# iMOTION™ Motion Control Engine

## Functional Reference Manual

## About this document

### Scope and purpose

iMOTION™ devices are offering control of permanent magnet motors by integrating both hardware and software.

These devices can perform sensorless or sensor-based Field-Oriented Control (FOC) over the full speed range of the motor, including stable control at deep field weakening speeds.  The IMOTION™ motor control software is offered under the name Motion Control Engine (MCE) hereafter. Besides motor control, MCE also offers Power Factor Correction Control (PFC) option. On top of that, MCE supports scripting function enabling users to implement system level functionalities beyond motor control and PFC and extend the functionality of MCE.

This software reference manual describes the various features supported by MCE including the following topics:

- Application-specific registers that are used to configure motor, PFC and power board parameters
- Flux estimator, speed and current control loop tuning and optimize the motor start-up parameters
- Motor drive performance verification and troubleshooting methods

While this reference manual describes all of the features, protections and configuration options of the MCE, a concrete product might only implement a subset of this functionality.  E.g. the power factor correction is only offered in dedicated devices. Please refer to the respective data sheet of particular devices for more information.

The electrical, mechanical, timing and quality parameters of the iMOTION™ products are described in the respective data sheets. The data sheets also specify the concrete IO pins for the functionalities described here.

This manual refers to MCE software revision MCE FW_V5.1.0.

### Intended audience

This document is targeting users of iMOTION™ devices with the integration of the Motion Control Engine (MCE).

## Table of contents

# 1 Introduction

This document describes the iMOTION™ software for motor control, power factor correction and additional functions. This Software is offered under the name Motion Control Engine (MCE). Key features of this software are listed below.

- Sensorless FOC control: High performance sensorless Field Oriented Control (FOC) of Permanent Magnet Synchronous Motor (surface mounted and interior mount magnet motors) utilizing fast ADC, integrated op-amps, comparator and motion peripherals of iMOTION™ devices.
- Hall sensor based FOC control: support 2 / 3 digital Hall sensor and 2 analog Hall sensor configurations with complementary Atan angle option.
- Angle sensing for initial rotor angle detection: Together with direct closed-loop start, initial angle sensing improves motor start performance.
- Single shunt or leg shunt motor current sensing: Provide unique single shunt and leg shunt current reconstruction. Integrated op-amps with configurable gain and A/D converter enable a direct shunt resistor interface to the iMOTION™ device while eliminating additional analog/digital circuitry. Single shunt option can use either minimum pulse method or the phase shift method. Phase Shift PWM provides better startup and low speed performance in single shunt configuration.
- Support 3ph and 2ph PWM modulation: 2ph SVPWM (Type-3) that allows reduction of the switching losses compared with three-phase SVPWM (symmetrical placement of zero vectors).
- Enhanced flux based control algorithm which provides quick and smooth start: The direct closed-loop control of both torque and stator flux (field weakening) are achieved using proportional-integral controllers and space vector modulation with over modulation strategy.
- Supports Boost Mode Power Factor Correction (PFC).
- Networking capability with user mode UART: Master and slave mode available, with up to 15 nodes and each node has its own address. Broadcast feature available to update all the slaves at once.
- 15 re-programmable parameter blocks: 15 configuration blocks can be programmed to save the control parameters and each parameter block is 256 bytes in size. Each block can be programmed individually or all 15 blocks at the same time using Solution Designer.
- Multiple motor parameter support: Each parameter block can be assigned to different motors or hardware platforms.
- Scripting support to enable users to write system level functionalities above motor control and PFC.

Table 1 List of Motor Control and Common Protection

| Type of Protection | Description | UL60730-1 Certification |
|---|---|---|
| Over Current (Gate kill) | This fault is set when there is over current and shutdown the PWM. This fault cannot be masked. | Yes |
| Critical Over Voltage | This fault is set when the voltage is above a threshold; all low side switches are clamped (zero-vector-braking) to protect the drive and brake the motor. The zero-vector is held until fault is cleared. This fault cannot be masked. | No |
| DC Over Voltage | This fault is set when the DC Bus voltage is above a threshold. | No |
| DC Under Voltage | This fault is set when the DC Bus voltage is below a threshold. | No |
| Flux PLL Out Of Control | This fault is set when motor flux PLL is not locked which could be due to wrong parameter configuration. | Yes |
| Over Temperature | This fault is set when the temperature is above a threshold. | No |
| Rotor Lock | This fault is set when the rotor is locked | Yes |
| Execution | This fault occurs if the CPU load is more than 95%. | Yes |
| Phase Loss | This fault is set if one or more motor phases are not connected | Yes |
| Parameter Load | This fault occurs when parameter block in flash is faulty. | Yes |
| Link Break Protection | This fault is set when there is no UART communication for a defined time limit. | Yes |
| Hall Invalid Protection | This fault is set when hall interface receives invalid Hall pattern. | No |
| Hall Timeout Protection | This fault is set when no Hall input transition is detected for a defined period of time. This fault is to detect rotor lock condition in Hall sensor / hybrid mode. | No |
| Current Offset Calibration Protection | This fault is checked in current OFFSET calibration state after offset measurement is completed. When this protection happens, system enters into fault state. | No |

Table 2 List of PFC Protection

| Type of Protection | Description | UL60730-1 Certification |
|---|---|---|
| Over Current (Gate kill) | This fault is set when there is over current and shutdown PWM. Cannot be masked. | Yes |
| DC Over Voltage | This fault is set when the DC Bus voltage is above a threshold. | No |
| DC Under Voltage | This fault is set when the DC Bus voltage is under a threshold. | No |
| AC over voltage | This fault is set when the AC input voltage to PFC is above a threshold. | Yes |
| AC under voltage | This fault is set when the AC input voltage to PFC is below a threshold. | Yes |
| Frequency fault | This fault is set when AC input frequency value to PFC is different from set value | Yes |
| Parameter Load | This fault occurs when wrong values in parameter block in the flash. | Yes |

# 2 Software Description

This section describes MCE motor control and power factor correction features and functions.

## 2.1 Motor Control: Sensorless / Hall Sensor Based FOC

The MCE provides an advanced sensorless or Hall sensor based Field Oriented Control (FOC) algorithm to drive Permanent Magnet Synchronous Motor (PMSM) loads including constant air-gap surface mounted permanent magnet (SPM) motors and interior permanent magnet (IPM) motors with variable-reluctance. A top-level sensorless / Hall sensor based FOC algorithm structure is depicted in Figure 1. The implementation follows the well-established cascaded control structure with an outer speed loop and inner current control loops that adjust the motor winding voltages to drive the motor to the target speed. The field weakening block extends the speed range of the drive.

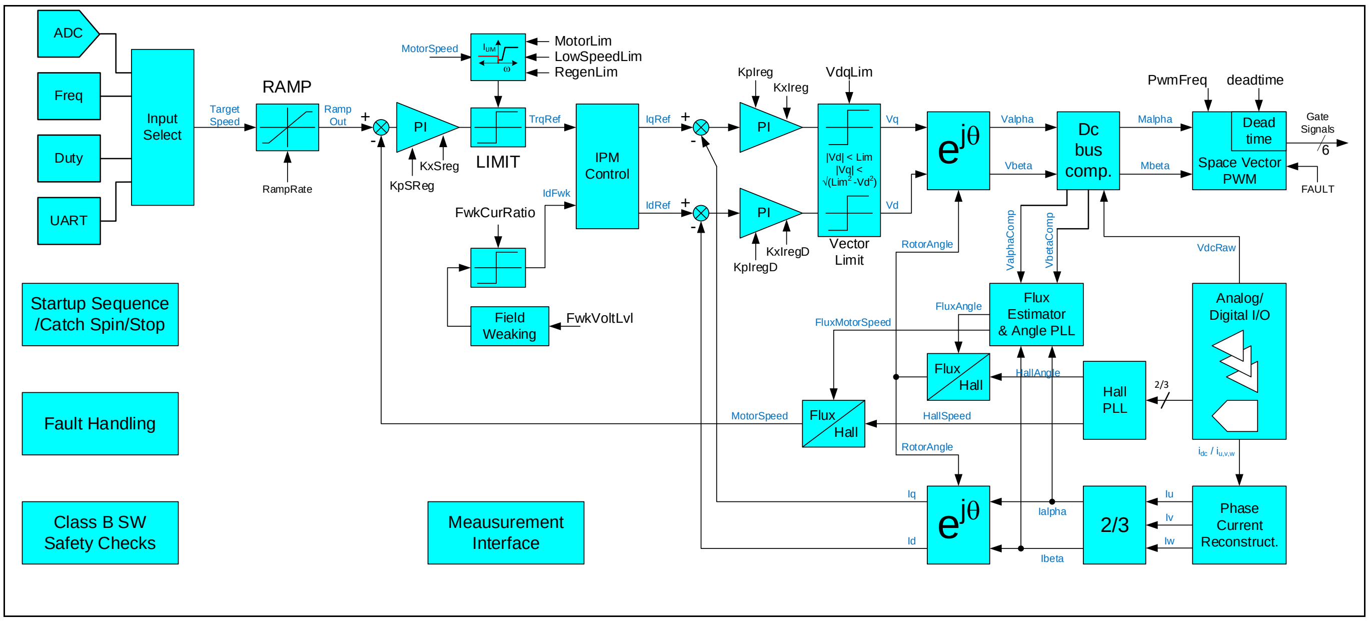


**Figure 1 Top Level Diagram of Speed Control Loop and Sensorless FOC**

The speed controller calculates the motor torque required to follow the target speed. While the current loops drive the motor currents needed to generate this torque. The proportional plus integral (PI) speed loop compensator acts on the error between the target speed and the actual (estimated) speed. The integral term forces the steady state error to zero while the proportional term improves the high frequency response. The PI compensator gains are adjusted depending on the motor and load characteristics to meet the target dynamic performance. The limiting function on the output of the PI compensator prevents integral windup and maintains the motor currents within the motor and drive capability.

The current loops calculate the inverter voltages to drive the motor currents needed to generate the desired torque. Field oriented control (FOC) uses the Clarke transform and a vector rotation to transform the motor winding currents into two quasi dc components, an $I_d$ component that reinforces or weakens the rotor field and an $I_q$ component that generates motor torque.

Two separate regulators control the $I_d$ and $I_q$ currents and a forward vector rotation transforms the current loop output voltages $V_d$ and $V_q$ into the two phase ac components ($V_\alpha$ and $V_\beta$). A DC bus compensation function adjusts the modulation index as a function of the dc bus voltage to reject dc bus ripple and improve current loop stability. The Space Vector Pulse Width Modulator (SVPWM) generates the three phase power inverter switching signals based on the $V_\alpha$ and $V_\beta$ voltage inputs.

Typically, the $I_q$ controller input is the torque reference from the speed controller and the $I_d$ reference current is set to zero. However, above a certain speed, known as the base speed, the inverter output voltage becomes limited by the dc bus voltage. In this situation, the field weakening controller generates a negative $I_d$ to oppose the rotor magnet field that reduces the winding back EMF. This enables operation at higher speeds but at a lower torque output. The controller includes a compensator that adjusts the $I_d$ current to maintain the motor voltage magnitude within the bus voltage limit.

The rotor magnet position estimator consists of a flux estimator and PLL.  Flux is calculated based on feedback current, estimated voltages (based on dc bus feedback voltage and modulation index) and motor parameters (inductance and resistance). The output of the flux estimator represents rotor magnet flux in the Alpha-Beta (stationary orthogonal frame, u-phase aligned with Alpha) two-phase components. The angle and frequency phase locked loop (PLL) estimates the flux angle and speed from the rotor magnet flux vector in Alpha-Beta components. The vector rotation calculates the error between the rotor flux angle and the estimated angle. The PI compensator and integrator in the closed loop path force angle and frequency estimate to track the angle and frequency of the rotor flux. The motor speed is derived from the rotor frequency according to the number of rotor poles.

When driving an interior permanent magnet (IPM) motor the rotor saliency can generate a reluctance torque component to augment the torque produced by the rotor magnet. When driving a surface magnet motor, there is zero saliency ($L_d = L_q$) and $I_d$ is set to zero for maximum efficiency. In the case of IPM motor which has saliency ($L_d < L_q$) a negative $I_d$ will produce positive reluctance torque. The most efficient operating point is when the total torque is maximized for a given current magnitude.

### 2.1.1 Variable Scaling

The MCE implements the control algorithms on a fixed point CPU core where physical voltage and current signals are represented by fixed point integers. The MCE algorithm uses appropriate scaling for control parameters and variables to optimize the precision of the motor and PFC control calculations. While all control parameters and variables are stored as integers the MCE Ecosystem tools support display of control variables and parameter settings as real numbers scaled to physical values.

The Figure 2 below describes the scaling used in different domains. In the hardware reference frame, current and voltage measurements are scaled according to the input circuit scaling and the resolution of the ADC. The α-β and d-q quasi-dc voltages are defined by the PWM modulator resolution and inverter DC bus voltage capability. There is different motor current scaling in the AC and control reference frames. The α-β current scaling is defined by the measurement scaling while the d-q scaling is defined by the motor current ratings. The motor speed scaling is defined by the application requirements. There are three different time scales, the Hardware timing is defined by the IC peripheral clock; the sampling and control timing is set by the PWM frequency while the Application reference frame timing is fixed. The full set of variable scaling will be described later in this document. All control parameter scaling is derived from the control variable and time scaling for the relevant reference frame.

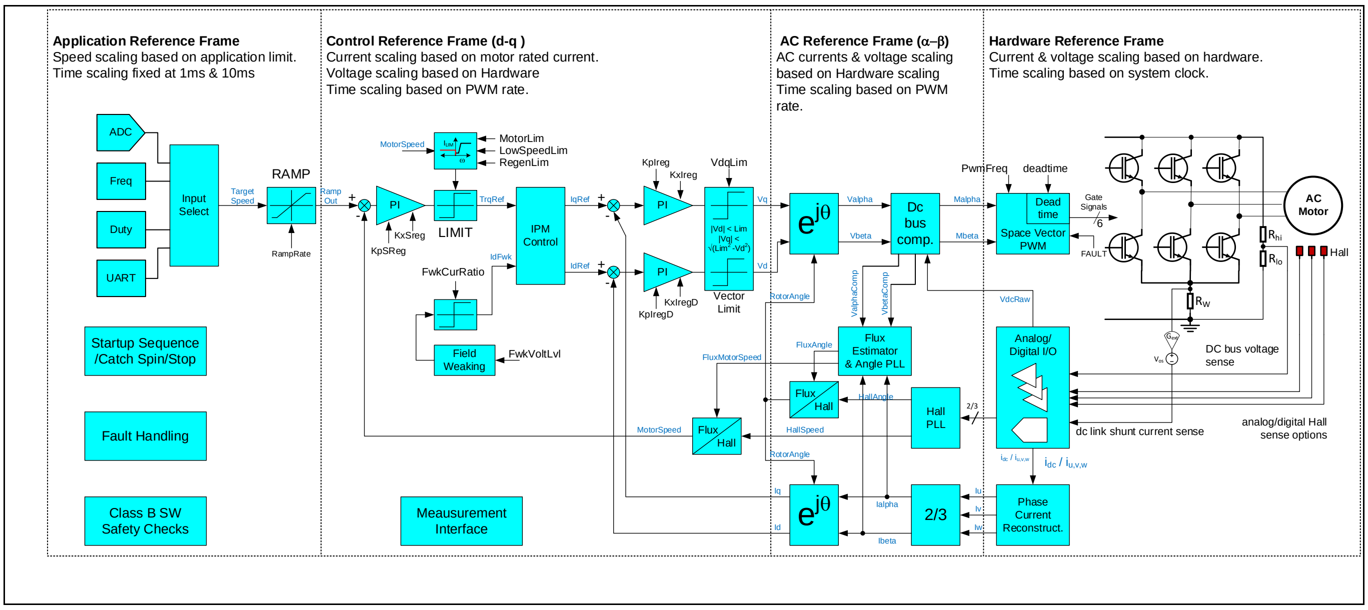

Application Reference Frame
Speed scaling based on application limit.
Time scaling fixed at 1ms & 10ms
Control Reference Frame (d-q )
Current scaling based on motor rated current.
Voltage scaling based on Hardware
Time scaling based on PWM rate.
AC Reference Frame (α–β)
AC currents & voltage scaling based on Hardware scaling
Time scaling based on PWM rate.
Hardware Reference Frame
Current & voltage scaling based on hardware.
Time scaling based on system clock.
ADC
Freq
Duty
UART
Input Select
RAMP
Startup Sequence /Catch Spin/Stop
Fault Handling
Class B SW Safety Checks
Measurement Interface
PI
LIMIT
IPM Control
Field Weaking
Vector Limit
Dc bus comp.
Flux Estimator & Angle PLL
Hall PLL
Dead time
Space Vector PWM
Analog/ Digital I/O
Phase Current Reconstruct.
AC Motor
DC bus voltage sense
dc link shunt current sense
analog/digital Hall sense options

Figure 2 Scaling Domains for Control Variables and Parameters

### 2.1.2 State Handling

The control software has a number of different operating states to support the various transient operating conditions between drive power-up and stable running of the motor under closed loop sensorless control. These include preparation of the drive for starting, running the motor before the flux estimator reaches stable operation, starting a motor that is already running and handling fault conditions. The Motion Control Engine includes a built-in state machine that takes care of all state-handling for starting, stopping and performing start-up. A state machine function is executed periodically (by default, every 1ms). In total there are 12 states; each state has a value between 0-12, the current state of the sequencer is stored in "Motor_SequencerState" variable.

Table 3 State Description and Transition

| State No | Sequence State | State Functionality | Transition Event | Next Sequence State |
|---|---|---|---|---|
| 0 | IDLE | After the controller power up, control enters into this state. If there is no valid parameter block, sequencer stays in this state. | Parameters are loaded successfully. | STOP |
| 1 | STOP | Wait for start command. Current and voltage measurement are done for protection. | Current Amplifier offset calculation is not done. | OFFSETCAL |
| | | | Start Command. | BTSCHARGE |
| 2 | OFFSETCAL | Offset calculation for motor current sensing input. This state takes 8192 PWM cycles. | Current offset calculation completed. | STOP |
| 3 | BTSCHARGE | Boot strap capacitor pre-charge. Current and voltage measurement are done for protection. | Bootstrap capacitor charge completed. | CATCHSPIN<br>ANGLESENSE<br>PARKING<br>OPENLOOP |

| State No | Sequence State | State Functionality | Transition Event | Next Sequence State |
|---|---|---|---|---|
| | | | | MOTORRUN (Flux/Hall/Hybrid) |
| 4/ 10/ 11/ 12 | MOTORRUN (Flux/Hall/Hybri d, Openloop) | Normal motor run mode in flux/hall/hybrid based rotor angle estimation. | Stop Command | STOP |
| 5 | FAULT | If any fault detected, motor will be stopped (if it was previously running) and enter FAULT state from any other state. | In UART control mode, Fault clear command by writing 1 to "FaultClear" variable | STOP |
| | | | In Frequency/ Duty/ VSP input control modes, after configured time. | STOP |
| 6 | CATCHSPIN | Flux estimator and flux PLL are running in order to detect the rotor position and measure the motor speed of free running motor. Speed regulator is disabled and the Id & Iq current commands are set to 0. | Measured absolute motor speed is above threshold ("DirectStartThr" parameter) | MOTORRUN (Flux/Hall/Hybrid) |
| | | | Measured absolute motor speed is less than threshold ("DirectStartThr" parameter) | ANGLESENSING<br>PARKING<br>OPENLOOP<br>MOTORRUN (Flux/Hall/Hybrid) |
| | | | | |
| 7 | PARKING | Parking state is to align the rotor to a known position by injecting a linearly increased current. The final current amplitude is decided by low speed current limit. Total time duration of this state is configured by "ParkTime" register. | Parking completed | ANGLESENSE<br>PARKING<br>OPENLOOP<br>MOTORRUN (Flux/Hall/Hybrid) |
| 8 | OPENLOOP | Move the rotor and accelerate from speed zero to MinSpd by using open loop angle. Flux estimator and flux PLL are executed in this state in order to provide smooth transition to MOTOR_RUN state. Speed acceleration of the open loop angle is configured by "OpenLoopRamp" register. | Speed reaches "MinSpd" register value | MOTOR_RUN (Flux/Hall/Hybrid) |

| State No | Sequence State | State Functionality | Transition Event | Next Sequence State |
|---|---|---|---|---|
| 9 | ANGLESENSING | Measure the initial rotor angle. The length of each sensing pulse is configured by "IS_Pulses" (in PWM cycles) register. | Angle Sensing completed | PARKING<br>OPENLOOP<br>MOTORRUN (Flux/Hall/Hybrid) |
| 13 | STANDBY | The MCE lowers standby power consumption by reduceing the CPU clock and switching off some of the controller peripherals. | System is in a stopped state, there are no faults and a configured delay time has expired | STOP |

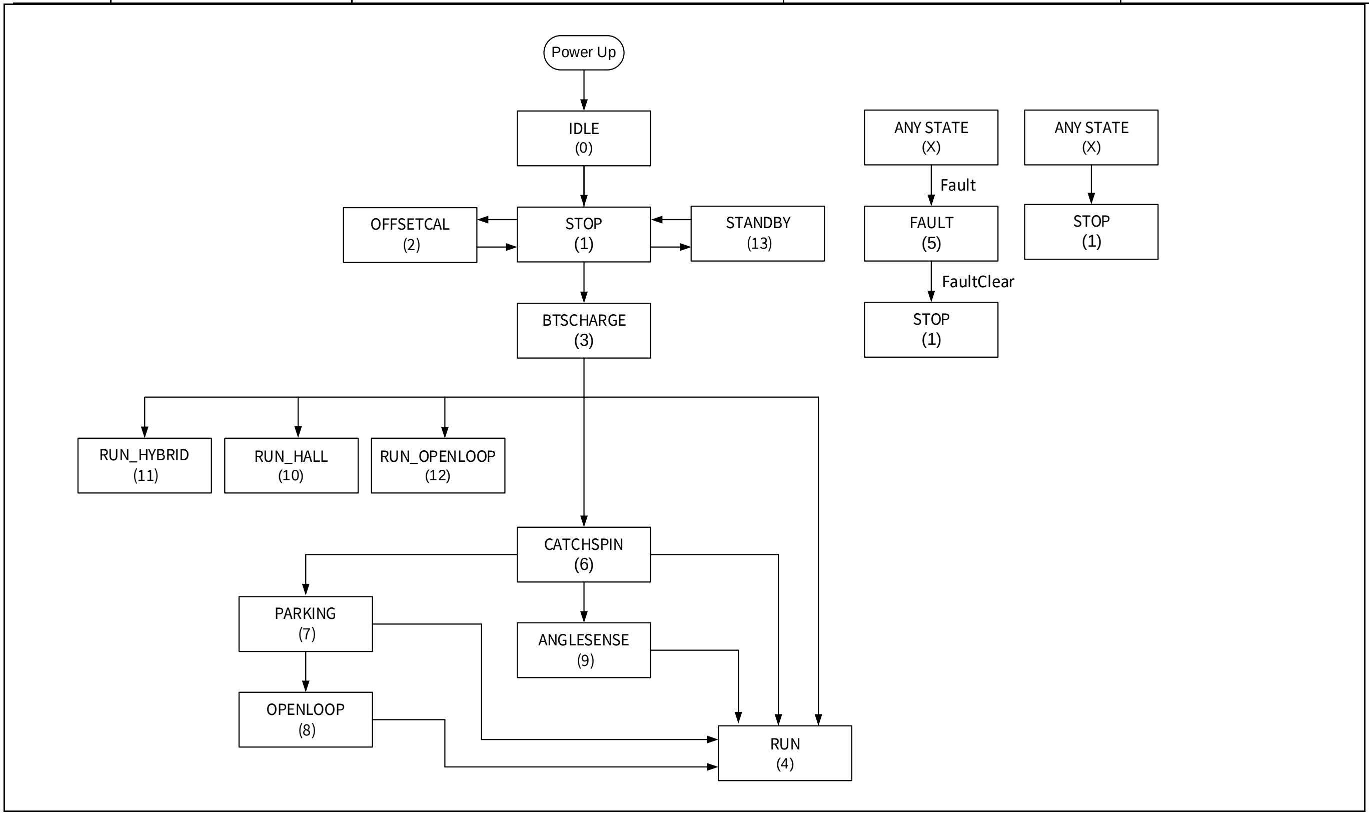


**Figure 3 State Handling and Start Control Flow Chart**

## 2.1.3 Current Sensing Offset Measurement

The current sensing offset is measured by the MCE during OFFSETCAL state when the inverter is not switching and there is no motor phase current flowing through the shunt resistor(s). During the OFFSETCAL state, the MCE measures the current sensing offset values at IU pin for phase U, at IV pin for phase V, and at IW pin for phase W for leg shunt configuration or on ISS pin for single shunt configuration every motor PWM cycle for a configurable number of cycles ($N = 2^{OffsetSample}$, where $OffsetSample$ is the value of the parameter 'OffsetSample'. At the end of the OFFSETCAL state, the N number of current offset measurement values for each phase are averaged and stored in variables 'IOffset0', 'IOffset1' and IOffset2' respectively.

The duration of OFFSETCAL state $T_{Offset_Cal}$can be estimated using the following equation: $T_{Offset\_Cal} = \frac{Fast\_Control\_Rate \times 2^{OffsetSample}}{F_{PWM}}$. By default, $OffsetSample = 13$, so OFFSETCAL states takes 8192 PWM cycles.

Customers have the choice to enable/disable current offset calculations after fault clears. If it's enabled, the state will go into current offset calculation again after fault occurs. Offset calibration time can be configured using "OffsetSample" parameter. If it's disabled, the current offset calculation is not executed anymore and will be held as the initial value. Offset Calibration is retriggeed by setting "APP_MOTOR0.Command" variable bit 1 while system is in stop state.

### 2.1.4 Bootstrap Capacitor Charge

Bootstrap capacitors are charged by turning on all three low side switches. The charging current is limited by the built-in pre-charge control function.

Instead of charging all low side devices simultaneously, the gate pre-charge control will schedule an alternating (U, V, W phase) charging sequence. Each phase charges the bootstrap capacitor for a duration of 1 / 3rd of the PWM cycle so each capacitor charge time is 1/3rd of the total pre-charge time.

Figure 4 illustrates the PWM signal during bootstrap capacitor charge state.

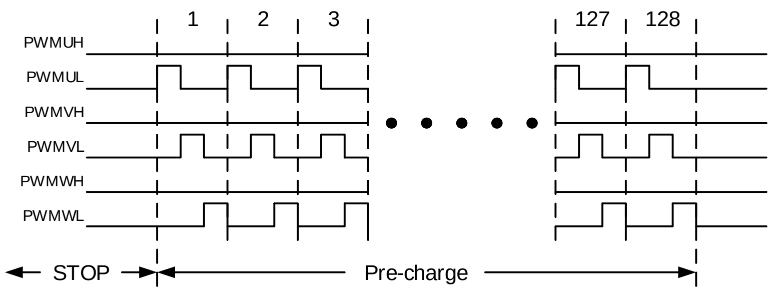


**Figure 4 Bootstrap Capacitor Pre-charge**

Total pre-charge time for each phase can be calculated from: $T_{Charge} = \frac{BtsChargeTime}{3*F_{PWM}}$ where the parameter 'BtsChargeTime' is the number of pre-charge PWM cycles.

For example, if PWM frequency is 10 kHz, and BtsChargeTime is 100, then the pre-charge time of each phase will be: $\frac{100}{3*10000} = 3.333\ (\text{ms})$.

### 2.1.5 Voltage measurement

The measurement of the DC bus voltage of the inverter board is required for voltage protection and DC bus voltage compensation. The voltage is measured at every PWM cycle. DC bus voltage of the inverter is measurement via a voltage divider circuit using 12-bit ADC. Measured DC bus voltage is internally represented in 12 bit format.

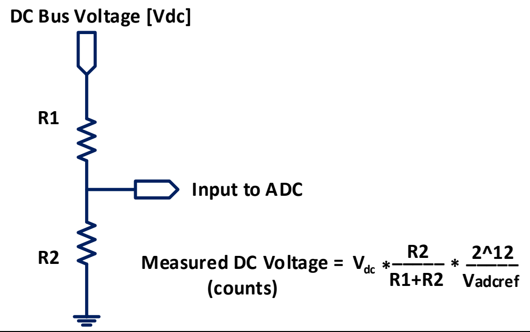

DC Bus Voltage [Vdc]
R1
Input to ADC
R2
Measured DC Voltage (counts) = $V_{dc} * \frac{R2}{R1+R2} * \frac{2^{\wedge}12}{Vadcref}$

**Figure 5 DC Bus voltage feedback signal path**

Example: R1 = 2MΩ, R2 = 13.3kΩ, $V_{adcref}$= 3.3V and $V_{dc}$ = 320V; Measured DC bus voltage = 2623 counts

***Attention:*** ***In Solution Designer R1 and R2 values shall be configured as per actual hardware used. Wrong configuration may lead to wrong under voltage/over voltage/ Critical over voltage fault or over voltage/under voltage/ critical over voltage conditions may not be detected correctly.***

### 2.1.6 DC Bus Compensation

DC bus voltage typically has high frequency ripple as well as 2 times AC input line frequency ($F_{line}$) ripple. The low frequency ripple is dominant due to limited size of DC bus capacitors. The instantaneous DC bus voltage is part of the motor current control loop gain. Thus, if the current loop bandwidth is not high enough, then there is not enough loop gain at 2 x $F_{line}$ frequency. As a result, the current loop won't be able to adjust the Modulation Index (MI) accordingly to ensure stable inverter output voltage. The resulting motor phase current would inevitably be modulated by 2 x $F_{line}$ frequency DC bus voltage ripple.

The MCE provides a DC bus voltage feedforwarding function to compensate for the effect of the DC bus voltage variation on the current control loop gain, so that the actual MI is not affected by the DC bus voltage ripple.

As shown in the following Figure 6, if DC bus compensation is enabled, Valpha and Vbeta, that are the 2 orthogonal components of the desired inverter output voltage, are adjusted by a factor of the ratio between 50% of DC bus full range voltage to the instantaneous DC bus voltage. The adjusted results, Malpha and Mbeta, are the 2 orthogonal components of the desired output voltage vector, based on which the SVPWM block generates the three phase PWM switching signals. Additionally, the vector voltage limit (parameter 'VdqLim') is also adjusted inversely by the DC bus compensation factor to make full inverter voltage available. If DC bus compensation is disabled, Valpha and Vbeta are directly coupled with Malpha and Mbeta without any additional adjustment. Flux estimator parameters are scaled based on 50% of DC bus full range voltage. So If DC bus compensation is disabled, it is required to compensate Voltage alpha (ValphaComp) and Voltage beta (VbetaComp) components used in flux estimator based on DC bus voltage. If DC bus compensation is enabled, no compensation required in voltage alpha and voltage beta components used in flux estimator. Motor voltage (Vdq) is calcuated based on Vd and Vq values. Calculation is done every 1ms.

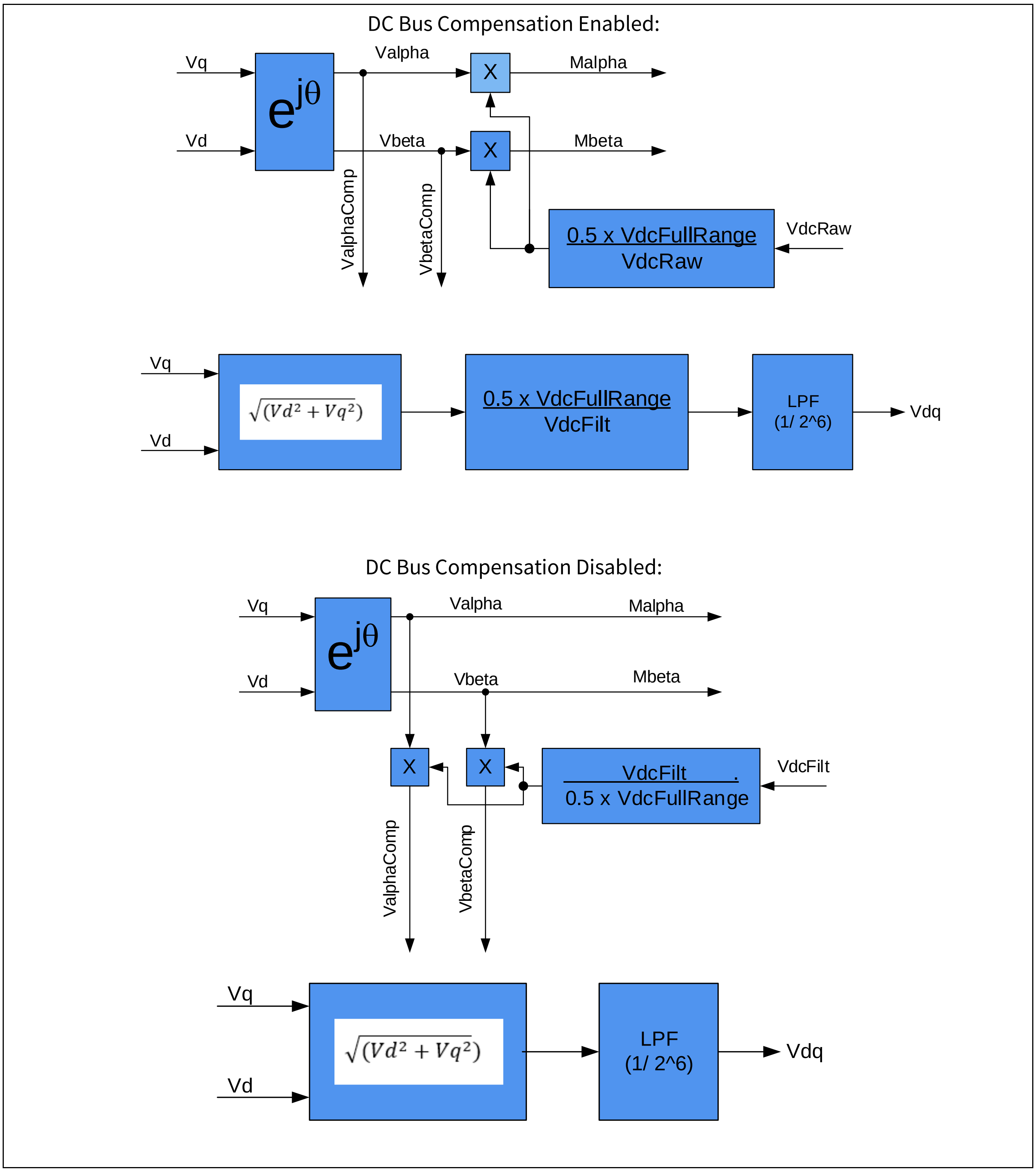


**Figure 6 DC Bus Compensation Functional Diagram**

The DC bus full range voltage is the maximum DC bus voltage that the ADC can sample up to given a specified voltage divider for DC bus voltage sensing. Referring to Figure 5, it can be calculated using the following equation.

$$V_{DCFullRange} = V_{ADC\_REF} \times \frac{R1 + R2}{R2}$$

DC bus compensation function can be enabled by setting bit [0] of 'SysConfig' parameter.

With DC bus compensation function disabled, the actual MI can be estimated using the variable 'MotorVoltage' following this equation:

$$MI = \frac{MotorVoltage}{4974}.$$

With DC bus compensation function enabled, the actual MI can be estimated using the variables 'MotorVoltage' and 'VdcRaw' following this equation:

$$MI = \frac{MotorVoltage \times \frac{2048}{VdcRaw}}{4974} \times 100\%.$$

## 2.1.7 Current Measurement

In order to implement sensorless field oriented control, it is crucial to measure the motor winding currents precisely. Motor phase current values are used for current control and flux estimator. Current is measured at every PWM cycle. The following Table 4 summarizes all the current measurement configurations supported by the MCE. The details of each configuration and its relevant PWM schemes will be described in the following sections.

Table 4 **Current Measurement Configurations & PWM Schemes**

| Current Measurement Configurations | Needed Number of Shunt Resistors | PWM Schemes |
|---|---|---|
| Leg shunt | 3 | Center aligned symmetrical PWM |
| Single shunt | 1 | Center aligned asymmetrical PWM<br>- Phase shift PWM<br>- Low noise phase shift PWM |

The internal amplifiers are used for current measurement, no external op-amp is required. The gain of the internal amplifier can be configured using Solution Designer.

The following Figure 7 shows the details of the motor phase current feedback signal path.

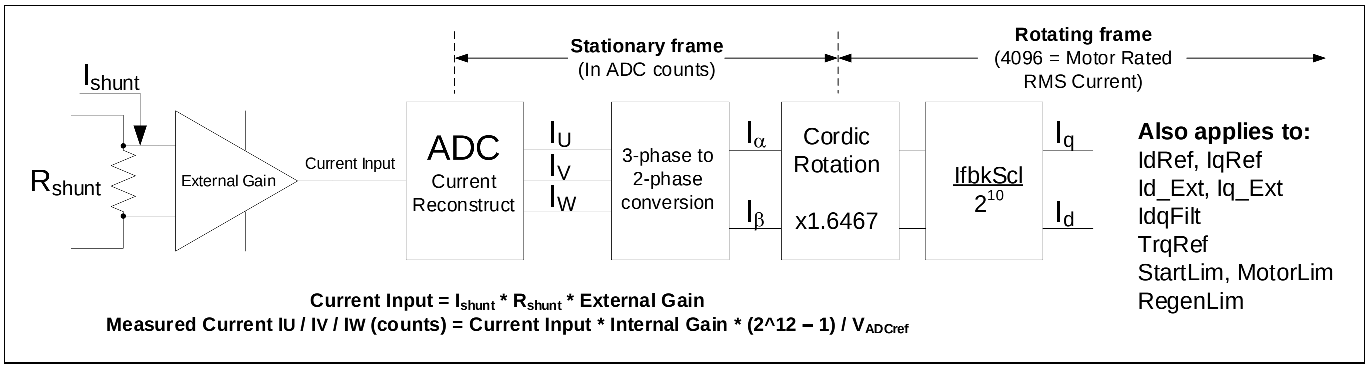


Figure 7 **Motor current feedback signal path (TminPhaseShift ≠ 0)**

***Attention:*** ***In Solution Designer current input value shall be configured as per actual hardware used. Wrong configuration may lead to wrong over current fault or over current conditions may not be detected correctly.***

### 2.1.7.1 Leg Shunt Current Measurement

Leg-shunt current sensing configuration uses 3 shunt resistors to sense the 3 inverter phases as shown in the following Figure 8. For 2 phase current sensing, the MCE only senses phase U and phase V current, and phase W current is calculated assuming the sum of the three phase current values is zero. For 3 phase current sensing, the MCE senses all three phase currents.The motor phase current would flow through the shunt resistor only when the low-side switch is closed. Accordingly, the MCE chooses to sense the motor phase current values during the zero vector [000] time in the vicinity of the start of a PWM cycle. Accordingly, a minimum duration of zero vector [000] ($T_{GB\_min}$) as shown in Figure 9 is needed to ensure proper sampling of motor phase current values. This minimum duration can be specified by using the parameter 'PwmGuardBand' following this equation $T_{GB_{min}} = PwmGuardBand \times 10.417ns$. Thanks to this minimum duration of zero vector [000], the ON time of each phase PWM signal would never be longer than $T_{PWM} - T_{BGmin}$.

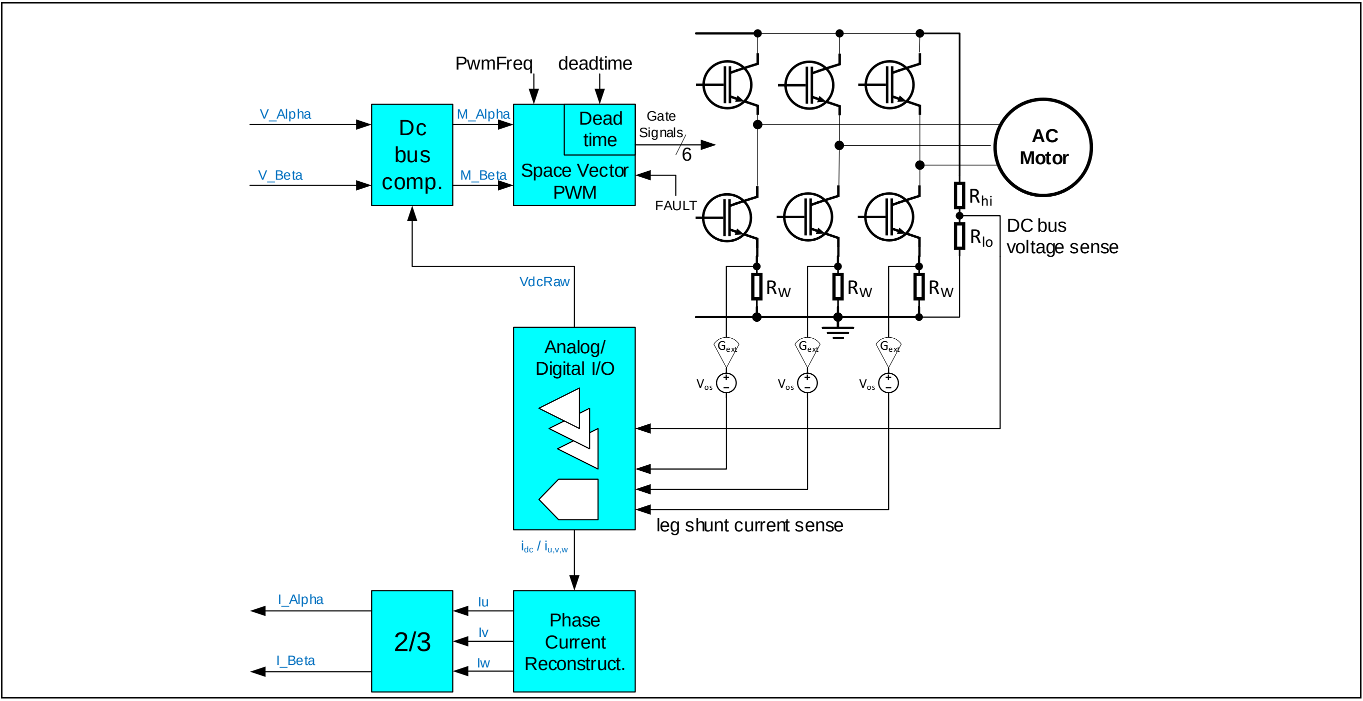


**Figure 8 Typical Circuit Diagram for Leg Shunt Current Measurement Configuration**

The current sensing timing for leg shunt configuration is shown in the following Figure 9. In each PWM cycle, $T_a$ and $T_b$ refer to the 2 active vector time respectively, and 2 x $T_0$ refers to the total zero vector ([000], [111]) time. The duration of zero vector [111] is the same as that of zero vector [000]. The first current sensing point (CS1) is the time to sense phase U current, and it occurs $T_{SD} + \frac{4}{f_{PCLK}} = T_{SD} + 41.668ns$ after the start of a PWM cycle. $T_{SD}$ is the needed ADC sampling delay time, and it can be positive or negative as required. $T_{SD}$ can be configured by using the parameter 'SHDelay' following this equation $T_{SD} = SHDelay \times 10.417ns$. The ADC sampling time $T_{sample}$ is about $0.333\mu s$, and the ADC conversion time $T_{conversion}$ is about $0.854\mu s$. The second current sensing point (CS2) is the time to sense phase V current. CS2 occurs right after the completion of the CS1 sampling and conversion operation.

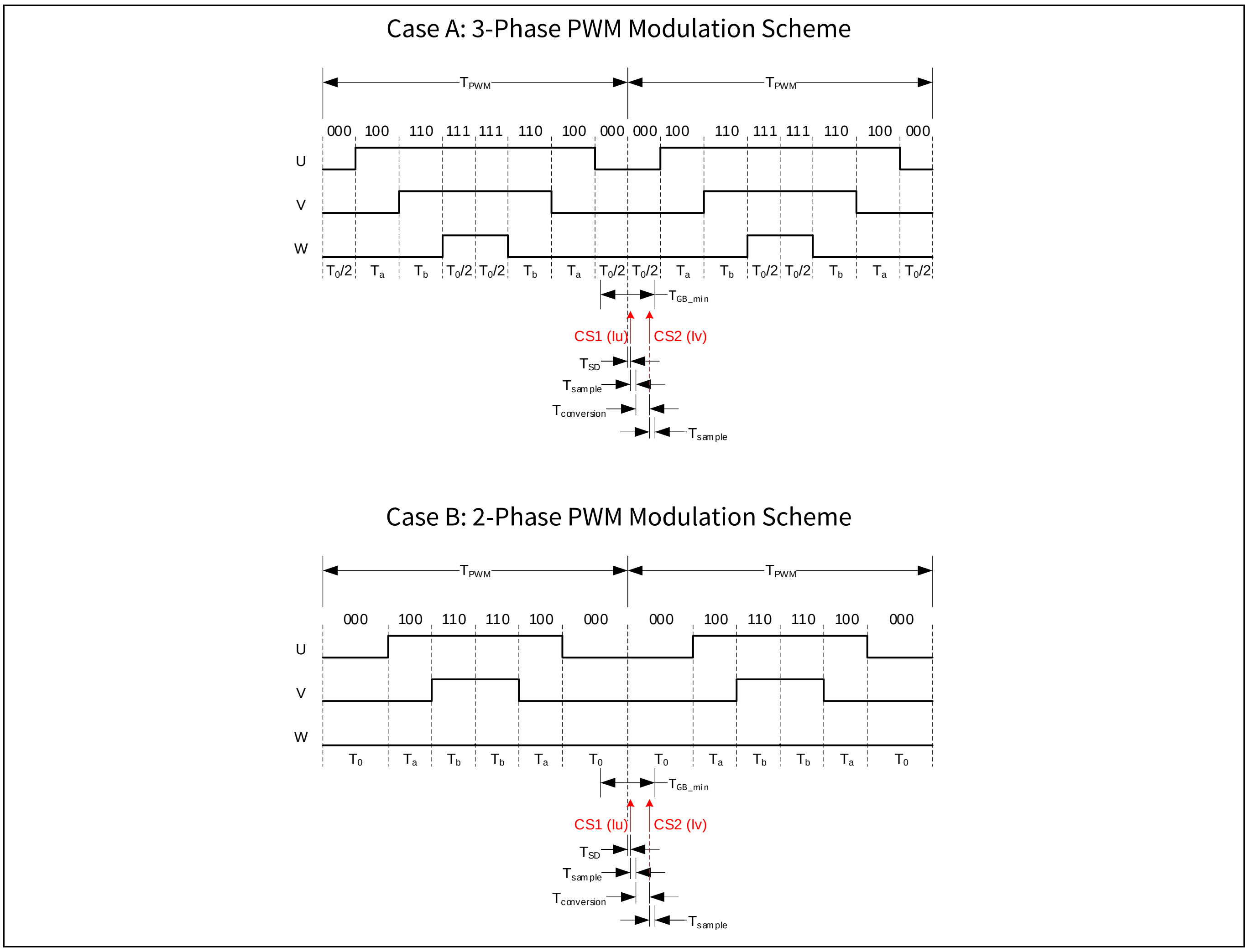

Case A: 3-Phase PWM Modulation Scheme
T_PWM
000 100 110 111 111 110 100 000 000 100 110 111 111 110 100 000
U
V
W
T0/2 Ta Tb T0/2 T0/2 Tb Ta T0/2 T0/2 Ta Tb T0/2 T0/2 Tb Ta T0/2
T_GB_min
CS1 (Iu)
CS2 (Iv)
T_SD
T_sample
T_conversion
Case B: 2-Phase PWM Modulation Scheme
000 100 110 110 100 000 000 100 110 110 100 000
T0 Ta Tb Tb Ta T0 T0 Ta Tb Tb Ta T0

**Figure 9 Leg Shunt Configuration Current Sensing Timing Diagram**

### 2.1.7.2 Single Shunt Current Measurement

Single-shunt current sensing configuration uses only one shunt resistor to sense the DC link current as shown in the following Figure 10. It is often used for the sake of cost advantage. With single-shunt configuration, only DC link current can be sampled by the MCE, and the information of motor phase current can be extracted from DC link current only when the active (non-zero) vectors are being applied during each PWM cycle. Two different active vectors are applied during each PWM cycle, and the DC link current during each active vector time represents some specific motor phase current depending on sector information. The third motor phase current value can be calculated assuming the sum of the three phase current values is zero.

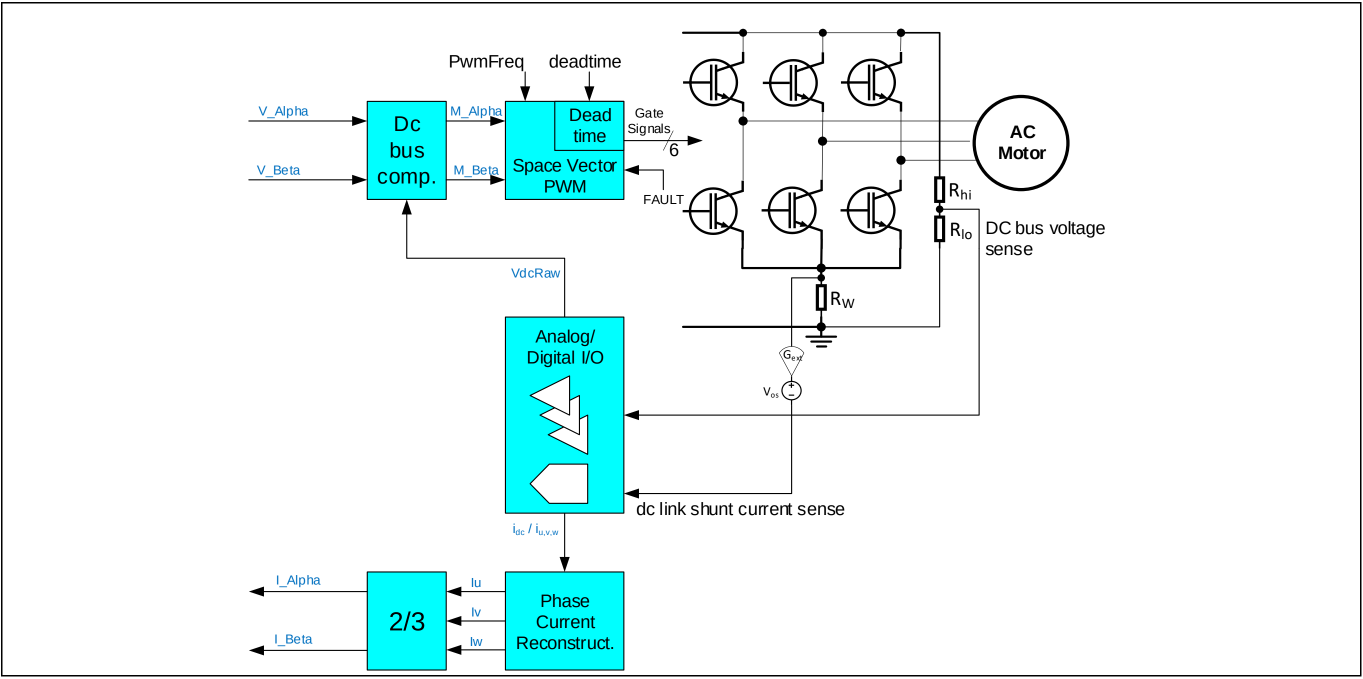


**Figure 10 Typical Circuit Diagram for Single Shunt Current Measurement Configuration**

The current sensing timing for single shunt configuration is shown in the following Figure 11. The first current sensing point (CS1) is for sensing phase current during one of the active vector time (In the case as shown in Figure 11, the sensed current is negative phase W current during the active vector [110] time). CS1 occurs $\frac{T_b}{2} + T_{SD}$ after the start of this active vector time (vector [110] in the case as shown in Figure 11). The second current sensing point (CS2) is for sensing phase current during the other active vector time (In the case as shown in Figure 11, the sensed current is phase U current during the active vector [100] time). CS2 occurs $\frac{T_a}{2} + T_{SD}$ after the start of this active vector time (vector [100] in the case as shown in Figure 11). $T_{SD}$ is the needed ADC sampling delay time, and it can be positive or negative as required. $T_{SD}$ can be configured by using the parameter 'SHDelay' following this equation $T_{SD} = SHDelay \times 10.417ns$.

If the desired CS1 or CS2 point is estimated to occur after the end of the PWM cycle following the above-mentioned equations, then the actual CS1 or CS2 point is adjusted to occur just before the end of this PWM cycle by the MCE to ensure the latest current sample values are available at the beginning of the following PWM cycle when the FOC calculation begins to execute.

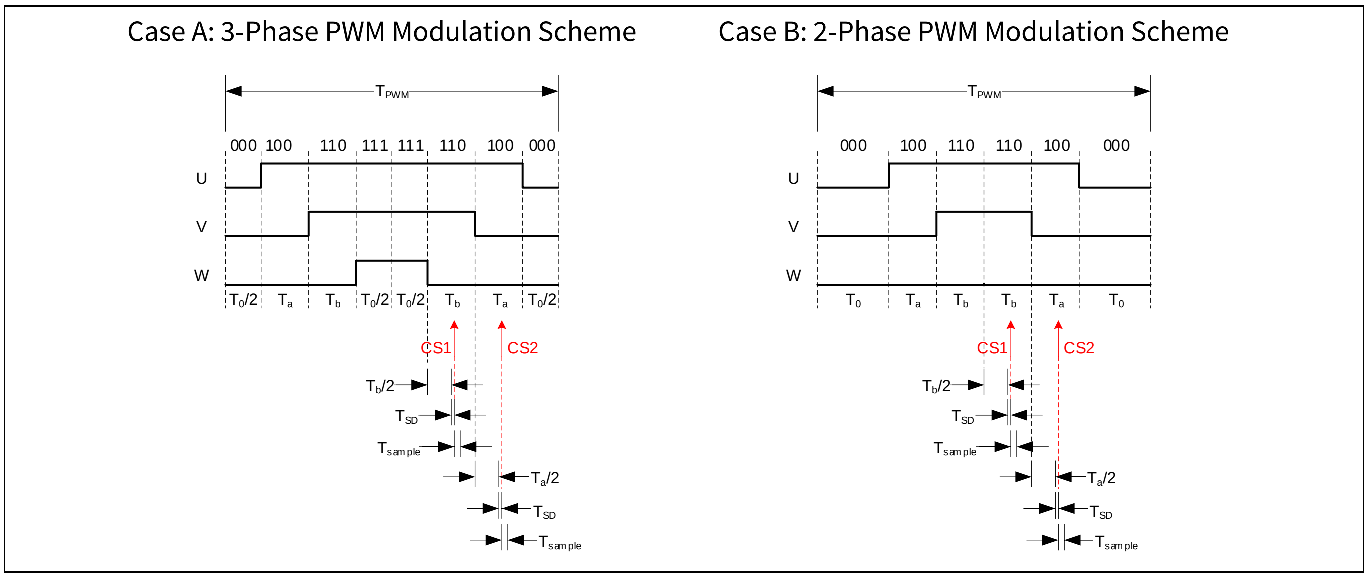

Case A: 3-Phase PWM Modulation Scheme
Case B: 2-Phase PWM Modulation Scheme
$T_{PWM}$
000 100 110 111 111 110 100 000
U
V
W
$T_0/2$ $T_a$ $T_b$ $T_0/2$ $T_0/2$ $T_b$ $T_a$ $T_0/2$
CS1
CS2
$T_b/2$
$T_{SD}$
$T_{sample}$
$T_a/2$
000 100 110 110 100 000
$T_0$ $T_a$ $T_b$ $T_b$ $T_a$ $T_0$

**Figure 11 Single Shunt Configuration Current Sensing Timing Diagram**

### 2.1.7.2.1 Minimum Pulse Width PWM

In single shunt reconstruction method, the current through one of the phases can be sensed across the shunt resistor during each active vector time. However, under certain operating conditions when the desired voltage vector is at sector cross-over regions or when the magnitude of the desired voltage vector is low (low modulation index), the duration of one or both active vectors is too short to guarantee reliable sampling of winding current data. These operating conditions are shaded in the space vector diagram shown in Figure 12. In the example shown in Figure 12, the active vector [110] time $T_b$ is not long enough to ensure reliable current sensing.

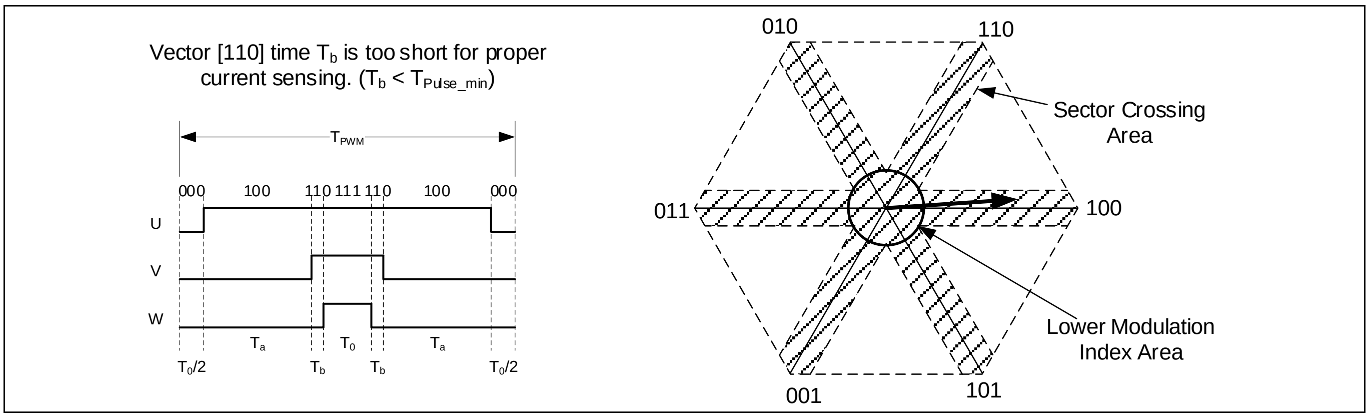

Vector [110] time $T_b$ is too short for proper current sensing. ($T_b < T_{Pulse\_min}$)
$T_{PWM}$
000 100 110 111 110 100 000
U
V
W
$T_a$ $T_0$ $T_a$
$T_0/2$ $T_b$ $T_b$ $T_0/2$
010
110
011
100
001
101
Sector Crossing Area
Lower Modulation Index Area

**Figure 12 Narrow pulse limitation of single shunt current sensing**

In order to guarantee reliable sampling of the winding current, a minimum pulse width limit ($T_{Pulse\_min}$) is imposed for each active vector in a PWM cycle. For an optimal control performance in this mode, 'SHDelay' parameter must be tuned per actual application hardware. This minimum pulse width restriction leads to output voltage distortion at lower modulation index or when the desired voltage vector is transitioning from one sector to another, because there is a difference between the target output voltage and the actual output voltage. This voltage distortion may cause audible noise and degradation of control performance, especially at lower speed. The shaded regions in the space vector diagram shown in Figure 12 mark the areas where output voltage distortion is introduced. Figure 13 illustrates the resulting distortion when the desired voltage vector is transitioning from one sector to another. As shown in Figure 13, the active vector [101] and [110] time $T_b$ is extended to $T_b' = T_{Pulse\_min}$ to accommodate the current sensing requirement during the sector crossing time.

The current sensing timing for single shunt configuration with minimum pulse width PWM scheme is the same as shown in Figure 11.

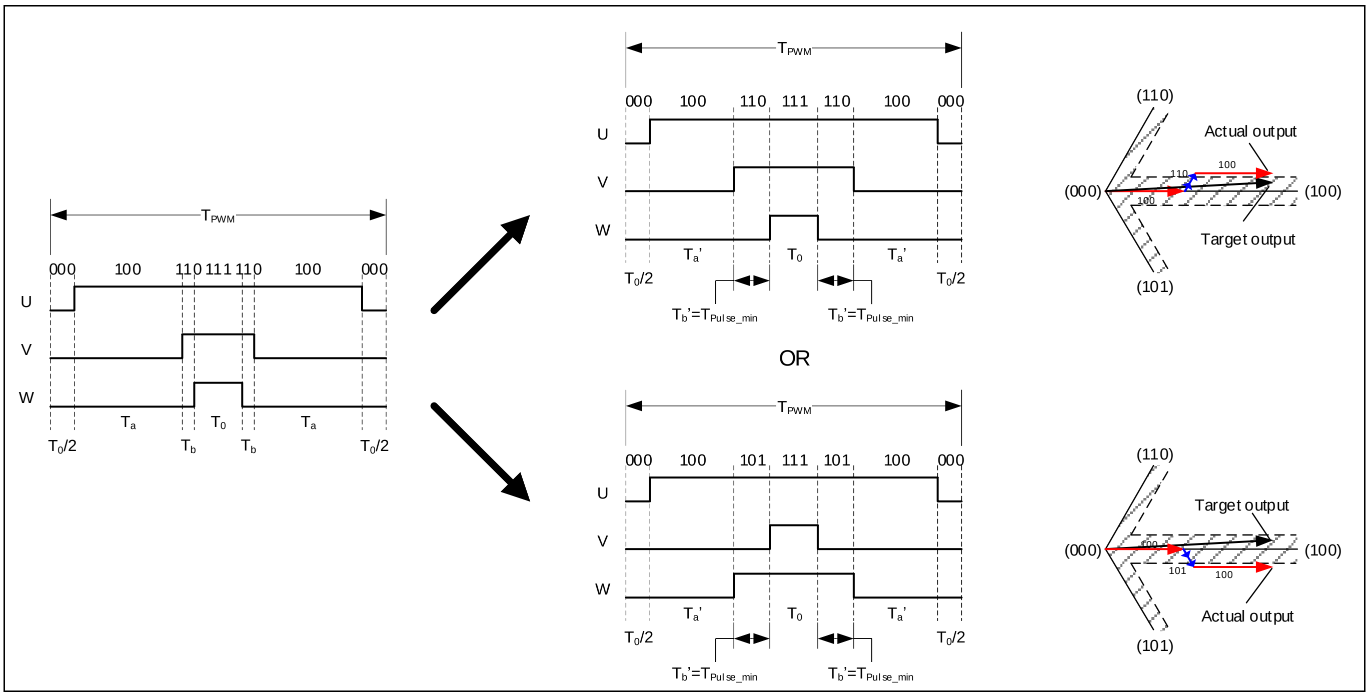


**Figure 13 Minimum pulse PWM scheme limitation**

### 2.1.7.2.2 Phase Shift PWM

In order to eliminate the minimum pulse limitation, MCE provides an option of phase shift PWM scheme. With phase shift PWM scheme, the output of each PWM is not always center aligned. A minimum active vector time ($T_{PSmin}$) is desired to ensure proper sampling of phase current. $T_{PSmin}$ can be specified by using the parameter 'TminPhaseShift' following this equation $T_{PSmin} = TminPhaseShift \times 10.417ns$. If the desired active vector time ($T_a$ or $T_b$) is longer than $T_{PSmin}$, then the PWM patterns remain intact. If the desired active vector time ($T_a$ or $T_b$) is less than $T_{PSmin}$, then the 3 phase PWM patterns are shifted accordingly to ensure that the actual active vector time at the falling edge is no less than the specified minimum active vector time $T_{PSmin}$.

As shown in Figure 14, the active vector [110] time at the falling edge is $T_{b2}$, and the active vector [100] time at the falling edge is $T_{a2}$. Given that the desired minimum active vector time $T_{PSmin} = 3 \times T_{a2}$, then $T_{a2}$ is not long enough while $T_{b2}$ is sufficient. Consequently, U phase PWM needs to be shifted right and V phase PWM needs to be shifted left to add enough time for active vector [100] ($T_{a2}' = T_{PSmin}$). It can be observed in Figure 14 case 1 that the PWM phase shift action equivalently adds an additional active vector [010] highlighted in red in Figure 14 that didn't exist originally. However, the impact of this additional vector is mitigated thanks to the extension of vector [100] time and the shrinking of vector [110] time.

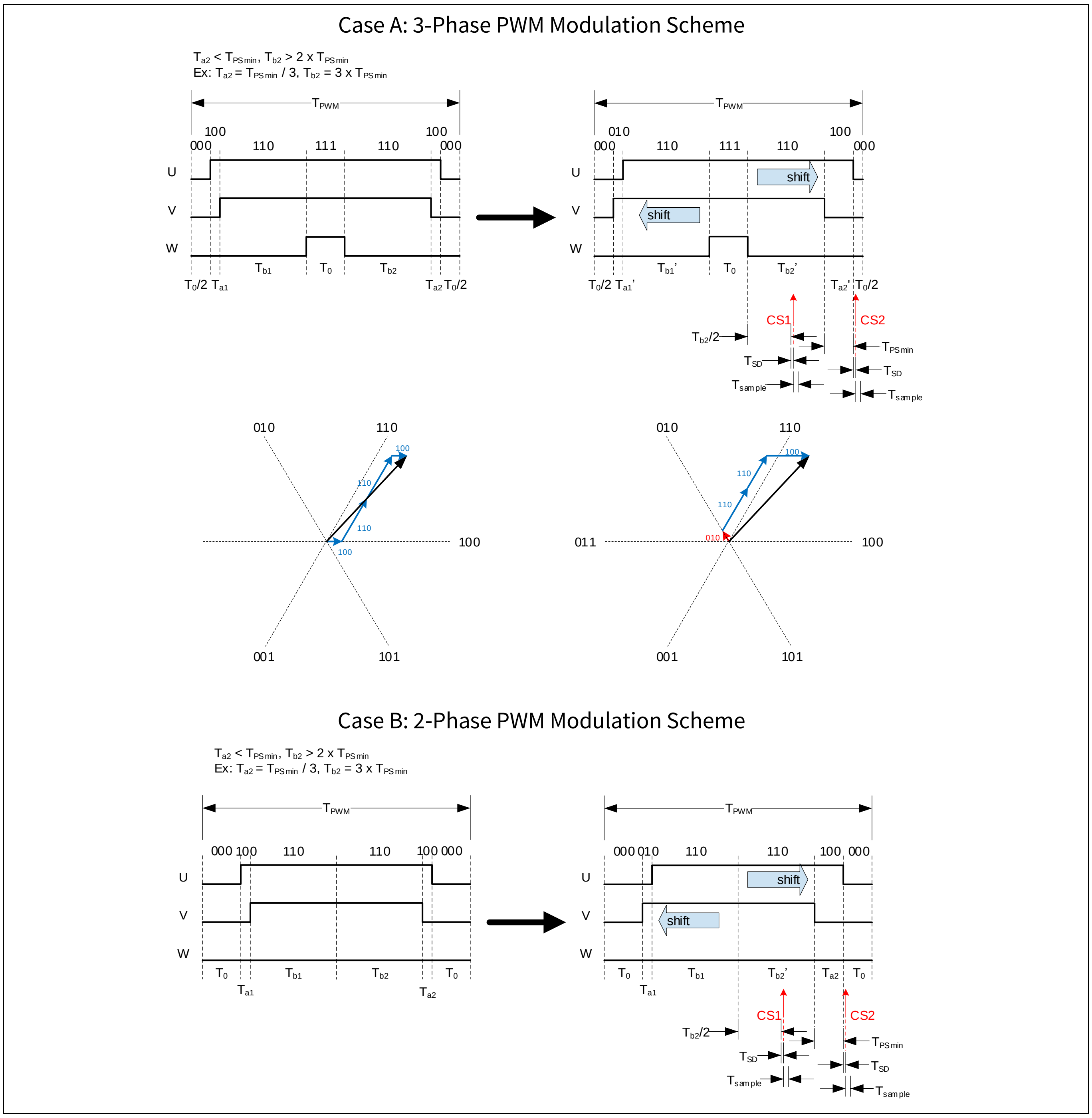

Case A: 3-Phase PWM Modulation Scheme
$T_{a2} < T_{PSmin}$, $T_{b2} > 2 \times T_{PSmin}$
Ex: $T_{a2} = T_{PSmin} / 3$, $T_{b2} = 3 \times T_{PSmin}$
$T_{PWM}$
U
V
W
shift
CS1
CS2
$T_{PSmin}$
$T_{SD}$
$T_{sample}$
$T_{b2}/2$
Case B: 2-Phase PWM Modulation Scheme
$T_{a2} < T_{PSmin}$, $T_{b2} > 2 \times T_{PSmin}$
Ex: $T_{a2} = T_{PSmin} / 3$, $T_{b2} = 3 \times T_{PSmin}$

**Figure 14 Single Shunt Configuration with Phase Shift PWM Scheme Current Sensing Timing Diagram (Case 1: $T_{a2} < T_{PSmin}$, $T_{b2} > 2 \times T_{PSmin}$)**

As shown in Figure 15, given that the desired minimum active vector time $T_{PSmin} = 3 \times T_{b2}$, then $T_{b2}$ is not long enough while $T_{a2}$ is sufficient. Consequently, W phase PWM needs to be shifted left to add enough time for active vector [110] ($T_{b2}' = T_{PSmin}$). It can be observed in Figure 15 the PWM phase shift action equivalently adds an additional active vector [101] highlighted in red in Figure 15 that didn't exist originally. However, the impact of this additional vector is mitigated thanks to the extension of vector [110] time and the shrinking of vector [100] time.

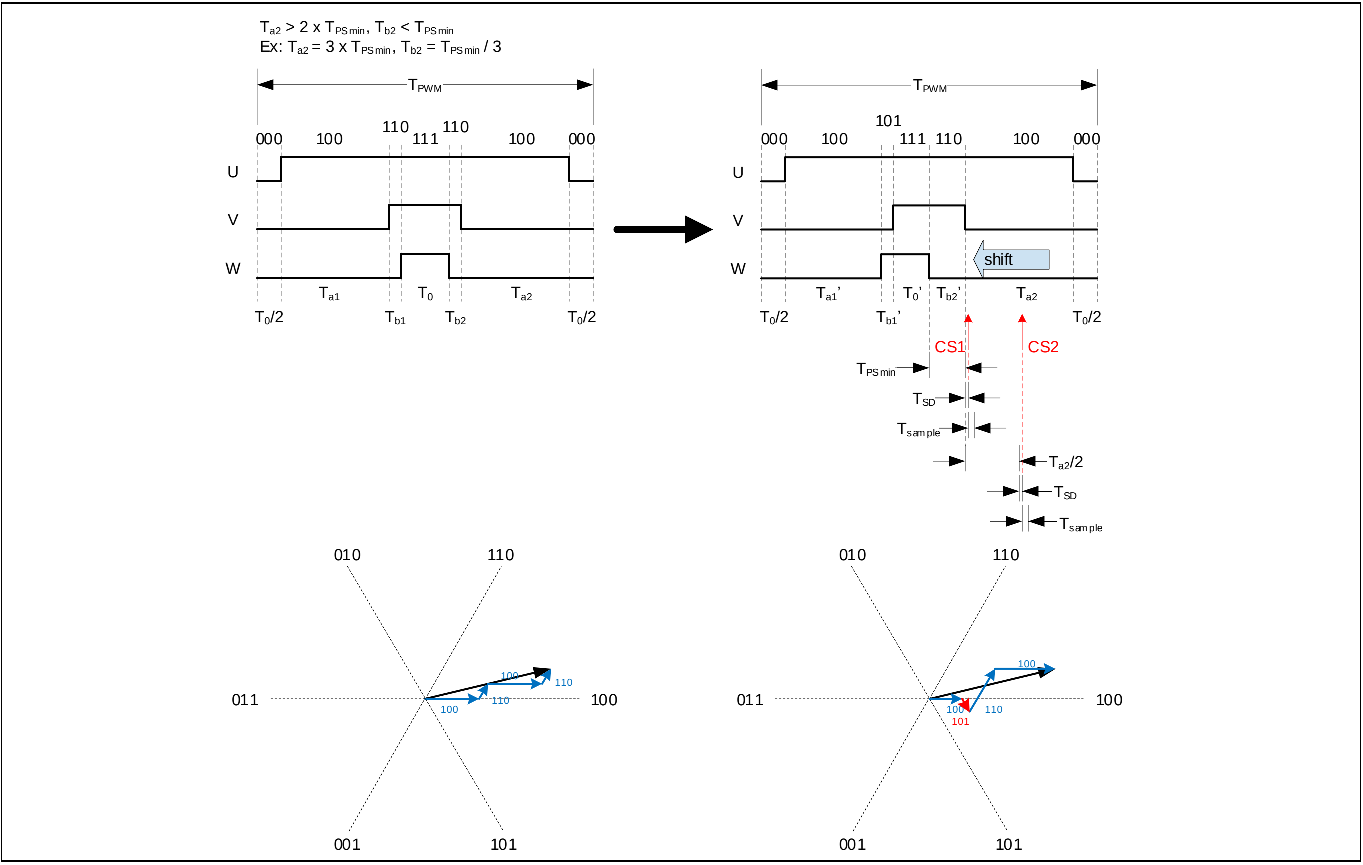

$T_{a2} > 2 \times T_{PSmin}$, $T_{b2} < T_{PSmin}$
Ex: $T_{a2} = 3 \times T_{PSmin}$, $T_{b2} = T_{PSmin} / 3$
$T_{PWM}$
U
V
W
shift
CS1
CS2
$T_{PSmin}$
$T_{SD}$
$T_{sample}$
$T_{a2}/2$
010
110
011
100
001
101

**Figure 15** **Single Shunt Configuration with Phase Shift PWM Scheme Current Sensing Timing Diagram (Case 2: $T_{a2} > 2 \times T_{PSmin}$, $T_{b2} < T_{PSmin}$)**

The current sensing timing for single shunt configuration with phase shift PWM scheme depends on the relationship between the active vector time ($T_a$ or $T_b$) and the desired minimum active vector time $T_{PSmin}$.

If $T_a$ or $T_b$ is more than 2 times $T_{PSmin}$, then the corresponding current sensing point occurs at the middle of that active vector time with a sampling delay time $T_{SD}$. Examples are $T_{b2}$ in Figure 14, and $T_{a2}$ in Figure 15. This is consistent with the current sensing timing described in Figure 11.

If $T_a$ or $T_b$ is within the range from $T_{PSmin}$ to 2 times $T_{PSmin}$, then the corresponding current sensing point occurs $T_{PSmin} + T_{SD}$ after the start of this active vector time. In the example shown in the following Figure 16, both $T_a$ and $T_b$ fall between $T_{PSmin}$ and $2 \times T_{PSmin}$. So, the CS1 occurs $T_{PSmin} + T_{SD}$ after the start of active vector [110] time, and the CS2 occurs $T_{PSmin} + T_{SD}$ after the start of active vector [100] time.

If $T_a$ or $T_b$ is less than $T_{PSmin}$, then necessary phase shift is applied to ensure desired minimum active vector time $T_{PSmin}$. Accordingly, the corresponding current sensing point occurs $T_{SD}$ after the end of $T_{PSmin}$. In Figure 14, $T_{a2}$ is less than $T_{PSmin}$. So, phase shift is applied to ensure the adjusted $T_{a2}' = T_{PSmin}$. The corresponding CS2 occurs $T_{SD}$ after the end of $T_{a2}'$. In Figure 15, $T_{b2}$ is less than $T_{PSmin}$. So, phase shift is applied to ensure the adjusted $T_{b2}' = T_{PSmin}$. The corresponding CS1 occurs $T_{SD}$ after the end of $T_{b2}'$.

If the desired CS1 or CS2 point is estimated to occur after the end of the PWM cycle, then the actual CS1 or CS2 point is adjusted to occur just before the end of this PWM cycle to ensure the latest sampled current values are available at the beginning of the following PWM cycle when the FOC calculation is executed.

By using phase shift scheme, the actual output during each PWM cycle will be exactly the same as target output. Control performance at lower speed can be improved compared to using minimum pulse width PWM scheme. To achieve optimal control performance in this mode, ‘TminPhaseShift’ and ‘SHDelay’ parameters need to be tuned appropriately.

a

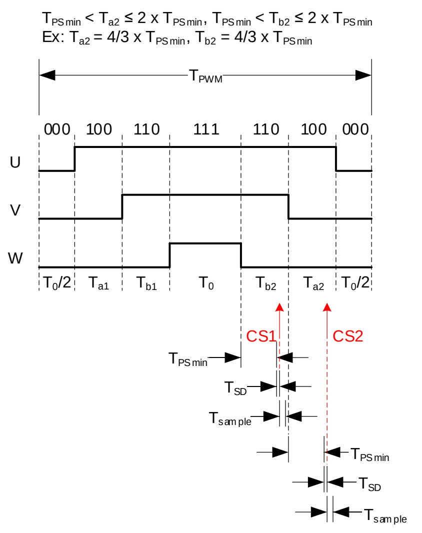

$T_{PSmin} < T_{a2} \leq 2 \times T_{PSmin}$, $T_{PSmin} < T_{b2} \leq 2 \times T_{PSmin}$
Ex: $T_{a2} = 4/3 \times T_{PSmin}$, $T_{b2} = 4/3 \times T_{PSmin}$
$T_{PWM}$
000
100
110
111
110
100
000
U
V
W
$T_0/2$
$T_{a1}$
$T_{b1}$
$T_0$
$T_{b2}$
$T_{a2}$
$T_0/2$
CS1
CS2
$T_{PSmin}$
$T_{SD}$
$T_{sample}$
$T_{PSmin}$
$T_{SD}$
$T_{sample}$

**Figure 16 Single Shunt Configuration with Phase Shift PWM Scheme Current Sensing Timing Diagram (Case 3: $T_{PSmin} \leq T_{a2} \leq 2 \times T_{PSmin}$, $T_{PSmin} \leq T_{b2} \leq 2 \times T_{PSmin}$)**

#### 2.1.7.2.3 Low Noise Phase Shift PWM

One of the drawbacks of the above-mentioned phase shift scheme is that the shifting patterns are different in different sectors, and the change in shifting patterns during the sector-crossing time would still cause some acoustic noise, especially when the motor is running at lower speed.

MCE provides an alternative option of low noise phase shift PWM scheme in order to further reduce the acoustic noise when the motor is running at lower speed. Compared to normal phase shift PWM scheme, the low noise phase shift PWM scheme adopts a fixed shifting pattern in all 6 PWM sectors, so that the acoustic noise caused by shifting pattern change is eliminated.

As shown in Figure 17, a fixed shifting pattern in the order of W->V->U is chosen with which the available vectors for single-shunt current sensing are vector [110] and [100]. With these 2 active vectors, motor current on W phase and U phase can be sensed consecutively. The duration of these 2 vectors ($T_{PSmin}$) can be configured by using the parameter ‘TMinPhaseShift’ following this equation $T_{PSmin} = TminPhaseShift \times 10.417ns$.

Figure 17 shows 5 typical output voltage vector examples (A, B, C, D, E) that fall within the sector-crossing area (grey area) using low noise phase shift PWM scheme.

In example A, vector [110] and [100] are already available but vector [100] is too short for sensing phase U current properly. With low noise phase shift PWM scheme, V phase PWM and W phase PWM are shifted asymmetrically to extend the period of vector [100] to form an appropriate window for sensing phase U current.

In example B, vector [110] and [100] are already available but vector [110] is too short for sensing phase W current properly. With low noise phase shift PWM scheme, V phase PWM and W phase PWM are shifted asymmetrically to extend the period of vector [110] to form an appropriate window for sensing phase W current.

In example C, vector [100] is already available, but vector [110] is not available. With low noise phase shift PWM scheme, an additional vector [110] is added to form an appropriate window for sensing phase W current by shifting V phase PWM and W phase PWM asymmetrically. The impact of introducing the additional vector [110] is mitigated thanks to the extension of vector [101] and shrinking of vector [100].

In example D, vector [100] is already available, but vector [110] is not available. With low noise phase shift PWM scheme, an additional vector [110] is added to form an appropriate window for sensing phase W current by shifting V phase PWM and W phase PWM asymmetrically. The impact of adding vector [110] is mitigated thanks to the addition of vector [001].

In example E, vector [100] is already available, but vector [110] is not available. With low noise phase shift PWM scheme, an additional vector [110] is added to form an appropriate window for sensing phase W current by shifting V phase PWM and W phase PWM asymmetrically. The impact of adding vector [110] is mitigated thanks to the addition of vector [001].

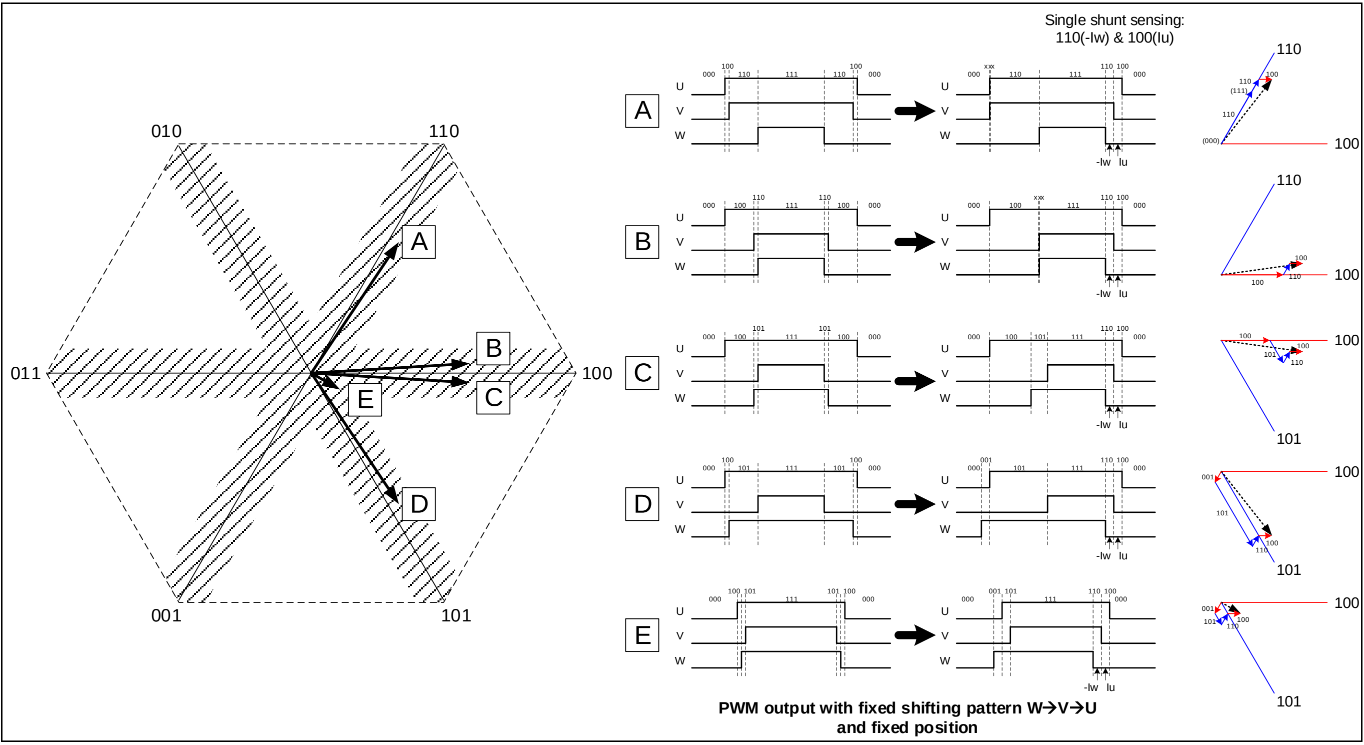


**Figure 17 Low Noise Phase Shift PWM Scheme**

The current sensing timing for single shunt configuration with low noise phase shift PWM is shown in the following Figure 18. With low noise phase shift PWM scheme, no matter if the active vector time $T_{a2}$ or $T_{b2}$ is sufficient or not compared to the desired minimum active vector time ($T_{PSmin}$), the phase PWM waveforms are always shifted to include the active vector [110] and [100] time with the duration of $T_{PSmin}$ ($T_{a2}' = T_{PSmin}$, $T_{b2}' = T_{PSmin}$) to satisfy the current sensing requirement. Consequently, the first current sensing point (CS1) occurs $T_{SD}$ after the end of the active vector [110] time $T_{b2}'$. The second current sensing point (CS2) occurs $T_{SD}$ after the end of the active vector [100] time $T_{a2}'$.

If the desired CS1 or CS2 point is estimated to occur after the end of the PWM cycle, then the actual CS1 or CS2 point is adjusted to occur just before the end of this PWM cycle to ensure the latest sampled current values are available at the beginning of the following PWM cycle when the FOC calculation is executed.

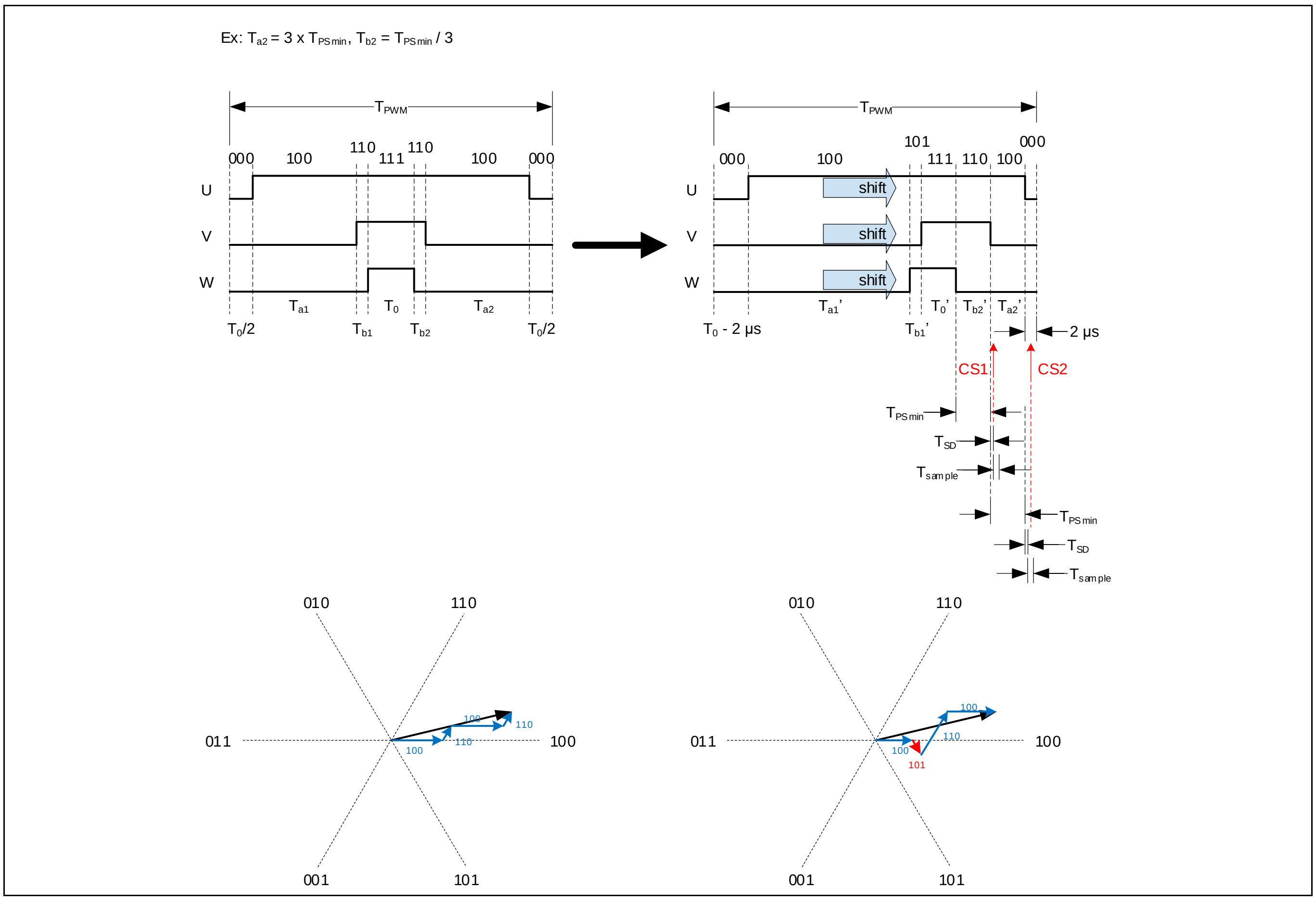


Figure 18 **Single Shunt Configuration with Low Noise Phase Shift PWM Scheme Current Sensing Timing Diagram**

Since the shifting pattern is fixed, low noise phase shift PWM is only applicable to 3-phase PWM modulation type, and the maximum PWM modulation index is limited. When low noise phase shift PWM scheme is enabled, the MCE automatically shifts to normal phase shift PWM scheme if the modulation index increases to more than 50%. If the modulation index is decreased below 35%, the MCE automatically shifts back to low noise phase shift PWM scheme.

With low noise phase shift PWM scheme, the actual output voltage during each PWM cycle is still exactly the same as the target output voltage. As a result, acoustic noise level at low speed and start-up performance is further improved compared to using normal phase shift PWM scheme. To achieve optimal control performance in this mode, 'TminPhaseShift' and 'SHDelay' parameters need to be tuned appropriately.

###### 2.1.7.2.4 Peak Current Tracking with No Phase Shift Window

Certain AC fan control applications are extremely sensitive to acoustic noise especially in the low speed operating range. In this case, a modulation control mode without a minimum pulse sampling window minimizes sinusoidal voltage modulation distortion and the associated acoustic noise. In the single shunt configuration, the lack of a minimum sampling window restricts inverter current sampling to PWM cycles with active vectors greater than the required minimum pulse width. This discontinuous current sampling does not support AC winding current reconstruction and limits the control to open loop modulation / voltage control. This does not significantly impact drive performance at low speeds but there is a need to limit motor currents in overload conditions. It is still possible to provide overload protection based on the available current samples but a sample rate lower than the PWM frequency. MCE provides an alternative peak current tracking method to realize peak current limiting function when the phase shift window is fully closed.

When TminPhaseShift = 0 with single shunt configuration, MCE automatically switches to peak current tracking mode in which it takes 2 consecutive current samplings during each PWM cycle, and the bigger value of the 2 current sample values is assigned to variable 'Ipeak'. Right after each sector change, the 'Ipeak' variable is reset to zero to prepare for the peak current tracking in the new sector. The 'Ipeak' value is then directly assigned to variable 'Iq' per PWM cycle so that the q axis regulator can limit the current. Meanwhile, the 'Id' variable is always set to zero in peak current tracking mode.

The motor phase current feedback signal path with TminPhaseShift = 0 is shown in the following Figure 19. The scaling factor for 'Ipeak' is designed in such a way that 'Ipeak' value is represented in the same way as how the 'Iq' value is represented. Using this peak current tracking method, one can still use the Iq current control loop to monitor and limit the peak current when TminPhaseShift = 0 with single shunt configuration.

When TminPhaseShift ≠ 0, 'Ipeak' variable is reset to zero, and MCE goes back to normal phase current reconstruction mode with single shunt configuration.

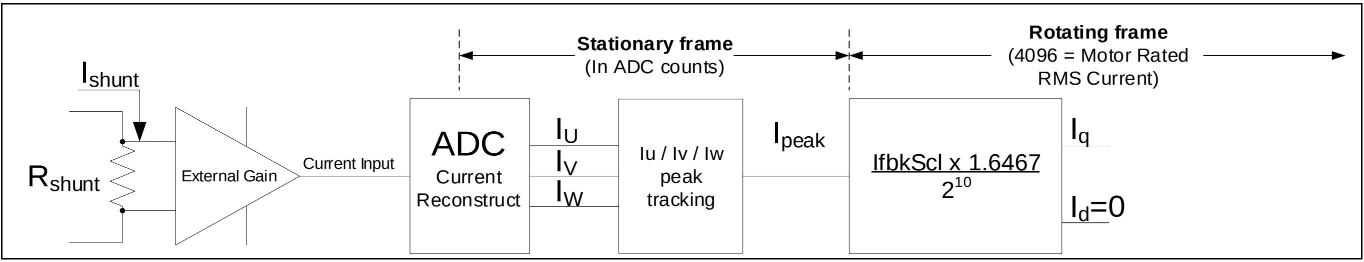


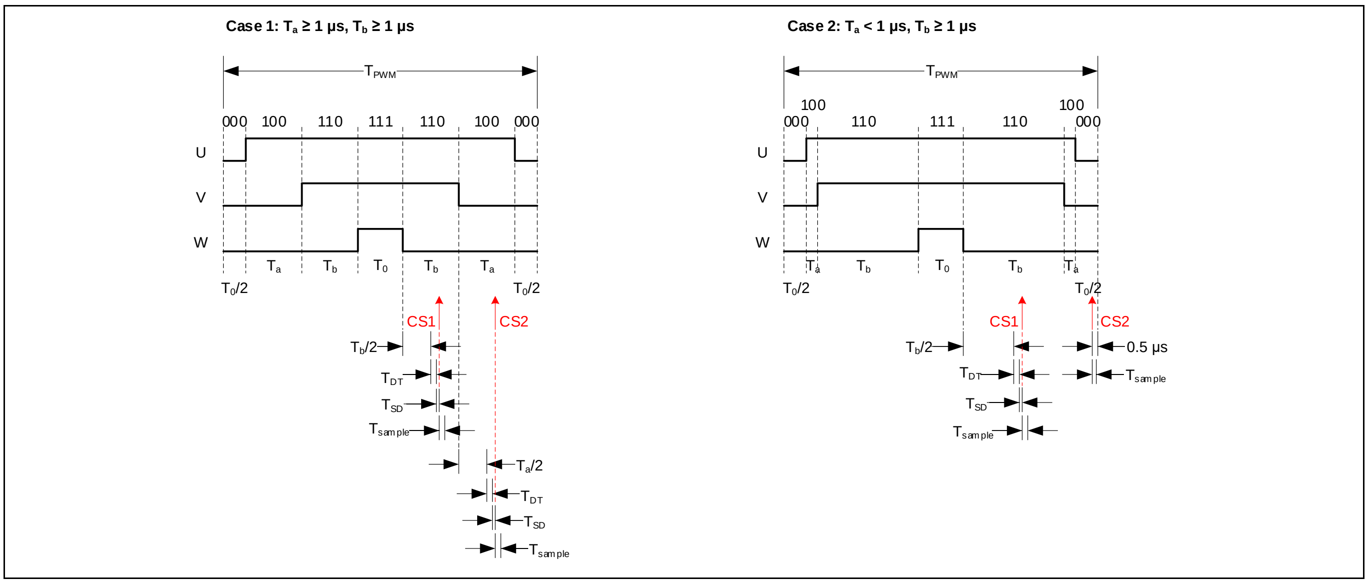
**Figure 19 Motor current feedback signal path (TminPhaseShift = 0, single shunt)**

In peak current tracking mode, the motor current sensing timing is adjusted as needed.

For normal phase shift PWM scheme in peak current tracking mode as shown in the following Figure 20, if both active vector time ($T_a$ and $T_b$) is longer than 1 µs (Case 1), then the first current sensing point (CS1) occurs $T_{DT} + T_{SD}$ after half of the active vector time $T_b$. The second current sensing point (CS2) occurs $T_{DT} + T_{SD}$ after half of the active vector time $T_a$.

If the active vector time $T_a$ is shorter than 1 µs (Case 2, Case 4), then CS2 point is relocated to 0.5 µs before the end of the current PWM cycle to avoid getting invalid current sensing value.

If the active vector time $T_b$ is shorter than 1 µs (Case 3, Case 4), then CS1 point is relocated to 0.5 µs after the start of the current PWM cycle to avoid getting invalid current sensing value.



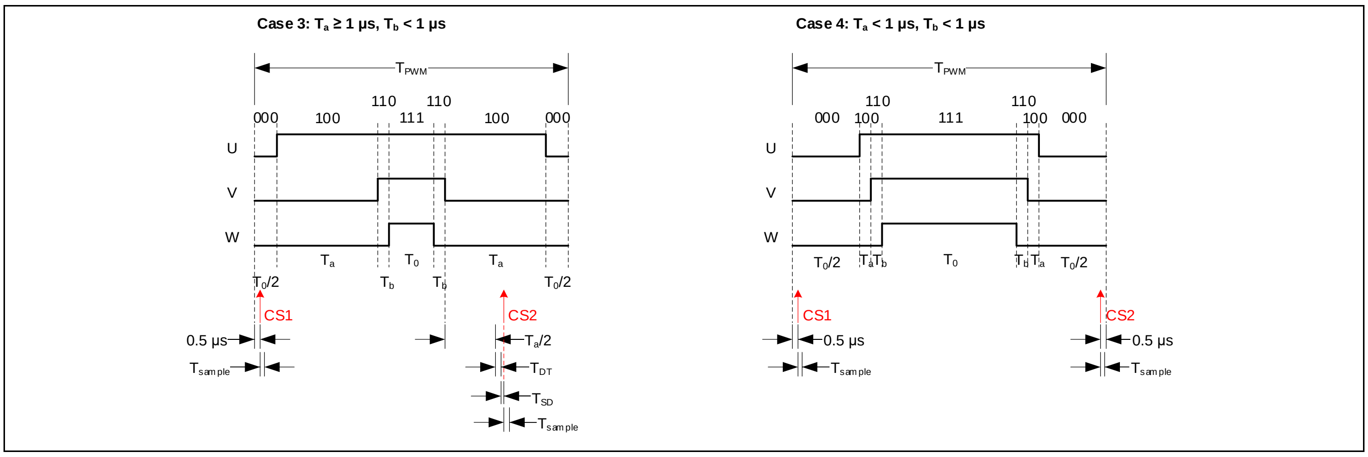


Figure 20 **Single Shunt Configuration with Phase Shift PWM Scheme in Peak Current Tracking Mode Current Sensing Timing Diagram**

For low noise phase shift PWM scheme in peak current tracking mode as shown in the following Figure 21, assuming the total active vector time is 2 x $T_a$ and 2 x $T_b$ respectively, if both $T_a$ and $T_b$ are longer than 1 µs (Case 1), then the first current sensing point (CS1) occurs $T_a + T_{DT} + T_{SD} + 1\mu s$ after the start of the active vector time 2 x $T_a$. The second current sensing point (CS2) occurs $T_b + T_{DT} + T_{SD} + 1\mu s$ after the start of the active vector time 2 x $T_b$.

If $T_a$ is shorter than 1 µs (Case 2, Case 4), then CS1 point is relocated to 1 µs before the start of the active vector time 2 x $T_a$ to avoid getting invalid current sensing value. If the desired CS1 point is estimated to occur before the start of the PWM cycle, then the actual CS1 point is adjusted to occur just after the start of this PWM cycle to avoid getting invalid current sensing value.

If $T_b$ is shorter than 1 µs (Case 3, Case 4), then CS2 point is relocated to 4 µs after the start of the zero vector [111] time $T_0$ to avoid getting invalid current sensing value.

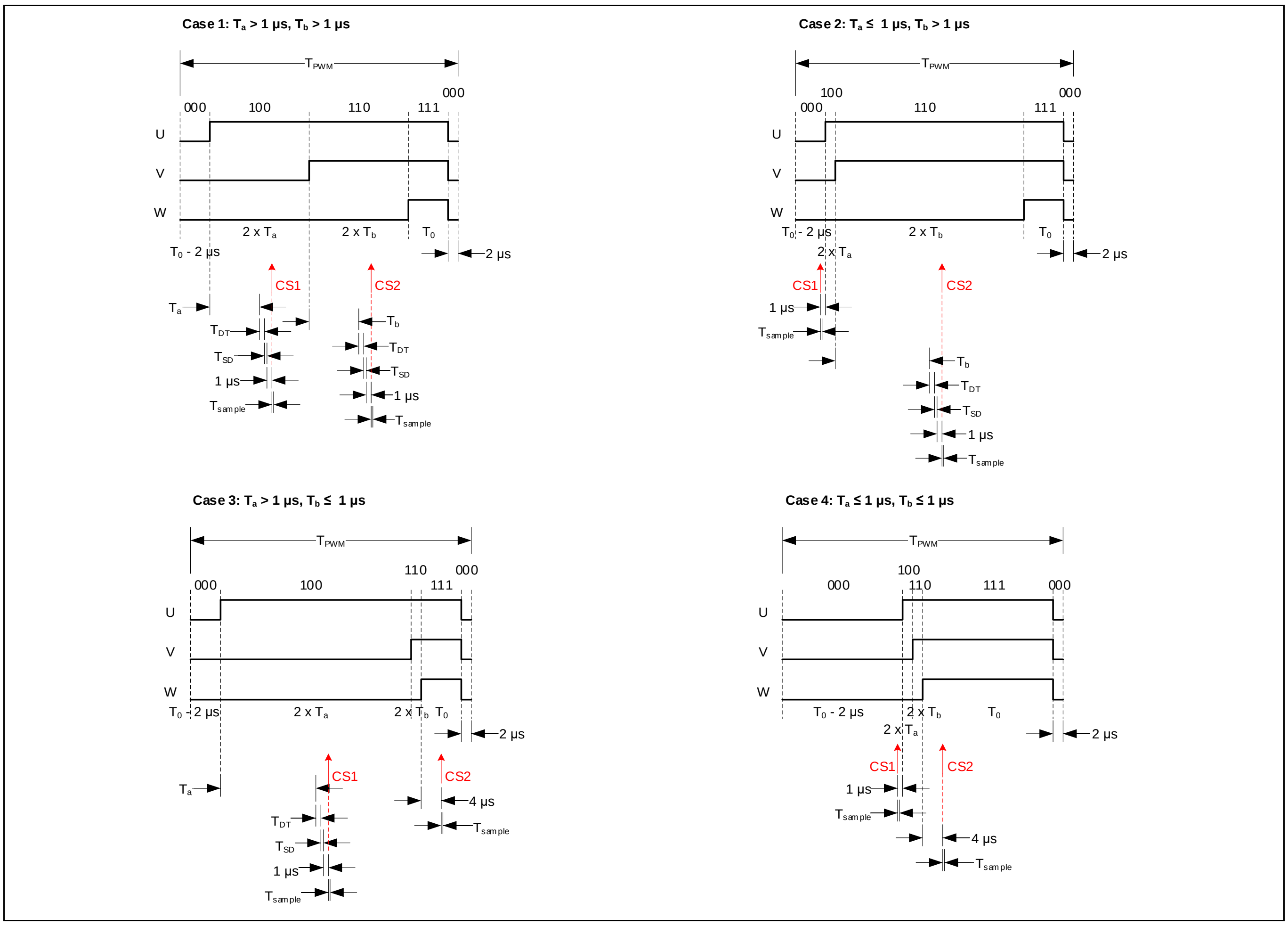

Case 1: Ta > 1 µs, Tb > 1 µs
Case 2: Ta ≤ 1 µs, Tb > 1 µs
Case 3: Ta > 1 µs, Tb ≤ 1 µs
Case 4: Ta ≤ 1 µs, Tb ≤ 1 µs
TPWM
000
100
110
111
U
V
W
2 x Ta
2 x Tb
T0
T0 - 2 µs
2 µs
CS1
CS2
Ta
Tb
TDT
TSD
1 µs
Tsample
4 µs

Figure 21 **Single Shunt Configuration with Low Noise Phase Shift PWM Scheme in Peak Current Tracking Mode Current Sensing Timing Diagram**

## 2.1.8 Motor Current Limit Profile

Some applications (such as fan) don't require high current at low speed. In other words, full torque is only required above a certain speed. The MCE provides a configurable dynamic motor current limit feature which reduces the current limit in the low speed region for a smooth startup. This feature provides smooth and quiet start up, and it also can reduce the rotor lock current.

Figure 22 depicts that the motor current limit changes dynamically as a function of motor speed. The MCE enables the motor load to work in both motoring mode (1st and 3rd quadrants in Figure 22) and regenerating mode (2nd and 4th quadrants Figure 22).

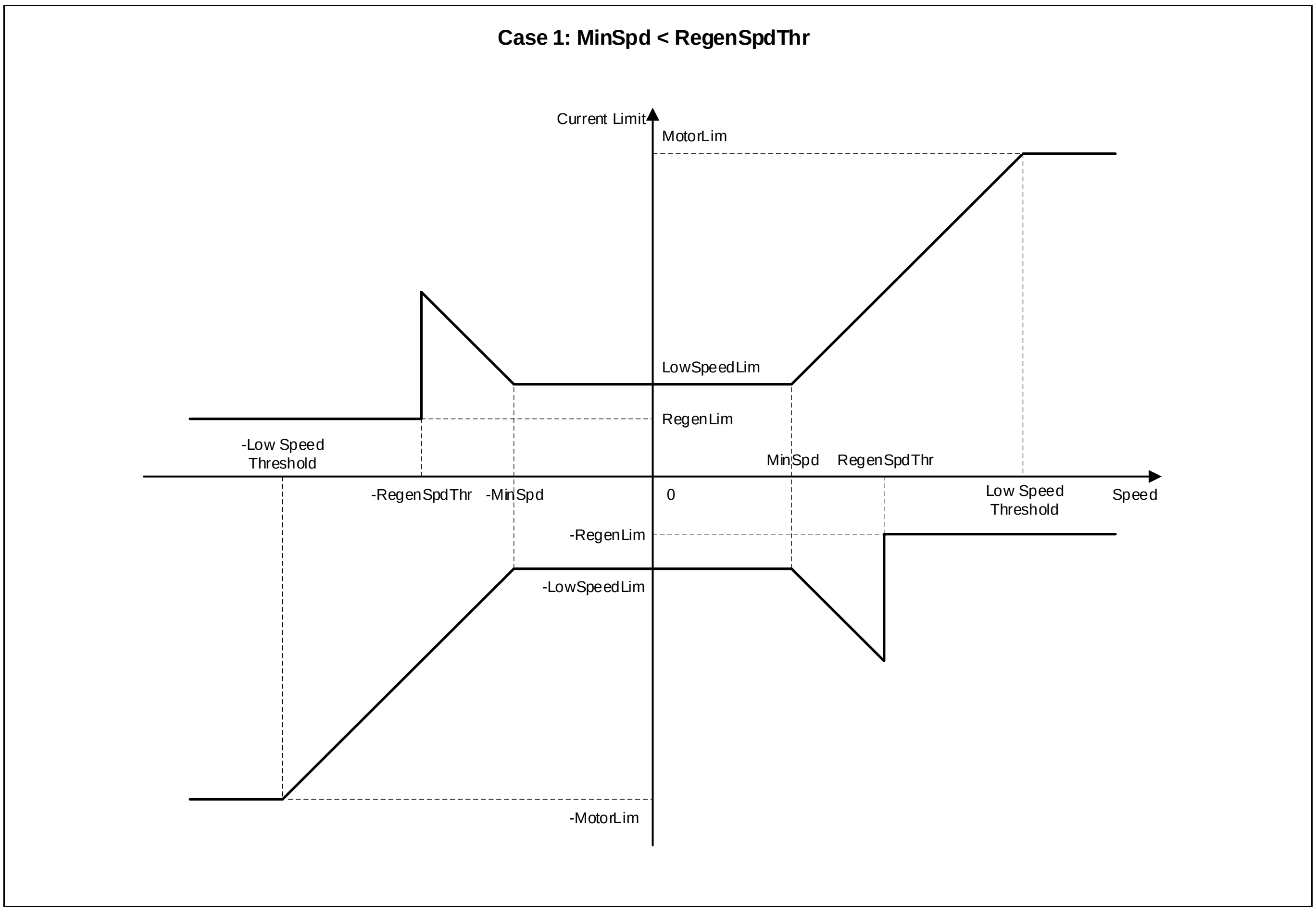
In motoring mode, when the absolute value of the motor speed is below the minimum speed specified by the parameter 'MinSpd' ($|MotorSpeed| \leq MinSpd$), the maximum motor current is limited to a threshold configured by parameter 'LowSpeedLim'. When the absolute value of the motor speed is between the minimum speed and the low speed threshold, the motor current limit increases linearly as the speed increases following the relationship as below.

$$Motor\ Current\ Limit = LowSpeedLim + (|MotorSpeed| - MinSpd) \times LowSpeedGain$$

When the motor speed goes beyond the low speed threshold, the maximum motor current is limited to the upper boundary specified by the parameter 'MotorLim'.

In regenerating mode, when the absolute value of motor speed is below a threshold specified by the parameter 'RegenSpdThr', the motor current limit follows the above-mentioned linear relationship. When the absolute value of motor speed goes beyond the threshold specified by the parameter 'RegenSpdThr', the maximum motor current is limited to a threshold specified by the parameter 'RegenLim'.

Having the freedom to adjust the motor current limit in motoring mode and regenerating mode independently allows users to tailor the acceleration torque as well as the regenerative braking torque separately to achieve optimal drive performance. If further customization of motor current limit is required, users can take advantage of script code to program the motor current limit ('MotorLim' parameter) to any arbitrary profile.



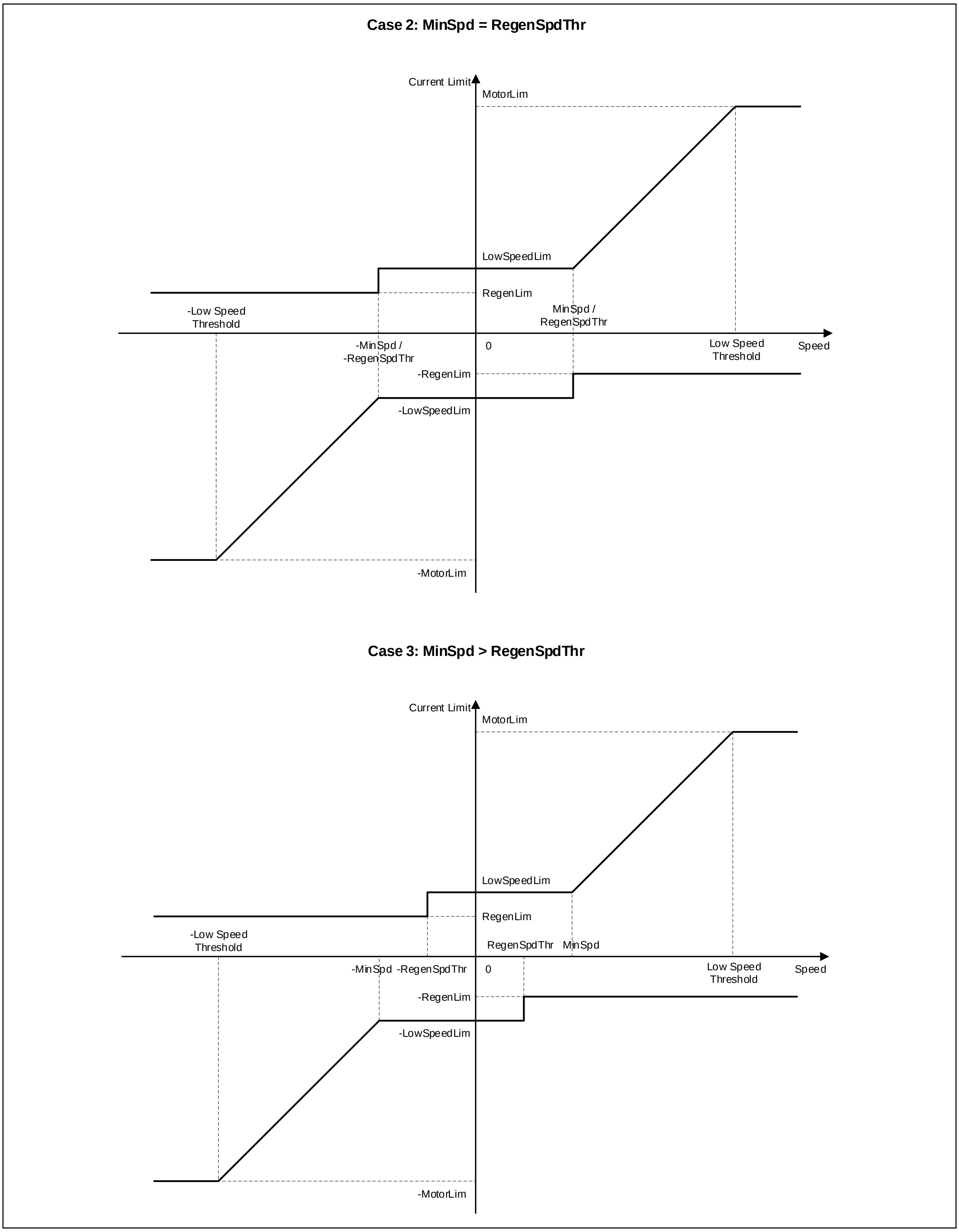


**Figure 22 Motor Current Limit Profile**

### 2.1.9 Initial Angle Sensing

Some fan applications requires starting up motors in the right direction reliably without reverse motion. Using the traditional parking + open-loop method would cause undesired reverse motion in some cases. Using direct start method sometimes might fail due to insufficient Back-EMF at low motor speed range.

MCE offers a patented initial angle sensing function that estimates the rotor angle by injecting six current pulses at different angles for a duration of a few milliseconds before starting. The initial angle is then calculated based on the current amplitude of those sensing pulses. After ANGLE_SENSING state is completed, the motor state machine would shift to MOTOR_RUN state to run the closed loop FOC control directly.

Using the initial angle sensing function can always starts the motor in the right direction and avoids potential reverse motion during parking when used in sensorless FOC control. The initial angle estimation relies on rotor magnetic saliency and performs well when the motor $L_d$ to $L_q$ ratio is less than 95% and the average inductance is greater than 0.1 mH.

The relevant control parameters (IS_Pulses, IS_Duty, IS_IqInit) are automatically calculated by Solution Designer based on the $L_d$ and $L_q$ motor parameters entered.

This method only takes care of the initial angle measurement so tuning the flux estimator may be required when driving high inertia loads. If the motor speed is not zero at the start-up, then the detected angle might not be accurate. It is then recommended to use catch-spin function in that scenario.

### 2.1.10 Over-Modulation

As shown in the following Figure 23, the linear modulation range is defined by the disk that fits within the hexagonal active voltage vector (a, b) timing limit boundary. The modulation index can be up to $\frac{\sqrt{3}}{2} = 0.866$ if the modulation stays within linear range. If maximizing output power is the priority and non-linear modulation is acceptable, then the modulation index can go up to 1 so that the active voltage vector goes outside the disk into the grey area to make full use of the DC bus voltage.

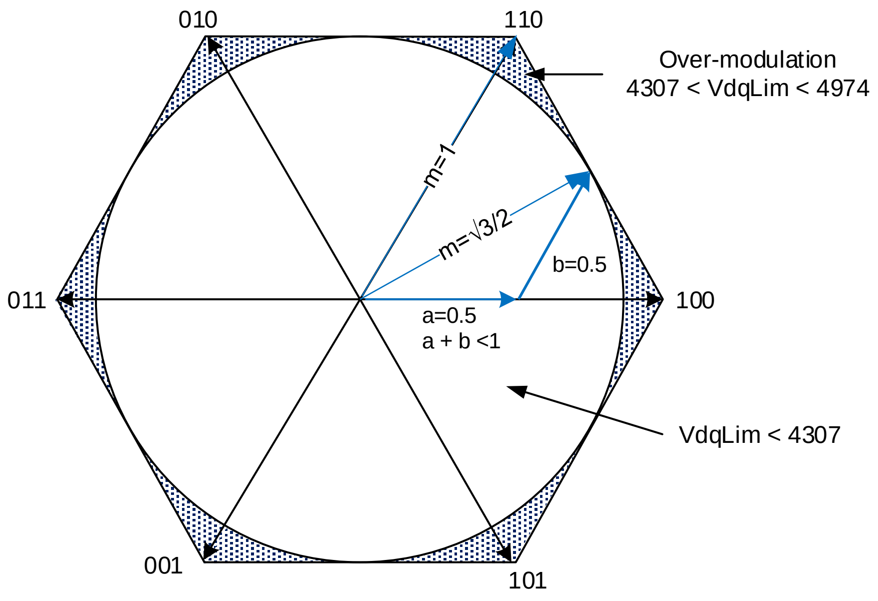


**Figure 23 SVPWM Vector Timing Limit Diagram**

The MCE offers the parameter 'VdqLim' to configure the desired modulation index limit. 100% modulation corresponds to 4974 counts for parameter 'VdqLim'. If users need to limit the modulation to only linear range, the parameter ‘VdqLim’ shall be set up to 4974 x 0.866 = 4307. If users need to take advantage of over-modulation, then the parameter ‘VdqLim’ shall be set up to 4974.

Although utilizing over-modulation helps maximize DC bus voltage utilization, it would introduce acoustic noise associated with the additional harmonics, and compromise the flux PLL operation and result in errors in RMS current and voltage based power or torque calculations.

### 2.1.11 2-Phase Modulation

MCE supports 2-phase type 3 (low-side clamping) space vector PWM modulation with a configurable switch-over threshold. As shown in the following Figure 24, 2-phase type 3 modulation clamps one motor winding to the negative inverter rail. Thus, it eliminates switching of one of the 3 inverter legs in each sector to reduce switching loss while keeping the output line voltage the same as compared to the case of 3-phase modulation. This is done by not using zero vector [111] and allocating all the zero vector time to the zero vector [000].

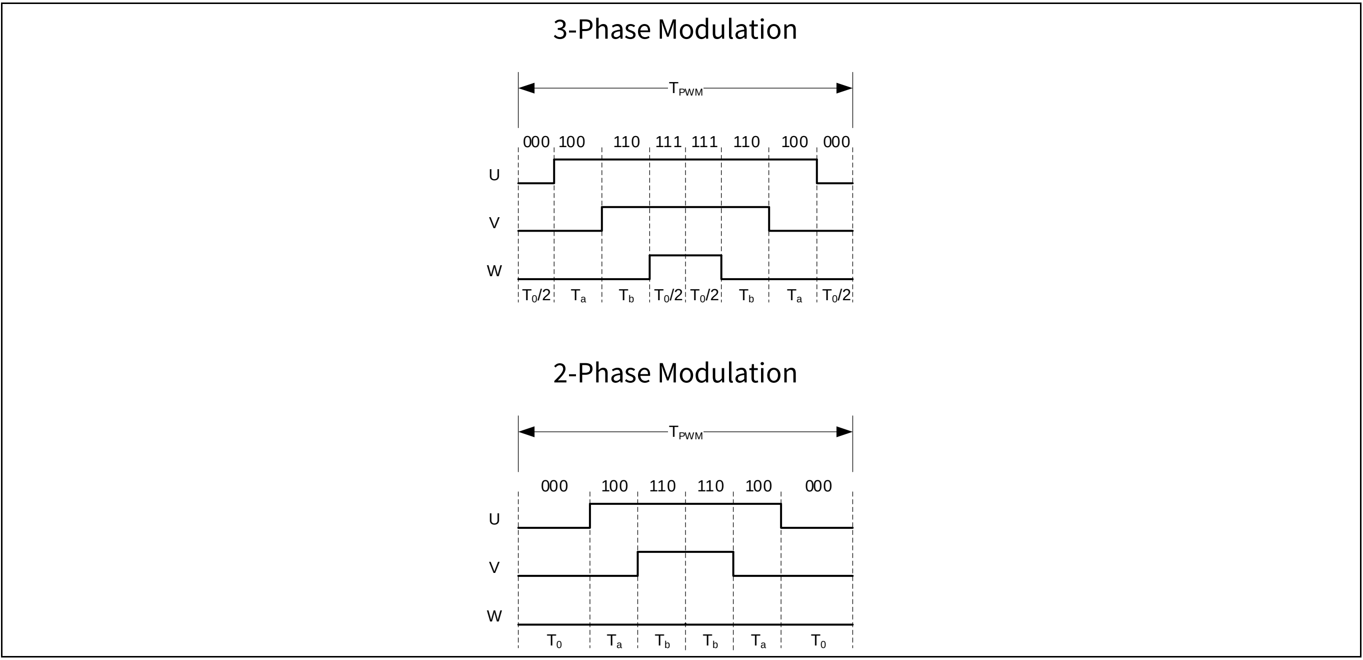


**Figure 24 3-Phase / 2-Phase Type 3 SV PWM Modulation Diagram**

2-phase type 3 PWM modulation cannot be used at low speeds when the high side gate driver uses a bootstrap diode to charge up the voltage rail. The bootstrap capacitor must be sized sufficiently to hold enough charge to drive the high side gate for the full duration of a SV PWM Modulation sector.

Bit field [4:3] of the parameter 'HwConfig' is used to enable 2-phase type 3 PWM modulation. As shown in the following Figure 25, if 2-phase type 3 SVM is enabled, at start-up 3-phase PWM modulation is used. When the motor absolute speed (variable 'Abs_MotorSpeed') goes above a configurable threshold (parameter 'Pwm2PhThr'), MCE would switch to using 2-phase type 3 PWM modulation. When the absolute motor speed goes below the configurable threshold (parameter 'Pwm2PhThr') with a hysteresis of 256 counts (1.6% of motor max RPM), MCE would switch back to 3-phase PWM modulation.

If the value of the parameter 'Pwm2PhThr' is 256 or lower (≤ 1.6% of motor max RPM), and 2-phase PWM modulation is enabled, after MCE has switched to 2-phase PWM modulation, it would not switch back to 3-phase PWM modulation automatically until motor is stopped and then is restarted.

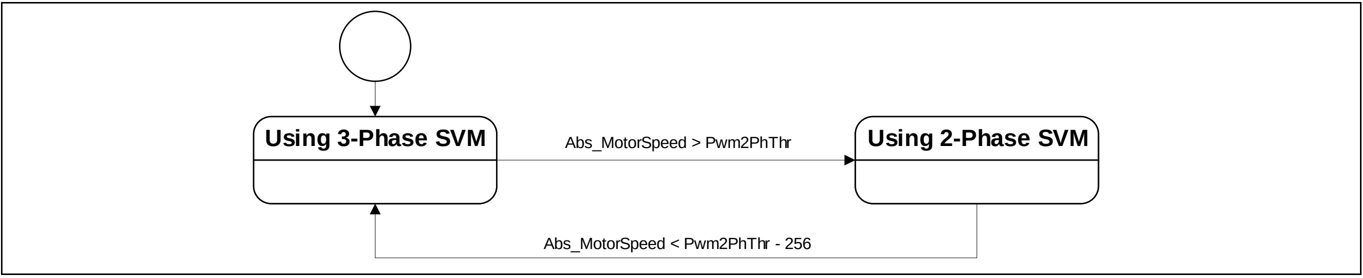


**Figure 25 3-Phase SVM and 2-Phase SVM State Transition Diagram**

### 2.1.12 Catch Spin

Before turning on the inverter, due to some external force, for example wind air flow in fan applications, the motor may be already spinning. The MCE offers 'Catch Spin' feature which is designed to synchronize the flux estimator and flux PLL with the actual motor speed before providing the torque to drive the motor. Catch spin cannot be done if the motor back EMF voltage is higher than the DC bus voltage, which usually occurs when the motor is running above rated speed. Hence, the catch spin is generally effective up to the rated speed of the motor. The catch spin starting process is part of the motor state machine and is executed at start-up if catch spin function is enabled.

In catch spin, the controller tracks the back EMF in order to determine if the motor is turning, and if so, in which direction. Catch spin sequence begins after the bootstrap capacitor charging stage is completed. During catch spin, both IqRef and IdRef are set to 0 (Speed regulator is disabled), meanwhile flux PLL attempts to lock to the actual motor speed (variable 'MotorSpeed') and rotor angle (variable 'RotorAngle'). Catch spin time, defined by TCatchSpin parameter. Once catch spin time is elapsed, calculated motor speed check with "DirectStartThr" parameter value. If motor speed is more than or equal to "DirectStartThr" parameter value, normal speed control starts, current motor speed will become the initial speed reference and also set as the speed ramp starting point. Depending on the set target speed, motor will decelerate (via regenerative braking) or accelerate to reach the desired speed. If motor speed is less than "DirectStartThr" parameter value, motor state changes to "ANGLESENSING" state.

Depending upon the direction of rotation, there are 3 types of catch spin scenarios

- Zero Speed Catch Spin
- Forward Catch Spin
- Reverse Catch Spin

#### 2.1.12.1 Zero Speed Catch Spin

If the motor is stationary, then the catch spin sequence is termed as 'Zero Speed Catch Spin'. Figure 26(A) shows an example for 'Zero Speed Catch Spin'. In this example, at the start command, the motor is stationary. After the start command, 'Zero Speed Catch Spin' sequence begins. During the catch spin sequence, no motoring current is injected. After the catch spin time has elapsed, the motor speed at that instance (which is 0 RPM) becomes initial speed reference and starting point for speed ramp reference. The motor continues to accelerate, following the speed ramp reference to reach the set target speed.

If catch spin is disabled, normal speed control starts immediately after the start command, without waiting for PLL to be locked. As shown in Figure 26 (B), after the start command, motoring current is injected directly as there is no catch spin sequence. The motor starts accelerating, following the speed ramp reference to reach the set target speed.

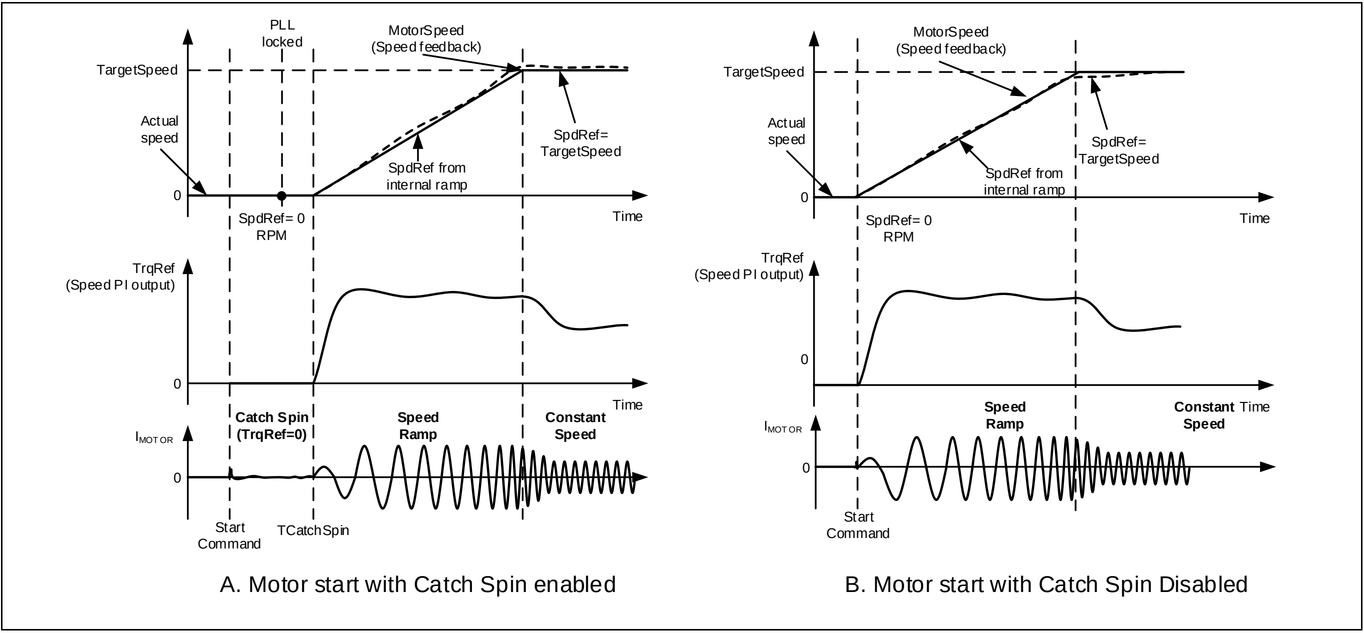


Figure 26 Zero Speed Catch Spin - Motor start with/without catch spin

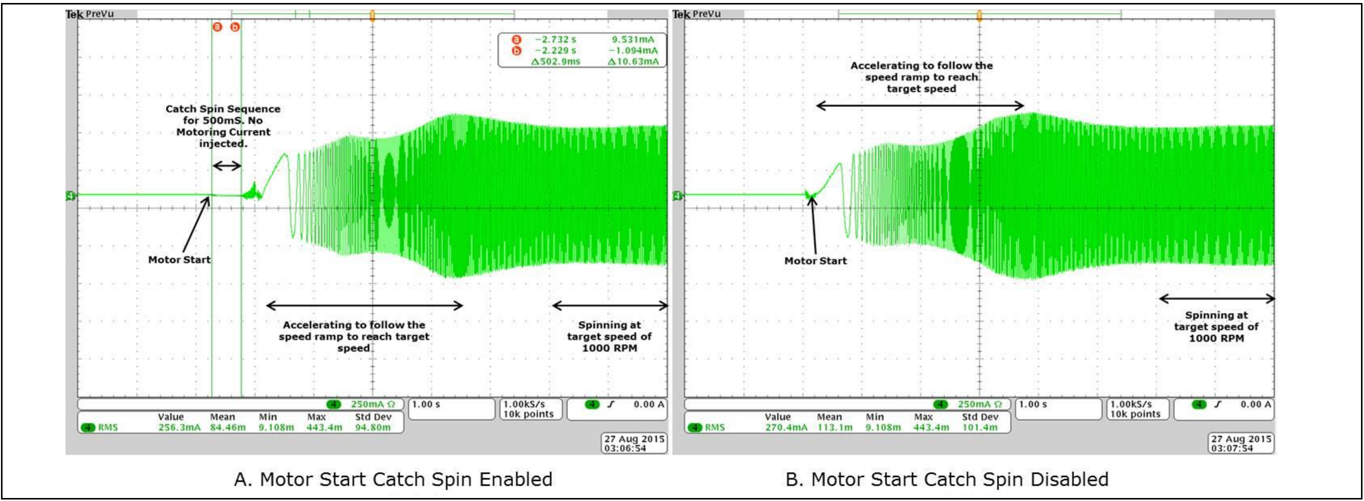


Figure 27 Motor Phase Current - Zero Speed Catch Spin - Motor start with/without catch spin

#### 2.1.12.2 Forward Catch Spin

If the motor is spinning in the same direction as desired, then the catch spin sequence is termed as ‘Forward Catch Spin’. Figure 28 (A) shows an example for ‘Forward Catch Spin’. In this example, at the start command the motor is already spinning (in the desired direction). During the catch spin sequence, no motoring current is injected. After the catch spin time has elapsed, assuming the flux PLL locks to the actual motor speed, the motor speed at that instance becomes initial speed reference and starting point for speed ramp reference. The motor continues to accelerate or decelerate, following the speed ramp reference to reach the set target speed.

If catch spin is disabled, normal speed control starts immediately after the start command, without waiting for PLL to be locked. Usually the control would still be able to start a spinning motor, but motor speed may not increase/decrease seamlessly. As shown in Figure 28 (B), after the start command, the actual motor speed is higher than speed reference (variable ‘SpeedRef). Hence, the motor is decelerated (using regenerative braking) to force the motor to follow the speed reference (variable ‘SpeedRef). As the speed of the motor is higher than Regen Speed Threshold (variable ‘RegenSpdThr’), the negative torque injected in the motor to achieve deceleration is limited by the value in RegenLim parameter. Once the motor speed matches the speed reference, the motor starts accelerating, following the speed ramp reference to reach the set target speed.

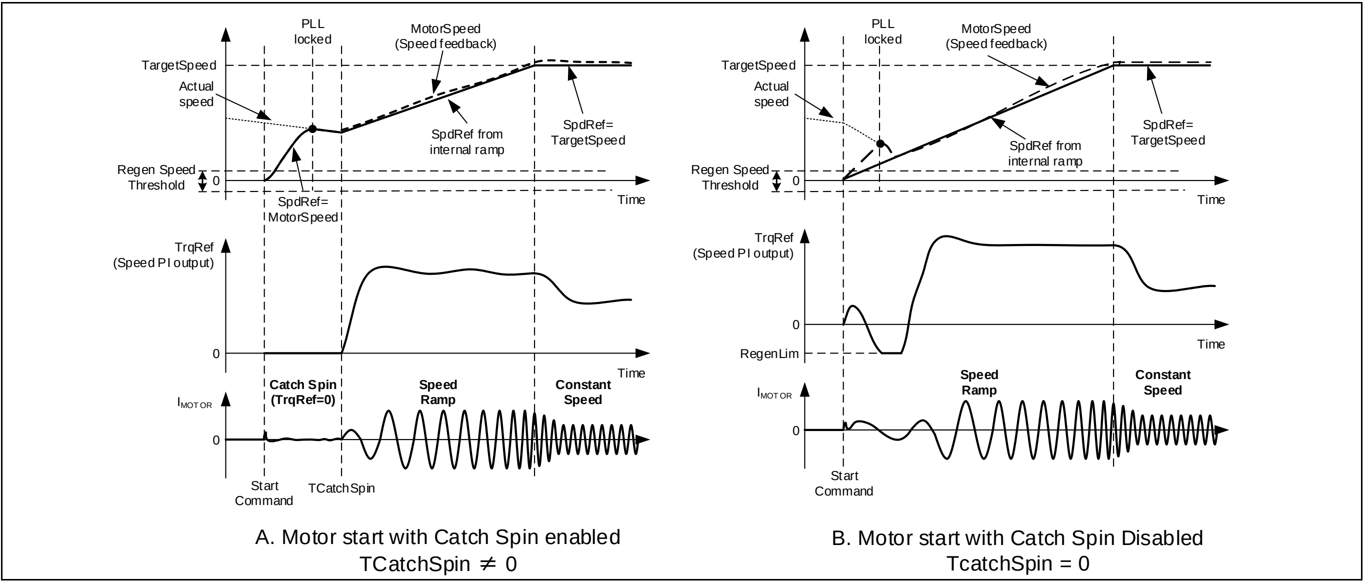


Figure 28 **Forward Catch Spin - Motor start with/without catch spin**

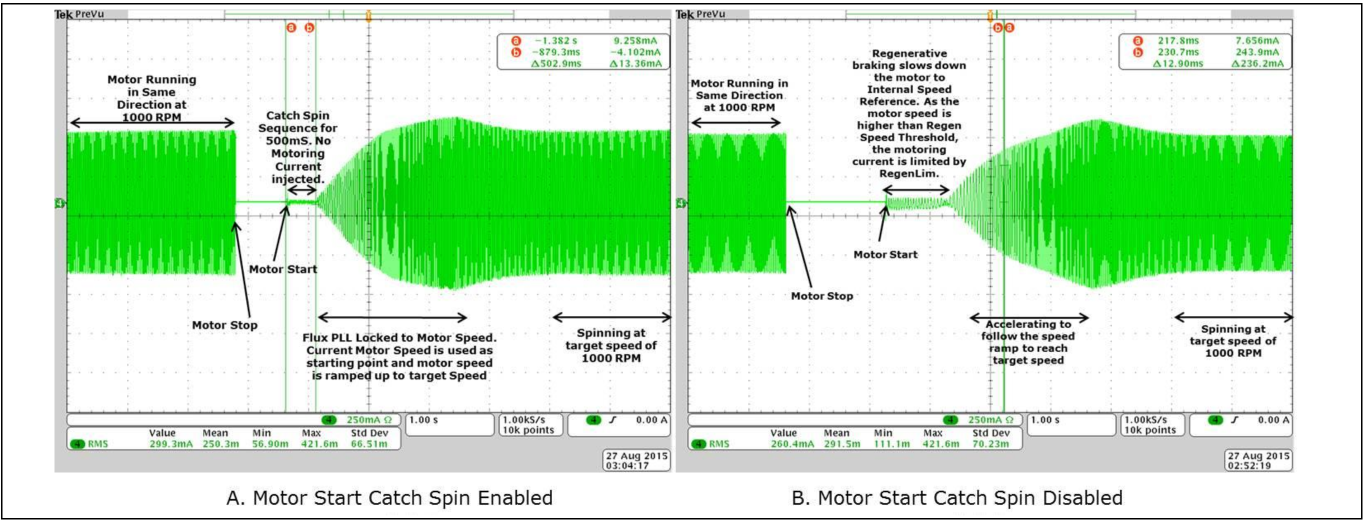


Figure 29 **Motor Phase Current Waveform - Forward Catch Spin - Motor start with/without catch**

### 2.1.12.3 Reverse Catch Spin

If the motor is spinning in the opposite direction as desired, then the catch spin sequence is termed as 'Reverse Catch Spin'. Figure 30 (A) shows an example of 'Reverse Catch Spin'. In this example, at the start command, the motor is already spinning (in the opposite direction). During the catch spin sequence, no motoring current is injected. After the TCatchSpin time has elapsed, the motor is still spinning in opposite direction at a speed higher than Regen Speed Threshold (RegenSpdThr), thus an injected torque, limited by the value defined in RegenLim parameter, forces the motor to decelerate via regenerative braking. Once the speed of the reverse spinning motor falls below Regen Speed Threshold (RegenSpdThr), the injected torque is limited by MotorLim (RegenLim<=MotorLim). The injected torque forces the motor to come to a stop and start accelerating in the desired spin direction, following the speed ramp reference to reach the set target speed.

If catch spin is disabled, normal speed control starts immediately after the start command, without waiting for PLL to be locked. Usually the control would still be able to start a spinning motor, but motor speed may not increase/decrease seamlessly. As shown in Figure 30 (B), after the start command, the motor is still spinning at a speed higher than Regen Speed Threshold (RegenSpdThr), hence the injected torque limited by the value defined in RegenLim parameter, forces the reverse spinning motor to decelerate via regenerative braking. Once the speed of the reverse spinning motor falls below Regen Speed Threshold (RegenSpdThr), the injected torque is limited by MotorLim (RegenLim<=MotorLim). The injected torque forces the motor to come to a stop and start accelerating in the desired spin direction, following the speed ramp reference to reach the set target speed.

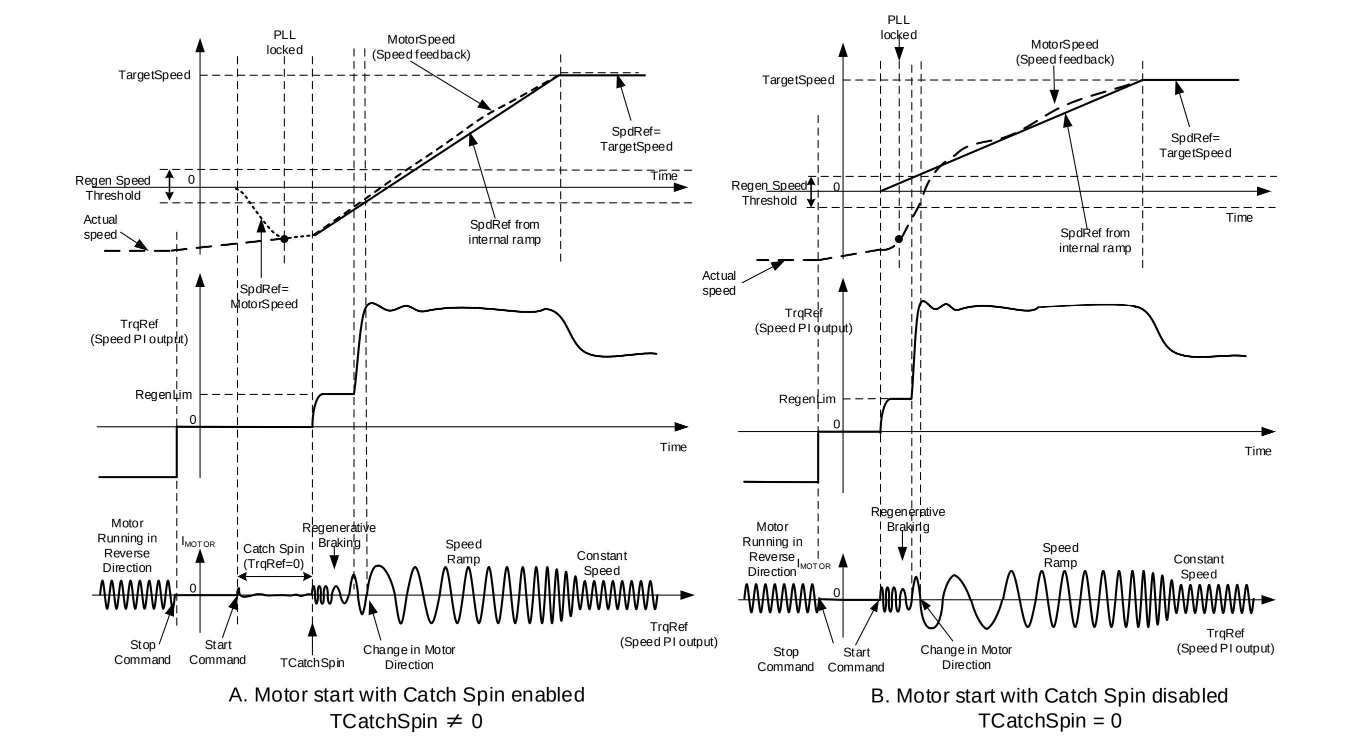


**Figure 30 Reverse Catch Spin - Motor start with/without catch spin**

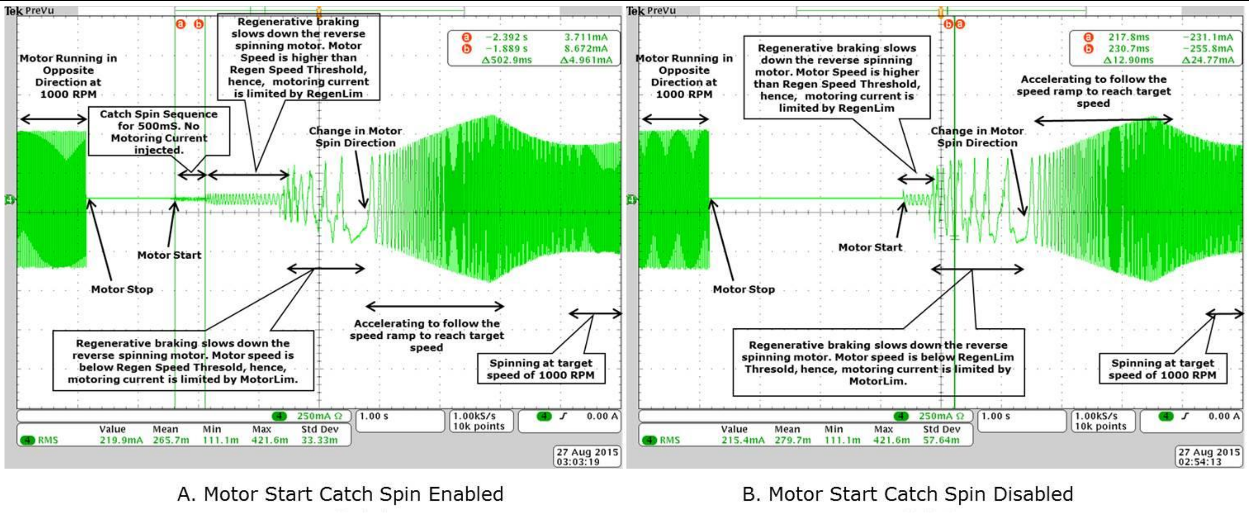


**Figure 31 Motor Phase Current Waveform - Reverse Catch Spin - Motor start with/without catch spin**

### 2.1.13 Control Input

MCE is able to control the motor from 4 types of inputs. Type of control input can be configured using Solution Designer.

- UART control
- Vsp analog input
- Frequency input
- Duty cycle input

#### 2.1.13.1 UART control

In UART control mode, motor start, stop and speed change are controlled by UART commands. Target speed can be positive or negative; motor will spin in reverse direction if Target Speed is negative. If any fault condition happens, motor will stop and stay in fault status. It is up to master controller when to clear the fault and restart the motor.

#### 2.1.13.2 Vsp Analog Input

In Vsp Analog Input control mode, the motor operations like motor start, motor stop and speed change are controlled by applying an analog voltage signal. Direction of the motor is controlled by a separate pin. If the direction pin is LOW, target speed will be set as positive and if the direction pin is HIGH, target speed will be set as negative value; motor will spin in reverse direction if target speed is negative. MCE uses “VSP” pin as the Vsp Analog input and uses “DIR” pin as motor direction input. The relationship between Vsp voltage and motor target speed is shown in Figure 32.

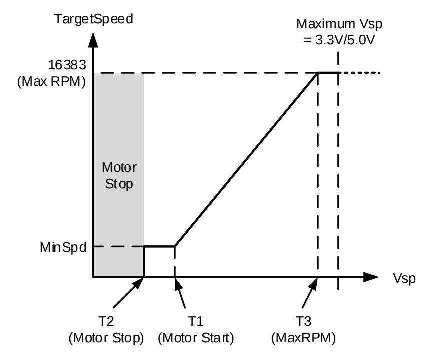


**Figure 32 Vsp Analog Input**

There are three input thresholds used to define the relationship between input voltage and target Speed.

- T1 (Input threshold for motor start): if the Vsp analog voltage is above this threshold, motor will start
- T2 (Input threshold for motor stop): if the Vsp analog voltage is below this threshold, motor will stop
- T3 (Input threshold for max RPM): if the Vsp analog voltage is higher or equal to this threshold, “TargetSpeed” variable will be 16383 which is maximum speed.

Solution Designer uses these three input thresholds to calculate the value of three parameters: “CmdStart”, “CmdStop” and “CmdGain”

$$CmdStop = Integer\left\{\left(\frac{T2*2}{Vadcref}*2048\right)+0.5\right\}$$

Where T2 = Analog Vsp Motor Stop Voltage in V.

$$CmdStart = Integer\left\{\left(\frac{T1*2}{Vadcref}*2048\right)+0.5\right\}$$

Where T1 = Analog Vsp Motor Start Voltage in V.

$$CmdGain = Integer\left\{\left(\frac{Speed_{Max}-Speed_{Min}}{Speed_{Max}}*2^{12}\right)*\left(\frac{2^{14}}{\left(\left(4096*32*\frac{T3}{Vadcref}\right)-(CmdStart*32)\right)}\right)+0.5\right\}$$

Where T3 = Analog Vsp Motor Max RPM Voltage in V

$Speed_{Max}$ = Maximum motor speed in RPM

$Speed_{Min}$ = Minimum motor speed in RPM

**Table 5 Specification for Analog Input Voltage**

| Recommended input range | Vsp Analog input (0.1V to $V_{adcref}$) |
|---|---|
| T1 | $<50\%$ of $V_{adcref}$ |
| T2* | $<50\%$ of $V_{adcref}$ |
| T3** | $< V_{adcref}$ |

*Note: * T2 must be < T1 and **T3 must be>T2*

Refer data sheet for input range for specific devices and pin details. This feature is not available in UART control mode.

### 2.1.13.3 Frequency input

In Frequency Input control mode, the motor operations like motor start, motor stop and speed change are controlled by applying a square wave frequency signal on digital IO pin. Direction of the motor is controlled by a separate pin. If the direction pin is LOW, target speed will be set as positive and if the direction pin is HIGH, target speed will be set as negative value; motor will spin in reverse direction if target speed is negative. MCE uses "DUTYFREQ" pin as the frequency input and uses "DIR" pin as motor direction input. The relationship between Frequency and motor target speed is shown in Figure 33

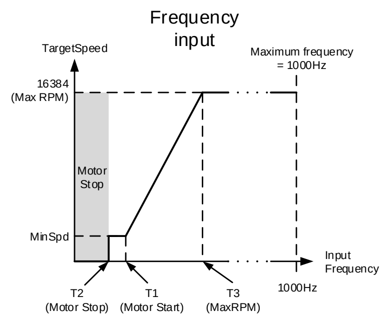


**Figure 33 Frequency Input**

There are three input thresholds used to define the relationship between frequency input and target Speed.

- T1 (Input threshold for motor start): if the frequency input is above this threshold, motor will start
- T2 (Input threshold for motor stop): if the frequency input is below this threshold, motor will stop

- T3 (Input threshold for max RPM): if the frequency input is higher or equal to this threshold, target Speed will be 16383 which is maximum speed.

Solution Designer uses these three input thresholds to calculate the value of three parameters: “CmdStart”, “CmdStop” and “CmdGain”

$$CmdStop = Integer\,\{T2 * 10 + 0.5\}$$

Where T2 = Motor Stop Speed Frequency in Hz.

$$CmdStart = Integer\,\{T1 * 10 + 0.5\}$$

Where T1 = Motor Start Speed Frequency in Hz.

$$CmdGain = Integer\left\{\left(2^{12} * \frac{\left(16384 - \left(\frac{Speed_{Min}}{Speed_{Max}} * 16384\right)\right)}{(T3 - T1) * 32 * 10}\right) + 0.5\right\}$$

Where T1 = Motor Start Speed Frequency in Hz,

T3 = Motor Max Speed Frequency in Hz,

$Speed_{Max}$ = Maximum motor speed in RPM,

$Speed_{Min}$ = Minimum motor speed in RPM.

**Table 6 Specification of Frequency Input**

| Recommended input range | Frequency input (5Hz – 1000Hz ,10% – 90% duty cycle) |
|---|---|
| T1 | ≤ 255Hz |
| T2* | ≤ 255Hz |
| T3** | ≤ 1000Hz |

*Note: * T2 must be < T1 and **T3 must be>T2*

Refer data sheet for input range for specific devices and pin details. This feature is not available in UART control mode.

#### 2.1.13.4 Duty Cycle Input Control

In Duty Cycle Input control mode, the motor operations like motor start, motor stop and speed change are controlled by varying the duty cycle of a rectangular wave signal on digital IO pin. Direction of the motor is controlled by a separate pin. If the direction pin is LOW, target speed will be set as positive and if the direction pin is HIGH, target speed will be set as negative value; motor will spin in reverse direction if target speed is negative. MCE uses “DUTYFREQ” pin as the duty input and uses “DIR” pin as motor direction input. The relationship between duty cycle and motor target speed is shown in Figure 34

In duty cycle control mode, the pre-scaler of capture timer has much wider range than frequency control mode. This allows higher input frequency in duty cycle control mode; the recommended input frequency range is 5Hz to 20 kHz. Please note that any external R/C low pass filter on the input pin may affect the duty cycle measurement especially when the input frequency is above 1 kHz.

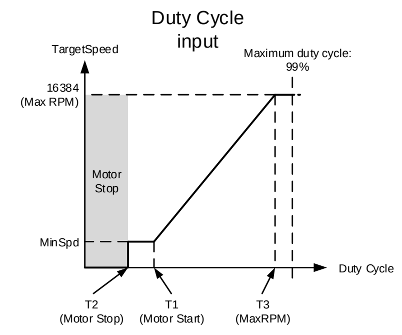


**Figure 34 Duty Cycle Input**

There are three input thresholds used to define the relationship between duty cycle input and target Speed.

- T1 (Input threshold for motor start): if the duty cycle input is above this threshold, motor will start
- T2 (Input threshold for motor stop): if the duty cycle input is below this threshold, motor will stop
- T3 (Input threshold for max RPM): if the input reaches or above this threshold, “TargetSpeed” variable will be 16383 which is maximum speed.

Solution Designer uses these three input thresholds to calculate the value of three parameters: “CmdStart”, “CmdStop” and “CmdGain”

$$CmdStop = Integer\ \{T2 * 10 + 0.5\}$$

Where T2 = Motor Stop Speed Duty Cycle in %.

$$CmdStart = Integer\ \{T1 * 10 + 0.5\}$$

Where T1 = Motor Start Speed Duty Cycle in %.

$$CmdGain = Integer\ \left\{\left(\frac{Speed_{Max} - Speed_{Min}}{Speed_{Max}} * 2^{12}\right) * \left(\frac{2^{14}}{\left((T3 * 10) - (CmdStart)\right) * 32}\right) + 0.5\right\}$$

Where T1 = Motor Start Speed Duty Cycle in %,

T3 = Motor Max Speed Duty Cycle in %,

SpeedMax = Maximum motor speed in RPM,

SpeedMin = Minimum motor speed in RPM.

Solution Designer uses these three input thresholds to calculate the value of three parameters: “CmdStart”, “CmdStop” and “CmdGain”

**Table 7 Specification of Duty Cycle Input**

| Recommended input range | Duty cycle input (5Hz – 20kHz, 1% – 99% duty cycle) |
|---|---|
| T1 | <50% |
| T2* | <50% |
| T3** | ≤ 99% |

*Note: * T2 must be < T1 and **T3 must be>T2*

Refer data sheet for input range for specific devices and pin details. This feature is not available in UART control mode.

#### 2.1.13.5 Automatic Restart

In Vsp, frequency or duty cycle control input mode, users have an option to specify retry times and intervals to restart the motor after any fault occurs and stops the motor. ‘FaultRetryNumber’ parameter configures the number of retry times after fault. A non-zero value of ‘FaultRetryNumber’ enables retry after fault. ‘FaultRetryPeriod’ parameter configures the retry interval.

This feature is not available in UART control mode.

#### 2.1.13.6 Forced control input change

If required by some debug purpose, it is possible to change the control inputs by sending UART command from master controller (or PC), and then a new mode will be effective immediately. If the control input is switched to UART control from the other three inputs, motor status (run/stop and “TargetSpeed” variable) will be unchanged until it receives a new motor control command.

#### 2.1.13.7 PG output

The MCE can output a pulse train (PG output) that represents the rotor postion. In case of Hall sensor/Hybrid mode, PG output will be enabled always irrespective of the motor state. In case of Sensorless mode, PG output will be enabled only in RUN state by default.

The ‘PGDeltaAngle’ parameter configures the PG output according to PGDeltaAngle = 256*(Motor poles)/PPR, where PPR is Pulses Per Revolution. For example, 4 PPR for an 8 poles motor (1 pulse per electrical cycle), then: PGDeltaAngle=256*8/4=512. Writing 0 to PGDeltaAngle will disable the PG output.
PG output is updated every PWM cycle, so the maximum PG output frequency is ½ Fpwm. The maximum value for PGDeltaAngle is 16383, which means 1 PG pulse take 32 electrical cycle (16384/512=32), on an 8 poles motor, the PG output will be 0.125PPR.
If PGDeltaAngle is $2^n$ (2,4,8,16….8192,16384), PG pulse will be synchronized with rotor angle. For example, if PGDeltaAngle=512 for an 8 poles motor (4PPR). There are 4 PG pulses every 4 electrical cycles and the PG transition (high to low or low to high) will happen at 0 and 180 electrical degree.

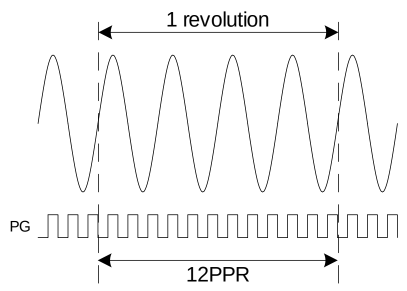


**Figure 35 PG Output**

#### 2.1.13.8 Control Input Customization

By default, the relationship between the control input (VSP analog input / frequency input / duty cycle input) and the motor target speed is linear as shown in Figure 32, Figure 33, and Figure 34. If an application requires implementation of an arbitrary mapping relationship between the control input and the motor target speed, then one can choose to disable the default linear control input method and use script language to realize control input customization.

For VSP analog input control method, the analog input voltage can be read from' 'adc_result0' variable using script.

To enable frequency or duty input control customization, one needs to set the 6th bit of 'AppConfig' variable, so that the 'FrequencyInput' and 'DutyInput' variables get updated with the relevant frequency and duty cycle measurement results every 10 ms. Supported input frequency range: 5Hz – 5000Hz. Supported input duty cycle range: 1% - 99%.

For frequency input control method, the measured input frequency can be read from 'FrequencyInput' variable using script.

For duty cycle input control method, the measured input duty cycle can be read from 'DutyInput' variable using script.

### 2.1.14 Hall Sensor Interface

The MCE Hall angle extraction algorithm estimates rotor angle and velocity signals per motor PWM cycle from the four times (2 Hall sensors) or six times (3 Hall sensors) per electrical cycle digital Hall input transition events. The optional Atan angle algorithm extracts rotor angle and velocity signals per motor PWM cycle from the two analog Hall sensor signals.

The MCE Hall sensor interface supports the following Hall sensor configurations as shown in Table 8.

**Table 8 Supported Hall Sensor Configurations**

| Interface Type | Supported Configuration | Sensor Displacement (Electrical Angle) |
|---|---|---|
| Digital | 2 / 3 digital Hall sensors | 120° |
| Analog | 2 analog Hall sensors | 120° |

#### 2.1.14.1 Interface Structure

As shown in the following Figure 36, the analog Hall sensor positive and negative outputs are connected to non-inverting and inverting inputs of the internal comparators with configurable hysteresis (bit field [7:6] of parameter 'HallConfig') respectively. During every Hall zero-crossing event between AHALLx+ and AHALLx- (x = 1, 2), the relevant comparator output toggles accordingly. The internal comparator outputs are connected via a multiplexer to H1 and H2 inputs of the Hall Event Capture block. The analog Hall sensor outputs are also connected to the four internal ADC channels through an equivalent gain stage of 1 for the purpose of sampling analog Hall sensor output voltage values, which are used to calculate Atan angle when Hall Atan angle calculation method is enabled.

The digital Hall sensor outputs are directly connected via the multiplexer to the corresponding H1, H2, and H3 inputs of the Hall Event Capture block, whose outputs are Hall event timing information and the Hall pattern that are used by Hall PLL block to estimate Hall angle and Hall speed.

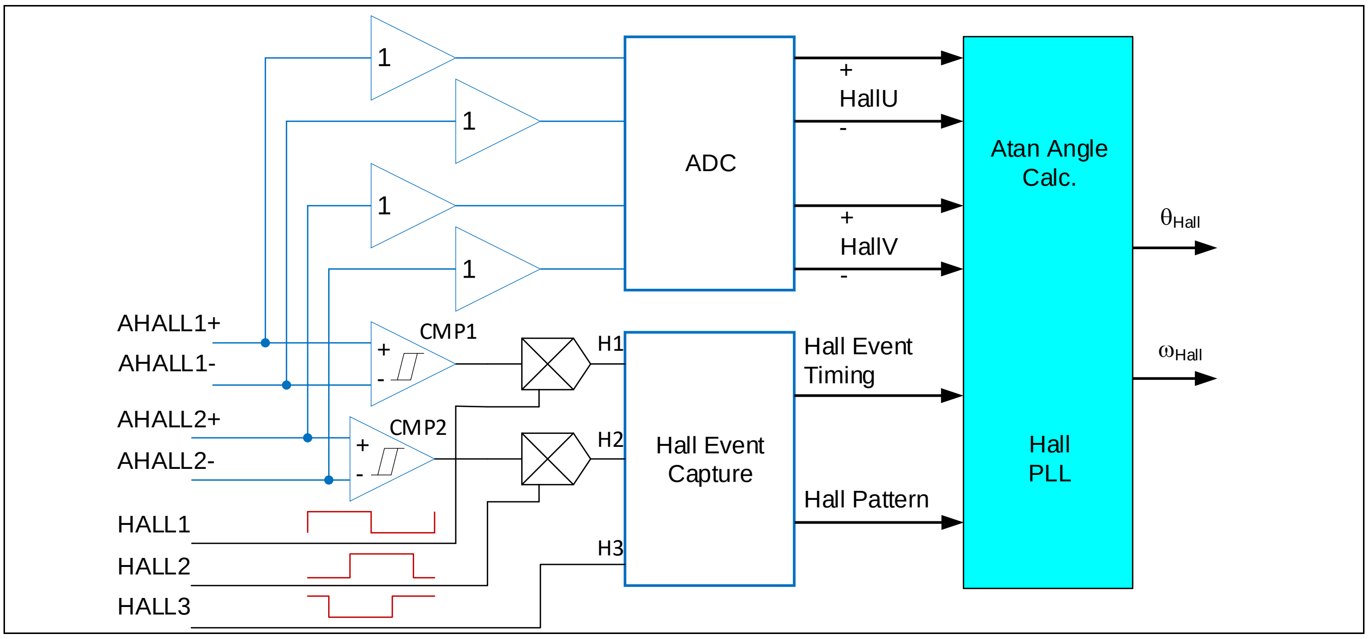


Figure 36 **Hall Sensor Interface High-Level Structure Overview**

### 2.1.14.2 Hall Sample De-Bounce Filter

A hardware noise filter is included in the Hall event capture block to provide de-bounce check mechanism before sampling Hall inputs. The noise filter timing mechanism is shown in the following Figure 37. Whenever there comes a transition detected at H1, H2 or H3 input, its status is not sampled until after a configurable de-bounce time ($T_{DB}$) has elapsed. If there comes another transition before $T_{DB}$ has elapsed, then the scheduled following sampling operation is cancelled and the de-bounce time counting starts over. This de-bounce time can be configured by using the parameter 'SampleFilter' following this equation $T_{DB} = SampleFilter \times 10.417ns$.

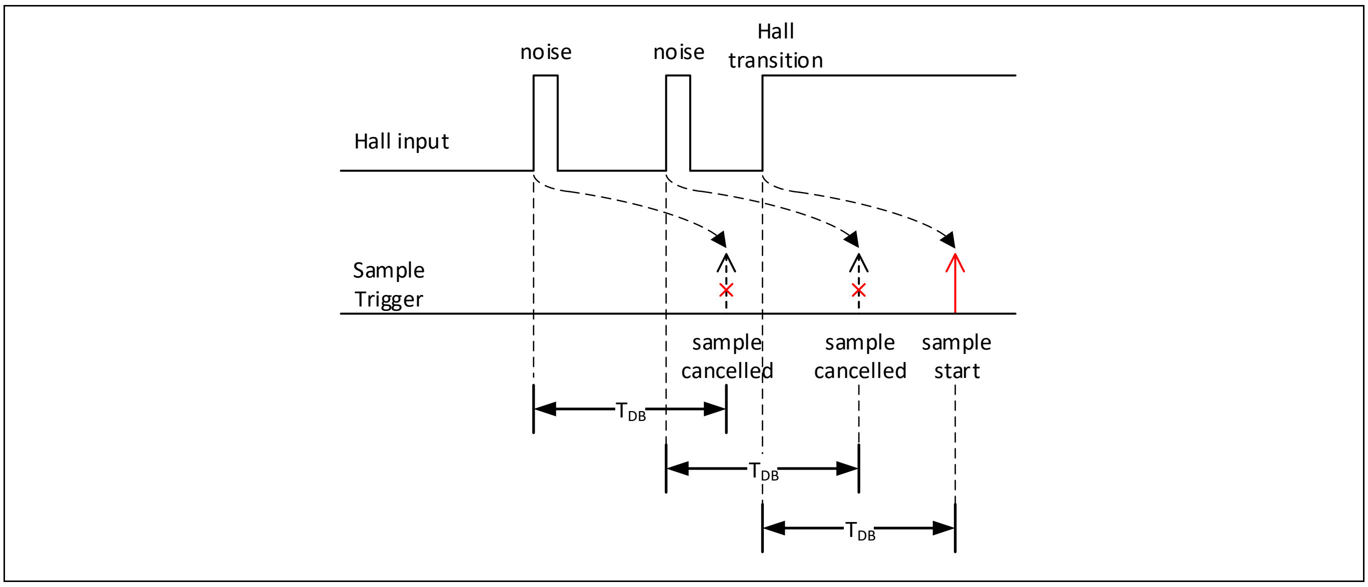


Figure 37 **Hall Sensor Noise Filter Timing Diagram**

#### 2.1.14.3 Hall Angle Estimation

Digital Hall sensors or comparator based analog Hall sensor interface provide discrete angle inputs at each Hall transition event. For 3 digital Hall sensor configuration, a Hall transition event occurs every 60° electrical angle. For 2 digital Hall sensor configuration, a Hall input transition event occurs every 60° electrical angle (normal sector) or 120° electrical angle (wide sector) alternately. For 2 analog Hall sensor configuration, the two internal comparators are used to detect zero-crossing events, and the corresponding Hall transition event occurs the same way as in the case of 2 digital Hall sensor configuration.

The MCE's Hall angle estimation algorithm estimates Hall angle between sequential hall transition events by integrating a Hall frequency estimate. It takes advantage of a PLL loop to keep track of the actual Hall frequency and correct angle estimation error by subtracting a compensation term to the Hall frequency integrator over the next Hall transition event cycle.

The status of the digital inputs of the Hall Event Capture block is sampled by the MCE's hardware peripheral when a Hall transition event occurs. The sampled Hall inputs form certain Hall pattern as described in Section 2.1.14.3.1.

The MCE's Hall angle estimation routine is executed during each motor PWM cycle. Details of the Hall angle estimation process is described in the following two sub-sections.

##### 2.1.14.3.1 Hall Angle Estimation with PLL

When Hall PLL is enabled (KpHallPLL > 0), the Hall angle estimation algorithm is depicted in the following Figure 38.

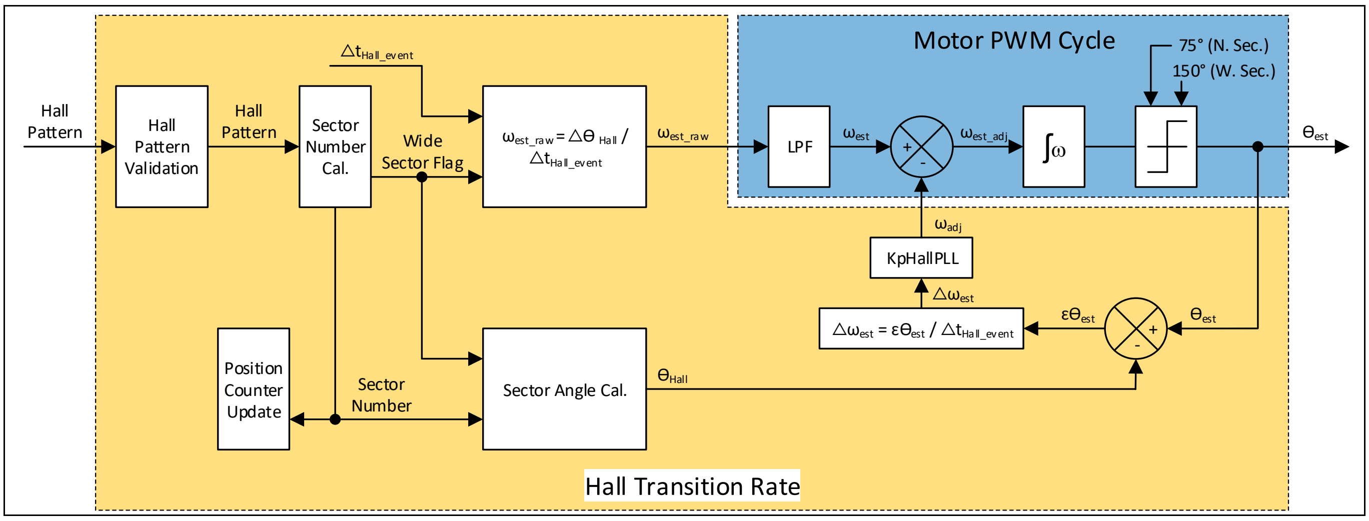


**Figure 38 Hall Angle Estimation Algorithm Diagram (Hall PLL Enabled)**

During each motor PWM cycle, the MCE's Hall angle estimation routine integrates the difference $\omega_{est\_adj}$ between the low-pass filtered Hall frequency estimate $\omega_{est}$ (variable 'HallFreq') and a compensation term $\omega_{adj}$ to generate the estimated Hall angle $\theta_{est}$ as shown in the blue block in Figure 38. If the estimated Hall angle increment is accumulated up to 75° (normal sector) or 150° (wide sector) since last Hall transition event, no further integration is performed and $\theta_{est}$ stays flat until next Hall transition event occurs.

The MCE also checks if there occurs a new Hall transition event since last check during each motor PWM cycle. If there exists a new Hall transition event, the following steps are performed as shown in the orange block in Figure 38.

The newly sampled Hall pattern is first validated against an expected pattern based on rotating direction.

If it is validated successfully, then a corresponding new sector number  is calculated based on a mapping relationship between Hall patterns and sector numbers as shown in Figure 40.

The Hall position counter as described in 2.1.14.6 is incremented or decremented accordingly based on rotating direction when the sector number is updated.

Next, raw Hall frequency estimate $\omega_{est\_raw}$, which represents the amount of angle change per PWM cycle, is calculated as the result of the division of angle difference between the two sequential Hall transition events $\Delta\theta_{Hall}$ and the time interval $\Delta t_{Hall\_event}$ between the two sequential Hall transition events ($\omega_{est\_raw} = \frac{\Delta\theta_{Hall}}{\Delta t_{Hall\_event}}$). If the time interval between the two sequential Hall transition events $\Delta t_{Hall\_event}$ is longer than 4096 PWM cycles, then it is considered timed out and $\omega_{est\_raw}$ is rest to zero. The updated raw Hall frequency estimate is low-pass filtered with a configurable time constant $T_{decay}$. To achieve a desired bandwidth $\omega_c = \frac{1}{T_{decay}}$ for this low-pass filter, please follow this equation to calculate the value for the variable 'FrequencyBW': $FreqeuncyBW = 2^{16} \cdot (1 - e^{-\frac{\omega_C \cdot Fast\_Control\_Rate}{F_{PWM}}})$. The filtered Hall frequency estimate $\omega_{est}$ (variable 'HallFreq') is available to be used for Hall angle estimation.

Next, the actual Hall angle $\theta_{Hall}$ is calculated based on the updated sector number and the rotating direction and wide sector flag. The estimated Hall angle $\theta_{est}$ is not adjusted immediately at each Hall transition event. The Hall angle estimation error $\varepsilon\theta_{est}$ is corrected by adding a compensation term $\omega_{adj}$ to the Hall frequency integrator ove the next Hall transition event cycle.  The frequency compensation term $\omega_{adj}$ is calculated as the product of a proportional factor (parameter 'KpHallPLL') and the division of the angle estimation error $\varepsilon\theta_{est}$ by the time interval between the two sequential Hall transiton events ($\omega_{adj} = KpHallPLL \times \frac{\varepsilon\theta_{est}}{\Delta t_{Hall\_event}}$). If the angle estimation error $\varepsilon\theta_{est}$ is greater than 15° (normal sector) or 30° (wide sector), then the estimated Hall angle $\theta_{est}$ is reset to the value of the actual Hall angle $\theta_{Hall}$, and the compensation term $\omega_{adj}$ is reset to 0.

Finally, the variable 'HallAngle' is updated following this equation: $HallAngle = \theta_{est} + HallAngleOffset$ . The configuration of the parameter 'HallAngleOffset' is described in Section 2.1.14.7. The parameter 'HallSpeed' is updated from the product of the low-pass filtered Hall frequency estimate $\omega_{est}$ with coresponding scaling factors.

With Hall PLL enabled, the angle estimation error $\varepsilon\theta_{est}$ is corrected over the following Hall event cycles, so that the estimated Hall angle $\theta_{est}$ wouldn't jump abruptly at each Hall transition event. Thus, the motor is expected to run relatively more smoothly when it is accelerating or decelerating. It is recommended to take advantage of Hall PLL by selecting a value for parameter 'KpHallPLL' between 0 and 4096 for better performance when using Hall sensors.

Users are advised to select the value of the parameter 'KpHallPLL' with the consideration of the trade-offs between torque / speed dynamics and operational smoothness depending on different application requirements. For example, door opener applications may prefer higher dynamics of torque, while fan applications may favor operational smoothness over torque dynamics. Higher 'KpHallPLL' value provides quicker speed/torque response with the compromise of operational smoothness due to sudden change of estimated Hall speed and angle. Lower 'KpHallPLL' value provides smoother torque / speed change while sacrificing dynamics.

### 2.1.14.3.2 Hall Angle Estimation without PLL

When Hall PLL is disabled (KpHallPLL = 0), the Hall angle estimation algorithm is depicted in the following Figure 39.

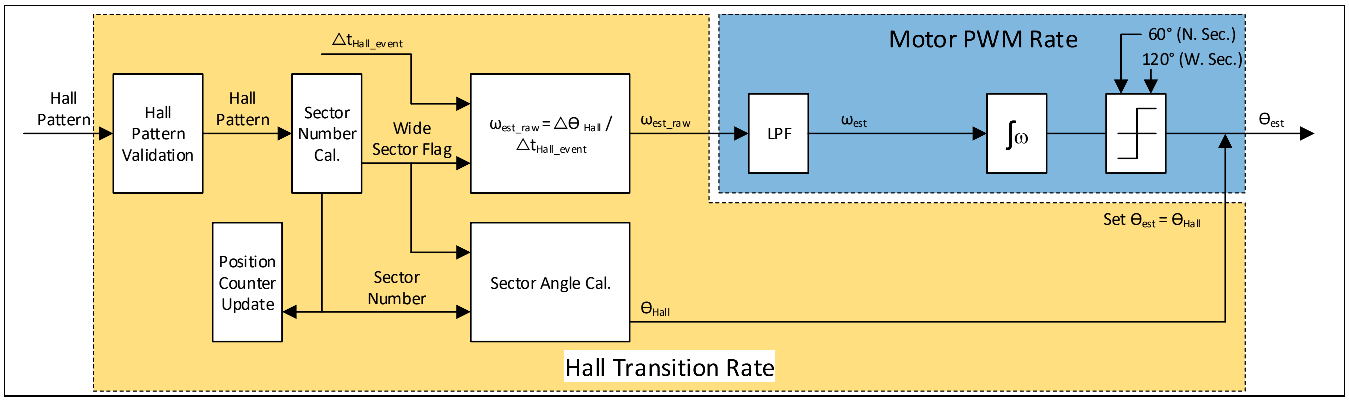


**Figure 39 Hall Angle Estimation Algorithm Diagram (Hall PLL Disabled)**

During each motor PWM cycle, the MCE's Hall angle estimation routine integrates the low-pass filtered Hall frequency estimate $\omega_{est}$ to generate the estimated Hall angle $\theta_{est}$ as shown in the blue block in Figure 39. If the estimated Hall angle increment is accumulated up to 60° (normal sector) or 120° (wide sector) since last Hall transition event, no further integration is performed and $\theta_{est}$ stays flat until next Hall transition event occurs.

The MCE also checks if there occurs a new Hall transition event since last check during each motor PWM cycle. If there exists a new Hall transition event, the MCE performs a similar set of steps compared to the scenario with PLL enabled as shown in the orange block in Figure 39.

Hall pattern validation, sector number calculation, position counter update, Hall frequency estimate calculation, actual Hall angle calculation, the variable 'HallAngle' and 'HallSpeed' update steps are the same as those in the scenario with PLL enabled.

The step that differs is the angle estimation error correction. The angle estimation error $\varepsilon\theta_{est}$ is corrected by adding the latest angle estimation error $\varepsilon\theta_{est}$ to the estimated Hall angle $\theta_{est}$ at each Hall transition event. In other words, the estimated Hall angle $\theta_{est}$ is reset to the actual Hall angle $\theta_{Hall}$ at each Hall transition event.

With Hall PLL disabled, when the motor is accelerating or decelerating, the estimated Hall angle $\theta_{est}$ would jump abruptly at each Hall transition event.

### 2.1.14.4 Hall Zero-Speed Check

When the motor control state machine is in 'MOTORRUN' state, if the time interval between the two sequential Hall transition events is longer than a threshold $T_{zf}$, then it is considered as an Hall zero frequency fault. The threshold $T_{zf}$ is calculated following this equation $T_{zf} = 4096 \times T_{PWM}$. Once the time interval between the two sequential Hall transition events is shorter than the threshold $T_{zf}$, this fault is automatically cleared.

The equivalent motor speed that would trigger Hall zero frequency fault consistently with 2 or 3 digital Hall sensor configurations can be calculated as follows:

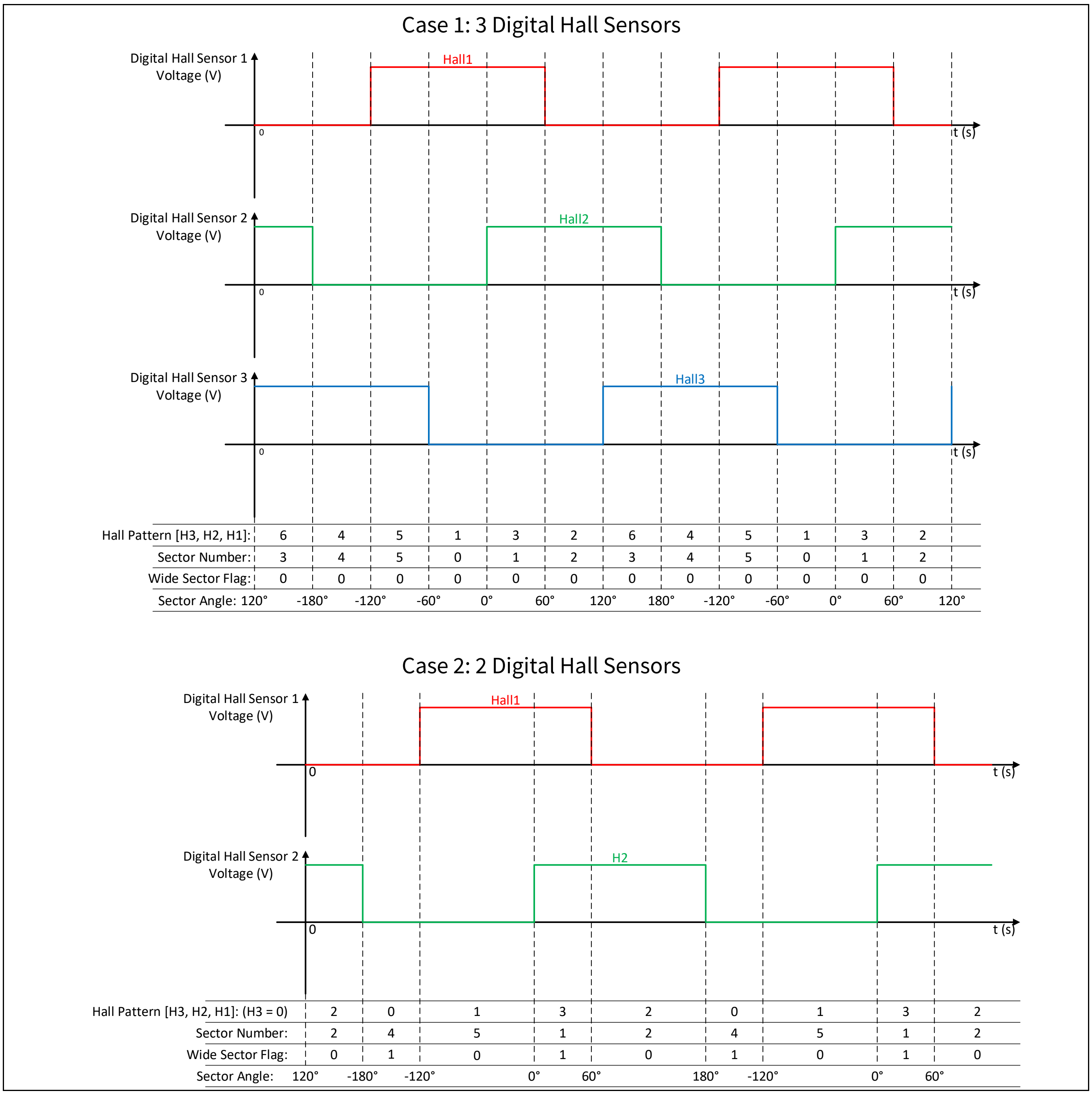
$$\omega_{zf\_3Hall}(rpm) = \frac{1}{4096 \times T_{PWM}} \times \frac{1}{6} \times \frac{60}{pole\_pair}$$

If this Hall zero frequency fault lasts as long as $T_{HallTimeOut}$, then a 'Hall Timeout' Fault is confirmed. The threshold $T_{HallTimeOut}$ can be configured using the parameter 'HallTimeoutPeriod' following this equation: $T_{HallTimeOut} = HallTimeoutPeriod \times 16ms$.

This fault is to detect rotor lock condition when Hall sensors are being used.

### 2.1.14.5 Hall Pattern Validation

Hall pattern is formed as a binary number ([H3, H2, H1]$_b$) by using the 3 digital inputs, H1, H2 and H3 of Hall Event Capture block, and assumes that H3 is bit 2, H2 is bit 1, and H1 is bit 0 as shown in the following Figure 40. For example, if H3 is logic high, H2 is logic low, and H3 is logic high, then the Hall pattern is recognized as [101]$_b$ = 5.


Case 1: 3 Digital Hall Sensors
Digital Hall Sensor 1 Voltage (V)
Hall1
0
t (s)
Digital Hall Sensor 2 Voltage (V)
Hall2
0
t (s)
Digital Hall Sensor 3 Voltage (V)
Hall3
0
t (s)

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Hall Pattern [H3, H2, H1]: | 6 | 4 | 5 | 1 | 3 | 2 | 6 | 4 | 5 | 1 | 3 | 2 |
| Sector Number: | 3 | 4 | 5 | 0 | 1 | 2 | 3 | 4 | 5 | 0 | 1 | 2 |
| Wide Sector Flag: | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

Sector Angle: 120° -180° -120° -60° 0° 60° 120° 180° -120° -60° 0° 60° 120°

Case 2: 2 Digital Hall Sensors
Digital Hall Sensor 1 Voltage (V)
Hall1
0
t (s)
Digital Hall Sensor 2 Voltage (V)
H2
0
t (s)

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| Hall Pattern [H3, H2, H1]: (H3 = 0) | 2 | 0 | 1 | 3 | 2 | 0 | 1 | 3 | 2 |
| Sector Number: | 2 | 4 | 5 | 1 | 2 | 4 | 5 | 1 | 2 |
| Wide Sector Flag: | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

Sector Angle: 120° -180° -120° 0° 60° 180° -120° 0° 60°

**Figure 40 Calculation of Hall Pattern, Sector Number, Wide Sector Flag, and Sector Angle from Hall Inputs**

Hall pattern validation starts by comparing the newly sampled Hall pattern with an expected Hall pattern from a pre-determined Hall pattern sequence based on motor rotating direction.

If the newly sampled Hall pattern is [111] or [000], then it is considered as an invalid pattern fault. If two consecutive occurrences of the invalid pattern fault are detected, then 'Hall Invalid' fault is confirmed and the

15th bit of variable 'FaultFlags' is set. Notice this invalid pattern check is only applicable to 3 digital Hall configuration.

If the newly sampled Hall pattern is valid but doesn't match either the expected Hall pattern from the CW rotating Hall pattern sequence or from the CCW rotating Hall pattern sequence, then it is considered as an unexpected pattern fault. If three consecutive occurrences of the unexpected pattern fault are detected, then 'Hall Invalid' fault is confirmed and the 15th bit of variable 'FaultFlags' is set.

If the newly sampled Hall pattern is validated successfully, then a new sector number (0~5) is extracted based on a mapping relationship between Hall patterns and sector numbers as shown in Figure 40.

#### 2.1.14.6 Hall Position Counter

The MCE maintains a 32-bit position counter that can be used to keep track of the position of the motor loads in some applications such as garage doors or blinds. When a new Hall transition event is validated and the sector number is updated by the MCE, the position counter is incremented with CW direction or decremented with CCW direction. The increment or decrement step is 1 count for normal sector (60° displacement) or 2 counts for wide sector (120° displacement). In other words, the position counter is a sector counter.

The value of the position counter can be read from the parameter 'PositionCounter' and 'PositionCounter_H'.

#### 2.1.14.7 Hall Angle Offset

For 2 analog Hall sensor configuration, assume that the angle difference between the zero-crossing of UV line to line back-EMF voltage waveform and the zero-crossing of analog Hall 1 differential waveform is defined as $\theta_{offset}$ as shown in the following Figure 41.

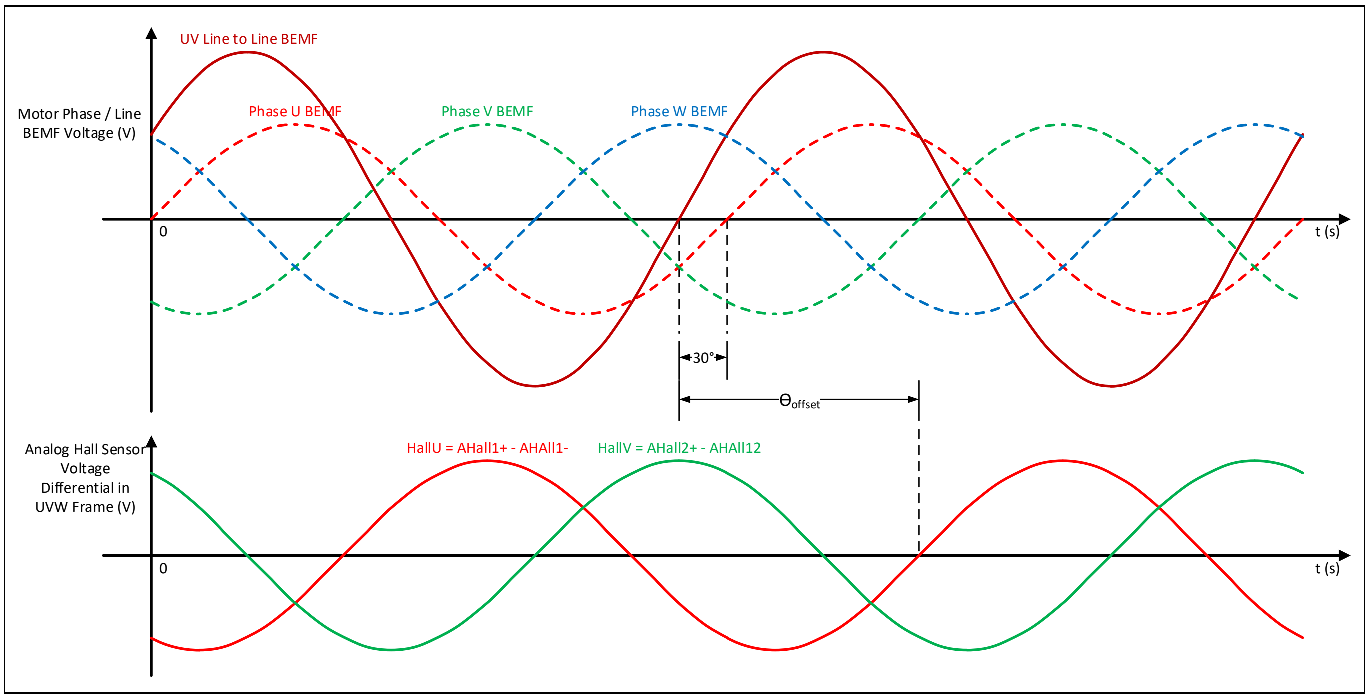


**Figure 41 Angle Offset Definition Diagram for 2 Analog Hall Sensor Configuration**

For 2 or 3 digital Hall sensor configuration, assumes that the angle difference between the zero-crossing of UV line to line back-EMF voltage waveform and the zero-crossing of analog Hall 1 differential waveform is defined as $\theta_{offset}$ as shown in the following Figure 42.

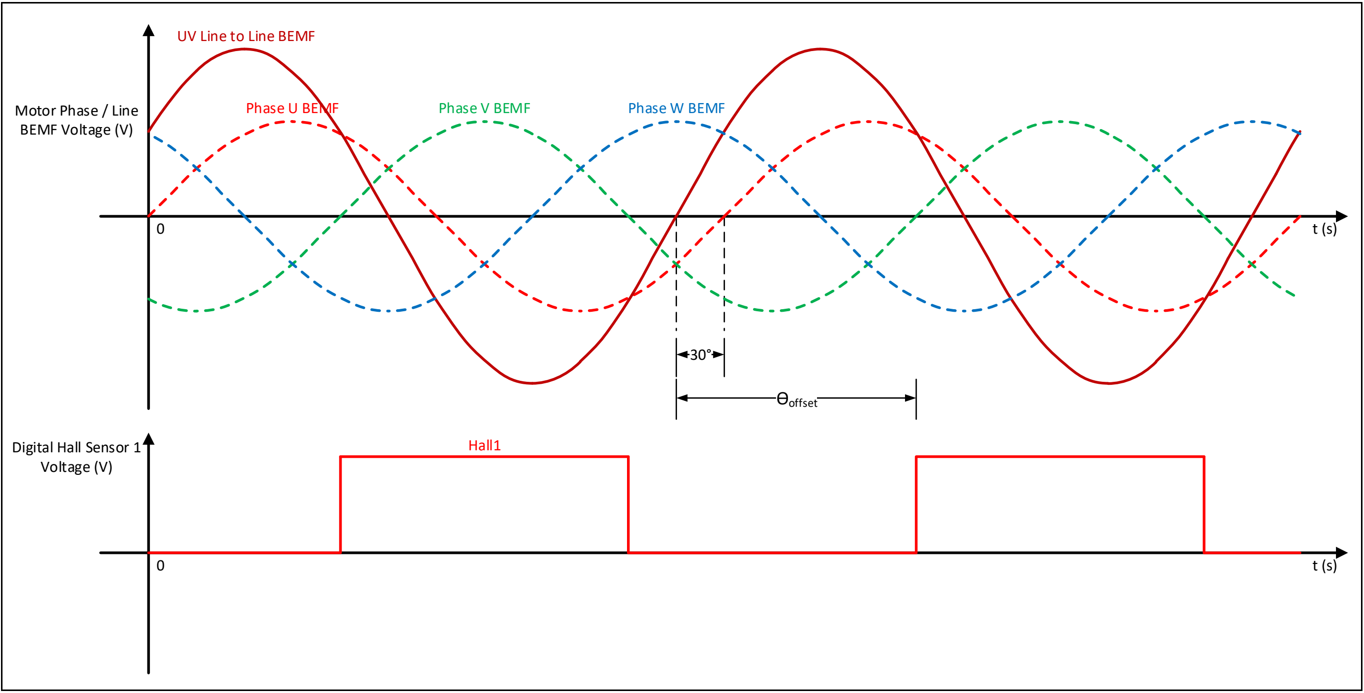


**Figure 42 Angle Offset Definition Diagram for 2 or 3 Digital Hall Sensor Configuration**

The parameter 'HallAngleOffset' shall be calculated following this equation:

$$HallAngleOffset = (\theta_{offset} - 90°) \times \frac{16384}{90°}$$

This parameter is used in the final calculation of variable 'HallAngle' during each motor PWM cycle to compensate for the angle difference between the rotor position and the Hall sensor (DHALL1 or AHALL1) mounting position.

#### 2.1.14.8 Atan Angle Calculation

The above-mentioned Hall angle calculation method based on Hall zero-crossing events using comparators renders a variable estimated angle error correction rate. The lower the motor speed is, the longer it takes for a Hall input transition event to occur, and the lower the estimated angle error correction rate becomes. As a result, when the motor is starting up using this Hall angle calculation method, it is inevitable that the estimated Hall angle would not accumulate smoothly during the first a few sectors due to the nature of lower Hall input transition event occurrence rate. This would sometimes cause undesirable acoustic noise and unsmooth motor start-up performance.

The MCE provides an Atan angle calculation method to complement the estimated Hall angle calculation during start-up for 2 analog Hall sensor configuration. The Atan angle calculation method can be enabled or disabled by using the 5th bit of the parameter 'HallConfig'. When Atan angle calculation method is enabled and Hall angle or hybrid angle is selected by the parameter 'AngleSelect'. Parameter HallATanPeriod specifies the number of sectors for which Hall Atan angle, represented by the parameter 'Atan_Angle', is being used as rotor angle during start-up.

The Atan angle calculation process is shown in the following Figure 43. The analog Hall sensor input (AHALL1+, AHALL1-, AHALL2+, and AHALL2-) voltage levels are sampled during each motor PWM cycle, and the voltage differential of each analog Hall sensor is calculated as HallU = AHALL1+ - AHALL1-, and HallV = AHALL2+ - AHALL2-. The MCE performs Clarke transformation to convert HallU and HallV components in UVW reference frame to Hallα and Hallβ components in a stationary αβ reference frame. Then Atan$(\frac{Hall_\beta}{Hall_\alpha})$ calculation is performed to generate Atan angle represented by the variable 'Atan_Angle' with the addition of Hall angle offset specified by the parameter 'HallAngleOffset'.

Using the complementary Atan angle calculation method, the rotor angle using Atan angle is expected to accumulate more smoothly with minimal acoustic noise during motor start-up compared to using the above-mentioned Hall angle estimation method. The analog Hall sensor signals bear higher order harmonics in some cases. As a result, the Atan angle calculation would yield undesired fluctuation that is not a true reflection of the rotor speed variation. Consequently, it is recommended to limit the usage of Hall Atan angle calculation method to a short duration during start-up for just several number of sectors as needed by configuring the parameter 'HallATanPeriod.

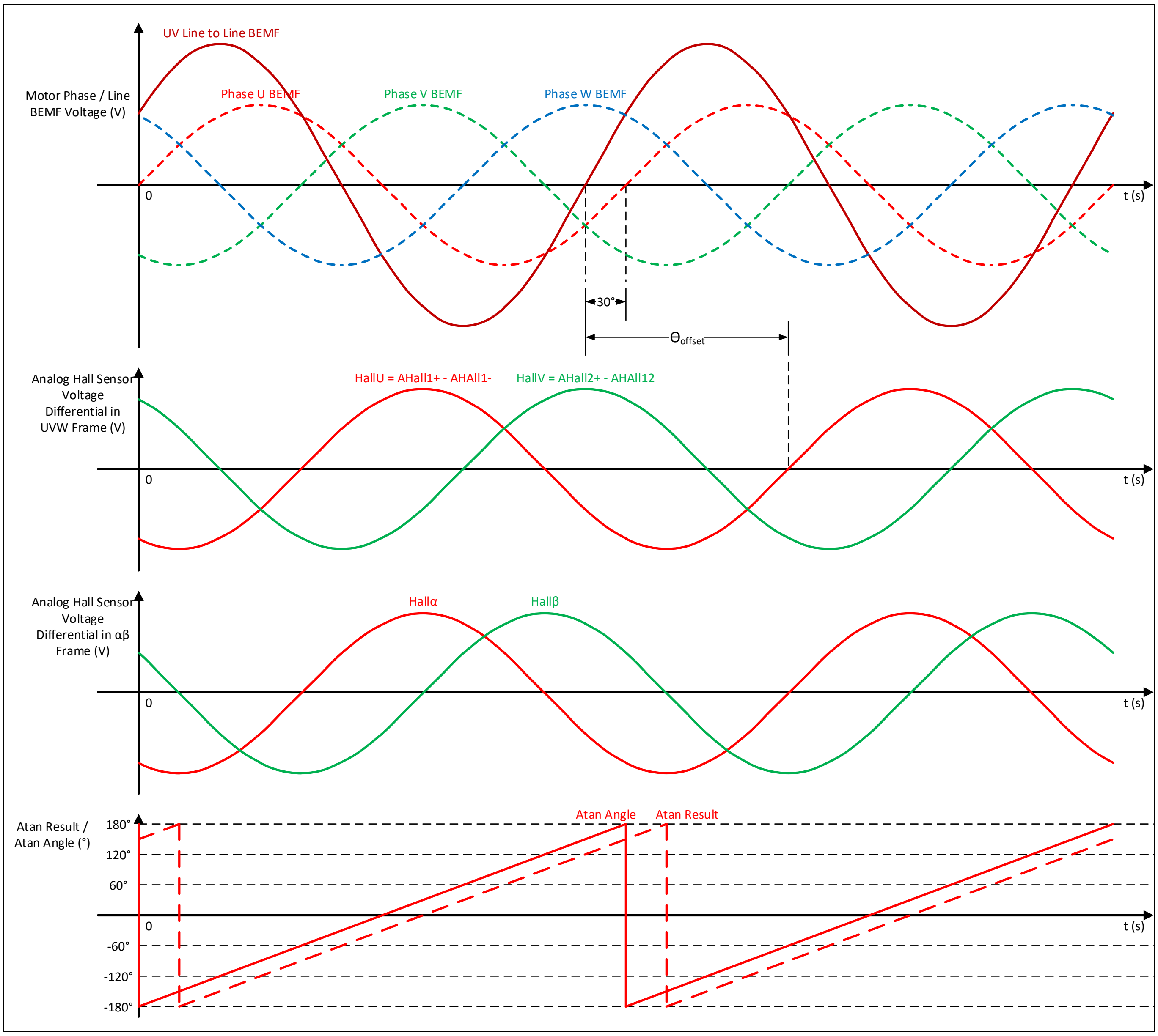


**Figure 43 Atan Angle Calculation Based on 2 Analog Hall Sensor Inputs (AHall1+, AHall1-, AHall2+, AHall2-)**

### 2.1.14.9 Hall Initial Position Estimation

For digital 2 or 3 Hall configurations, as well as analog 2 Hall configuration without using Atan angle calculation method, the initial rotor position at the start-up is estimated by the MCE based on the initial Hall inputs. The MCE assumes that the rotor starts in the middle of the angle range which is interpreted from the initial Hall pattern. The following Table 9 and Table 10 show the initial angle estimation details for 3 Hall and 2 Hall configurations.

**Table 9 Hall Initial Position Estimation (3 Hall, HallAngleOffset = 0)**

| Hall pattern [H3, H2, H1] | 1 ($001_b$) | 3 ($011_b$) | 2($010_b$) | 6($110_b$) | 4($100_b$) | 5($101_b$) |
|---|---|---|---|---|---|---|
| Angle range | -60° to 0° | 0° to 60° | 60° to 120° | 120° to 180° | 180° to 240° | 240° to 300° |
| Initial angle | -30° | 30° | 90° | 150° | 210° | 270° |

**Table 10 Hall Initial Position Estimation (2 Hall, HallAngleOffset = 0)**

| Hall pattern [H3, H2, H1] (H3 = 0) | 1 ($001_b$) | 3 ($011_b$) | 2 ($010_b$) | 0 ($000_b$) |
|---|---|---|---|---|
| Angle range | -120° to 0° | 0° to 60° | 60° to 180° | 180° to 240° |
| Initial angle | -60° | 30° | 120° | 210° |

### 2.1.14.10 Hall Sensor / Sensorless Hybrid Operation

The MCE supports a hybrid mode where both the Hall sensor interface driver and the flux estimator and flux PLL are active. As shown in the following Figure 44, the rotor angle uses estimated Hall angle from the Hall sensor interface driver during the start-up. As the motor speed increases to more than the Hall-to-Flux speed threshold configured by the parameter 'Hall2FluxThr', the rotor angle switches over to using estimated flux angle from the flux estimator and flux PLL. While the rotor angle is fed from flux angle, if the motor speed decreases to below the Flux-to-Hall speed threshold configured by the parameter 'Flux2HallThr', the rotor angle switched back to using estimated Hall angle from the Hall sensor interface driver. In hybrid mode, both the Hall sensor interface driver and the flux estimator and flux PLL are running concurrently although only one out of the two outputs is being used as rotor angle to close the angle loop.

While the MCE offers an advanced sensorless algorithm with excellent performance, some applications require better performance at start-up and / or very low speed operations. In this case, using Hall sensors can complement the sensorless option in providing superior start-up and low speed performance. Thus, it is recommended to select hybrid mode to take advantage of both the sensorless mode and the Hall sensor mode to ensure a consistent high performance of a drive system across a wide speed range including start-up.

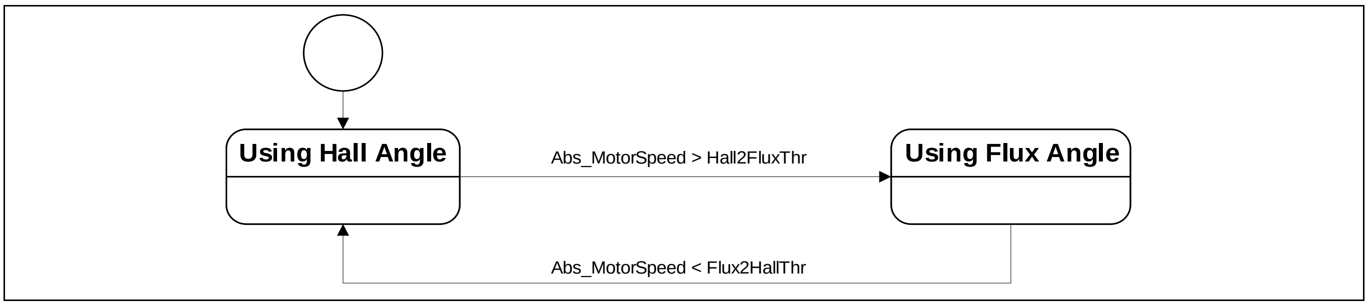


**Figure 44 Hall Sensor / Sensorless Hybrid Mode Diagram**

### 2.1.15 Torque Compensation

For single rotary compressor based air-conditioners or refrigerator applications, big variation in the torque demand exists within a mechanical cycle from absorption stage and compression stage. Because of the limited speed loop bandwidth, the motor speed would vary due to the varying torque demand within one mechanical cycle, causing noticeable mechanical vibration and undesirable noise. To solve this problem, the MCE provides a torque compensation function that is able to detect and synchronize with the mechanical cycle and uses a feed forwarding loop to modulate torque reference following a sinusoidal compensation curve per mechanical cycle to minimize speed variation and thus reduce vibration. This function uses the torque reference and flux angle (rotor electrical angle in sensorless mode) as inputs. It has two primary operating modes over a configurable speed range. In one mode it synchronizes with the peak load torque within a mechanical cycle, while in the other mode it calculates the feedforward compensation torque. The MCE's torque compensation function supports 4-pole or 6-pole compressor motor types.

The torque compensation function can be enabled or disabled by using bit [1] of the parameter 'SysConfig'.

Figure 45 depicts the state transitions for the MCE's torque compensation function.

When torque compensation is disabled by resetting bit [1] of the parameter 'SysConfig', it stays in TC_Disabled state and goes through an initialization process (TorqueComp_init()) where relevant variables including 'TrqCompBaseAngle', 'TrqCompStatus' and 'TrqCompOutput' are reset.

When torque compensation is enabled, by setting bit [1] of the parameter 'SysConfig', it shifts to TC_Enabled state.

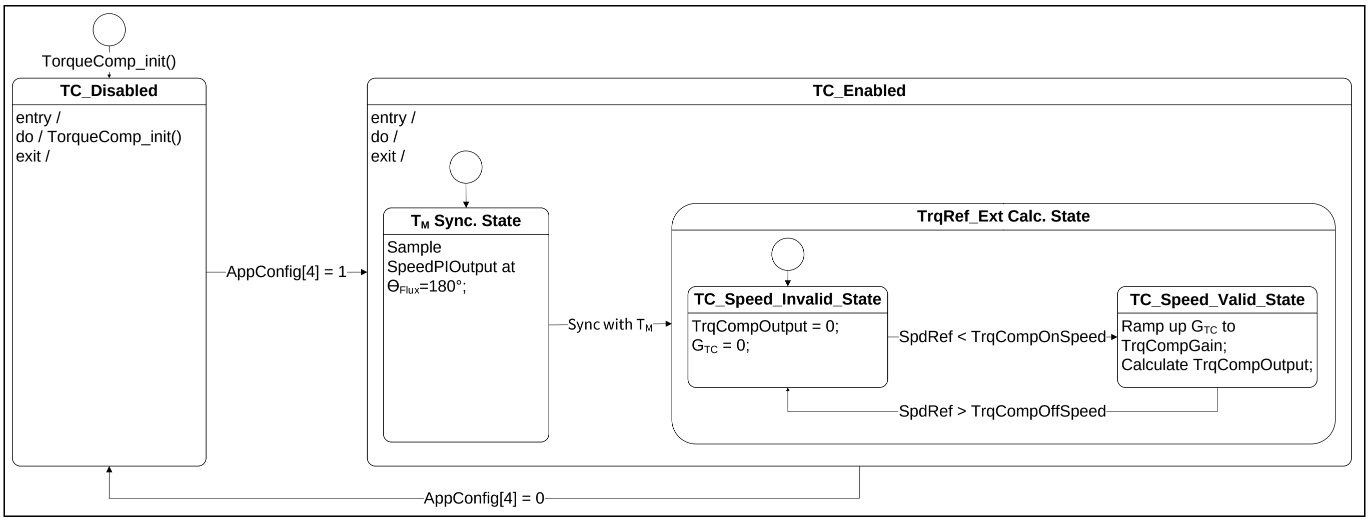


**Figure 45 Torque Compensation State Transition Diagram**

As shown in Figure 45 and Figure 46, there are 2 sub-states within TC_Speed_Valid state. When entering TC_Speed_Valid state, it starts from $T_M$ Synchronization sub-state where it samples the torque reference (variable 'SpeedPIOutput') once every electrical cycle when flux angle $\theta_{Flux} = 180°$. If the torque reference samples at the $k_{th}$ sample time match the following criteria: SpeedPIOutput [k] > SpeedPIOutput [k-1], SpeedPIOutput [k] > SpeedPIOutput [k-2], SpeedPIOutput [k-3] > SpeedPIOutput [k-1], SpeedPIOutput [k-3] > SpeedPIOutput [k-2] for 10 consecutive mechanical cycles $T_M$, then it is considered as having synchronized with peak load torque within a mechanical cycle $T_M$, and that moment marks the zero point of torque compensation base angle $\theta_{base}$ (variable 'TrqCompBaseAngle'). Then it shifts to TrqCompOutput Calculation sub-state.

There are two sub-states inside TrqCompOutput Calculation sub-state. If the motor speed reference (variable 'SpeedRef) is lower than the turn-on threshold configured by the parameter 'TrqCompOnSpeed', then it shifts to TC_Speed_Valid sub-state where torque compensation function becomes active. While it is in TC_Speed_Valid sub-state, if the motor speed reference becomes higher than the turn-off threshold configured by the parameter 'TrqCompOffSpeed', then it shifts back to TC_Speed_Invalid sub-state where torque compensation fuction becomes inactive with $TrqCompOutput$ and $G_{TC}$ being reset to zero.

It shall be pointed out that once the torque compensation function achieves synchronization with mechanical cycle $T_M$, it doesn't lose synchronization with mechanical cycle $T_M$ whether the motor speed reference is within the valid speed range (active) or not (inactive).

The active status of torque compensation function is reflected in bit[0] of variable 'TrqCompStatus'.

When torque compensation fuction is active, the desired sinusoidal compensation torque reference (variable 'TrqCompOutput') is synthesized following this equation:

$$TrqCompOutput = G_{TC} \times TrqRef_{Filt} \times \cos(\frac{\theta_{Flux}-180°}{pole\_pair} + \theta_{base} + \theta_{os}) \times k_{CORDIC}.$$

$G_{TC}$represents the gain factor for the desired compensation torque reference 'TrqCompOutput'.

$TrqRef_{Filt}$represents the averaged value of the desired torque reference from speed PI regulator output. It is the low-pass filtered result from variable 'SpeedPIOutput' with upper limit. As shown in Figure 46, if $TrqRef_{Filt}$ is greater than the value of the parameter 'TrqCompLim', then it is limited to the value of 'TrqCompLim'.

$k_{COR}$ is an internal fixed gain factor ($k_{COR} = 1.647$).

The amplitude of the desired sinusoidal compensation torque reference is $G_{TC} \times TrqRef_{Filt} \times k_{CORDIC}$ . $G_{TC}$ starts from zero and ramps up at a rate of 8 counts per electrical cycle till it reaches the value of parameter 'TrqCompGain'.

The torque compensation base angle $\theta_{base}$ increments by 120° (6-pole) or 180° (4-pole) every electrical cycle.

The torque compensation angle offset $\theta_{TCos}$ (parameter 'TrqCompAngOfst') specifies the angle difference between the peak load torque within a mechanical cycle and the peak of the synthesized sinusoidal compensation torque reference.

The status of synchronization with mechanical cycle $T_M$ is reflected in bit[1] of variable 'TrqCompStatus'.

As shown in Figure 46, the synthesized compensation torque reference 'TrqCompOutput' is summed up with 'SpeedPIOutput' to form total torque reference (variable 'TrqRef'), which is used in the following IPM (Interior Permanent Magnet) control block to generate current references for d and q axis current loops.

If it is needed to restart the synchronization with the mechanical cycle $T_M$, the torque compensation function shall be disabled and enabled again by toggling bit [1] of the parameter 'SysConfig'.

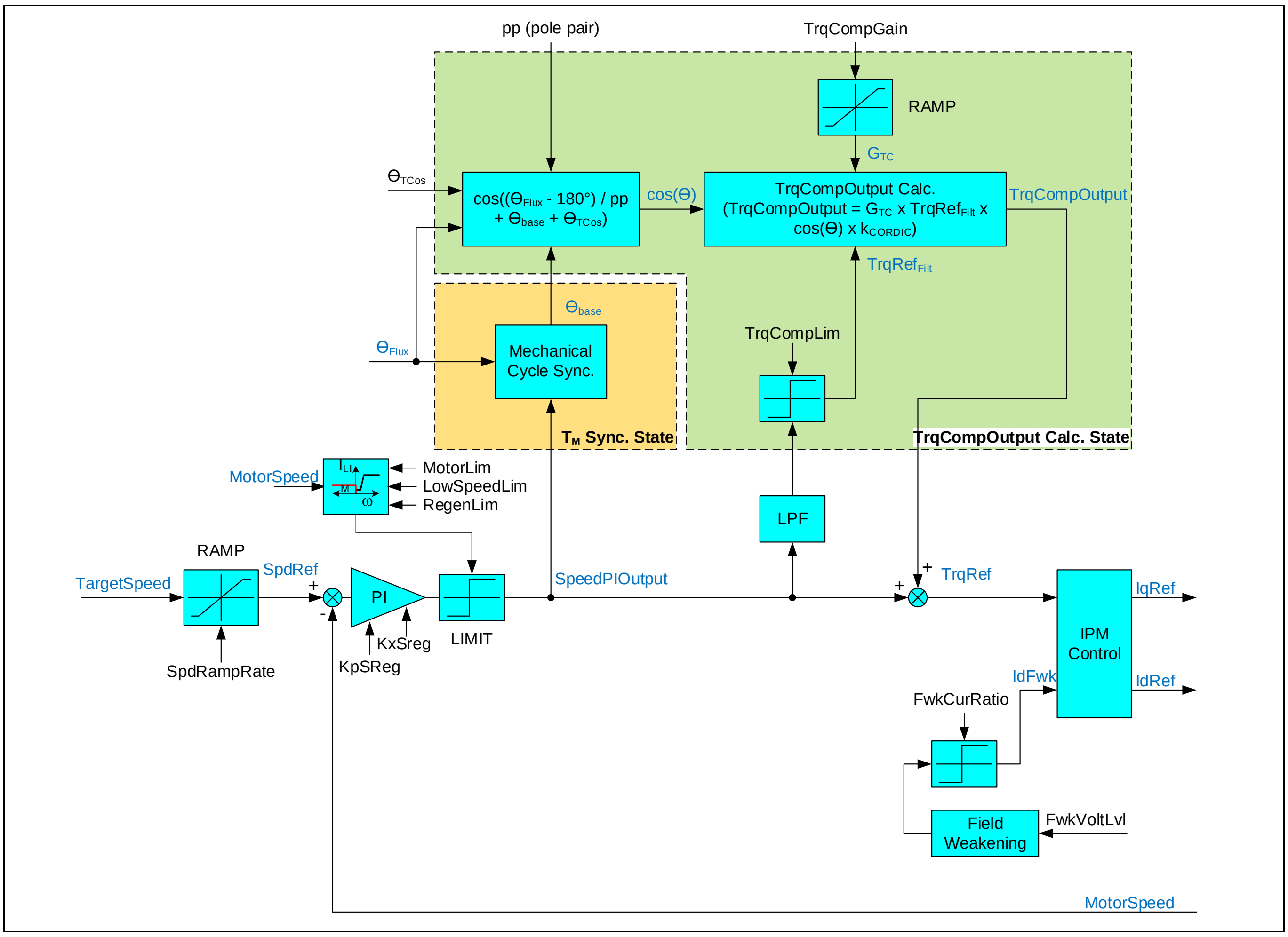


**Figure 46 Torque Compensation Top-Level Algorithm Diagram**

The following steps are recommended to tune the torque compensation related parameters.

a) Set initial parameter values: $G_{TC} = 0.5$ (TrqCompGain = 128) and 50% compensation torque limit (TrqCompLim = 2048).

b) Set the speed rising threshold (TrqCompOffSpeed) above which torque compensation function shall be inactive. Set the speed falling threshold (TrqCompOnSpeed) below which torque compensation function shall be active. These two parameters shall have about 2% hysteresis to avoid oscillation.

c) Set bit [1] of the parameter 'SysConfig' to enable torque conpensation function.

d) Use tracing function in Solution Designer to plot variable 'SpeedError'.

e) Adjust 'TrqCompAngOfst' value to a value with which the amplitude of 'SpeedError' as well as compressor vibration is minimized.

f) Increase 'TrqCompGain' value to further reduce the compressor vibration if needed.

## 2.1.16 Protection

### 2.1.16.1 Flux PLL Out-of-Control Protection

When the Flux PLL is locked to the correct rotor angle, Pll_M, which represent the flux of the permanent magnet of the motor, should be a DC value normalized at 2048 counts. Instead, if the PLL is not locked to correct rotor angle, Pll_M becomes either unstable or its value is far off from 2048 counts. Flux PLL out-of-control protection is the mechanism designed to detect this fault condition.

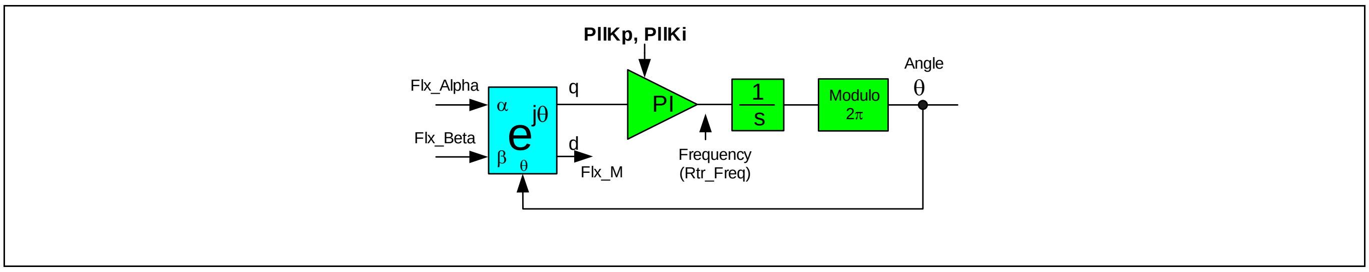


**Figure 47** Simplified block diagram of a Flux PLL

The MCE keeps monitoring Pll_M, within certain time slot (configured by 'FluxFaultTime' parameter), if Pll_M value is below 512 or above 8192, and if this happens in 8 continuous time slots (each slot time is equal to FluxFaultTime/8), flux PLL is considered "out-of-control". See Figure 48 for details.

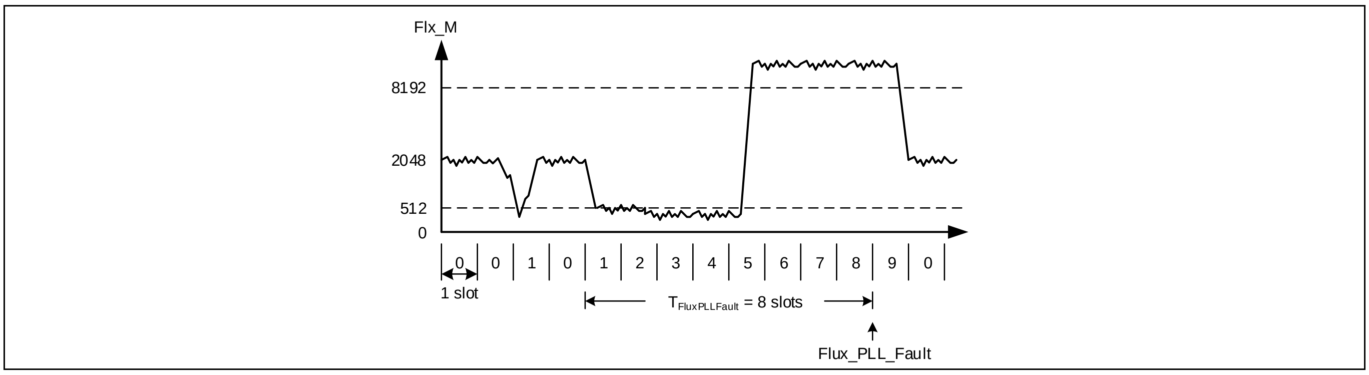


**Figure 48** **Flux PLL Out-of-Control Protection**

If the Flux PLL out-of-control fault is confirmed, then it will be reported by setting the bit 4 in FaultFlags motor variable, and the motor speed loop gets reset. If the bit 4 in 'FaultEnable' motor dynamic parameter is set, then this fault will be reflected in 'SwFaults' motor variable and the motor state machine will shift to FAULT state causing the motor to stop running. If this bit is not set, then the corresponding bit in 'SwFaults' variable will be masked by 'FaultEnable' parameter, so that this fault will not be reflected in 'SwFaults' variable, and the motor state machine will not shift to FAULT state. This protection is also able to detect phase loss condition.

The PLL out-of-control fault response time can be configured by setting motor parameter 'FluxFaultTime'. The valid range of its value is from 0 to 65535. The value of 1 corresponds to 0.016 seconds. The default value is set to 500, which corresponds to a response time of 8 seconds.

### 2.1.16.2 Rotor Lock Protection

As shown in the following Figure 49, rotor lock fault is detected if the speed PI regulator output (variable 'TrqRef') is being saturated for a defined amount of time $T_{Rotor\_Lock}$. The rock lock detection time $T_{Rotor\_Lock}$ can be configured by using parameter 'RotorLocktime' following this equation $T_{Rotor\_Lock} = RotorLockTime \times 16ms$ . Rotor lock protection is active when the motor speed ranges from min motor speed to 25% of maximum speed. Rotor lock protection becomes inactive when the motor speed goes beyond 25% of maximum speed to avoid erroneous fault reporting.

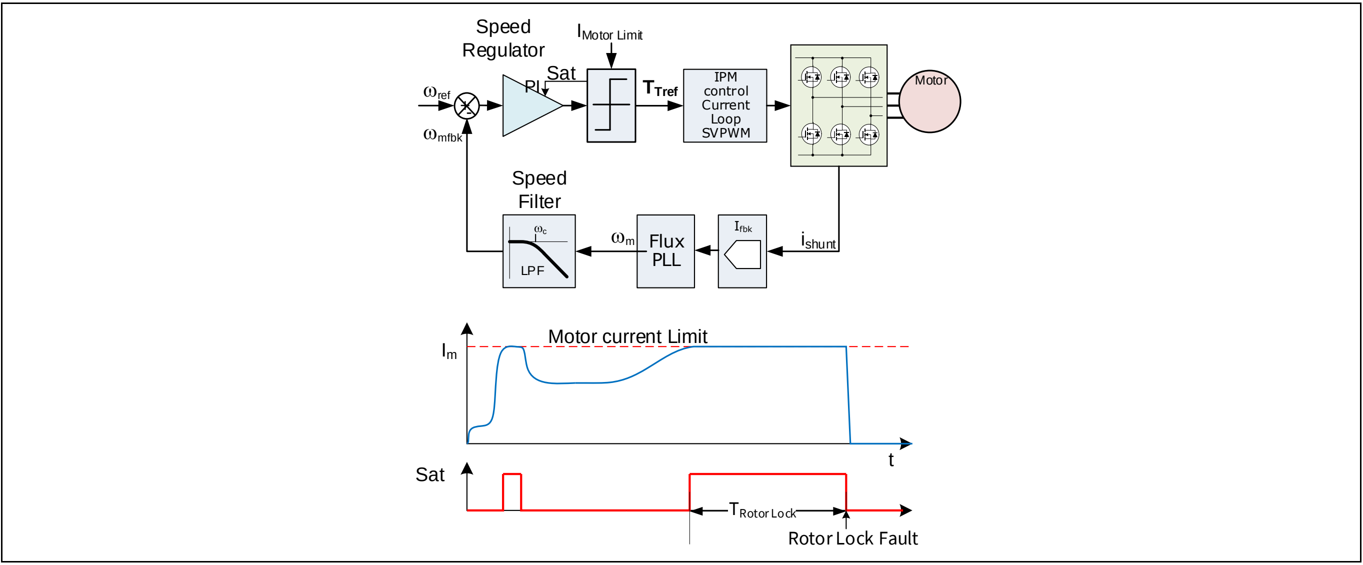


**Figure 49 Rotor Rock Protection Mechanism Diagram**

If the rotor lock fault is confirmed, then it will be reported by setting the bit 7 in FaultFlags motor variable. If the bit 7 in FaultEnable motor dynamic parameter is set, then this fault will be reflected in SwFaults motor variable, and the motor state machine will shift to FAULT state causing the motor to stop running. If this bit is not set, then the corresponding bit in SwFaults variable will be masked by FaultEnable parameter, so that this fault will not be reflected in SwFaults variable, and the motor state machine will not shift to FAULT state and the motor will keep running.

Please note if rotor lock detect time $T_{Rotor\_Lock}$ is set too short, it might trigger the fault during acceleration or momentary high load conditions.

Rotor lock detection is not 100% guaranteed to report the fault especially when the motor is running at low speed. The reason is, in rotor lock condition, the PLL might be locked at a false speed which may not cause speed PI output to be saturated.

#### 2.1.16.3 Motor Over Current Protection (OCP)

Motor gatekill fault is set during over current condition. This over current condition can be detected by the following two input sources.

1. Direct GK pin: gatekill fault is set if input is LOW
2. Internal comparators

It is possible to select either both or any one of the two sources for over current detection logic. Over current detection source can be selected by Solution Designer. Bit 0 in FaultFlag will be set in case of over current condition detected via any of the two sources. In case over current condition is detected via direct gate kill pin, bit 5 of FaultFlags will also set apart from bit 0 of FaultFlags.

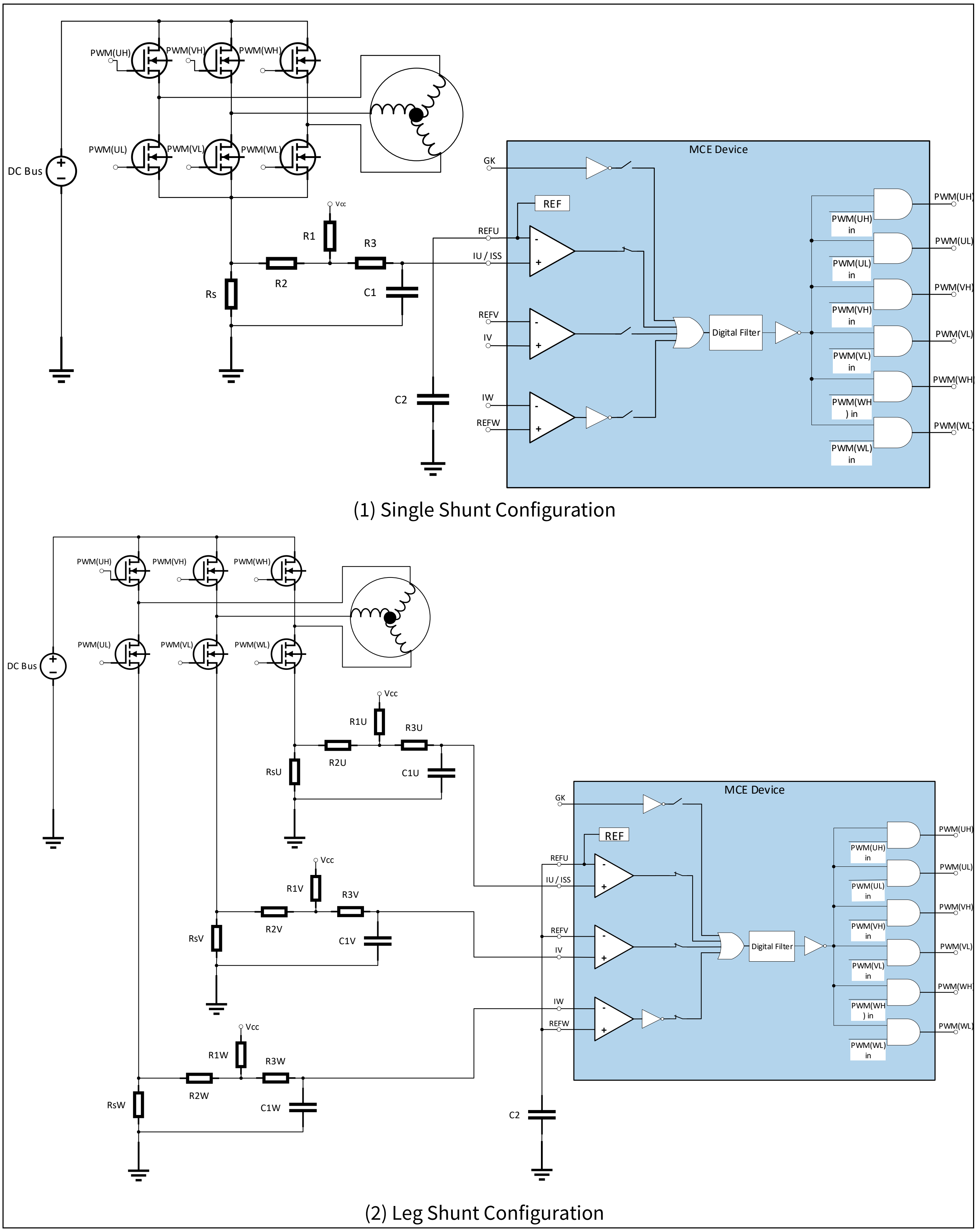


**Figure 50 Typical Motor OCP Implementation Using Internal Comparators**

User can select using either the dedicated GK pin or the internal comparators to realize the over-current protection function. In the case of using the GK pin, it is configured to be active LOW. In the case of using the internal comparators, the exact tripping voltage level can be specified by setting the 'CompRef' motor parameter. The current tripping level for the internal comparator can be configured using Solution Designer, the 'CompRef' parameter holds the current trip level value. As shown in Figure 50 (1), for single shunt current measurement configuration, only one internal comparator is used. For leg shunt current measurement configuration, three internal comparators are used to detect over current condition as shown in Figure 50 (2).

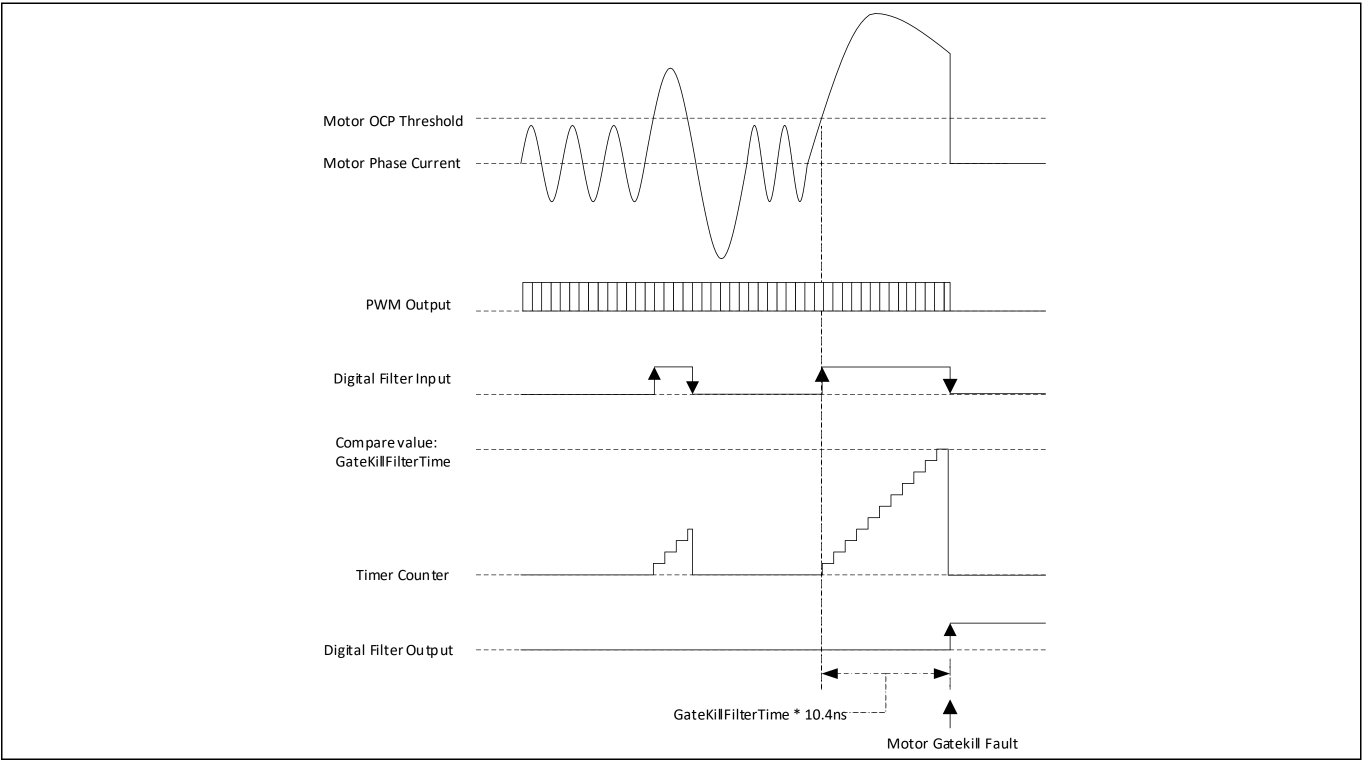


**Figure 51 Digital Filter Timing Diagram for Motor Gatekill Fault**

An internal configurable digital filter is used to de-bounce the input signal to prevent high frequency noise from mis-triggering a gate kill fault. "GatekillfilterTime" parameter holds the gate kill filter time value in clock cycles. Input signal needs to remain stable for the duration of the specified gate kill filter time to trigger the fault condition.

Gatekill filter timer is configured to be level triggered by the external GK pin or the internal comparator output. As shown in Figure 51, if the phase current goes beyond the specified OCP threshold, a timer in the digital filter starts counting up. If the digital filter input goes to logic LOW (external GK pin goes logic HIGH or the internal comparator output voltage level changes to logic LOW), then the timer gets reset. If the over-current condition is persistent when the timer counts up to 'GateKillFilterTime' value, then the digital filter output immediately goes to logic HIGH which forces entering Trap State upon which the PWM outputs all go to the programmed passive levels. The motor gatekill fault can only be cleared by writing 1 to the 'FaultClear' motor variable. This fault cannot be masked, so that it will be reflected in SwFaults motor variable, and the motor state machine will shift to FAULT state, causing the motor to stop running.

GateKillFilterTime is a type of static motor parameter that specifies the gatekill response time for over-current fault detection. The valid range of its value is from 4 to 960 in clock cycles. The value of 1 corresponds to 1/96MHz = 10.4167ns. The default value is 96, which is 1µs.

### 2.1.16.4 Over Temperature Protection

As shown in the following Figure 52, MCE provides an over-temperature protection (OTP) function with the help of an external NTC thermistor. Typically, the NTC thermistor and a pull-up resistor form a voltage divider. The MCE senses the output of the voltage divider and compares with a configurable OTP shutdown threshold $V_{Shutdown}$ that corresponds to the desired temperature $T_{Shutdown}$ where the system shall be shut down. If the output of the thermistor voltage divider is below $V_{Shutdown}$, then an OTP fault would be reported. The OTP shutdown threshold $V_{Shutdown}$can be configured using the parameter 'Tshutdown'.

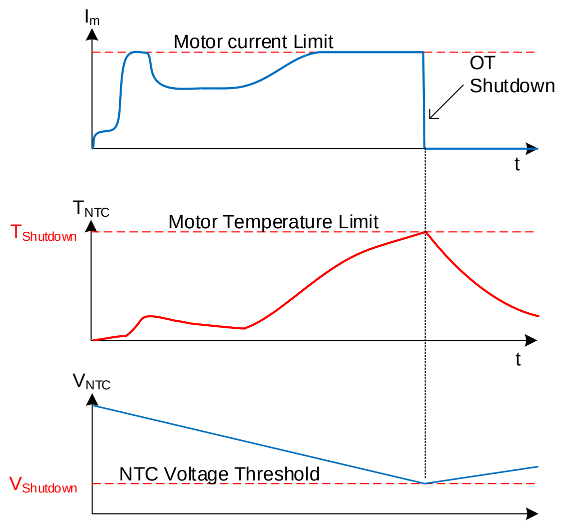


**Figure 52 Over-Temperature Protection Mechanism Diagram**

The action corresponding to the occurrence of over-temperature fault can be configured by use of the bit 6 in FaultEnable dynamic motor parameter. If this bit is set, then the motor state machine will go to FAULT state and the motor will stop running. If this bit is not set, then the motor state machine will not go to FAULT state and the motor will keep running.

### 2.1.16.5 DC Bus Over / Under Voltage Protection

Over/ under voltage fault is detected when DC bus voltage goes above or below the relevant protection voltage threshold values.

DC bus voltage is being sampled every motor PWM cycle. The sampled DC bus voltage goes through a Low-Pass Filter to attenuate high-frequency noise, which can be read from the variable 'VdcFilt'. The time constant of the LPF depends on the motor control PWM frequency, and it can be calculated using the following equation: $T_{decay} = \frac{Fast\_Control\_Rate}{F_{PWM} \cdot Ln(\frac{2^{16}}{2^{16}-2^{11}})}$. For example, if the motor control PWM frequency is 15 kHz, then the DC bus voltage sampling rate is 15 kHz. In that case, the time constant $T_{decay}$ is about 2.1ms, and the cut-off frequency is about 76Hz.

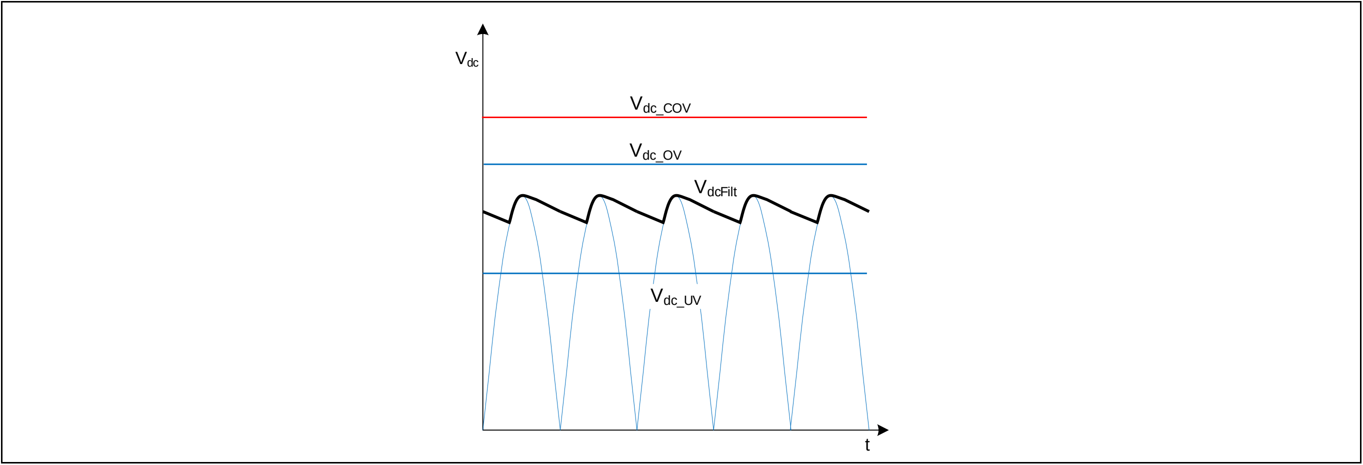


**Figure 53 DC Bus Over / Under Voltage Protection Threshold Diagram**

As shown in Figure 53, if the 'VdcFilt' value is greater than $V_{dc\_OV}$ (configured by the variable 'DcBusOvLevel'), then a corresponding bit 2 in FaultFlags motor variable is set. If the bit 2 in FaultEnable motor dynamic parameter is set, then this fault will be reflected in SwFaults motor variable, and the motor state machine will shift to FAULT state causing the motor to stop running. If this bit is not set, then the corresponding bit in SwFaults variable will be masked by FaultEnable parameter, so that this fault will not be reflected in SwFaults variable, and the motor state machine will not shift to FAULT state and the motor will keep running.

If the 'VdcFilt' value is lower than $V_{dc\_UV}$ (configured by the variable 'DcBusLvLevel'), then a corresponding bit 3 in FaultFlags motor variable is set. If the bit 3 in FaultEnable motor dynamic parameter is set, then this fault will be reflected in SwFaults motor variable, and the motor state machine will shift to FAULT state causing the motor to stop running. If this bit is not set, then the corresponding bit in SwFaults variable will be masked by FaultEnable parameter, so that this fault will not be reflected in SwFaults variable, and the motor state machine will not shift to FAULT state and the motor will keep running.

If the 'VdcFilt' value is above $V_{dc\_COV}$ (configured by the variable 'CriticalVdcOvLevel'), motor will be stopped immediately and zero vector [000] is applied until the fault is cleared, during which time 'critical over voltage' fault would be reported. This 'critical over voltage' fault cannot be disabled.

#### 2.1.16.6 Phase Loss Protection

The MCE is capable of detecting motor phase loss fault. If one of the motor phases is disconnected, or the motor windings are shorted together, the parking currents will not have the correct value. If any of the phase current value is less than $I_{phase\_loss}$ at the end of PARKING state, then phase loss fault is confirmed.

The $I_{phase\_loss}$ can be configured by using the parameter 'PhaseLossLevel'. The default value of 'PhaseLossLevel' is automatically calculated by Solution Designer following this equation:

$$PhaseLossLevel = 25\% \cdot \frac{LowSpeedLim}{4096} \cdot I_{rated\_rms} \cdot \sqrt{2} \cdot R_s \cdot G_{ext} \cdot G_{int} \cdot \frac{4096}{V_{ref\_ADC}}.$$

When phase loss fault is confirmed, if bit[8] of the parameter 'FaultEnable' is set, then this fault will be reflected in the variable 'SwFaults', and the motor state machine will shift to FAULT state causing the motor to stop running. If this bit is not set, then the corresponding bit in SwFaults variable will be masked by 'FaultEnable' parameter, so that this fault will not be reflected in 'SwFaults' variable, and the motor state machine will not shift to FAULT state and the motor will keep running.

#### 2.1.16.7 Current Offset Calibration Protection

This protection is checked in current OFFSET calibration state after offset measurement is completed. When this protection happens, system enters into fault state.

If any of measured current input offset values are not within a specific range, currentoffset fault will be triggered, then it will be reported by setting the bit 9 in FaultFlags variable. This flag gets clear when FaultClear is requested, while motor is in FAULT state.

If the bit 9 in FaultEnable motor dynamic parameter is set, then this fault will be reflected in SwFaults motor variable, and the motor state machine will shift to FAULT state. If this bit is not set, then the corresponding bit in SwFaults variable will be masked by FaultEnable parameter, so that this fault will not be reflected in SwFaults variable, and the motor state machine will not shift to FAULT state and move to STOP state.

CurrentOffsetMax and CurrentOffsetMin parameter defne the maximum and minimum level of current offset value protection level. The level is defined in ADC counts, 4095 represent Vadc reference value.

## 2.2 Power Factor Correction

Power Factor Correction (PFC) is a technique used to match the input current waveform to the input voltage, as required by government regulation in certain applications. The power factor, which varies from 0 to 1, is the ratio between the real power and apparent power in a load. A high power factor can reduce transmission losses and improve voltage regulation. The MCE supports full digital control and protection of a Continuous Conduction Mode (CCM) boost PFC using average current control scheme.

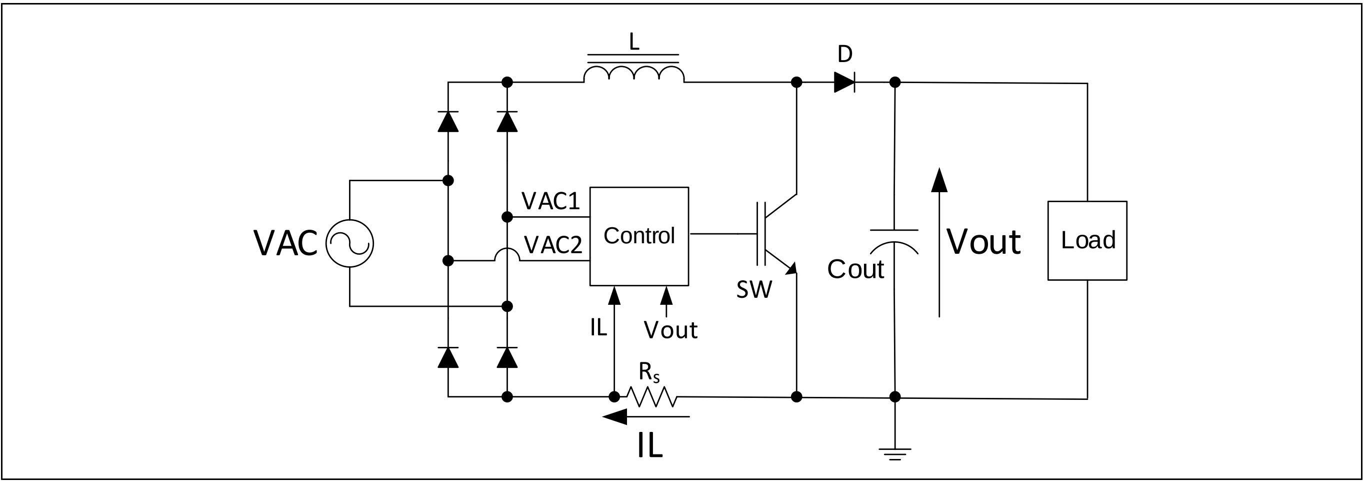


**Figure 54 Boost PFC topology.**

Starting from the left-hand side of Figure 54, an AC-voltage (VAC) is rectified by a full bridge rectifier. The boost converter itself consists of an inductor (L), a diode (D), and a switch (SW). The output is filtered by a DC-capacitor (Cout) that smooths the output voltage (Vout). To measure the inductor current (IL), a resistive shunt ($R_s$) is placed typically in the return path of the input current. There are several other options for measuring current but, by putting the sensor in the return path, both the sensing circuit and the gate of the switch can be referenced to the same potential as the output voltage. The boost converter requires a controller to regulate the inductor current as well and the output voltage. As feedback the controller relies on measurements of the inductor current, the output voltage and the AC voltage.

### 2.2.1 PFC Algorithm

Closed-loop control ensures that the output voltage is kept at its desired value and that the AC current is sinusoidal and in phase with the AC voltage. The PFC control algorithm of the MCE is a multiplier-based average current control scheme, which means there are two control loops: an inner current loop and an outer voltage loop. In addition, there are feedforward terms which enhances dynamic response. The output of the voltage controller is multiplied by the instantaneous rectified AC voltage value and then divided by the square of AC voltage Root-Mean-Square (RMS) value to produce a reference for the current controller which in turn generates the duty-cycle command. This PFC control scheme requires sensing of the inductor current, AC line voltage and DC-bus voltage. With Figure 55 as reference, each of the main element of the MCE control algorithm will be described.

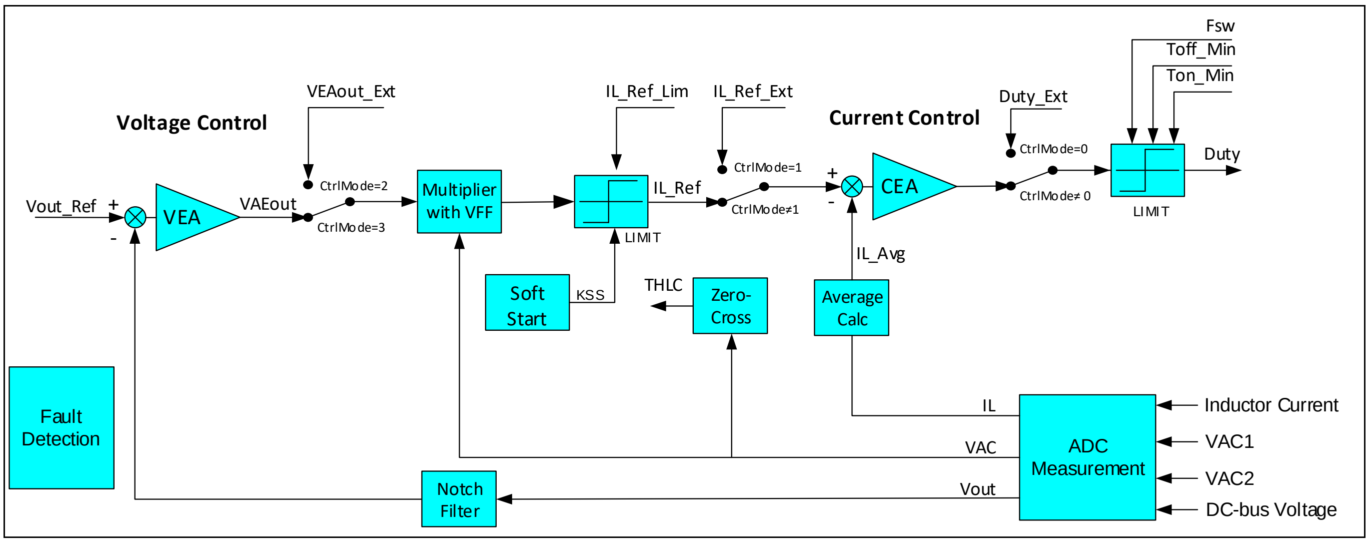


**Figure 55 Control system for the Boost PFC.**

#### 2.2.1.1 ADC Measurement

As feedback, the system in Figure 55 requires three measurements for close loop control:

1. The DC-bus voltage, Vout, to ensure that it is maintained at the reference level Vout_Ref.
2. The AC voltage, VAC, to provide a sinusoidal shaped reference for the input current.
3. The inductor current, IL, to ensure that it tracks the reference IL_Ref.

These three signals are also used for over- and under voltage protection and for over current protection.

Vout is measured across the DC-link and with reference to power ground. VAC is measured in front of the rectifier and therefore not referenced to power ground. However, by measuring both the phase voltage, VAC1, and neutral voltage, VAC2, (see Figure 54) the actual AC voltage, VAC, is reconstructed as:

$$VAC = VAC1 - VAC2$$

All three signals are measured at the update rate of the current control loop (base rate). Scheduling and sample rates will be discussed in more detail in section 2.2.3.

#### 2.2.1.2 Current Control

The inner control loop ensures that the inductor current, tracks the current reference, IL_Ref. The central element of the loop is a Current Error Amplifier (CEA) which calculates the duty-cycle command for the boost converter switch. As feedback, the loop relies on the inductor current averaged over a PWM switching period, IL_avg. The output of the CEA is fed to limiter that ensures minimum on/off times are observed.

The bandwidth of the current controller is determined by tuning but typically falls in the range of 3-9 kHz. Solution Designer calculates parameters for the current controller for optimized performance.

#### 2.2.1.3 Average Current Calculation

Continuous Conduction Mode (CCM) is preferred for higher power boost PFC converters due to EMC and power component utilization. However, at low load the inductor current can become discontinuous during a portion of each half-line cycle and the boost converter enters what referred to as Discontinuous Conduction Mode (DCM). From a control point of view the two conductions modes are quite different and require separate handling. The algorithm in the MCE is optimized for CCM but it has additional features that enhances input current waveform during DCM (low load).

Figure 56 (A) shows the gate signal and corresponding inductor current during CCM. During the ON-time of the gate, $T_{ON}$, the inductor is charged by the input voltage and during the OFF-time, $T_{OFF}$, the charge is released to the DC-link. The PWM period, $T_{PWM}$, equals $T_{PWM}=T_{ON}+T_{OFF}$. At no point during the PWM period does the inductor current drop to 0A and a continuous current flows through the inductor.

Figure 56 (B) shows the gate signal and corresponding inductor current during DCM. As with CCM, the ON-time of the gate, $T_{ON}$, charges the inductor and during the OFF-time, $T_{OFF}$, the charge is released to the DC-link. The difference is that the inductor current drops all the way to 0 during the OFF-time, meaning there is period, $T_{DIS}$, where no current flows through the inductor. In other words, the current flow is discontinuous. $T_{DIS}$ depends on a number of factors, such as input voltage, output voltage, duty-cycle, inductor size and switching frequency.

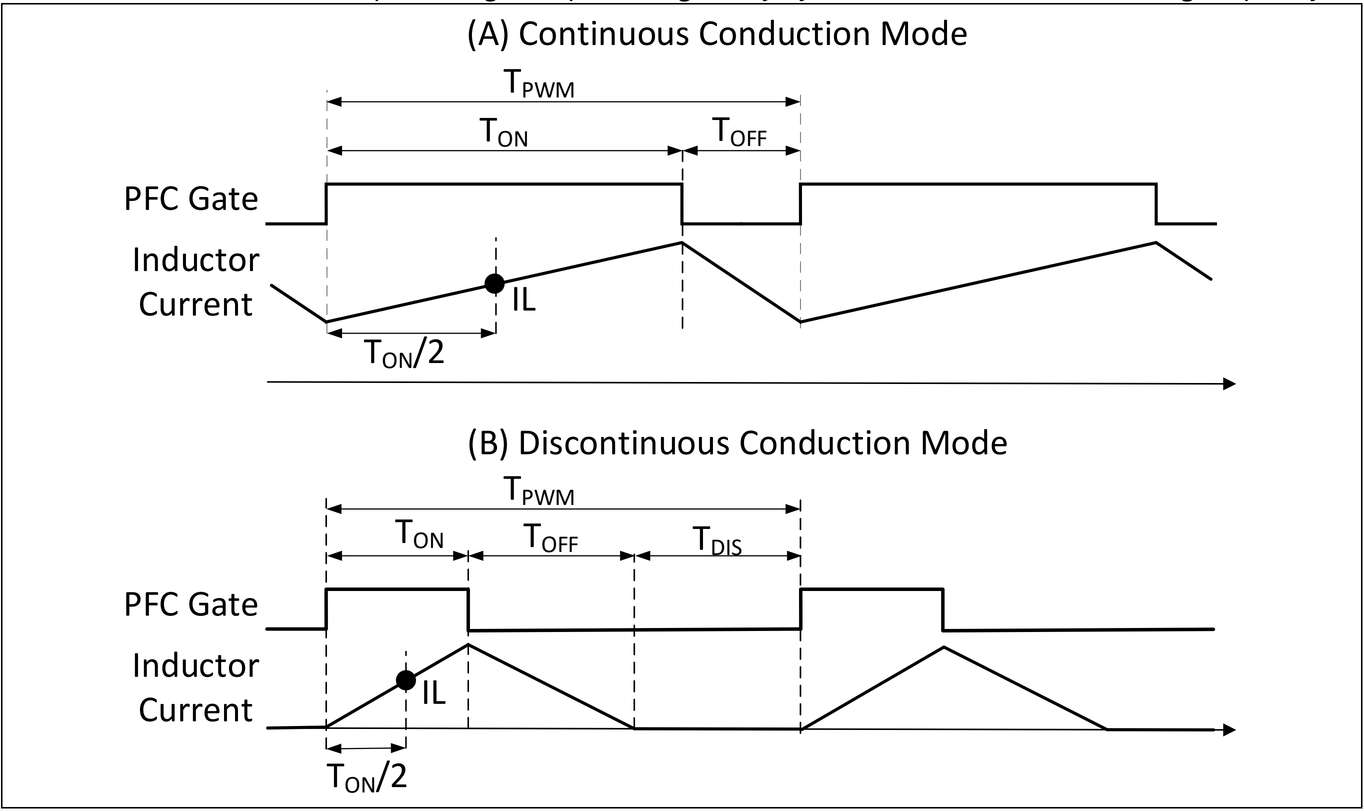


**Figure 56 (A) Continuous Conduction Mode. (B) Discontinuous Conduction Mode.**

The digital PFC control algorithm requires a measurement of average inductor current once per control period. With CCM, the average current is easily obtained by sampling at the center of the ON-time, i.e. at $T_{ON}/2$. In Figure 56 this measurement is shown as the dot labeled IL. With DCM, extraction of the average current is more complicated as the duration of the non-conducting interval, $T_{DIS}$, has to be considered. The average inductor current can be expressed as a function of $T_{ON}$, $T_{OFF}$ and $T_{DIS}$.

$$IL\_Avg = \frac{IL \cdot (T_{ON} + T_{OFF})}{T_{ON} + T_{OFF} + T_{DIS}}$$

Where IL is the inductor current measured at the center of the ON-time. With Continuous Conduction Mode, $T_{DIS}$=0 and IL_Avg = IL. With Discontinuous Conduction Mode $T_{DIS}$>0, and IL_Avg ≠ IL.

Using IL as feedback in DCM will result in phase current distortion and reduced control performance. To properly control the current regardless of conduction mode, the average must be used as feedback for the loop. Measuring the average current is difficult in a digital system but estimation through calculation is possible.

The PFC algorithm in the MCE is capable determining the average current regardless of conduction mode. An estimator takes IL as input and, given operating conditions, calculates $T_{DIS}$. Based on $T_{DIS}$ the average inductor current is estimated and then used a feedback for the current control loop.

It should be noted that the PFC algorithm in the MCE is optimized for CCM and should be used together with a converter designed for CCM. The purpose of the average estimator is to enhance performance during low load conditions where the converter enters DCM.

#### 2.2.1.4 Voltage Control

The Voltage Error Amplifier (VEA) calculates the input current amplitude required to maintain the DC-bus voltage at the reference, Vout_Ref, under varying load and input voltage conditions. Feedback for the loop is a notch filtered version of the actual DC-bus voltage. Output from the compensator, VEAout, expresses the desired magnitude of the input current and is passed on to the multiplier where it is shaped according to the input voltage waveform, see 2.2.1.5.

The outer loop is much slower than the inner loop with a typically the bandwidth of a few 10s-of-Hertz. Update rate of the loop is configurable but the loop runs at a sub-rate (Primary Rate) of the inner current loop (Base Rate)

#### 2.2.1.5 Multiplier with Voltage Feed-Forward

The Multiplier has two purposes. First, it has to shape the current reference so it resembles the waveform of the input voltage. Second, it has to ensure constant voltage control loop gain during all operating conditions, commonly referred to as voltage feed-forward or VFF. The multiplier used in the MCE algorithm is shown in Figure 57.

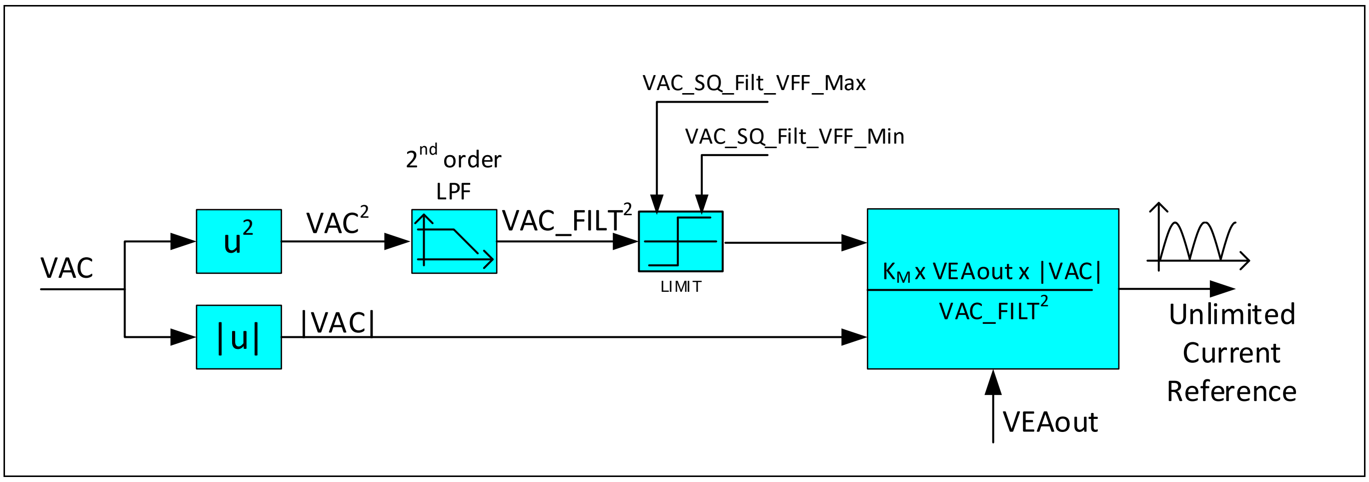


Figure 57 Reference Multiplier with Voltage Feed-Forward.

The inputs to the multiplier are the absolute value of instantaneous input voltage, |VAC|, and a 2nd ordered low-pass filtered version of the squared input voltage, $VAC\_FILT^2$. |VAC| shapes the current reference to be proportional to the input voltage after the rectifier, hence ensuring unity power factor. $VAC\_FILT^2$ is representing the squared RMS of the input voltage.

The overall gain of the voltage loop is proportional to the square of the input RMS voltage. That means the loop has a higher gain at high AC input voltages than it does at low AC input voltages. Since a boost PFC typically operates over a wide AC input voltage range, it is impossible to design one controller that operates well under all conditions. If the controller is optimized for high voltage operation it becomes sluggish at low voltage. Even worse, if the controller is optimized for operation at low voltage, it could become unstable at high voltage.

The MCE algorithm ensures constant voltage loop gain by normalizing the output of the voltage controller, VEAout, with the inverse of the RMS input voltage squared, see Figure 57. This make the control loop independent of the input voltage throughout the universal input range. The factor $K_M$ is calculated by Solution Designer to ensure the current reference can reach the peak current needed to deliver the specified max power at minimum AC input voltage when the VEAout reaches maximum limit.

The multiplier uses a lowpass filtered version of the squared AC line voltage to represent the squared RMS value of the AC line voltage. The filter is designed to attenuate the component at twice the AC line frequency. However, with a finite attenuation some AC content will remain at twice the AC line frequency and this ripple couples through multiplier and ends up modulating onto the current reference as a 2nd order harmonic distortion. The current controller can easily track a 2nd order component so the distortion ends up in the actual input current as a 3rd order harmonic. Solution Designer helps the user designing the filter based on acceptable 3rd harmonic input current requirement.

$VAC\_FILT^2$ gets limited if it falls outside the configurable parameters VAC_SQ_Filt_VFF_Min and VAC_SQ_Filt_VFF_Max. This effectively disables voltage feedforward when operating beyond these limits.

The output of the Multiplier represents the unlimited inductor current reference. The unlimted reference is passed on to a limiter that ensures the current reference never exceeds IL_Ref_Lim.

#### 2.2.1.6 Notch Filter

With both the input voltage and current being sinusoidal, the power drawn from the grid has a squared sinusoidal waveform pulsating at twice the grid frequency. For example, at 50Hz supply the power will pulsate at 100Hz. The job of the DC-link capacitor is to filter out this pulsating component so the load sees an output voltage close to an ideal DC. However, due to cost and physical size constraints it is not possible to fully eliminate the DC-bus voltage ripple. The result is that any one-phase boost PFC will have a DC-bus voltage ripple alternating at twice the grid frequency.

Typically, the DC-bus ripple does not have major negative effect on the load. However, voltage ripple couples through the outer voltage control loop and ends up modulating the current reference amplitude as a second order harmonic distortion. The current controller can easily track a second order component so the distortion ends up in the actual input current.

The two feedback loops of the PFC boost have somewhat conflicting objectives. A fast outer loop gives good performance in terms of disturbance rejection and stabilizes the output voltage under all operating conditions. However, a strongly tuned voltage loop will deteriorate the power factor by commanding an input current that ensures a fixed output voltage rather than the desired sinusoidal-shaped current. To limit the distortion of the input current reference, the traditional approach is to reduce the control loop gain at the second harmonic frequency. This approach attenuates the voltage ripple coupled through the loop but it is undesirable in terms of dynamic control performance.

The PFC algorithm in the MCE, solves the problem of second harmonic distortion caused by the voltage control loop by introducing a second order notch-filter in the feedback path of the voltage loop, see Figure 55. The filter is tuned to have high attenuation (notch) at twice the grid frequency, and therefore removes the voltage ripple from the feedback signal, while leaving all other frequencies unaltered. With the voltage ripple removed from the feedback, detuning of the voltage loop gain/bandwidth to avoid current distortion, is no longer needed.

Figure 58 shows the magnitude plot of the notch filter implemented in the MCE when tuned for a 100Hz center frequency (notch) which is suitable for a 50Hz input supply. The bandwidth and the attenuation of the notch filter are configurable. In this example, the filter was designed for a 20Hz width of notch (-3dB to -3dB) and the attenuation at the notch is designed for -100dB. The filter is updated at the primary rate, which in this example is 2500Hz. The notch filter is fully tuned and parametrized by Solution Designer.

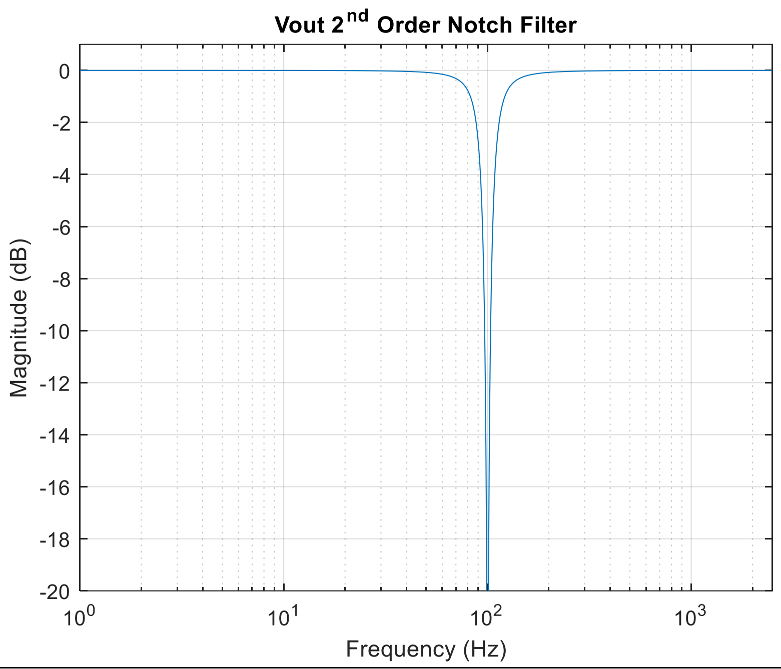


**Figure 58 2nd order Notch Filter tuned to remove 100Hz ripple.**

### 2.2.1.7 Zero-Cross Detection

The PFC algorithm relies on information about the line frequency and half line cycle period, THLC. Both of these values are determined by measuring the time between zero-crossings of the line voltage as illustrated in Figure 59. The top part of the figure shows the line voltage, VAC, along with an AC Polarity signal. The AC Polarity signal indicates whether VAC is in a positive- or negative half cycle of the voltage and it changes state at every zero-crossing. The period of the half line cycle, THLC, is time between zero-crossings of the line voltage.

In Figure 59, a positive-to-negative and a negative-to-positive going zero-crossing are highlighted by dashed eclipses and close-up views are shown in the bottom half of the figure. Stating with the positive-to-negative zero-cross detection (bottom left), the first step is to determine when VAC is less than the threshold ZCD_Step1_Thr but greater than 0V. If the voltage stays within this range for a deglitch time of ZCD_Step1_Deglitch_Time, the detection proceeds to the second step. If the voltage fails to stay within the thresholds throughout the deglitch window, the detection starts over from the beginning.

Second step of the detection is to validate the zero-crossing from first step. For a successful completion of the second step, VAC must stay below the threshold -ZCD_Step2_Thr for a deglitch window with a duration of ZCD_Step2_Deglitch_Time. When that happens, a new-zero crossing has been detected and the half line cycle time gets updated based on the elapsed time since last zero-crossing. If the voltage fails to stay below the threshold throughout the deglitch window, the detection goes back to the beginning of the first step.

Similarly, with the negative-to-positive zero-cross detection (bottom right), the first step is to determine when VAC is greater than the threshold -ZCD_Step1_Thr but less than 0V. If the voltage stays within this range for a deglitch time of ZCD_Step1_Deglitch_Time, the detection proceeds to the second step. If the voltage fails to stay within the thresholds throughout the deglitch window, the detection starts over from the beginning.

For a successful completion of the second step, VAC must stay above the threshold ZCD_Step2_Thr for a deglitch window with a duration of ZCD_Step2_Deglitch_Time. If the voltage fails to stay below the threshold throughout the deglitch window, the detection goes back to the beginning of the first step.

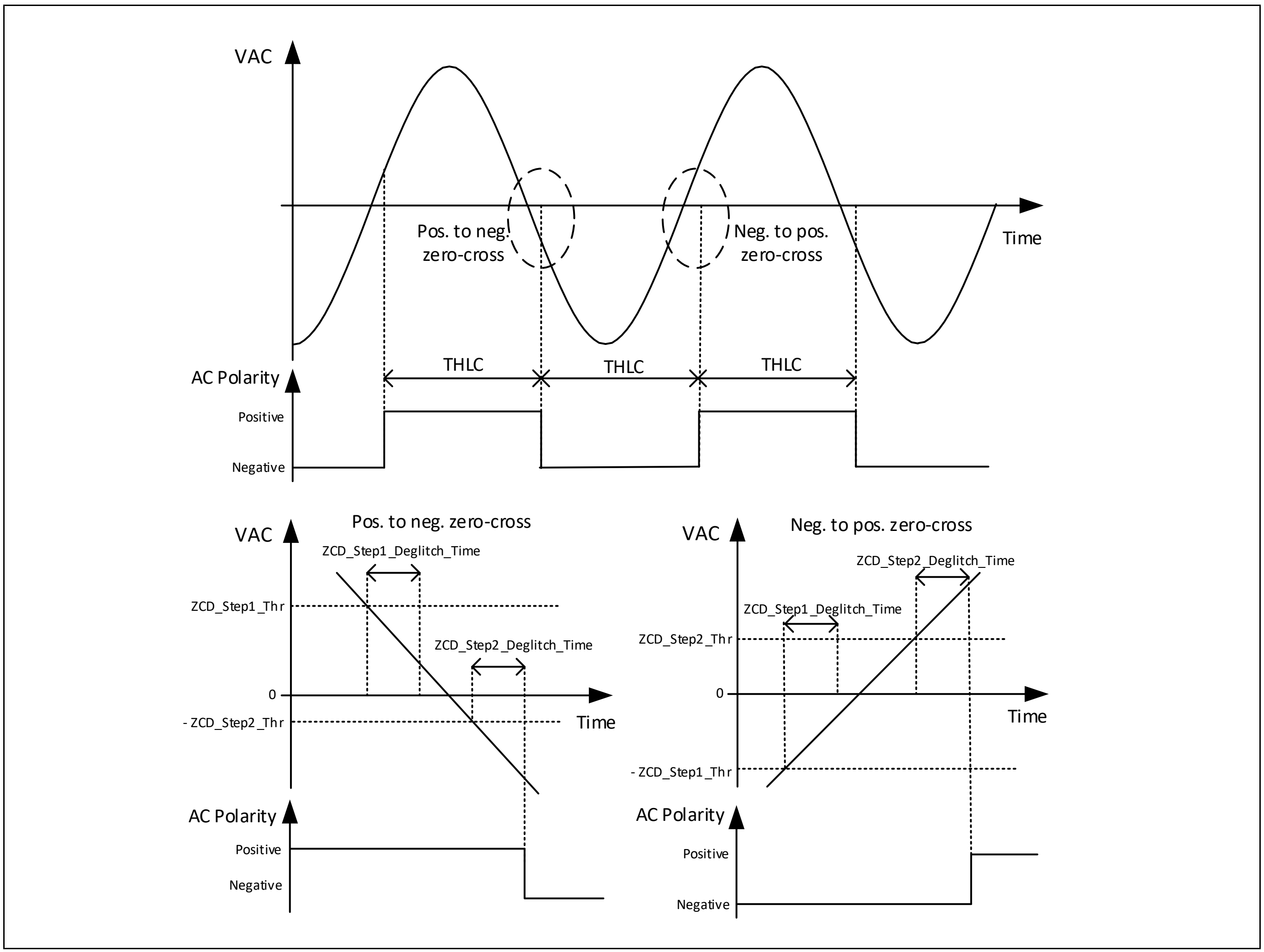


**Figure 59 Zero-cross detection of the line voltage.**

In addition to the described thresholds and deglitch windows, there is a timeout on step 2. After completion of step 1, the voltage must drop below -ZCD_Step2_Thr within a time of ZCD_Step2_Check_Time to avoid time-out. In case of a time-out, the detection starts over from the beginningZCD_Step2_Check_Time.

If the detection algorithm fails to find a valid zero-cross within the configurable time ZCDTimeout_Thr, the parameter ZCDTimeoutFlag is set to 1 to indicate the system is supplied by a DC source. If a zero-cross is detected within ZCDTimeout_Thr, the parameter ZCDTimeoutFlag is set to 0 to indicate the system is supplied by an AC source.

#### 2.2.1.8 Soft Start

At startup, there is typically a big difference between the actual DC-bus voltage and the requested voltage, Vout_Ref. That results in a large control error which can lead to DC-bus overshoot when starting up with no load or light load thanks to low bandwidth of the voltage control loop. To avoid this the MCE has a Soft Start feature that gently charges the DC-bus capacitor by gradually increasing a scaling factor, KSS, for the inductor current limit, IL_Ref_Lim. Soft start is complete when the current reference reaches 100%.

Soft Start ramps the DC-bus voltage up at every start of the PFC, including when PFC operation has been interrupted by a fault. The actually state of the soft start sequence can be read from bitfield SSStatus in parameter PFCStatus.

#### 2.2.1.9 Vout Ready Monitor

The MCE has Vout Ready monitor function that checks the DC-bus voltage against a configurable threshold. One possible use case for this check is during sequencing of PFC- and motor startup where the monitor function can be used to determine a safe time to start the motor.

The working principle of the Vout Ready monitor function is illustrated in Figure 60. When the DC-bus voltage exceeds Vout_Ready_Thr, and remains higher than the threshold during a deglitch window of length Vout_Ready_Deglitch_Time, bit 8 of PFCStatus is set. If the voltage drops below the threshold during the deglitch window, the status bit is not set. The status bit is cleared if voltage drops below Vout_Ready_Thr minus a hysteresis, Vout_Ready_Hyst, and stays below the threshold during a deglitch window with a length of Vout_Norm_Deglitch_Time.

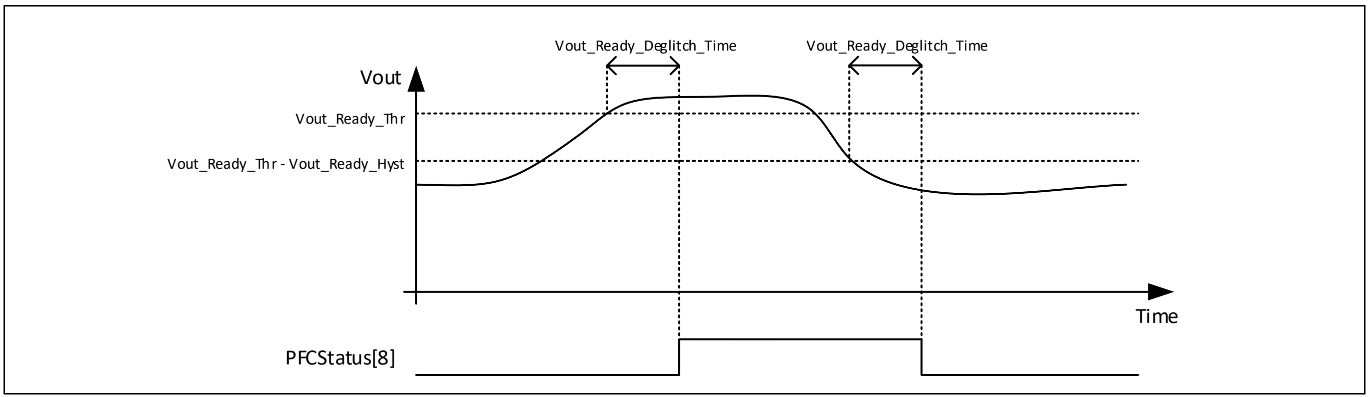


**Figure 60Vout Ready Check.**

#### 2.2.1.10 Control Modes

The PFC system offers manual control modes that let the user overwrite parts of the closed-loop control system. In Figure 55 the control modes are symbolized by switches whose positions are determined by setting of the bitfield CtrlMode of parameter SysConfig. The supported modes are:

**Table 11Control Modes**

| Control Mode | Function |
|---|---|
| 0 | Open loop current control and open loop voltage control. External input, Duty_Ext, sets the PFC duty cycle. |
| 1 | Closed loop current control and open loop voltage control. External input, IL_Ref_Ext, sets the PFC current reference. |
| 2 | Closed loop current control and open loop voltage control but with multiplier enabled. External input, VEAout_Ext, sets the PFC voltage error amplifier output. |
| 3 | Closed loop current control and closed loop voltage control with multiplier enabled. |

Normal PFC operation happens with ControlMode = 3. When ControlMode = 0-2 is selected, it is the user’s responsibility to set appropriate external references.

### 2.2.2 State Handling

The Motion Control Engine includes a built-in state machine which manages sequencing of the PFC. The state machine is updated at the Sequencer Rate (1kHz). Current state of sequencer is stored in PFC_SequencerState parameter. The states and transitions are listed in Table 12 and illustrated in Figure 61.

**Table 12State Description and Transition**

| Sequence State | PFC_SeqeuncerState | State Functionality | Condition for Next State |
|---|---|---|---|
| PFC_IDLE | 0 | After power-up, control enters this state. Setup and configuration of PFC. | Valid parameter set. |
| PFC_OFFSETCAL | 1 | Offset calculation for PFC current measurement channel. This state takes $2^{OffsetCalTotalTime}$ PWM cycles to complete. | If the offset falls within min/max levels, process to a RUN_CTRLMODEx state. If not, proceed to PFC_FAULT |
| PFC_FAULT | 5 | Fault state if current measurement offset check failed. | Once entered, the PFC cannot leave this state. Fault cannot be cleared. |
| RUN_CTRLMODE0 | 2 | Run mode with external duty-cycle reference.<br>Control is either waiting for an enable command, applying PWM or shut down by a fault. | Once entered, the PFC cannot leave this state. |
| RUN_CTRLMODE1 | 3 | Run mode with external current reference.<br>Control is either waiting for an enable command, applying PWM or shut down by a fault. | Once entered, the PFC cannot leave this state. |
| RUN_CTRLMODE2 | 4 | Run mode with external multiplier reference.<br>Control is either waiting for an enable command, applying PWM or shut down by a fault. | Once entered, the PFC cannot leave this state. |
| RUN_CTRLMODE3 | 6 | Normal PFC run mode with full close loop control. Control is either waiting for an enable command, applying PWM or shut-down by a fault. | Once entered, the PFC cannot leave this state. |
| PFC_STANDBY | 7 | The MCE lowers standby power consumption by reducing the CPU clock and by switching off some of the controller's peripherals. To enter STANDBY the PFC must be in RUN_CTRLMODE3 and PFC PWM disabled. In addition, a configured delay time must expire before entering STANDBY. | A motor- or PFC start command or a fault. |

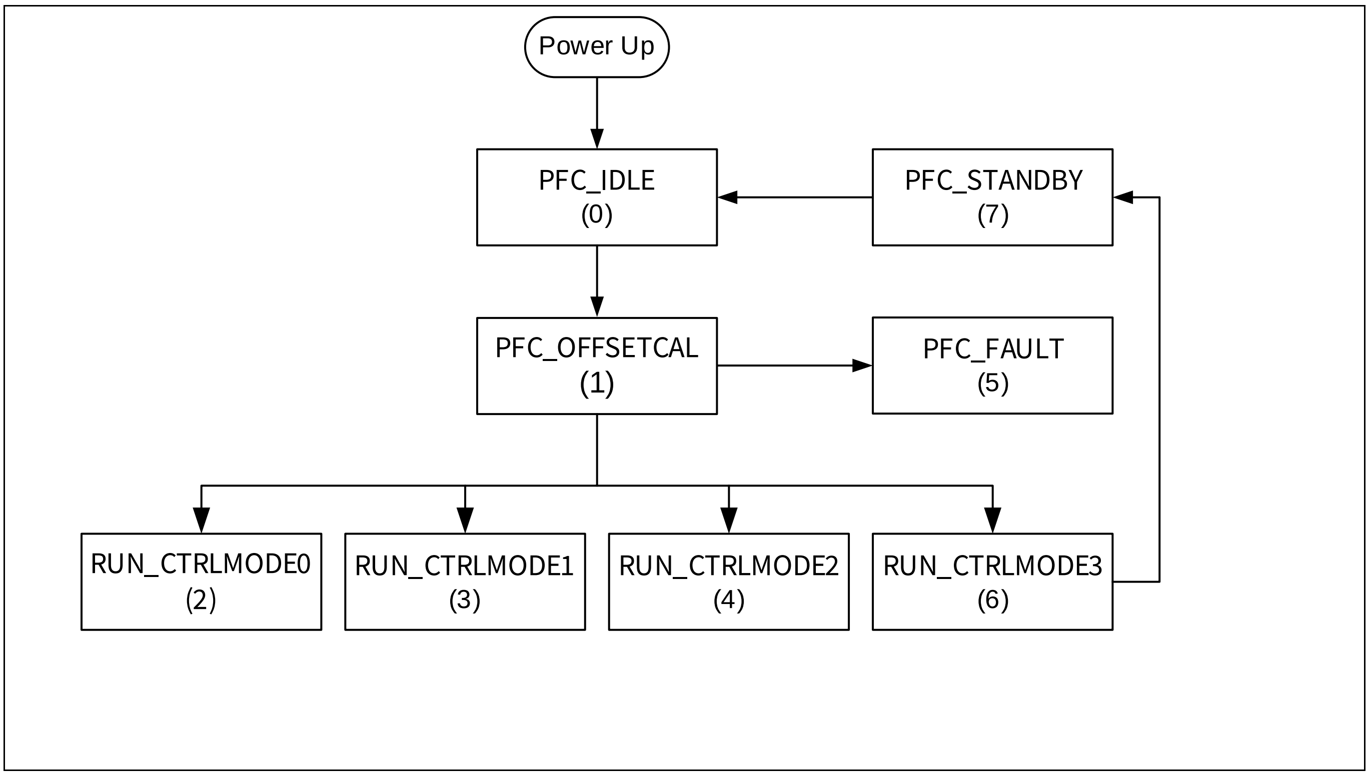


**Figure 61 State handling of the PFC.**

Once a RUNCTRLx (x = 0, 1, 2 or 3) state has been entered, the PFC cannot leave the state (the exception being standby which is discussed below). PFC operation can be enabled/disabled by setting of the Command parameter but the system stays in the RUN_CTRLx state regardless of the command. In case of a fault (excluding current offset fault), PFC gate operation is shut down until the fault clears but the system remains in the RUN_CTRLx state throughout the fault condition. In case of current offset fault, the PFC enters PFC_FAULT state and stay in that state until power cycles. The status of the PFC operation (enable/disable) can be read from bitfield SWStatus of parameter PFCStatus.

In state RUNCTRLMODE3 the system can transition to PFC_STANDBY if the PFC is disabled and a configured delay time has expired. In standby the MCE lowers power consumption by reducing the CPU clock and by switching off some of the controller's peripherals. PFC_STANDBY is terminated when a motor start command is received or a motor fault occurs. PFC_STNADBY is left through the PFC_IDLE state and followed by an offset calibration before normal PFC operation is resumed.

### 2.2.3 Scheduling and Timing

The time constants involved in the control of a Boost PFC varies greatly. For best control performance, and to minimize execution load, it is beneficial to split the algorithm into sub systems based on time constants and execute those subsystems at different rates. The PFC algorithm in the MCE is updated at 4 different rates as listed in the table below:

**Table 13 Execution Rates**

| Rate Name | Execution Events | Update Rate |
|---|---|---|
| PWM | Switching frequency of PFC gate | Configurable. Typical 20-100kHz |
| Base | Measurement System<br>Reference Multiplier<br>Current Controller | Configurable. Sub-rate of PWM Rate.<br>Possible ratios are 1:1 and 1:2<br>Typical 20-60kHz |

| Rate Name | Execution Events | Update Rate |
|---|---|---|
| | PWM duty-cycle update<br>VAC zero-cross detection | |
| Primary | Voltage Controller<br>Current Reference Limiter<br>Notch filter<br>Soft Start<br>Feed-Forward voltage calculation<br>Vout UV/OV update<br>PFC Status update | Configurable. Sub-rate of Base Rate.<br>Typical 1-10kHz |
| Sequencer | Preparation of Power calculation<br>Preparation of VAC RMS calculation<br>Preparation of IAC RMS calculation<br>VAC Drop-out update<br>VAC UV/OV update<br>AC Frequency validation<br>Vout Ready update<br>Overcurrent trip status update | Fixed at 1kHz |

The current control loop is executed at the Base Rate and it is the most time critical part of the system. As mentioned, the Base Rate is derived as a sub rate of the PWM rate. When the processor load allows, it is preferred to use a 1:1 ratio between the PWM- and Base Rate, meaning the PWM duty-cycle is updated every PWM cycle. In systems where a high PWM rate is required, say 100kHz, a 1:1 ratio is not allowed due to the execution load of the MCE. For these high PWM rate systems, the MCE supports a 1:2 ratio between the PWM- and Base Rates.

Base- and primary rate are set by parameters FastControlRate and PrimaryControlLoop.

#### 2.2.3.1 1:1 Base Rate

If the PWM rate allows the current controller to be updated every PWM cycle, a 1:1 ratio between the PWM- and Base rate is preferred. In this case $T_{BaseRate} = T_{PWM}$. A timing diagram of this operating mode is shown in Figure 62.

From the top down, the figure shows the PFC Gate signal which is defined by its on-time, $T_{ON}$, its off-time, $T_{OFF}$, and its PWM switching period, $T_{PWM}=T_{ON}+T_{OFF}$. During the on-time the Inductor Current, IL, increases and during the off-time it decreases as the stored energy is release to the DC-link. Assuming Continuous Conduction Mode, the Inductor Current assumes its average value at the center of the on-time. Under ideal conditions, the center of the on-time is the correct instant to sample the current but the system has delays, such as gate driver propagation delay and measurement channel delay. To compensate for the delays, the ADC Trigger is offset by $T_{SHDelay}$. In addition to the inductor current IL, the AC voltages, VAC1/VAC2, and the DC-bus voltage, Vout are also measured by the ADC. The inductor current is sampled during the gate on-time and the 3 voltages are sequentially sampled during the gate off-time. Each measurement consists of a sample-and-hold stage and a conversion state. The sample-and-hold stage takes $T_{SH}$=333ns to complete and the conversion takes $T_{conv}$=767ns to complete.

When the ADC is finished converting IL, the current loop is updated based on the newly acquired feedback. The result is a new duty-cycle which is applied at beginning of the next PWM cycle.

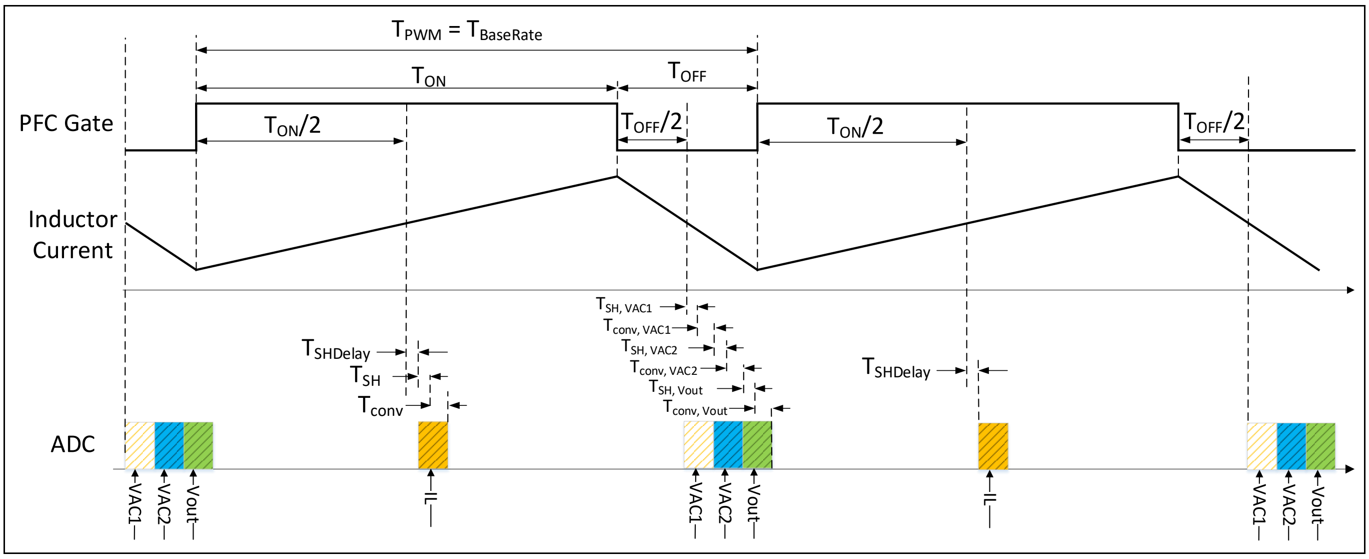


Figure 62 Timing diagram when the Base- to PWM rate is 1:1.

### 2.2.3.2 1:2 Base Rate

At higher PWM rates the execution load of the PFC algorithm does not leave enough room for the motor control algorithm. For these cases, the MCE offers a 1:2 base rate, meaning the current controller is only updated every second PWM cycle. In this case $T_{BaseRate} = 2 \times T_{PWM}$. The timing diagram in Figure 63, illustrates operation with a 1:2 Base Rate. Note how the Inductor Current sampling and execution of the control algorithm are skipped every second PWM cycle.

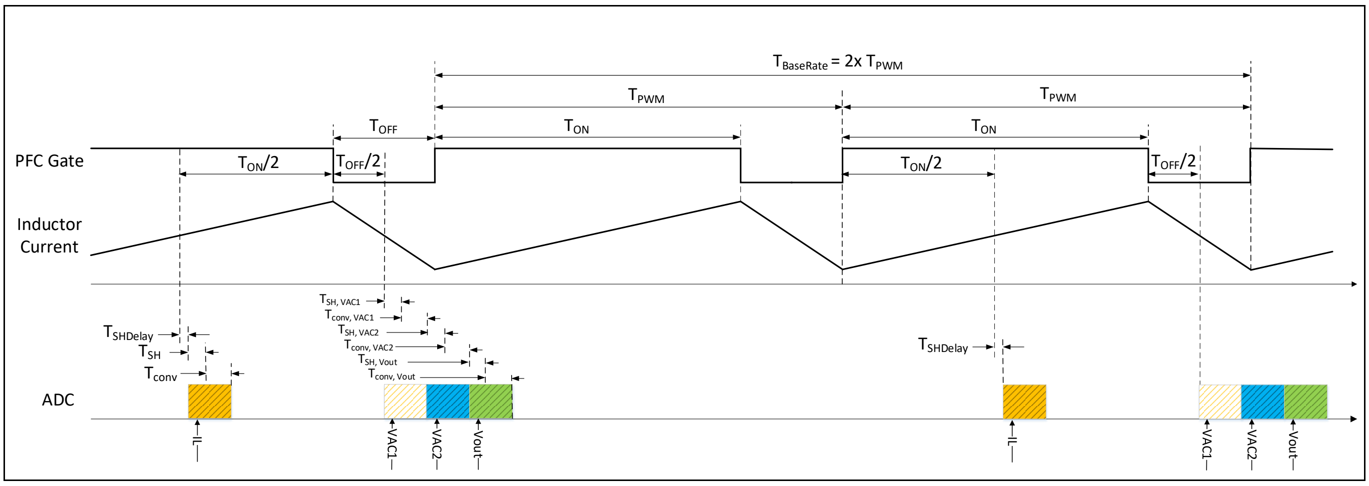


Figure 63 Timing diagram when the Base- to PWM rate is 1:2.

### 2.2.3.3 Co-existence of PFC and Motor Control Algorithm

Both the PFC and the Motor Control are real-time control system that must be executed in a time consistent manner. This requires special attention when both algorithms are running on the same single core device. The PFC algorithm typically executes at a higher rate than the motor control algorithm but the PFC execution time is only a fraction of motor execution time. On the MCE, execution of the PFC algorithm is given highest possible priority and it can preempt execution of the motor algorithm. This ensures both PFC- and motor algorithms can coexist on the same device. Neither phase nor rate of PFC PWM are synchronized to Motor PWM.

### 2.2.4 Protection

The MCE has a total of eight PFC protection functions as summarized in the table below. The functions have been designed to protect the PFC from operating under potentially damaging conditions while at the same time ensure maximum robustness of the PFC operation.

**Table 14Protection Functions**

| Protection Function | Description | Fault Actions |
|---|---|---|
| Over Current (OCP) | Fast, HW-based cycle-by-cycle overcurrent protection | PFC gate to inactive level.<br>Automatic recover at the beginning of the following PWM cycle when fault clears.<br>FaultFlags status update. |
| VAC Drop-out | Low instantaneous input voltage | FaultFlags status update. |
| VAC Overvoltage | High RMS input voltage protection | Limit PFC gate duty-cycle to 0.<br>Automatic recover when fault clears.<br>FaultFlags status update. |
| VAC Brown-out | Low RMS input voltage (undervoltage) protection | Limit PFC gate duty-cycle to 0.<br>Automatic recover when fault clears.<br>FaultFlags status update. |
| VAC Frequency | Out of range AC line frequency | Limit PFC gate duty-cycle to 0.<br>Automatic recover when fault clears.<br>FaultFlags status update. |
| Vout Overvoltage | DC-bus overvoltage protection | Limit PFC gate duty-cycle to 0.<br>Automatic recover when fault clears.<br>FaultFlags status update. |
| Vout Open-Loop | DC-bus voltage open-loop (undervoltage) protection | Limit PFC gate duty-cycle to 0.<br>Automatic recover when fault clears.<br>FaultFlags status update. |
| Current Measurement Offset | Out of range current measurement offset before start of PFC | Abort start-up of PFC.<br>Enter PFC_FAULT state and latch until power is cycled.<br>FaultFlags status update. |

Except for VAC Drop-out protection, the protection system automatically brings the system into a safe mode when a fault is detected. Most protection functions force the PFC duty-cycle to 0 in case of a fault and, when the fault clears, restores normal operating conditions. The exception to this handling approach is OCP- and Current Measurement Offset faults. In case of OCP fault, the PFC gate is switched to the inactive state for the remainder of the PWM cycle but by the start of the following PWM cycle the gate is automatically reenabled if the OCP condition is cleared. A Current Measurement Offset fault forces the system into an inactive state form which there is no recovery until power is cycled.

The status of the PFC operation can be read from bitfield SWStatus of parameter PFCStatus. If the Command parameter is set to Enable and there are no fault conditions, SWStatus will be set to Enable. If the Command parameter is set to Disable, or the duty-cycle has been forced to zero by a fault conditions, SWStatus will be set to Disable. Note that the faults capable of forcing the duty-cycle to zero are VAC Overvoltage-, VAC Brown-out-, VAC Frequency-, Vout Overvoltage- and Vout Open-Loop fault.

It should be noted that a PFC fault does not shut down operation of the motor. Likewise, a motor fault does not shut down operation of the PFC. If such an application specific fault synchronization is required, the application script must take care of the required handling. Fault status is accessible to the script through the parameter FaultFlags.

Though not recommended, a protection function can be disabled by configuring the thresholds outside the operating conditions of the system. OCP is always enabled.

#### 2.2.4.1 Over Current Protection

The MCE provides an over-current protection (OPC) function by comparing the instantaneous inductor current against a pre-configured OCP threshold and disables the PWM output when the current exceeds the OCP threshold. The OCP function is fully implemented by hardware and operates independently of the software. As shown in Figure 64 the over-current tripping mechanism makes use of an internal comparator. The tripping level is programmed using external resistors Rh and Rl which set the tripping level at the PFCREF pin. The voltage across the resistive shunt Rs, is scaled and offset by R1, R2 and R3 and then fed to the PFCITRIP pin. If the voltage at the PFCITRIP pin exceeds the voltage at the PFCREF pin, the comparator output goes high and sets the RS-flip-flop.

When the inductor current exceeds the specified OCP threshold, the internal comparator output goes logic high. As a result, the PWM output immediately goes to logic low, and stays low until the end of this PWM cycle, even if the inductor current drops below the PFC OCP threshold. At the beginning of the following PWM cycle, if the inductor current is below the PFC OCP threshold, then PWM output resumes. If the inductor current is still higher than the PFC OCP threshold, then the PWM output remains logic LOW. This type of OCP is commonly referred to as cycle-by-cycle protection.

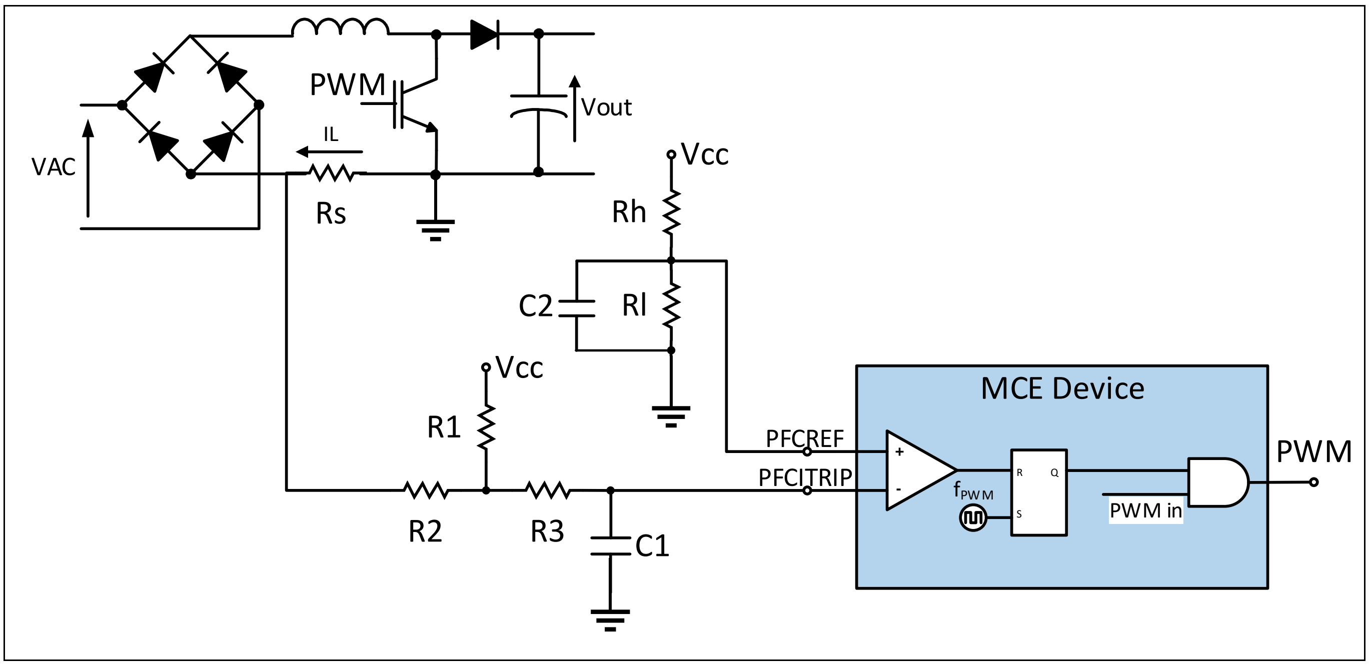


**Figure 64 Cycle-by-cycle OCP circuit.**

A timing diagram of cycle-by-cycle OCP is shown in Figure 65. PWM operates with a switching period of $T_{PWM}$. During the on-time the Inductor Current, IL, increases and during the off-time the inductor current decreases. When the inductor current exceeds the Trip Level, PWM is forced to the inactive state to prevent damage to the converter. At the beginning of the next PWM cycle, the fault is cleared and normal PWM resumes. In Figure 65, a second OCP fault is detected the following PWM cycle and the sequence repeats.

Note that PWM reenable is synchronized to the beginning of a new PWM cycle, guaranteeing the PWM switching frequency remains constant and as configured even during OCP conditions.

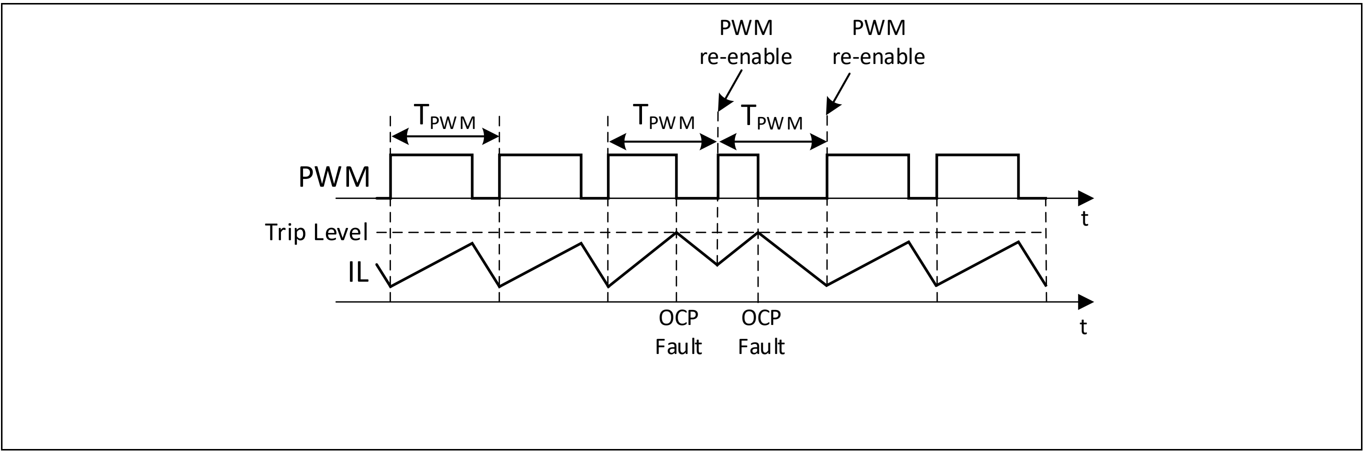


**Figure 65 Cycle-by-cycle OCP timing diagram.**

OCP fault notification is illustrated in Figure 66. When a fault occurs, an Internal Trap Flag is set. A configurable update time, OCP_Status_Update_Time, determines how often the trap flag gets read and copied to bit 0 of FaultFlags parameter. Upon latch of the trap flag, and if the OCP fault condition is no longer present, the flag is cleared. If the fault persists, then the trap flag remains set. Bit 0 of FaultFlags will be set to 1 for a duration of OCP_Status_Update_Time and then automatically cleared. If the application requires system level handling of the OCP fault, it is up to the application script to capture FaultFlags in a timely manner and take the appropriate action.

In cased of OCP fault, the PFC state machine remains in the 'Run State' and PWM will be chopped by the OCP comparator on a cycle-by-cycle basis until user stops the PFC by setting parameter Command to disable.

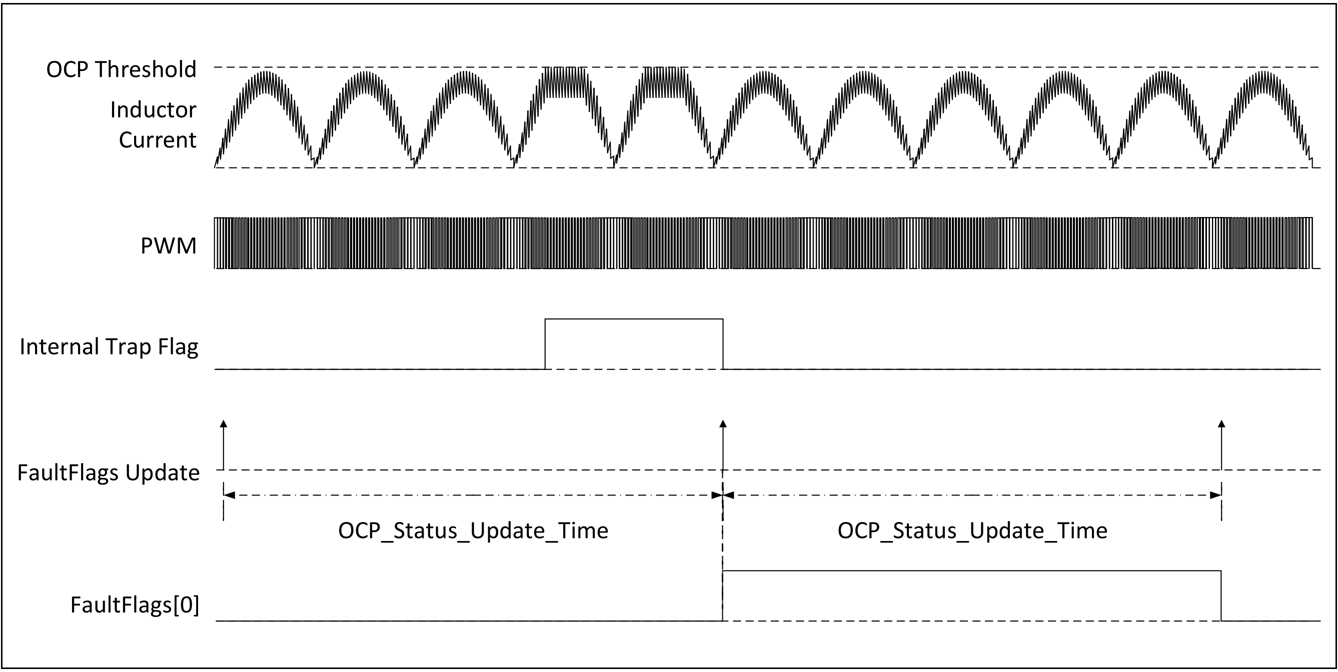


**Figure 66 OCP fault signaling.**

#### 2.2.4.1 VAC Drop-out Protection

VAC Drop-out fault becomes active when the instantaneous, absolute value of the AC input voltage drops below a configurable threshold. A hysteresis and a deglitch window are added to prevent rapid toggling between normal- and fault conditions. VAC Drop-out Fault does not force the PFC duty-cycle to 0 meaning PFC operation will continue in the event of a fault. It is up to the application script to take the appropriate action in case of VAC drop-out fault.

The AC input voltage is sampled every PFC base-rate cycle and drop-out detection relies on the absolute value of this measurement. The protection function is executed at Sequencer Period (1ms).

VAC Drop-out detection and clear is illustrated in Figure 67. If the instantaneous absolute value of the input voltage drops below VAC_DO_Thr, and remains lower than the threshold during a deglitch window of length VAC_DO_Deglitch_Time, bit 6 of FaultFlags is set. If the voltage exceeds the threshold during the deglitch window, the fault is not set. To clear the drop-out fault, the voltage must exceed VAC_DO_Thr plus a hysteresis, VAC_DO_Hyst, and stay above this threshold for the duration of a deglitch window with a length of VAC_Recov_Deglitch_Time. If the voltage fails to stay above the threshold throughout the deglitch window, FaultFlags[6] remains set.

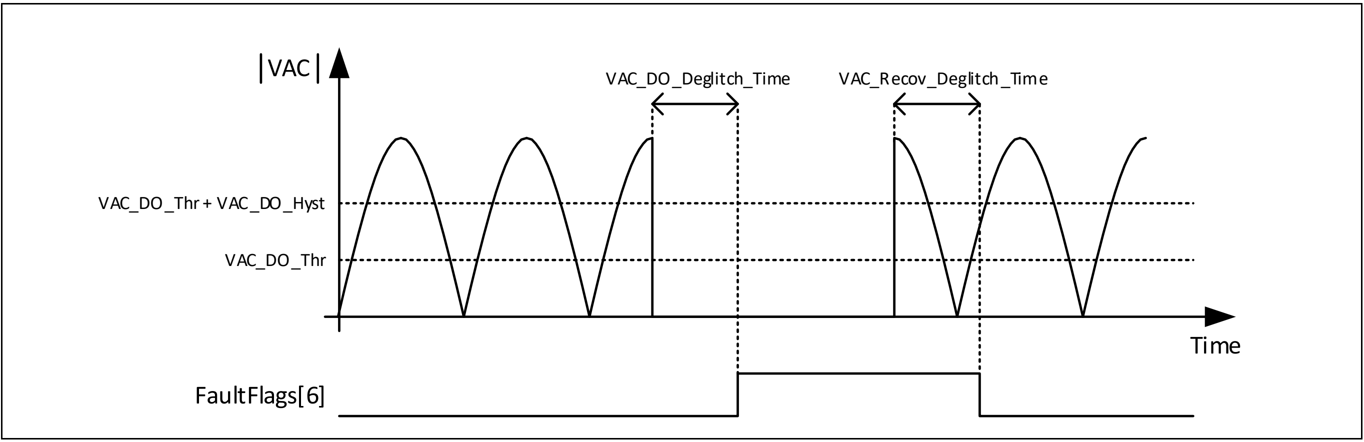


**Figure 67 VAC drop-out voltage detection can clear.**

#### 2.2.4.2 VAC Over Voltage and Brown-out Protection

AC Over Voltage fault becomes active when the AC input voltage RMS value is above a configurable threshold and AC brown-out (undervoltage) fault becomes active when the AC input voltage RMS value is below a configurable threshold. A hysteresis and a deglitch window are added to prevent rapid toggling between normal- and fault conditions.

Both over voltage- and brown-out protection operates on the RMS of the input voltage. The instantaneous AC input voltage is sampled every PFC base-rate cycle and the RMS value of the AC input voltage is updated every half-line cycle when AC input voltage zero-crossing is detected.

The VAC over-voltage detection and clear is illustrated in Figure 68 with the absolute value of the instantaneous AC voltage, VAC, shown on the top, RMS value of the AC voltage, VAC RMS, in the middle and fault reporting parameter FaultFlags, at the bottom. Note how the VAC RMS voltage is updated at every zero-crossing of VAC. If the VAC RMS value exceeds VAC_OVP_Thr, and remains higher than the threshold during a deglitch window of length VAC_OVP_Deglitch_Time, bit 5 of FaultFlags is set. If the voltage drops below the threshold during the deglitch window, the fault is not set. To clear the overvoltage fault, the voltage must drop below VAC_OVP_Thr minus a hysteresis, VAC_OVP_Hyst and stay below this threshold for the duration of a deglitch window with a length of VAC_Norm_Deglitch_Time. If the voltage fails to stay below the threshold throughout the deglitch window, FaultFlags[5] remains set. The length of the deglitch window is an integer number of half-line-cycle periods, THLC.

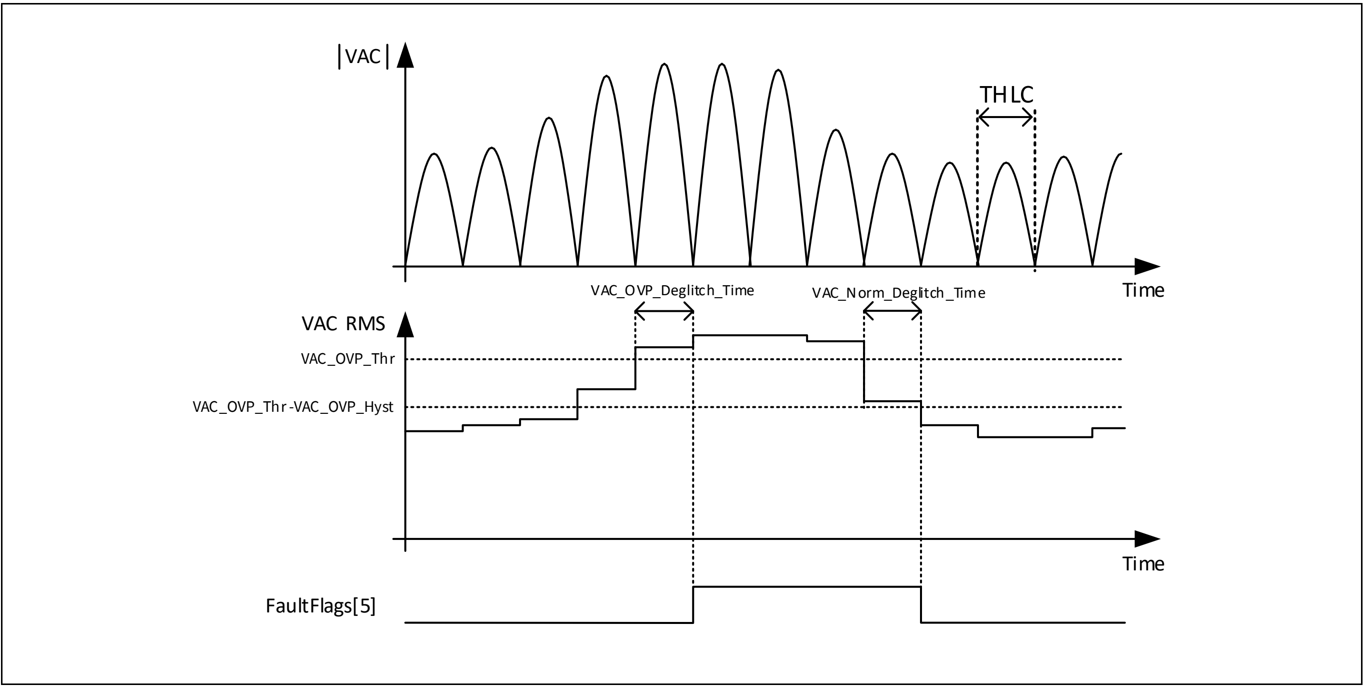


**Figure 68 VAC over voltage detection and clear.**

The VAC Brown-out (undervoltage) detection and clear is illustrated in Figure 69, with the absolute value of the instantaneous AC voltage, VAC, shown on the top, RMS value of the AC voltage, VAC RMS, in the middle and fault reporting parameter FaultFlags, at the bottom. If the calculated VAC RMS drops VAC_BO_Thr, and remains lower than the threshold during a deglitch window of length VAC_BO_Deglitch_Time, bit 4 of FaultFlags is set. If the voltage exceeds the threshold during the deglitch window, the fault is not set. To clear the brown-out fault, the voltage must exceed VAC_BO_Thr plus a hysteresis, VAC_BO_Hyst, and stay above this threshold for the duration of a deglitch window with a length of VAC_Norm_Deglitch_Time. If the voltage fails to stay above the threshold throughout the deglitch window, FaultFlags[4] remains set.

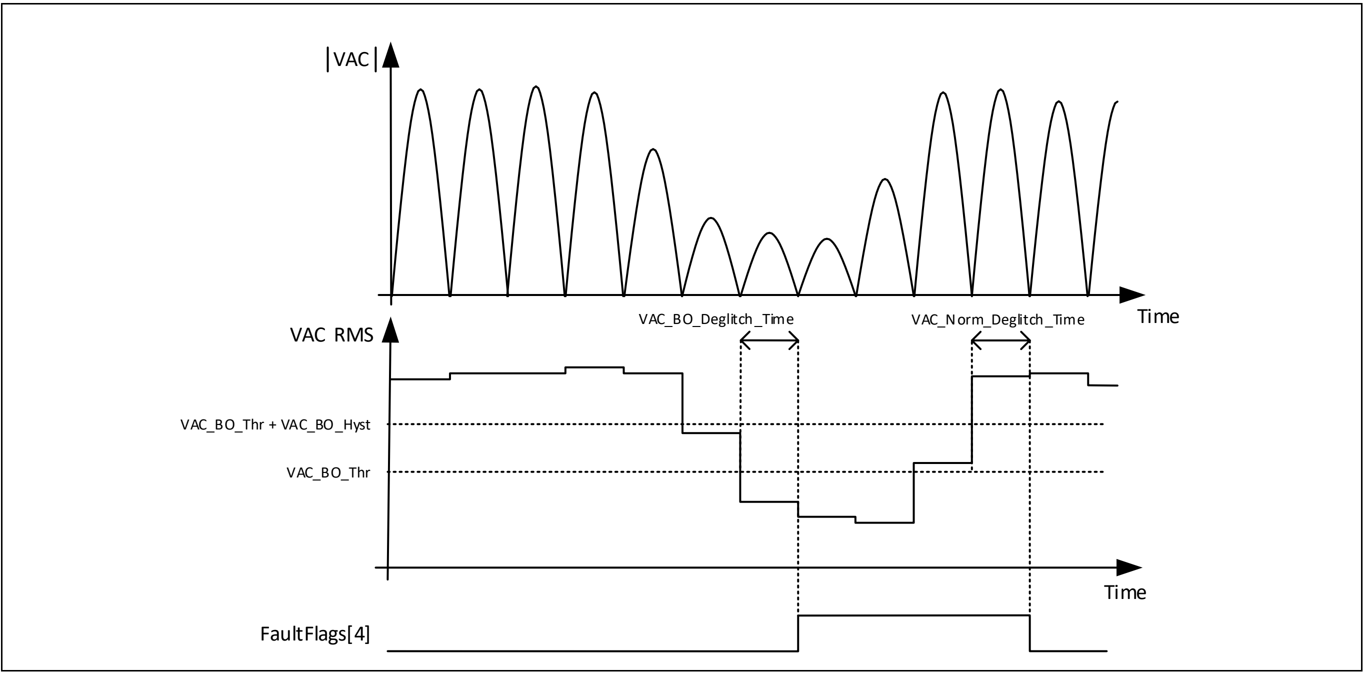


**Figure 69 VAC brown-out detection and clear.**

### 2.2.4.3 Input Frequency Protection

The MCE monitors the actual AC line frequency and asserts a fault if it falls outside a configured window. The user can choose between 50Hz or 60Hz as nominal line frequency. Valid range of the actual AC input frequency is also configurable. For example, with a nominal AC frequency of 50Hz, a typical valid range of actual AC input frequency is from 47 to 53Hz.

The AC input frequency is determined by measuring the time between zero-crossings of the input voltage. During each Base Rate cycle the MCE checks for zero-crossing and increments a counter when a zero-crossing is *not* detected. When a zero-crossing *is* detected the counter value is latched and stored in parameter THLC which then holds the number of Base Rate cycle per half line cycle period. Note, that a long half line cycle period corresponds to a low frequency. AC input frequency is checked against min/max limits at the Sequencer Update Rate (1kHz).

The principle behind the Input Frequency Protection function is shown in Figure 70. If the measured positive- or negative half line cycle period, THCL, is greater than the max limit, THLC_Validation_Max_Thr, and remains higher than the threshold during a deglitch window of length THLC_Validation_Deglitch_Time, bit 3 of FaultFlags is set to indicate a low line frequency. If the half line cycle period drops below the threshold during the deglitch window, the fault is not set. The frequency fault is cleared when the half line cycle period drops below THLC_Validation_Max_Thr minus a hysteresis, THLC_Validation_Max_Hyst and stays below this threshold for the duration of a deglitch window with a length of THLC_Validation_Deglitch_Time. If the half line cycle period fails to stay below the threshold throughout the deglitch window, FaultFlags[3] remains set.

Similarly, If the measured half line cycle period, THCL, is less than the max limit, THLC_Validation_Min_Thr, and remains lower than the threshold during a deglitch window of length THLC_Validation_Deglitch_Time, bit 3 of FaultFlags is set to indicate a high line frequency. If the half line cycle period exceeds the threshold during the deglitch window, the fault is not set. The frequency fault is cleared when the half line cycle period exceeds THLC_Validation_Min_Thr plus a hysteresis, THLC_Validation_Min_Hyst and stays above this threshold for the duration of a deglitch window with a length of THLC_Validation_Deglitch_Time. If the half line cycle period fails to stay above the threshold throughout the deglitch window, FaultFlags[3] remains set.

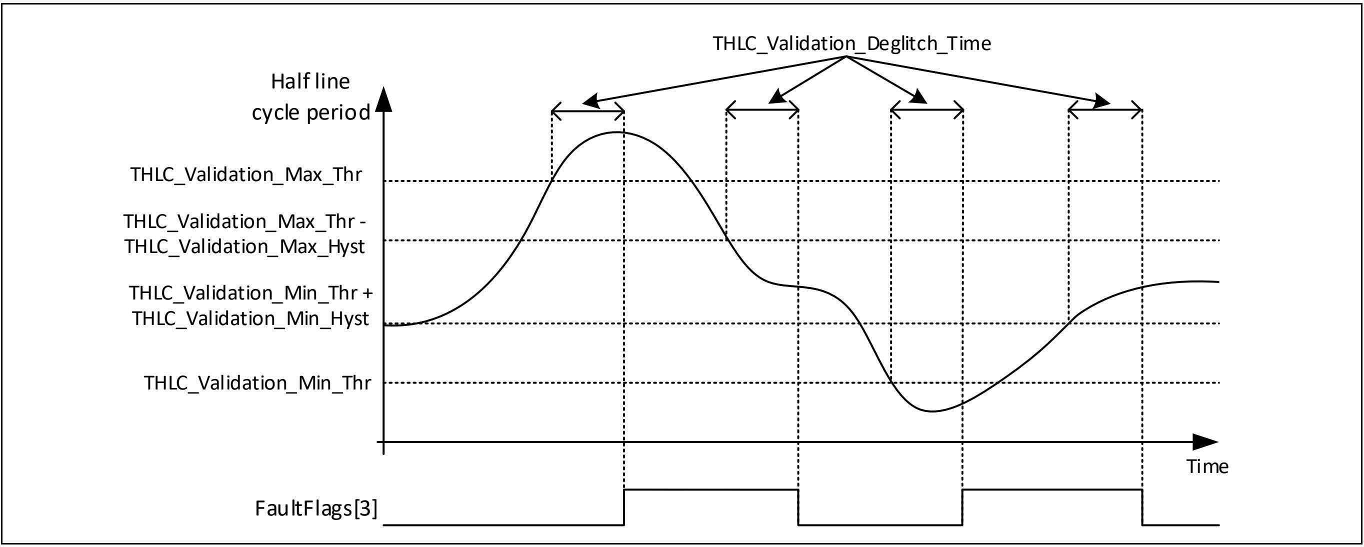

THLC_Validation_Deglitch_Time
Half line cycle period
THLC_Validation_Max_Thr
THLC_Validation_Max_Thr - THLC_Validation_Max_Hyst
THLC_Validation_Min_Thr + THLC_Validation_Min_Hyst
THLC_Validation_Min_Thr
Time
FaultFlags[3]

**Figure 70 Min/max input frequency detection and clear.**

### 2.2.4.4 Vout Over Voltage and Open Loop Protection

Vout over voltage fault is active when the DC-bus voltage exceeds a configurable threshold and Vout Open Loop (undervoltage) fault is active when the DC-bus voltage drops below a configurable threshold. A hysteresis and a deglitch window are added to prevent rapid toggling between normal- and fault conditions. The DC-bus voltage is sampled every PFC switching cycle and can be read from the parameter Vout.

The DC-bus over-voltage detection and clearing is illustrated in Figure 71. If the DC-bus voltage exceeds Vout_OVP_Thr, and remains higher than the threshold during a deglitch window of length Vout_OVP_Deglitch_Time, bit 2 of FaultFlags is set. If the voltage drops below the threshold during the deglitch window, the fault is not set. To clear the overvoltage fault, the voltage must drop below Vout_OVP_Thr minus a hysteresis, Vout_OVP_Hyst and stay below this threshold for the duration of a deglitch window with a length of Vout_Norm_Deglitch_Time. If the voltage fails to stay below the threshold throughout the deglitch window, FaultFlags[2] remains set.

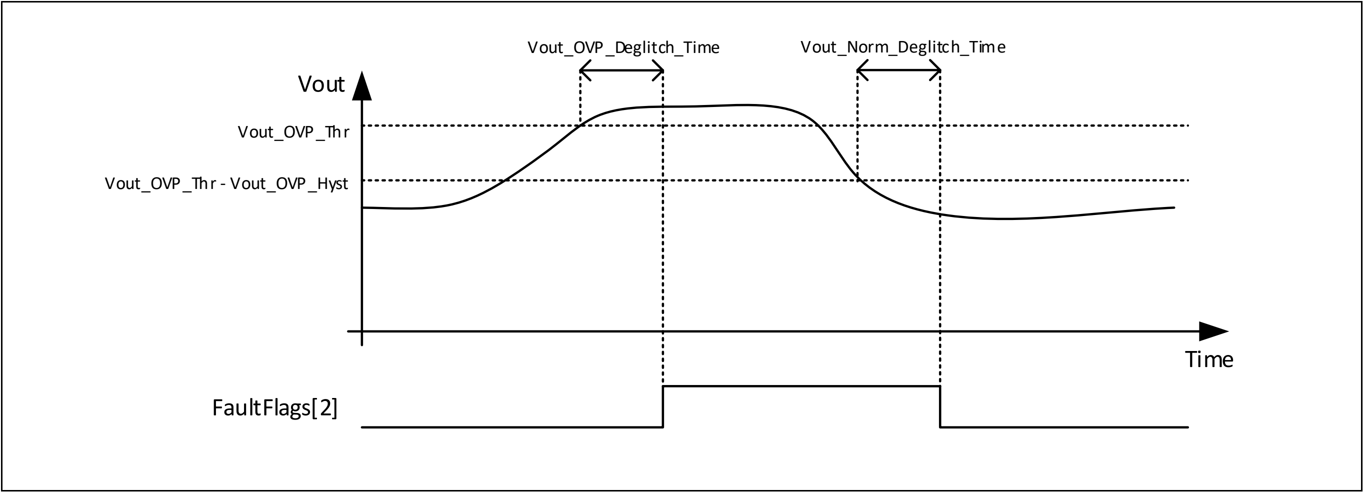


**Figure 71 Vout over voltage detection and clear.**

The Vout Open-Loop (undervoltage) detection and clear is illustrated in Figure 72. If the DC-bus voltage drops below Vout_OLP_Thr, and remains lower than the threshold during a deglitch window of length Vout_OLP_Deglitch_Time, bit 1 of FaultFlags is set. If the voltage rises above the threshold during the deglitch window, the fault is not set. To clear the brown-out fault, the voltage must exceed Vout_OLP_Thr plus a hysteresis, Vout_OLP_Hyst, and stay above this threshold for the duration of a deglitch window with a length of Vout_Norm_Deglitch_Time. If voltage fails to stay above the threshold throughout the deglitch window, FaultFlags[1] remains set.

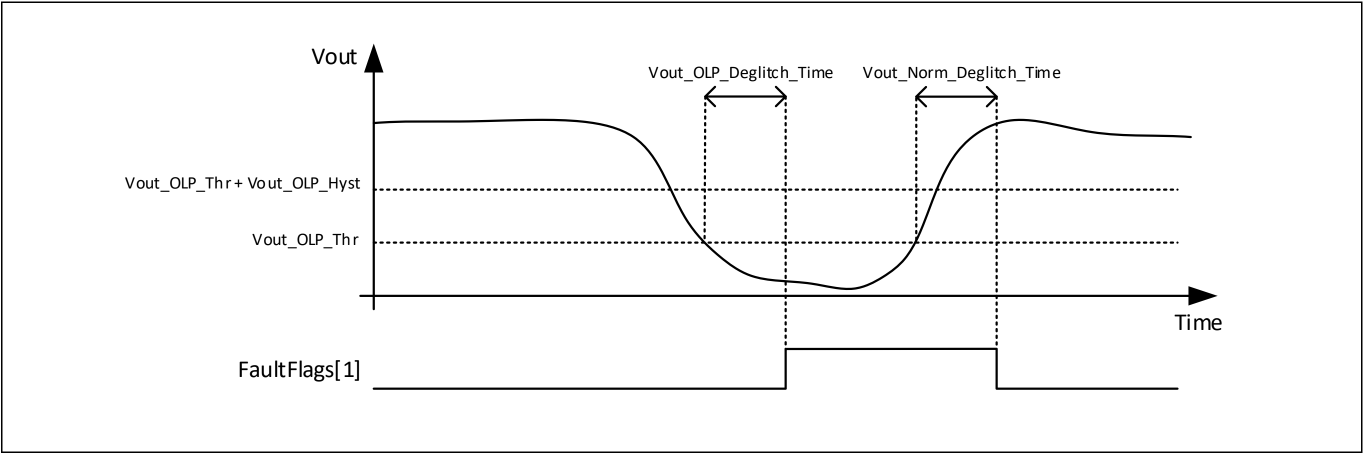


**Figure 72 Vout open loop detection and clear.**

### 2.2.4.5 Current Measurement Offset

As part of the PFC startup, the offset of the current measurement channel, ILOffset, is determined and used for calibration of the inductor current samples. If this measured offset falls outside configurable limits, ILOffset_Min and ILOffset_Max, a protection function prevents the PFC from starting and raises a FaultFlags[7]. Out of range Current Measurement Offset is the only fault function that brings the PFC into a dedicated fault state.

An out of range offset indicates a faulting measurement circuit that cannot be relied on for closed loop current control. To prevent PFC operation, the fault cannot be cleared by the user nor will it automatically clear. Only an MCE reset will initiate a new start attempt.

## 2.3 User Mode UART

The user mode UART communication is designed to provide a simple, reliable and scalable communication protocol for motor control application. The protocol is simple so that it can be easily implemented even in low-end microcontrollers which work as master to control the motor. It supports networking (up to 15 nodes on same network) which is required in some industrial fan/pump applications. Each UART commands are processed every 1ms.
If users intend to implement a customized UART communication protocol, it can be realized by using those configurable UART driver methods described in section 2.6.10.3.

### 2.3.1 Baud Rate

The MCE supports the following Baud rate configuration for user mode UART: 2400 bps, 9600 bps, 19200 bps, 67500 bps, 115200 bps, and 230400 bps.

### 2.3.2 Data Frame

The format of the data frame is shown in Figure 73. Notice that it follows little endian format.

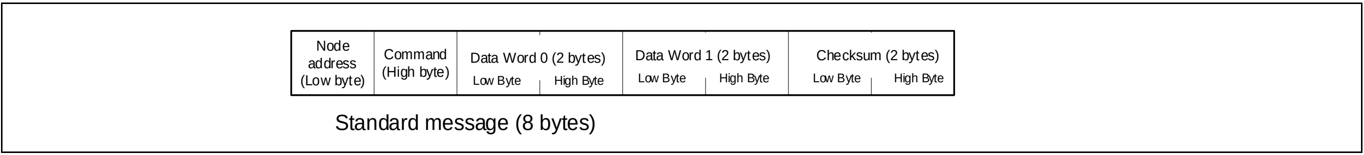


**Figure 73 UART Data Frame**

### 2.3.3 Node Address

Node address is the first byte in a data frame. It is designed to allow one master controlling multiple slaves in the same network. Each slave node has its unique node ID. The slave only acknowledges and responds to the message with same ID. There are two broadcast addresses (0x00 and 0xFF) defined for different usage. If a message is received with address=0x00, all the slaves execute the command but will not send a reply to the master. This is useful in a multiple slave network and the master needs to control all the slaves at the same time, for example, turn on all the motor by sending only one message. If received a frame with address=0xFF, the slave will execute the command and also send a reply to the master. This is useful in 1-to-1 configuration when the master doesn't know or doesn't need to know the slave node address.

**Table 15 Node Address Definition**

| Node Address | Command |
|---|---|
| 0x00 | All nodes receive and execute command, no response. |
| 0x01 to 0x0F | Only the node that has same address executes the command and replies the master. |
| 0x10 to 0xFE | Reserved |
| 0xFF | All nodes receive and execute the command and reply the master. Only used in 1-to-1 configuration. It will cause conflict if multiple nodes connected to the same network |

### 2.3.4 Link Break Protection

Link break protection is to stop the motor if there is no UART communication for certain period of time. In some application, the main controller maintains communication with the motor controller. In case of a loss of communication or line break, it is desired to stop the motor for protection. This protection feature is enabled or disable and Link break timeout is configured in Solution Designer.

### 2.3.5 Command

UART command is the second byte in a data frame. Bit [6:0] specifies the command code. Bit [7] is the indication bit indicates the direction of the data frame. All data frames sent by master must have bit 7 cleared (=0), all reply data frames sent by slave must have bit 7 set (=1).

**Table 16 UART Command Definition**

| Command (Bit[6:0]) | Description |
|---|---|
| 0 | Read Status |
| 1 | Request to clear fault flag |
| 2 | Select Control input mode |
| 3 | Set motor control target speed |
| 4 | Not used, slave will not reply to master |
| 5 | Read Register |
| 6 | Write Register |
| 7 - 31 | Not used, slave will not reply to master |
| 32 | Load or save parameter set |
| 33-127 | Not used, slave will not reply to master |

### 2.3.6 Checksum

Checksum is 16-bit format and it shall be calculated as below:

[Command: Node address] + Data Word 0 + Data Word 1 + Checksum = 0x0000

Notice that when sending the checksum word to the user UART interface, little endian format shall be followed as shown in Figure 73.

Checksum calculation example:
Input: Node address = 1, command = 2, Data Word 0 = 0x1122 and Data Word 1 = 0x3344
[Command: Node address] = 0x0201
Checksum = -1 x (0x0201 + 0x1122 + 0x3344) = 0xB999
Data frame: 0x01 (node address byte), 0x02 (command byte), 0x22 (lower byte of data word 0), 0x11 (higher byte of data word 0), 0x44 (lower byte of data word 1), 0x33 (higher byte of data word 1), 0x99 (lower byte of checksum word), 0xB9 (higher byte of checksum word)

## 2.3.7 UART message

### 2.3.7.1 Read Status: Command = 0x00

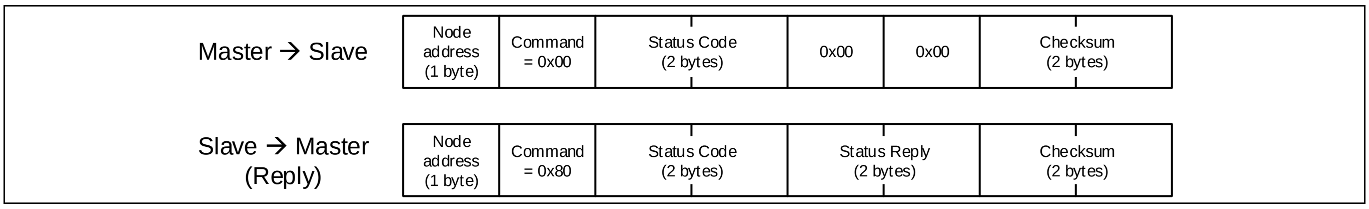


**Figure 74 Read Status command**

**Table 17 Status code and status reply**

| Status code | status reply |
|---|---|
| 0x0000 | Fault Flags |
| 0x0001 | Motor Speed |
| 0x0002 | Motor State |
| 0x0003 | Node ID |
| 0x0004 – 0xFFFF | 0x0000 |

Clear Fault: Command =0x01

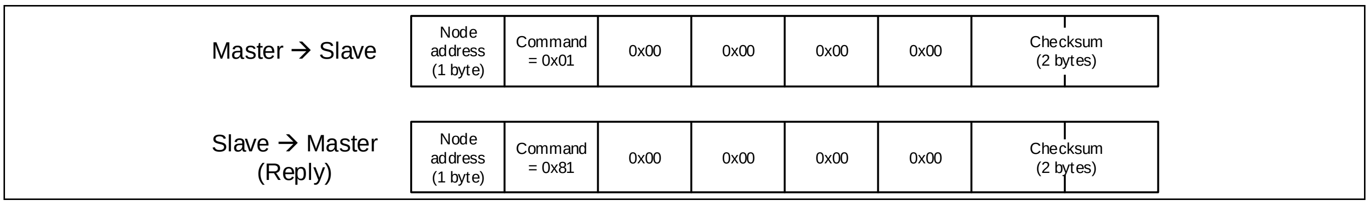


**Figure 75 Clear fault command**

### 2.3.7.2 Change Control Input Mode: Command =0x02

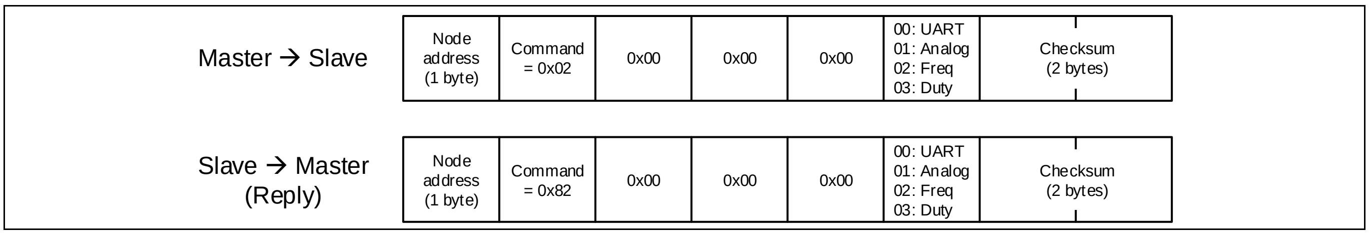


**Figure 76 Control input mode command**

### 2.3.7.3 Motor Control: Command =0x03

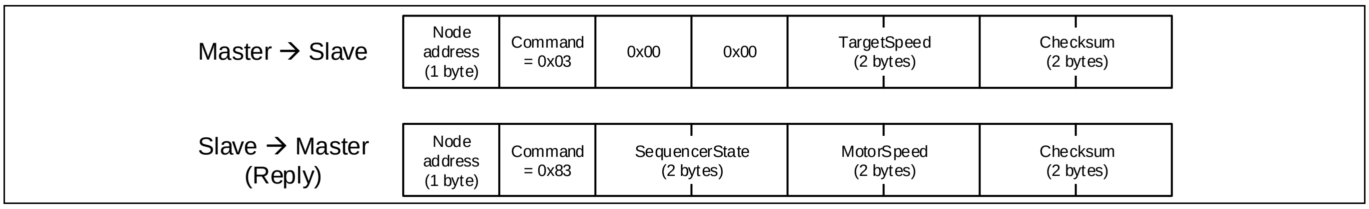


**Figure 77 Motor control Command**

*Note: Target Speed=0: motor stop, TargetSpeed≠0: motor start*

### 2.3.7.4 Register Read: Command = 0x05

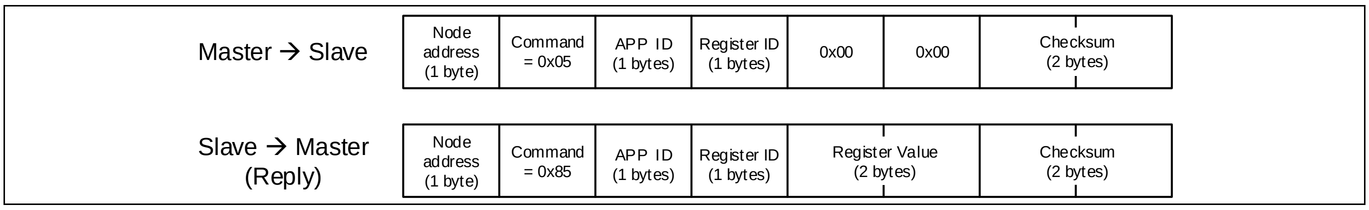


**Figure 78 Register Read Command**

### 2.3.7.5 Register Write: Command = 0x06

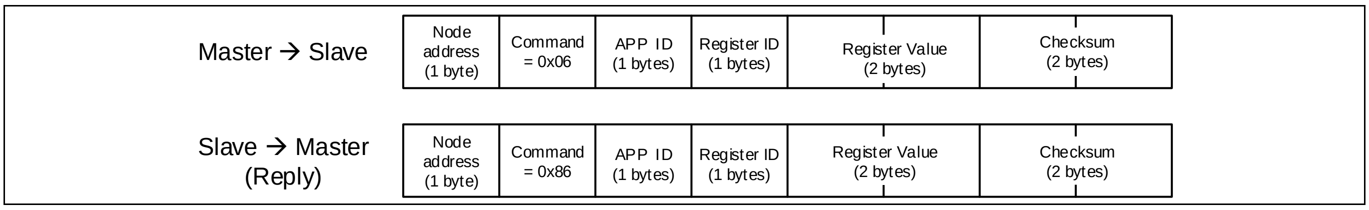


**Figure 79 Register Write Command**

### 2.3.7.6 Load and Save Parameter: Command = 0x20

'Load parameter' command loads the parameters from the specified parameter set stored in FLASH into the RAM. The valid range of the parameter set number is: 0 – 14. In the reply frame, data 0 word contains the value of 'Status' (0: success; 1: fail; 2: parameter set number not supported.)

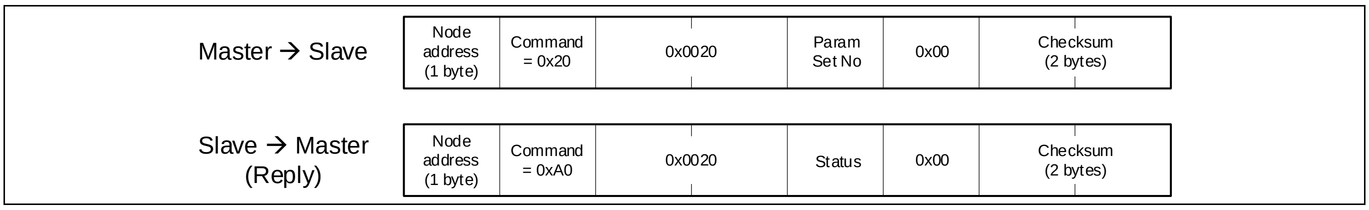


**Figure 80 Load parameter Command**

'Save parameter' command erases the selected parameter set first and saves the parameters of the specified App ID to this parameter set. The valid range of the parameter set number is: 0 – 14. The valid App ID value is: 1 or 3. In the reply frame, data 0 word contains the value of 'Status' (0: success; 1: fail; 2: parameter set number not supported.)

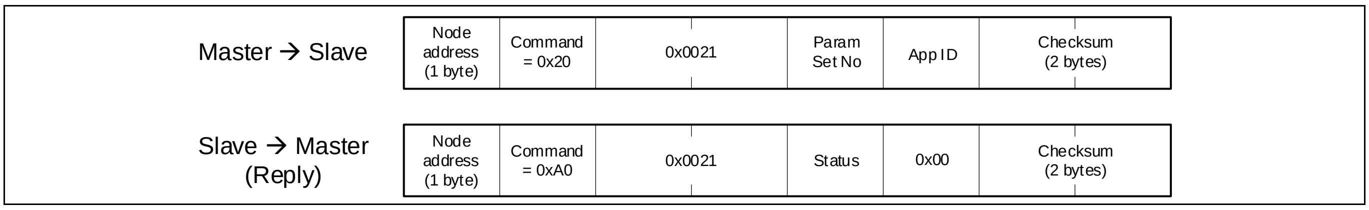


**Figure 81 Save Parameter Command**

### 2.3.8 Connecting multiple nodes to same network

It is possible to connect multiple MCE to same UART network, see Figure 82 detail.
For the TXD pin of each MCE node, it needs to connect a Schottky diode before connect to the same wire, and on the master controller side, a 4.7kOhm pull up resister is required.

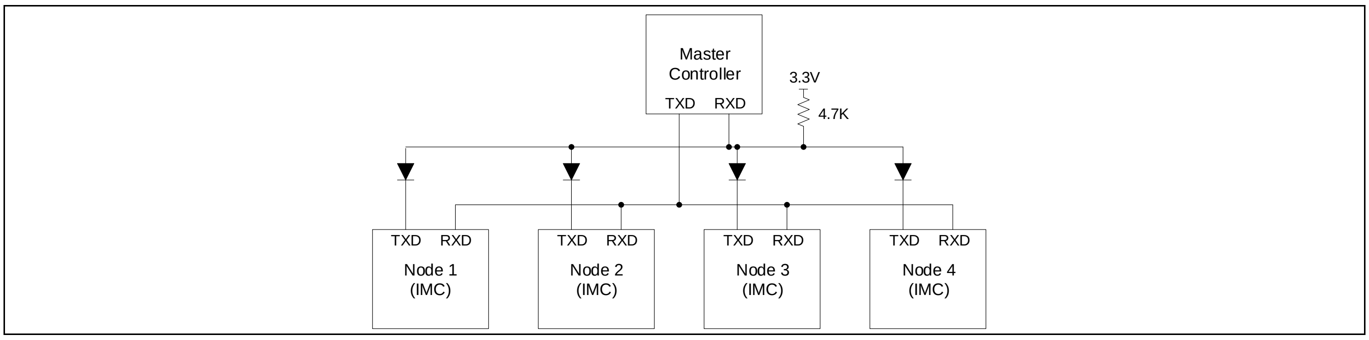


**Figure 82 UART network connection**

### 2.3.9 UART Transmission Delay

A configurable delay (bit [14:7] of parameter 'UARTConf') can be inserted between the reception of a message from the host and the transmission of a response message.

## 2.4 JCOM Inter-Chip Communication

The JCOM interface is designed to provide a means of point-to-point bi-directional communication for dual-core products between the motor control core running the MCE (named T core hereafter) and the integrated MCU (named A core hereafter). JCOM interface utilizes an internal serial port. JCOM protocol assumes one master and one slave during communication. JCOM interface can be enabled by using bit field [5:3] of the parameter 'TargetInterfaceConf' and parameter 'JCOMConf'.

### 2.4.1 Operation Mode

JCOM interface supports asynchronous mode between the master and the slave.

#### 2.4.1.1 Asynchronous Mode

In asynchronous mode, the A core (MCU) serves as the master, while the T core (MCE based motor control) serves as the slave. All communication activities are initiated by the master.

From the slave side, JCOM interface driver is interrupt driven to ensure that the response from T core is handled with minimum delay. As soon as enough data is accumulated in the reception FIFO, the JCOM interrupt handler is triggered where the received frame is parsed to extract the message payload. Based on the Message Object (MO) number, relevant action is executed per the Command and Response Protocol. Then, the response frame is constructed and sent to the transmission FIFO.

### 2.4.2 Baud Rate

The Baud rate of JCOM interface can be configured at the start-up or during run-time. The valid range is from 6.1 Kbps to 6 Mbps. The default Baud rate is 1 Mbps.

If the T core JCOM interface experiences some frame error more than 3 times due to mismatch of Baud rate configuration between the A core and the T core, then the Baud rate of JCOM interface of the T core would be reset to the default value (1 Mbps) automatically.

### 2.4.3 Message Frame Structure

Each JCOM message frame consists of the following fields assuming transmission sequence is from left to right. The following Figure 83 shows the details of the JCOM message frame structure.

| Flag | Seq | Res | Message | | | | | CRC | Flag |
|---|---|---|---|---|---|---|---|---|---|
| | | | MO | Data[0] | Data[1] | Data[2] | Data[3] | | |
| 1 byte (0x7E) | 2 bit | 2 bit | 4 bit | 4 bytes | | | | 1 byte | 1 byte (0x7E) |

**Figure 83 JCOM Message Frame Structure**

Flag: Indication of the start and end of a frame.

Seq: This sequence number is used to detect a wrong sequence fault. During normal operation, Seq number is incremented per frame and checked at the receiver side. If the Seq number doesn't match, then the entire frame is ignored and no response is sent.

Res: Reserved for future use.

MO: This Message Object number defines how the data is interpreted.

Data[x]: These data fields contain the payload of the message.

CRC: The CRC byte is calculated over the message fields including the MO number. If CRC check fails, then the entire frame is ignored and no response is sent.

## 2.4.4 Command and Response Protocol

The command and response protocol is used when JCOM interface works in asynchronous mode. The message contains a Message Object number and 4 data bytes. Under the 'direction' column found in the following Message Structure figures, 'DS' refers to communication from master (A core) to slave (T core), and 'US' refers to communication from slave (T core) to master (A core). If a command frame sent from the master is successfully received by the slave and passes CRC check, then a corresponding response frame would be sent from the slave. If the command frame sent from the master is out of synchronization due to Seq number mismatch, or fails the CRC check, then the entire command frame is ignored by the slave with no response. Some time-out recovery mechanism is recommended from the master side to deal with those faults. The following Table 188 summarizes the functions corresponding to different MO numbers.

**Table 18 Message Object Function Table**

| Message Object | Functions |
|---|---|
| 0 | State machine inquiry;<br>Execution time and CPU load inquiry. |
| 1 | System configuration protection;<br>Reset T core;<br>Access static parameter;<br>Set boot mode;<br>Set JCOM Baud rate. |
| 6 | Get parameter. |
| 7 | Set parameter. |
| Others | Reserved for future use. |

### 2.4.4.1 Message Object: 0

The following Figure 84 shows the details of the message structure with MO set to 0. With MO = 0, data[0] contains a status byte that represents the type of objects whose status is requested.

| Direction | Data[0] | Data[1] | Data[2] | Data[3] | Comments |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| DS | Status | x | x | x | Status = 0: returns the state number of the SM0 (motor) and SM1 (PFC) state machines.<br>Status = 1: returns execution time for system task and CPU_Load. |
| US | SM0 state | SM1 state | 0xFF | 0xFF | Status = 0 |
| US | exe_sys | | cpu_load | | Status = 1 |

**Figure 84 Message Structure (MO = 0)**

#### 2.4.4.1.1 State Machine Inquiry

If the status byte = 0 in the command frame, then the relevant state numbers of the motor and PFC state machines are requested by the master. The response frame is supposed to contain the state number ('Motor_SequencerState') of the motor state machine in data[0] and the state number ('PFC_SequencerState') of the PFC state machine in data[1].

#### 2.4.4.1.2 Execution Time and CPU Load Inquiry

If the status byte = 1 in the command frame, then the execution time for the system task scheduled in systick ISR (typically every 1 ms) and the CPU load are requested. The response frame is supposed to contain the execution time word (1 count = 0.33 µs) for the system task in data[0] (lower 8 bit of execution time word) and data[1] (higher 8 bit of execution time word), as well as the CPU_Load word (1 count = 0.1%) in data[2] (lower 8 bit of CPU_Load word) and data[3] (higher 8 bit of CPU_Load word).

### 2.4.4.2 Message Object: 1

The following Figure 85 shows the details of the message structure with MO set to 1. With MO = 1, the command frame contains a Command word in data[0] and data[1] and a Value word when applicable in data[2] and data[3]. The response frame is supposed to contain the same Command word in data[0] and data[1] and the same Value word in data[2] and data[3] to acknowledge successful reception.

| **Direction** | **Data[0]** | **Data[1]** | **Data[2]** | **Data[3]** | **Comments** |
|---|---|---|---|---|---|
| | **Command** | | **Value** | | |
| **System Configuration** | | | | | |
| DS | 0x0000 | | p | | Configuration protection:<br>p = 0: protected<br>0 < p < 3: unprotected for the next p commands |
| DS | 0x0001 | | 0 | | Reset (immediately) |
| DS | 0x0002 | | a | | Static parameter access:<br>a = 0: disable<br>a = 1: enable |
| DS | 0x00BD | | (~bmd<<8)+bmd | | Set boot mode |
| **JCOM Configuration** | | | | | |
| DS | 0x0100 | | Baud rate | | Set JCOM Baud rate |
| **Parameter Handler Commands** | | | | | |
| DS | 0x0200 | | x | | Enable coherent parameter handling |

| | | | |
|---|---|---|---|
| DS | 0x0201 | x | Disable coherent parameter handling |
| DS | 0x0202 | x | Set parameter coherently |
| **File Handler Commands** | | | |
| DS | 0x0300 | page | load parameter file |
| DS | 0x0301 | (appID<<8) + page | save parameter file |
| DS | 0x0302 | page | erase parameter file |
| **Response** | | | |
| US | Command | Value | Acknowledge from slave |

**Figure 85 Message Structure (MO = 1)**

#### 2.4.4.2.1 System Configuration Protection

Changing system configuration requires going through a 2-step unlock process for safety concerns. Those operations include resetting T core, accessing static parameters, as well as setting boot mode.

The 1st step is to have the master send a command frame (MO = 1) with Command = 0x0000 and Value = p to unprotect the next p commands. p can be set to 1 or 2.

The 2nd step is to have the master send a command frame (MO = 1) with one of those system configuration related commands to change system configuration.

#### 2.4.4.2.2 Reset T Core

A core can perform a reset request for T core by the following steps.

The 1st step is to have the master send a command frame (MO = 1) with Command = 0x0000 and Value = 1 to unprotect the next 1 command.

The 2nd step is to have the master send a command frame (MO = 1) with Command = 0x0001 and Value = 0. Upon receiving this frame, the T core will immediately reset itself with no response US frame.

#### 2.4.4.2.3 Access Static Parameter

Writing to those static type of parameters is not allowed by default. A 2-step unlock process is needed to obtain write access to the static type of parameters. Without going through this process, attempting to write to those static type of parameters would have no effect.

The 1st step is to have the master send a command frame (MO = 1) with Command = 0x0000 and Value = 1 to unprotect the next 1 command.

The 2nd step is to have the master send a command frame (MO = 1) with Command = 0x0002 and Value = 1 to grant write access to those static type of parameters.

Then the master has the right to write to those static type of parameters using a command frame with MO = 7. After the write operation is completed, it is recommended to disable the write access to those static type of parameters by the same 2-step lock process.

The 1st step is to have the master send a command frame (MO = 1) with Command = 0x0000 and Value = 1 to unprotect the next 1 command.

The 2nd step is to have the master send a command frame (MO = 1) with Command = 0x0002 and Value = 0 to disable write access to those static type of parameters.

##### 2.4.4.2.4 Set Boot Mode

By default T core (MCE) operates in Application Mode. A core can request changing the MCE to Configuration Mode (BMD = 0xCD) or Boot-Loader Mode (BMD = 0x5D) by the following steps.

The 1st step is to have the master send a command frame (MO = 1) with Command = 0x0000 and Value = 1 to unprotect the next 1 command.

The 2nd step is to have the master send a command frame (MO = 1) with Command = 0x00BD and Value = 0x32CD to set the boot mode to Configuration Mode, or Value = 0xA25D to set the boot mode to Boot-Loader Mode.

##### 2.4.4.2.5 Set JCOM Baud Rate

The master can request changing the Baud rate of the JCOM interface of the slave by sending a command frame (MO = 1) with Command = 0x0100 and Value = desired Baud rate (bps) / 100.

#### 2.4.4.3 Message Object: 6

##### 2.4.4.3.1 Get Parameter

The following Figure 86 shows the details of the message structure with MO set to 6. Each parameter can be addressed using it unique App ID and Index number as described in Section **Error! Reference source not found.**. With MO = 6, the command frame contains the App ID byte in data[0] and the Index byte in data[1] of the specified parameter or variable. The response frame is supposed to contain the same App ID byte in data[0], the same Index byte in data[1], and the Value word of the requested parameter or variable in data[2] and data[3].

| **Direction** | **Data[0]** | **Data[1]** | **Data[2]** | **Data[3]** | **Comments** |
|---|---|---|---|---|---|
| DS | App ID | Index | 0x0000 | | Get parameter |
| | | | **Response** | | |
| US | App ID | Index | Value | | Send requested parameter |

**Figure 86 Message Structure (MO = 6)**

#### 2.4.4.4 Message Object: 7

##### 2.4.4.4.1 Set Parameter

The following Figure 87 shows the details of the message structure with MO set to 7. With MO = 7, the command frame contains the App ID byte in data[0], the Index byte in data[1], and the Value word in data[2] and data[3] of the specified parameter or variable. The response frame is supposed to contain the same App ID byte in data[0], the same Index byte in data[1], and the same Value word of the requested parameter or variable in data[2] and data[3] to confirm a successful operation.

| **Direction** | **Data[0]** | **Data[1]** | **Data[2]** | **Data[3]** | **Comments** |
|---|---|---|---|---|---|
| DS | App ID | Index | Value | | Set parameter |

| response | | | | |
|---|---|---|---|---|
| US | App ID | Index | Value | Send back parameter for confirmation |

Figure 87 Message Structure (MO = 7)

## 2.5 Multiple Parameter Programming

### 2.5.1 Parameter Page Layout

In iMOTION$^{TM}$ product, 4k bytes of flash memory are used to store control parameter data. There are totally 16 parameter blocks, each parameter block is 256 bytes in size. Multiple parameter blocks up to a maximum of 15 can be used to support different motor types or hardware. Block 15 is reserved to store system parameters (App ID = 0).

Active parameter set is specified by a parameter set number, which can be configured using Solution Designer. Solution Designer output (*.txt) that contains the parameter values, can be programmed into the parameter block using Solution Designer. Solution Designer output file contains the specified parameter set number. Solution Designer loads the parameter values into the corresponding parameter block. Each parameter block can be updated individually multiple times.

For a system with only a motor control (App ID = 1) function, each parameter set will take one parameter block. In this case, the valid parameter set IDs can range from 0 to 14.

For a system with motor control (App ID = 1) and PFC (App ID = 3) functions, each parameter set will take two consecutive parameter blocks. The motor control parameter set will be stored into the selected parameter block and the PFC parameter set will be stored into the immediate following parameter block. In this case, the valid parameter set IDs are 0, 2, 4, 6, 8, 10, and 12.

### 2.5.2 Parameter Block Selection

MCE supports to select the parameter block in 4 different methods.

- Direct Select : ParSetConf[3:0] =0
- UART Control : ParSetConf[3:0] =1
- Analog Input: ParSetConf[3:0] =2
- GPIO Pins : : ParSetConf[3:4] =3

Parameter block selection input configuration is available in Solution Designer and Solution Designer updated "ParSetConf" parameter.

*Note: Not all of the 4 methods to select parameter block are available in all iMOTION$^{TM}$ devices, due to pin availabilities. Refer specific device datasheet for available methods to select parameter block.*

#### 2.5.2.1 Direct Select

Parameters block selection is based on "ParSetConf [7:4]" parameter bit field value. "ParSetConf [7:4]" parameter bit field value can be updated from Solution Designer.

#### 2.5.2.2 UART Control

Specific UART messages are defined to load the parameter block from flash to RAM and save the parameter set from RAM to flash. Refer section 2.3.7.6 for message format.

#### 2.5.2.3 Analog Input

Parameter block is selected based on the analog input value. MCE uses "PARAM" pin as the Analog input for parameter set selection. Mapping between parameter page selections based on Analog input mentioned below

$$ParameterBlock = Integer\left\{\left(\frac{AnalogInput}{Vadcref} * 15\right)\right\}$$

*Example if AnalogInput = 1.2V and $V_{adcref}$ =3.3V, then ParameterBlock = 5*

*Note: Maximum value of parameter block is 14.*

#### 2.5.2.4 GPIO Pins

Parameter block is selected based on the four GPIO pins. GPIO pins used for parameter set selection are named as "PAR0", "PAR1", "PAR2" and "PAR3". Mapping between parameter page selections based on GPIO pins are listed in the Table 19.

**Table 19 Parameter page Selection for GPIO**

| GPIO Input | | | | Parameter Block |
|---|---|---|---|---|
| **PAR3** | **PAR2** | **PAR1** | **PAR0** | |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 2 |
| 0 | 0 | 1 | 1 | 3 |
| 0 | 1 | 0 | 0 | 4 |
| 0 | 1 | 0 | 1 | 5 |
| 0 | 1 | 1 | 0 | 6 |
| 0 | 1 | 1 | 1 | 7 |
| 1 | 0 | 0 | 0 | 8 |
| 1 | 0 | 0 | 1 | 9 |
| 1 | 0 | 1 | 0 | 10 |
| 1 | 0 | 1 | 1 | 11 |
| 1 | 1 | 0 | 0 | 12 |
| 1 | 1 | 0 | 1 | 13 |
| 1 | 1 | 1 | 0 | 14 |
| 1 | 1 | 1 | 1 | 14 |

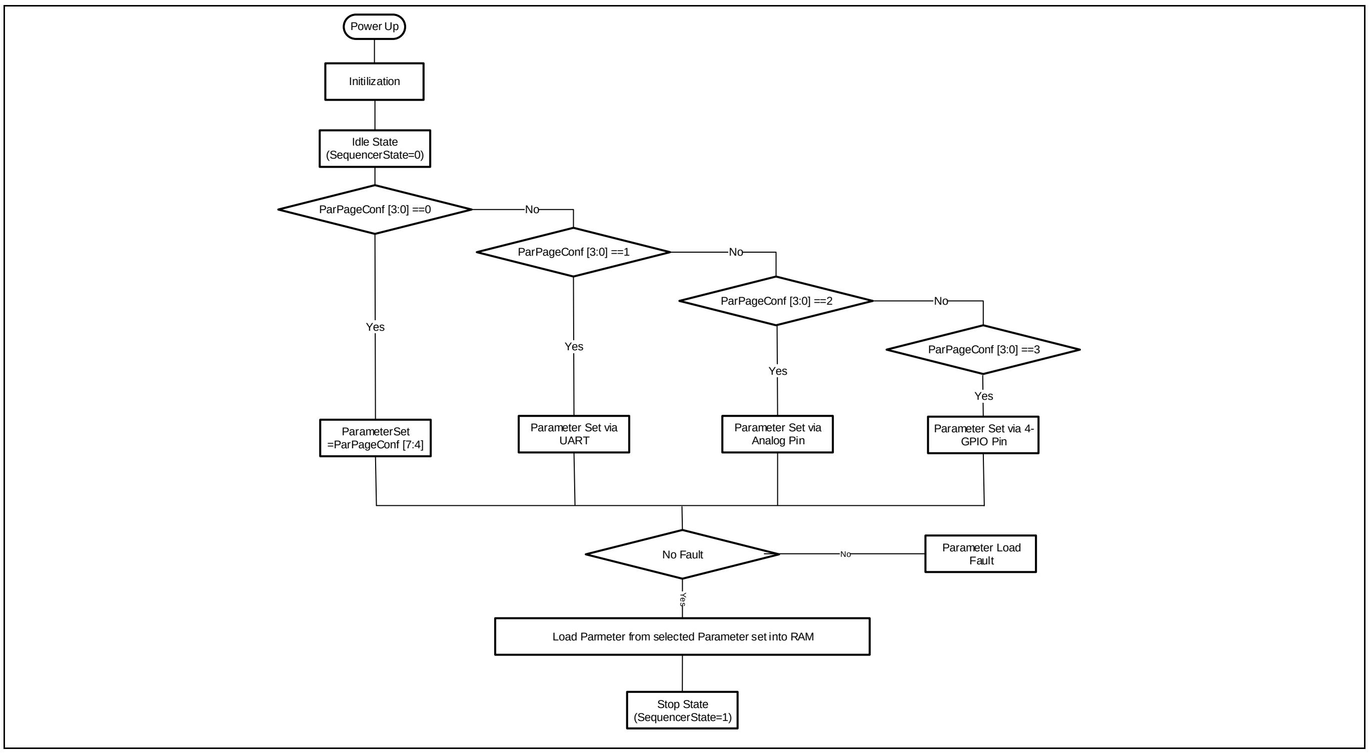

Power Up
Initilization
Idle State (SequencerState=0)
ParPageConf [3:0] ==0
No
ParPageConf [3:0] ==1
No
ParPageConf [3:0] ==2
No
ParPageConf [3:0] ==3
Yes
Yes
Yes
Yes
ParameterSet =ParPageConf [7:4]
Parameter Set via UART
Parameter Set via Analog Pin
Parameter Set via 4-GPIO Pin
No Fault
No
Parameter Load Fault
Yes
Load Parmeter from selected Parameter set into RAM
Stop State (SequencerState=1)

**Figure 88 Parameter Load Procedure**

## 2.5.3 Parameter load fault

If there is no parameter data available in the selected parameter block, MCE stays in IDLE state. It is not possible to start the motor from IDLE state. If there is no valid parameter data is available in the selected parameter block, MCE report parameter load fault and stays in IDLE state. In this condition, it is required to load the right parameter data or select right parameter block.

If there is no other fault, the MCE load parameter values into RAM then go to STOP state and is ready to run the motor.

## 2.6 Script Engine

Script engine is a light-weight "C" language like virtual machine running on the MCE. The script engine enables users to implement system level functionalities beyond motor control and PFC function. Key advantages of script engine are:

- Extend capabilities of manipulating additional digital and analog pins that are not used by motor control and/or PFC.
- Scalable for any future functional extension beyond motor control and PFC.
- Read and write all MCE parameters and variables.
- Multi-tasking capability

### 2.6.1 Overview

Script code follows 'C'-like syntax.  The script engine executes the script code from two different tasks (Task0 and Task1) with different priority. The script engine supports arithmetic, binary logical operators, decision statement (if...else statement) and loop statement (FOR statement).  In the MCE, 16kB of flash memory is reserved for script byte code and constant data. Consequently, the maximum allowed script byte code size is 16kB (Approximately 1.5k lines of code).  256 bytes of data memory is allocated for script global variables and 128 bytes of data memory is allocated for local variables in each task separately.

### 2.6.2 Script Program Structure

The script program consists of the following parts:

- Set Commands: Define script user version and script task execution period.
- Functions:  Script code should be written inside four predefined functions- Script_Task0_init (), Script_Task0 (), Script_Task1_init () and Script_Task1 ().
- Variables, parameters and script methods.
- Statement and Expressions:  Each individual statement must be ended with a semicolon.
- Comments: Starts with a slash asterisk /* and ends with an asterisk slash */ for multiple line comments or prefix double slash // to comment single lines.

### 2.6.3 Script Program Execution

The script engine executes script code from two independent tasks, named Task0 and Task1.  Both the tasks are executed periodically.
Global priority of Script language tasks is lower than that of the MCE embedded tasks such as the FOC, PFC tasks, and others. In the other word CPU computation resource is allocated to the MCE first and then to Script language tasks by utilizing the remaining CPU resource of MCE.  If the embedded MCE function of FOC and PFC utilizes a full amount of CPU loading (i.e. high PWM carrier update for the FOC and/or PFC) in a specific application environment, then Script language tasks have no room for their computation. Therefore, the CPU resource availability is highly dependent on a specific application condition.

Task execution period can be configured using "SCRIPT_TASK0_EXECUTION_PERIOD" and "SCRIPT_TASK1_EXECUTION_PERIOD", for each task.  Each task has separate initialization functions (Script_Taskx_init ( )) to initialize script variables and MCE parameters. Also, it is possible to write script code inside the initialization function. These init functions are called only once during start-up. Task0/Task1 script functions (Script_Taskx()) are called periodically based on task execution period value.

Among script tasks, Task0 has higher priority than Task1.

For Task0, by default, the execution step is 1 and the execution period is 50 (50 x 1 ms = 50 ms). So, Task0 executes one line of script code or script instruction every 1 ms by default, and starts over the execution of the entire script loop every 50 ms. For Task1, by default, the execution step is 10, the execution period is 10 (10 x 10 ms = 100 ms). So Task1 executes 10 lines of script code or script instruction every 10 ms by default, and starts over the execution of the entire loop every 100 ms.

Total script execution time for Task0 or Task1 can be calculated based on number of script instructions in the script code. For example, if the number of script instructions in Task0 is 20, then by default, Task0 takes 20 ms to finish executing the entire script code. No script code is executed in the remaining 30 ms. After 50 ms, Task0 starts to execute the first script instruction again.

Execution step and execution period of each task is configurable. For example, if Task0 execution period is set to 100 ms (SCRIPT_TASK0_EXECUTION_PERIOD =100), then Task0 execution is repeated every 100 ms.

If Task0 execution period is set to 100ms (SCRIPT_TASK0_EXECUTION_PERIOD =100), and number of lines in Task0 is 150, task0 script function takes 150ms to execute the complete script code once. After finishing execution, it immediately starts over again.

#### 2.6.3.1 Execution Time Adjustment

As mentioned, Task0 executes one line of script code or script instruction every 1ms and Task1 executes 10 lines of script code or script instruction for every 10ms by default. It is possible to increase the number of lines executed by Task0 or Task1 per step, to accelerate the script execution.

Number of lines to be executed every 1ms in Task0 can be configured in "SCRIPT_TASK0_EXECUTION_STEP". If Task0 execution period is set to 100ms (SCRIPT_TASK0_EXECUTION_PERIOD =100), Task0 number of lines to be executed every 1ms is set to 2 (SCRIPT_TASK0_EXECUTION_STEP=2) and number of lines in Task0 is 100. Task0 script function takes 50ms to execute the complete script code once.

Similarly, in Task1, number of lines to be executed every 10ms can be configured in "SCRIPT_TASK1_EXECUTION_STEP".

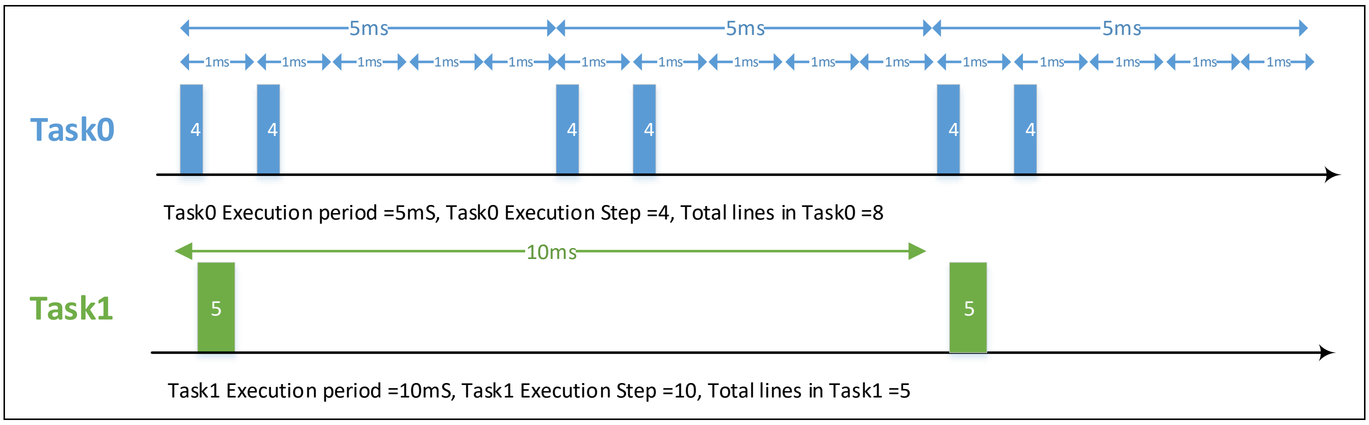

5ms
5ms
5ms
1ms
Task0
4
Task0 Execution period =5mS, Task0 Execution Step =4, Total lines in Task0 =8
10ms
Task1
5
Task1 Execution period =10mS, Task1 Execution Step =10, Total lines in Task1 =5

**Figure 89 Script Task Execution**

### 2.6.4 Constants

Script supports only integer literals in decimal and hexadecimal representation. Hexadecimal value should be prefixed with 0x. Constant value should not have any suffix, for example U or L.

If any variable is assigned with float literals, digits after the decimal place are ignored by the script translator.

Script translator supports up to 100 constant definitions. To define a constant, use descriptor `CONST` or `const` in front of the variable type keyword.

### 2.6.5 Variable types and scope

The script engine supports global and local variables. The global variables can be accessed from both tasks and local variables can only be accessed within the respective task.

The script engine supports the following variable types:

**Table 20 Script Variable Types**

| Type | Storage Size | Value range | Description |
|---|---|---|---|
| uint8_t | 1 byte | 0 to 255 | Byte length unsigned integer |
| int8_t | 1 byte | -128 to 127 | Byte length integer |
| uint16_t | 2 bytes | 0 to 65,535 | Short unsigned integer |
| int16_t | 2 bytes | -32,768 to 32,767 | Short integer |
| int32_t | 4 bytes | -2,147,483,648 to 2,147,483,647 | integer |
| int | 4 bytes | -2,147,483,648 to 2,147,483,647 | integer |

In the MCE, 256 bytes of data memory is allocated for script global variables and 128 bytes of data memory is allocated for local variables in each task.

Script variable name should only consist of alphanumerical characters and underscore symbol (‘_’). Variable name is case-sensitive. All the variable names including global and local should be unique.

Variables declared outside the Task0 or Task1 functions are treated as global variables. Variables declared inside Task0 or Task1 functions are local to Task0 or Task1 respectively.

*Note: Variable cannot be initialized during declaration.*

### 2.6.6 MCE Parameter Access

All MCE parameters and variables can be accessed from script. Parameters and variables can be used directly in the script code without declaration. Only DYNAMIC type parameters and READWRITE type variables are writable from the script code.

A set of parameters and variables can be updated simultaneously using the coherent update method. Two methods (EnableCoherentUpdate () and DoCoherentUpdate ()) are defined in script to do simultaneous update of parameter and variables.

If Coherent update is enabled (by called EnableCoherentUpdate () method), write operation will not update parameters and variables values immediately. Instead, all the values are stored into a buffer and all parameters and variables are updated simultaneously after calling DoCoherentUpdate (). Script supports simultaneous update of up to 32 parameters and variables. (Refer 2.6.10.2)

### 2.6.7 Operators

An operator is a symbol that informs the script to perform a specific mathematical or logical function. A list of operators supported in script function are listed below:

**Table 21 Arithmetic Operators**

| Operator | Description |
|---|---|
| + | Adds two operands. |
| - | Subtracts second operand from the first. |

| Operator | Description |
|---|---|
| * | Multiplies both operands. |
| / | Divides numerator by de-numerator. |
| % | Modulus Operator, remainder after an integer division. |

**Table 22 Binary Operators**

| Operator | Description |
|---|---|
| \| | Binary OR Operator copies a bit to the result if it exists in either operand. |
| & | Binary AND Operator copies a bit to the result if it exists in both operands. |
| ^ | Binary XOR Operator copies a bit to the result if it is set in one operand but not both. |
| ~ | Binary Ones Complement Operator is unary and has the effect of 'flipping' bits. |
| << | Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand. |
| >> | Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand. |

**Table 23 Assignment Operators**

| Operator | Description |
|---|---|
| = | Simple assignment operator. Assigns values from right side operands to left side operand. |

**Table 24 Relational Operators**

| Operator | Description |
|---|---|
| == | Checks if the values of two operands are equal. If yes, then the condition becomes true. |
| != | Checks if the values of two operands are not equal. If yes, then the condition becomes true. |
| > | Checks if the value of left operand is greater than the value of right operand. If yes, then the condition becomes true. |
| >= | Checks if the value of left operand is greater than or equal to the value of right operand. If yes, then the condition becomes true. |
| < | Checks if the value of left operand is less than the value of right operand. If yes, then the condition becomes true. |
| <= | Checks if the value of left operand is less than or equal to the value of right operand. If yes, then the condition becomes true. |

**Table 25 Logical Operators**

| Operator | Description |
|---|---|
| && | Logical AND operator used to combine two or more conditions. Operator returns true when both the conditions in consideration are satisfied. Otherwise it returns false. |
| \|\| | Logical OR Operator used to combine two or more conditions. Operator returns true when any one of the conditions in consideration are satisfied. Otherwise it returns false. |

The precedence and associativity of all the operators in script languages are summarized in below table.

**Table 26 Script Operator Precedence**

| Precedence | Operator | Description | Associativity |
|---|---|---|---|
| 8<br>Highest | ~<br>- | Bitwise NOT (One's Complement)<br>Unary minus | Right to left |
| 7 | *<br>/<br>% | Multiplication<br>Division<br>Modulo (remainder) | Left to right |
| 6 | +<br>- | Addition<br>Subtraction | Left to right |
| 5 | <<<br>>> | Bitwise left shift<br>Bitwise right shift | Left to right |
| 4 | <<br><=<br>><br>>= | Less than<br>Less than or equal to<br>Greater than<br>Greater than or equal to | Left to right |
| 3 | ==<br>!= | Equal to<br>Not equal to | Left to right |
| 2 | &<br>\|<br>^ | Bitwise AND<br>Bitwise OR<br>Bitwise XOR (exclusive or) | Left to right |
| 1<br>Lowest | &&<br>\|\| | Logical AND<br>Logical OR | Left to right |

The order of precedence can be overridden by using parentheses. Simply enclose within a set of parentheses the part of the equation that needs to be executed first.

## 2.6.8 Decision Structures

Decision structures are used for branching. The script engine provides if-statement for decision making. If statements can be followed by an optional else statement, which executes when the Boolean expression is false. Boolean expression can consist of relational operator and logical operators. Syntax of if…else statement in script language is shown below:

```
if(boolean_expression)
{
  /*Statement(s) will execute if the expression is true*/
}
else
{
  /*Statement(s) will execute if the expression is false*/
}

If and else statement should be followed by curly braces
```

**Code Listing 1 If...else statement syntax**

Script programming assumes any non-zero and non-null values as true, and if it is either zero or null, then it is assumed as false value. Depth of nested if condition is limited to 15.

### 2.6.9 Loop Structures

The MCE supports FOR-statements for repeat processes. Syntax of FOR statement in script language is shown below:

```
for(<ScriptVariable> = <Startvalue> : <Endvalue>)
{
  /*Statement(s) will execute for defined loop time*/
}
```

**Code Listing 2 for statement syntax**

Statements inside the for loop are executed for Endvalue - Startvalue+1 time. The FOR statement does not support count down mode (decreasing index). The start value must always be less than end value.

### 2.6.10 Methods

Predefined methods are available for specific operations. Methods supported in script functions are described in the following sections.

#### 2.6.10.1 Bit access Methods

Three methods are defined in the script to read or write particular bit of script variables or motor control/PFC related variables or parameters.

**Table 27 Bit Access Methods**

| Methods | Description |
|---|---|
| void SET_BIT(<Var>, <bitposition>) | Set the particular bit of variable. |
| void CLEAR_BIT(<Var>, <bitposition>) | Clear the particular bit of variable. |
| uint32_t GET_BIT(<Var>, <bitposition>) | Read the particular bit of variable. |

*Note: Bit position value must be 0 to 15.*

#### 2.6.10.2 Coherent update methods

These methods are used for updating motor control and/or PFC parameters and variables simultaneously.

**Table 28 Coherent Methods**

| Methods | Description |
|---|---|
| EnableCoherentUpdate() | Enable simultaneous update of parameter/variables. |
| DoCoherentUpdate() | Trigger simultaneous update of parameter/variables. |

*Note: Maximum 32 parameters/variables can be updated simultaneously.*

When a coherent update is enabled, values are not updated into parameter/variables immediately. Instead values are stored into buffer and update the actual variable/parameter after trigger the coherent update.

### 2.6.10.3 Configurable UART API

Table 29 Configurable UART API

| API name | Brief description |
|---|---|
| UART_DriverInit() | Initializes the UART hardware driver. |
| UART_DriverDeinit() | De-initializes the UART hardware driver. |
| UART_FifoInit() | Initialize UART hardware FIFO. |
| UART_BufferInit() | Initialize UART software buffer. |
| UART_GetStatus() | Get the status word for the UART communication status. |
| UART_GetRxDelay() | Returns the delay time between receive frames. |
| UART_Control() | Writes to the Control Word that defines UART control commands. |
| UART_RxFifo() | Returns one byte from the receive FIFO. |
| UART_TxFifo() | Puts one byte to the transmit FIFO. |
| UART_RxBuffer() | Returns one byte from the receive buffer from a specified location. |
| UART_TxBuffer() | Puts one byte in the transmit buffer at a specified location. |

#### 2.6.10.3.1 UART_DriverInit()

Declaration:

```
void UART_DriverInit(channel, rxInvert, txInvert, baudrate, dataBits, stopBits)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| channel | 0 | 1 | Selects which UART channel to be used.<br>0: UART 0<br>1: UART 1 |
| rxInvert | 0 | 1 | Configures the data interpretation logic for the received data.<br>0: non-inverting<br>1: inverting |
| txInvert | 0 | 1 | Configures the data interpretation logic for the transmitted data.<br>0: non-inverting<br>1: inverting |
| baudrate | 600 bps | 115,200 bps<br>(230,400 bps in FIFO mode) | Configures the baudrate for the UART in bits-per-second. |
| dataBits | 5 bits | 8 bits | Configures the length of the data bits in a UART byte. |
| stopBits | 1 bit | 2 bits | Configures the number of stop bits in a UART byte. |

Description:

This API initializes the UART driver.

#### 2.6.10.3.2 UART_DriverDeinit()

Declaration:

```
void UART_DriverDeinit(void)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return type | Description |
|---|---|
| N/A | N/A |

Description:

This API deinitializes the UART driver.

#### 2.6.10.3.3 UART_FifoInit()

Declaration:

```
void UART_FifoInit(rxFifoSize, txFifoSize)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| rxFifoSize | 1 byte | 31 bytes | Size of the FIFO buffer allotted for receive in bytes. |
| txFifoSize | 1 byte | 31 bytes | Size of the FIFO buffer allotted for transmit in bytes. |

Description:

This API initializes the UART FIFO.

#### 2.6.10.3.4 UART_BufferInit()

Declaration:

```
 void UART_BufferInit(halfDuplex, rxTimeout, txDelay, txByteDelay, rxFlag, txFlag,
rxDataLength, txDataLength)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| halfDuplex | 0 | 1 | Configure the UART buffer for half or full duplex communication.<br>0: Full duplex<br>1: Half duplex |
| rxTimeout | 0 | 65535 | Configure the longest expected time to receive a frame. If a frame is not received within this time an RxTimeout will occur. |
| txDelay | 0 | 65535 | Configure the delay time from having received a frame and starting to transmit a frame in ms. |

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| txByteDelay | 0 | 65535 | Configure the delay time between each byte in a transmit frame. |
| rxFlag | 0 | 65535 | rxFlag is a byte that signifies the beginning of a receive frame.<br>0-255: valid flag byte<br>256-65535: invalid flag / no flag byte is used |
| txFlag | 0 | 65535 | txFlag is a byte that signifies the beginning of a transmit frame.<br>0-255: valid flag byte<br>256-65535: invalid flag / no flag byte is used |
| rxDataLength | 1 byte | 8 bytes | Configure the length of the receive frame, in bytes, not including the start flag byte. |
| txDataLength | 1 byte | 8 bytes | Configure the length of the transmit frame, in bytes, not including the start flag byte. |

Description:

This API configures the UART software buffer.

#### 2.6.10.3.5 UART_GetStatus()

Declaration:

```
uint32_t UART_GetStatus(void)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | Returns the status word whose bitfield representation is described below.<br>**FIFO status:**<br>Bit 0 – **IsRxFIFOEmpty:** is receive FIFO empty bit<br>0: receive FIFO is not empty<br>1: receive FIFO is empty<br>Bit 1 – **IsRxFIFOFull:** is receive FIFO full bit<br>0: receive FIFO is not full<br>1: receive FIFO is full<br>Bit 2 – **IsTxFIFOEmpty:** is transmit FIFO empty bit<br>0: transmit FIFO is not empty<br>1: transmit FIFO is empty<br>Bit 3 – **IsTxFIFOFull:** is transmit FIFO full bit<br>0: transmit FIFO is not full<br>1: transmit FIFO is full<br>Bit 4:7- **reserved:** read as ‘0’<br><br>**Buffer status:**<br>Bit 8 – **IsRxBufferFull:** is receive buffer full bit<br>0: receive buffer is not full<br>1: receive buffer is full |

| Return Type | Description |
|---|---|
| | Bit 9 – **reserved:** read as '0'<br>Bit 10 –**IsTxBufferEmpty:** is transmit buffer empty bit<br>0: transmit buffer is not empty<br>1: transmit buffer is empty<br>Bit 11:14 – **reserved:** read as '0'<br>Bit 10 –**IsBufferMode:** is Buffer Mode Initialized bit<br>0: the frame buffer and the driver handler is not initialized<br>1: the frame buffer and the driver handler is initialized<br><br>**Handler status:**<br>Bit 16 – **IsRxTimeout:** is receive frame timeout bit<br>0: receive frame is not timed out<br>1: receive frame is timed out<br>Bit 17 – **IsCollision:** is collision detected bit<br>0: collision is not detected<br>1: collision is detected<br>Bit 18:19 – **HandlerState:** Handler state bitfield<br>00: FRAME_START<br>01: FRAME_RECEIVE<br>10: FRAME_DELAY<br>11: FRAME_TRANSMIT<br>Bit 20:22- **reserved:** read as '0'<br>Bit 23 – **IsHalfDuplex:** is half duplex bit<br>0: the driver handler is not initialized in half-duplex mode<br>1: the driver handler is initialized in half-duplex mode<br><br>**Driver status:**<br>Bit 24 – **IsRxNoiseDetected:** is receive noise detected bit<br>0: noise on the receive line has not been detected<br>1: noise on the receive line has been detected<br>Bit 25 – **IsParityError:** is parity error bit<br>0: a parity error has not occurred<br>1: a parity error has occurred<br>Bit 26 – **IsStopBitError:** is stop bit error<br>0: a stop bit error has not occurred.<br>1: a stop bit error has occurred<br>Bit 27:30 – **reserved:** read as '0'<br>Bit 31 – **IsInitialized:** is initialized bit<br>0: the driver is not initialized.<br>1: the driver handler is initialized. |

Description:

This API returns the status word.

#### 2.6.10.3.6 UART_GetRxDelay()

Declaration:

```
uint32_t UART_GetRxDelay(void)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return type | Description |
|---|---|
| uint32_t | Returns the time, in ms, between receive frames. Timing begins from the last byte of the current receive frame and ends at the first byte of the next receive frame. |

Description:

This API returns the delay, in ms, between receive frames.

#### 2.6.10.3.7 UART_Control()

Declaration:

```
void UART_Control(command)
```

| Input Parameters | Description |
|---|---|
| command | Writes the Control Word whose bit field representation is described below.<br>**FIFO control:**<br>Bit 0 – **reserved:**<br>Bit 1 – **ClrRxFIFO:** Clear RX FIFO bit<br>0: N/A<br>1: clear the receive FIFO<br>Bit 2 – **reserved:**<br>Bit 3 – **ClrTxFIFO:** Clear TX FIFO bit<br>0: N/A<br>1: clear transmit FIFO<br>Bit 4:7 – **reserved:**<br><br>**Buffer control:**<br>Bit 8 – **ClrRxBufferFlag:** Clear RX Buffer flag bit<br>0: N/A<br>1: clear receive buffer flag<br>Bit 9 – **reserved:**<br>Bit 10 –**SendTxBuffer:** Send TX Buffer flag<br>0: N/A<br>1: Initiate the sending of bytes from the transmit buffer through the specified UART channel.<br>Bit 11:15 – **reserved:**<br><br>**Handler control:**<br>Bit 16 – **ClrRxTimeoutFlag:** Clear RX time-out Flag bit |

| Input Parameters | Description |
|---|---|
| | 0: N/A<br>1: clear receive time-out flag<br>Bit 17 – **ClrCollisionFlag:** Clear Collision detected Flag bit<br>0: N/A<br>1: clear collision detection flag<br>Bit 18 – **RstBufferControl: Reset Buffer Control bit**<br>0: N/A<br>1: reset buffer control state machine<br>Bit 19:23 – **reserved:**<br><br>**Driver Control:**<br>Bit 24 – **ClrRxNoiseFlag:** Clear RX Noise Flag bit<br>0: N/A<br>1: clear the receive noise flag<br>Bit 25 – **ClrParityErrorFlag:** Clear Parity Error Flag bit<br>0: N/A<br>1: clear the parity error flag.<br>Bit 26 – **ClrStopbitErrorFlag:** Cleare Stop bit Error Flag bit<br>0: N/A<br>1: clear the stop bit error flag<br>Bit 27:31 – **reserved:** |

Description:

This API controls the UART's buffer mode, FIFO mode, driver control, and handler control.

#### 2.6.10.3.8 UART_RxFifo()

Declaration:

```
uint32_t UART_RxFifo(void)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | Returns one byte from the receive FIFO. |

Description:

This API returns one byte of data from the receive FIFO in First In First Out order.

#### 2.6.10.3.9 UART_TxFifo()

Declaration:

```
void UART_TxFifo(data)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| data | 0 | 255 | One byte of data placed in the transmit FIFO. |

Description:

This API pushes data into the transmit FIFO in First In First Out order.

#### 2.6.10.3.10 UART_RxBuffer()

Declaration:

```
uint32_t UART_RxBuffer(uint32_t idx)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| idx | 0 | 7 | Specifies which byte of data in the receive buffer to return. |

| Return Type | Description |
|---|---|
| uint32_t | Returns one byte from the receive buffer specified by idx. |

Description:

This API returns one byte of data from the receive buffer. In buffer mode one can select which byte of data to be returned by specifying the byte using idx.

#### 2.6.10.3.11 UART_TxBuffer()

Declaration:

```
void UART_TxBuffer(idx, data)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| idx | 0 | 7 | Specifies at which index to place one byte of data in the transmit buffer. |
| data | 0 | 255 | One byte of data to be placed at index idx in the transmit buffer. |

Description:

This places one byte of data into the transmit buffer. In Buffer mode one can place the byte of data anywhere in the buffer specified by idx.

### 2.6.10.4 User GPIOs

The Script enables access to digital pins and analog inputs not used by motor control and PFC. Read and write of digital pins is supported and read of analog inputs are supported.

#### 2.6.10.4.1 Digital Input and Output Pins

Digital pins available to users can be configured as input or output pins. All configured digital input/output pins values are read/write by the script every 1 ms.

Four dedicated variables are defined in the MCE to read or write digital input/output pins.

| Variable Name | Type | Description |
|---|---|---|
| FB_GPIO.GPIO_Status | READONLY | Holds digital input/output (GPIO0 to GPIO29) pin values. |
| FB_GPIO.GPIO_Set | READWRITE | Sets or resets digital output pin (GPIO0 to GPIO29) |

The logic level of a GPIO pin can be read via the read–only registers “FB_GPIO.GPIO_Status”. Read “FB_GPIO.GPIO_Status” register always returns the current logical value the GPIO pin, regardless of the pin direction (input or output). It is possible to read the complete variable or binary data.

“FB_GPIO.GPIO_Set” register determines the value of a digital pin when it is configured as output. Writing a 0 to a bit position delivers a low level at the corresponding output pin. Likewise, writing a 1 to a bit position delivers a high level at the corresponding output pin. . It is possible to read the complete variable or binary data.

#### 2.6.10.4.2 Analog pins

Analog pins available to the user are read by MCE every 1 ms. The result value is accessible to the script code.

12 dedicated variables are defined in the MCE to read analog input pins value.

| Variable Name | Type | Description |
|---|---|---|
| FB_ADC.adc_result0 | READONLY | Holds AIN0 analog input value (12 bit value) |
| FB_ADC.adc_result1 | READONLY | Holds AIN1 analog input value (12 bit value) |
| FB_ADC.adc_result2 | READONLY | Holds AIN2 analog input value (12 bit value) |
| FB_ADC.adc_result3 | READONLY | Holds AIN3 analog input value (12 bit value) |
| FB_ADC.adc_result4 | READONLY | Holds AIN4 analog input value (12 bit value) |
| FB_ADC.adc_result5 | READONLY | Holds AIN5 analog input value (12 bit value) |
| FB_ADC.adc_result6 | READONLY | Holds AIN6 analog input value (12 bit value) |
| FB_ADC.adc_result7 | READONLY | Holds AIN7 analog input value (12 bit value) |
| FB_ADC.adc_result8 | READONLY | Holds AIN8 analog input value (12 bit value) |
| FB_ADC.adc_result9 | READONLY | Holds AIN9 analog input value (12 bit value) |
| FB_ADC.adc_result0 | READONLY | Holds AIN10 analog input value (12 bit value) |
| FB_ADC.adc_result11 | READONLY | Holds AIN11 analog input value (12 bit value) |

### 2.6.10.5 Infrared Interface

The MCE firmware includes an IR interface that consists of a plug-in of the scripting engine and script APIs. This allows IR signals to be interpreted directly from an IR sensor, as long as the transmitter’s protocol is supported. MCE parameters can be changed based on the IR command transmitted via scripting within the Solution Designer. This can, for example, be used for setting the motor speed based on the press of a remote’s button, or customized for setting of MCE parameters. This can be done by creating a simple script, then connecting an IR sensor to the chosen device input pin.

###### 2.6.10.5.1 Infrared Protocols

The IR Interface supports the following protocols: NEC, NEC Extended, RC5 Phillips. The protocols are characterized by:

| Protocol | Number of bits | Order of transmitted parts | Length of each transmission | Carrier Frequency |
|---|---|---|---|---|
| NEC | 32 | Address, inverted address, command, inverted command | 67.5 ms | 38.222kHz |
| NEC extended | 32 | Address low 8 bits, address high 8 bits, command, inverted command | 67.5 ms | 38.222kHz |
| Phillips RC5 | 12 | 1 toggle bit + 5 address bits + 6 command bits | 24.892 ms | 36.0kHz |

NEC, NEC Extended, and RC-5 operate with 'regular data frames' for transmission of commands and with 'repeat frames' for transmission of a repeated command. Both types of frames are supported by the IR script plug-in.

###### 2.6.10.5.2 IR Pins

The MCE supports IR data on 3 different pins: VSP, RXD0 and RXD1. Not all options are available on all devices. The user must select one of these when utilizing the IR interface. The pins are enabled and assigned IR-function through the API IR_DriverInit(). When using the IR interface, do not enable the IR-pin (VSP/RXD0/RXD1) anywhere else, including in the Solution Designer.

###### 2.6.10.5.1 Infrared Interface APIs

The APIs of the Infrared Interface plug-in are summarized in the table below.

| API name | Brief description |
|---|---|
| IR_DriverInit() | Initializes IR Driver based on key parameters |
| IR_DriverDeinit() | De-initializes IR Driver |
| IR_RxBuffer() | Returns most recent transmission |
| IR_GetStatus() | Returns status |
| IR_RxCommand() | Returns "Command" section of transmission |
| IR_RxAddress() | Returns "Address" section of transmission |
| IR_RxRepeats() | Returns numbers of transmissions repeated |
| IR_RxReceived() | Returns true if transmission has been received |
| IR_RxRepeating() | Returns true if transmission has not been fully received |

### 2.6.10.5.2 IR_DriverInit()

Declaration:

```
uint32_t IR_DriverInit(channel, rxInvert, protocol, address)
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| `channel` | 0 | 2 | Specifies the channel (pin) to received IR data. Availability of pins depends on the device.<br><br>0: RX0<br>1: RX1<br>2: VSP |
| `rxInvert` | 0 | 1 | Indicates if the IR receiver sensor is sending the signal inverted or non-inverter.<br><br>0: Disable invert<br>1: Enable Invert |
| `protocol` | 0 | 2 | Selects the IR protocol<br><br>0: RC-5<br>1: NEC<br>2: NEC Extended |
| `address` | 0 | 65535 | Configure the address to use. Must match transmitter (remote) address. Range varies based on protocol |

| Return Type | Description |
|---|---|
| uint32_t | Initialization status.<br><br>0 – **Driver successfully initialized**<br>1 – **IR driver not available**<br>2 – **Protocol recognized**<br>3 - **Address out of range** |

Description:

Initializes driver and related peripherals.

#### 2.6.10.5.1 IR_DriverDeinit()

Declaration:

```
uint32_t IR_DriverDeinit()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | Returns success/failure of API execution<br><br>0 – **Driver successfully de-initialized**<br>1 - **IR driver was available** |

Description:

De-initializes driver and peripherals.

#### 2.6.10.5.1 IR_RxBuffer()

Declaration:

```
uint32_t IR_RxBuffer()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | Returns raw data buffer. Data are organized according to protocol.<br><br>**Phillips RC-5**<br>Bit 0 – toggle bit<br>Bit 1:5 – address<br>Bit 6:11 – command<br><br>**NEC**<br>Bit 0:7 – address<br>Bit 8:15 – $\overline{\text{address}}$<br>Bit 16:23 – command<br>Bit 24:31 – $\overline{\text{command}}$<br><br>**NEC extended**<br>Bit 0:15 – address<br>Bit 16:23 – command<br>Bit 24:31 – $\overline{\text{command}}$ |

Description:

Returns all transmitted data, and sets IR_RxReceived() return value to 0. Clears bit 2 and 3 of IR_GetStatus() return value.

### 2.6.10.5.1 IR_GetStatus()

Declaration:

```
uint32_t IR_GetStatus()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | IR interpreted signal status<br><br>Bit 0:1 – **ProtocolSelected:**<br>0: Phillips RC5<br>1: NEC<br>2: NEC Extended<br>Bit 2 – **IsRawDataAvailable:**<br>0: no data received<br>1: data received<br>Bit 3 – **IsRawDataValid:**<br>0: invalid IR signal received or received address does not match configured address<br>1: valid IR signal received and received address matches configured address<br>Bit 4 – **IsDataAvailable:**<br>0: invalid IR signal received or received address does not match address<br>1: valid IR signal received and received address matches configured address<br>Bit 5 – **IsReceiving:**<br>0: waiting for IR transmission<br>1: target is currently receiving data<br>Bit 6 – reserved<br>**IR Error bits**<br>Bit 7 – **IsAddressIncorrect:**<br>0: no mismatch<br>1: received address doesn’t match address defined in IR_DriverInit()<br>Bit 8:30 – **reserved**<br><br>**Driver Status**<br>Bit 31 – **IsInitialized:**<br>0: driver not initialized<br>1: driver Initialized |

Description:

Returns status of IR driver

### 2.6.10.5.1 IR_RxCommand()

Declaration:

```
uint32_t IR_RxCommand()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | The API returns the command of a full frame when address and protocol match configuration. On repeat codes, the last valid command is kept as return value. |

Description:

Returns command section of transmission as long as the protocol and address match configuration. IR_RxReceived() will return 0 after calling this API.

#### 2.6.10.5.1 IR_RxAddress ()

Declaration:

```
uint32_t IR_RxAddress()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | The API returns the address of a full frame when address and protocol match configuration. The API returns 0 on repeat codes. |

Description:

Returns address section of IR transmission.

#### 2.6.10.5.1 IR_RxRepeats ()

Declaration:

```
uint32_t IR_RxRepeats()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | Contains the number of transmissions repeated after IR remote’s button has been released. If target is currently receiving, 0 is returned. |

Description:

Returns the number of transmissions repeated after IR remote’s button has been released. This API allows for script to respond according to how long a button is pressed, such as holding an “increase” button.

#### 2.6.10.5.1 IR_RxReceived ()

Declaration:

```
uint32_t IR_RxReceived()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | Bit 0 – **Receive status**<br>0: IR message frame has not been received<br>1: IR message frame has been received |

Description:

Returns 1 after the first frame of a new transmission is received. Calling IR_RxCommand API automatically clears the return value of this API.

#### 2.6.10.5.1 IR_RxRepeating ()

Declaration:

```
uint32_t IR_RxRepeating()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | Bit 0 – **Repeat status**<br>0: Transmission is not repeating (has stopped).<br>1: Transmission is repeating (button is held down) |

Description:

Indicates a command is currently repeating via a button being held down. Returns 0 when the transmission stops.

#### 2.6.10.5.2 Timing Considerations

The IR buffer update period is 50ms and it is recommended the script checks for new data at a rate similar or faster than this. This is to ensure no data are lost in case where commands are transmitted at a high rate. Practical limitations, such as the implausibility of pushing a remote button every 50ms, may enable lower update rates.

#### 2.6.10.5.1 Configuration

In order to configure the IR interface, scripting shall be utilized. The process of programming a script to a device is described in the app note "How to use iMOTION™ Script Language". The first step in configuring the IR interface is to use the IR_Driverinit(), an API that initializes the IR driver. This API allows you to choose which device input pin the IR receiver sensor will be connected to, which IR protocol will be read, and what address will be sent by the transmitter. After executing this API intrepered commands sent by the transmitter are read with IR_RxCommand() and by using conditional statements.

### 2.6.10.6 Flash Data Storage

The MCE firmware includes an interface that allows the user access to the embedded flash of the device. The user is allowed access to 160 bytes of storage in which script variables can be persistent stored. This interface consists of a plug-in to the script engine, providing the user a set of APIs for easy use. Therefore, this interface is compatible with any iMOTION device that supports the scripting feature.

#### 2.6.10.6.1 Flash Data Type

Script variables declared with the keyword 'flash' can be stored in flash upon request from the user. The script variables will be initialized by stored flash value at initialization. All variable types, supported by scripting, can be declared as flash variables and both local and global types are supported. To declare a flash variable, use the syntax '`flash varType varName`', for example:

```
flash uint8_t FlashVar1;
flash uint16_t FlashVar2;
flash int32_t FlashVar3;
```

This specifies that this variable will be stored in flash after the Flash_Write() API is called. If a variable with the same name has already been stored, the stored value will be assigned to this variable.

Up to 160 bytes of flash variables are supported in any combination of script variable types. For example, 160 8-bit size variables, 80 32-bit size variables, 40 32-bit size variables, or any size mix of variables.

When flash is empty, a variable of type 'flash' will be initialized to 0 after reset. When flash data is invalid, no content will be loaded from flash. It is up to the user to handle the correct initialization of variables in these situations.

During execution of the script, the user can write to flash variables at any point. However, the content of that variable will not be committed to flash until the user calls the API Flash_Write(). Only then, the variable is stored in flash. It is up to the user to decide when an appropriate time to commit to flash is. At initialization, the variable is assigned the value last committed to flash.

#### 2.6.10.6.1 Limitations of Flash

Flash storage differs from EEPROM in some ways that are crucial to be aware of. Compared to flash, EEPROM is faster to write/read/erase, and can be written to more times over its lifetime. These limitations need to be kept in mind.

#### 2.6.10.6.2 Flash Data Storage APIs

The APIs of the Flash Data Storage plug-in are summarized in the table below.

| API name | Brief description |
|---|---|
| Flash_Write() | Writes all "flash" type variables to flash |
| Flash_Erase() | Erases all data in allocated storage |
| Flash_GetWriteCount() | Returns amount of times flash has been written over lifetime |
| Flash_GetStatus() | Returns status from flash driver |

#### 2.6.10.6.3 Flash_Write()

Declaration:

```
uint32_t Flash_Write()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | Bit 0 – **Write status**<br>0: Write success<br>1: Write failure |

Description:

Writes all "flash" type variables to flash and returns number based on success or failure to write. NOTE: motor/PFC must be stopped before calling.

#### 2.6.10.6.4 Flash_Erase()

Declaration:

```
uint32_t Flash_Erase()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|
| uint32_t | Bit 0 – **Erase status**<br>0: Erase success<br>1: Erase failure |

Description:

Erases all data in storage allocated for data storage interface. Returns status based on success of erase cycle. Note Flash_Erase() erases content of the flash data storage only. Variables of type flash are initialized after reset and the erase value will not be assigned to the flash variables until next initialization.

#### 2.6.10.6.5 Flash_WriteCount()

Declaration:

```
uint32_t Flash_WriteCount()
```

| Input Parameters | Min | Max | Description |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| Return Type | Description |
|---|---|

| | |
|---|---|
| uint32_t | Bit 0-31 – **Number of writes** |

Description:

Returns the number of times flash has been written. Write Counter is cleared when flash is erased.

#### 2.6.10.6.6 Flash_GetStatus()

Declaration:

```
uint32_t Flash_GetStatus(void)
```

| **Input Parameters** | **Min** | **Max** | **Description** |
|---|---|---|---|
| N/A | N/A | N/A | N/A |

| **Return Type** | **Description** |
|---|---|
| uint32_t | Bit 0 – **FlashErased:** Indicates if flash has been erased<br>0: data in flash<br>1: flash has been erased<br>Bit 1 – **FlashInvalid:** Indicates if flash is corrupt<br>0: flash content is valid<br>1: flash is corrupt<br>Bit 2 – **isFlashWriteError:** Indicates if flash failed to write<br>0: no error exists<br>1: flash failed to write<br>Bit 3:31 – **reserved** |

Description:

Returns status message from Flash Data Storage driver. Status of Flash that has never been used or flash that has been erased will show that 'Flash has been erased' (bit 0). If checksum of stored data does not match content, status will show 'Flash content is not valid' (bit 1). It is up to the user to initialize flash variable appropriately when flash is empty or not valid.

#### 2.6.10.6.7 Timing Considerations

The APIs Flash_Write and Flash_Erase() are blocking meaning all other tasks and functions will not be serviced while performing flash write/erase. Calling these APIs must be done at non-critical times and tightly synchronized to the system state. Before calling Flash_Write and Flash_Erase(), it is up to the user to make sure the motor and PFC (if supported) are in a stopped and passive state. Therefore, the user can only use the data storage interface to write and erase while the motor is stopped-, fault or standby state.

Flash write time is determined by the carateristics of the embedded flash and of how many variables are written. Maximum write time of flash wariables is 8.6ms.

The erase process takes 7.0ms and leaves the content of flash initialized to 0xF's. When flash is empty, variable of type 'flash' will be initialized to 0 after reset.

#### 2.6.10.6.8 Endurance Considerations

The Flash Data Storage utilizes the MCE's embedded flash which has less endurance than traditional EEPROM. Maximum numbers of writes supported is 50000 and it is up to be user to keep track of number of write cycles using the API Flash_WriteCount() and make sure the endurance of the flash is not exceeded.

## 2.7 Internal Oscillator Calibration with Respect to Temperature

### 2.7.1 Overview

The internal oscillator frequency of MCE varies as the temperature changes. The accuracy of the internal oscillator can be improved by a calibration process with respect to temperature changes. The MCE implements a run-time calibration routine that measures the die temperature using its on-chip temperature sensor, and applies an offset value to adjust the internal oscillator accordingly to achieve higher accuracy. This calibration routine is executed every 20 ms automatically.

This internal oscillator calibration function can be enabled by setting the 3rd bit of 'SysTaskConfig' parameter. See respective product datasheets for detailed specification of achievable accuracy.

## 2.8 Low Power Standby

The MCE provides two types of standby modes, CPU Idle Sleep and Low Power, that enabled reduced power consumption of the MCE when the motor/PFC are not operating. These two modes can be used together or independently. CPU Idle Sleep puts the CPU into sleep mode when there are no tasks to execute. Power reduction depends on the MCE idle time (CPU load) and power reduction is limited. Low Power Mode achieves more power reduction by reducing the CPU clock and switching off some of the internal controller peripherals. In Low Power mode, the Motor PWM unit and the PFC PWM unit (if supported) are disabled. In Low Power mode, VDC is measured every 1ms. To detect a critical over voltage, over voltage, or under voltage fault two consecutive VDC measurments must be above the over voltage threshold for a fault to be triggered, giving a total response time of 2ms. The measured VDC is not filtered and the raw value is used for the voltage protection functions.

The highest power saving is achieved when enabling both CPU Idle Sleep and Low Power Mode simultaneously. The Bitfields of the parameter 'StandbyConf' enables/disables the standby modes.

Available features depend on the standby mode, where CPU Idle Sleep generally supports all available features and Low Power Mode only supports a subset. The table below gives a general overview of the features supported in each mode.

| **Group** | **Protection Fault** | **CPU Idle Sleep** | **Low Power Mode** |
|---|---|---|---|
| Interface | iSD DashBoard | x | x |
| | iSD Oscilloscope | x | x |
| | JCOM | x | x |
| | USER UART | x | x |
| | Digital/Analog Hall | x | |
| | PFC | x | |
| Control Input | VSP | x | x |
| | DUTY | x | x |
| | FREQ | x | x |
| Scripting | Scripting | x | x |
| | IR Interface | x | x |
| | Data Storage | x | x |

Available protection functions also depend on the standby mode. With CPU Idle Sleep all protection functions are supported and with Low Power Mode only a subset of protection functions is supported. The table below gives a general overview of the functions supported in each mode.

| Group | Protection Fault | CPU Idle Sleep | Low Power Mode |
|---|---|---|---|
| Motor | Critical Over Voltage | x | x |
| | Over Voltage | x | x |
| | Under Voltage | x | x |
| | Gate Kill | x | |
| | Gate Kill Pin | x | |
| | Flux | x | |
| | Over Temperature | x | x |
| | Rotor Lock | x | |
| | Phase Loss | x | |
| | Current Offset | x | |
| PFC | Over Current (OCP) | x | |
| | VAC Drop-out | x | |
| | VAC Overvoltage | x | |
| | VAC Brown-out | x | |
| | VAC Frequency | x | |
| | Vout Overvoltage | x | |
| | Vout Open-Loop | x | |
| | Current Meas. Offset | x | |
| System | UART link break | x | x |
| | CPU Execution | x | x |
| | Parameter Load | x | x |

### 2.8.1 Entry and Exit Conditions

If enabled, CPU Idle Sleep is entered as soon as the MCE is in an execution idle state. No other conditions need to be met. CPU Idle Sleep is exited when there are tasks to execute.

Before entering to Low Power mode, a configurable delay time must expire and specific conditions must be met. During this time the system remains fully active and can cancel entry to Low Power mode by running the motor or triggering a fault. The delay time is configured by the 'StandbyPauseTimeout' parameter with a maximum of 65535 milliseconds. This parameter cannot be modified at runtime from script.

Following are the conditions for entering and exiting to/from Low Power mode for MOTOR and PFC (if supported):

**Entry Conditions:**

1. Motor is in STOP state.
2. PFC is disabled (if supported).
3. No motor faults are present.
4. Pause before entering low power mode has expired.

**Exit Conditions:**

1. Motor start command has been received.
2. PFC is enabled (if supported).

3. Fault has occurred.

A motor start command or a fault is needed to wakeup the MCE from Low Power Mode. The start command can come from any of the active sources which include Control Input (VSP/FREQ/DUTY), scripting, JCOM, User UART or Solution Designer.

## 2.8.2 Scripting

Script functionality is fully supported with instructions limitations per Execution Step and a minimum a task period of 1 millisecond (Task 0).

- Up to 40 instructions when using Motor and no PFC
- Up to 20 instruction when Motor and PFC

Both Motor_SequencerState = 13 and PFC_SequencerState = 7 (if supported) will indicate that MOTOR and PFC are in Low Power Mode. For more information on state handling during standby, refer to section 2.1.2 and 0.

### 2.8.2.1 Timing

Entering Low Power Mode takes Overhead Time + Pause Before Entering to Low Power + 1 millisecond state transition in the worst case. Average Overhead Time is 106 microseconds, corresponding to ramp-down process and stand-by request.

Exiting from STAND-BY mode takes Overhead Time + 1 ms state transition in the worst case. Average Overhead Time is 34.24 microseconds where the system reconfigures what was disabled in the ramp down process.

# 3 Motor Tuning

Solution Designer calculates hardware parameters, motor parameters, control parameters/features, protection parameters/features as well as features for the complete system based on configuration input. Creating the parameter file in Solution Designer is the first step that users need to do before running a motor.

Correct motor parameters are important for sensorless FOC to be able to run the motor in a steady state. MCE uses advanced flux-based sensorless algorithm which makes it easy to start a motor. Although the motor can start, depending on the application requirements, motor startup and dynamic performance may still need to be tuned in real load condition.

Below are some common problems and basic tuning techniques when using the MCE:

## 3.1 How to check if the current sensing setup is good

To check the current sensing setup, it is better to setup the motor system without the load, start the motor and set to a speed that motor can run smoothly. Use oscilloscope to measure the motor RMS current. In Solution Designer, output current display is usually slightly higher than measured motor current due to sensing noise. However, the values shown in Solution Designer should be as close to measured motor current as possible.

If current sensing noise is not good, here list the possible causes:

- Bad PCB layout
- Power devices switch too fast which cause too much noise
- Current sensing parameters don't match the hardware, related parameters:
    1. Deadtime
    2. PwmGuandBand (leg shunt only)
    3. SHDelay
    4. TminPhaseShift (single shunt only)

In single-shunt configuration, phase shift PWM provides better control performance. TminPhaseShift and SHDelay are two key parameters to achieve good single shunt current sensing in phase shift PWM mode.

To achieve good single shunt current sensing signal, TminPhaseShift and SHDelay should be configured following below guidelines:

$$TMinPhaseShift > Dead\ time + Ringing + ADC\ sampling\ time$$

$$SHDelay < Hardware\ delay\ time - ADC\ sampling\ time$$

$$TMinPhaseShift + SHDelay > Hardware\ delay\ time + Dead\ time + Ringing$$

Please note that TminPhaseShift may cause acoustic noise so it should be set to a value as small as possible.

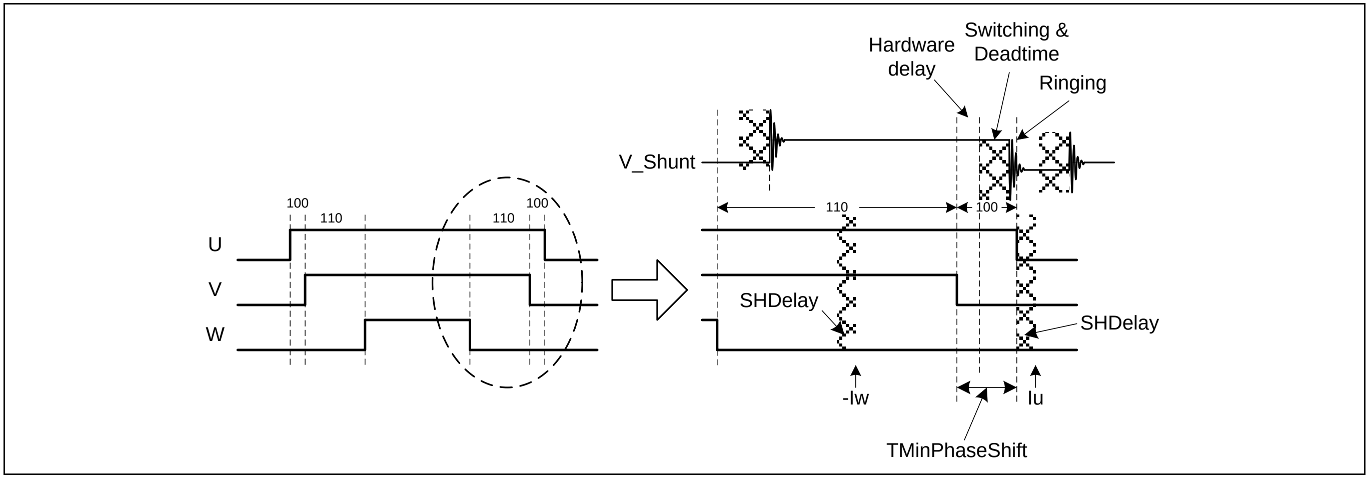


**Figure 90 Single Shunt Current Sensing for Phase Shift PWM**

There are three timings: Dead time, hardware delay time and ringing time. Dead time is already known since we set it in Solution Designer. What we need to measure on the hardware board is hardware delay time and ringing time.

Example of setting proper TminPhaseShift and SHDelay:

Below is an example showing how to measure the hardware and fine tune these two parameters.

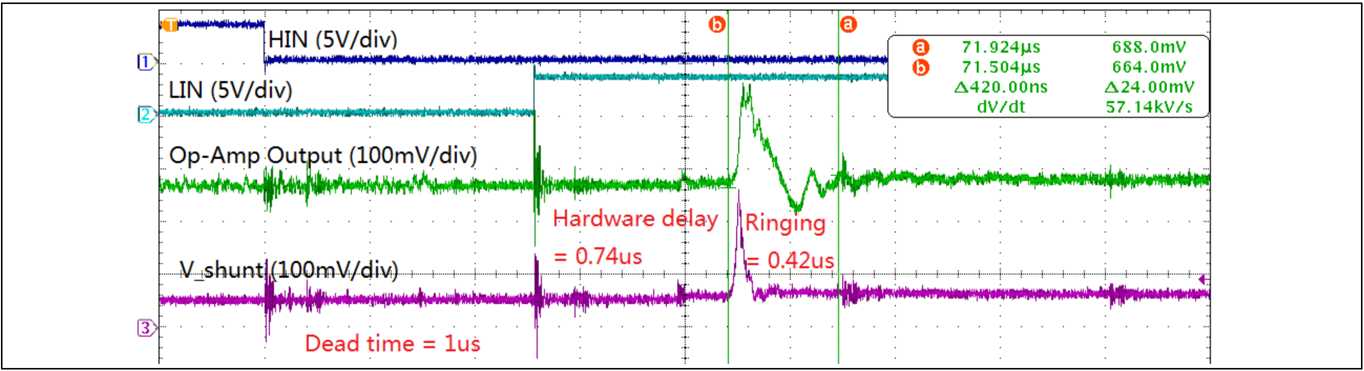


**Figure 91 Measuring hardware delay and ringing time**

$$TMinPhaseShift > 1us + 0.42us = 1.42us$$

$$SHDelay < 0.74us$$

$$TMinPhaseShift + SHDelay > 0.74us + 1us + 0.42us = 2.16us$$

We can easily configure TminPhaseShift=2.2us and SHDelay=0 to meet above criteria. But the optimum value should with minimum TminPhaseShift value to minimize acoustic noise cause by phase shift PWM. The optimum value should be:

$$TMinPhaseShift = 1.6us$$

$$SHDelay = 0.6us$$

## 3.2 Current regulator tuning

The MCE current controller utilizes field-oriented, synchronously rotating reference frame type regulators. Field-orientation provides significant simplification to the control dynamics of the current loop. There are two current regulators (one for the d-channel and one for the q-channel) employed for current regulation. The q-channel (torque) control structure is identical to the d-channel (flux). The current control dynamics of the d-channel is depicted in Figure 92. The motor windings can be represented by a first order lag with a time constant = L/R. This time constant is a function of the motor inductance and equivalent resistance (R = cable + winding). For a surface-mounted permanent magnet (SPM) motor, the d and q channel inductances are almost equal. In the case of an interior permanent magnet (IPM) motor, the q-channel inductance is normally higher than the d-channel inductance.

In the current control continuous time domain model Figure 92, the forward gain A models the conversion of the digital controller output to voltage (including inverter gain) and the feedback gain B models the transformation of the current feedback (Amps) to internal digital counts via an A/D converter. The calculation of the PI compensator gains ($KI_{Ireg}$, $Kp_{Ireg\_D}$) is done by using a pole-zero cancellation technique as illustrated in Figure 92, where the current controller is rearranged to give transfer function block C(s). Setting $Kp_{Ireg\_D}$ / $KI_{Ireg}$ of C(s) equal to the time constant of the motor ($\tau = L/R$), the controller zero will cancel the motor pole (pole-zero cancellation). Therefore, the model of the controller dynamics can be further simplified as shown in Figure 94. The equivalent transfer function of Figure 94 is a first order lag with time constant $\tau_c$. By selecting an appropriate current regulator response (typically 1 to 5 msec) for a particular application, the current regulator gains can be readily obtained. It may be noticed that using the pole zero cancellation technique, the motor inductance enters into proportional gain calculations and the resistance enters into integral gain calculations.

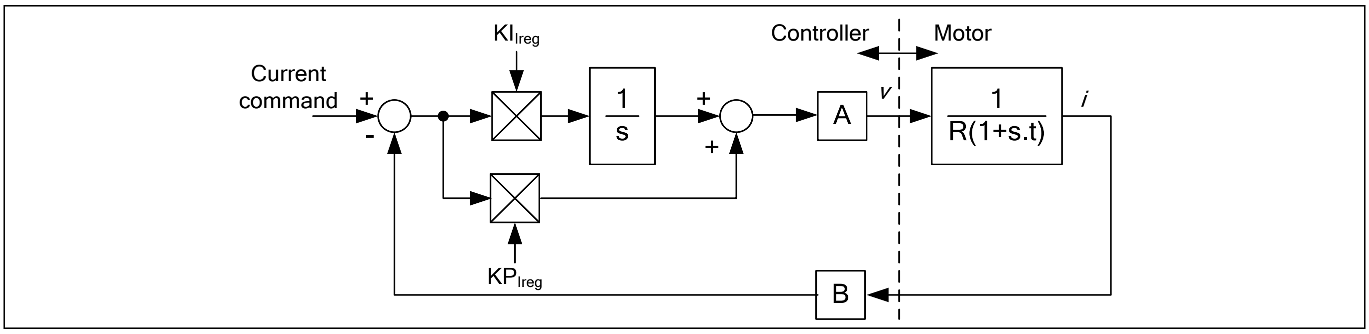


**Figure 92 Current controller dynamics**

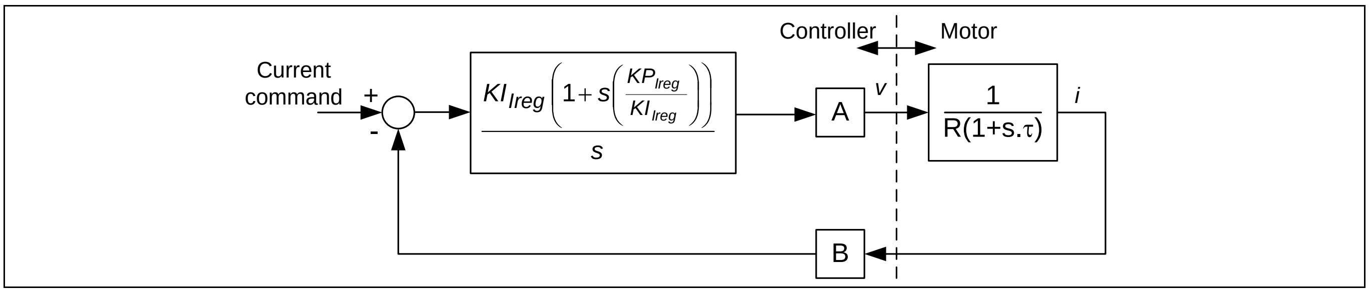

Controller
Motor
Current command
+
-
$KI_{Ireg}\left(1+s\left(\frac{KP_{Ireg}}{KI_{Ireg}}\right)\right)/s$
A
v
1/R(1+s.τ)
i
B

**Figure 93 Pole zero cancellation**

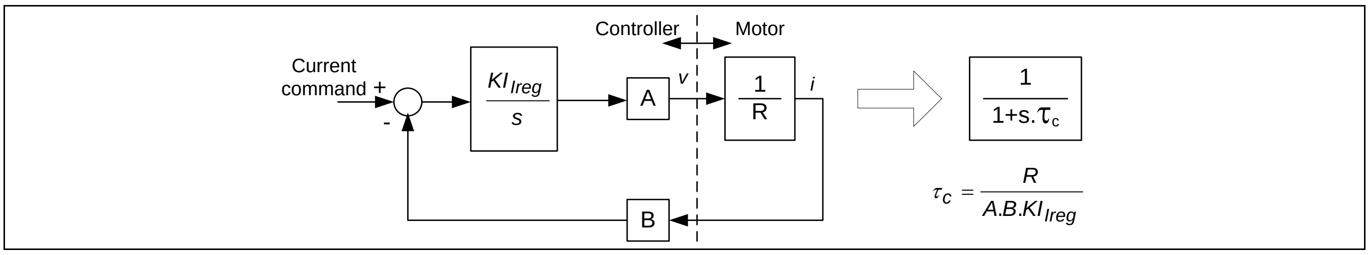


**Figure 94 Simplified current control dynamics due to pole zero cancellation**

Based on the pole-zero cancellation technique the controller gains in the continuous time domain model are evaluated by:

$$Kp_{Ireg} = \frac{L_q \cdot CurrentRegBW}{A \cdot B}$$

$$KI_{Ireg} = \frac{R \cdot CurrentRegBW}{A \cdot B}$$

Where A and B are the voltage and current scaling.

In the digital controller implementation, the integrator is a digital accumulator and so the discrete time domain model for the PI compensator must be used for the integrator. In this case the digital integrator gain, $Kx_{Ireg}$ and $Kx_{Ireg}D$ includes a scaling factor for the compensator sampling time.

$$Kx_{Ireg} = KI_{Ireg} \cdot T$$

T is the controller sampling time, which in this case is equal to the PWM period.

The voltage scaling, A, must account for gains in the forward rotation and the space vector modulator. The three phase inverter produces a peak line voltage equal to the dc bus voltage $V_{dc}$, so at 86.6% modulation (max. linear range) the rms phase voltage is $V_{dc}/\sqrt{2}/\sqrt{3}$. The modulator produces 86.6% modulation for a digital input of 7094 while the forward rotation function has a gain of 1.64676. Therefore, the current loop voltage scaling A is given by this equation:

$$A = \frac{V_{dc}/\sqrt{6}}{7094/1.64676} \; (in\ V_{rms}/cts)$$

The current loop feedback scaling, B, is defined by the shunt resistor, the amplifier gain, the A/D converter gain and the current feedback scaling parameter, IfbkScl. However, Solution Designer calculates IfbkScl so that a count of 4096 is equivalent to the motor rated rms current. Therefore, the current loop feedback scaling is simply given by:

$$B = \frac{4096}{I_{RATED}} \; (in\ cts/A_{rms})$$

The controller gains calculated for the current loop typically yield numbers that are less than one and so the current loop PI regulators include post multiplication scaling on the Kp and Kx inputs to increase the precision of the regulator gains. The multiplier on the Kp input is followed by a shift of 14 bits while the regulator on the Kx input is shifted by 19 bits. Therefore, the control gains calculated for this digital implementation are given by:

$$Kp_{Ireg} = \frac{L_q \cdot CurrentRegBW \cdot 2^{14}}{A \cdot B}$$

$$Kx_{Ireg} = \frac{R \cdot CurrentRegBW \cdot T \cdot 2^{19}}{A \cdot B}$$

Current regulator step response can be measured by using current control mode. Follow below steps to put the control into current control mode for current regulator step response diagnostic:

Step 1 – park the rotor to 0°:

a. Connect the motor and measure U phase current from oscilloscope.
b. AngleSelect = 0, disconnect flux rotor angle and use internal open loop angle.
c. CtrlModeSelect = 1, this is set to current control mode and disable the speed regulator.
d. TargetSpeed = 0, set open loop angle rotating speed to 0, thus angle will remain 0 during the test.
e. IdRef = 1024, apply 25% rated current to D axis.
f. Command = 1, start the drive, the control will regulate the current at 0° and the rotor will be aligned at 0°. The current is flowing out from U phase and flow into V and W phase.

We want to measure the step response without rotor movement. Step 1 is to park the rotor to certain angle so that the following steps will not cause any rotor movement. If the load inertia is high (such as fan blade), rotor will oscillate around parking angle and it may take long time to stop oscillating. If possible, use hand to stop oscillation and help it park at 0°.

Step 2 – apply initial 10% Id current:

a. IdRef = 410, apply 10% rated current to D axis.

Step 3 – apply 50% Id current:

a. IdRef = 2048, step change Id reference to 50%.

This is the step response we want to observe. Capture the U phase current waveform by using oscilloscope.

Step 4 – Stop the drive and recover the control to sensorless speed control mode:

a. Command = 0, stop the drive.
b. AngleSelect = 2, use flux rotor angle.
c. CtrlModeSelect = 2, set to speed mode.

Figure 95 shows measured step response with different current regulator bandwidth settings. Step response time constant is defined as the time duration from current start to rise until it reaches 63.2% $(1 - 1/e)$ of final current (not including over shooting). At lower current regulator bandwidth, actual step response time constant is quite close to theoretical value (9.88ms vs 10ms, 4.84ms vs 5ms, 2.4ms vs 2.5ms). At high current regulator bandwidth, actual time constant becomes much smaller than theoretical value (1.02ms vs 1.25ms, 0.428ms vs 0.625ms) and over-shoot start to appear. To achieve better step response performance, it is recommended to reduce $Kx_{Ireg}$ for high current regulator bandwidth.

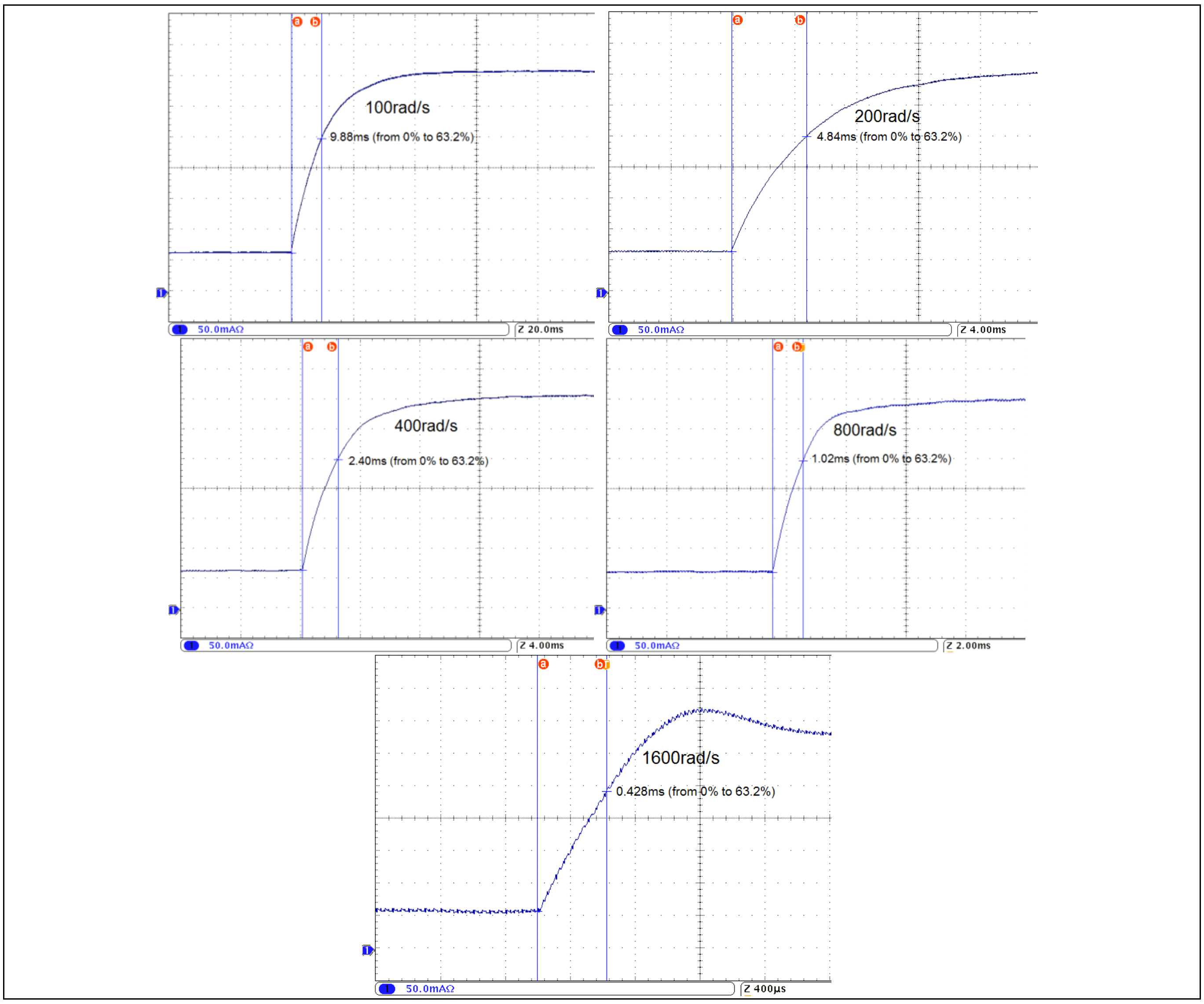


**Figure 95 Current regulator step response (100/200/400/800/1600rad/s)**

## 3.3 Difficulty to start the motor

- Make sure current sensing is good
- Make sure motor parameter is correct
- Adjust speed regulator PI gain and speed feedback filter time constant
- Adjust minimum speed
- Adjust speed accelerate and decelerate ramp
- Adjust flux estimator time constant
- Increase motor current limit

## 3.4 Motor speed not stable

- If speed is not stable at low speed, check if current sensing is good
- If motor speed oscillate, reduce speed regulator PI, especially I gain
- If motor speed change too much when load change, increase speed PI gain, especially P gain
- If two phase modulation is enabled, make sure 3ph to 2ph switch over speed is high enough, or temporarily disable 2 phase PWM

## 3.5 Motor current not stable in field weakening

- Adjust FwkKx together with speed regulator PI gain
- Adjust current regulator PI gain. In field weakening mode, make D axis current regulator higher bandwidth than Q axis, try increase KpIregD 2x higher or more than KpIreg.

## 3.6 Reducing acoustic noise

There are many reasons cause acoustic noise. Here are the most common reasons:

- Noise from current sensing circuit. Try to improve current sensing circuit, such as optimizing PCB layout, adjust op-amp load capacitor and feedback capacitor value, optimizing current sensing parameters, etc.
- Noise from high current regulator bandwidth, there is always noise from current sensing; improper current regulator may amplify the noise. To reduce noise from current regulator, try reduce current regulator PI gain, while doing this, make sure the control performance (especially at startup and high load) still good enough
- Noise from low PWM frequency or two phase PWM. Try increase PWM frequency. If the hardware is not suitable for higher PWM frequency, turn off two phase PWM and use 3-phase PWM only.
- Noise from the phase shift PWM scheme (single shunt configuration). Noise caused by phase shift PWM scheme can be reduced by reducing parameter value of TminPhaseShift. Please note in either case, SHDelay value also needs to be adjusted. It's not possible to eliminate noise in single shunt, if the application requires very low acoustic noise; change to leg shunt may solve the problem.
- Noise from over-modulation. When the motor is running at high speed, over-modulation can be used to maximize DC bus utilization. The drawback of over-modulation is that the output voltage is not sinusoidal; it contains high order harmonics which causes acoustic noise. If in this case, disable over-modulation.

# 4 Revision History

| Document version | Date of release | Description of changes |
|---|---|---|
| 1.0 | 2021-11-09 | Initial release |
| 1.1 | 2022-12-21 | Documente updated to refelct MCE software revision MCE FW_V5.1.0 |
| | | |
| | | |
| | | |
| | | |
| | | |
| | | |

**Edition 2021-11-09**

**Published by**

**Infineon Technologies AG**

**81726 Munich, Germany**

**Do you have a question about this document?**

**Email: erratum@infineon.com**

**Document reference**

**ifx1**