	Description	Value
1	SYS_VDC_FACTOR	DC voltage sample resistor factor	20.1
2	MOTOR_SHUNT_NUM	Number of shunts used to sense current	2
3	ADC_VOLT_REF	AD reference voltage	3.3V
4	ADC_VALUE_MAX	AD accuracy set, 12-bit AD is set to '0xFFF'	4096
5	COMP_ADC_CH_IU	ADC channel for U phase current	0
6	COMP_ADC_CH_IW	ADC channel for W phase current	1
7	SYS_ADC_CH_VDC	ADC channel for VBUS	2
8	MOTOR_SPEED_VR	ADC channel for	3
9	Motor_f32IuvwSampleResistor	Iuvw sample resistor (ohm)	0.03ohm
10	Motor_i32IuvwAmplifierFactor	Iuvw calculation factor	4.16

Especially, in the [Table 5](#),

- SYS_VDC_FACTOR: The factor for calculating Vbus, which is determined by the circuit
- $SYS_VDC_FACTOR = (R9 + R10) / R10$.

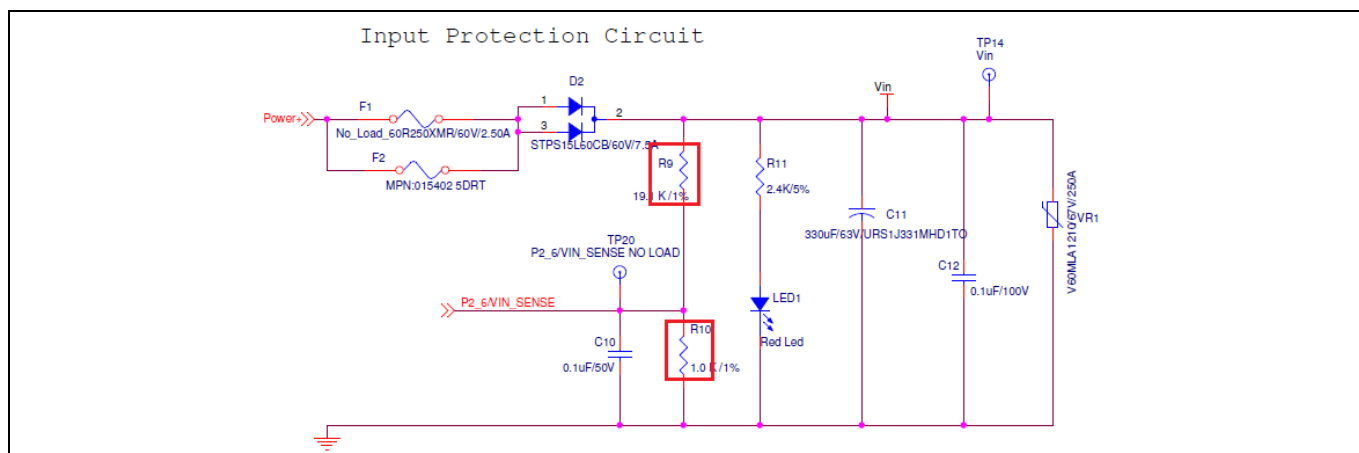


Figure 44 Power Circuit

- MOTOR_SHUNT_NUM: The number of shunts used to sense current, which is dependent on your circuit of motor current detection.
- ADC_VOLT_REF: The ADC sampling reference voltage of system. If you choose 5 V in [Figure 13](#) or use 5 V in your own systems, change this parameter to 5.0 and set it in PSoC Creator as shown in [Figure 45](#).

Appendix F: Tunable Parameters List

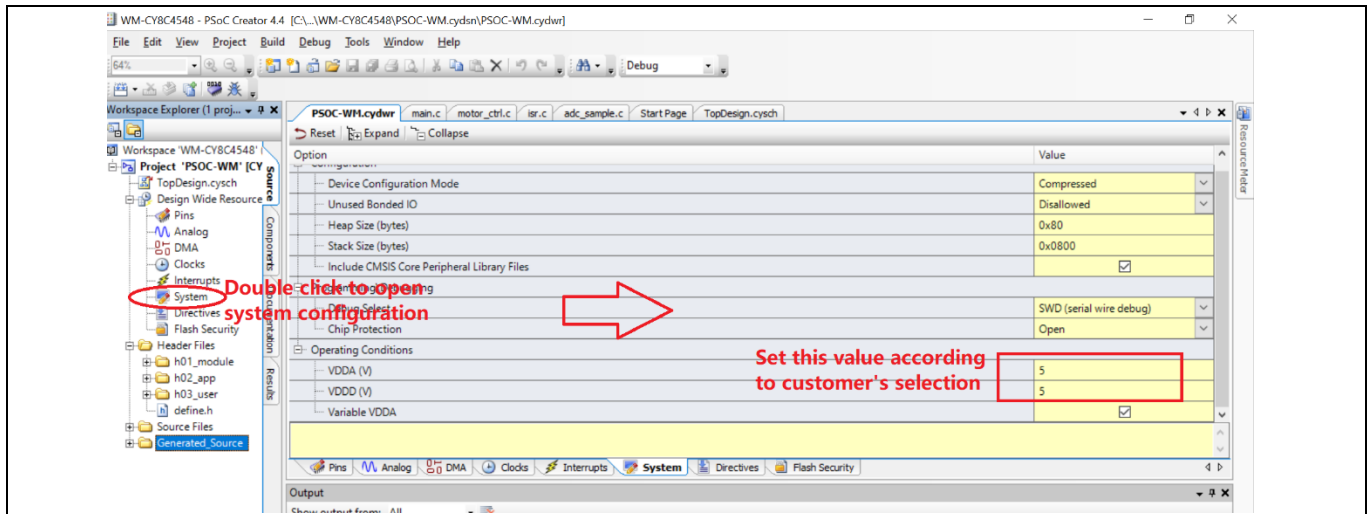


Figure 45 Setting System Voltage

- **ADC_VALUE_MAX:** Depends on the accuracy of ADC; the accuracy of the internal ADC is 12-bits; thus the maximum ADC value is 4096. You need to change the value according to your own schematic.
- **COMP_ADC_CH_IU:** ADC channel number for motor U phase current sense, that is channel 0 of ADC1.
- **COMP_ADC_CH_IW:** ADC channel number for motor W phase current sense, that is channel 1 of ADC1.
- **SYS_ADC_CH_VDC:** ADC channel number for Bus voltage sense, that is channel 0 of ADC.
- **MOTOR_SPEED_VR:** ADC channel number for potentiometer input sense, that is channel 2 of ADC1.

The four parameters (**COMP_ADC_CH_IU**, **COMP_ADC_CH_IW**, **SYS_ADC_CH_VDC**, **MOTOR_SPEED_VR**) are set by the top-design, and the motor phase current sense is according to the circuit of current detection. In CY8CKIT-037 the circuit detects the current of U and W phase. If the order ADC channels in **Figure 46** is changed, for example, if **OP_Ia_Vout_Filt** and **VR-In** are interchanged, then the parameter **COMP_ADC_CH_IU** should set to 3, and parameter **MOTOR_SPEED_VR** should set to 0.

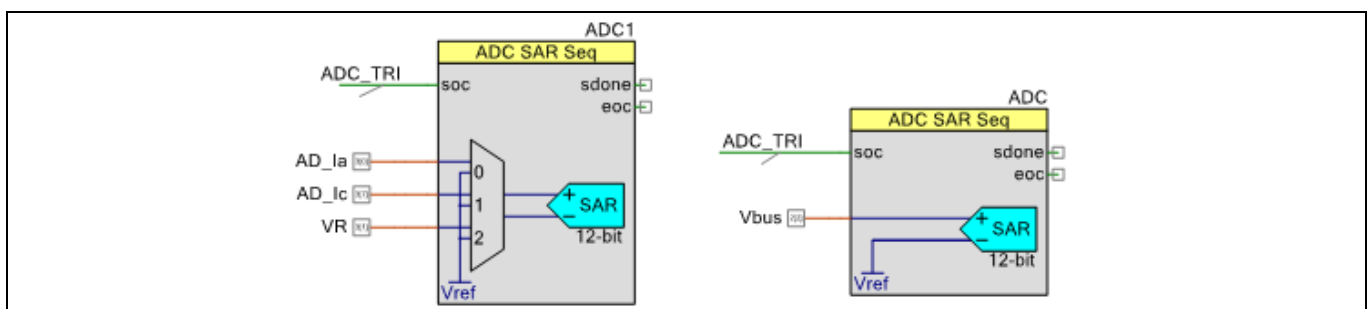


Figure 46 ADC Channel Number Set

- **Motor_f32IuvwSampleResistor:** The value of sample resistor in current detection circuit.
- **Motor_i32IuvwAmplifierFactor:** The factor of amplification multiples in amplification circuit.

13.2 Firmware Parameter Setting

The firmware parameters are defined for motor running. The firmware parameters include motor parameters, motor carry frequency, PI parameters, motor start-up parameters, and so on, which are in the `s03_user\customer_interface.c` file.

Appendix F: Tunable Parameters List

13.3 Motor Parameters

The motor parameters include the motor pole pairs, phase current, and phase inductance. [Table 6](#) lists the details of these parameters.

Table 6 Motor Parameters Setting

No.	Variable Name	Description
1	Motor_i32PolePairs	Motor's Pole pairs
2	Motor_f32Ld	Phase inductance of d axis, unit: mH
3	Motor_f32Lq	Phase inductance of q axis, unit: mH
4	Motor_f32Res	Resistance between two phases, unit: ohm

The motor parameters are dependent on the motor that you choose.

The motor pole pair is usually labeled in the motor nameplate. The phase inductance of d/q axis and the phase resistor can be detected by the RLC measuring instrument.

13.4 ADC Sampling Parameters

These parameters are defined for ADC sampling. The value of sample resistor is related to the circuit. [Table 7](#) lists the details of these parameters.

Table 7 ADC Sampling Parameters Setting

No.	Variable Name	Description
1	Motor_i32luvwOffsetNormal	The middle value of 12-bits ADC: $4096/2=2048$
2	Motor_i32luvwOffsetRange	ADC offset range of luvw sampling. If the error of the ADC checked value is out of this range, system will warn the AD_MIDDLE_ERROR fault.
3	Motor_i32luvwOffsetCheckTimes	luvw ADC sample offset check times
4	Motor_f32DeadTimeMicroSec	Dead time (us) of PWM
5	Motor_u16CarrierFreq	Motor carry frequency (Hz)

- **Motor_i32luvwOffsetNormal:** The middle value of 12-bits ADC. For example: if your system is 5 VDDA, then the maximum ADC input is 5 V and the normal offset value is 2.5 V. Thus, the ADC normal offset output is 2048.
- **Motor_i32luvwOffsetRange:** The range of current offset check. If the offset check result is out of this range, system will warn the AD_MIDDLE_ERROR fault. Do not set a higher value for this parameter as the motor current will fluctuate a lot if there is something wrong with the current detection circuit. This parameter can be set to a value of 150~200.
- **Motor_i32luvwOffsetCheckTimes:** luvw ADC sample offset check times. The offset check result is an average value of the sum of those check values. You can set this value based on your requirement. However, the value should not exceed 256.
- **Motor_f32DeadTimeMicroSec:** The dead time of PWM express in us. This parameter is set according to the inverter circuit or the IPM blocks that you use.
- **Motor_u16CarrierFreq:** This parameter should be set based on the MCU and the FOC execute time. You should set it according to your own MCU and load (motor). But, if this parameter is set to a higher value, the inverter service life will be reduced.

Appendix F: Tunable Parameters List

13.5 PI Regulator Parameters

The PI regulator parameters are listed in [Table 8](#).

Table 8 PI Regulator Parameters Setting

No.	Variable Name	Description
1	Motor_f32Dki	d axis current PI regulator integral constant
2	Motor_f32Dkp	d axis current PI regulator proportion constant
3	Motor_f32Qki	q axis current PI regulator integral constant
4	Motor_f32Qkp	q axis current PI regulator proportion constant
5	Motor_f32LowSpdKi	Speed PI regulator integral constant at low speed
6	Motor_f32LowSpdKp	Speed PI regulator proportion constant at low speed
7	Motor_f32Ski	Speed PI regulator integral constant at high speed
8	Motor_f32Skp	Speed PI regulator proportion constant at high speed
9	Motor_u16ChgPiSpdHz	PI parameters change at this speed

These parameters is set for current and speed PI loop. You should change the values according to your own motor and prior experience.

13.6 Start-up Parameters

The motor start up parameters are shown in [Table 9](#).

Table 9 Motor Start up Parameters Setting

No.	Variable Name	Description
1	Motor_u8RunLevel	Motor run stage: 1 → orientation, 2 → open loop running, 3 → closed loop running, 4 → change speed enable
2	Motor_i16Q8_OrientEndIqRef	Orientation current when motor in orient stage, unit: A
3	Motor_i16Q8_OrientInitIqRef	Orientation start current, unit: A
4	Motor_f32OrientIqrefIncAPS	Reference vary step in orient stage
5	Motor_f32OrientTime	Orientation time, unit: s
6	Motor_u16OpenLoopSpdInitHz	Open loop start speed, unit: Hz
7	Motor_u16OpenLoopSpdEndHz	Open loop end speed, this value should be the same with the speed when motor change to closed loop, unit: Hz
8	Motor_u16OpenLoopSpdIncHz	Acceleration speed of open loop, unit: Hz
9	Motor_i16Q8_OpenLoopInitIqRef	q axis current reference in open loop, unit: A
10	Motor_i16Q8_OpenLoopEndIqRef	q axis current reference in open loop, unit: A
11	Motor_f32OpenLoopIqrefIncAPS	q axis current reference vary step in open loop

- Motor_u8RunLevel: This parameter is to set the motor running stage. Set this parameter to 4 if you need to change the speed while the motor is running.
- Motor_i16Q8_OrientEndIqRef, Motor_i16Q8_OrientInitIqRef, Motor_f32OrientIqrefIncAPS, and Motor_f32OrientTime: These parameters are explained in [Figure 47](#).

Appendix F: Tunable Parameters List

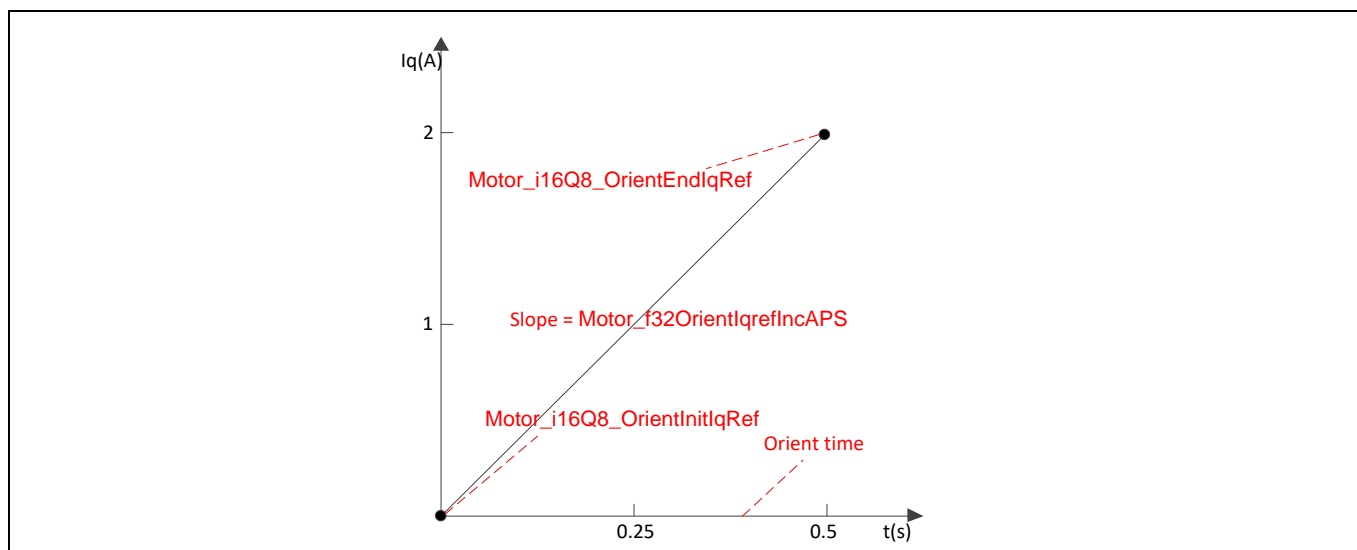


Figure 47 q-axis Current Set in Orient Stage

- Parameters 6 to 11 are similar to the parameters shown in [Figure 47](#).
- The values of parameters Motor_i16Q8_OpenLoopInitIqRef and Motor_i16Q8_OpenLoopEndIqRef should be the same as parameter 2.

13.7 Close Loop Running Parameters

The parameters when motor enter the closed loop stage are defined in [Table 10](#), mainly includes the target speed when motor switch to closed loop from open loop; the max and min speed, the acceleration speed when motor is running.

Table 10 Close Loop Running Parameters Setting

No.	Variable Name	Description
1	Motor_u16ColseLoopTargetSpdHz	Target speed when switching to close loop, unit: Hz
2	Motor_u8RunningDirection	Motor run direction 0: CW, 1: CCW
3	Motor_i16Q8_CloseLoopsMax	Maximum torque current when motor running, unit: A
4	Motor_i16Q8_CloseLoopIqRefMax	Maximum value of q axis current reference in closed loop, unit: A
5	Motor_u16SpdMax	Motor run maximum speed rpm, unit: rpm
6	Motor_u16SpdMin	Motor run minimum speed rpm, unit: rpm
7	Motor_f32SpdAccelerationHz	Acceleration speed, unit: Hz
8	Motor_f32SpdDecelerationHz	Deceleration speed, unit: Hz

- Motor_u16ColseLoopTargetSpdHz: When the motor reaches this speed, it will switch to close loop stage
- Motor_u8RunningDirection: This parameter determines the rotate direction of the motor when you first start the motor. If the direction of rotation does not suit the situation, change it to counter clock wise.
- Motor_i16Q8_CloseLoopsMax and Motor_i16Q8_CloseLoopIqRefMax: These parameters limit the maximum current when motor is running.
- Motor_u16SpdMax and Motor_u16SpdMin: These parameters limit the speed of the motor. The values are set according to the motor's rated speed.

Appendix F: Tunable Parameters List

- Motor_f32SpdAccelerationHz, Motor_f32SpdDecelerationHz are set for the acceleration/deceleration speed when motor's speed is changed. This value should not set too large. You can set it according to your needs.

13.8 Protection Parameters

The protection parameters are defined in [Table 11](#).

Table 11 Protection Parameters Setting

t	Variable Name	Description
1	Motor_i16Q8_CurrentMax	Motor phase current peak, unit: A
2	Motor_u16VbusMax	The maximum value of DC voltage, unit: V
3	Motor_u16VbusMin	The minimum value of DC voltage, unit: V

- Motor_i16Q8_CurrentMax: This parameter specifies the peak of motor phase current when motor is running. If the motor current exceeds this value, system will enter software over current protection process, and the MotorCtrl_stcRunPar.u32ErrType will be set to SW_OVER_CURRENT fault.
- Motor_u16VbusMax/ Motor_u16VbusMin: This parameter specifies maximum/minimum value of Bus voltage. If the Bus voltage that ADC sampled is out of this range, system will enter voltage protection process, and the MotorCtrl_stcRunPar.u32ErrType will be set to OVER_VOLTAGE / UNDER_VOLTAGE fault.

13.9 Other global Parameters

Table 12 Other Global Parameters

Variable's Name in Project	Structure Member	Comments
Name: MotorCtrl_stcIqPidReg Type: stc_pid_t Location: motor_ctrl.h Comments: PID controller for Iq	int32_t i32Q15_kp	p coefficient for pid calculate
	int32_t i32Q15_ki	i coefficient for pid calculate
	int32_t i32Q15_kd	d coefficient for pid calculate
	int32_t I_cnt	counter for I Out calculate
	int32_t I_timer	cycle for I Out calculate
	int32_t i32_Pout	Output: Item P
	int32_t i32_Iout	Output: Item I
	int32_t i32_Dout	Output: Item D
	int32_t i32_Out	Output: pid regulator
	int32_t i32_OutPre	Last Output: pid regulator
	int32_t i32QN_Iout	Output: Item I QN format
	int32_t i32_OutMax	Output upper limitation
	int32_t i32_OutMin	Output lower limitation
	int32_t i32_E0	input-error
	int32_t i32_E1	last input-error
	int32_t i32_E0Max	input-error max limitation
	int32_t i32_E0Min	input-error min limitation

Appendix F: Tunable Parameters List

Variable's Name in Project	Structure Member	Comments
Name: MotorCtrl_stcIdPidReg Type: stc_pid_t Location: motor_ctrl.h Comments: PID controller for Id	The same with PID_Iq	The same with PID_Iq
Name: MotorCtrl_stcWmPidReg Type: stc_pid_t Location: motor_ctrl.h Comments: PID controller for speed	The same with PID_Iq	The same with PID_Iq
Name: MotorCtrl_stcRunPar Type: stc_motor_run_t Location: motor_ctrl.h Comments: Structure for motor control	int32_t i32TargetSpeedRpm	Motor target speed
	int32_t i32MotorSpdLpf	Motor max target speed
	int32_t i32TargetSpeedRpmMax	Controller output
	int32_t i32TargetSpeedRpmMin	motor min target speed
	int32_t i32Q8_EstimWmHz	Motor speed Hz
	int32_t i32Q8_EstimWmHzf	Motor speed Hz Lpf
	uint8_t u8Status	motor running status
	Uint32_t u32ErroType	motor running error type
	int32_t i32Q8_Vbus	sampled bus voltage
	int32_t i32Q8_VR	sampled VR value
	int32_t i32Q22_DeltaThetaTs	Delta theta
	int32_t i32Q22_DeltaThetaKTs	Delta theta calculate factor
	int32_t i32Q8_TargetSpeedWmHz	motor target speed Hz format
	int32_t i32Q22_TargetSpeedWmHz	motor target speed Hz format
	int32_t i32Q22_TargetWmIncTs	motor target speed acceleration
	int32_t i32Q22_TargetWmDecTs	motor target speed decleration
	int32_t i32Q22_ElecAngle	motor target electric angle
	uint8_t u8SpeedPiEn	flag for speed PI enable or disable
	uint8_t u8StartupCompleteFlag	flag for startup complete
	uint8_t u8RunningStage	motor running stage
	uint8_t u8Runninglevel	Motor running level
	uint8_t u8CloseloopFlag	flag for enter closed loop or not
	uint8_t u8ChangeSpeedEn	flag for change speed or not
Name: MotorCtrl_stcluvwSensed Type: stc_uvw_t	int32_t i32Q8_Xu	phase-a variable
	int32_t i32Q8_Xv	phase-b variable

Appendix F: Tunable Parameters List

Variable's Name in Project	Structure Member	Comments
Location: motor_ctrl.h Comments: Structure for motor current sampling results	int32_t i32Q8_Xw	phase-c variable
Name: MotorCtrl_stcIabSensed Type: stc_ab_t Location: motor_ctrl.h Comments: Structure for alpha-beta axis current	int32_t i32Q8_Xa	alpha variable of fixed 2- phase
	int32_t i32Q8_Xb	beta variable of fixed 2- phase
Name: MotorCtrl_stcIdqSensed Type: stc_dq_t Location: motor_ctrl.h Comments: Structure for d-q axis current	int32_t i32Q8_Xd	d-axis variable
	int32_t i32Q8_Xq	q-axis variable
	int32_t i32Q12_Cos	cosine value with angle
	int32_t i32Q12_Sin	sine value with angle
Name: MotorCtrl_stcIdqRef Type: stc_dq_t Location: motor_ctrl.h Comments: Structure for d-q axis reference current	int32_t i32Q8_Xd	d-axis variable
	int32_t i32Q8_Xq	q-axis variable
	int32_t i32Q12_Cos	cosine value with angle
	int32_t i32Q12_Sin	sine value with angle
Name: MotorCtrl_stcVdqRef Type: stc_dq_t Location: motor_ctrl.h Comments: Structure for d-q axis reference current	int32_t i32Q8_Xd	d-axis variable
	int32_t i32Q8_Xq	q-axis variable
	int32_t i32Q12_Cos	cosine value with angle
	int32_t i32Q12_Sin	sine value with angle

Revision history**Revision history**

Document version	Date of release	Description of Change
**	2015-04-01	New application note.
*A	2017-05-02	Updated logo and Copyright.
*B	2018-12-12	Updated to new template. Completing Sunset Review.
*C	2019-07-23	Update Top design, firmware structure with cypress CD motor control library Update the description of PI controller Update the contained code in this AN and the tunable parameters Update Appendix E and Appendix F: Tunable Parameters List
*D	2020-12-02	Update Top design and pictures to match the CY8CKIT-045S

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2020-12-02

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2020 Infineon Technologies AG.

All Rights Reserved.

Do you have a question about this document?

Go to www.cypress.com/support

Document reference

001-93637 Rev. *D

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.