AC445
Application Note
Motor Control Design Using SmartFusion2 and IGLOO2 Devices

Microsemi
Power Matters.™

Microsemi Corporate Headquarters
One Enterprise, Aliso Viejo,
CA 92656 USA
Within the USA: +1 (800) 713-4113
Outside the USA: +1 (949) 380-6100
Fax: +1 (949) 215-4996
Email: sales.support@microsemi.com
www.microsemi.com

About Microsemi

Microsemi Corporation (Nasdaq: MSCC) offers a comprehensive portfolio of semiconductor and system solutions for aerospace & defense, communications, data center and industrial markets. Products include high-performance and radiation-hardened analog mixed-signal integrated circuits, FPGAs, SoCs and ASICs; power management products; timing and synchronization devices and precise time solutions, setting the world's standard for time; voice processing devices; RF solutions; discrete components; enterprise storage and communication solutions, security technologies and scalable anti-tamper products; Ethernet solutions; Power-over-Ethernet ICs and midspans; as well as custom design capabilities and services. Microsemi is headquartered in Aliso Viejo, California, and has approximately 4,800 employees globally. Learn more at www.microsemi.com.

51900445. 3.0 11/17

Contents

Figures

Tables

1 Revision History

The revision history describes the changes that were implemented in the document. The changes are listed by revision, starting with the current publication.

1.1 Revision 3.0

The following is a summary of changes made in revision 3.0 of the document.

- Design Requirements, page 3, Design Prerequisites, page 3 and References, page 3 were added.
- FOC Algorithm Blocks, page 5 was edited to add the list of blocks in FOC algorithm.
- Hall Interface, page 23 and Resolver Interface, page 24 were added in Position and Speed Calculation, page 19.
- BLDC Motor Control Operation, page 30 was edited.
- Figure 29, page 23, Figure 30, page 24 and Figure 32, page 25 were added.
- Input and output port details were updated. For more information see Table 2, page 32.
- The parameters in Table 3, page 33,Table 4, page 33 and Table 5, page 37 were modified
- Table 6, page 38 was added.
- Conclusion, page 38 was edited.
- Figure 3, page 6, Figure 4, page 7, Figure 9, page 11, Figure 11, page 13, Figure 12, page 14,Figure 16, page 17, Figure 17, page 17, Figure 20, page 19, Figure 21, page 19, Figure 23, page 20, Figure 28, page 23, Figure 31, page 24, Figure 33, page 26, Figure 35, page 27 and Figure 40, page 29 were added.
- Added the following new sections:
  - Sinc3 filter, page 6
  - ADC Scaling, page 7
  - PWM Scaling, page 13
  - Sequence Controller, page 16

1.2 Revision 2.0

Updated with SAR 81439 and converted to the new template.

1.3 Revision 1.0

Revision 1.0 was the first publication of this document.

2 Motor Control Design Using SmartFusion2 and IGLOO2 Devices

This application note provides information about the top-level FPGA hardware design of field oriented control (FOC) algorithm for permanent-magnet synchronous motor (PMSM) control and a microstepping algorithm based stepper motor control using SmartFusion®2 SoC FPGA and IGLOO®2 FPGA devices. Microsemi offers simple and easy to use reference designs to implement with SmartFusion2 devices to develop motor control applications. Following are the distinctive features of the Microsemi motor control reference design:

- Advanced motor control for three-phase PMSM motors
- Sensorless control with current and speed regulation
- Sensorless FOC loop time of ~2 µs when implemented in the FPGA fabric
- Rotation in forward and backward directions and changing direction on-the-fly
- All the developed IP blocks can be scaled and configured
- SmartFusion2 devices contain:
    - A hard embedded microcontroller subsystem (MSS)
    - An FPGA fabric consisting of programmable logic tiles
    - Mathblocks
    - Static random access memory (SRAM)
    - SERDES channels
    - Phase-locked loops (PLLs)

The highly integrated SmartFusion2 device has major advantages in terms of the following components:

- MSS
- FPGA fabric
- Hard mathblocks
- Ethernet
- Controller area network (CAN)
- Universal serial bus (USB)
- Serial peripheral interface (SPI)
- Inter-integrated circuit (I^2C)
- 3.3 V I/O interfaces

These components make SmartFusion2 a preferred choice in the development of motor driver control, power supply regulators, UPS, solar inverters, and so on. The SmartFusion2 device offers low-power advantage, high-immunity to single event upset (SEU), and high-level of security. With an FPGA-based motor controller. Designers have flexibility in terms of design while achieving a reliable and deterministic performance.

2.1 Design Requirements

The following table describes the hardware, software requirements, and IP cores required for motor control design.

Table 1 • **Design Requirements**

Requirement	Version
Hardware	
SmartFusion2 Motor Control Starter Kit	
Host PC	(To run Libero v11.8 SP2)
Software	
Libero® SoC	v11.8 SP2
SoftConsole	v4.0
IP Cores	
ADC Scaling	4.2.0
BLDC Estimator	4.2.0
Encoder Interface	4.1.1
FOC Transforms	4.1.1
Hall Interface	4.1.0
Open Loop Manager	4.1.1
PI Controller	4.1.0
PWM Scaling	4.1.0
Rate Limiter	4.1.0
SVM	4.2.0
Sequence Controller	4.1.0
Speed ID IQ PI Controller	4.1.1
Stepper Angle Generation	4.1.0
Three Phase PWM	4.1.2
Sinc3_filter	4.2.0

2.2 Design Prerequisites

The following list of encrypted IP cores and reference designs are the design prerequisites for this application note: These IP cores can be downloaded from *https://www.microsemi.com/form/67-motor-control-encrypted-ip.*

- BLDC Sensorless and Stepper Motor
- BLDC with Encoder
- BLDC with HALL
- BLDC with Resolver

2.3 References

See the following webpage for a complete and up-to-date listing of motor control documentation: *http://www.microsemi.com/applications/motor-control#resources*.

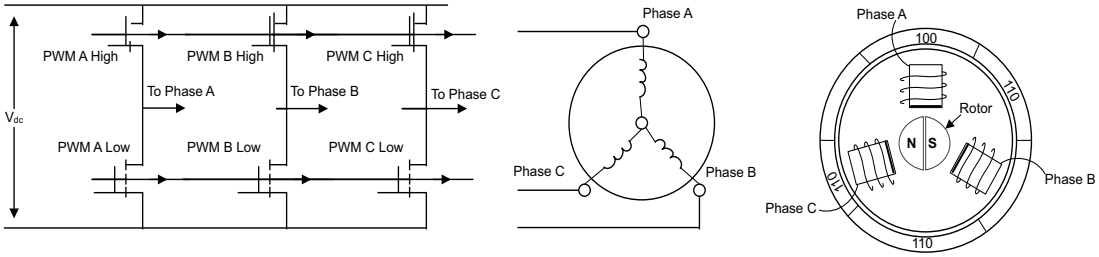
2.4 Control Theory

The following sections describe the sensorless control of permanent magnet synchronous motors (PMSM) and various IP blocks used in the design.

2.4.1 Permanent Magnet Synchronous Motor

PMSMs are rotating electrical machines in which the stator has phase windings and the rotor has permanent magnets. The air gap magnetic field is provided by these permanent magnets and they remain constant. While a conventional direct current (DC) motor commutates using mechanical commutation, a PMSM needs an electronic commutation for the direction and magnitude control of current through its windings. As PMSM motors have the armature coils on the stator, they need to be commutated externally with the help of an external switching circuit and a three-phase inverter. The following figure shows PMSM and the corresponding driving inverter bridge topology.

Figure 1 • **PMSM and Driving Inverter Topology**



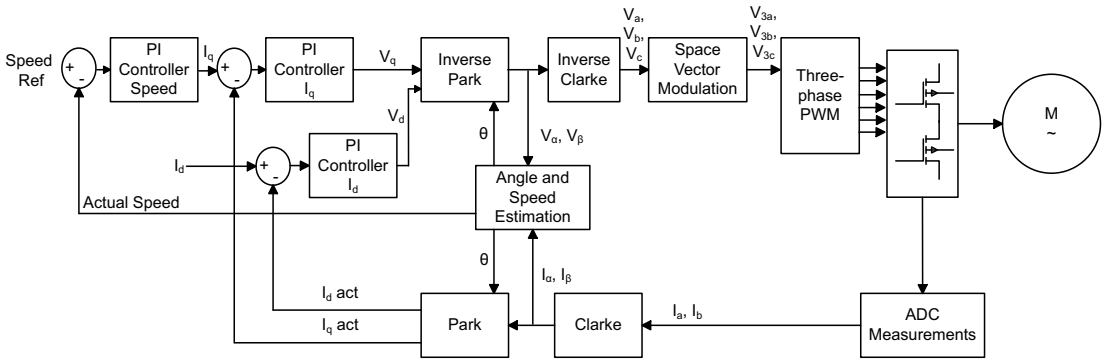
In permanent magnet motors, one of the magnetic fields is created by the permanent magnets and the other is created by the stator coils. Torque is produced due to the interaction of the two magnetic fields, which causes the motor to rotate. The maximum torque is produced when the magnetic vector of the rotor is at 90º to the magnetic vector of the stator. PMSMs are classified based on the wave shape of the induced electromotive force (EMF), that is, sinusoidal and trapezoidal. The sinusoidal type is known as PMSM and the trapezoidal type is known as permanent magnet brushless DC (BLDC) machine. However, both motors can be controlled using sinusoidal control based on FOC. The rotor position information is required to synchronize the machine line currents and their sinusoidal back-EMF. The rotor position can be obtained using resolvers, encoders, or hall sensors depending on the level of accuracy required by the application. Regardless of the choice, these position transducers have inherent disadvantages such as reduced reliability due to their sensitivity to vibration, high temperature, electromagnetic noise, increased costs, and weight. For these reasons, the sensorless control of PMSM is preferred.

In sensorless control of a PMSM, the rotor position can be estimated by the back-EMF of the motor. A PLL based estimator is used to estimate the machine's back-EMF and generate the estimated speed and rotor position.

2.4.2 Sensorless FOC Theory

The following figure shows the block diagram of a sensorless FOC algorithm. It has an inner current-control loop and an outer speed-control loop along with the other blocks required for the PMSM FOC

Figure 2 • **Block Diagram of Sensorless FOC**



In DC motors, the flux and torque producing currents are orthogonal and can be controlled independently. The magneto motive forces developed by these currents are also held orthogonally.

The torque equation is given below

$$T_e = K_a \Phi(I_f) I_a$$

where,

$\Phi(I_f)$ = Flux as a function of field current.

I_a = Armature current.

Flux is only dependent on the field winding current. If the flux is kept constant, then the torque can be controlled by the armature current. Therefore, DC machines have decoupled or independent control of the torque and flux.

In AC machines, stator and rotor fields are not orthogonal to each other. The only current that can be controlled is the stator current. FOC is the technique used to achieve a decoupled control of the torque and flux by transforming the stator current quantities (phase currents) from stationary reference frame into torque and flux producing current components into rotating reference frame.

Following are the advantages of FOC:

- Transformation of a complex and coupled AC model into a simple linear system
- Independent control of torque and flux, similar to a DC motor
- Fast dynamic response and good transient and steady-state performance
- High torque and low current at startup
- High efficiency
- Wide speed range through field weakening

2.4.3 FOC Algorithm Blocks

The following sections describe the blocks used to implement the FOC system.The FOC Algorithm consists of the following blocks:

2.4.3.1 Current Measurement

The current measurement block interfaces with an analog to digital converter (ADC) that accurately measures the current at periodic intervals. The current measurement block triggers the beginning of the ADC conversion process and collects sampled results from up to six channels. It triggers ADC start conversion (sampling) process and collects serial data from ADC and converts it to respective channel information. The phase currents are typically measured from the current measurement block for every FOC loop.

2.4.3.2 Sinc3 filter

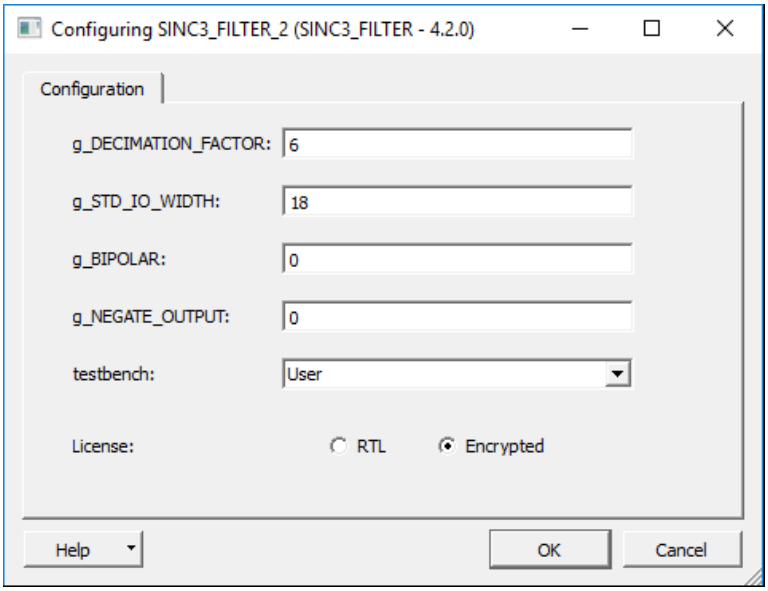
Sinc3 filter is a low-pass filter, which removes all high frequency components above the cutoff frequency without affecting the frequencies below the cutoff frequency. An ideal Sinc3 filter requires an infinite delay. A practical Sinc3 filter has three stages of integration and three stages of differentiation at a decimated clock.

Sinc3 filter is used to:

- De-modulate data from a delta-sigma modulator, which is widely used in delta-sigma type ADC
- Convert the serial bitstream from ADC to parallel data

The following figure shows the configuration of Sinc3 filter block.

Figure 3 • **Configuration of Sinc3 Filter Block**



2.4.3.3 ADC Scaling

An analog to digital converter (ADC) converts a voltage signal as input to a digital signal in a number of bits that depends on ADC resolution. Signals that are not available in voltage form, such as currents, are converted to a voltage signal before conversion to digital data. The digital signal output of the ADC that is relative to its bit width must be scaled to a value that can be properly interpreted by the system that processes the signal. The digital output of ADC can have an unintended DC offset that could be generated by the signal processing circuit before ADC input or the ADC itself. The offset value in signal measurement must be removed before using it in digital algorithm. The ADC scaling block performs the following functions:

- Performs auto offset computation initially:
  Involves disabling Pulse width modulation (PWM) and triggering the ADC by a pre-determined number of times (8192) to determine ADC offset.

$$ChX\_offset = adc\_result\_chX\_i + ChX\_offset$$

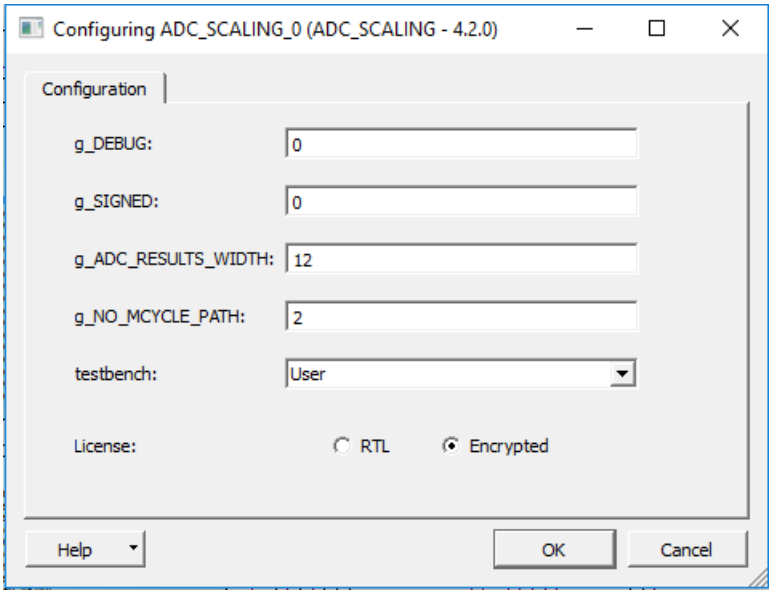
- Scales raw ADC data into phase currents:

$$I_{ph} = \left(adc\_result\_chX\_i - \left(\frac{ChX\_offset}{8192}\right)\right) \times \left(\frac{adc\_scale\_val\_i}{256}\right)$$

- Detects over current fault after phase current computation, which can be used to stop the motors immediately.

The following figure shows the configuration of ADC scaling block.

Figure 4 • **Configuration of ADC Scaling Block**



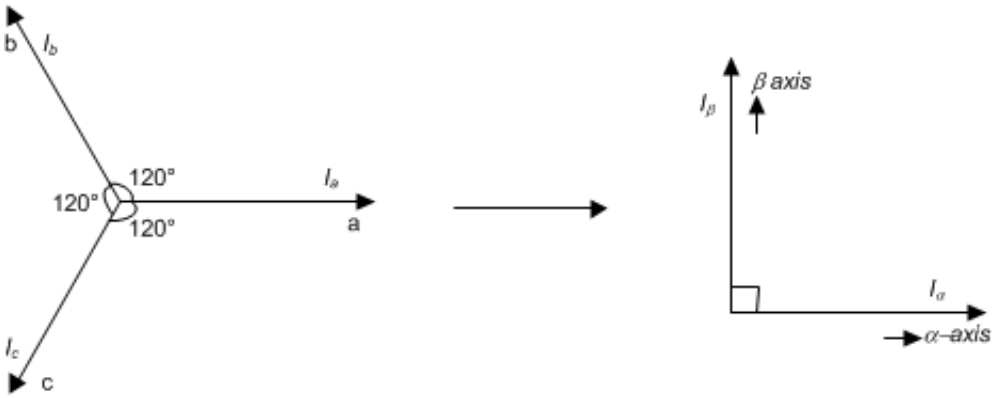
2.4.3.4 FOC Transformations

The FOC transformations block provides Clarke, Park, Inverse Clarke, and Inverse Park functionalities. The transforms use a shared multiplier block for optimum usage of resources.

2.4.3.4.1 Clarke Transformation

The following figure shows the measured motor phase currents that are transformed from a stationary three-phase reference frame to an orthogonal two-axis stationary reference frame.

Figure 5 • **Clarke Transformation**



The Clarke transformation is expressed as:

$$I_\alpha = I_a$$

$$I_\beta = \frac{1}{\sqrt{3}}(I_a + 2I_b)$$

where,

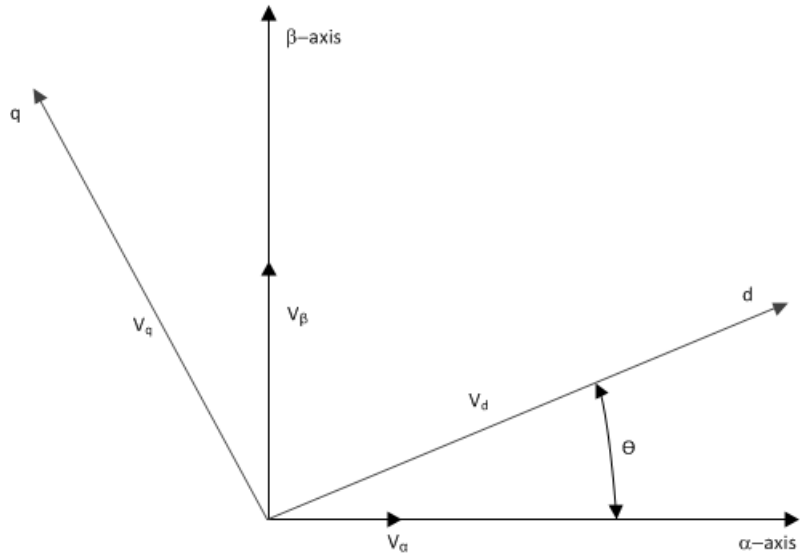
I_a and I_b = Three-phase quantities.

I_α and I_β = Stationary orthogonal reference frame quantities.

2.4.3.4.2 Inverse Clarke Transformation

The following figure shows the transformation from a two-axis orthogonal stationary reference frame to a three-phase stationary reference frame that is accomplished using Inverse Clarke transformation.

Figure 6 • **Inverse Clarke Transformation**



The Inverse Clarke transformation is expressed as:

$$V_a = V_\alpha$$

$$V_b = \frac{-V_\alpha + \sqrt{3} \times V_\beta}{2}$$

$$V_c = \frac{-V_\alpha - \sqrt{3} \times V_\beta}{2}$$

where,

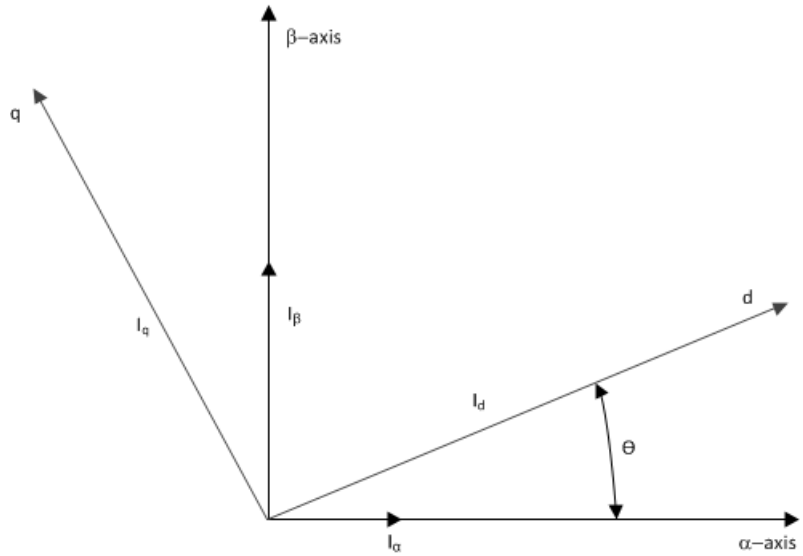
V_a, and V_b = Three-phase quantities.

V_α and V_β = Stationary orthogonal reference frame quantities.

2.4.3.4.3 Park Transformation

The following figure shows the two-axis orthogonal stationary reference frame quantities that are transformed into rotating reference frame quantities using Park transformation.

Figure 7 • **Park Transformation**



The Park transformation is expressed as:

$$I_d = I_\alpha \times \cos(\theta) + I_\beta \times \sin(\theta)$$

$$I_q = I_\beta \times \cos(\theta) - I_\alpha \times \sin(\theta)$$

where,

I_d and I_q = Rotating reference frame quantities.

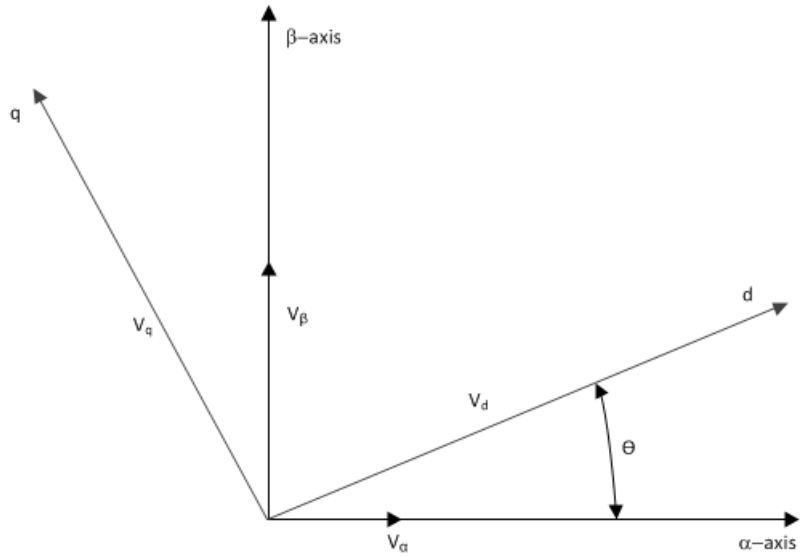
I_α and I_β = Orthogonal stationary reference frame quantities.

θ = Angle of rotating reference frame.

2.4.3.4.4 Inverse Park Transformation

The following figure shows the quantities in rotating reference frame that are transformed to two-axis orthogonal stationary reference frame using Inverse Park transformation.

Figure 8 • **Inverse Park Transformation**



The Inverse Park transformation is expressed as:

$$V_\alpha = V_d \times \cos(\theta) + V_q \times \sin(\theta)$$

$$V_\beta = V_q \times \cos(\theta) + V_d \times \sin(\theta)$$

where,

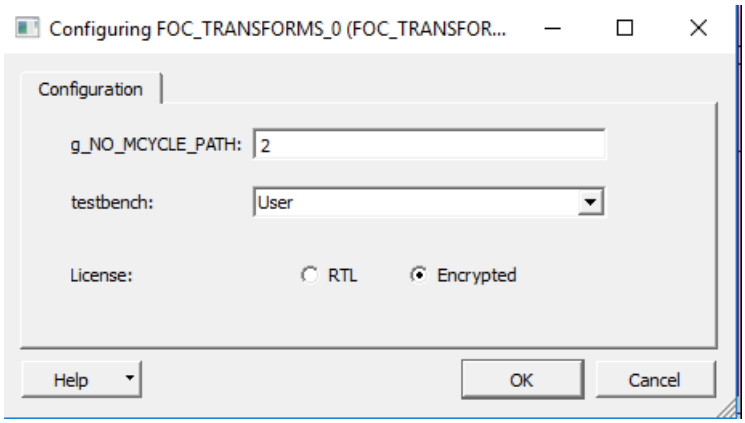
V_α and V_β = Orthogonal stationary reference frame quantities.

V_d and V_q = Rotating reference frame quantities.

θ = Angle of rotation.

The following figure shows the configuration of FOC transformation block.

Figure 9 • **Configuration of FOC Transforms Block**



2.4.3.5 Space Vector Modulation

The output of the Inverse Clarke transformation provides the duty cycles of the PWM channels that correspond to the three-phase voltages. For sinusoidal excitation of the phase voltages, these duty cycle values can be used directly.

However, by using the space vector modulation (SVM) technique, the DC voltage utilization factor is increased. There are many conventional ways of implementing the available SVM algorithms. A simplified method, which is equivalent to the conventional modulation strategy, is used in the current implementation.

In this method, the instantaneous average of the minimum and maximum of all three-phase voltages is calculated as the voltage offset. This instantaneous voltage offset is then subtracted from each of the instantaneous three-phase voltages. This is known as the SVM Min-Max method.

The following equations are used in the SVM Min-Max method (sine with third harmonics injection):

$$V_{offset} = \frac{[Min(V_a, V_b, V_c) + Max(V_a, V_b, V_c)]}{2}$$

$$V_a' = \frac{2}{\sqrt{3}}(V_a - V_{offset})$$

$$V_b' = \frac{2}{\sqrt{3}}(V_b - V_{offset})$$

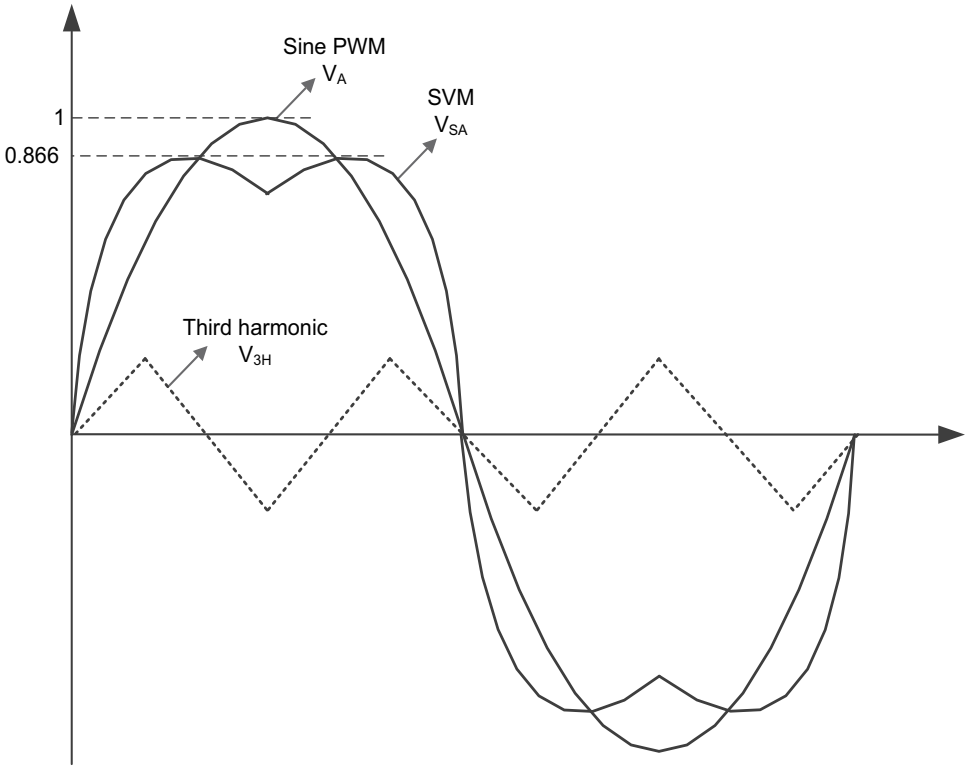
$$V_c' = \frac{2}{\sqrt{3}}(V_c - V_{offset})$$

where,

V_a', V_b', V_c' = Third harmonic injected phase voltages.

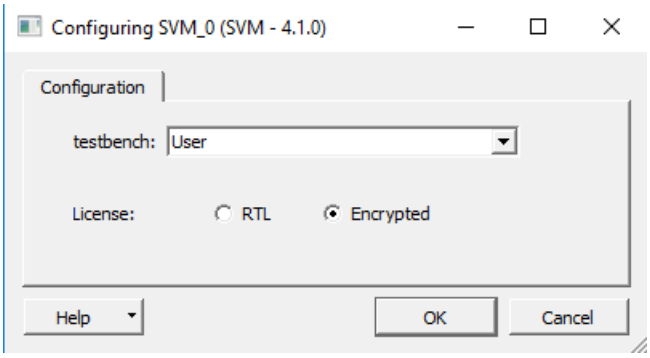
The following figure shows the final third harmonic injected phase voltage waveforms of each phase of the SVM module.

Figure 10 • **Space Vector Modulation**



The following figure shows the configuration of space vector modulation block.

Figure 11 • **Configuration of Space Vector Modulation Block**



2.4.3.6 PWM Scaling

PWM scaling is used to scale down the voltages computed from the FOC to fit with in the PWM carrier wave magnitude range. It also adds a bias to shift negative voltages to positive level.

The PWM scaling IP block performs the following functions:

- Scaling of phase voltages according to the following equation:

$$V_{ph\_o} = (pwm\_period\_i \times 32768 + (pwm\_gain\_i \times V_{ph\_i})/2)/65536$$

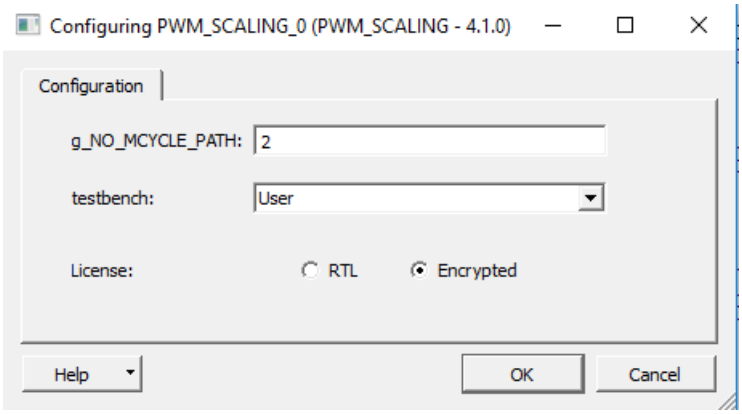
- To use the advantage of voltage boost provided by SVM, pwm_gain_i can be multiplied by a factor of 1.15 as shown in the following equation:

$$pwm\_gain\_i = \frac{pwm\_period\_i \times 1182}{1024}$$

Note: The pwm_period variable is related to PWM switching frequency configured in PWM3ph IP. For more information about PWM switching frequency, see *UG0362: Three-phase PWM User Guide.*

The following figure shows the configuration of PWM scaling block.

Figure 12 • **Configuration of PWM Scaling Block**



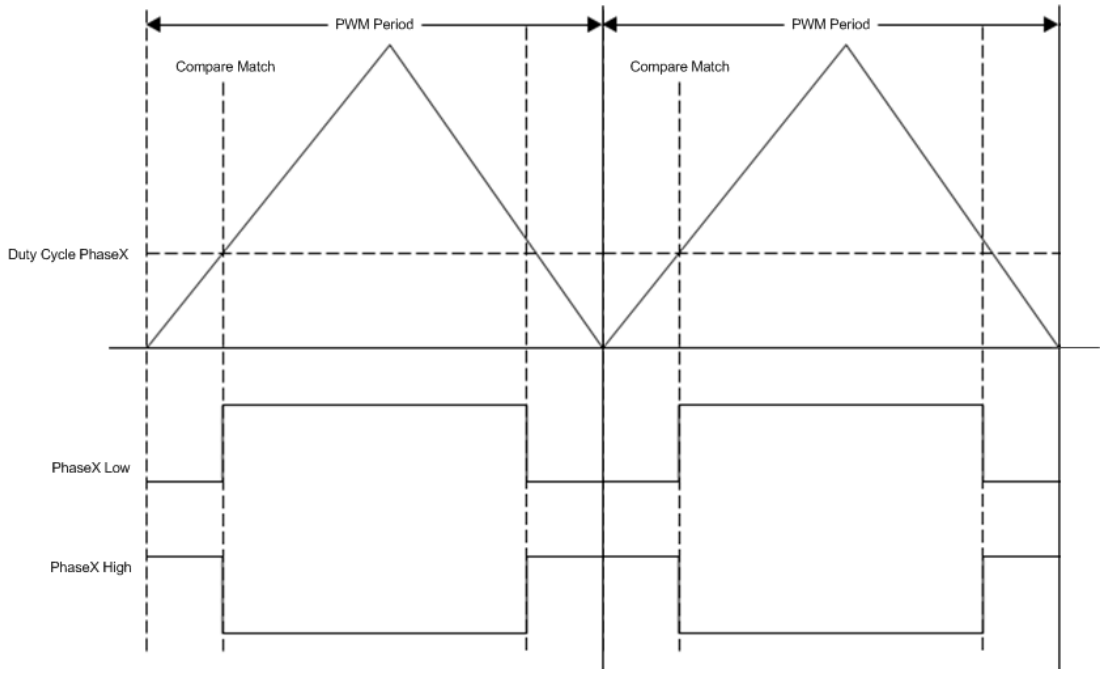
2.4.3.7 PWM Generation

Generation of three-phase, center aligned PWM is supported in the demo design. Dead time insertion logic is included to avoid catastrophic short circuit conditions of the inverter's high and low-side switches. A total of six PWM signals are generated; three for the high-side switches and three for the low-side switches. The PWM for high and low-side switches are complementary for the same inverter leg.

Center Aligned PWM

The following figure shows the center-aligned PWM operation.

Figure 13 • **PWM Generation of Center-Aligned Mode**

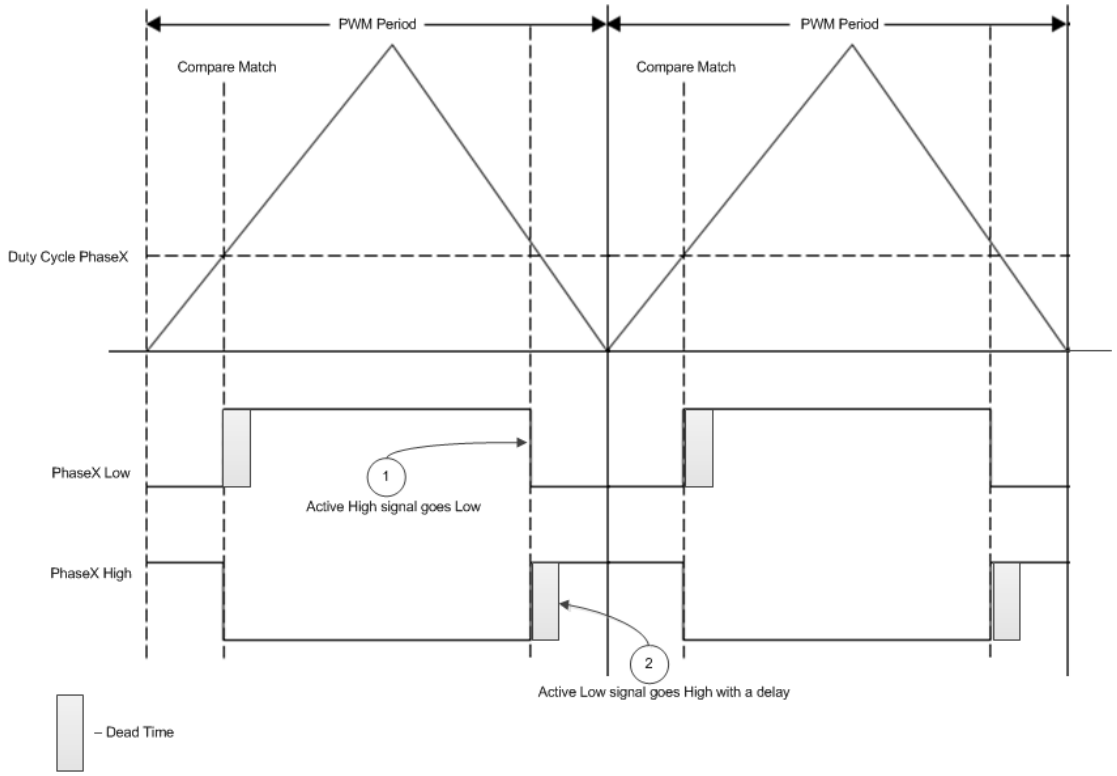


Dead Time Configuration

Turn-off time is one of the characteristics of switching devices. This is the time between removing the gate signal and extinguishing the current completely. In an inverter, when one of the two phase switches is turned OFF, and the other switch is turned ON before the lower switch extinguishes the current flowing through it completely, a dead short occurs. To avoid this, a break-before-make logic feature is implemented.

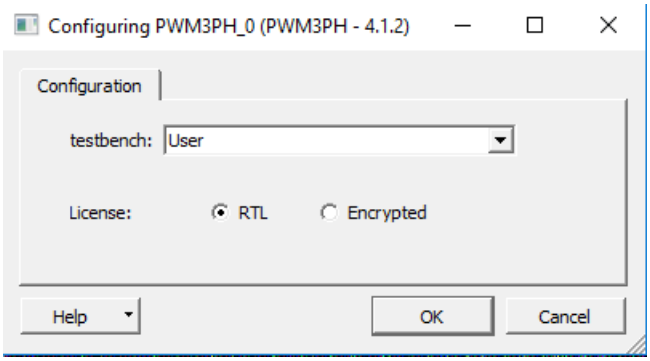
The following figure shows the dead time configuration of the center-aligned PWM. The PWM generation IP block is implemented on the FPGA fabric for a predictable and safe operation.

Figure 14 • **Dead Time Configuration for Center-Aligned PWM**



The following figure shows the configuration of PWM configuration block.

Figure 15 • **Configuration of PWM Configuration Block**

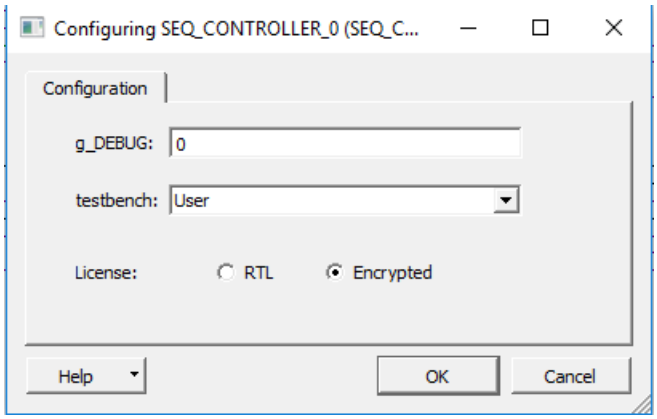


2.4.3.8 Sequence Controller

Implementation of FOC of AC motor needs an intelligent state machine (FSM) apart from the transformations and closed loop control. It is useful to have all the state transitions managed in a single IP module. The Sequence Controller IP manages the starting, stopping, fault, and fault clear operations through FSM. It also manages the transition from closed loop to open loop and vice versa. It acts as a master block that controls all other IPs involved in FOC. The sequence controller triggers the ADC sampling and conversion, enables and disables the PWM based on the motor operating state and also enables and disables current and speed PI controllers.

The following figure shows the configuration of sequence controller block.

Figure 16 • **Configuration of Sequence Controller Block**



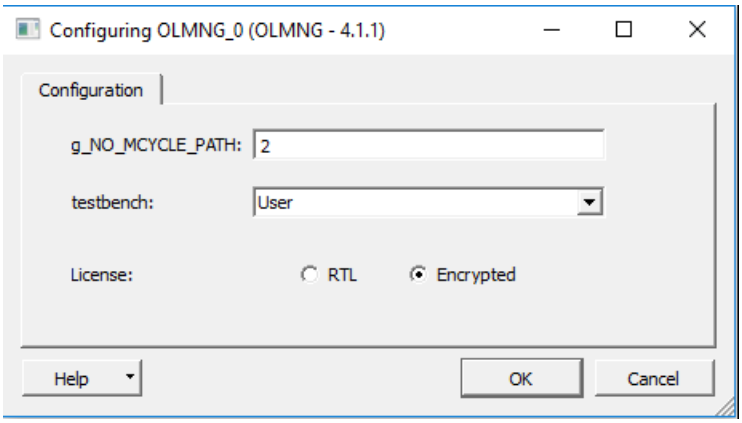
2.4.3.9 Open-Loop Management Block

The open-loop management block provides the following functionality:

- Calculating open-loop angle based on the speed reference
- Switching between open-loop angle and closed-loop angle
- Providing open-loop current references or open-loop voltage references

The following figure shows the configuration of open-loop management block.

Figure 17 • **Configuration of Open-Loop Management Block**

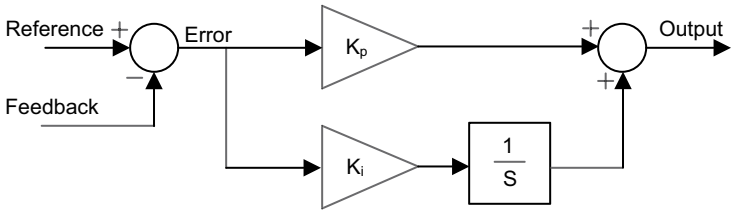


2.4.3.10 PI Controller

PI controller is the widely used closed-loop controller for controlling a first order system. The basic functionality of a PI controller is to make the feedback measurement track the reference input. It controls its output till the error between reference and feedback signals is zero.There are two components that contribute to the output, the proportional term and the integral term as shown in the following figure.

The proportional term depends only on the instantaneous value of the error signal, whereas the integral term depends only on the present and previous values of error.

Figure 18 • **PI Controller in Continuous Domain**

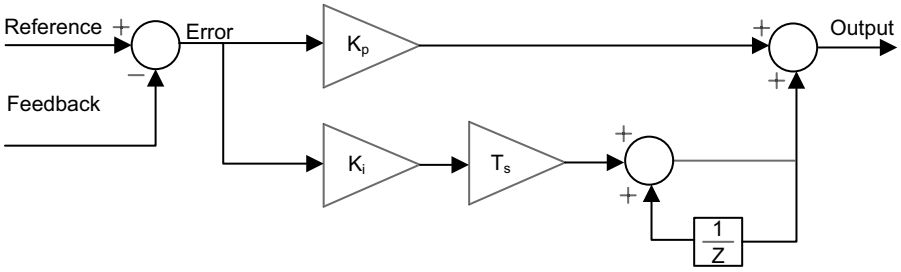


The PI controller in continuous time domain is expressed as:

$$y(t) = K_p \times e(t) + K_i \times \int_0^t e(t)dt$$

To implement the PI controller in digital domain, it has to be discretized. The discretized form of PI controller based on zero order hold method is shown in the following figure.

Figure 19 • **PI Controller - Zero Order Hold Method**



The PI controller based on zero order hold method is expressed as:

$$P(n) = K_p \times e(n)$$

$$I(n) = K_i \times T_s \times e(n) + I(n-1)$$

$$Y(n) = P(n) + I(n)$$

where,

P(n) = Proportional term output.

I(n) = Integral term output.

I(n-1) = Previous (buffered) value of Integral output.

T_S = Sampling time in discrete domain.

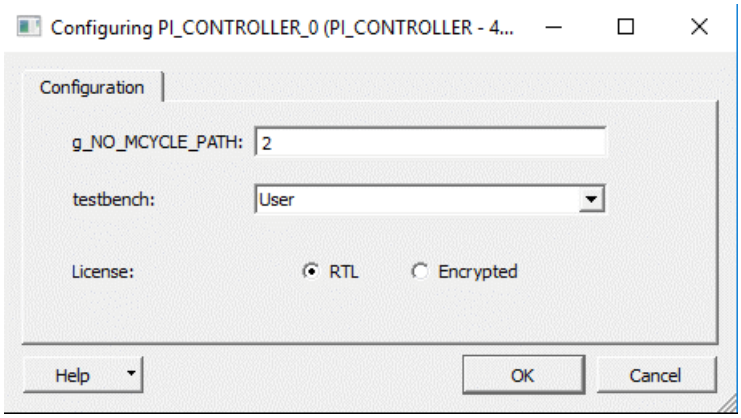
Anti-Windup and Initialization

The PI controller has minimum and maximum limits for its output to keep it within practical values. If a non-zero error signal persists for a long time, the integral component of the controller increases and reaches a maximum bit width limit. This phenomenon is called integrator windup and has to be avoided to have proper dynamic response. The PI controller IP has an automatic anti-windup function that limits the integrator when the PI controller reaches the saturation point.

In applications such as motor control, it is important to initialize the PI controller to a proper value before enabling it. Initializing the PI controller to a good value avoids jerky operation. The IP block has an enable input to enable or disable the PI controller. If disabled, the output is equal to the *initial* input and when enabled, the output is the PI computed value.

The following figure shows the configuration of PI controller block.

Figure 20 • **Configuration of PI Controller Block**



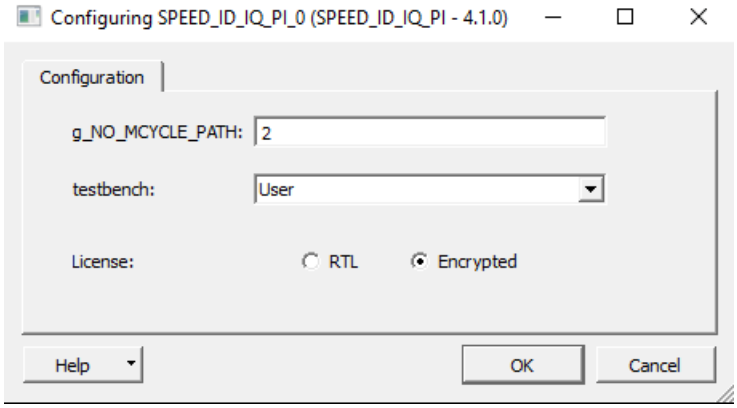
Time Sharing of PI Controllers

In FOC algorithm, there are three PI controllers for speed, d-axis current I_d, and q-axis current I_q. The input of one PI controller depends on the output of other PI controller and therefore they are executed sequentially. At any instant, there is only one instance of PI controller in operation. Therefore, instead of using three individual PI controllers, a single PI controller is time shared for speed, I_d, and I_q for optimum usage of resources.

The Speed_Id_Iq_PI module allows sharing PI controller through start and done signals for each of speed, I_d, and I_q. The tuning parameters K_p, K_i, and minimum and maximum limits of each instance of the controller can be configured independently through corresponding inputs.

The following figure shows the configuration of Speed_Id_Iq_PI module.

Figure 21 • **Configuration of Speed_Id_Iq_PI Module**



2.4.3.11 Position and Speed Calculation

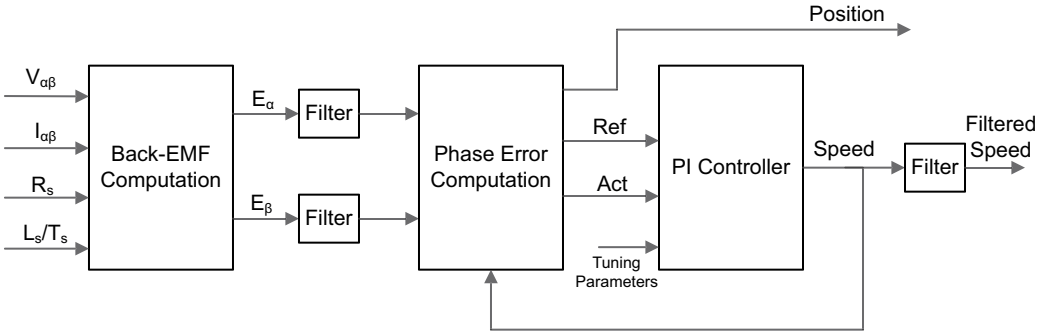
The following sections describe various methods to determine the rotor position and speed of the motor using sensorless algorithm and various sensor interfaces.

2.4.3.11.1 *Sensorless Control*

The position and speed estimator block computes the rotor position based on the motor parameters, voltages, and currents as shown in the following figure. The algorithm is based on back-emf estimation and filtering. The motor parameters R_s, L_s, and the sampling time T_s are used to build the motor model. The voltages that are fed to the actual motor are fed to the motor model along with motor currents and are used to compute back-EMF.

A PLL structure is used to find the angle of filtered-back EMF, which is aligned to the rotor electrical position.

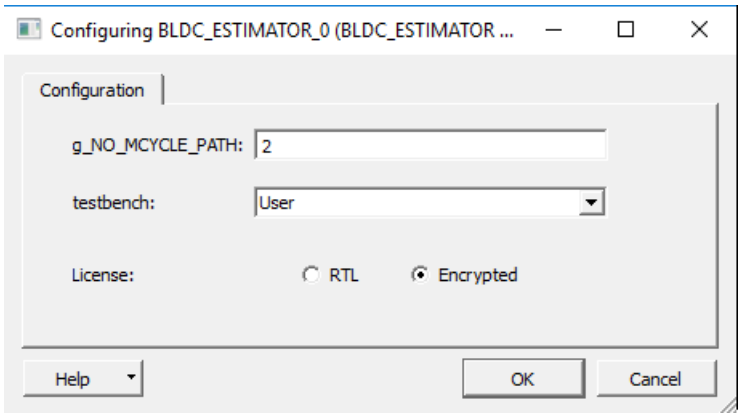
Figure 22 • **Block Diagram of Position and Speed Estimator**



The motor can accelerate or decelerate rapidly in which case the rate of change of rotor position has to be dynamically and accurately tracked by the PLL. This is achieved by proper tuning of the PI controller that is part of the PLL.

The following figure shows the configuration of BLDC estimator block.

Figure 23 • **Configuration of BLDC Estimator Block**

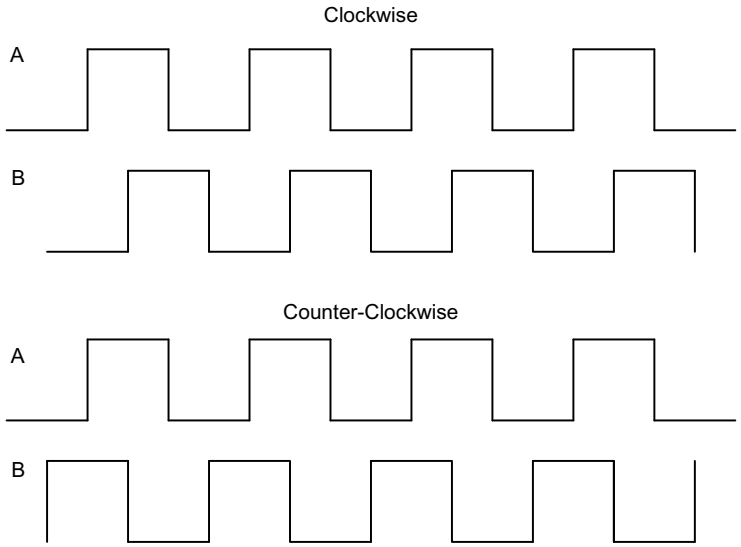


2.4.3.11.2 Encoder Interface

Incremental encoder is the most common sensor used for FOC of BLDC or PMSM. This sensor gives relative angular position as output in the form of pulses. A quadrature encoder typically produces two outputs, which have pulses phase shifted by 90° as shown in the following figure.

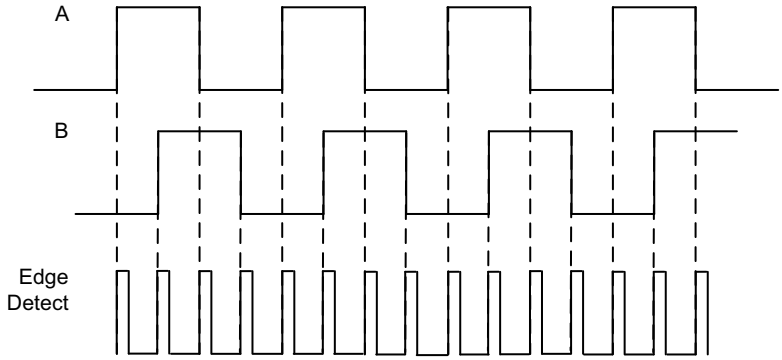
The phase shift between the two signals A and B represents the direction of rotation.

Figure 24 • **Encoder Signals in Clockwise and Counter-Clockwise Directions**



The encoder interface logic uses edge detection on rising edge and falling edge of A and B, as shown in the following figure. This gives a resolution that is four times the encoder resolution and produces a higher resolution from a low cost encoder.

Figure 25 • **Edge Detection of Encoder Pulses for Higher Resolution**



After the edge detection, counters are used to get rotor angular position in terms of electrical angle so that it can be directly used for FOC. The angle output ranges from 0 to 262143, where 262143 represents 360°. The variation of angle output with respect to edges is shown in Figure 26, page 22 for positive speed and Figure 27, page 22 for negative speed. The speed output is calculated based on rate of change of angular position.

The following parameters are used to configure the encoder interface IP:

$$\text{Angle factor} = \frac{16777216 \times \text{Number of pole pairs}}{2}$$

$$\text{Speed factor} = \frac{384000}{\text{Encoder Resolution}}$$

The above equation computes the speed in rpm.

$$\text{Speed factor} = \frac{384000 \times 65535}{\text{Encoder Resolution} \times \text{Rated speed}}$$

The above equation computes the speed per unit, where 65535 represents rated speed.

Figure 26 • **Theta Output for Positive Direction**

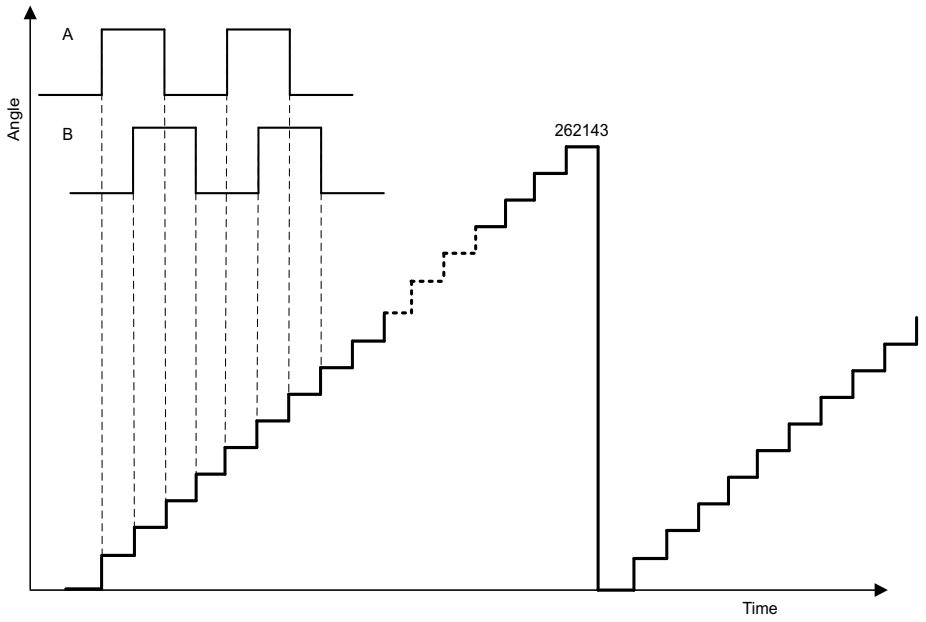
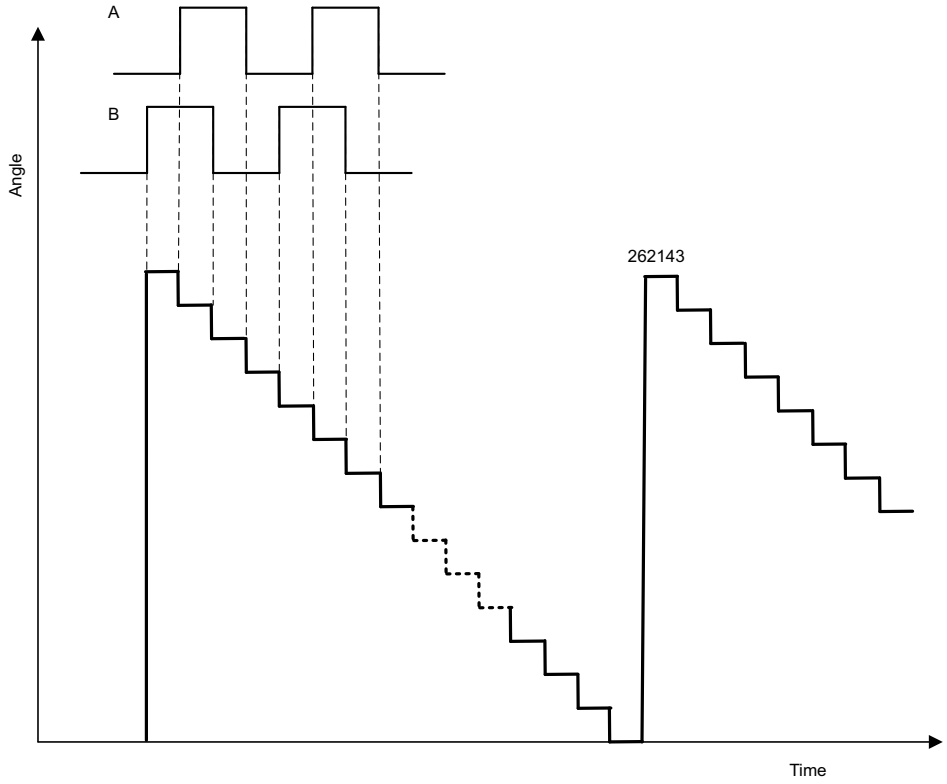
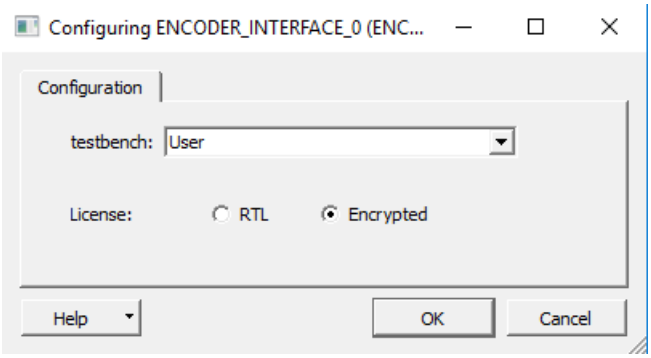


Figure 27 • **Theta Output for Negative Direction**



The following figure shows the configuration of encoder interface block.

Figure 28 • **Configuration of Encoder Interface Block**

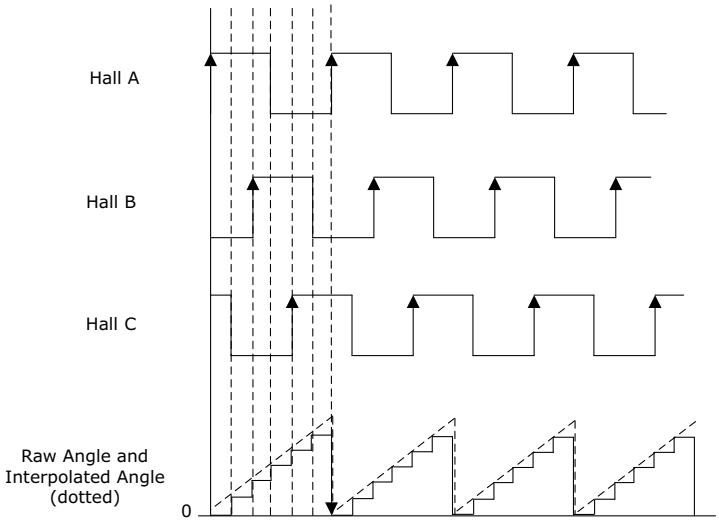


2.4.3.11.3 Hall Interface

A Hall event is a change of state of the Hall signal. In one cycle of the Hall signal, six Hall events can be detected. Each event indicates an angle movement of one sixth of the total cycle. This means that each Hall event indicates a change of 60° (electrical).

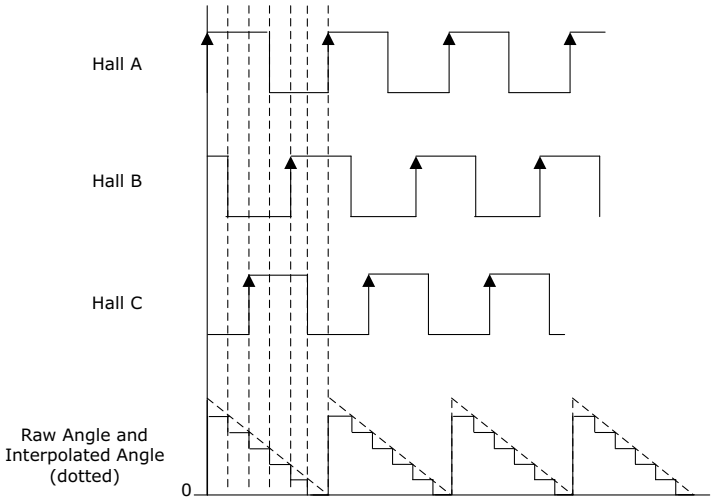
For ripple free operation, the raw angle is smoothened by interpolation to obtain a finer angle. This angle is used in the FOC transformation. The following figure shows the Hall angle signals and the angle generated from these signals (assuming the Hall signals are active high).

Figure 29 • **Hall Signals and Angle Generation**



The following figure shows the computed angle generation for the reverse direction of the active high Hall signals.

Figure 30 • **Hall Signals and Angle Generation for Reverse Direction**



Speed Calculation

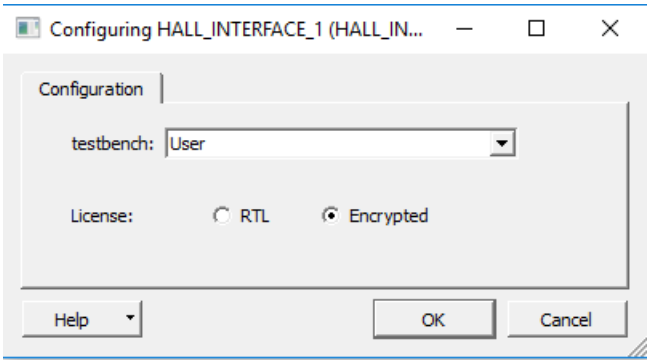
The rotor speed is calculated based on the change of angle in a given time window. The time window varies based on the motor speed and the speed obtained is scaled appropriately.

The Hall sensors provide the electrical angle of the rotor. The mechanical speed is affected by the number of pole pairs. The Hall interface block uses the number of pole pairs to compute the scaling factor. The speed factor is calculated as:

$$\text{Speed Factor} = \frac{384 \times \text{Motors speed} \times \text{Number of pole pairs}}{1600}$$

The following figure shows the configuration of hall interface block.

Figure 31 • **Configuration of Hall Interface Block**

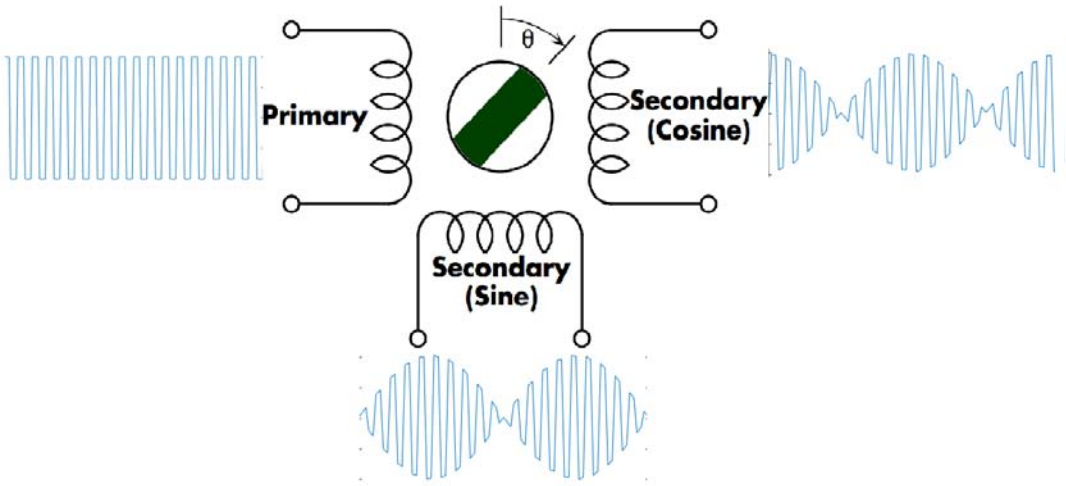


2.4.3.11.4 Resolver Interface

The operating principle of a resolver is similar to the operating principle of a synchro. Resolvers are typically built like small motors with a rotor that is attached to the shaft whose position is to be measured and a stator (stationary part) which takes the excitation signals and produces the output signals. A resolver typically consists of a primary winding, also called excitation winding and two secondary windings called cosine and sine windings. The secondary windings are geometrically placed such that winding signals are cosine and sine function of rotor angle.

The following figure shows the signals generated by resolver.

Figure 32 • **Signal Generation in Resolver**



The device used to process the resolver signals and convert them in to a digital angle format is called a resolver to digital converter (RDC). The resolver IP implements the functionality of a RDC. The IP provides excitation signal in the form of a square wave to the primary winding whose frequency can be configured. The IP processes the secondary signals, demodulates them and calculates the angle and speed of the rotor.

The resolver interface IP generates a square wave that is fed to the primary winding of the resolver. The frequency of the square wave can be configured through hf_sig_period_i input. The cos_i and sin_i signals from the secondary windings are demodulated and filtered to get effective cosine and sine signals. A phase-locked loop (PLL) is used to extract angle and speed from cosine and sine signals.

The PLL uses a PI controller whose gains pll_pi_kp_i and pll_pi_ki_i can be tuned to get required response time. A higher value for gains results in quick response to angle and speed changes but can also induce noise in angle and speed outputs.

In motor control application, the resolver zero position must be aligned with motor magnetic zero position. To achieve this, a calib_angle_i signal is used. During calibration process, the signal goes high and the motor is forced to align its rotor to magnetic zero position. The angle output is reset to zero during this period and is taken as reference for measuring absolute angle. A motor and resolver can have multiple pole pairs in which the motor control algorithm needs multiple theta transitions (3600) for one mechanical rotation of the rotor. This feature can be configured through the pp_ratio_i port, listed in Table 4, page 33.

The theta_factor constant is calculated by using the following equation. The calculated speed can be scaled to per unit using theta_factor_i.

$$\text{theta\_factor} = \frac{18.12 \ * \ \text{Rated Motor Speed(RPM)}}{\text{System Clock (MHz)}}$$

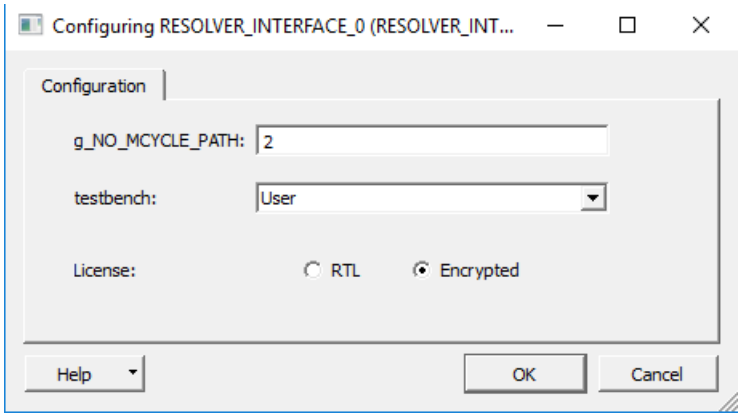
The hf_sig_period input determines the frequency of the square wave injected into resolver primary, calculated by using the following equation.

$$\text{hf\_sig\_period\_i} = \frac{f_{sys\_clk}}{\text{hf\_freq} \times 2}$$

where,
hf_freq = Frequency of the square wave injected into resolver primary
fsys_clk = Frequency of the system clock provided at sys_clk_i input

The following figure shows the configuration of resolver interface block.

Figure 33 • **Configuration of Resolver Interface Block**



2.4.3.12 Rate Limiter

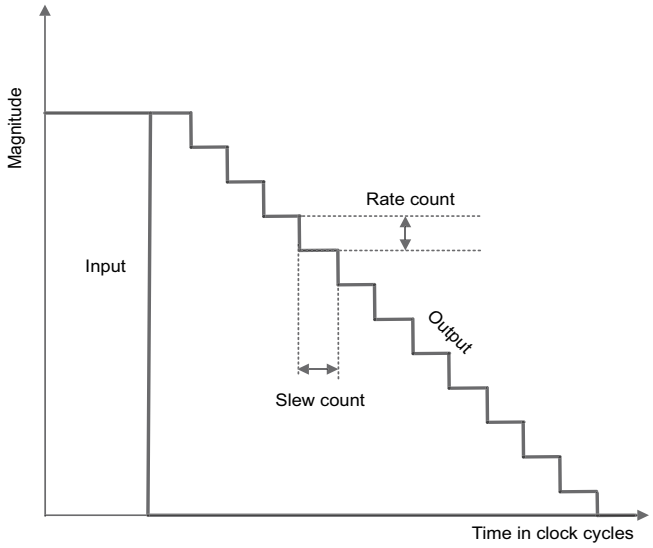
Rate limiter is generally used to generate a smooth speed profile while changing from one speed to another. The rate of change of output remains the same whether the output increases or decreases with respect to time. The output slope is configured by two parameters—rate count and slew count.

The rate limiter IP also has a reset ramp input, which forces the output to become zero instantaneously, irrespective of the input value. This feature is useful for auto-restart in motor control application to initialize speed reference output from the rate limiter to zero before starting the motor again.

The soft stop input forces the output to become zero, irrespective of the input value. The output goes towards zero according to the ramp profile configured by the slew count and the rate count. When the output reaches zero, the soft stop is asserted in acknowledgment.

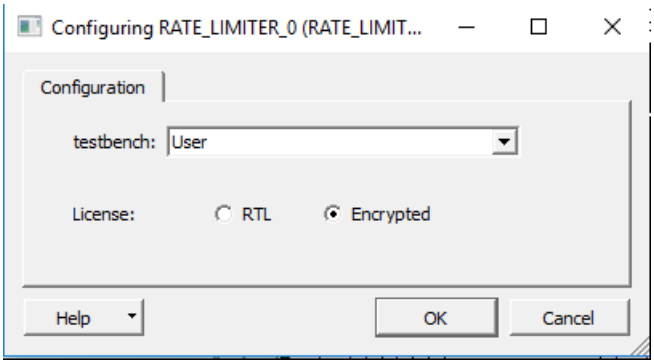
The following figure shows the operation of the rate limiter block.

Figure 34 • **Rate Limiter Operation**



The following figure shows the configuration of rate limiter block.

Figure 35 • **Configuration of Rate Limiter Block**



2.4.3.13 Stepper Theta Generation

Stepper motor is used for position control by moving through certain number of steps. While a stepper motor has a fixed number of steps per revolution, it is possible to move through microsteps, thereby improving step resolution. Microstepping also reduces torque ripple and power losses in the motor. The IP block generates theta that is used by stepper motor control algorithm. It is possible to select micro-stepping up to 2048 microsteps. The IP allows running the motor in speed mode or position mode. The profile of the stepper theta generation output and the resultant current for various microstepping options are shown in the following figures.

Figure 36 • **Currents and Theta in Full Step Mode**

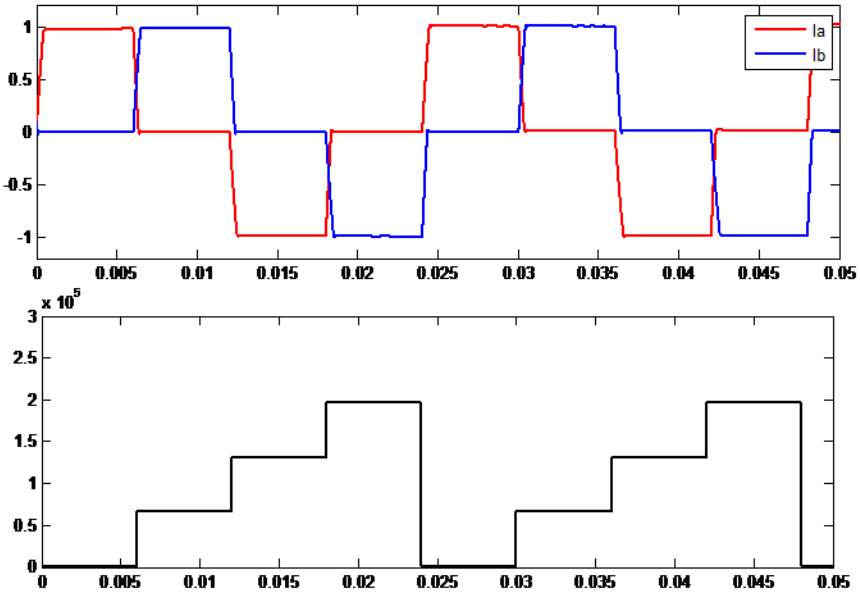


Figure 37 • **Currents and Theta in Half Step Mode**

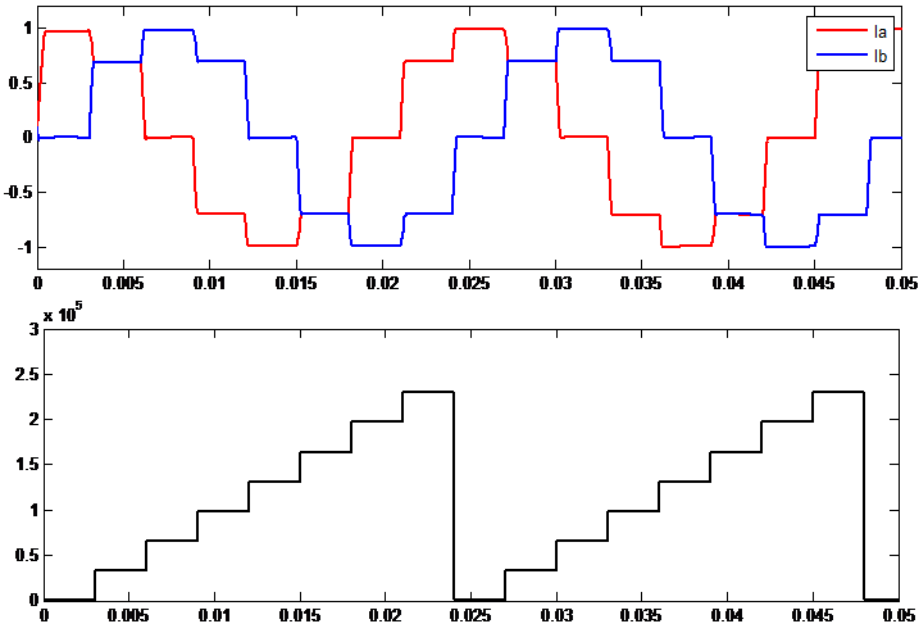


Figure 38 • **Currents and Theta in Quarter Step Mode (Micro-Stepping)**

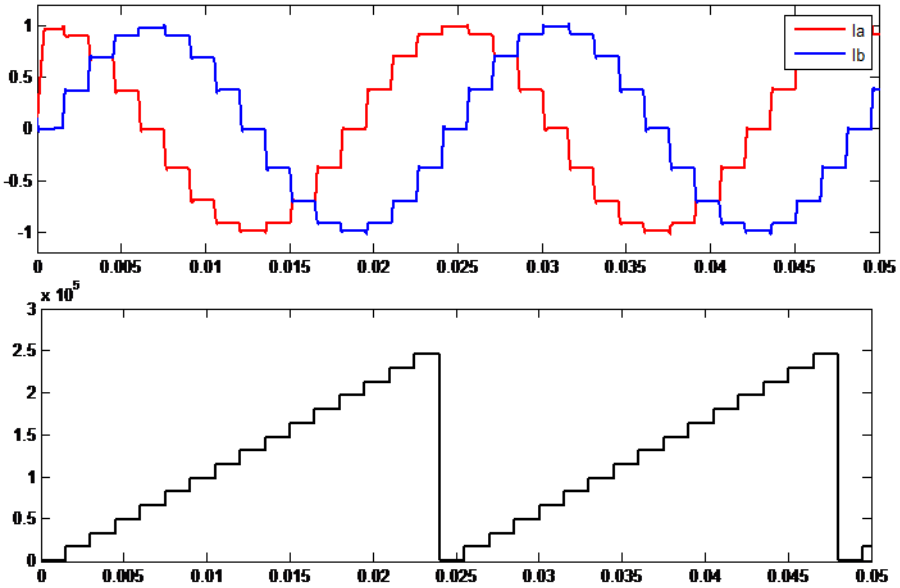
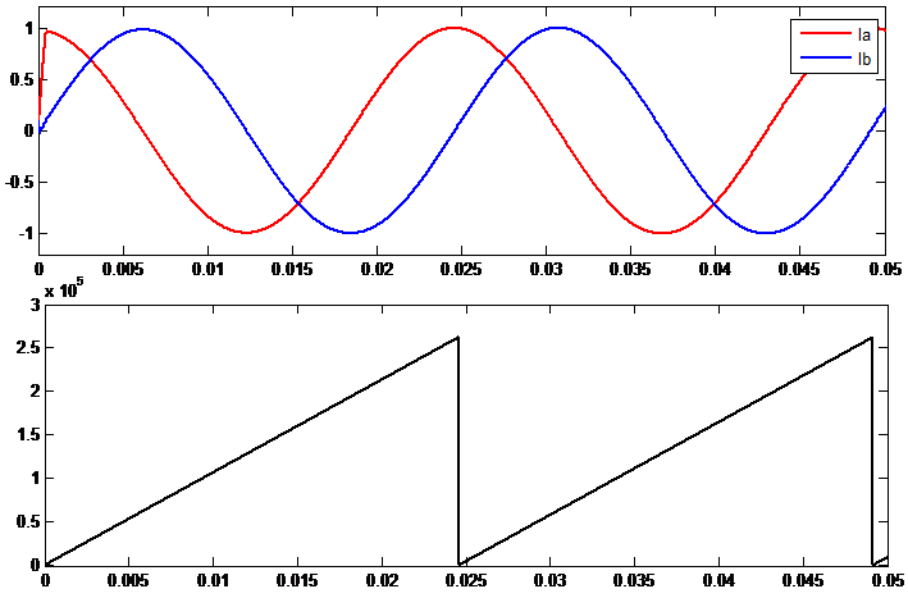


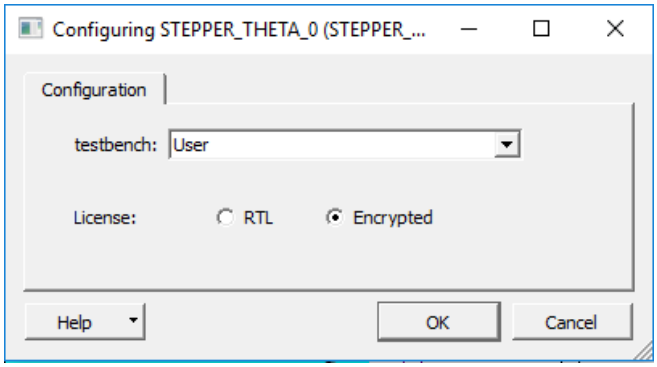
Figure 39 • **Currents and Theta in 1/1024 Step Mode (Micro-Stepping)**



The amount of microstepping is decided by the rate limit input and must be an exponent of two. The slew count input then decides the speed at which the theta value is updated. The output theta is generated till the command number of steps are met and then theta is held at the last updated value until command steps change. However, in speed mode, the output theta is continuously updated.

The following figure shows the configuration of stepper theta generation block.

Figure 40 • **Configuration of Stepper Theta Generation Block**



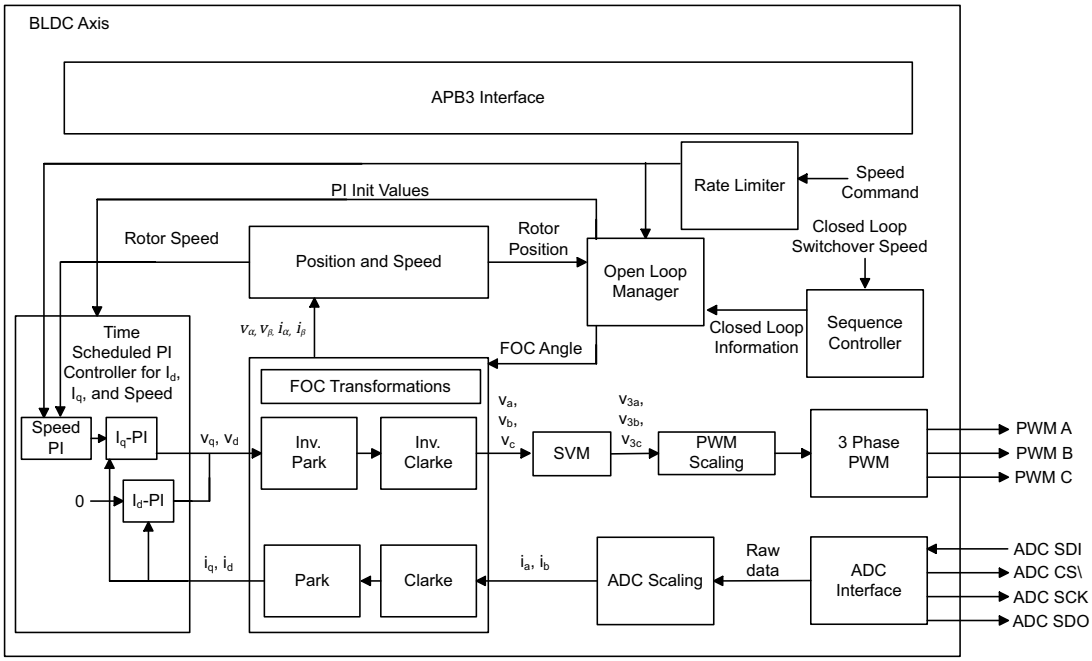
2.5 FPGA Fabric Design

The FPGA design is implemented using VHDL hardware description language and is designed to efficiently utilize SmartFusion2 and IGLOO2 fabric resources.

2.5.1 Fabric Implementation of BLDC Motor Control

The following figure shows the FOC sensorless implementation of PMSM motor, which can control speed and torque of a BLDC motor.

Figure 41 • **Fabric Implementation - BLDC Motor Control**



2.5.2 BLDC Motor Control Operation

The APB3 interface programs registers in various FPGA blocks with parameters from the MSS. The sequence controller block controls various functions through a finite-state machine (FSM). The sequence controller block can check for rotor slip and attempt to restart the motor. The open-loop manager block produces the open-loop angle based on the reference speed from the rate limiter. The rate limiter block converts a step change in reference speed to a controlled change in speed in smaller steps. The position and speed block estimates the rotor angle and rotor speed based on the inputs from hall sensors, encoders, or the currents in the sensorless control design.

The following steps summarize the operation of sensorless FOC loop:

1. The ADC interface obtains raw data from the ADC. This data is passed to the ADC scaling block.
2. The ADC scaling block scales the raw data and removes the offset to produce the motor phase currents. The result is passed to the FOC transformations block as Clarke inputs. The ADC scaling block can also detect if the current level is above safe levels and issue a fault signal to the sequence controller.
3. The FOC transformations block uses the phase currents in the three-phase stationary reference frame to compute the two-phase orthogonal stationary reference frame currents (I_α and I_β). These currents are then used to compute the two-phase rotating reference frame currents ($I_d$and I_q) using the FOC angle. The currents obtained are regulated using a PI controller.
4. The PI controller block is time scheduled to operate on speed, I_d, and I_q. The actual motor speed is available from the position and speed estimator block, while the speed reference is generated by the rate limiter block. The speed PI output is used as the reference for the I_q PI and the actual I_q value is received from the FOC transformations block. The output of the I_q PI is assigned as V_q. The reference to the I_d PI is tied to zero and the actual value is obtained from the FOC transformations block. The output of the block is assigned as V_d.
5. The FOC transformations block converts V_d and V_q values (two-phase rotating reference frame) into V_α and V_β (two-phase orthogonal stationary reference frame) using the Inverse Park transform, and

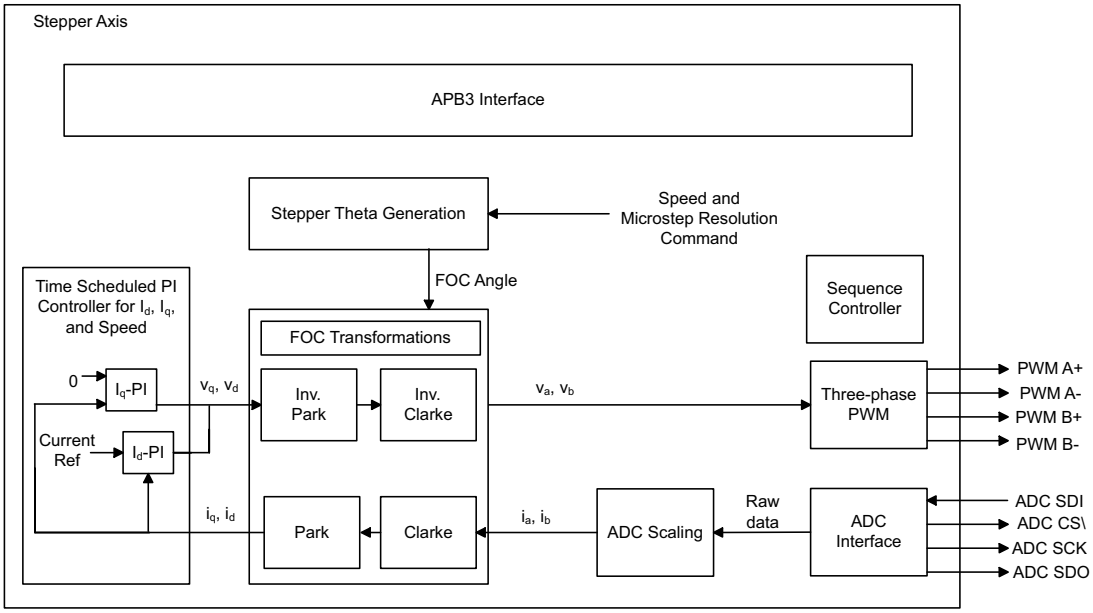
converts the V_α and V_β into V_a, V_b, and V_c (three-phase stationary reference frame) using the Inverse Clarke transform.

6. The SVM block adds a third harmonic component to V_a, V_b, and V_c.
7. The PWM scaling block scales the voltage signals in a unit to a value comparable with PWM period.
8. The three-phase PWM block converts the voltages into PWM signals with dead time and delay time.

2.5.3 Fabric Implementation of Stepper Motor Control

The following figure shows the FOC implementation to control a stepper motor.

Figure 42 • **Fabric Implementation - Stepper Motor Control**



2.5.4 Stepper Motor Control Operation

The APB3 interface programs registers in various blocks from the MSS. The FOC angle is generated by the stepper theta generation block, which can be configured to produce angles at a given motor speed and step resolution.

The following steps summarize the operation of FOC implementation for a stepper motor:

1. The ADC interface obtains raw data from the ADC. This data is passed to the ADC scaling block.
2. The ADC scaling block scales the raw data and removes the offset to produce the motor phase currents. The result is passed to the FOC transformations block as Park inputs. The ADC scaling block can also detect, if the current level is above safe levels and issues a fault signal to the sequence controller.
3. The FOC transformations block uses the phase currents that are already orthogonal to compute the two-phase rotating reference frame currents ($I_d$and I_q) using the FOC angle. The currents obtained are regulated using the PI controller.
4. The PI controller block is time scheduled to operate on I_d and I_q. Depending on the torque requirement, a current reference is set to the I_d PI, while the actual I_d value is received from the FOC transformations block. The output of the I_d PI is assigned as V_d. The reference to the I_q PI is tied to zero, while the actual value is obtained from the FOC transformations block. The output of the block is assigned as V_q.
5. The FOC transformations block converts V_d and V_q values (two-phase rotating reference frame) into V_α and V_β (two-phase orthogonal stationary reference frame) using the inverse Park transform.
6. The three-phase PWM block converts the voltages into PWM signals. Only two of the three phases are used.

2.5.5 Input and Output Ports

The following table shows the input and output ports of the top block of the fabric.

Table 2 • **Input and Output Ports**

Signal Name	Direction	Description
A0_ADC_SDI_I	Input	Serial data input to device from ADC - BLDC motor
A1_ADC_SDI_I	Input	Serial data input to device from ADC - Stepper motor
A0_ADC_SDO_O	Output	Serial data output to ADC from device -BLDC motor
A0_ADC_SCK_O	Output	Serial clock to ADC - BLDC motor
A0_ADC_CS_O	Output	Chip select signal to ADC - BLDC motor
A1_ADC_SDO_O	Output	Serial data output to ADC from device - Stepper motor
A1_ADC_SCK_O	Output	Serial clock to ADC - Stepper motor
A1_ADC_CS_O	Output	Chip select signal to ADC - Stepper motor
A0_PWM_UH_O	Output	Channel A of PWM for the top switch of the BLDC motor
A0_PWM_VH_O	Output	Channel B of PWM for the top switch of the BLDC motor
A0_PWM_WH_O	Output	Channel C of PWM for the top switch of the BLDC motor
A0_PWM_UL_O	Output	Channel A of PWM for the bottom switch of the BLDC motor
A0_PWM_VL_O	Output	Channel B of PWM for the bottom switch of the BLDC motor
A0_PWM_WL_O	Output	Channel C of PWM for the bottom switch of the BLDC motor
A1_PWM_UH_O	Output	Channel A of PWM for the top switch of the Stepper motor
A1_PWM_VH_O	Output	Channel B of PWM for the top switch of the Stepper motor
A1_PWM_WH_O	Output	Channel A enable signal
A1_PWM_UL_O	Output	Channel A of PWM for the bottom switch of the Stepper Motor
A1_PWM_VL_O	Output	Channel B of PWM for the bottom switch of the Stepper Motor
A1_PWM_WL_O	Output	Channel B enable signal
A0_RUNNING_LED	Output	Running status of the BLDC motor
A1_RUNNING_LED	Output	Running status of the Stepper motor
HA_I	Input	Hall A input signal (Hall design)
HB_I	Input	Hall B input signal (Hall design)
HC_I	Input	Hall C input signal (Hall design)
QA_I	Input	Encoder input A (Encoder design)
QB_I	Input	Encoder input B (Encoder design)
DATA_ADC_COS_I	Input	Cosine winding input signal from ADC (Resolver design)
DATA_ADC_SIN_I	Input	Sine winding input from ADC (Resolver design)
HF_SIGNAL_O	Output	Square wave signal used to drive primary winding of resolver

2.5.6 Generic Parameters

The following table shows the generic parameters used to configure individual IP blocks. Their values can dynamically vary based on the compile time requirement.

Table 3 • **Configuration Parameters**

Name	Description
G_SCK_DIVIDER	ADC clock = System Clock / 2^(g_SCK_DIVIDER)
G_ADC_RESULTS_WIDTH	Width of ADC results register
G_ADC_RES	ADC Resolution
G_NO_MCYCLE_PATH	Number of clock cycle delays before multiplier result ready signal is asserted
G_APB3_IF_ADDR_WIDTH	Address width of APB interface
G_APB3_IF_DATA_WIDTH	Data width of APB interface
G_DEBUG	When 0, synthesis is enabled When 1, enables simulation (with reduced number of offset computation iterations)
G_STD_IO_WIDTH	Width of output register in sinc3 filter
G_DECIMATION FACTOR	Division factor used to generate decimated clock from the system clock output
G_SIGNED	Enables signed computations in ADC Scaling When 0, calculations are unsigned When 1, calculations are signed

2.5.7 Configuration and Debug Parameters

The following sections describe the configuration parameters for the MSS and the debug, status parameters and their addresses that can be read from the FPGA fabric

2.5.7.1 Configuration Parameters

The following table shows the fabric configuration parameters that MSS needs to set. These configuration parameters are used to configure various IP blocks in fabric. The motor control GUI has a feature that computes the programmable parameters in configuration registers.

The run time parameters are configured through the advanced peripheral bus (APB). The run time parameters can also be
hard-coded for devices without MSS.

Table 4 • **List of Macros and Constants to be Modified for Change in Motor Parameters**

Port	APB Address	IP Block	Function
ADC_CONTROL_REG_VAL_O	x000	ADC interface	ADC control register input
ADC_CHANNEL_CH0_O	0x04	ADC interface	ADC configuration value to be transmitted to ADC to obtain Channel 0 conversion results
ADC_CHANNEL_CH1_O	0x08	ADC interface	ADC configuration value to be transmitted to ADC to obtain Channel 1 conversion results
ADC_CHANNEL_CH2_O	0x0C	ADC interface	ADC configuration value to be transmitted to ADC to obtain Channel 2 conversion results
ADC_CHANNEL_CH3_O	0x10	ADC interface	ADC configuration value to be transmitted to ADC to obtain Channel 3 conversion results
ADC_SCALE_VAL_O	x020	ADC scaling	Scaling constant for current

Table 4 • **List of Macros and Constants to be Modified for Change in Motor Parameters *(continued)***

Port	APB Address	IP Block	Function
ADC_OC_THRESH_O	0x24	ADC scaling	ADC overcurrent threshold limit
PWM_PERIOD_VAL_O	x034	Three-phase PWM, PWM scaling	Time period of PWM output signal
PWM_DEAD_TIME_VAL_O	x038	Three-phase PWM	Dead time input to PWM module
PWM_DELAY_TIME_VAL_O	x03C	Three-phase PWM	Delay time input to PWM module
PWM_GAIN_VAL_O	x040	PWM scaling	Gain value for PWM
SPEED_PI_KP_O	x050	Speed ID IQ PI	Kp parameter for Speed PI controller
SPEED_PI_KI_O	x054	Speed ID IQ PI	Ki parameter for Speed PI controller
IDQ_PI_KP_O	x060	Speed ID IQ PI	Kp parameter for current PI controller
IDQ_PI_KI_O	x064	Speed ID IQ PI	Ki parameter for current PI controller
RATE_LIMIT_REF_IN_O	x070	Rate limiter	Speed reference input to rate limiter
RATE_LIMIT_SLEW_CNT_O	x074	Rate limiter	Slew count input to rate limiter
RATE_LIMIT_RATE_CNT_O	x078	Rate limiter	Rate count input to rate limiter
DIRECTION_CONFIG_O	x090	Sequence controller	When 1, motor runs clockwise Otherwise, motor runs anti-clockwise
START_MOTOR_O	x094	Start motor	Starts the motor
OLMNG_THETA_FACTOR_O	x098	Open loop manager	Theta factor input
SEQ_CNTL_CONFIG_O	x09C	Sequence controller	Bit 3: soft stop Bit 2: when 1, the motor runs with the current reference set by SQMNG_IQ_REF_IN_O (with PI controllers enabled) Otherwise, the motor runs with SQMNG_DV_O as voltage reference (PI controllers disabled) Bit 1: sensor calibration 40Bit 0: auto restart enable
OLMNG_DV_O	x0A0	Open loop manager	Voltage input to inverse park when SQMNG_C_BY_F_O is set to 0
OLMNG_IQ_REF_IN_O	x0A4	Open loop manager	Current reference input to I-q PI controller when SQMNG_C_BY_F_O is set to 1
SEQ_CNTL_CL_OMEGA_O	x0A8	Sequence controller	Speed at which the algorithm switches over to angle closed loop
STOP_MOTOR_O	0xAC	Sequence controller	Stops the motor (has higher priority over start motor)
AUTO_RESTART_NO_O	0xB4	Sequence controller	Number of auto-restarts before asserting fault condition due to rotor lock
FAULT_CLR_O	0xB8	Sequence Controller	Active high fault clear signal
ANGLE_LB_FIRST_CONST_O	x200	BLDC estimator	First constant for angle estimation algorithm
ANGLE_LB_SECOND_CONST_O	x204	BLDC estimator	Second constant for angle estimation algorithm
ANGLE_KP_O	x208	BLDC estimator	Kp parameter for angle estimation PI controller
ANGLE_KI_O	x20C	BLDC estimator	Ki parameter for angle estimation PI controller

Table 4 • **List of Macros and Constants to be Modified for Change in Motor Parameters *(continued)***

Port	APB Address	IP Block	Function
FILTER_FACTOR_BEMF_O	x210	BLDC estimator	Defines back-EMF filter time constant in exponent of two times the sampling time
FILTER_FACTOR_OMEGA_O	x214	BLDC estimator	Defines back-EMF filter time constant in exponent of two times the sampling time
THETA_GEN_SLEW_COUNT_O	x700	Stepper Theta	Defines the speed at which angle is changed.
THETA_GEN_RATE_LIMIT_O	x704	Stepper Theta	Defines the angle increment for each time period
THETA_GEN_CMD_STEP_NO_O	x708	Stepper Theta	Defines number of steps or microsteps that the motor should move
THETA_GEN_MODE_O	x70C	Stepper Theta	Speed or Position mode selection
ID_REF_VAL_O	x710	Speed ID IQ PI Controller	Defines the current reference
EN_SPEED_FACTOR	x300	Encoder Interface	Defines the motor speed output scaling multiplier (see, *UG0659: Encoder Interface User Guide*)
EN_ANGLE_FACTOR	x304	Encoder Interface	Defines the angle output scaling multiplier (see, *UG0659: Encoder Interface User Guide*)
EN_ANGLE_COUNT_MAX	x308	Encoder Interface	Maximum angle count value in terms of encoder pulse events
EN_SPEED_FILTER_FACTOR	x30C	Encoder Interface	Represents speed filter time constant – input as an exponent of 2 – 2^(filter_factor)*sampling time
EN_SPEED_WINDOW	x310	Encoder Interface	The time window for speed computation in multiples of 10µs.
EN_POSITION_PI_KP	x314	PI Controller (position)	Kp parameter of Position PI controller
EN_POSITION_PI_KI	x318	PI Controller (position)	Ki parameter of Position PI controller
EN_POSITION_PI_EN	x31C	PI Controller (position)	Enable signal to Position PI controller
EN_POSITION_PI_REF_ADDR	x320	PI Controller (position)	Reference (position) input to Position PI controller
HALL_SENSOR_POLARITY_O	x408	Hall interface	Polarity of hall sensor inputs, When 1, Active high When 0, Active low
HALL_FILTER_FACTOR_O	x40C	Hall interface	Represents speed filter time constant – input as an exponent of 2 – 2^(filter_factor)*sampling time
HALL_SPEED_FACTOR_O	x410	Hall interface	Speed factor input (see, *UG0690: Hall Interface User Guide*)
PLL_PI_KP_O	x920	Resolver Interface	Kp of PI controller used for PLL
PLL_PI_KI_O	x924	Resolver Interface	Ki of PI controller used for PLL
THETA_FACTOR_O	x928	Resolver Interface	Theta factor constant (see, *UG0735: Resolver Interface v4.2 User Guide*)

Table 4 • **List of Macros and Constants to be Modified for Change in Motor Parameters *(continued)***

Port	APB Address	IP Block	Function
HF_SIG_PERIOD_O	x92C	Resolver Interface	Half the value of the high frequency square wave time period (see, *UG0735: Resolver Interface v4.2 User Guide*)
DC_FILTER_FACTOR_O	x930	Resolver Interface	Filter time constant of high-pass filter used to eliminate modulation wave frequency component for sine and cosine signals.
AC_FILTER_FACTOR_O	x934	Resolver Interface	Filter time constant of low-pass filter used to eliminate modulation wave frequency component for sine and cosine signals
PP_RATIO_O	x938	Resolver Interface	The ratio of number of motor poles to number of resolver poles

2.5.7.2 Debug and Status Parameters

The following is a list of debug and status parameters and their addresses that can be read from the FPGA fabric. They allow for visualizing internal parameters for debugging purposes.

The following table shows the list of APB addresses from which internal parameters can be read back.

Table 5 • **Debug and Status Parameters**

Port	APB Address	Function
IA_I	x800	Current in phase A
IB_I	x804	Current in phase B
IALPHA_I	x808	Current in alpha-axis (orthogonal stationary reference frame)
IBETA_I	x80C	Current in beta-axis (orthogonal stationary reference frame)
ID_I	x810	Current in D-axis (rotating reference frame)
IQ_I	x814	Current in Q-axis (rotating reference frame)
VALPHA_I	x818	Voltage in alpha-axis (orthogonal stationary reference frame)
VBETA_I	x81C	Voltage in beta-axis (orthogonal stationary reference frame)
RL_OUT_I	x820	Rate limiter output
SPEED_PI_OUT_I	x824	Speed PI controller output
ID_PI_OUT_I	x828	D-axis current PI controller output
IQ_PI_OUT_I	x82C	Q-axis current PI controller output
THETA_I	x830	Rotor angle input to FOC transformations
ADC_CH0_VAL_I	x838	Channel 0 data from ADC
ADC_CH1_VAL_I	x83C	Channel 1 data from ADC
ADC_CH2_VAL_I	x840	Channel 2 data from ADC
ADC_CH3_VAL_I	x844	Channel 3 data from ADC
FSM_STATE_VAL_I	x0B0	Current FSM state
EALPHA_I	x900	Back-emf on alpha-axis
EBETA_I	x904	Back-emf on beta-axis
EALPHA_FILTER_OUT_I	x908	Filtered alpha-axis back-emf
EBETA_FILTER_OUT_I	x90C	Filtered beta-axis back-emf
OMEGA_PI_OUT_I	x910	Output of angle estimation PI controller
PLL_THETA_I	x914	Estimated angle from PLL
OMEGA_FILTER_OUT_I	x918	Filtered speed
STEP_CNT_I	xE00	Number of steps covered by Stepper Motor
THETA_I	xE04	Angle generated by stepper theta generation block
EN_SPEED_I	xA00	Calculated rotor speed from encoder interface
EN_SPEED_I	xA04	Calculated rotor angle from encoder interface
HALL_ANGLE_RAW_I	xB00	Raw angle output (6 step) based on Hall sensor inputs
HALL_THETA_OUT_I	xB04	Rotor angle output
HALL_OMEGA_OUT_I	xB0C	Speed output based on hall inputs

2.6 Resource Utilization

The following table shows the resource utilization of various designs in SmartFusion2 and IGLOO2 devices.

Table 6 • **Resource Utilization**

Resource Type	Dual Axis	Encoder	Hall	Resolver
Combinational(4LUT)	7320	3830	3820	5300
Sequential	5010	2840	2530	3730
RAM64	0	0	0	0
RAM1k	0	0	0	0
MACC	12	8	7	7
Chip Globals	3	2	2	3

2.7 Conclusion

The SmartFusion2 based dual axis motor control design is a powerful platform for developers to prototype motor control designs with quick turn around time. The design can be used to run one stepper motor and one BLDC motor in sensorless mode, or by using hall or encoder or resolver interface.The design consists of several modular IP blocks that are developed to optimally utilize device resources.