iMOTION™ Motion Control Engine

Software 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
- Guides through design, testing and optimization of application-specific hardware
- 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_V1.03.

Intended audience

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

Table of contents

About this document **1**
Table of contents **2**
1 Introduction **9**
2 Software Description **11**
2.1 Motor Control: Sensor-less / Hall Sensor Based FOC 11
2.1.1 Variable Scaling 12
2.1.2 State Handling 13
2.1.3 Current Sensing Offset Measurement 16
2.1.4 Bootstrap Capacitor Charge 16
2.1.5 Voltage measurement 16
2.1.6 DC Bus Compensation 17
2.1.7 Current Measurement 18
2.1.7.1 Leg Shunt Current Measurement 19
2.1.7.2 Single Shunt Current Measurement 20
2.1.8 Motor Current Limit Profile 31
2.1.9 Initial Angle Sensing 33
2.1.10 Over-Modulation 34
2.1.11 2-Phase Modulation 34
2.1.12 Catch Spin 36
2.1.12.1 Zero Speed Catch Spin 37
2.1.12.2 Forward Catch Spin 38
2.1.12.3 Reverse Catch Spin 39
2.1.13 Control Input 41
2.1.13.1 UART control 41
2.1.13.2 Vsp Analog Input 41
2.1.13.3 Frequency input 42
2.1.13.4 Duty Cycle Input Control 43
2.1.13.5 Automatic Restart 45
2.1.13.6 Forced control input change 45
2.1.13.7 Control Input Customization 46
2.1.14 Hall Sensor Interface 46
2.1.14.1 Interface Structure 46
2.1.14.2 Hall Sample De-Bounce Filter 47
2.1.14.3 Hall Angle Estimation 48
2.1.14.4 Hall Zero-Speed Check 50
2.1.14.5 Hall Pattern Validation 51
2.1.14.6 Hall Position Counter 52
2.1.14.7 Hall Angle Offset 52
2.1.14.8 Atan Angle Calculation 53
2.1.14.9 Hall Initial Position Estimation 55
2.1.14.10 Hall Sensor / Sensor-less Hybrid Operation 55
2.1.15 Torque Compensation 56
2.1.16 Protection 59
2.1.16.1 Flux PLL Out-of-Control Protection 59
2.1.16.2 Rotor Lock Protection 60
2.1.16.3 Motor Over Current Protection (OCP) 60
2.1.16.4 Over Temperature Protection 62
2.1.16.5 DC Bus Over / Under Voltage Protection 63
2.1.16.6 Phase Loss Protection 64

2.1.16.7 Execution Fault 64
2.2 Power Factor Correction 65
2.2.1 State Handling 67
2.2.2 Protection 68
2.2.2.1 PFC Over Current Protection (OCP) 68
2.2.2.2 DC Over/Under Voltage Protection 70
2.2.2.3 AC Over/Under Voltage Protection 71
2.2.2.4 Input Frequency Protection 72
2.3 User Mode UART 73
2.3.1 Baud Rate 73
2.3.2 Data Frame 73
2.3.3 Node Address 73
2.3.4 Link Break Protection 73
2.3.5 Command 74
2.3.6 Checksum 74
2.3.7 UART message 74
2.3.7.1 Read Status: Command = 0x00 74
2.3.7.2 Change Control Input Mode: Command =0x02 75
2.3.7.3 Motor Control: Command =0x03 75
2.3.7.4 Register Read: Command = 0x05 75
2.3.7.5 Register Write: Command = 0x06 76
2.3.7.6 Load and Save Parameter: Command = 0x20 76
2.3.8 Connecting multiple nodes to same network 77
2.3.9 UART Transmission Delay 78
2.4 JCOM Inter-Chip Communication 78
2.4.1 Operation Mode 78
2.4.1.1 Asynchronous Mode 78
2.4.2 Baud Rate 78
2.4.3 Message Frame Structure 78
2.4.4 Command and Response Protocol 79
2.4.4.1 Message Object: 0 79
2.4.4.2 Message Object: 1 80
2.4.4.3 Message Object: 6 82
2.4.4.4 Message Object: 7 82
2.5 Multiple Parameter Programming 83
2.5.1 Parameter Page Layout 83
2.5.2 Parameter Block Selection 83
2.5.2.1 Direct Select 83
2.5.2.2 UART Control 83
2.5.2.3 Analog Input 84
2.5.2.4 GPIO Pins 84
2.5.3 Parameter load fault 85
2.6 Script Engine 86
2.6.1 Overview 86
2.6.2 Script Program Structure 87
2.6.3 Script Program Execution 88
2.6.3.1 Execution Time Adjustment 88
2.6.3.2 Free Running timer 89
2.6.4 Constants 90
2.6.5 Variable types and scope 90
2.6.6 Motor and PFC Parameter Access 92
2.6.7 Operators 92

2.6.8 Expressions 93
2.6.9 Decision Structures 93
2.6.10 Loop Structures 95
2.6.11 Methods 96
2.6.11.1 Bit access Methods 96
2.6.11.2 Coherent update methods 97
2.6.11.3 Configurable UART API 98
2.6.11.4 User GPIOs 106
2.7 Internal Oscillator Calibration with Respect to Temperature 110
2.7.1 Overview 110
3 Register Description 111
3.1 System Control Register (App ID =0) 113
3.1.1 ParPageConf 113
3.1.2 InterfaceConf0 114
3.1.3 InterfaceConf1 114
3.1.4 SysTaskTime 115
3.1.5 CPU_Load 115
3.1.6 CPU_Load_Peak 115
3.1.7 FeatureID_selectH 115
3.1.8 GKConf 116
3.1.9 SW_Version 116
3.1.10 InternalTemp 116
3.1.11 SysTaskConfig 117
3.1.12 GPIOs[x] (x: 0 - 29) 118
3.2 Motor Control Register (App ID =1) 119
3.2.1 Control Register Group 125
3.2.1.1 HwConfig 125
3.2.1.2 SysConfig 126
3.2.1.3 AngleSelect 127
3.2.1.4 CtrlModeSelect 127
3.2.1.5 APPConfig 128
3.2.1.6 PrimaryControlRate 129
3.2.1.7 Command 129
3.2.1.8 SequencerState 130
3.2.1.9 MotorStatus 130
3.2.2 PWM Register Group 131
3.2.2.1 PwmFreq 131
3.2.2.2 PWMDeadtimeR 131
3.2.2.3 PWMDeadtimeF 132
3.2.2.4 SHDelay 132
3.2.2.5 TMinPhaseShift 132
3.2.2.6 TCntMin 133
3.2.2.7 PwmGuardBand 133
3.2.2.8 Pwm2PhThr 133
3.2.3 Speed Control Register Group 134
3.2.3.1 KpSreg 134
3.2.3.2 KxSreg 134
3.2.3.3 MotorLim 134
3.2.3.4 RegenLim 134
3.2.3.5 RegenSpdThr 135
3.2.3.6 LowSpeedLim 135

3.2.3.7 LowSpeedGain 135
3.2.3.8 SpdRampRate 135
3.2.3.9 MinSpd 136
3.2.3.10 TargetSpeed 136
3.2.3.11 TrqRef 136
3.2.3.1 SpdRef 137
3.2.3.2 RotorAngle 137
3.2.4 Hall Sensor InterfaceRegister Group 137
3.2.4.1 HallAngleOffset 137
3.2.4.2 Hall2FluxThr 137
3.2.4.3 Flux2HallThr 138
3.2.4.4 HallSampleFilter 138
3.2.4.5 HallSpdFiltBW 138
3.2.4.6 HallTimeoutPeriod 139
3.2.4.7 KpHallPLL 139
3.2.4.8 HallAngle 139
3.2.4.9 HallMotorSpeed 139
3.2.4.10 PositionCounter 140
3.2.4.11 PositionCounter_H 140
3.2.4.12 HallStatus 141
3.2.4.13 Hall_FrequencyOut 141
3.2.4.14 HallPLL_FrequencyAdjust 142
3.2.4.15 Hall_Atan_Angle 142
3.2.4.16 HallU 142
3.2.4.17 HallV 142
3.2.5 Flux Estimator PLL Register Group 143
3.2.5.1 Rs 143
3.2.5.2 L0 143
3.2.5.3 LSIncy 143
3.2.5.4 VoltScl 144
3.2.5.5 PllKp 144
3.2.5.6 PllKi 144
3.2.5.7 PllFreqLim 145
3.2.5.8 FlxTau 145
3.2.5.9 AtanTau 146
3.2.5.10 AngMTPA 146
3.2.5.11 SpdFiltBW 146
3.2.5.12 SpeedScale 146
3.2.5.13 MotorSpeed 147
3.2.5.14 FluxAngle 147
3.2.5.15 Flx_M 147
3.2.5.16 Abs_MotorSpeed 147
3.2.5.17 OpenLoopAngle 147
3.2.5.18 FluxMotorSpeed 148
3.2.6 FOC Register Group 148
3.2.6.1 IfbkScl 148
3.2.6.2 KpIreg 148
3.2.6.3 KpIregD 149
3.2.6.4 KxIreg 150
3.2.6.5 FwkVoltLvl 150
3.2.6.6 FwkKx 150
3.2.6.7 FwkCurRatio 150

3.2.6.8 VdqLim ... 151
3.2.6.9 AngDel ... 151
3.2.6.10 AngLim ... 151
3.2.6.11 IdqFiltBw ... 152
3.2.6.12 IdRef_Ext ... 152
3.2.6.13 IqRef_Ext ... 152
3.2.6.14 IdFilt ... 152
3.2.6.15 IqFilt ... 153
3.2.6.16 IdFwk ... 153
3.2.6.17 Id ... 153
3.2.6.18 Iq ... 153
3.2.6.19 MotorCurrent ... 153
3.2.7 Measurement Register Group ... 154
3.2.7.1 Iu ... 154
3.2.7.2 Iv ... 154
3.2.7.3 IW ... 154
3.2.7.4 I_Alpha ... 154
3.2.7.5 I_Beta ... 155
3.2.7.6 VdcRaw ... 155
3.2.7.7 VdcFilt ... 155
3.2.7.8 VTH ... 155
3.2.7.9 Ipeak ... 156
3.2.7.10 CurrentAmpOffset0 ... 156
3.2.7.11 CurrentAmpOffset1 ... 156
3.2.8 Protection Register Group ... 157
3.2.8.1 FaultEnable ... 157
3.2.8.2 VdcOvLevel ... 157
3.2.8.3 VdcUvLevel ... 158
3.2.8.4 CriticalOvLevel ... 158
3.2.8.5 RotorLockTime ... 158
3.2.8.6 FluxFaultTime ... 158
3.2.8.7 GateKillFilterTime ... 159
3.2.8.8 CompRef ... 159
3.2.8.9 Tshutdown ... 159
3.2.8.10 PhaseLossLevel ... 159
3.2.8.11 SwFaults ... 160
3.2.8.12 FaultClear ... 160
3.2.8.13 FaultRetryPeriod ... 160
3.2.8.14 FaultClear ... 160
3.2.8.15 FaultFlags ... 161
3.2.9 Start Control Register Group ... 162
3.2.9.1 BtsChargeTime ... 162
3.2.9.2 ParkAngle ... 162
3.2.9.3 ParkTime ... 163
3.2.9.4 OpenLoopRamp ... 163
3.2.9.5 IS_Pulses ... 163
3.2.9.6 IS_Duty ... 164
3.2.9.7 IS_IqInit ... 164
3.2.10 Catch Spin Register Group ... 165
3.2.10.1 TCatchSpin ... 165
3.2.10.1 DirectStartThr ... 165
3.2.11 Control Input Register Group ... 166

3.2.11.1 PGDeltaAngle 166
3.2.11.2 CmdStart 166
3.2.11.3 CmdStop 167
3.2.11.4 CmdGain 167
3.2.11.5 ControlFreq 167
3.2.11.6 ControlDuty 167
3.2.12 Voltage Control Register Group 168
3.2.12.1 Vd_Ext 168
3.2.12.2 Vq_Ext 168
3.2.12.3 V_Alpha 169
3.2.12.4 V_Beta 169
3.2.12.5 Vd 169
3.2.12.6 Vq 169
3.2.12.7 MotorVoltage 170
3.2.13 Torque Compensation Register Group 170
3.2.13.1 TrqCompGain 170
3.2.13.2 TrqCompAngOfst 170
3.2.13.3 TrqCompLim 170
3.2.13.4 TrqCompOnSpeed 171
3.2.13.1 TrqCompOffSpeed 171
3.2.13.2 TrqRef_Ext 171
3.2.13.3 TrqRef_Total 171
3.2.13.4 TrqCompBaseAngle 172
3.2.13.5 TrqCompStatus 172
3.3 PFC Control Register (App ID =3) 172
3.3.1 Control Register Group 175
3.3.1.1 PFC_HwConfig 175
3.3.1.2 PFC_SysConfig 176
3.3.1.3 PFC_ SequencerState 176
3.3.1.4 PFC_Command 177
3.3.2 PWM Register Group 178
3.3.2.1 PFC_PwmFreq 178
3.3.2.2 PFC_TMinOff 178
3.3.2.3 PFC_Deadtime 178
3.3.2.4 PFC_SHDelay 178
3.3.3 Voltage Control Register Group 179
3.3.3.1 PFC_IRectLim 179
3.3.3.2 PFC_IGenLim 179
3.3.3.3 PFC_VdcRampRate 179
3.3.3.4 PFC_KpVreg 180
3.3.3.5 PFC_KxVreg 180
3.3.3.6 PFC_TargetVoltInit 180
3.3.3.7 PFC_TargetVolt 180
3.3.3.8 PFC_VoltagePIoutput 181
3.3.4 Current Control Register Group 181
3.3.4.1 PFC_KpIreg 181
3.3.4.2 PFC_KxIreg 181
3.3.4.3 PFC_AcDcScale 182
3.3.4.4 PFC_LFactor 182
3.3.4.5 PFC_CurrentPIoutput 182
3.3.5 Protection Register Group 183
3.3.5.1 PFC_GateKillTime 183

3.3.5.2 PFC_VacOvLevel ... 183
3.3.5.3 PFC_VacLvLevel ... 183
3.3.5.4 PFC_VdcOvLevel ... 183
3.3.5.5 PFC_VdcUvLevel ... 184
3.3.5.6 PFC_FaultEnable ... 184
3.3.5.7 PFC_FaultClear ... 184
3.3.5.8 PFC_SwFaults ... 185
3.3.5.9 PFC_FaultFlags ... 185
3.3.6 Measurement Register Group ... 186
3.3.6.1 PFC_VdcRaw ... 186
3.3.6.2 PFC_VdcFilt ... 186
3.3.6.3 PFC_IpfcRaw ... 186
3.3.6.4 PFC_IpfcAvg ... 186
3.3.6.5 PFC_IpfcRMS ... 187
3.3.6.6 PFC_VacRaw ... 187
3.3.6.7 PFC_AbsVacRaw ... 187
3.3.6.8 PFC_VacRMS ... 187
3.3.6.9 PFC_ACPower ... 187
3.3.6.10 VACPk ... 188
3.4 Script Register (App ID = 4) ... 188
3.4.1 Script_UserVersion ... 189
3.4.2 Script_Command ... 189
3.4.3 ADC_Resultx [x: 0 to 11] ... 189
3.4.4 GPIO_IN_L ... 190
3.4.5 GPIO_IN_H ... 191
3.4.6 GPIO_OUT_L ... 192
3.4.7 GPIO_OUT_H ... 193
4 Motor Tuning ... 194
4.1 How to check if the current sensing setup is good ... 194
4.2 Current regulator tuning ... 195
4.3 Difficulty to start the motor ... 199
4.4 Motor speed not stable ... 199
4.5 Motor current not stable in field weakening ... 199
4.6 Reducing acoustic noise ... 199
5 Revision History ... 200

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 and Totem-Pole 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 MCEDesigner.
- 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% or if background tasks are not executed at least once every 60s	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

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: Sensor-less / Hall Sensor Based FOC

The MCE provides an advanced sensor-less 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.

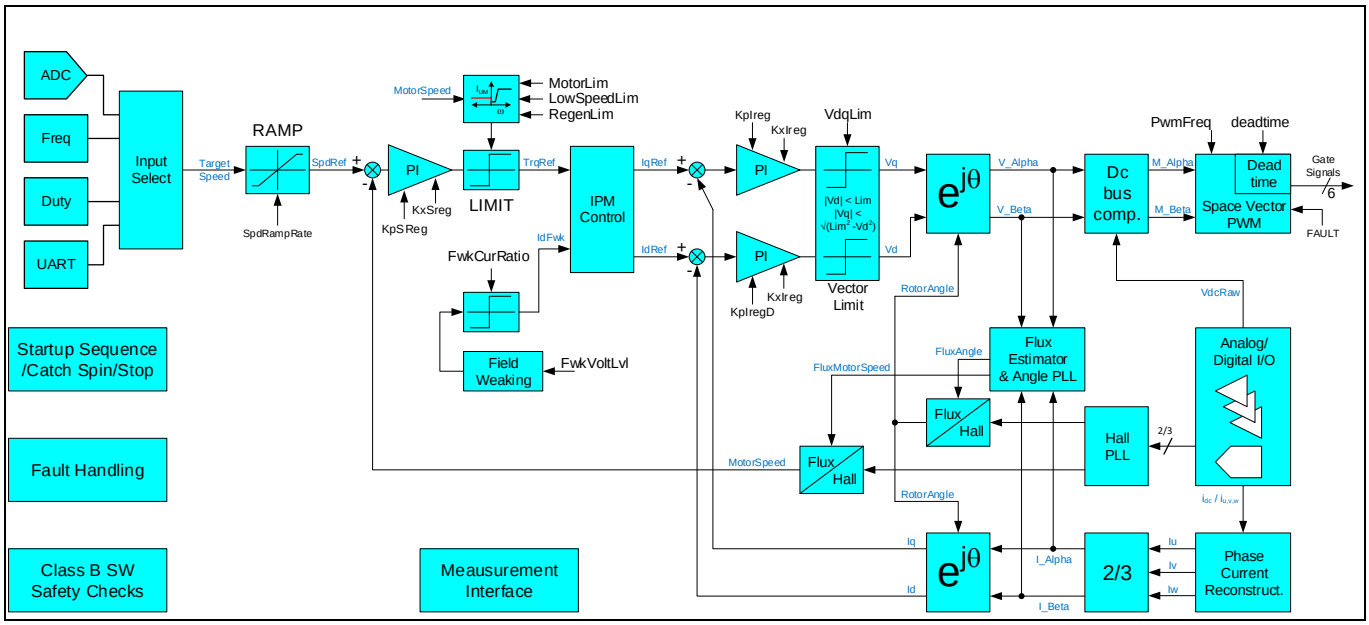


Figure 1 Top Level Diagram of Speed Control Loop and Sensorless FOC

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_α and V_β). 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_α and V_β 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.

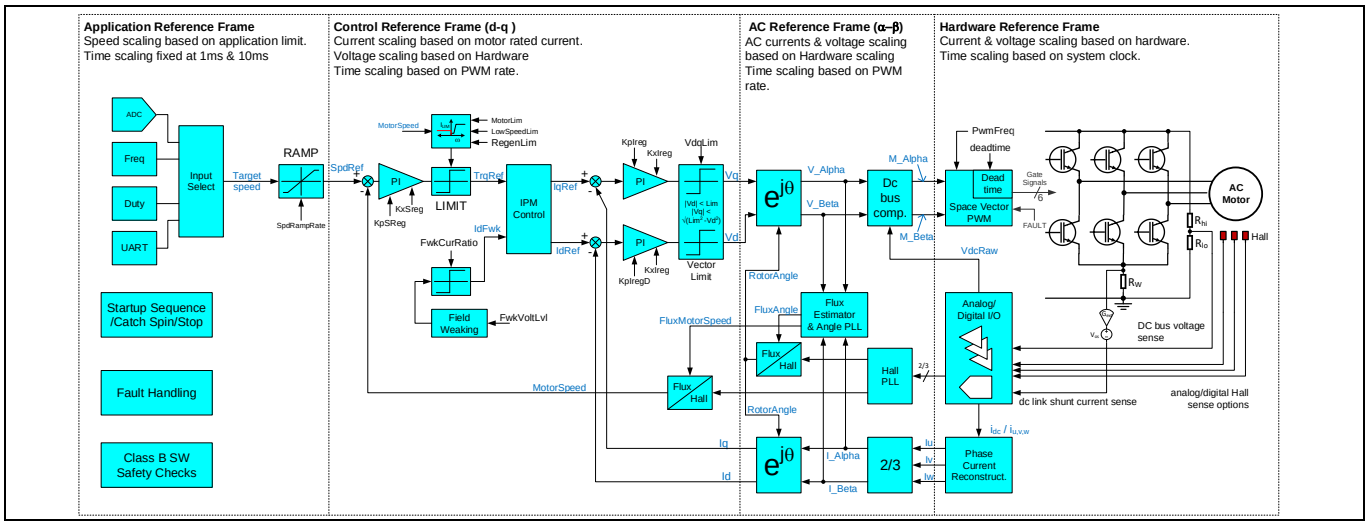


Figure 2 Scaling Domains for Control Variables and Parameters

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 10 states; each state has a value between 0-9, the current state of the sequencer is stored in “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.	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
4	MOTORRUN	Normal motor run mode	Stop Command	STOP

State No	Sequence State	State Functionality	Transition Event	Next Sequence State
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 10 seconds.	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
			Measured absolute motor speed is less threshold (“DirectStartThr” parameter)	ANGLESENSING
			Switch to next state if “TCatchSpin” register value is set to zero.	ANGLESENSING
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	OPEN_LOOP
			Switch to next state immediately if “ParkTime” parameter is set to zero.	OPEN_LOOP
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
			Switch to next state immediately if “OpenLoopRamp” parameter is set to zero	MOTOR_RUN
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	MOTORRUN
			Switch to next state immediately if ”IS_Pulses” parameter is set to zero	PARKING

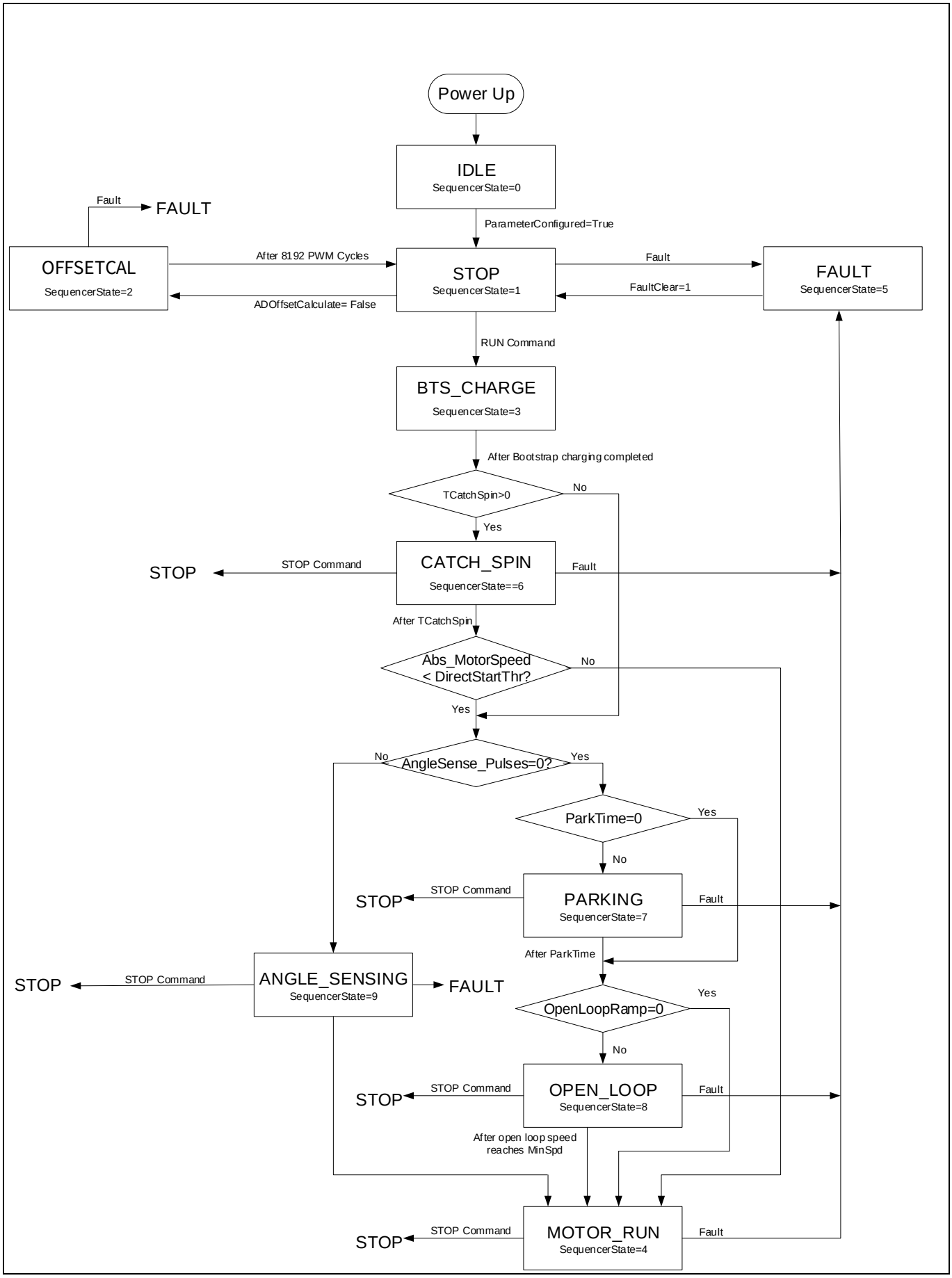


Figure 3 State Handling and Start Control Flow Chart

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 and at IV pin for phase V for leg shunt configuration or on ISS pin for single shunt configuration every motor PWM cycle for a configurable number of cycles ($N = 2^{13-CurrentOffsetCal_Psc}$, where $CurrentOffsetCal_Psc$ is the value of the bit field [15:13] of the parameter 'HwConfig'. At the end of the OFFSETCAL state, the N number of current offset measurement values for each phase are averaged and stored in variables 'CurrentAmpOffset0' and 'CurrentAmpOffset1' respectively.

The duration of OFFSETCAL state T_{Offset_Cal} can be estimated using the following equation: $T_{Offset_Cal} = \frac{Fast_Control_Rate \times 2^{13-CurrentOffsetCal_Psc}}{F_{PWM}}$. By default, $CurrentOffsetCal_PSC = 0$, so OFFSETCAL states takes 8192 PWM cycles.

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.

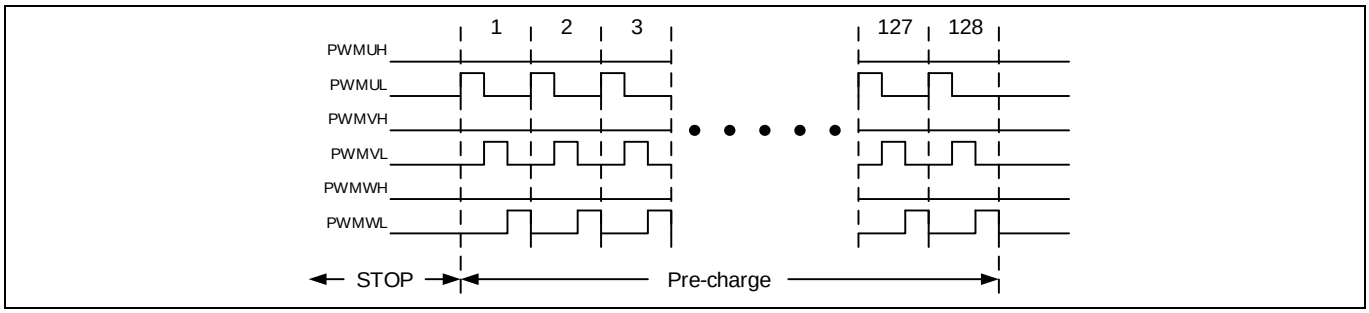


Figure 4 Bootstrap Capacitor Pre-charge

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.

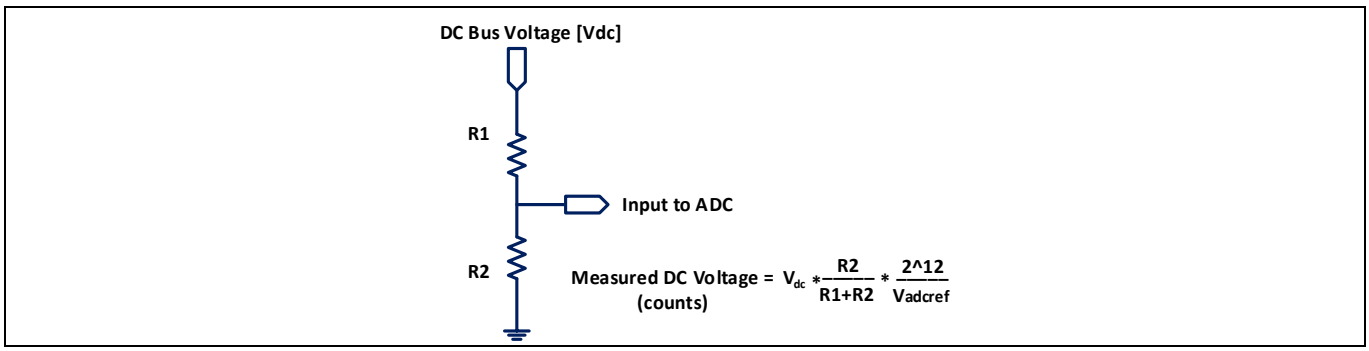


Figure 5 DC Bus voltage feedback signal path

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

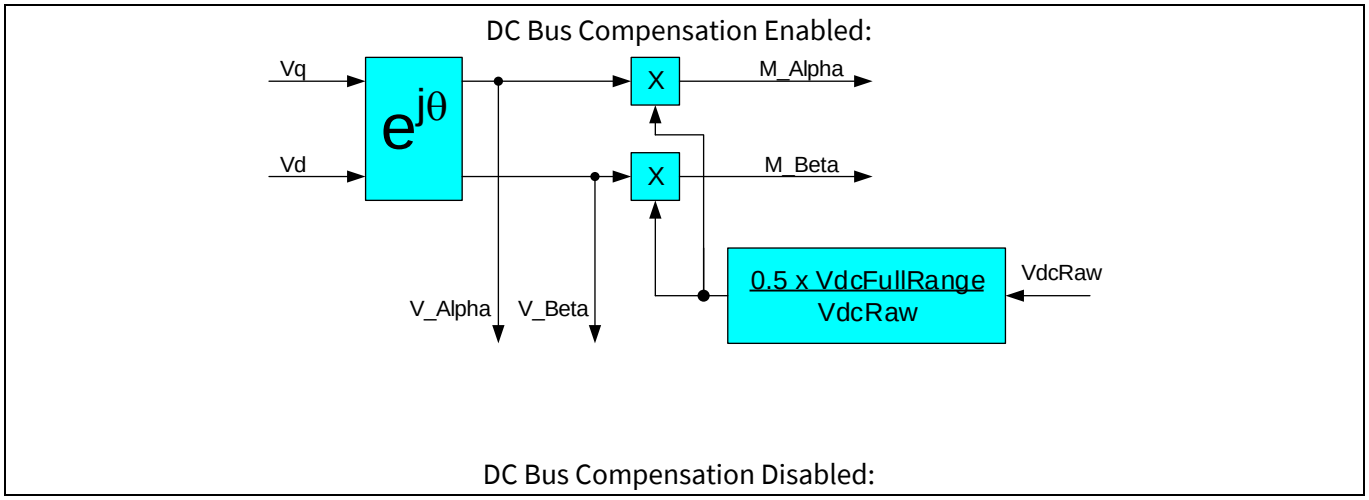
Attention: ***In MCEWizard R1 and R2 values shall be configured as per actual hardware used. Wrong configuration may lead to wrong under voltage/over voltage/ Critical over voltage fault or over voltage/under voltage/ critical over voltage conditions may not be detected correctly.***

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. Since the instantaneous DC bus voltage is part of the motor current control loop gain, if the current loop bandwidth is not high enough so that there is not enough loop gain at 2 x F_{line} frequency, then 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, V_Alpha and V_Beta, which 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, M_Alpha and M_Beta, are the 2 orthogonal components of the desired output voltage vector, based on which the SVPWM block generates the three phase PWM switching signals. M_Alpha and M_Beta are internal variables and not accessible to users. 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, V_Alpha and V_Beta are directly coupled with M_Alpha and M_Beta without any additional adjustment.



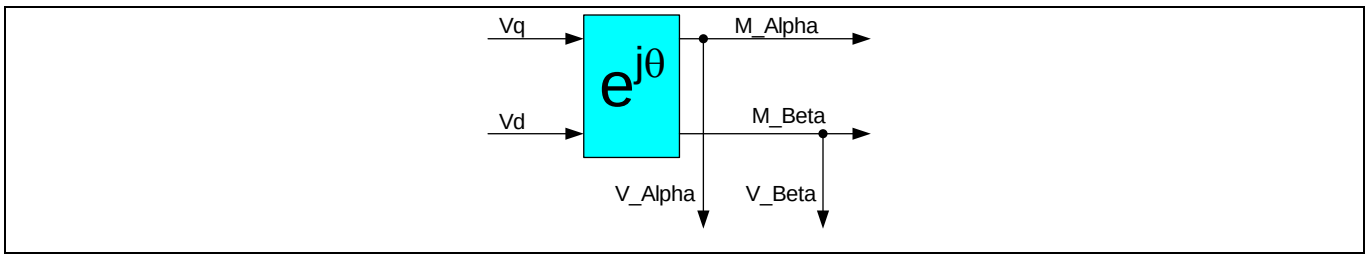


Figure 6 DC Bus Compensation Functional Diagram

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 sensor-less 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 symmetrical PWM - Minimum pulse width PWM Center aligned asymmetrical PWM - Phase shift PWM - 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 MCEWizard.

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

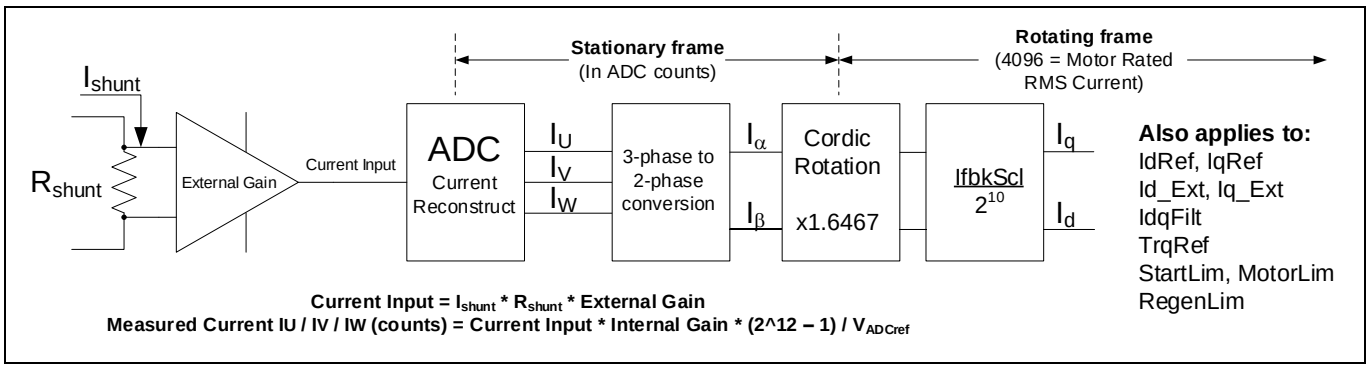


Figure 7 Motor current feedback signal path (TminPhaseShift ≠ 0)

Attention: ***In MCEWizard 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. 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. 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 U and phase V 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 $T_{PWM} - T_{BGmin}$.

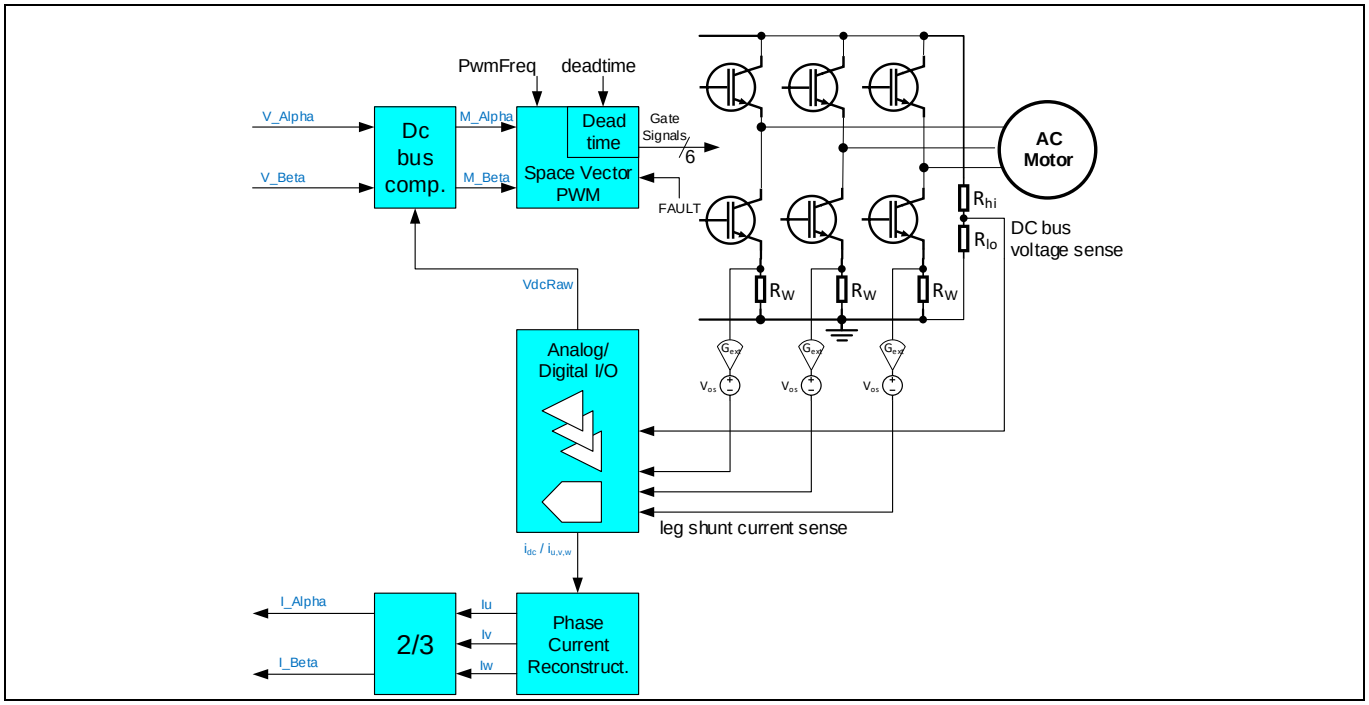


Figure 8 Typical Circuit Diagram for Leg Shunt Current Measurement Configuration

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.

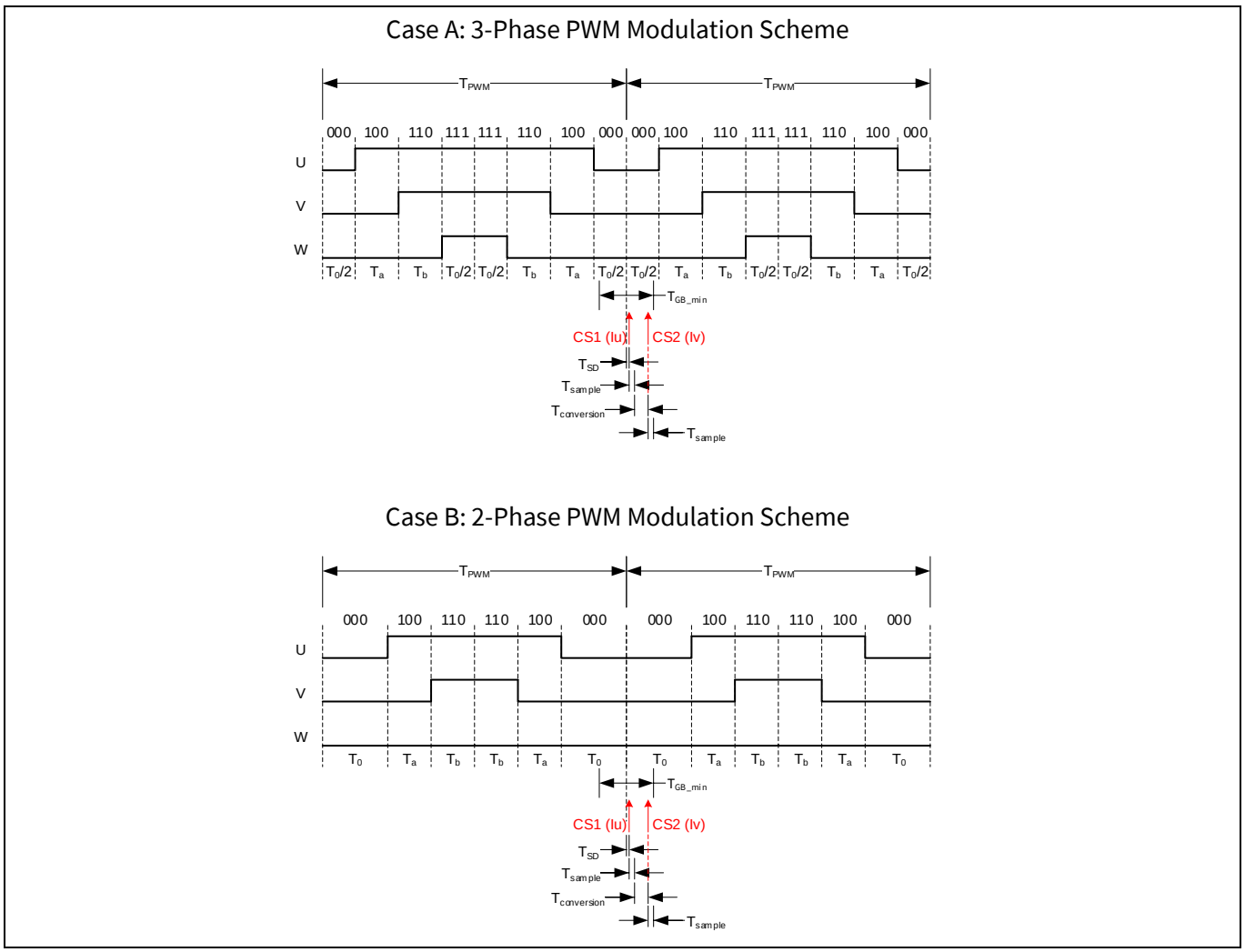


Figure 9 Leg Shunt Configuration Current Sensing Timing Diagram

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.

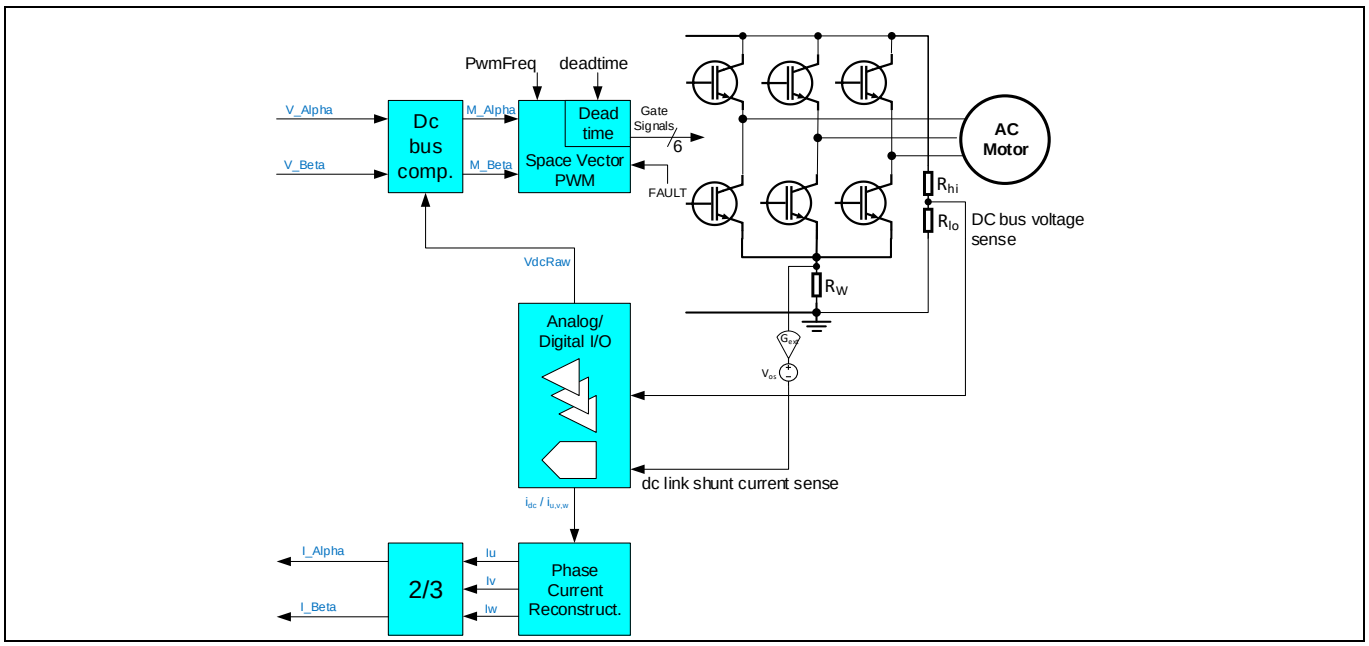


Figure 10 Typical Circuit Diagram for Single Shunt Current Measurement Configuration

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.

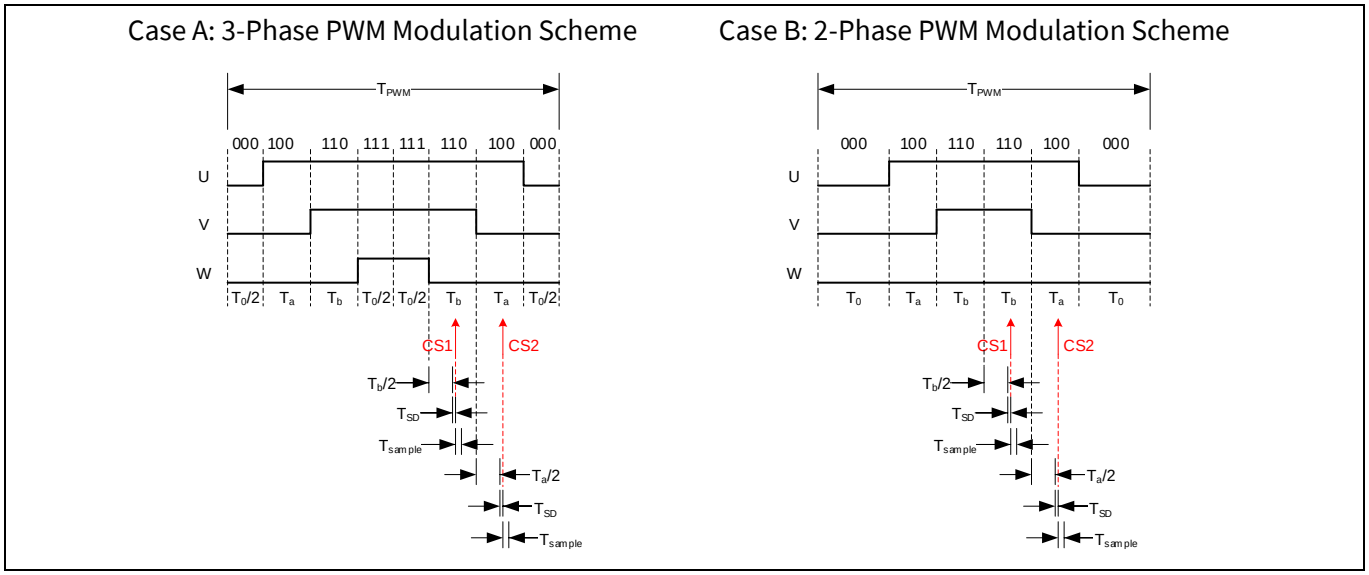


Figure 11 Single Shunt Configuration Current Sensing Timing Diagram

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.

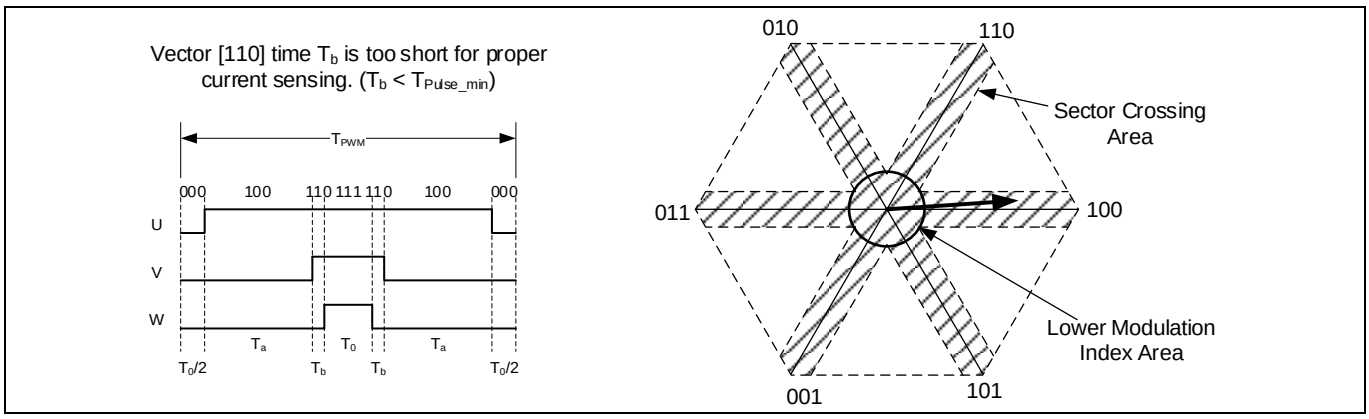


Figure 12 Narrow pulse limitation of single shunt current sensing

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. This minimum time is set by the parameter 'TcntMin' following this equation $T_{Pulse_min} = TcntMin \times 10.417ns$. 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.

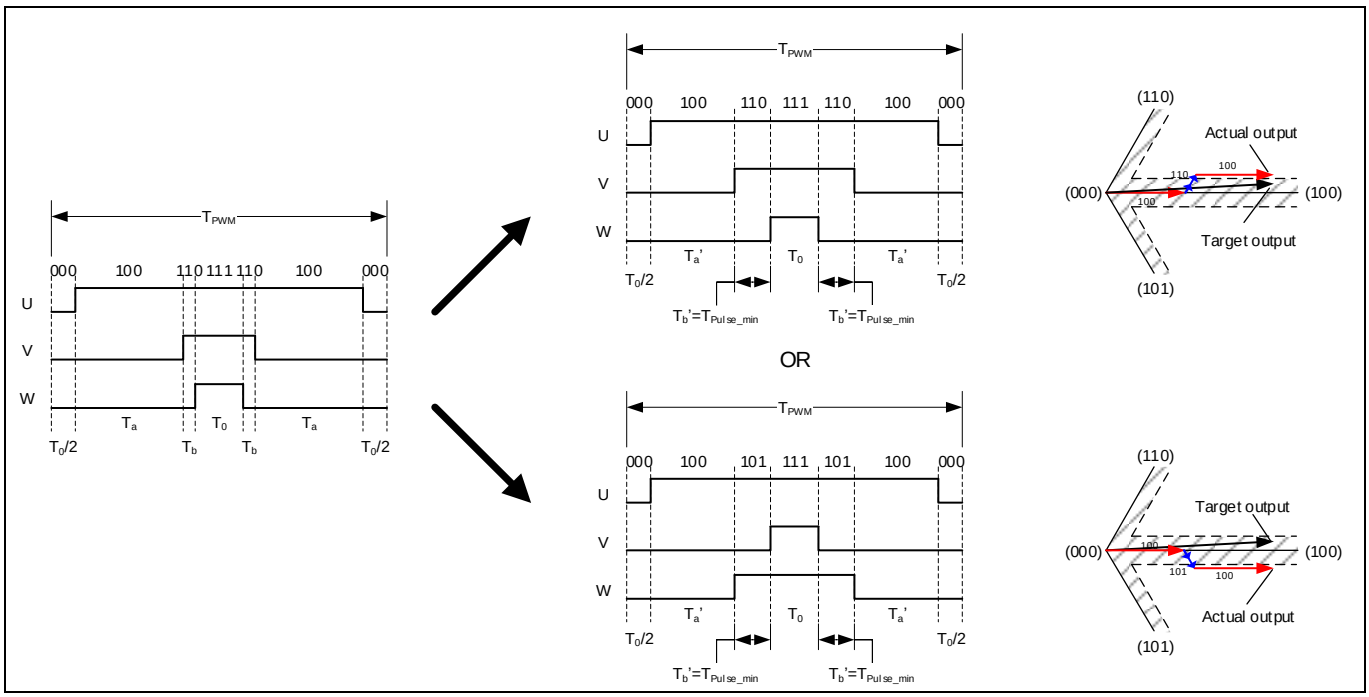


Figure 13 Minimum pulse PWM scheme limitation

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.

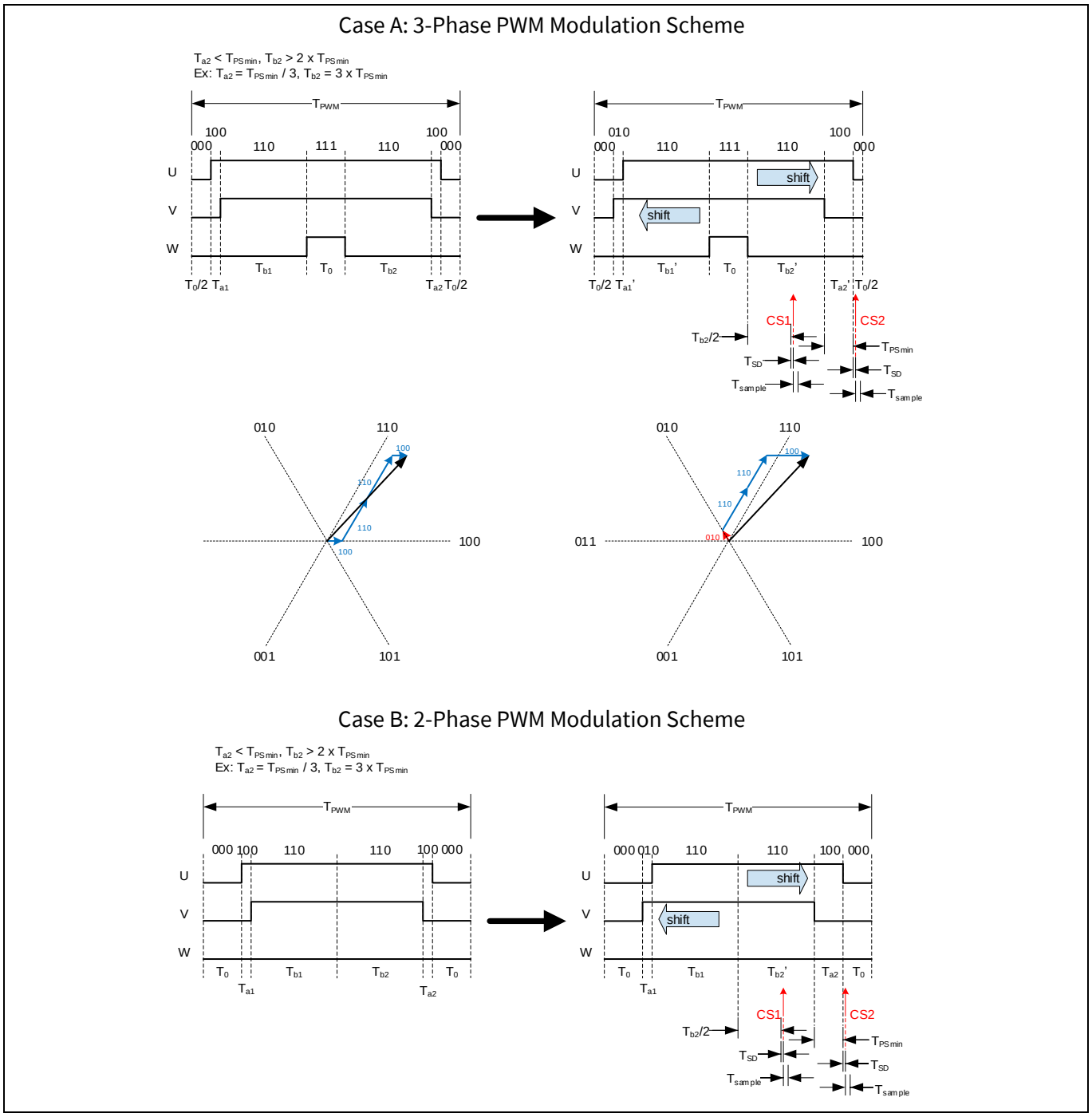


Figure 14 Single Shunt Configuration with Phase Shift PWM Scheme Current Sensing Timing Diagram (Case 1: $T_{a2} < T_{PSmin}$, $T_{b2} > 2 \times T_{PSmin}$)

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.

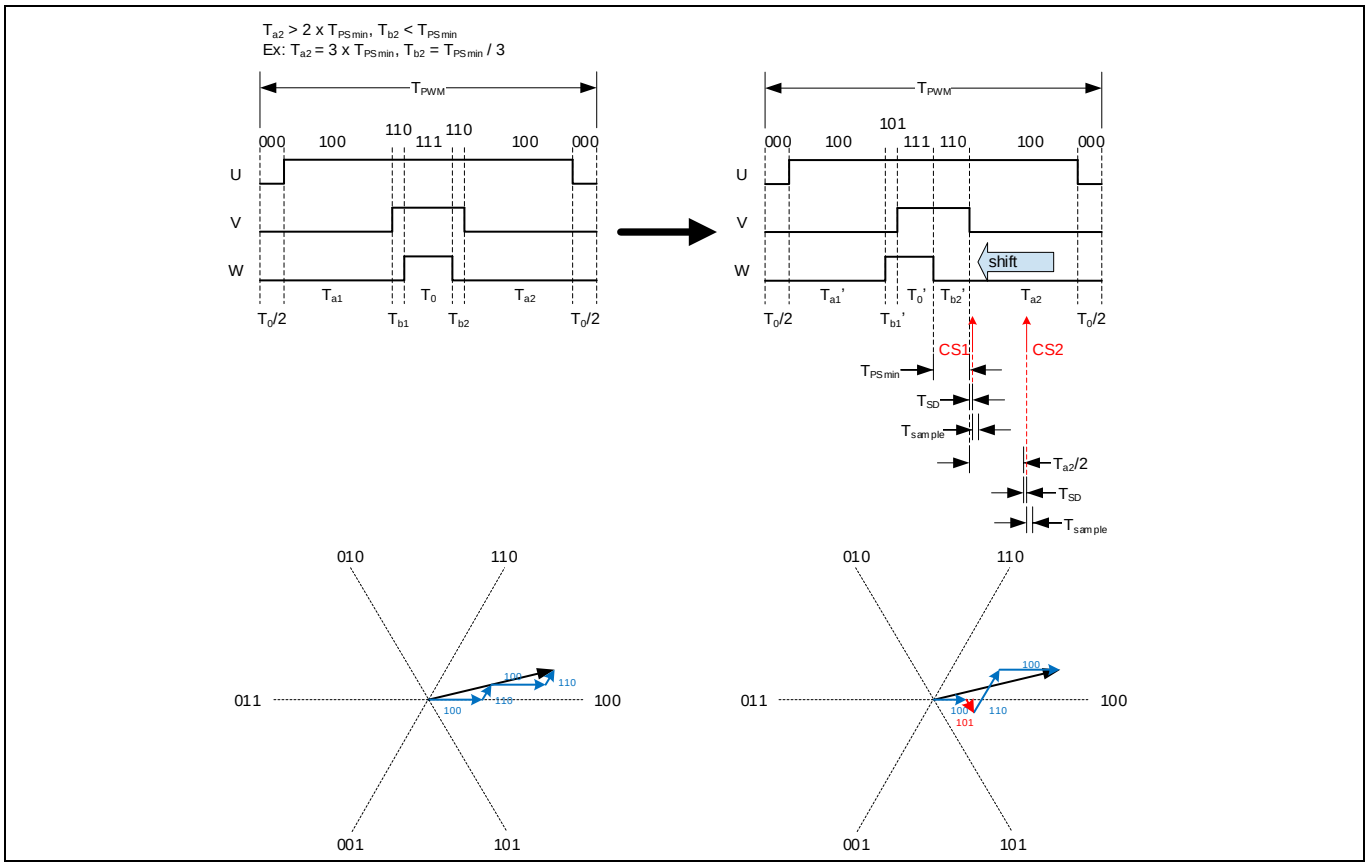


Figure 15 Single Shunt Configuration with Phase Shift PWM Scheme Current Sensing Timing Diagram (Case 2: T_{a2} > 2 x T_{PSmin}, T_{b2} < T_{PSmin})

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 x 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

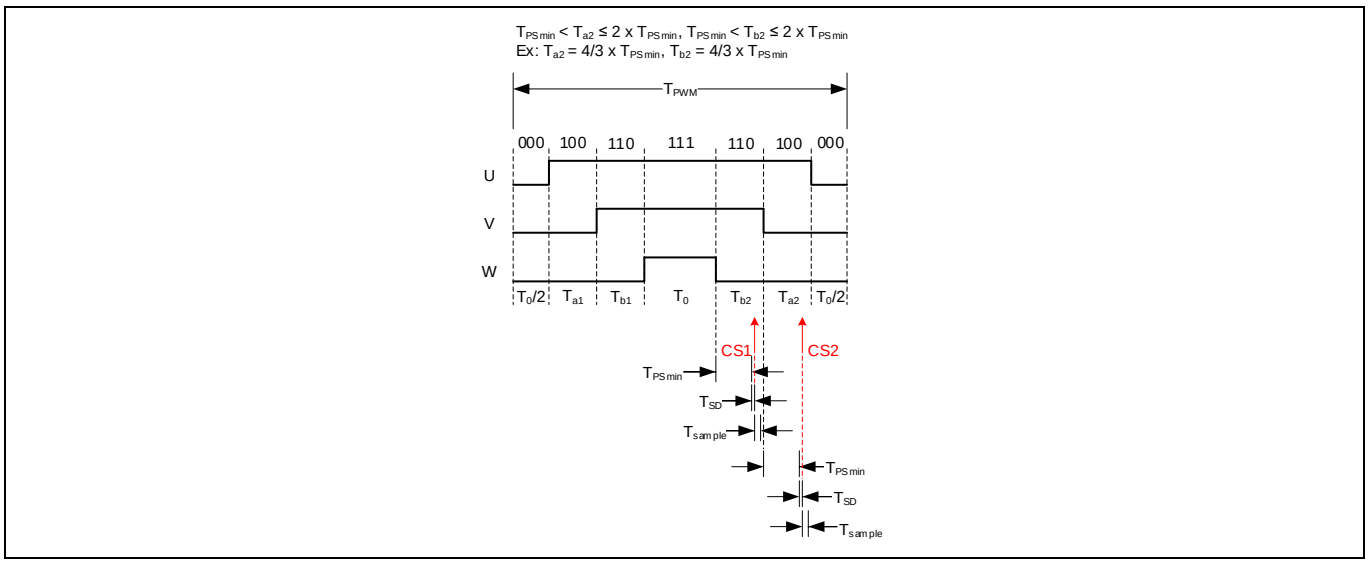


Figure 16 Single Shunt Configuration with Phase Shift PWM Scheme Current Sensing Timing Diagram (Case 3: $T_{PSmin} \leq T_{a2} \leq 2 \times T_{PSmin}$, $T_{PSmin} \leq T_{b2} \leq 2 \times T_{PSmin}$)

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].

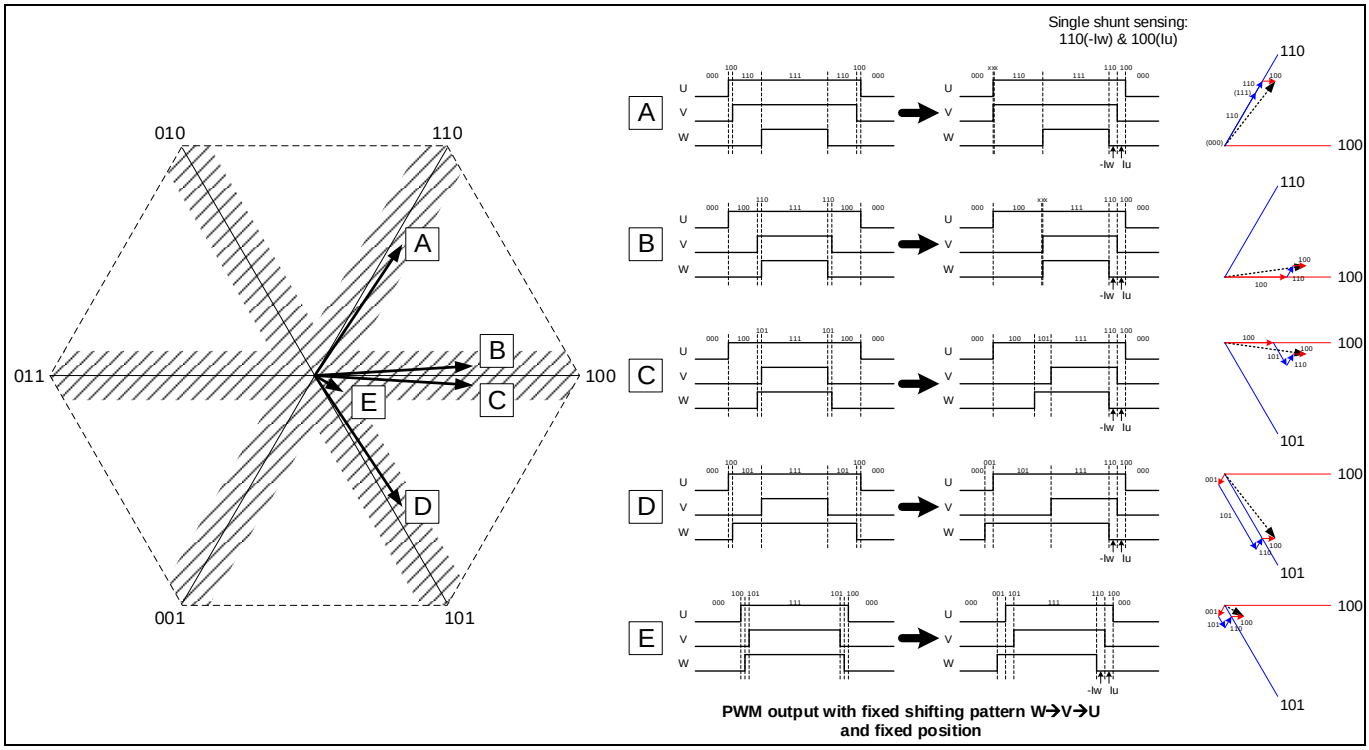


Figure 17 Low Noise Phase Shift PWM Scheme

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.

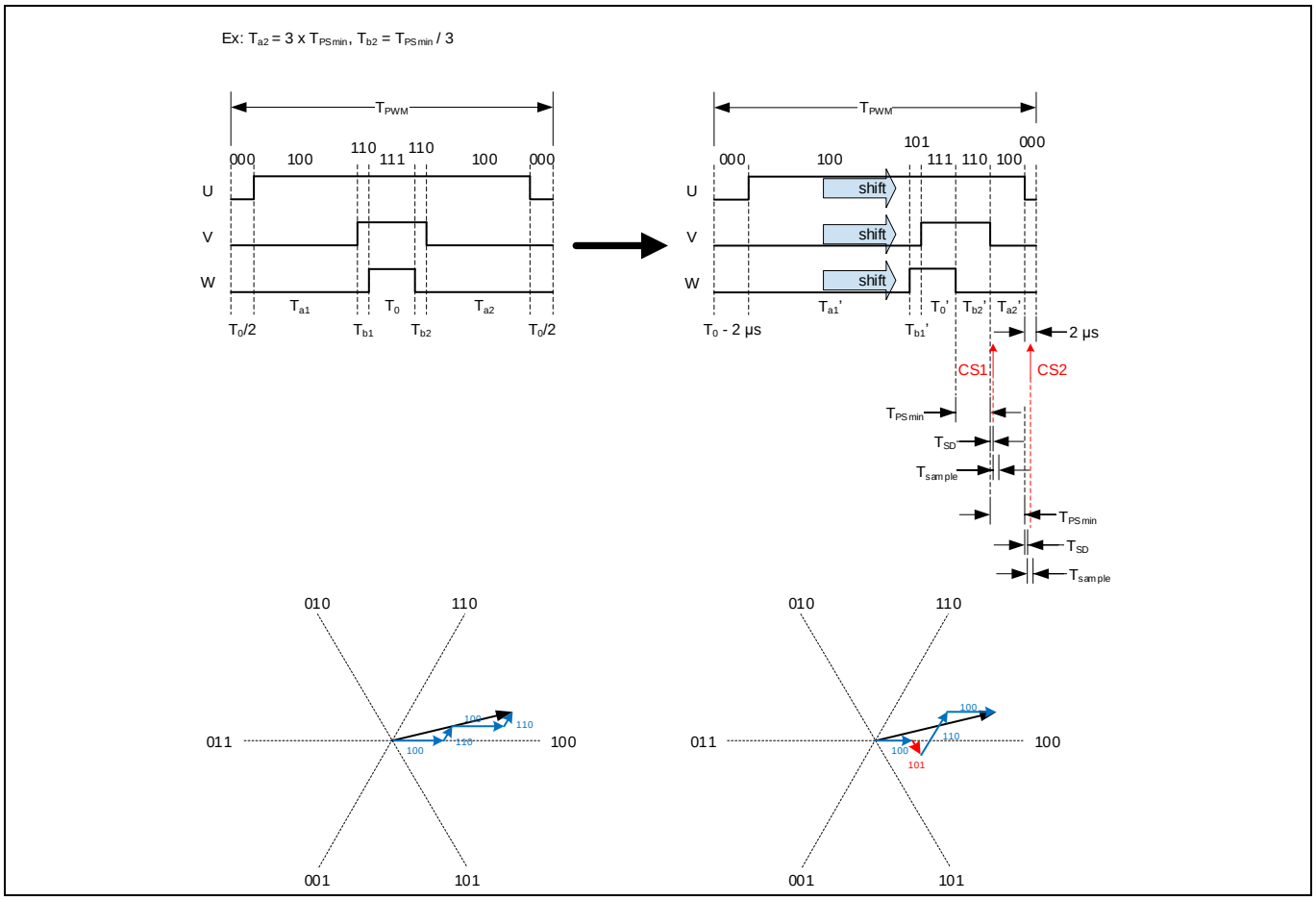


Figure 18 Single Shunt Configuration with Low Noise Phase Shift PWM Scheme Current Sensing Timing Diagram

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.

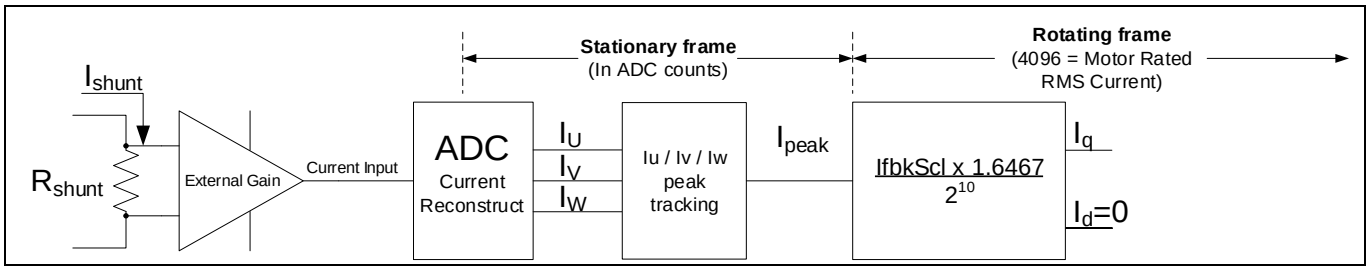


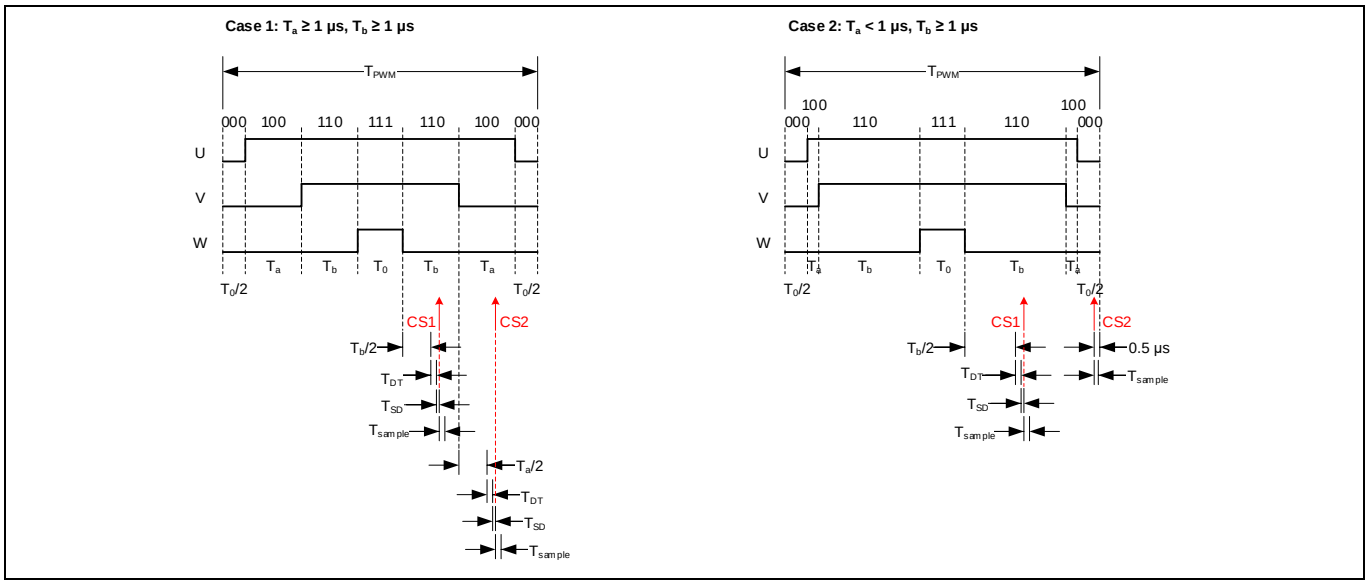
Figure 19 Motor current feedback signal path (TminPhaseShift = 0, single shunt)

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.



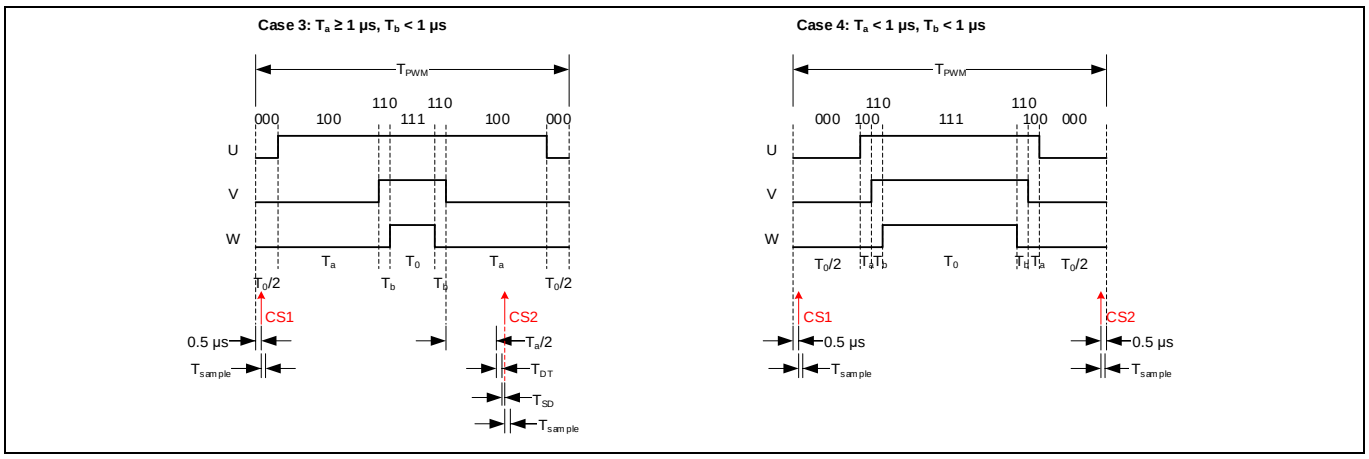
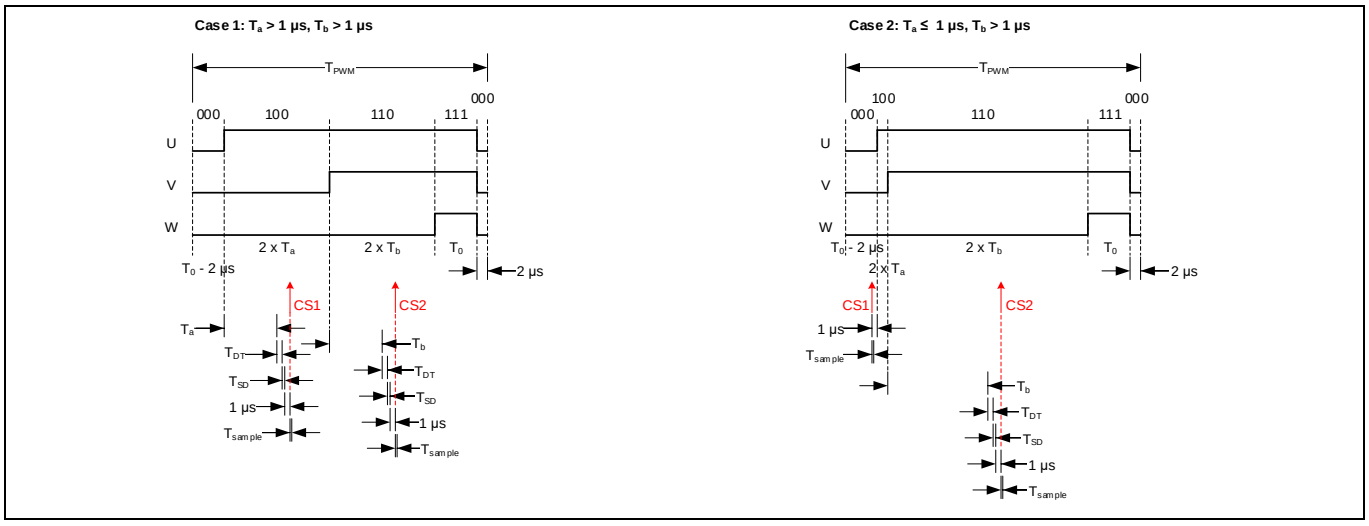


Figure 20 Single Shunt Configuration with Phase Shift PWM Scheme in Peak Current Tracking Mode Current Sensing Timing Diagram

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.



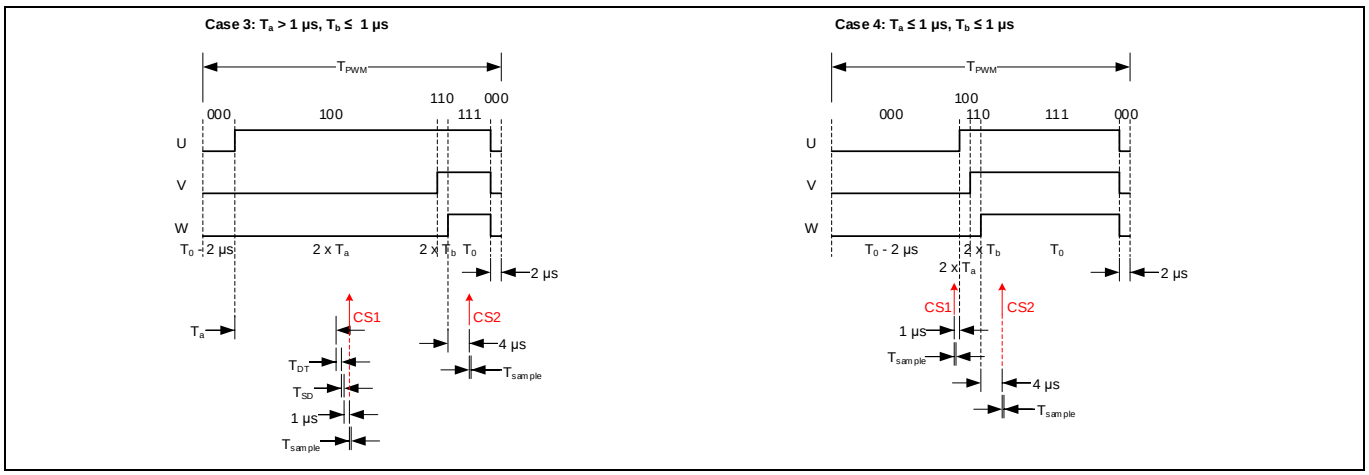


Figure 21 Single Shunt Configuration with Low Noise Phase Shift PWM Scheme in Peak Current Tracking Mode Current Sensing Timing Diagram

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.

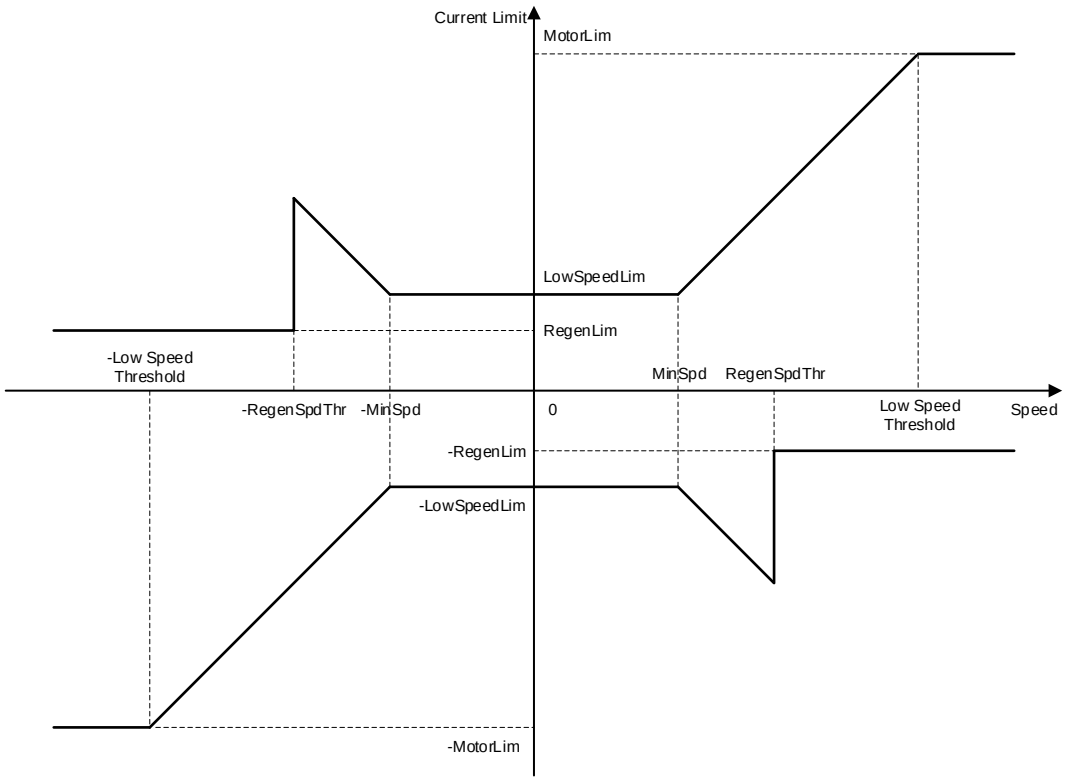
$$Motor\ Current\ Limit = LowSpeedLim + (|MotorSpeed| - MinSpd) \times LowSpeedGain$$

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'.

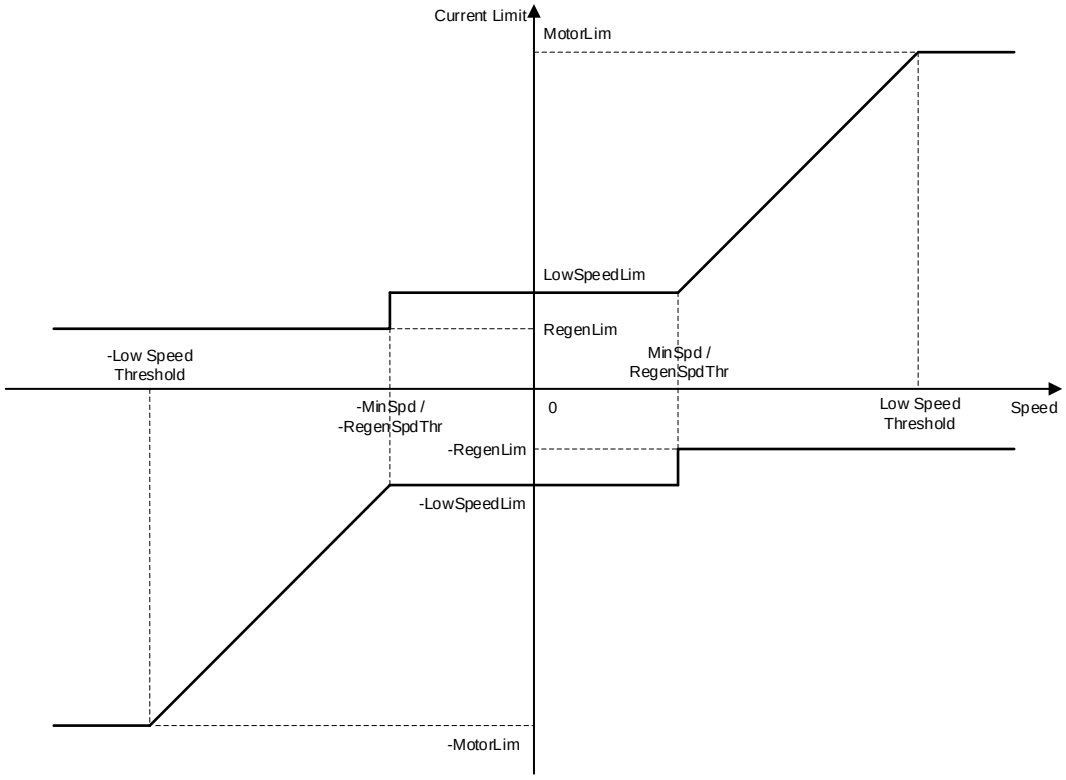
In regenerating mode, when the absolute value of motor speed is below a threshold specified by the parameter 'RegenSpdThr', the motor current limit follows the above-mentioned linear relationship. When the absolute value of motor speed goes beyond the threshold specified by the parameter 'RegenSpdThr',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

Case 1: MinSpd < RegenSpdThr



Case 2: MinSpd = RegenSpdThr



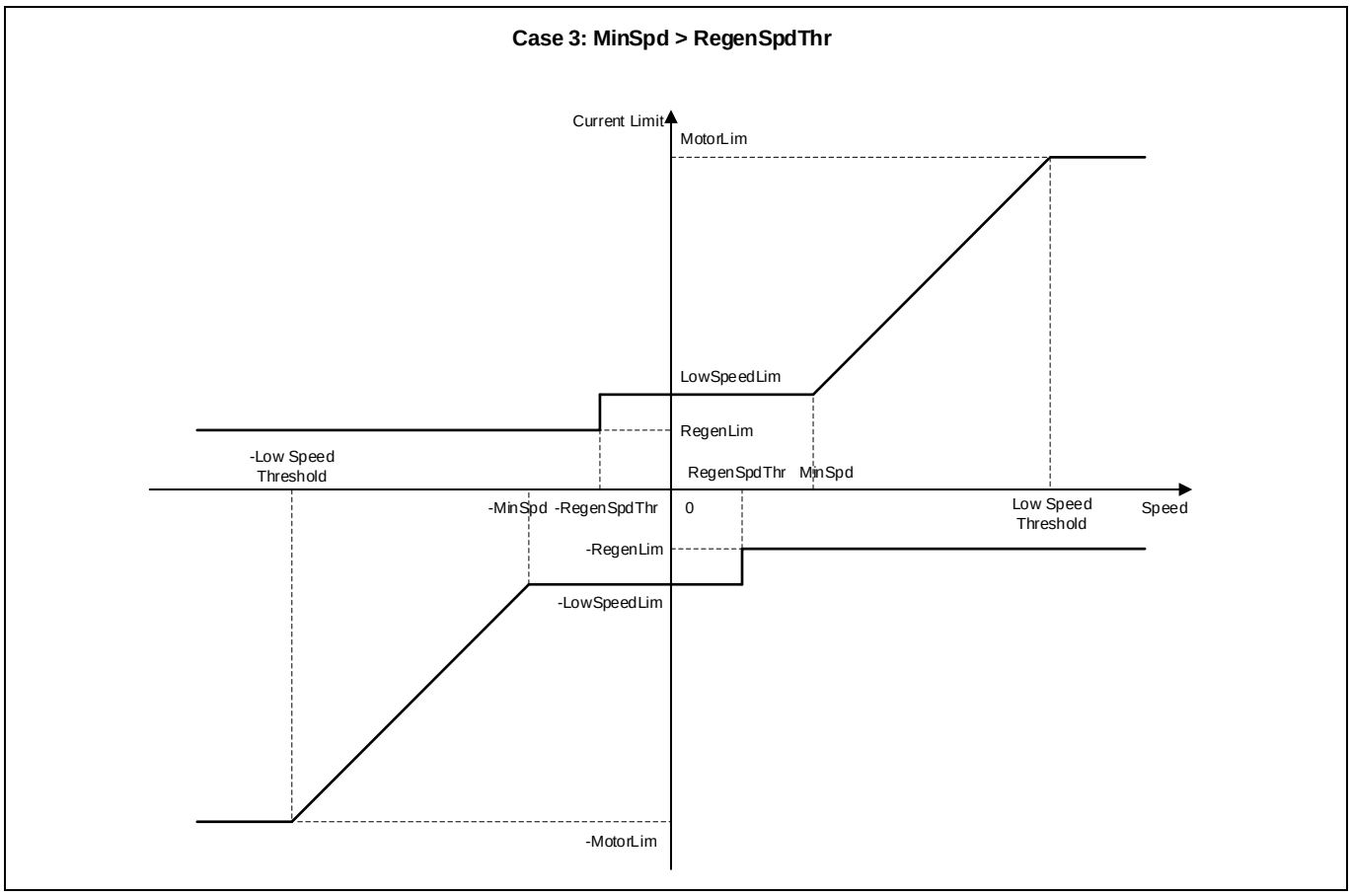


Figure 22 Motor Current Limit Profile

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 MCEWizard 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.

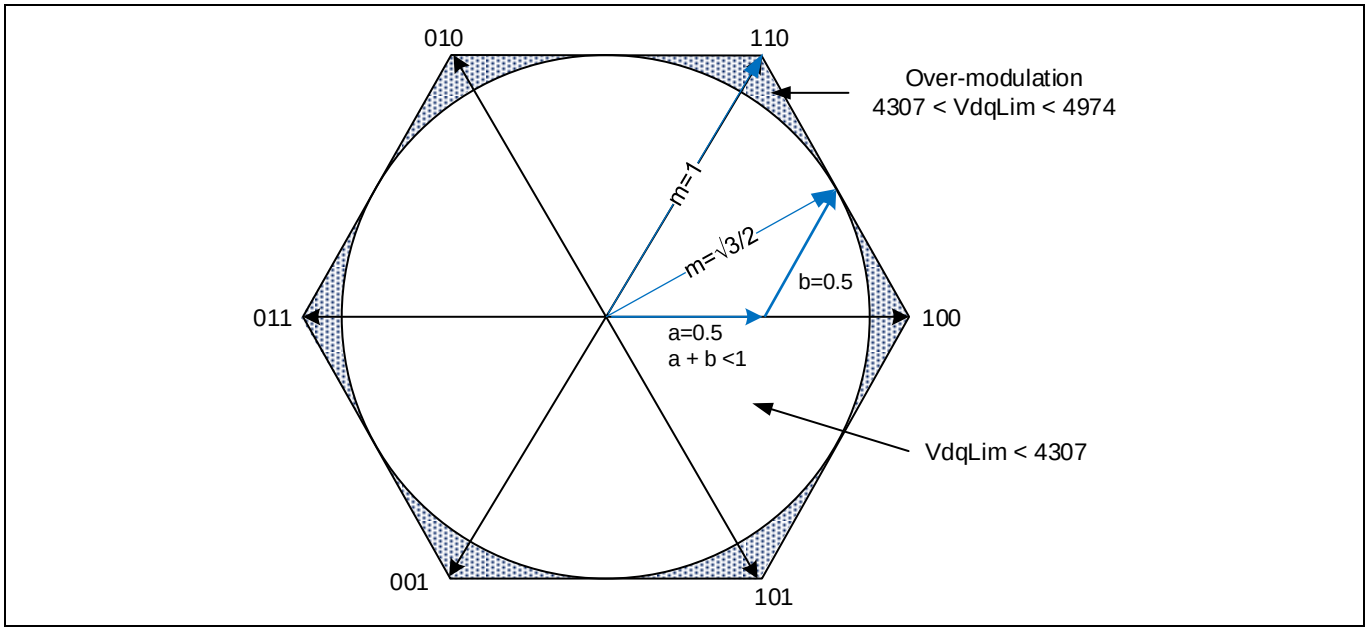


Figure 23 SVPWM Vector Timing Limit Diagram

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].

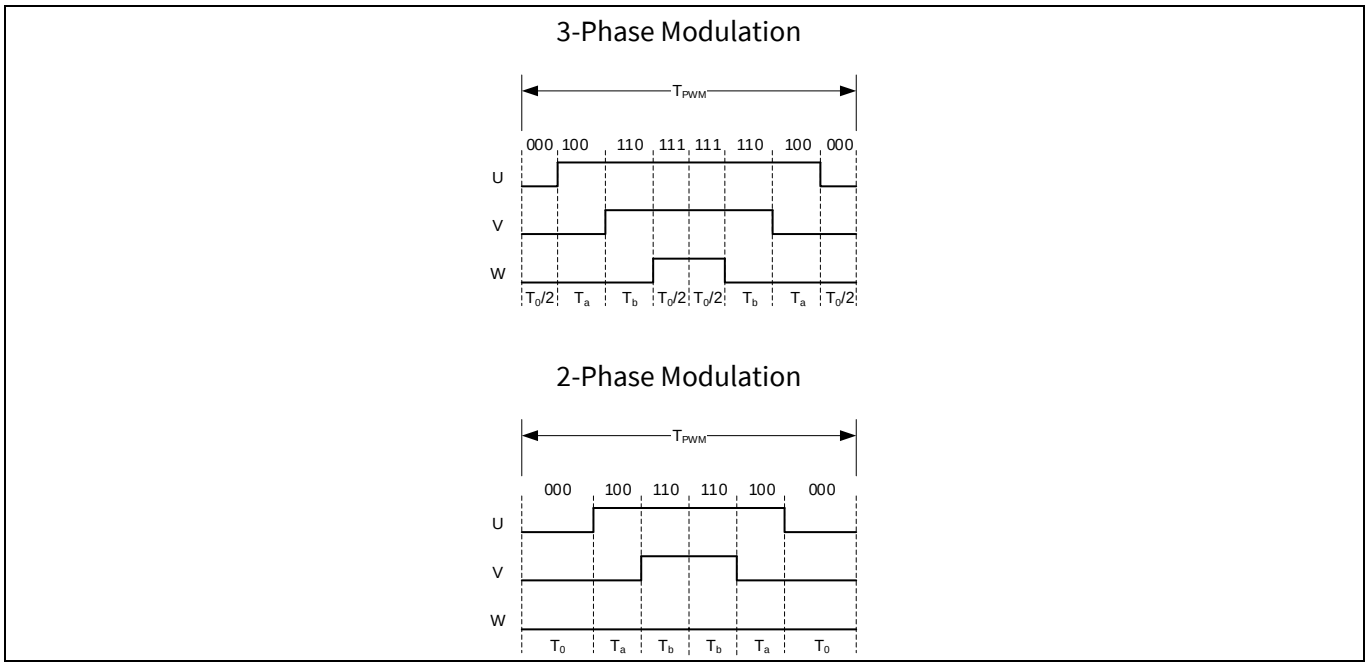


Figure 24 3-Phase / 2-Phase Type 3 SV PWM Modulation Diagram

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.

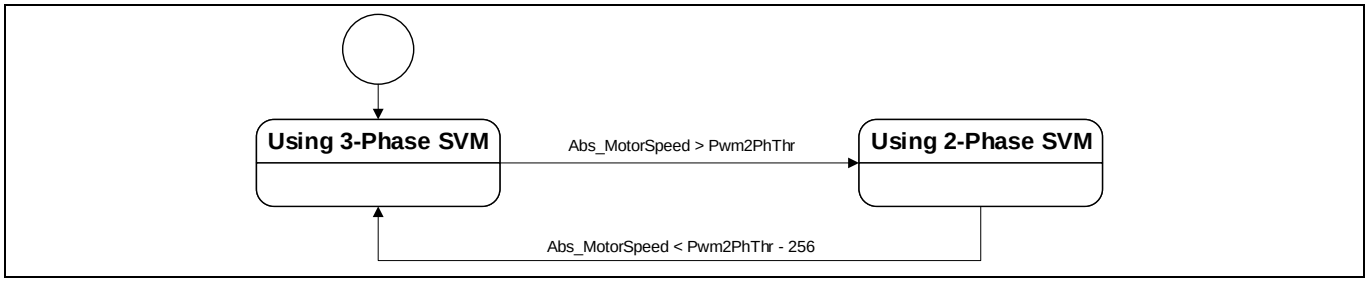


Figure 25 3-Phase SVM and 2-Phase SVM State Transition Diagram

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.

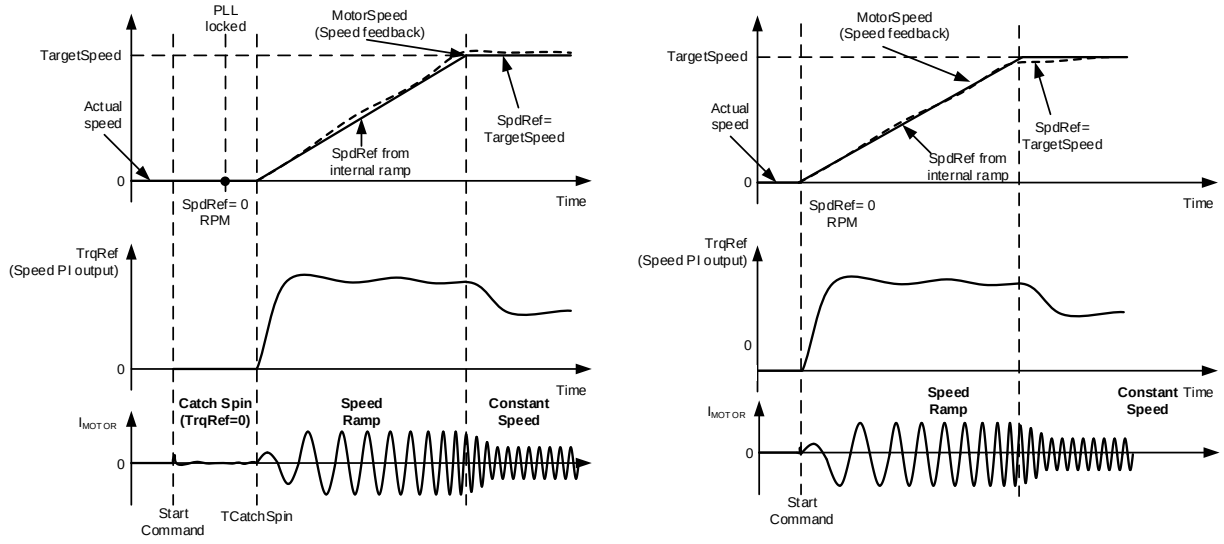


Figure 26 Zero Speed Catch Spin - Motor start with/without catch spin

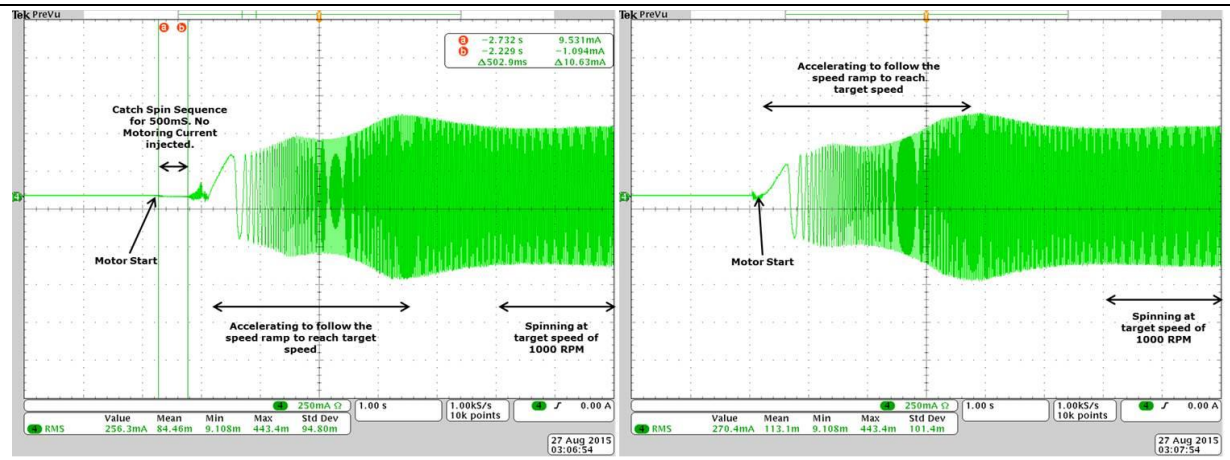


Figure 27 Motor Phase Current - Zero Speed Catch Spin - Motor start with/without catch spin

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 ‘SpdRef’). Hence, the motor is decelerated (using regenerative braking) to force the motor to follow the speed reference (variable ‘SpdRef’). 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.

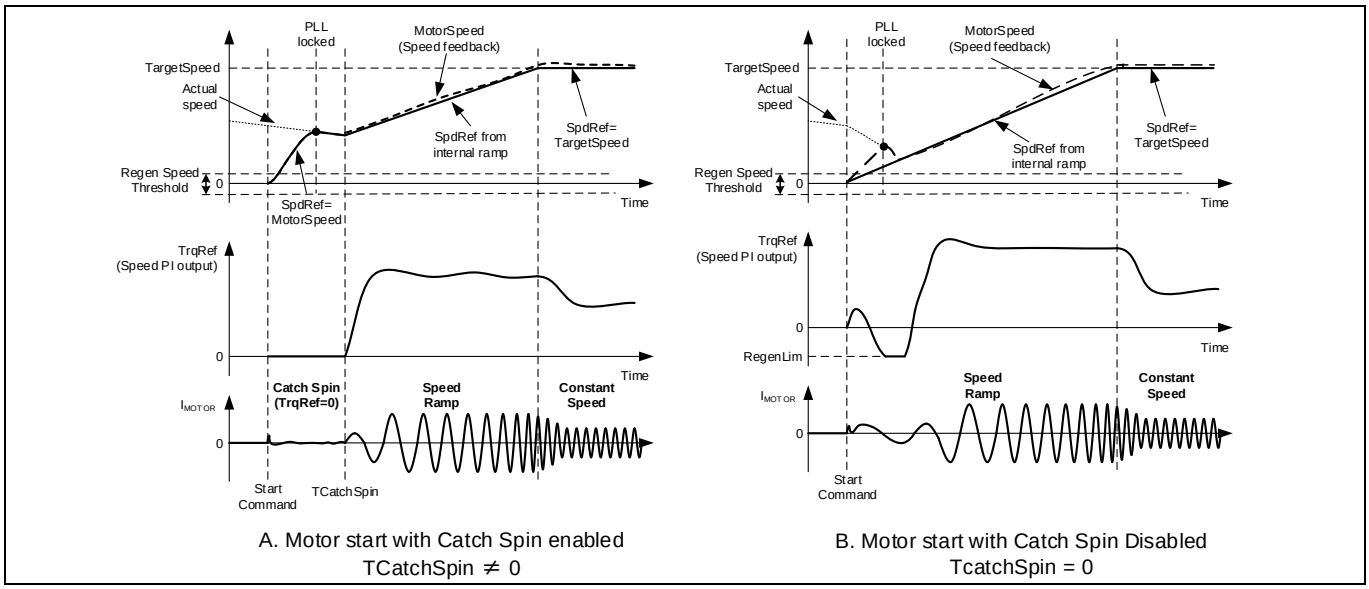


Figure 28 Forward Catch Spin - Motor start with/without catch spin

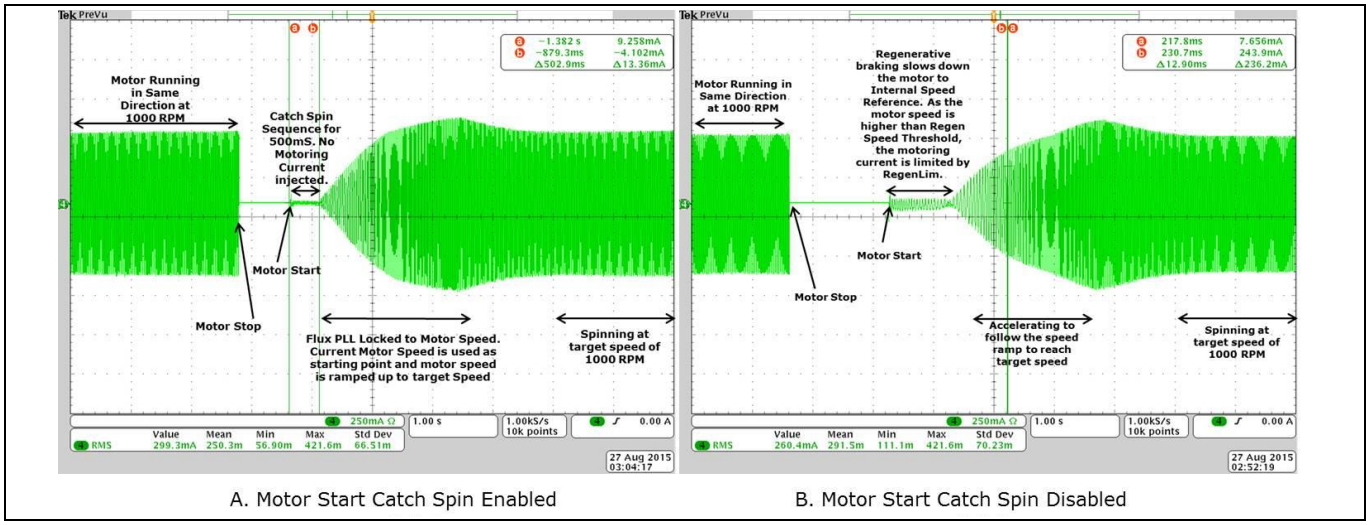


Figure 29 Motor Phase Current Waveform - Forward Catch Spin - Motor start with/without catch

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.

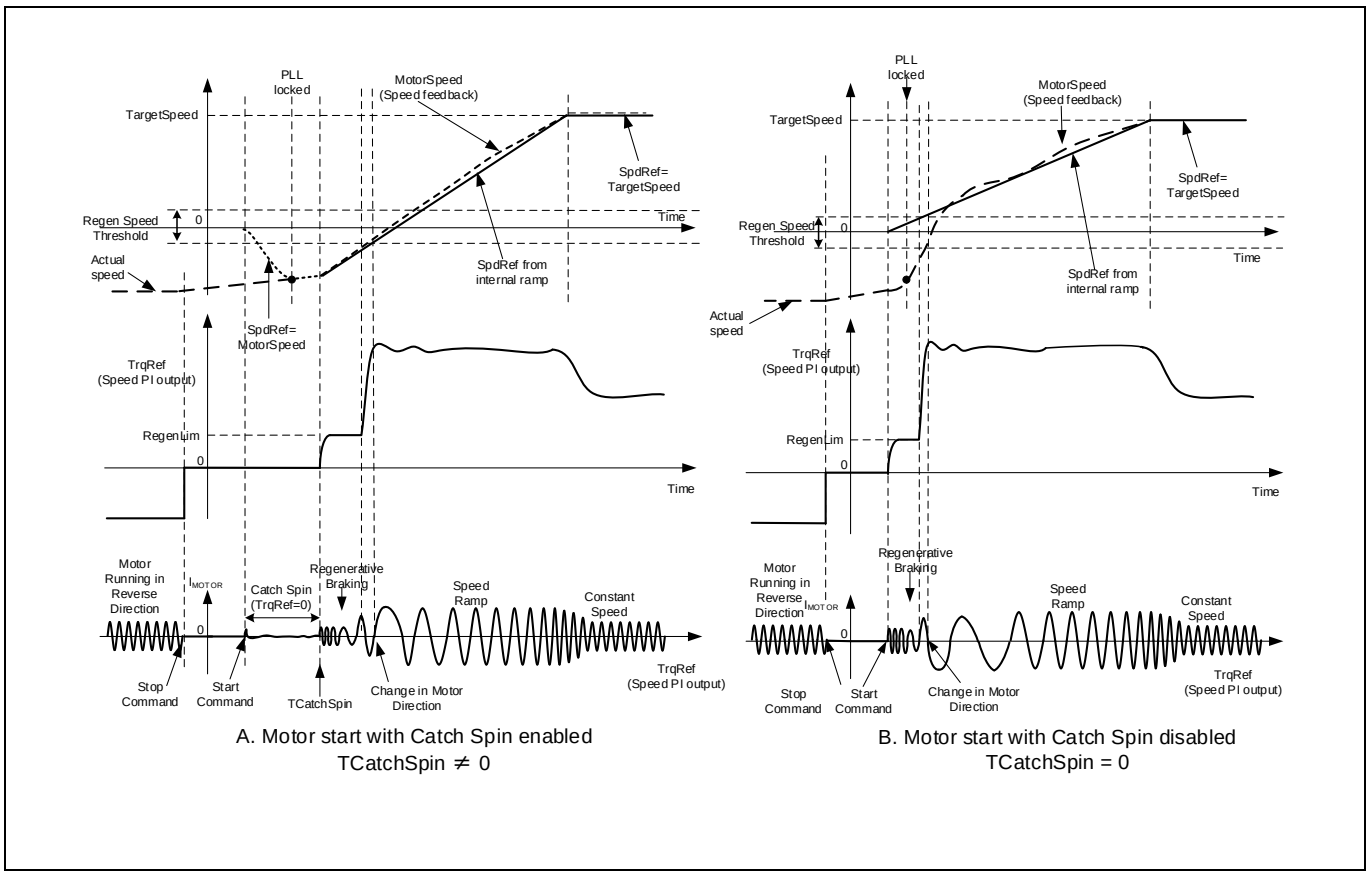


Figure 30 Reverse Catch Spin - Motor start with/without catch spin

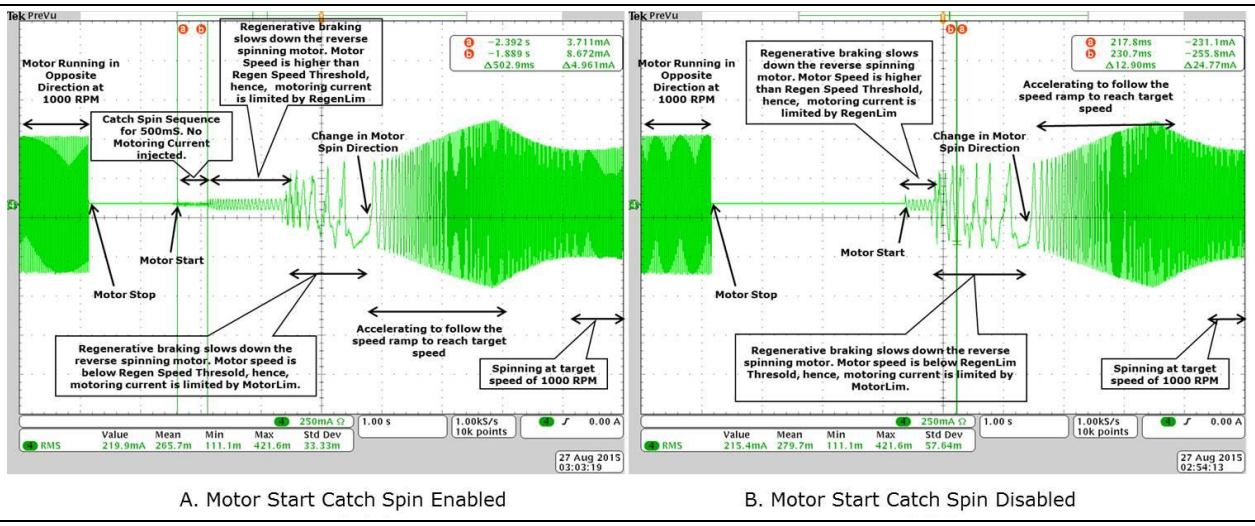


Figure 31 Motor Phase Current Waveform - Reverse Catch Spin - Motor start with/without catch spin

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 MCEWizard.

- 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.

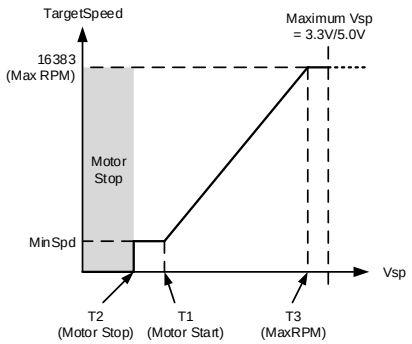


Figure 32 Vsp Analog Input

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.

MCEWizard 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 IMC 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

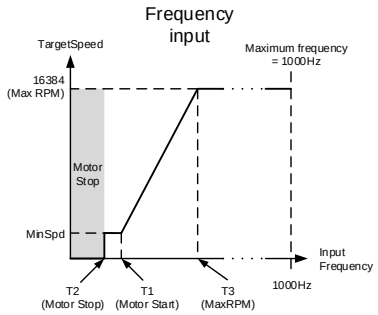


Figure 33 Frequency Input

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.

MCEWizard 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 IMC 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.

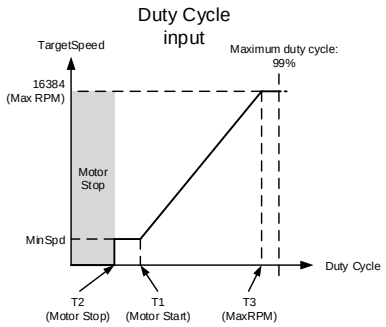


Figure 34 Duty Cycle Input

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.

MCEWizard 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 %.

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

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.

MCEWizard 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 IMC 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. ‘FaultRetryPeriod’ parameter is used to configure the number of retry times and 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 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 'ControlFreq' and 'ControlDuty' 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 'ControlFreq' variable using script.

For duty cycle input control method, the measured input duty cycle can be read from 'ControlDuty' 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 35, 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 [15:14] of parameter 'SysConfig') 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.

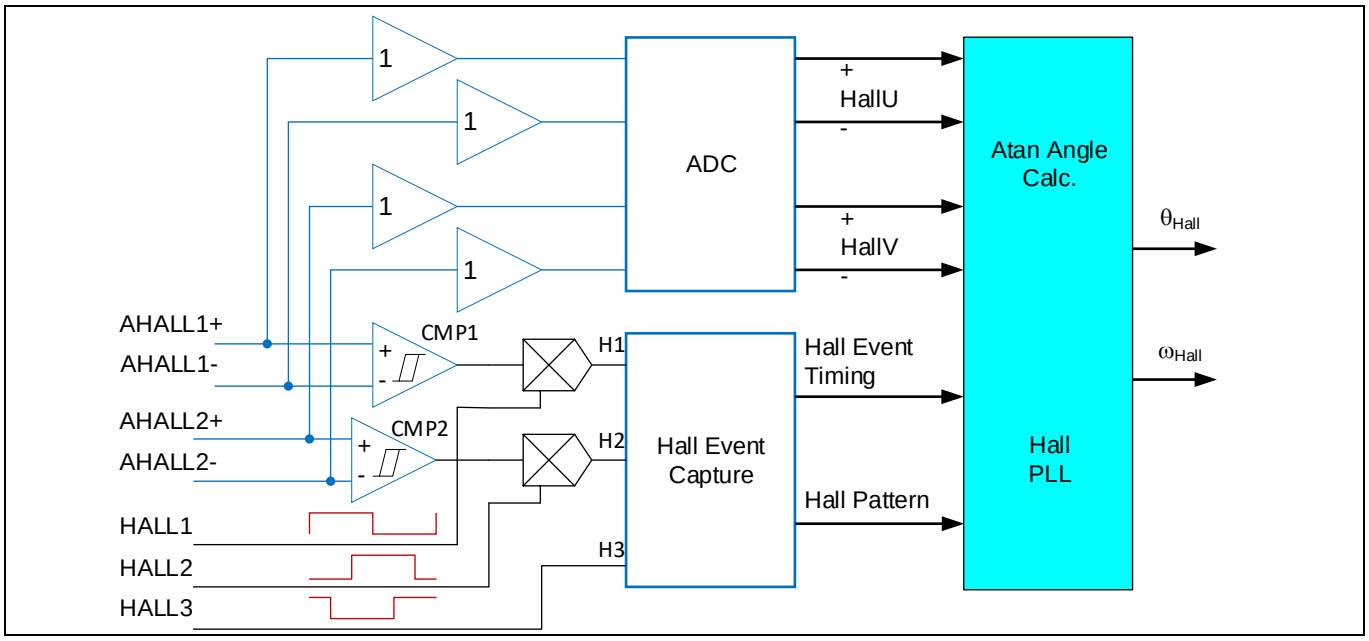


Figure 35 Hall Sensor Interface High-Level Structure Overview

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 36. 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 'HallSampleFilter' following this equation $T_{DB} = HallSampleFilter \times 10.417ns$.

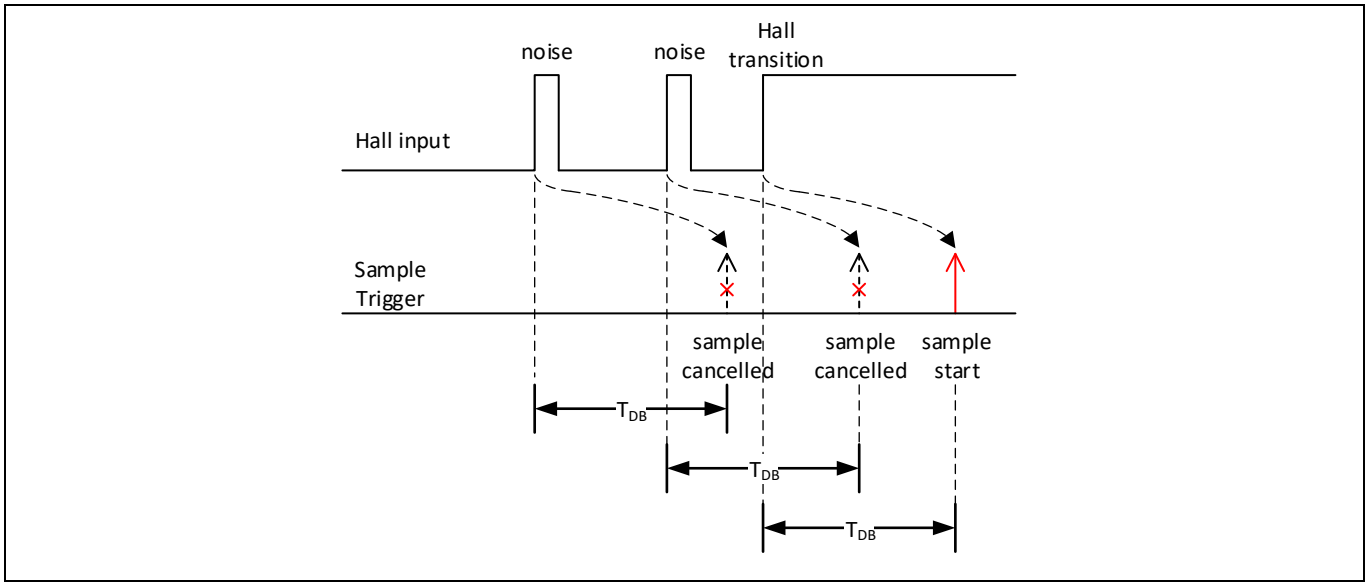


Figure 36 Hall Sensor Noise Filter Timing Diagram

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 37.

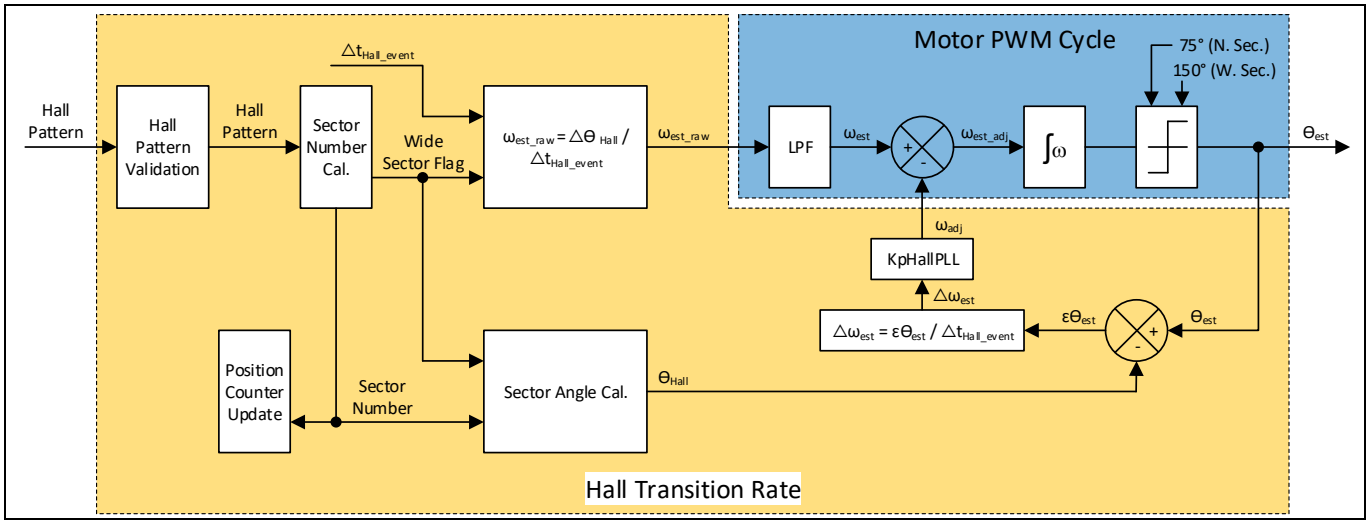


Figure 37 Hall Angle Estimation Algorithm Diagram (Hall PLL Enabled)

During each motor PWM cycle, the MCE's Hall angle estimation routine integrates the difference ω_{est_adj} between the low-pass filtered Hall frequency estimate ω_{est} (variable 'Hall_FrequencyOut') and a compensation term ω_{adj} (variable 'HallPLL_FrequencyAdjust') to generate the estimated Hall angle θ_{est} as shown in the blue block in Figure 37. 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 θ_{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 37.

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 (bit field [6:4] of the parameter 'HallStatus') is calculated based on a mapping relationship between Hall patterns and sector numbers as shown in Figure 39.

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 ω_{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 Δt_{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 Δt_{Hall_event} is longer than 4096 PWM cycles, then it is considered timed out and ω_{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 'HallSpdFiltBW': $HallSpdFiltBW = 2^{16} \cdot (1 - e^{-\frac{\omega_c \cdot Fast_Control_Rate}{F_{PWM}}})$. The filtered Hall frequency estimate ω_{est} (variable 'Hall_FrequencyOut') is available to be used for Hall angle estimation.

Next, the actual Hall angle θ_{Hall} is calculated based on the updated sector number and the rotating direction and wide sector flag. The estimated Hall angle θ_{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 ω_{adj} to the Hall frequency integrator ove the next Hall transition event cycle. The frequency compensation term ω_{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 θ_{est} is reset to the value of the actual Hall angle θ_{Hall}, and the compensation term ω_{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 'HallMotorSpeed' is updated from the product of the low-pass filtered Hall frequency estimate ω_{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 θ_{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 38.

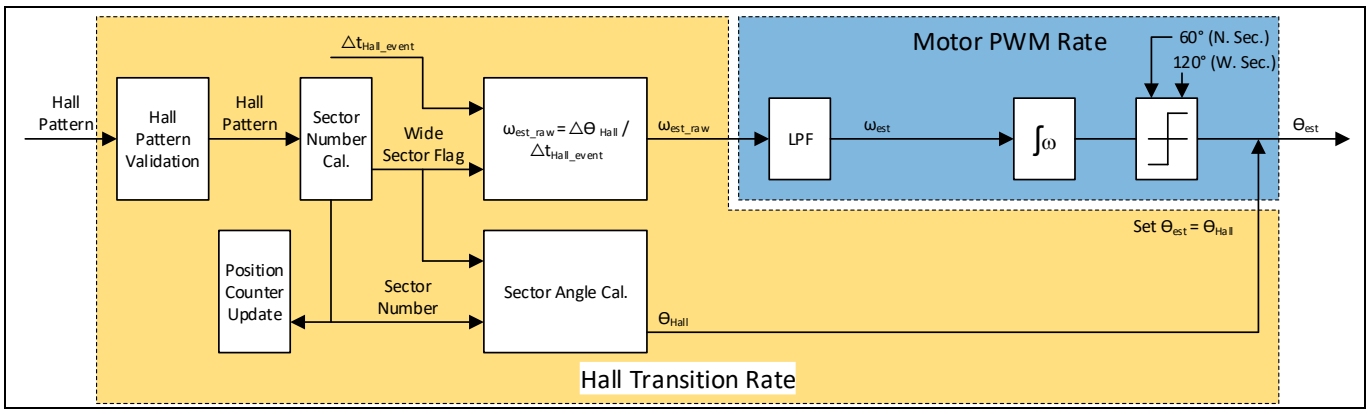


Figure 38 Hall Angle Estimation Algorithm Diagram (Hall PLL Disabled)

During each motor PWM cycle, the MCE's Hall angle estimation routine integrates the low-pass filtered Hall frequency estimate ω_{est} to generate the estimated Hall angle θ_{est} as shown in the blue block in Figure 38. 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 θ_{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 38.

Hall pattern validation, sector number calculation, position counter update, Hall frequency estimate calculation, actual Hall angle calculation, the variable 'HallAngle' and 'HallMotorSpeed' 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 θ_{est} at each Hall transition event. In other words, the estimated Hall angle θ_{est} is reset to the actual Hall angle θ_{Hall} at each Hall transition event.

With Hall PLL disabled, when the motor is accelerating or decelerating, the estimated Hall angle θ_{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, and accordingly, bit [2] of the variable 'HallStatus' is asserted. 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: $T_{HallTimeOut} = HallTimeoutPeriod \times 10ms$.

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 39. 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.

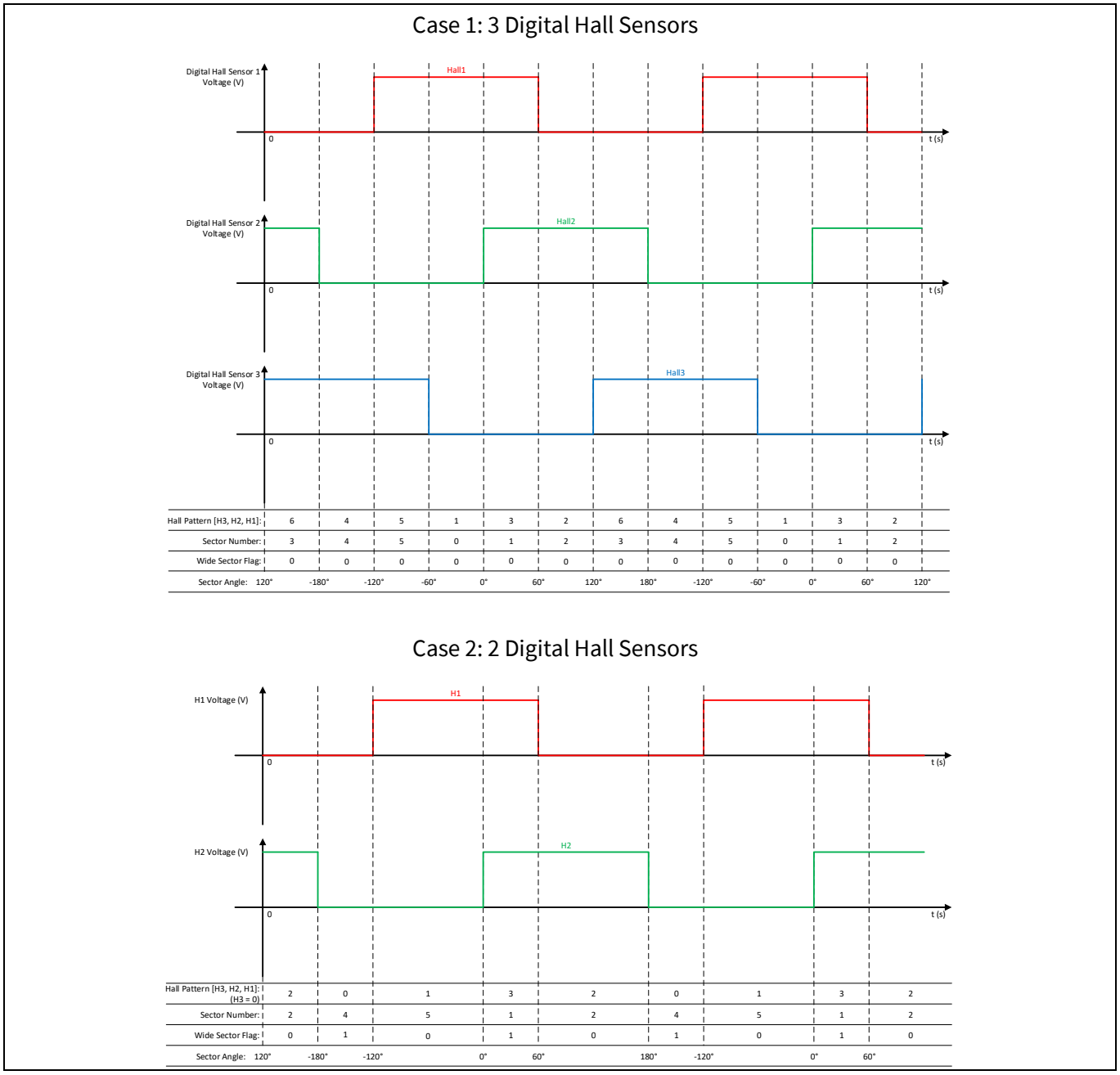


Figure 39 Calculation of Hall Pattern, Sector Number, Wide Sector Flag, and Sector Angle from Hall Inputs

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 39.

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 θ_{offset} as shown in the following Figure 40.

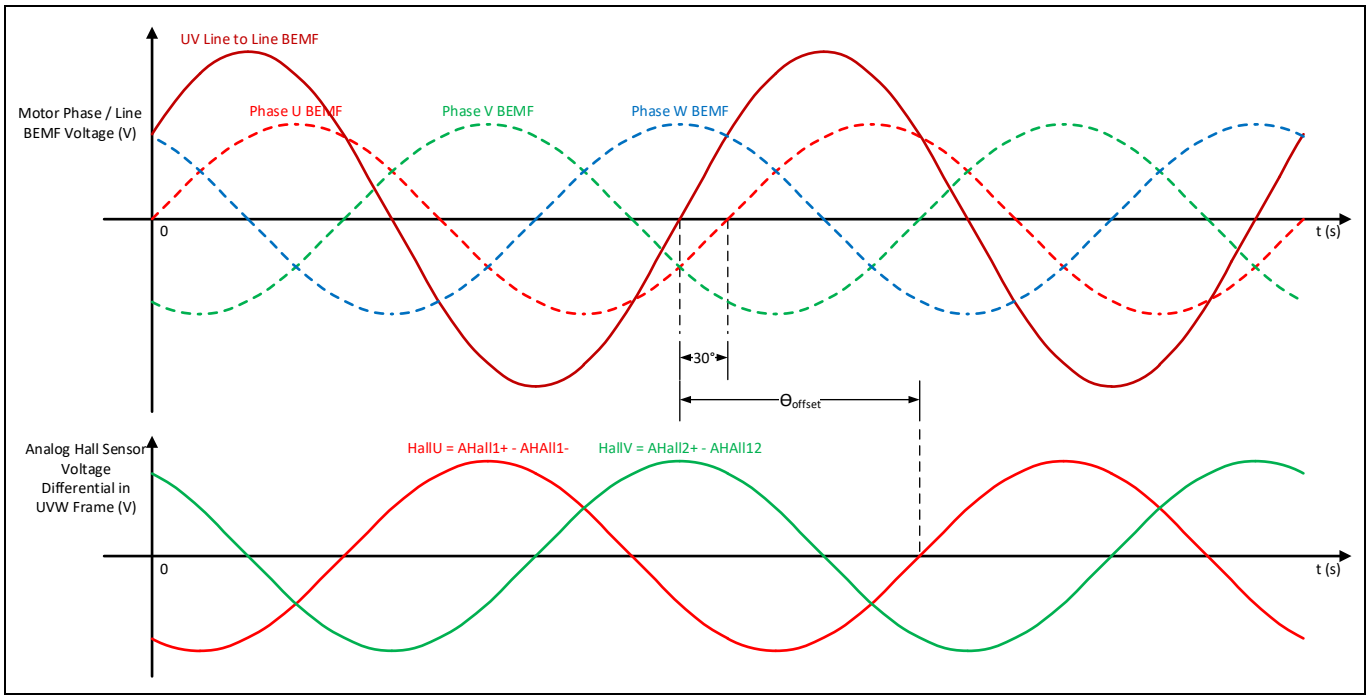


Figure 40 Angle Offset Definition Diagram for 2 Analog Hall Sensor Configuration

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 θ_{offset} as shown in the following Figure 41.

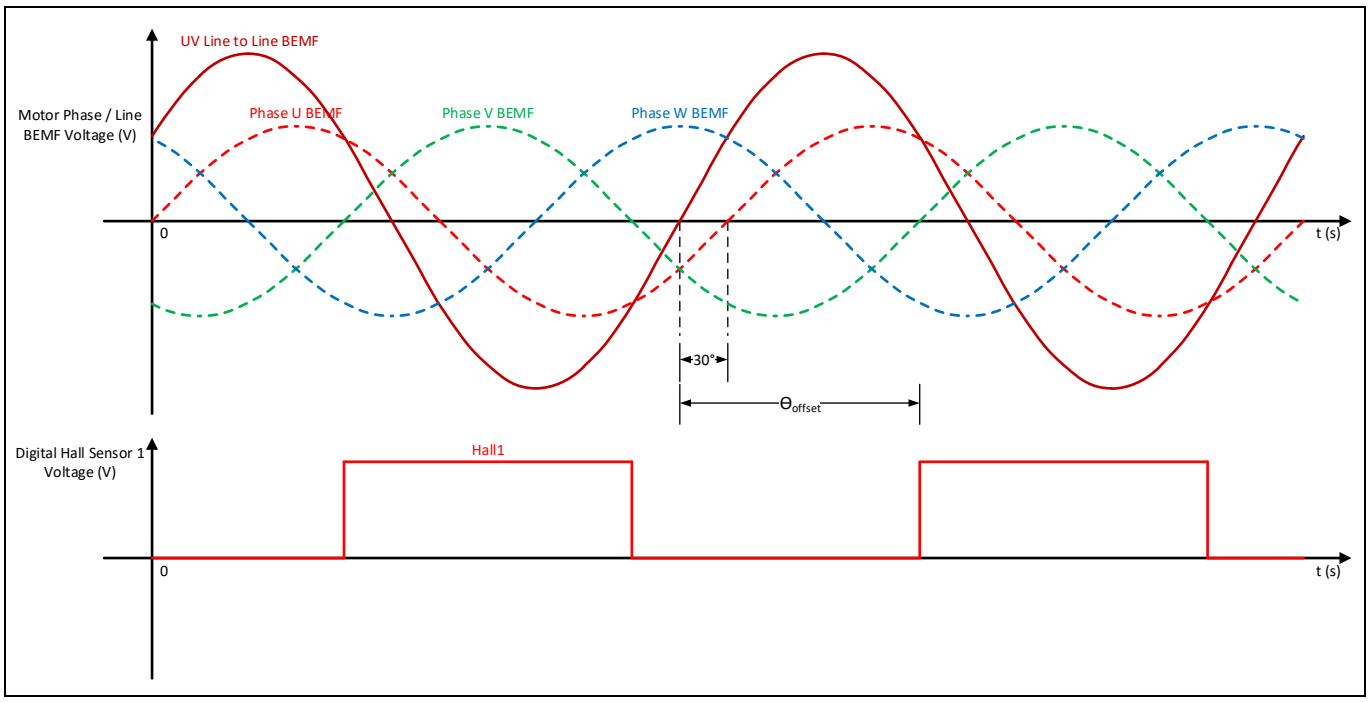


Figure 41 Angle Offset Definition Diagram for 2 or 3 Digital Hall Sensor Configuration

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 13th bit of the parameter 'SysConfig'. When Atan angle calculation method is enabled and Hall angle or hybrid angle is selected by the parameter 'AngleSelect', users can use the higher 8 bit of the parameter 'APPConfig' to specify the number of sectors for which Hall Atan angle represented by the parameter 'Hall_Atan_Angle' is being used as rotor angle during start-up.

The Atan angle calculation process is shown in the following Figure 42. 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 $\mathrm{Atan}(\frac{Hall_\beta}{Hall_\alpha})$ calculation is

performed to generate Atan angle represented by the variable ‘Hall_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 higher 8 bit of the parameter ‘APPConfig’.

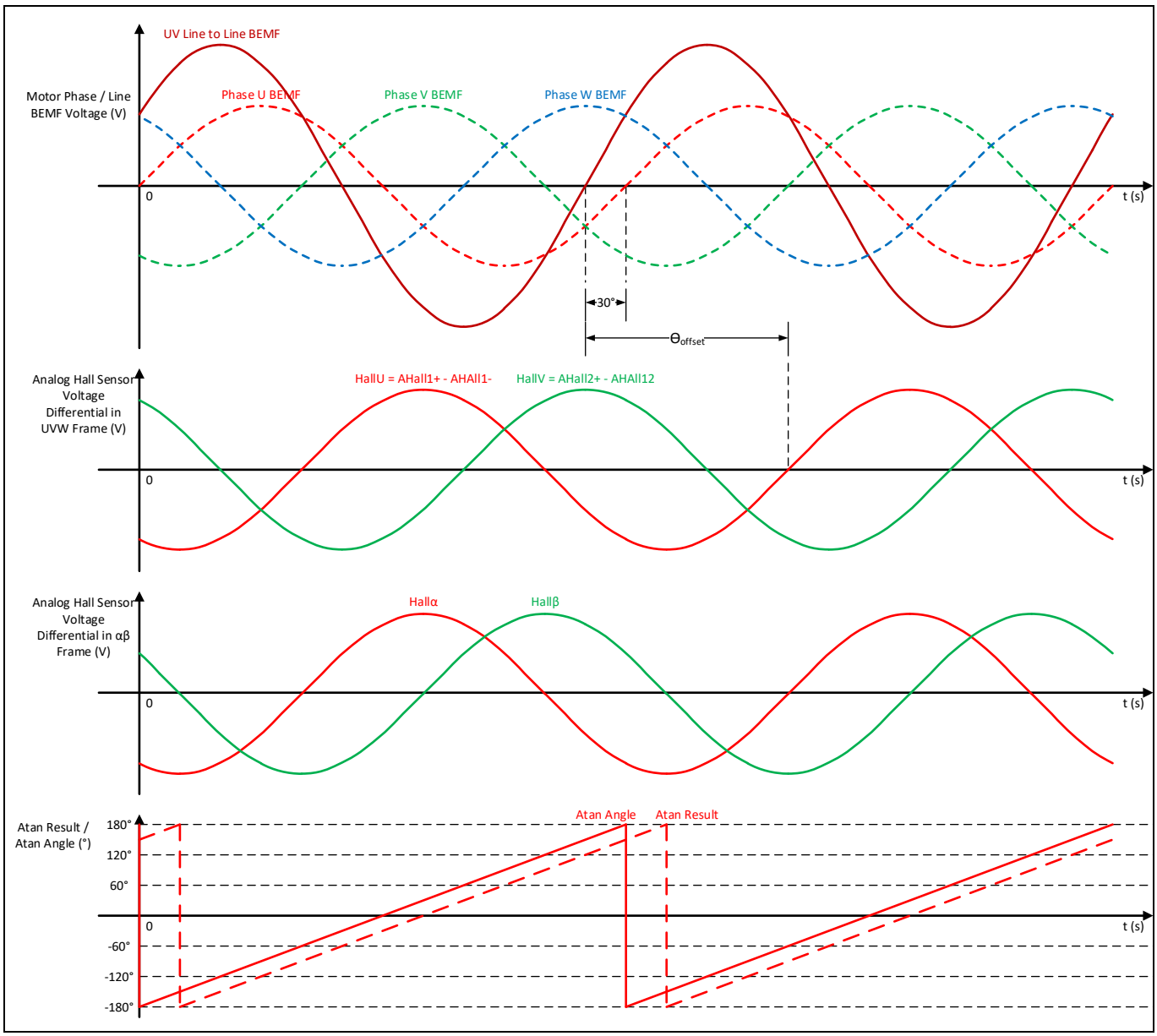


Figure 42 Atan Angle Calculation Based on 2 Analog Hall Sensor Inputs (AHall1+, AHall1-, AHall2+, AHall2-)

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 / Sensor-less 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 43, 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 sensor-less 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 sensor-less option in providing superior start-up and low speed performance. Thus, it is recommended to select hybrid mode to take advantage of both the sensor-less mode and the Hall sensor mode to ensure a consistent high performance of a drive system across a wide speed range including start-up.

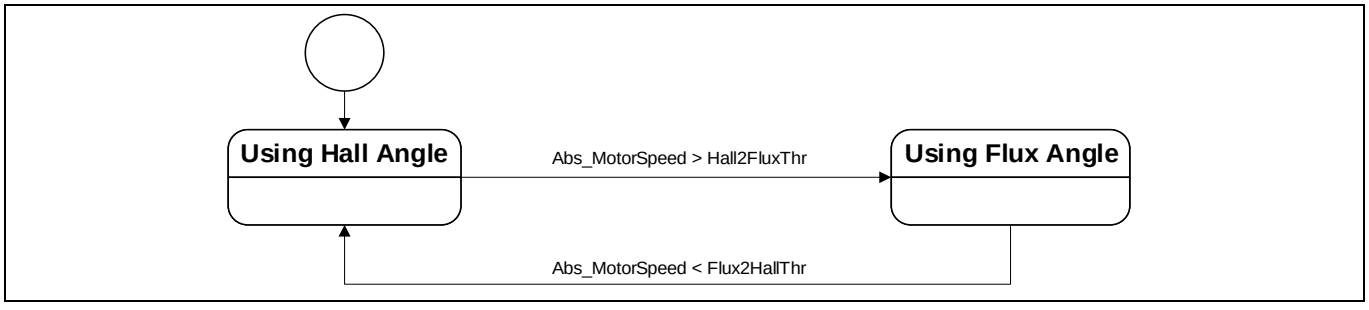


Figure 43 Hall Sensor / Sensor-less Hybrid Mode Diagram

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 sensor-less 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 [4] of the parameter 'AppConfig'.

Figure 44 depicts the state transitions for the MCE's torque compensation function.

When torque compensation is disabled by resetting bit [4] of the parameter 'AppConfig', it stays in TC_Disabled state and goes through an initialization process (TorqueComp_init()) where relevant variables including 'TrqCompBaseAngle', 'TrqCompStatus' and 'TrqRef_Ext' are reset.

When torque compensation is enabled by setting bit [4] of the parameter 'AppConfig', it shift to TC_Enabled state.

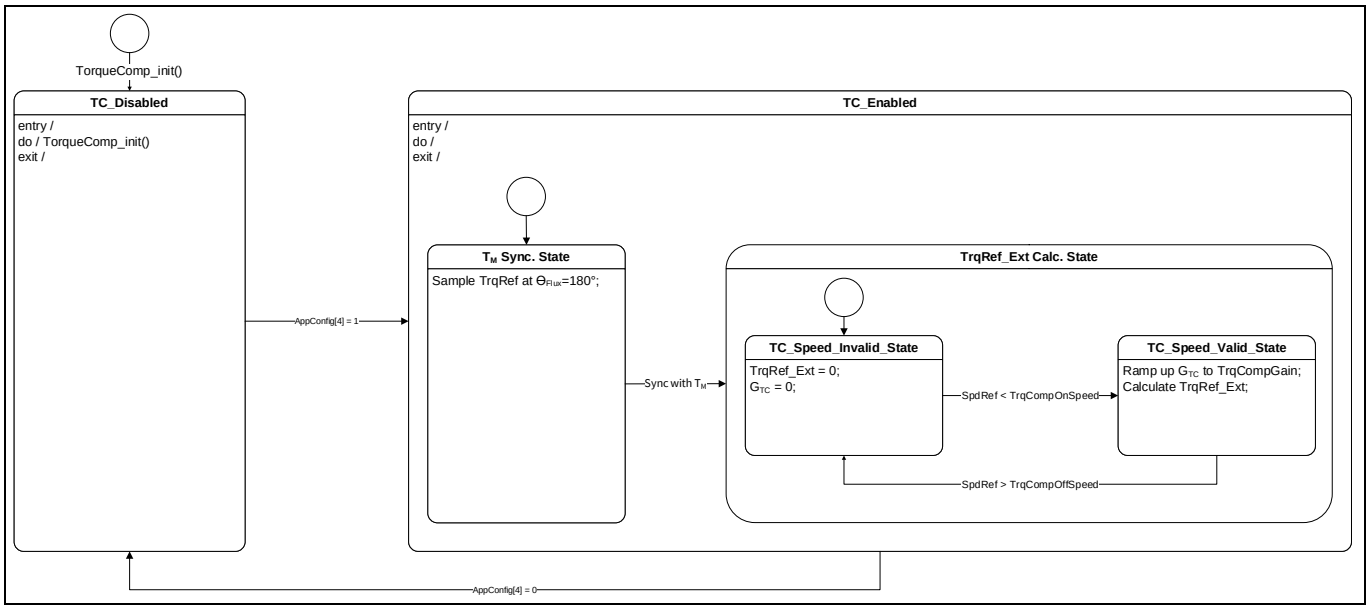


Figure 44 Torque Compensation State Transition Diagram

As shown in Figure 44 and Figure 45, 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 'TrqRef') 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: TrqRef[k] > TrqRef[k-1], TrqRef[k] > TrqRef[k-2], TrqRef[k-3] > TrqRef[k-1], TrqRef[k-3] > TrqRef[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 θ_{base} (variable 'TrqCompBaseAngle'). Then it shifts to TrqRef_Ext Calculation sub-state.

There are two sub-states inside TrqRef_Ext Calculation sub-state. If the motor speed reference (variable 'SpdRef') 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 $TrqRef_Ext$ 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 'TrqRef_Ext') is synthesized following this equation:

$$TrqRef_Ext = G_{TC} \times TrqRef_{Filt} \times \cos(\frac{\theta_{Flux}-180^{\circ}}{pole_pair}+\theta_{base}+\theta_{os}) \times k_{CORDIC}.$$

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

$TrqRef_{Filt}$represents the averaged value of the desired torque reference from speed PI regulator output. It is the low-pass filtered result from variable 'TrqRef' with upper limit. As shown in Figure 45, if $TrqRef_{Filt}$ is greater than the value of the parameter 'TrqCompLim', then it is limited to the value of 'TrqCompLim'.

k_{CORDIC} is the CORDIC gain factor. $k_{CORDIC} = 1.647$.

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 θ_{base} increments by 120° (6-pole) or 180° (4-pole) every electrical cycle.

The torque compensation angle offset θ_{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 45, the synthesized compensation torque reference 'TrqRef_Ext' is summed up with 'TrqRef' to form total torque reference (variable 'TrqRef_Total'), which is used in the following IPM 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 [4] of the parameter 'AppConfig'.

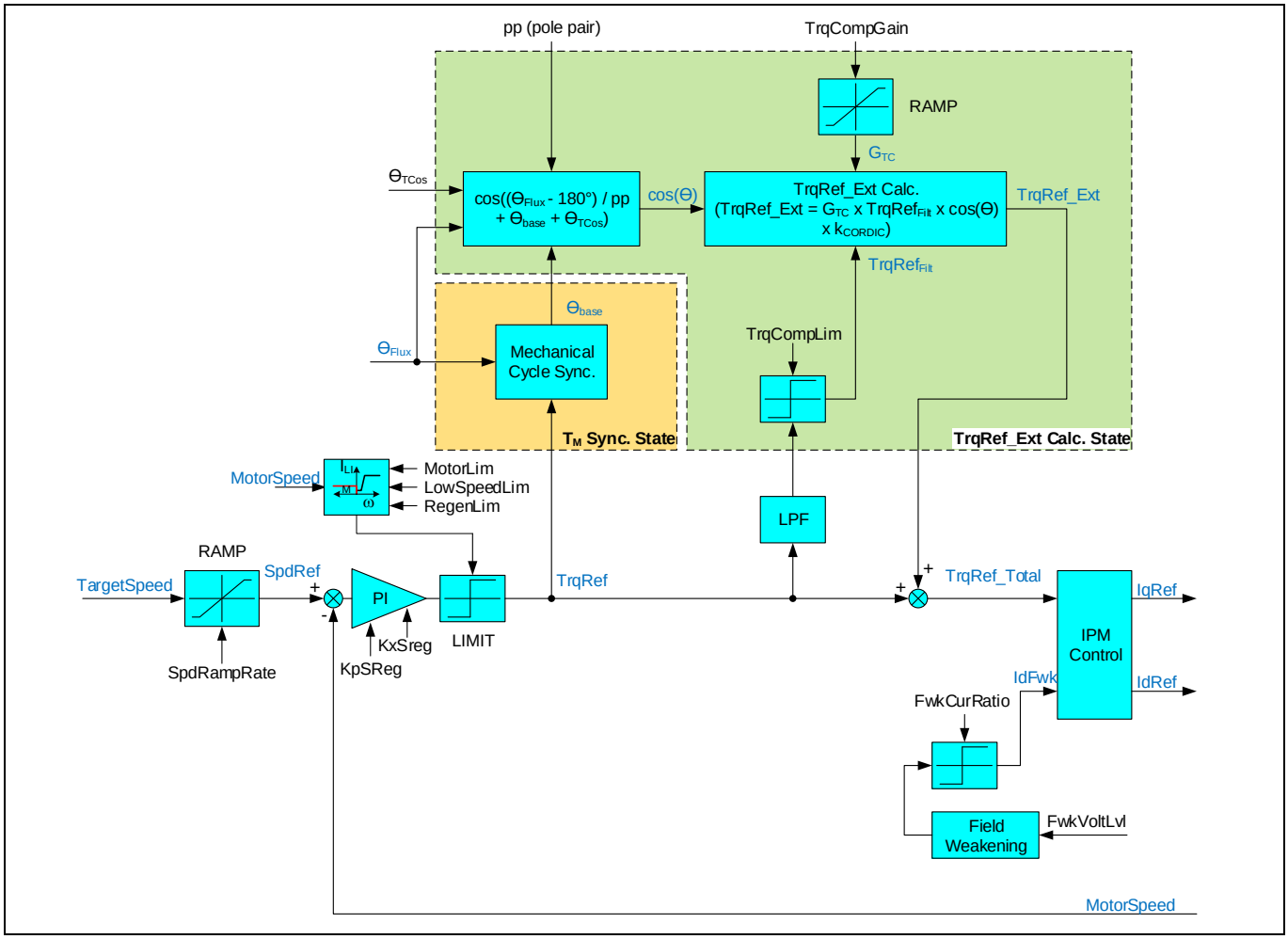


Figure 45 Torque Compensation Top-Level Algorithm Diagram

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 [4] of the parameter 'AppConfig' to enable torque conpensation function.

d) Use tracing function in MCEDesigner 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, Flx_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, Flx_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.

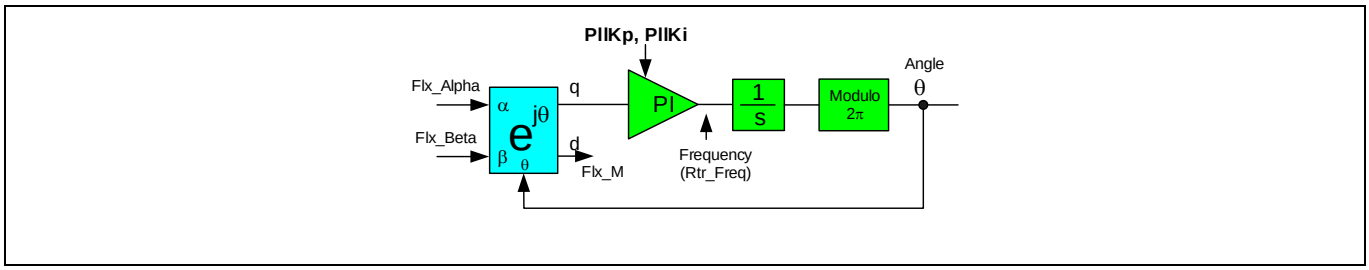


Figure 46 Simplified block diagram of a Flux PLL

The MCE keeps monitoring Flx_M, within certain time slot (configured by 'FluxFaultTime' parameter), if Flx_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 47 for details.

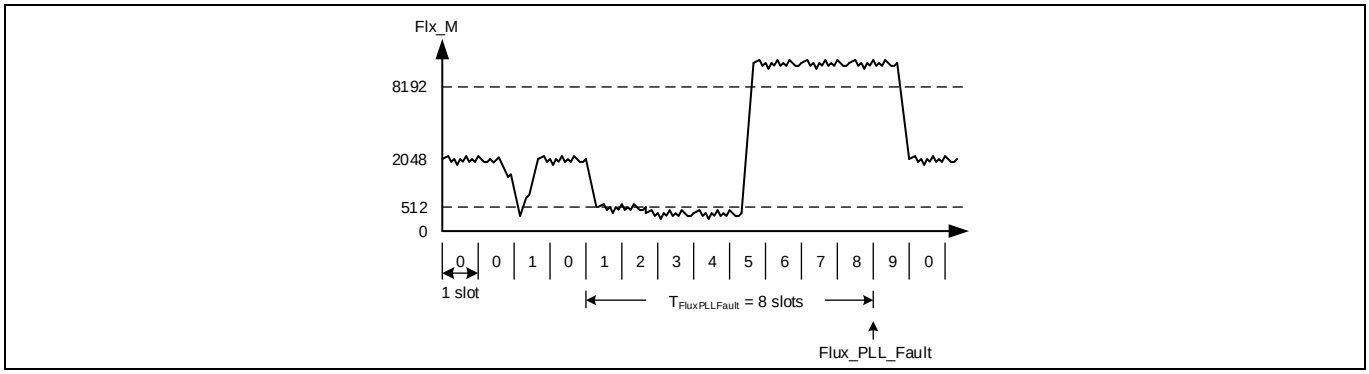


Figure 47 Flux PLL Out-of-Control Protection

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 0.01 seconds. The default value is set to 800, which corresponds to a response time of 8 seconds.

2.1.16.2 Rotor Lock Protection

As shown in the following Figure 48, 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 $T_{Rotor_Lock} = RotorLockTime \times 10ms$. 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.

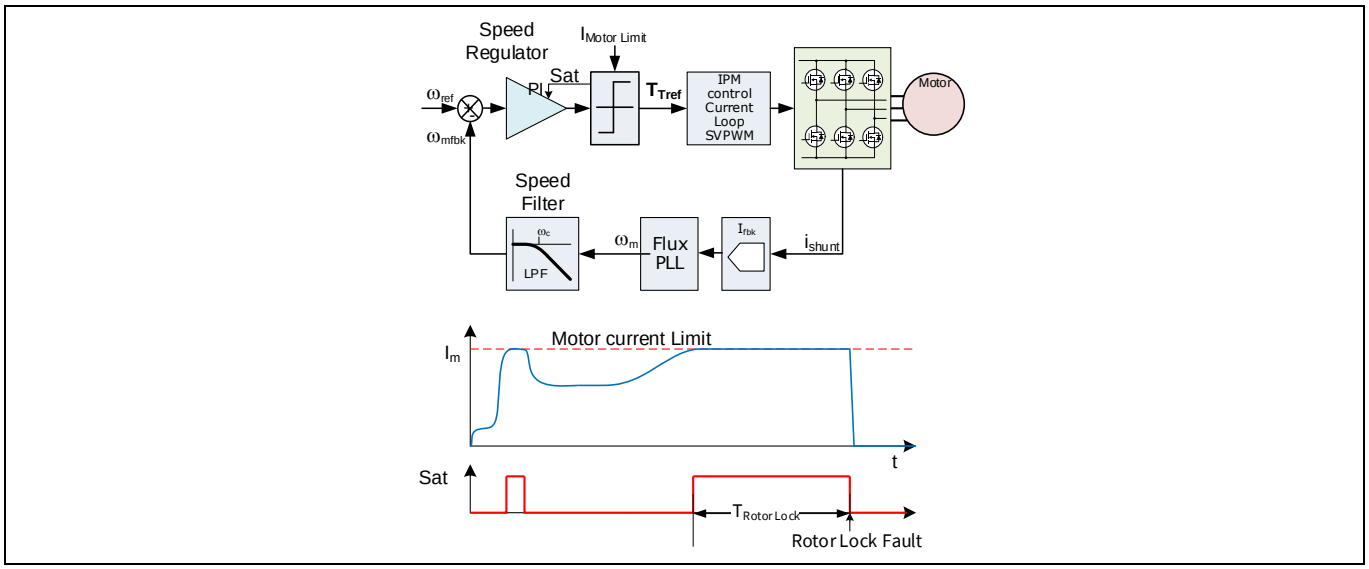


Figure 48 Rotor Rock Protection Mechanism Diagram

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 MCEWizard.

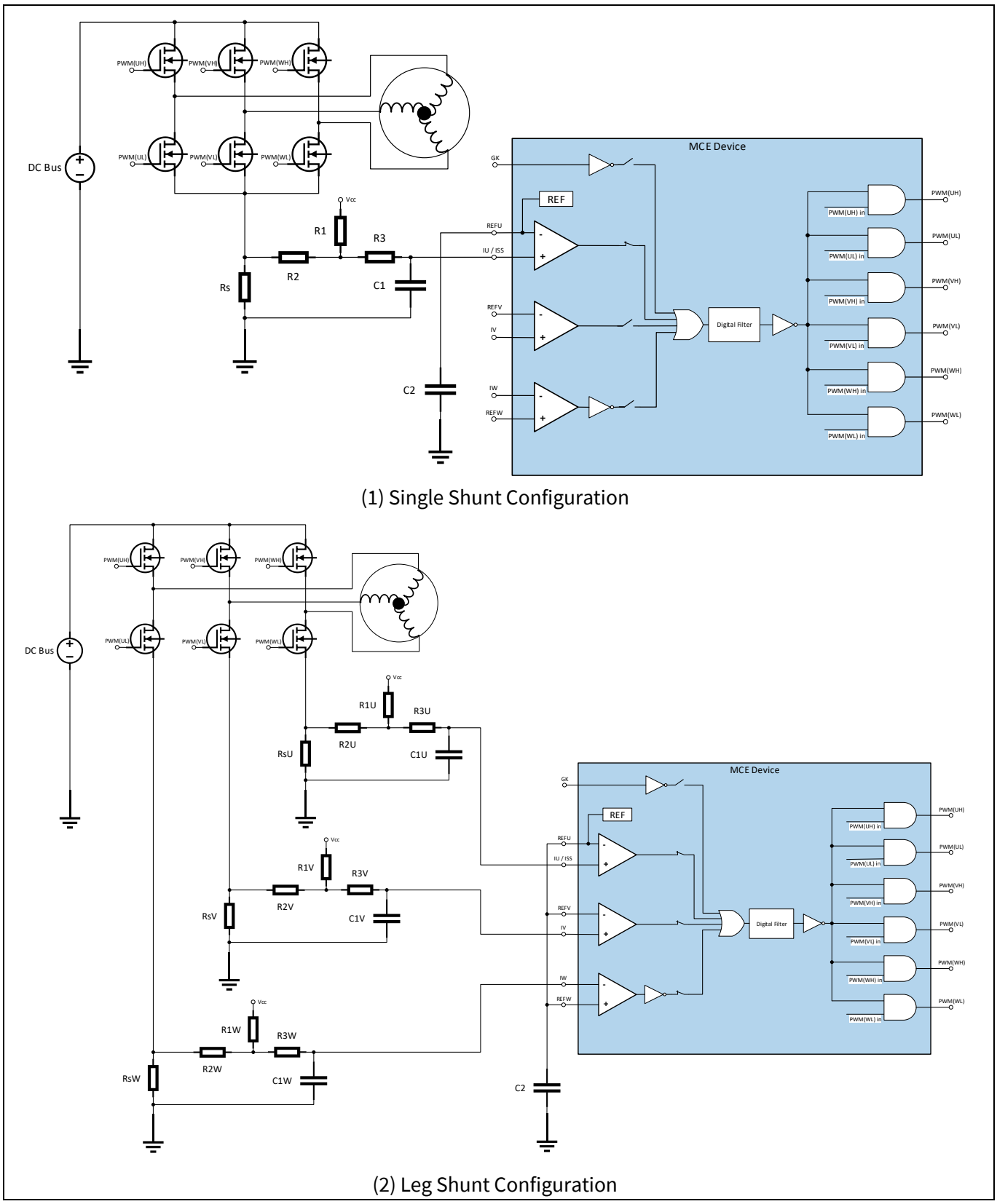


Figure 49 Typical Motor OCP Implementation Using Internal Comparators

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 MCEWizard, the ‘CompRef’ parameter holds the current trip level value. As shown in Figure 49 (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 49 (2).

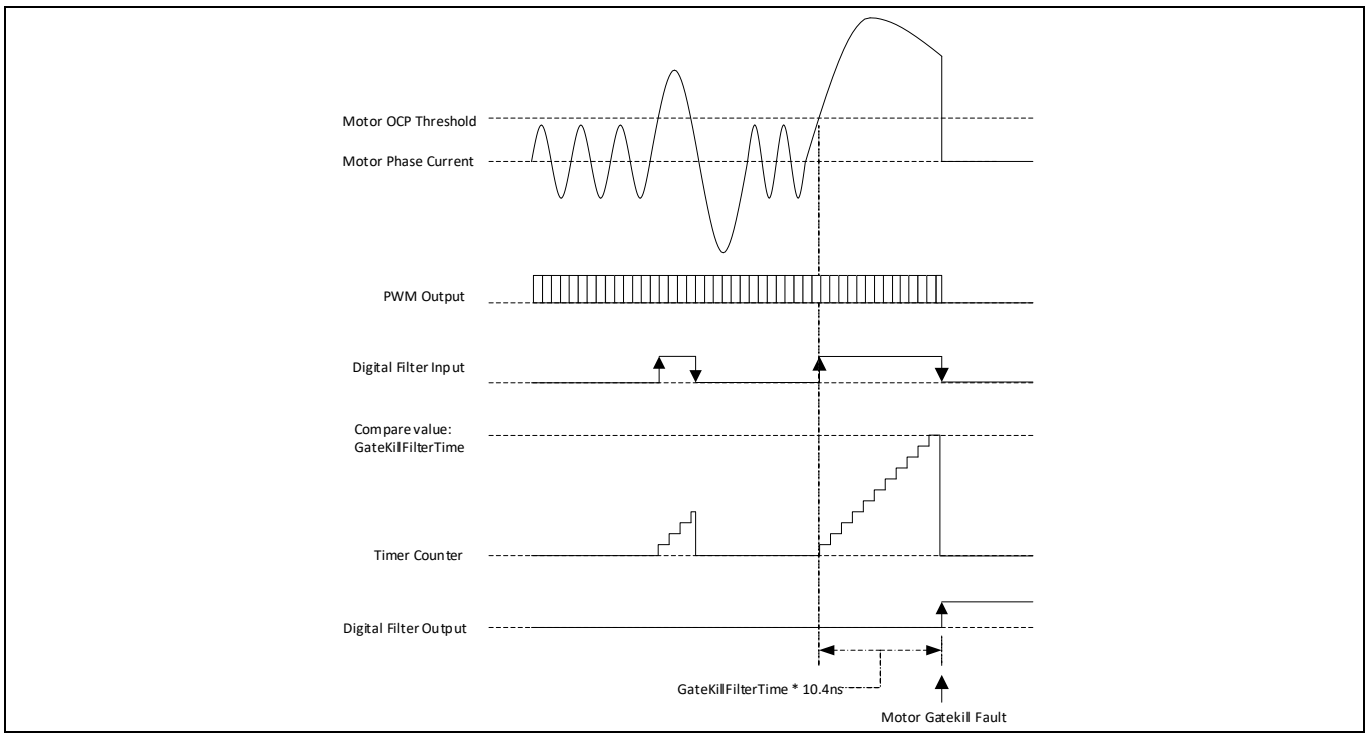


Figure 50 Digital Filter Timing Diagram for Motor Gatekill Fault

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 50, 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 51, 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’.

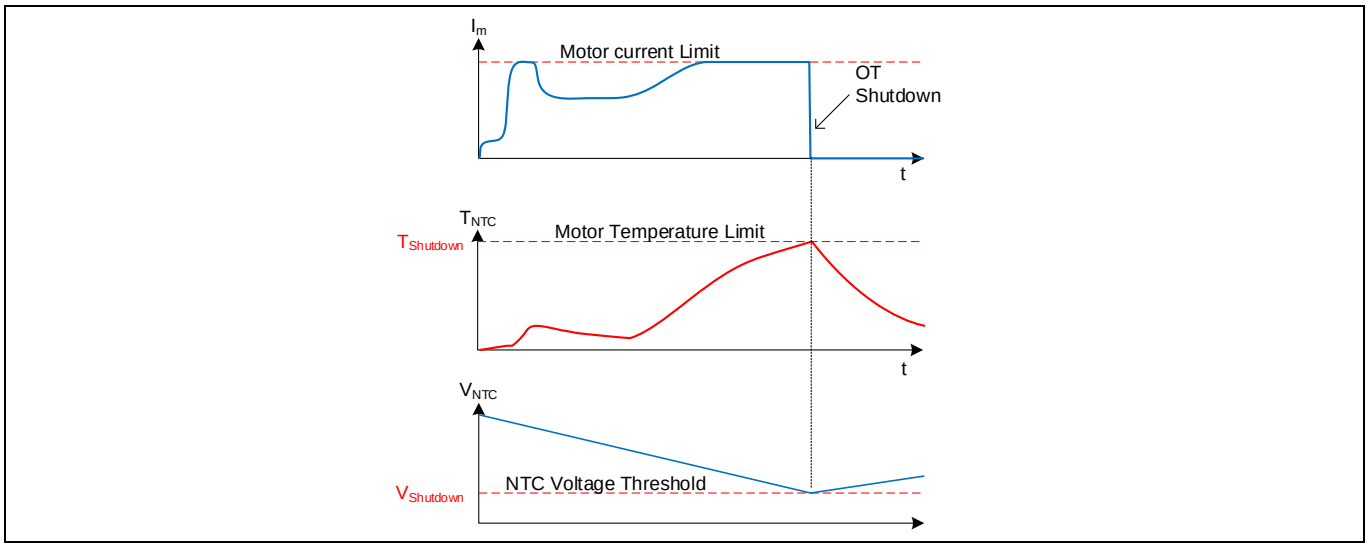


Figure 51 Over-Temperature Protection Mechanism Diagram

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.

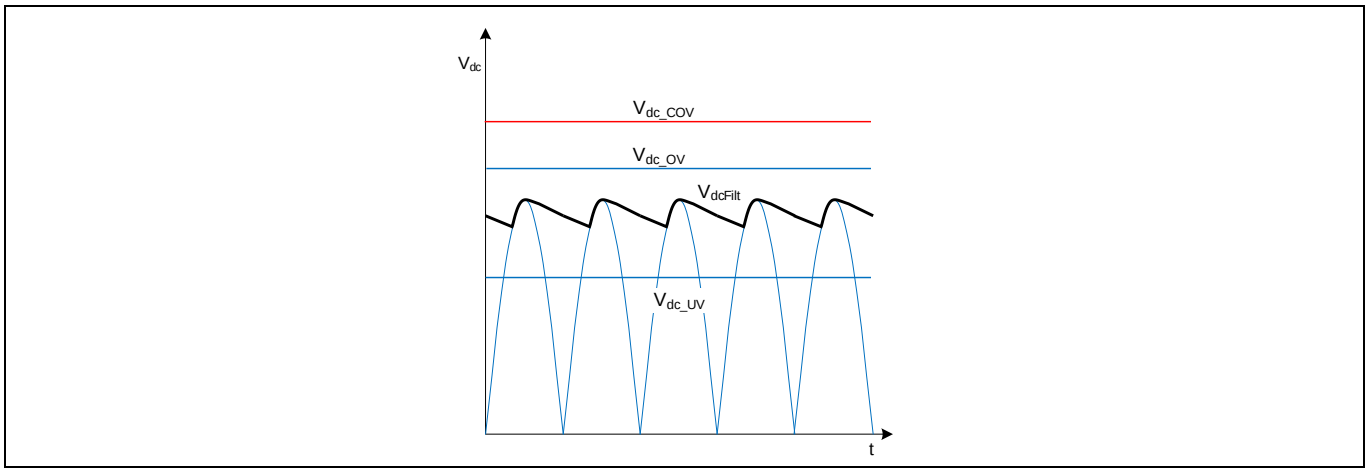


Figure 52 DC Bus Over / Under Voltage Protection Threshold Diagram

As shown in Figure 52, 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 'CriticalOvLevel'), 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 MCEWizard 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 Execution Fault

Two conditions lead the MCE to detecting an Execution Fault. Firstly, the MCE monitors CPU execution load and if it exceeds 95%, Execution Fault becomes active. Sencondly, if any of the background tasks (UART interface, Control input and script Task1) are not executed at least once every 60s, Execution Fault becomes active.

The MCE always reports Execution Fault by setting bit[10] of the parameter 'FaultFlags'. Furthermore, if bit[10] of the parameter 'FaultEnable' is set, Execution Fault will be reflected in the parameter 'SwFaults' and the motor state machine will shift to FAULT state causing the motor to stop running when the fault is detected. If bit[10] of 'FaultEnable' is not set, the corresponding bit of 'SwFaults' will be masked and and the motor state machine will not shift to FAULT state.

The user can configure the controller to reset when any of the background tasks are not executed for 60s. To enable reset, set bit[8] (enable background task monitor reset) of the "SysTaskConfig" parameter.

The MCE supports Scripting based controller reset which makes it possible to reset the controller with timing different from 60s. Reset from scripting is performed by setting "Script_command = 0xAE51" The reset command can be issued from both Task0 and Task1. However, since Task1 background function itself, it cannot be relied on to monitor exectution background task. The reset command must be issued from Task0.

Note, if safety function is enabled and any of background tasks are not executed for 1s, the MCE enters failsafe mode.

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. Regulations will specify the condition at which to demonstrate the efficiency of the PFC.

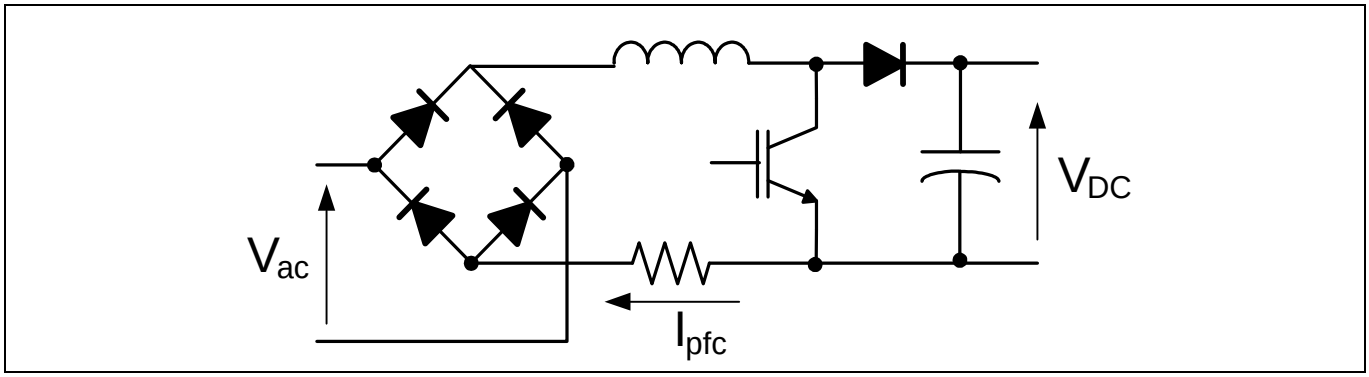


Figure 53 Basic Boost PFC Circuit

Above figure shows the simplified circuit of the boost PFC topology.

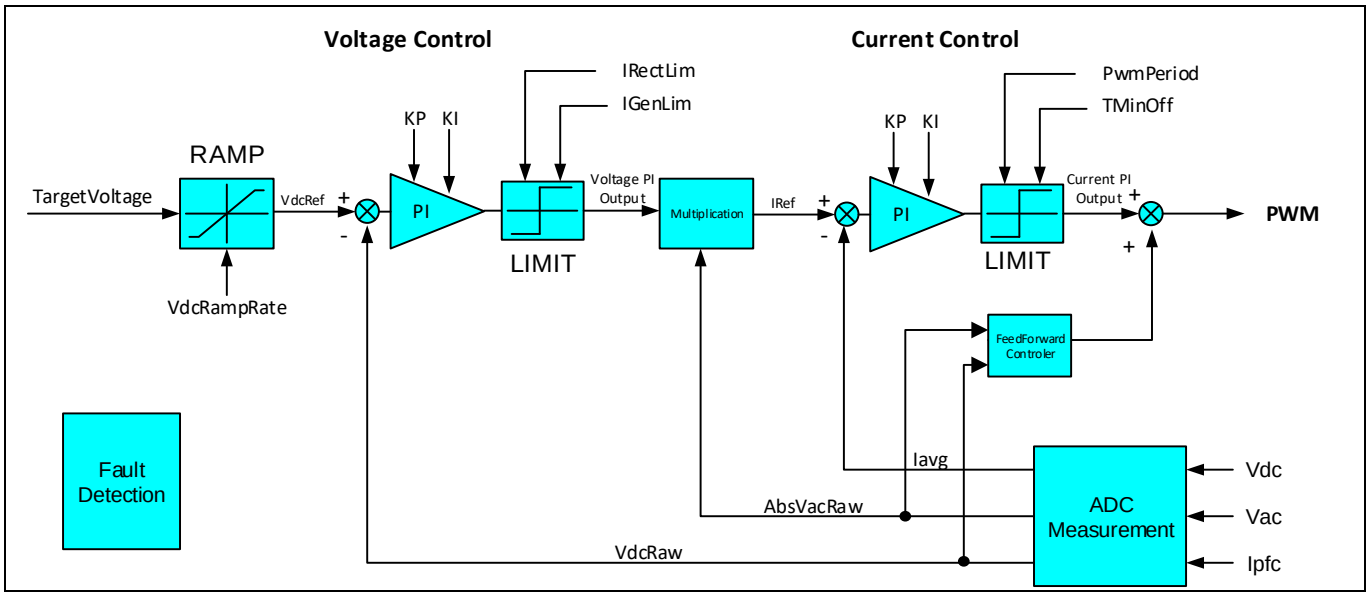


Figure 54 Top Level Diagram of Power Factor Correction

MCE PFC is multiplier based control, which means there are two control loops in PFC, an inner current loop and an outer voltage loop, along with a feedforward component. The output of the voltage controller is multiplied by the rectified AC voltage to produce a current reference. The output of the current controller is added to the feedforward output to generate the modulation command. This PFC control scheme requires sensing of the inductor current, AC line voltage and DC bus voltage.

MCE supports two types of PFC topologies.

1. Boost Mode PFC
2. Totem-Pole PFC

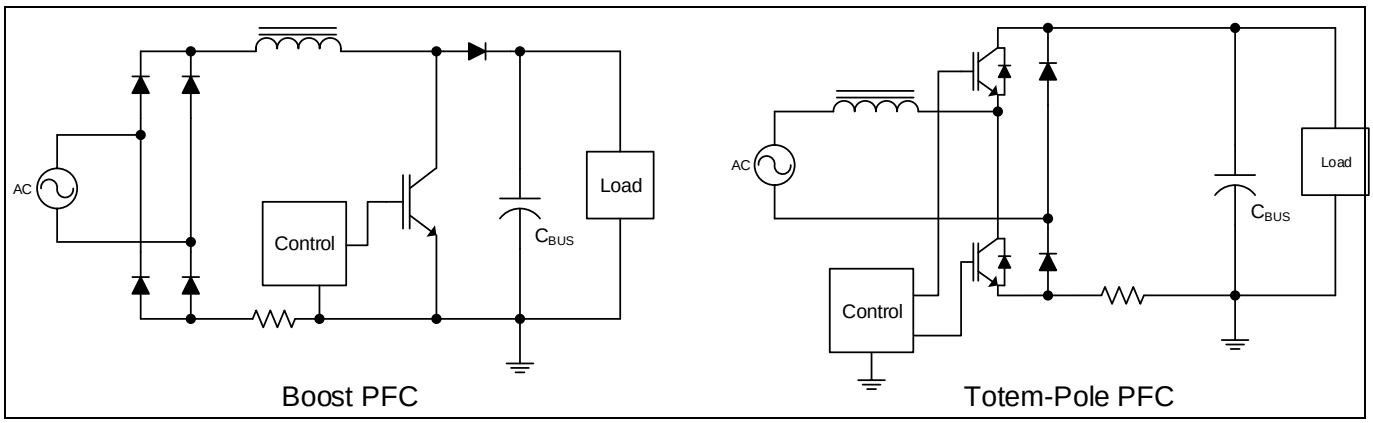


Figure 55 PFC topologies

Boost PFC is most common PFC topology because it's easy to control. Boost topology is not very efficient due to high losses on bridge diodes. There are some bridgeless designs which are targeting to reduce the bridge diode losses, but most of the bridgeless PFC solutions suffer from EMI issue which makes it impossible to be used in appliance application such as inverter air-conditioner. Totem pole PFC is a type of bridgeless PFC but it doesn't have EMI issue. With development of fast IGBT and commercial availability of high bandgap switches such as SiC and GaN, totem pole PFC attracts more attention as a candidate to replace traditional boost PFC.

It is challenging to design a totem pole PFC control circuit without using expensive sensors for AC voltage and inductor current sensing. The nature of totem pole PFC topology decides it needs more complicated control circuit compare to boost PFC. The main target of MCE totem-pole design is to minimize complexity regarding hardware of control circuit. It uses differential sensing for AC voltage and uses single shunt resistor on DC link for inductor current sensing. There is no additional hardware to detect AC voltage polarity. Digital control also makes it possible to re-construct the inductor current information from single shunt on DC link.

2.2.1 State Handling

Motion Control Engine (MCE) includes a built-in state machine which takes care of all state-handling for starting, stopping and performing start-up. A state machine function is executed every 1ms. Totally there are 5 states. Current state of sequencer is stored in “PFC_SequencerState” variable.

Table 11 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, control stay in this state.	Parameters are loaded successfully.	STOP
1	STOP	Wait for start command. Current and voltage measurement for protection	Current Amplifier offset calculation is not done.	OFFSETCAL
			Start Command.	RUN
2	OFFSETCAL	Offset calculation for PFC current sensing input. This state takes 256 PWM cycles.	Current offset calculation completed.	STOP
4	RUN	Normal PFC run mode	Stop Command	STOP
5	FAULT	If any fault detected, PFC will be stopped (if it was previously running) and enter FAULT state from any other state.	Fault clear command by writing 1 to “PFC_FaultClear” variable	STOP

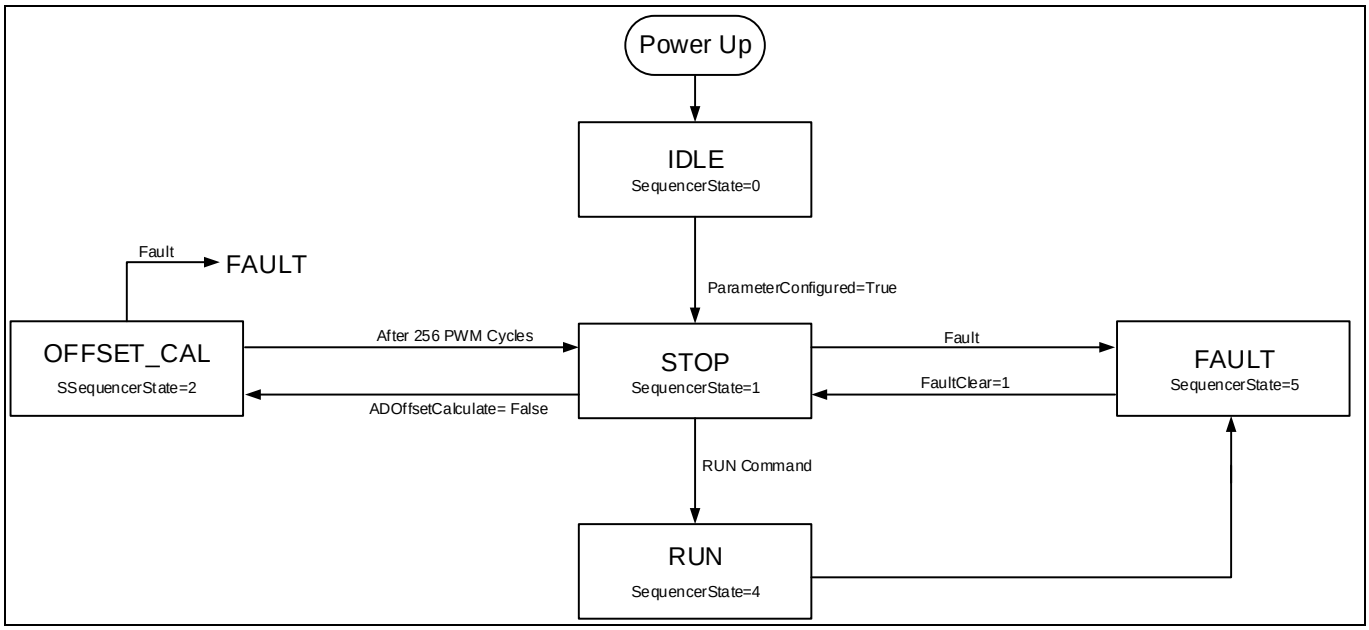
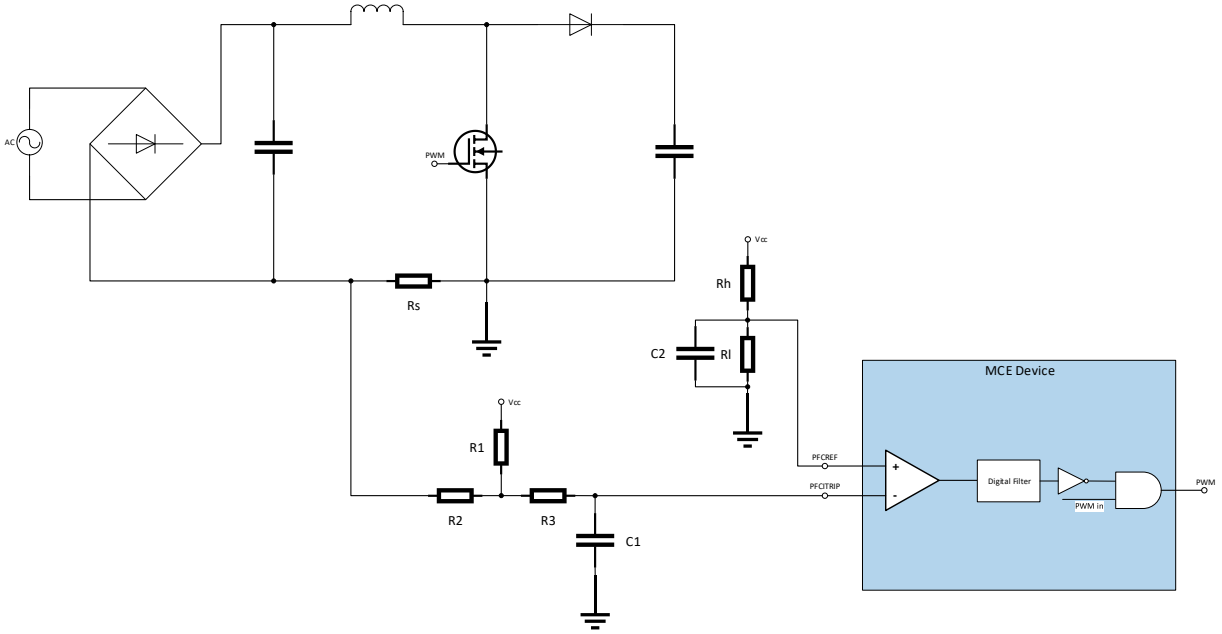


Figure 56 State handling flow chart

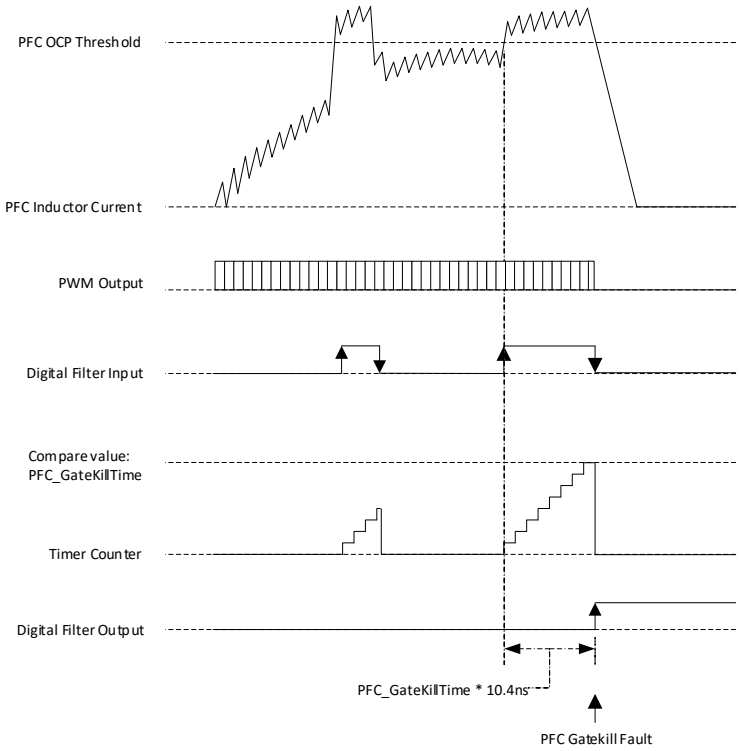
2.2.2 Protection

2.2.2.1 PFC Over Current Protection (OCP)

MCE provides an over-current protection function by comparing the sensed instantaneous PFC inductor current against a pre-configured OCP threshold and disables the PWM output when the sensed inductor current exceeds the OCP threshold.



(1) Latch-off OCP Application Diagram



(2) Latch-Off OCP Timing Diagram

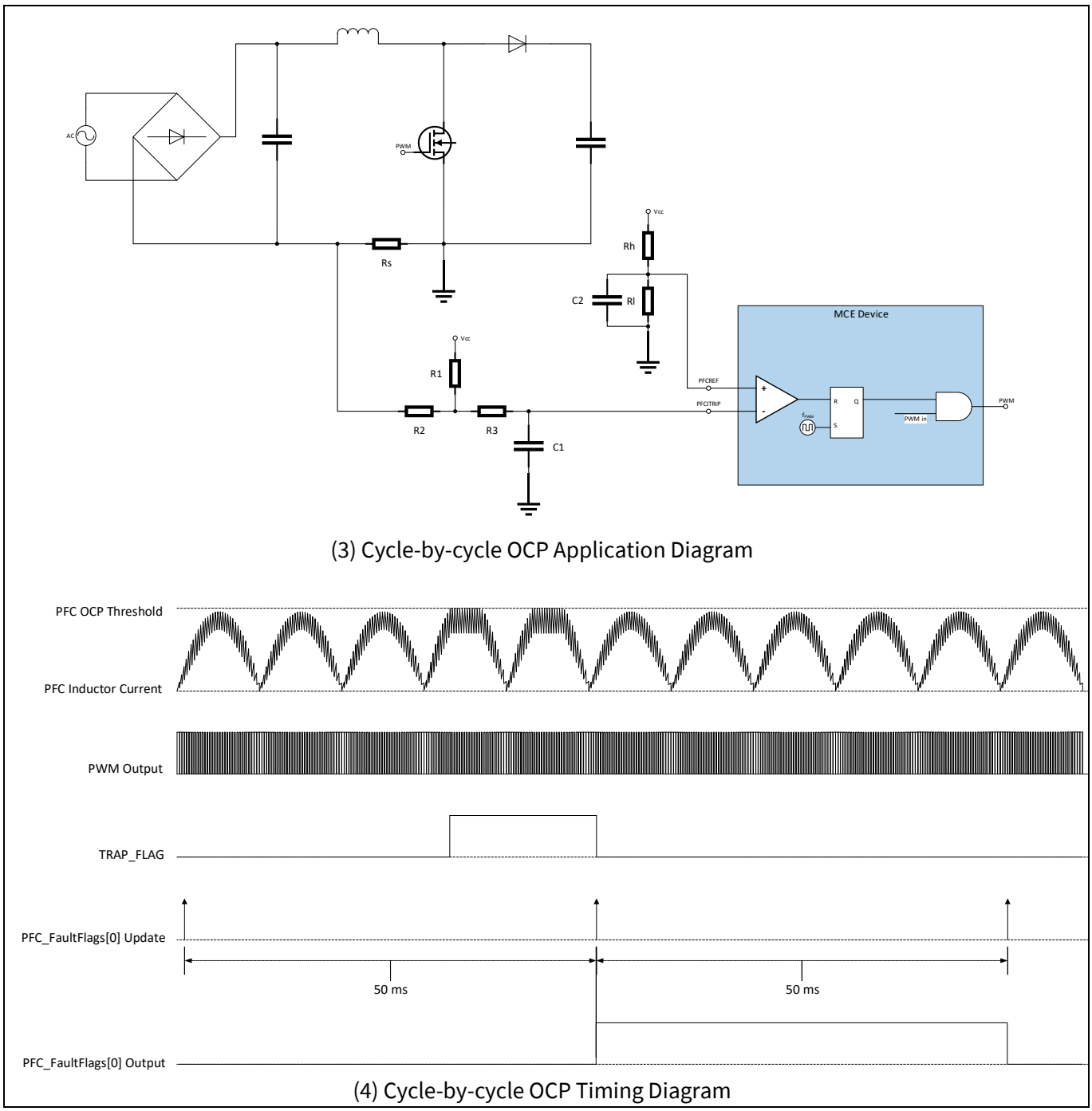


Figure 57 PFC Gatekill setup

As shown in the Figure 57 (2) and Figure 57 (4), the over-current tripping mechanism makes use of an internal comparator. The tripping level can be programmed externally using a voltage divider driven by a reference voltage whose output is connected to PFCREF pin. The calculation of OCP tripping level is shown below.

iTripLevel [V] = iTripCurrentLevel[Amps]*CurrentInputScale [V/Amps] + AmplifierOffset

CurrentInputScale= Rshunt*ExternalAmpliferGain

There are 2 available options regarding the OCP mechanisms: latch-off OCP, and cycle-by-cycle OCP.

For latch-off OCP option as shown in Figure 57 (1), an internal configurable digital filter is available to avoid any high frequency noise. The customer can configure the gatekill response time by adjusting the value of

'PFC_GateKillTime' parameter. The input signal needs to remain stable for the specified GateKillFilterTime period to trigger the over-current fault. This fault cannot be disabled.

The filter timer is configured to be level triggered by the internal comparator output. As shown in Figure 57 (2), when the inductor current goes higher than the specified PFC OCP threshold, the internal comparator output goes logic HIGH. As a result, the timer starts counting. If the comparator output voltage level changes down to logic zero, then the timer gets reset. If the over-current condition is persistent until the timer counts to 'PFC_GateKillTime', then the PWM outputs go into the programmed passive levels. This fault cannot be masked, so that it will be reflected in 'PFC_SwFaults' PFC variable, and the PFC state machine will shift to FAULT state, causing the PFC to stop running.

This fault can be cleared by writing 1 to PFC_FaultClear PFC variable while the over-current condition is no longer present. If the fault clear operation is successful, then the PFC state machine will shift to STOP state. Users need to set 'PFC_Command' = 1 to force the PFC state machine to go back to RUN state to restart the PFC operation after an PFC OCP fault is confirmed.

'PFC_GateKillTime' is a type of static PFC parameter that specifies the gatekill response time for over-current fault detection. The valid range of its value is from 0 to 960 in clock cycles. The value of 1 corresponds to 1/96MHz = 10.4167ns. The default value is 48, which is 0.5µs.

For cycle-by-cycle OCP option as shown in Figure 57 (3) and Figure 57 (4), when the inductor current goes higher than the specified PFC 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 goes 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. When an OCP fault occurs, an internal TRPF bit is set automatically and requires manual reset. Every 50 ms, the value of TRPF bit is read and copied to bit 0 of 'PFC_FaultFlags' variable. Then TRPF bit gets cleared. If the OCP fault still remains, then the TRPF bit won't be cleared successfully. If a PFC OCP fault occurs for just one PWM cycle, then bit 0 of 'PFC_FaultFlags' variable will be set to 1 for a duration of 50 ms. When cycle-by-cycle OCP option is selected, if an PFC OCP fault occurs, the PFC state machine remains in RUN state, and bit 0 of 'PFC_SwFaults' variable doesn't get set. Users may read the value of bit 0 of 'PFC_FaultFlags' to find out the occurrence of PFC OCP fault.

2.2.2.2 DC Over/Under Voltage Protection

Under voltage is set when the DC Bus voltage is below a threshold and over voltage is set when the DC Bus voltage is above a threshold.

DC bus voltage is being sampled every PFC switching cycle. The sampled DC bus voltage, which can be read from 'PFC_VdcRaw' PFC variable, goes through a Low-Pass Filter to attenuate high-frequency noise. The filtered DC bus voltage can be read from 'PFC_VdcFilt' PFC variable. The time constant of the LPF depends on the PFC PWM frequency.

If the 'PFC_VdcFilt' value is lower than 'PFC_VdcLvLevel', then bit 1 in 'PFC_FaultFlags' PFC variable is set. If the bit 1 in 'PFC_FaultEnable' PFC dynamic parameter is set, then this fault will be reflected in 'PFC_SwFaults' PFC variable, and the PFC state machine will shift to FAULT state causing the PFC to stop running. If this bit is not set, then the corresponding bit in 'PFC_SwFaults' variable will be masked by 'PFC_FaultEnable' parameter, so that this fault will not be reflected in 'PFC_SwFaults' variable, and the PFC state machine will not shift to FAULT state and the PFC will keep running.

If the 'PFC_VdcFilt' value is higher than 'PFC_VdcOvLevel', then bit 2 in 'PFC_FaultFlags' PFC variable is set. If the bit 2 in 'PFC_FaultEnable' PFC dynamic parameter is set, then this fault will be reflected in 'PFC_SwFaults 'PFC variable, and the PFC state machine will shift to FAULT state causing the PFC to stop running. If this bit is not set, then the corresponding bit in 'PFC_SwFaults' variable will be masked by 'PFC_FaultEnable' parameter,

so that this fault will not be reflected in 'PFC_SwFaults' variable, and the PFC state machine will not shift to FAULT state and the PFC will keep running.

These fault can be cleared by writing 1 to 'PFC_FaultClear' PFC variable while the DC bus over voltage or under voltage condition is no longer present. If the fault clear operation is successful, then the PFC state machine will shift to STOP state.

2.2.2.3 AC Over/Under Voltage Protection

AC over voltage fault is set when the AC input voltage to PFC is above a threshold and AC under voltage fault is set when the AC input voltage to PFC is below a threshold

AC input voltage is being sampled during every PFC switching cycle. The RMS value of the AC input voltage is calculated every PFC state machine update (Default value is 1ms).

The AC over-voltage fault is checked by comparing the calculated VAC RMS value against 'PFC_VacOvLevel' value. If the VAC RMS value is higher than 'PFC_VacOvLevel', then bit 5 in 'PFC_FaultFlags' PFC variable is set. If the bit 5 in 'PFC_FaultEnable' PFC dynamic parameter is set, then this fault will be reflected in 'PFC_SwFaults' PFC variable, and the PFC state machine will shift to FAULT state causing the PFC to stop running. If this bit is not set, then the corresponding bit in 'PFC_SwFaults' variable will be masked by 'PFC_FaultEnable' parameter, so that this fault will not be reflected in 'PFC_SwFaults' variable, and the PFC state machine will not shift to FAULT state and the PFC will keep running.

The AC under-voltage fault is checked by comparing the calculated VAC RMS value against 'PFC_VacLvLevel' value. If the VAC RMS value is less than 'PFC_VacLvLevel', then bit 4 in 'PFC_FaultFlags' PFC variable is set. If the bit 4 in 'PFC_FaultEnable' PFC dynamic parameter is set, then this fault will be reflected in 'PFC_SwFaults' PFC variable, and the PFC state machine will shift to FAULT state causing the PFC to stop running. If this bit is not set, then the corresponding bit in 'PFC_SwFaults' variable will be masked by 'PFC_FaultEnable' parameter, so that this fault will not be reflected in 'PFC_SwFaults' variable, and the PFC state machine will not shift to FAULT state and the PFC will keep running.

This fault can be cleared by writing 1 to PFC_FaultClear PFC variable while the AC input over voltage condition or under voltage condition is no longer present. If the fault clear operation is successful, then the PFC state machine will shift to STOP state.

2.2.2.4 Input Frequency Protection

This fault is set when AC input frequency value to PFC is different from set value.

The AC input frequency max and min limits are configured by MCEWizard automatically based on the selected nominal AC input frequency. If the AC input frequency nominal value is selected as 50Hz, then the valid range of actual AC input frequency is from 45 to 55Hz. If the AC input frequency nominal value is selected as 60Hz, then the valid range of actual AC input frequency is from 55 to 65Hz.

AC input frequency min limit is checked every time the PFC state machine updated (Default value is1ms). If the measured positive or negative half line cycle is lower than the min limit, then bit 3 in 'PFC_FaultFlags' PFC variable is set. This fault cannot be masked, so that it will be reflected in PFC_SwFaults PFC variable and the PFC state machine will shift to FAULT state causing the PFC to stop running.

AC input frequency max limit is checked in the process of finding zero crossing executed every PFC PWM cycle. During Each PFC PWM cycle, a counter is incremented and compared against the max limit. If the counter value is higher than the max limit, it indicates that zero crossing is not found within the max amount of valid half cycle time. If it is set, then bit 3 in 'PFC_FaultFlags' PFC variable is set. This fault will be reflected in 'PFC_SwFaults' PFC variable, and the PFC state machine will shift to FAULT state causing the PFC to stop running.

This fault can be cleared by writing 1 to PFC_FaultClear PFC variable while the AC input frequency fault is no longer present. If the fault clear operation is successful, then the PFC state machine will shift to STOP state.

Note: PFC will be stopped during any fault in the motor control.

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 **Error! Reference source not found.**.

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 585. Notice that it follows little endian format.

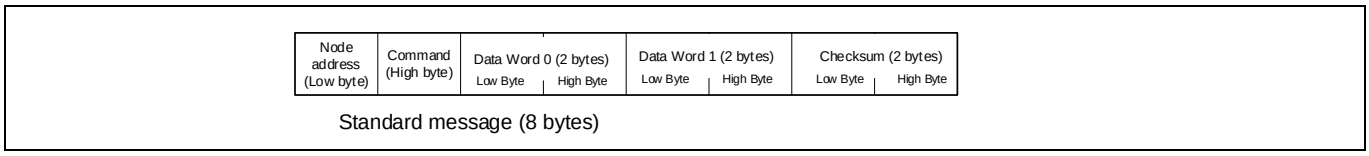


Figure 58 UART Data Frame

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 12 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 MCEWizard.

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 13 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 58.

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

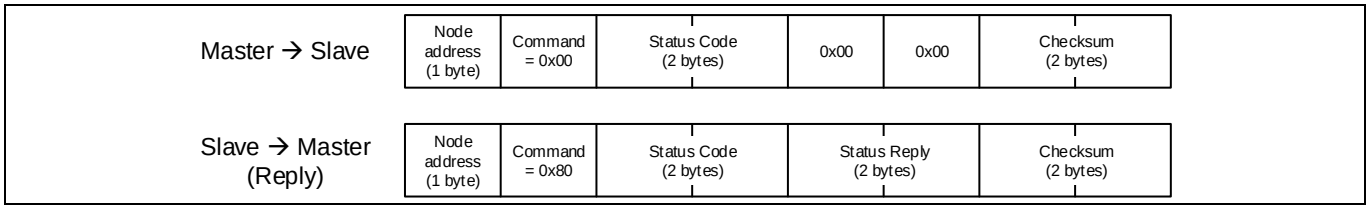


Figure 59 Read Status command

Table 14 Status code and status reply

Status code	status reply
0x0000	Fault Flags

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

Clear Fault: Command =0x01

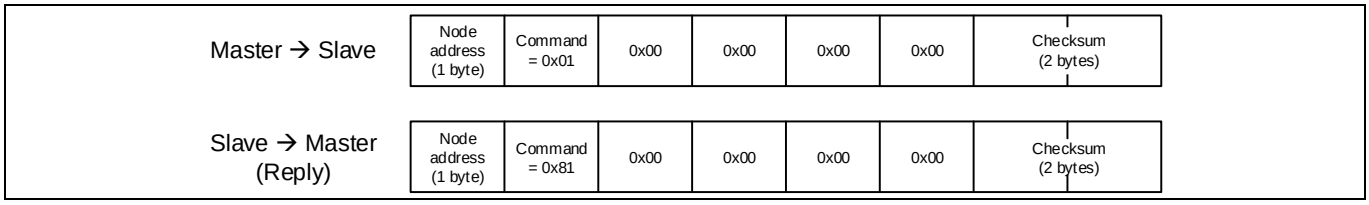


Figure 60 Clear fault command

2.3.7.2 Change Control Input Mode: Command =0x02

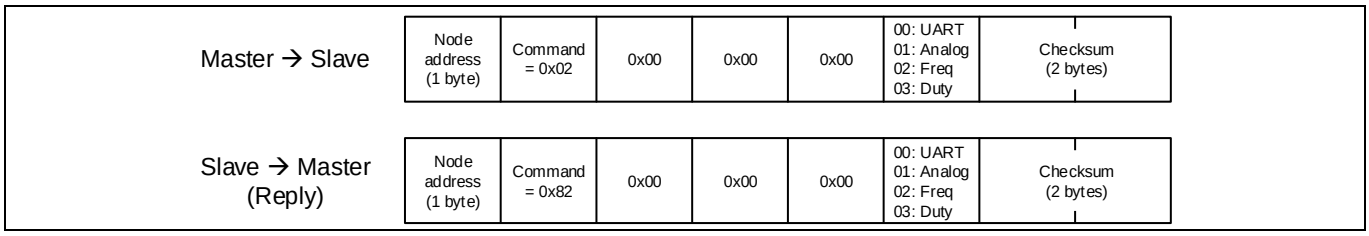


Figure 61 Control input mode command

2.3.7.3 Motor Control: Command =0x03

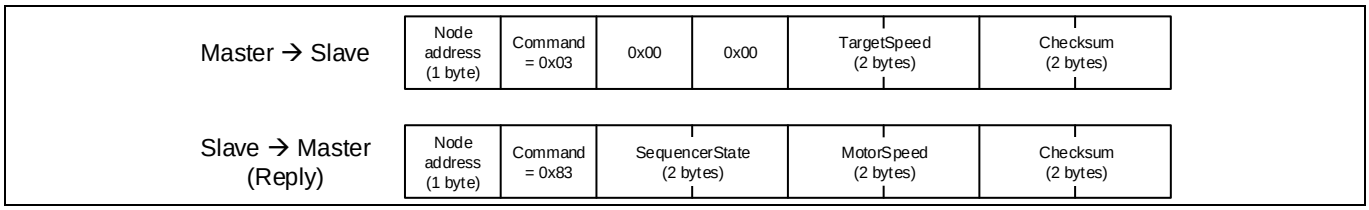


Figure 62 Motor control Command

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

2.3.7.4 Register Read: Command = 0x05

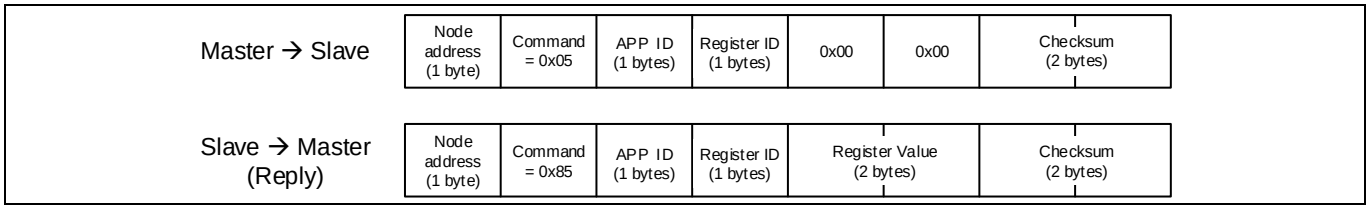


Figure 63 Register Read Command

2.3.7.5 Register Write: Command = 0x06

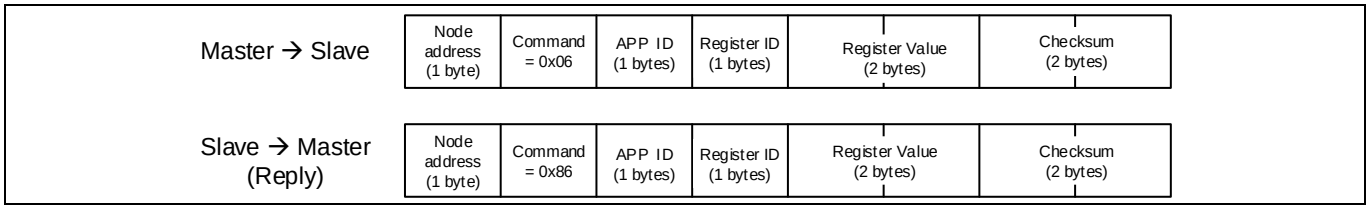


Figure 64 Register Write Command

2.3.7.6 Load and Save Parameter: Command = 0x20

'Load parameter' command loads the parameters from the specified parameter set 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.)

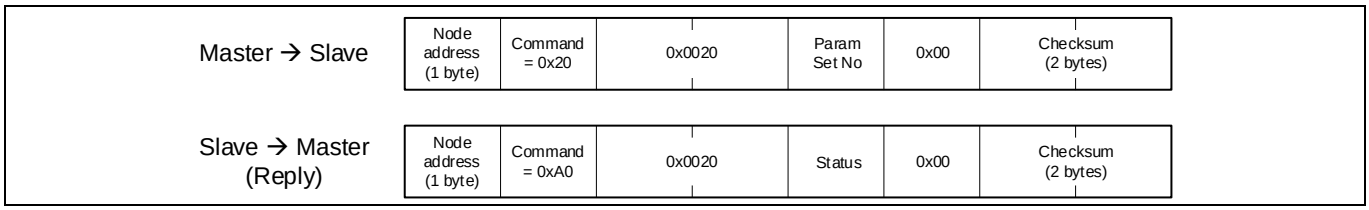


Figure 65 Load parameter Command

'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.)

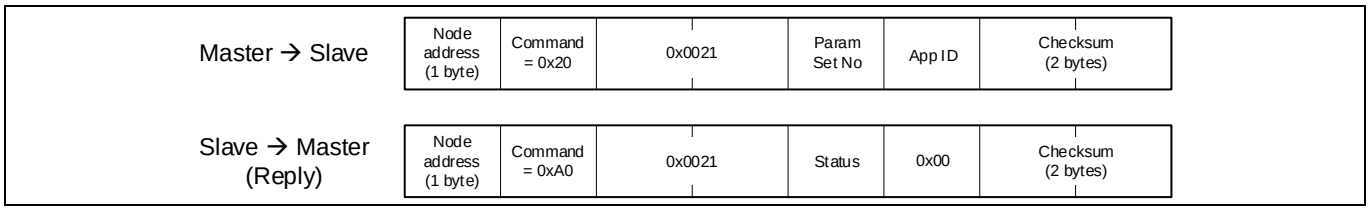


Figure 66 Save Parameter Command

2.3.8 Connecting multiple nodes to same network

It is possible to connect multiple MCE to same UART network, see Figure 67 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.

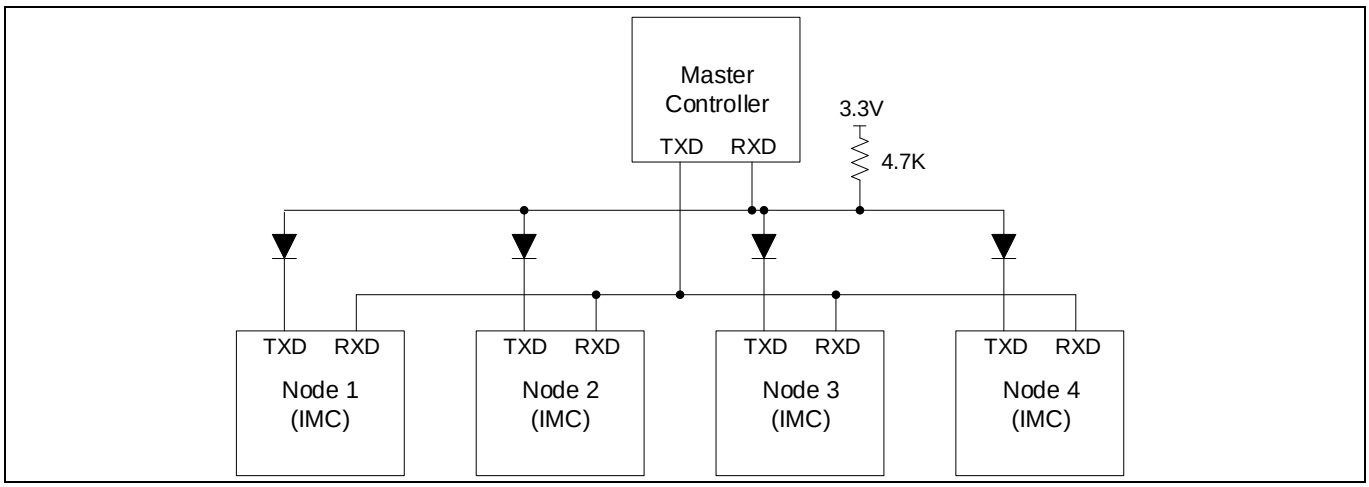


Figure 67 UART network connection

2.3.9 UART Transmission Delay

A configurable delay (bit [15:8] of parameter 'InterfaceConf0') 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 [2:0] of the parameter 'InterfaceConf1'.

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 68 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 68 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 15 summarizes the functions corresponding to different MO numbers.

Table 15 Message Object Function Table

Message Object	Functions
0	State machine inquiry; Execution time and CPU load inquiry.
1	System configuration protection; Reset T core; Access static parameter; Set boot mode; 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 69 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. 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 69 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 ('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 70 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: p = 0: protected 0 < p < 3: unprotected for the next p commands
DS	0x0001		0		Reset (immediately)
DS	0x0002		a		Static parameter access: a = 0: disable 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 70 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 71 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 3. 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 71 Message Structure (MO = 6)

2.4.4.4 Message Object: 7

2.4.4.4.1 Set Parameter

The following Figure 72 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 72 Message Structure (MO = 7)

2.5 Multiple Parameter Programming

2.5.1 Parameter Page Layout

In iMOTION™ 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 maximum of 15 can be used to support different motor types or hardware and 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 MCEWizard. MCEWizard output (*.txt) that contains the parameter values, can be programmed into the parameter block using MCEDesigner. MCEWizard output file contains the specified parameter set number. MCEDesigner loads the parameter values into the corresponding parameter block. Each parameter block can be updated 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, 12, and 14.

2.5.2 Parameter Block Selection

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

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

Parameter block selection input configuration is available in MCEWizard and MCEWizard updated “ParPageConf” parameter.

Note: Not all of the 4 methods to select parameter block are available in all iMOTION™ 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 “ParPageConf [7:4]” parameter bit field value. “ParPageConf [7:4]” parameter bit field value can be updated from MCEWizard.

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 16.

Table 16 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

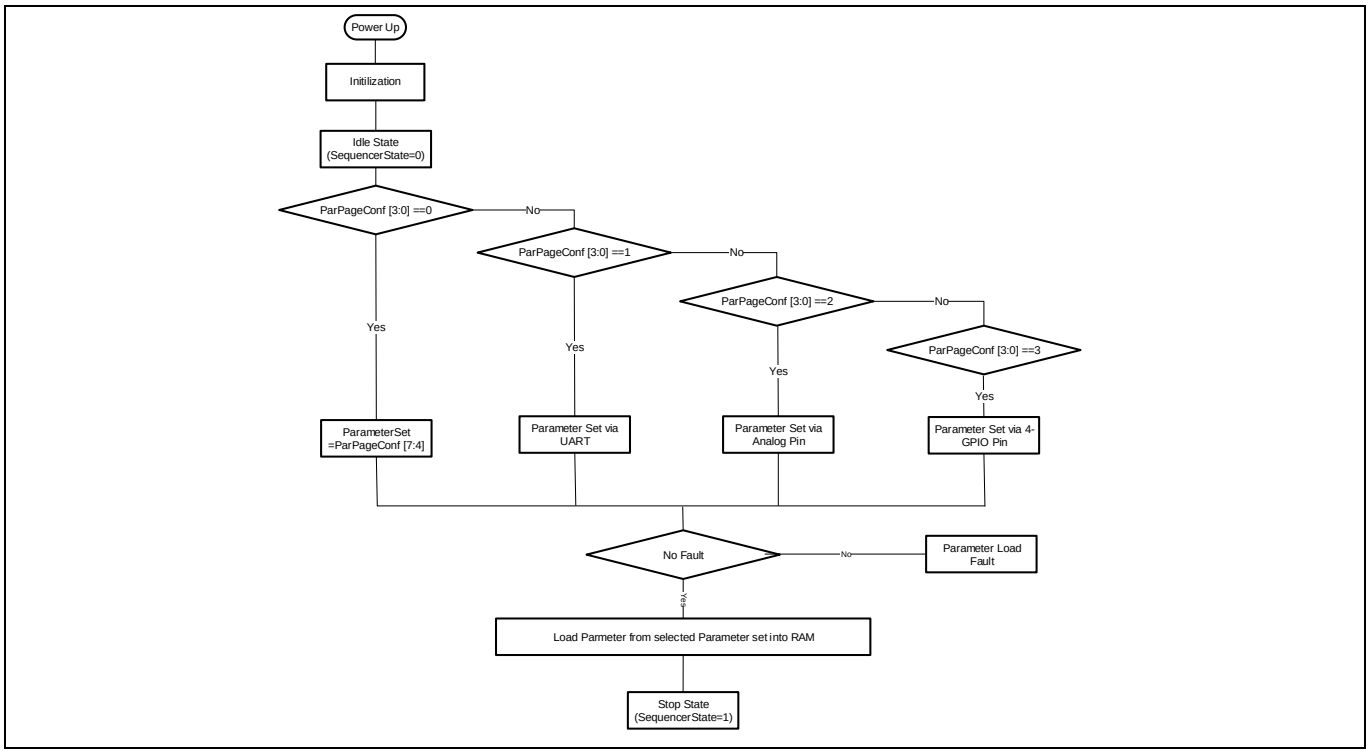


Figure 73 Parameter Load Procedure

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 virtual machine running in MCE. Script Engine enables user to implement system level functionalities beyond motor control and PFC. Key advantages of script engine are:

- Extend capabilities of Turnkey devices by allowing to use 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 the motor control and PFC parameters and variables.

Some of the script use cases are listed below:

- Customization of System Start-up behaviour
  - Start motor and PFC based on external sensor or control inputs
  - Validate the system status before start motor and PFC
  - Modify motor current limit, voltage limit, Speed Ramp rate etc.
- Define specific speed profile and Parameter Configuration
  - Set target speed value based on analog input or DC bus voltage or switch relay or fixed profile
  - Runtime adjustment of speed Ramp rate
  - PI value profiling at different speed range
  - Synchronization between PFC and motor control operation
- Fault handling and Parameter configuration
  - Define system specific fault handling, reduce the speed or current during any fault conditions and recovery scheme after fault
- Implementation of customized UART protocol using configurable UART driver

2.6.1 Overview

Script code follows ‘C’-like syntax. Script engine executes the script code from two different task with different priority. Script engine supports arithmetic, binary logical operators, decision statement (If...else statement) and loop statement (FOR statement). User can define variables in script code and these variable can be monitored from MCEDesigner. In iMOTION product, 16kB of memory area is reserved to store the script code. Consequently, the maximum allowed script byte code size is 16kB (Approximately 1.5k lines of code). Script code generation flow is depicted in Figure 74.

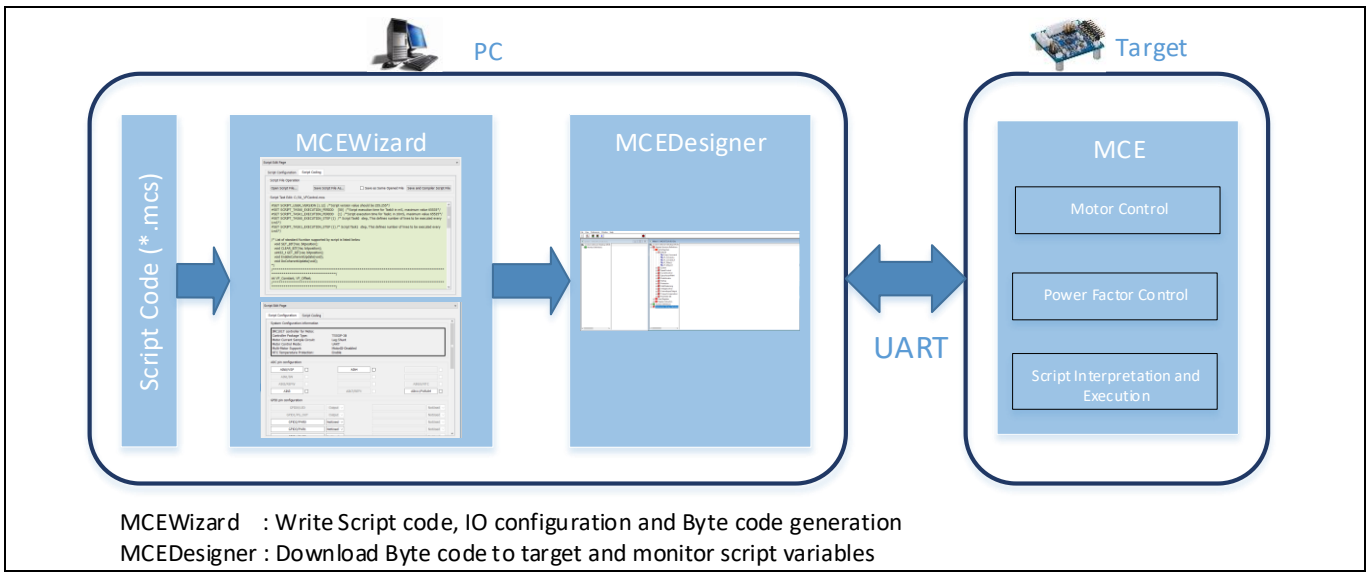


Figure 74 Script Code Generation flow

2.6.2 Script Program Structure

Script program consists of the following part

- Set Commands : Define script user version and script task execution period
- Functions: Script code should be written inside four predefined function- Script_Task0_init (), Script_Task0 (), Script_Task1_init () and Script_Task1 ().
- Variables and Parameters
- 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

```
001   /*************************************************************/
002   /*Script user version value, should be 255.255*/
003   #SET SCRIPT_USER_VERSION (1.12)
004    /*Script execution time for Task0 in ms, maximum value 65535*/
005   #SET SCRIPT_TASK0_EXECUTION_PERIOD    (500)
006   /*Script execution time for Task1 in 10ms, maximum value 65535*/
007   #SET SCRIPT_TASK1_EXECUTION_PERIOD    (1)
008   /*************************************************************/
009   /* Global variable definition */
010   int Var1;
011   /*************************************************************/
012   /*Task0 init function*/
013   Script_Task0_init()
014   {
015     /* local variable definition */
016     int Task0Var1;
017     Task0Var1 =0;  /*Initialize local variable*/
018     Var1 =0;       /*Initialize global variable*/
019   }
020   /*Task0 script function*/
021   Script_Task0()
022   {
023      Task0Var1 =  Task0Var1+1; /*Increment Task0Var1*/
024      Var1      =  Var1+1;      /*Increment Var1*/
025   }
026   /*************************************************************/
027   /*Task1 init function*/
028   Script_Task1_init()
029   {
030     /* local variable definition */
031     int Task1Var1;
032     Task1Var1 =0;    /*Initialize local variable*/
033   }
034   /*Task0 script function*/
035   Script_Task1()
036   {
037      Task1Var1 =  Task1Var1+1; //Increment Task1Var1
038      Var1      =  Var1+1;      /*Increment Var1*/
039   }
```

Code Listing 1 Run time counter usage example

2.6.3 Script Program Execution

Script engine executes script code from two independent tasks, named Task0 and Task1. Both the tasks are executed periodically. Task execution period can be configured using “SCRIPT_TASK0_EXECUTION_PERIOD” and “SCRIPT_TASK1_EXECUTION_PERIOD” parameters in script input file (*.mcs), for each tasks. Each tasks have separate initialization functions (Script_Taskx_init ( )) to initialize script variable and motor/PFC parameters. Also it is possible to write script code inside the initialization function. These functions are called only once during start-up. Task0/Task1 script functions (Script_Taskx) are called periodically based on task execution period value.

Script tasks have lower priority than motor control or PFC control loop functions. Among script tasks, Task0 has higher priority than Task1.

For Task0, by default, the execution step is 1, the execution period is 50 (50 x 1 ms = 50 ms). For Task1, by default, the execution step is 10, the execution period is 10 (10 x 10 ms = 100 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. 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.

Users can configure the execution step and execution period of each task to their likings. 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 and after finishing the current execution, it immediate starts over the execution 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. It is possible to increase number of lines executed by Task0 or Task1, to accelerate the script execution.

Number of lines to be executed every 1ms in Task0 can be configured in script input file using set parameter called “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 input file using set parameter called “SCRIPT_TASK1_EXECUTION_STEP”.

```
001  /*************************************************************/
002  /*Script user version value, should be 255.255*/
003  #SET SCRIPT_USER_VERSION (1.12)
004   /*Script execution time for Task0 in ms, maximum value 65535*/
005  #SET SCRIPT_TASK0_EXECUTION_PERIOD    (500)
006  /*Script execution time for Task1 in 10ms, maximum value 65535*/
007  #SET SCRIPT_TASK1_EXECUTION_PERIOD    (1)
008  /*Defines number of lines to be executed every 1ms in Task0*/
009  #SET SCRIPT_TASK0_EXECUTION_STEP (2)
```

```
010       /*Defines number of lines to be executed every 10ms in Task1*/
011       #SET SCRIPT_TASK1_EXECUTION_STEP (10)
012
```

Code Listing 2 Run time counter usage example

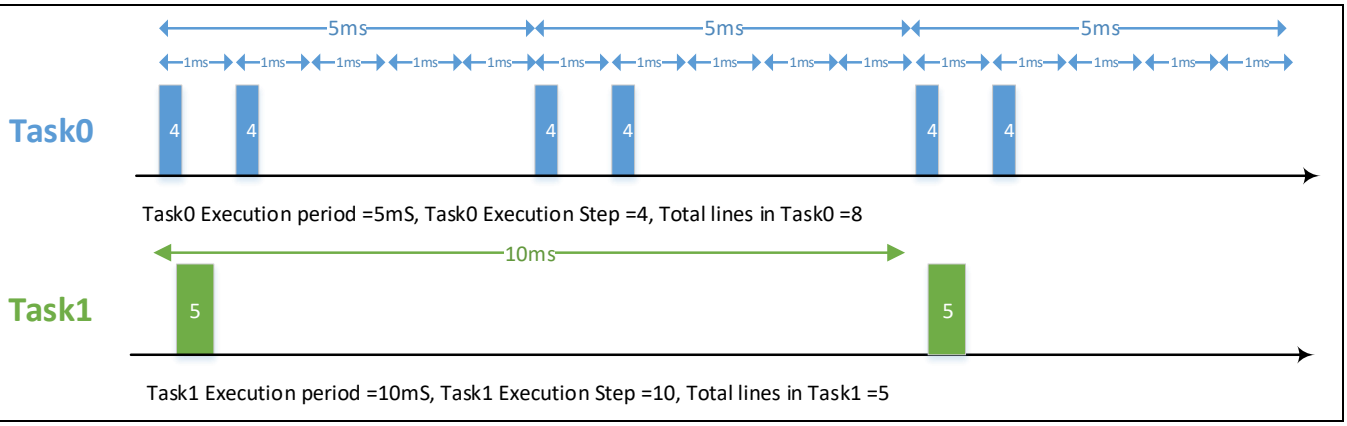


Figure 75 Script Task Execution

2.6.3.2 Free Running timer

One free running timer with 1ms resolution is available in the script engine to schedule periodic operation. Free running timer value (variable name: RunTimeCounter, size: 32 bit, type: Read only) can be directly accessed from script code. Example of RunTimeCounter is mentioned in the Code Listing 3

```
001       /*Task0 init function*/
002       Script_Task0_init()
003       {
004         /* local variable definition */
005         int sSVar0,sSVar1;
006         sSVar0 = RunTimeCounter;
007       }
008       /*Task0 script function*/
009       Script_Task0()
010       {
011         /* sSVar1 value toggles for every 10 seconds*/
012         if((RunTimeCounter-sSVar0)>10000)
013         {
014           sSVar0 =RunTimeCounter;
015           if(sSVar1==0)
016           {
017             sSVar1 = 1;
018           }
019           else
020           {
021             sSVar1 = 0;
022           }
023         }
024       }
```

Code Listing 3 Run time counter usage example

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, example U or L.

If any variable is assigned with float literals, value after decimal place is ignored by 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. The following Code Listing shows an example of constant definition.

Code Listing 4

```
/*************************************************************/
/* Constant definition */
CONST int Const1 = 1;
const int Const2 = 2;
/*************************************************************/
/*Task0 init function*/
Script_Task0_init()
{
/* local variable definition */
      int Task0Var1, Task0Var2;
      Task0Var1 = Const1; /*Initialize local variable*/
      Task0Var2 = Const2; /*Initialize local variable*/
}
```

2.6.5 Variable types and scope

Script engine supports maximum of 30 global variables, these variables can be accessed from both tasks. Each task has maximum of 24 dedicated local variables, which can only be accessed within the respective task. All the variables are 32-bit signed variables. Only global variables can be accessed from MCEDesigner or User UART interface.

User can assign any name to script variable during declaration. Keyword ‘int’ should be used to declare script variable. 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. Keyword ‘int’ should be used to declare script variable.

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

Note: Variable can’t be initialized during declaration.

```
/*************************************************************/
/* Global variable definition */
int Var1,Var2;
/*************************************************************/
/*Task0 init function*/
Script_Task0_init()
{
  /* local variable definition */
  int Task0Var1;
  Task0Var1 =0;  /*Initialize local variable*/
  Var1 =0;       /*Initialize global variable*/
}
```

```
013       /*Task0 script function*/
014       Script_Task0()
015       {
016          Task0Var1 =  Task0Var1+1; /*Increment Task0Var1*/
017          Var1      =  Var1+1;      /*Increment Var1*/
018       }
019       /*************************************************************/
020       /*Task1 init function*/
021       Script_Task1_init()
022       {
023         /* local variable definition */
024         int Task1Var1;
025         Task1Var1 =0;    /*Initialize local variable*/
026       }
027       /*Task0 script function*/
028       Script_Task1()
029       {
030          Task1Var1 =  Task1Var1+1; //Increment Task1Var1
031          Var2      =  Var1+1;
032       }
```

Code Listing 5 Script Global and Local Variables

2.6.6 Motor and PFC Parameter Access

All the motor control and PFC parameter and variables listed Table 28, Table 29, Table 30 and Table 31 can be accessed from script. Parameter and variables can be used directly in the script code without declaration. Only DYNAMIC type parameters and READWRITE type variables can be write from the script code. While writing this parameter or variables, range check will do performed before update the parameter or variable. If the value is out of range, parameter/variable won't be updated and error bit will be set to 0x13. It is possible to read the error flag (variable name: "`ErrorFlag`") from script code.

If write operation is performed on STATIC type parameter or READONLY type variable, parameter or variable won't be updated and error bit will be set to 0x10. This error flag can be cleared from script code directly.

Set of parameter and variables can be updated simultaneously using 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 be updated parameter and variables values immediately. Instead, all the values are stored into a buffer and update all parameter and variable simultaneously after calling DoCoherentUpdate () method. Script supports simultaneous update maximum of 32 parameter and variable. (Refer 2.6.11.2)

2.6.7 Operators

An operator is a symbol that inform the script to perform specific mathematical or logical functions. List of operators supported in script function are mentioned below

Table 17 Arithmetic Operators

Operator	Description
+	Adds two operands
-	Subtracts second operand from the first.
*	Multiplies both operands.
/	Divides numerator by de-numerator.
%	Modulus Operator and remainder of after an integer division

Table 18 Binary Operators

Operator	Description
|	Binary OR Operator copies a bit 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 the bit 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 19 Assignment Operators

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

Table 20 Relational Operators

Operator	Description
==	Checks if the values of two operands are equal or not. 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 less than the value of right operand. If yes, then the condition becomes true.

Table 21 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

2.6.8 Expressions

An expression can contain numbers, script variables, motor control/ PFC variables and parameters. Left and right parentheses can be used.

Example: TargetSpeed = ADCResult10*(InputScale+10)

In an expression all the operator has same precedence and executed from left side to right side. So it is required to use left and right parentheses to force order of evaluation.

2.6.9 Decision Structures

Decision structures are used for branching. Script engine provides if statement for decision making. If statement 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 mentioned below

```
001        if(boolean_expression)
002        {
003          /*Statement(s) will execute if the expression is true*/
004        }
005        else
006        {
007          /*Statement(s) will execute if the expression is false*/
008        }

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

Code Listing 6 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.

```
001        /*Task0 init function*/
002        Script_Task0_init()
003        {
004          /* local variable definition */
005          int InputVal,OutputVal;
006          InputVal =1;
007        }
008        /*Task0 script function*/
009        Script_Task0()
010        {
011          /*Check the boolean condition*/
012          if(InputVal)
013          {/* if condition is true then assign OutputVal =10 */
014            OutputVal=10;
015          }
016          else
017          {/* if condition is false then assign OutputVal =100 */
018            OutputVal=100;
019          }
020        }

Result : OutputVal =10
```

Code Listing 7 Example If...else statement

More example for if statement Boolean expression supported by script are listed below

Table 22 Example If Statement Boolean expressions

Boolean expression	Description
If(InputVal1 ==1)	Condition is true if Inputval1 is equal to 1
if(InputVal1)	Condition is true if Inputval1 is equal to 1
if(InputVal1 &0x01)	Condition is true if Inputval1 first bit is SET(1)
if(InputVal1 |0x01)	Condition is always true
if(InputVal1 ^0x01)	Condition is true if Inputval1 is not equal to 1
if((InputVal1 ==1)&&(TargetSpeed==0))	Condition is true if Inputval1 is equal to 1 and TargetSpeed is equal to 0
if((InputVal1 ==1)||(TargetSpeed==0))	Condition is true if Inputval1 is equal to 1 or TargetSpeed is equal to 0
if((InputVal1 ==1) && ((TargetSpeed==0)||(InputVal2 ==0)))	Condition is true if Inputval1 is equal to 1 and TargetSpeed or Inputval2 is equal to 0
if((InputVal1 ==1)||((TargetSpeed==0)||(InputVal2 ==1)))	Condition is true if Inputval1 is equal to 1 or TargetSpeed is equal to 0 or Inputval2 is equal to 1

It is possible to write nested if conditions, depth of nested if condition is limited to 15.

```
001         if(boolean_expression1)
002         {
003           /*Statement(s) will execute if the expression1 is true*/
004           if((boolean_expression2)
005           {
006             /*Statement(s) will execute if the expression2 is true*/
007             if((boolean_expression3)
008             {
009               /*Statement(s) will execute if the expression3 is true*/
010             }
011           }
012         }
```

Code Listing 8 Nested If... statement syntax

Script code only support “if” and “else” key words in decision structure. Code Listing 9 provide if…elseif..else statement syntax

```
001         if(boolean_expression1)
002         {
003           /*Statement(s) will execute if the expression1 is true*/
004         }
005         else
006         {
007           if(boolean_expression)
008           {
009             /*Statement(s) will execute if the expression is true*/
010           }
011           else
012           {
013             /*Statement(s) will execute if the expression is false*/
014           }
015
016         }
```

Code Listing 9 Syntax for if... Elseif...else statement

2.6.10 Loop Structures

Loop structures are used for repeat process. FOR statement is supported for repeat processes.

Syntax of FOR statement in script language is mentioned below

```
001         for(<ScriptVariable> = <Startvalue> : <Endvalue>)
002         {
003           /*Statement(s) will execute for defined loop time*/
004         }
```

Code Listing 10 for statement syntax

Statements inside for loop are executed for Endvalue- Startvalue+1 times.

```
001        /*Task0 init function*/
002        Script_Task0_init()
003        {
004          /* local variable definition */
005          int InputVal,OutputVal;
006          OutputVal=0;
007        }
008        /*Task0 script function*/
009        Script_Task0()
010        {
011          if(OutputVal==0)
012          {
013            for(InputVal =1 : 10)
014            {/* for loop executed for 10 times*/
015              OutputVal= OutputVal+1;
016            }
017          }
018        }

Result : OutputVal =10
```

Code Listing 11 for statement Example

FOR statement does not counting down mode, always start value should be less than end value.

2.6.11 Methods

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

2.6.11.1 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 23 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 postion value should be 0 to 15

```
001        /*Task0 init function*/
002        Script_Task0_init()
003        {
004          /* local variable definition */
005          int InputVal,OutputVal1, OutputVal2;
006          InputVal =0;
007        }
008        /*Task0 script function*/
009        Script_Task0()
010        {
011          SET_BIT(InputVal,15);/*Set 15 bit of InputVal, InputVal =0x8000*/
012          /*Read 15 bit of InputVal and assign to OutputVal1*/
013          OutputVal1=GET_BIT(InputVal,15); /*OutputVal1=1*/
014          CLEAR_BIT(InputVal,15);/*clear 15 bit of InputVal, InputVal =0*/
015          /*Read 15 bit of InputVal and assign to OutputVal1*/
016          OutputVal2=GET_BIT(InputVal,15);/*OutputVal2=0*/
017        }

   Result : OutputVal1 =1 and OutputVal2=0
```

Code Listing 12 Bit Access Methods Example

2.6.11.2 Coherent update methods

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

Table 24 Coherent Methods

Methods	Description
void EnableCoherentUpdate(void)	Enable simultaneous update of parameter/variables
void DoCoherentUpdate(void)	Trigger simultaneous update of parameter/variables

Note: Maximum 32 variables can be updated simultaneously.

When 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.

```
001        /*Task1 init function*/
002        Script_Task0_init()
003        {
004          EnableCoherentUpdate();
005          AngleSelect =0; //Set to open loop
006          CtrlModeSelect =0;// voltage control mode
007          TargetSpeed = 0x258; //Set speed value
008          DoCoherentUpdate();
009        }
```

Code Listing 13 Coherent update Methods Example

2.6.11.3 Configurable UART API

Table 25 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.

2.6.11.3.1 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 by the Configurable UART. 0: UART 0 1: UART 1
rxInvert	0	1	Configures the data interpretation logic for the received data. 0: non-inverting 1: inverting
txInvert	0	1	Configures the data interpretation logic for the transmitted data. 0: non-inverting 1: inverting
baudrate	600 bps	115,200 bps (230,400 bps in FIFO mode)	Configures the baudrate for the UART in bps.
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.

2.6.11.3.2 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.

2.6.11.3.3 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.

2.6.11.3.4 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. 0: Full duplex 1: Half duplex
rxTimeout	0	65535	Configure the longest expected time to receive a frame. If a frame isn't

Input Parameters	Min	Max	Description
			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.
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. 0-255: valid flag byte 256-65535: invalid flag / no flag byte is used
txFlag	0	65535	txFlag is a byte that signifies the beginning of a transmit frame. 0-255: valid flag byte 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.

2.6.11.3.5 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 as below. **FIFO status:** Bit 0 – **IsRxFIFOEmpty:** is receive FIFO empty bit 0: receive FIFO is not empty

Return Type	Description
	1: receive FIFO is empty Bit 1 – **IsRxFIFOFull:** is receive FIFO full bit 0: receive FIFO is not full 1: receive FIFO is full Bit 2 – **IsTxFIFOEmpty:** is transmit FIFO empty bit 0: transmit FIFO is not empty 1: transmit FIFO is empty Bit 3 – **IsTxFIFOFull:** is transmit FIFO full bit 0: transmit FIFO is not full 1: transmit FIFO is full Bit 4:7- **reserved:** read as '0' **Buffer status:** Bit 8 – **IsRxBufferFull:** is receive buffer full bit 0: receive buffer is not full 1: receive buffer is full Bit 9 – **reserved:** read as '0' Bit 10 –**IsTxBufferEmpty:** is transmit buffer empty bit 0: transmit buffer is not empty 1: transmit buffer is empty Bit 11:14 – **reserved:** read as '0' Bit 10 –**IsBufferMode:** is Buffer Mode Initialized bit 0: the frame buffer and the driver handler is not initialized 1: the frame buffer and the driver handler is initialized **Handler status:** Bit 16 – **IsRxTimeout:** is receive frame timeout bit 0: receive frame is not timed out 1: receive frame is timed out Bit 17 – **IsCollision:** is collision detected bit 0: collision is not detected 1: collision is detected Bit 18:19 – **HandlerState:** Handler state bitfield 00: FRAME_START 01: FRAME_RECEIVE 10: FRAME_DELAY 11: FRAME_TRANSMIT Bit 20:22- **reserved:** read as '0' Bit 23 – **IsHalfDuplex:** is half duplex bit 0: the driver handler is not initialized in half-duplex mode 1: the driver handler is initialized in half-duplex mode

Return Type	Description
	Driver status: Bit 24 – **IsRxNoiseDetected:** is receive noise detected bit 0: noise on the receive line has not been detected 1: noise on the receive line has been detected Bit 25 – **IsParityError:** is parity error bit 0: a parity error has not occured 1: a parity error has occured Bit 26 – **IsStopBitError:** is stop bit error 0: a stop bit error has not occurred. 1: a stop bit error has occured Bit 27:30 – **reserved:** read as '0' Bit 31 – **IsInitialized:** is initialized bit 0: the driver is not initialized. 1: the driver handler is initialized.

Description:

This API returns the status word

2.6.11.3.6 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.

2.6.11.3.7 UART_Control()

Declaration:

```
void UART_Control(command)
```

Input Parameters	Description
command	Writes the Control Word whose bit field representation is described as below. **FIFO control:** Bit 0 – **reserved:** read as '0' Bit 1 – **ClrRxFIFO:** Clear RX FIFO bit 0: N/A

Input Parameters	Description
	1: clear the receive FIFO Bit 2 – **reserved:** read as '0' Bit 3 – **ClrTxFIFO:** Clear TX FIFO bit 0: N/A 1: clear transmit FIFO Bit 4:7 – **reserved:** read as '0' **Buffer control:** Bit 8 – **ClrRxBufferFlag:** Clear RX Buffer flag bit 0: N/A 1: clear receive buffer flag Bit 9 – **reserved:** read as '0' Bit 10 –**SendTxBuffer:** Send TX Buffer flag 0: N/A 1: Initiate the sending of bytes from the transmit buffer through the specified UART channel. Bit 11:15 – **reserved:** read as '0' **Handler control:** Bit 16 – **ClrRxTimeoutFlag:** Clear RX time-out Flag bit 0: N/A 1: clear receive time-out flag Bit 17 – **ClrCollisionFlag:** Clear Collision detected Flag bit 0: N/A 1: clear collision detection flag Bit 18 – **RstBufferControl:** Reset Buffer Control bit 0: N/A 1: reset buffer control state machine Bit 19:23 – **reserved:** read as '0' **Driver Control:** Bit 24 – **ClrRxNoiseFlag:** Clear RX Noise Flag bit 0: N/A 1: clear the receive noise flag Bit 25 – **ClrParityErrorFlag:** Clear Parity Error Flag bit 0: N/A 1: clear the parity error flag. Bit 26 – **ClrStopbitErrorFlag:** Cleare Stop bit Error Flag bit 0: N/A 1: clear the stop bit error flag Bit 27:31 – **reserved:** read as '0'

Description:

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

2.6.11.3.8 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.

2.6.11.3.9 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.

2.6.11.3.10 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 is to be returned.

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.

2.6.11.3.11 UART_TxBuffer()

Declaration:

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

Input Parameters	Min	Max	Description
idx	0	7	Specifies 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.

2.6.11.4 User GPIOs

Script enables to read or write the digital pins available for user (digital pins not used by motor control and PFC). Also read the analog pin value that are available for user.

2.6.11.4.1 Digital Input and Output Pins

Digital pins available for users can be configured as input or output pins using MCEWizard. All configured digital input/output pins values are read/write by MCE every 1 ms and the value can be read by script code.

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

Variable Name	Type	Description
GPIO_IN_L	READONLY	Holds digital input/output (GPIO0 to GPIO15) pins values.
GPIO_IN_H	READONLY	Holds digital input/output (GPIO16 to GPIO29) pins values.
GPIO_OUT_L	READWRITE	Set or reset digital output pin (GPIO0 to GPIO15)
GPIO_OUT_H	READWRITE	Set or reset digital output pin (GPIO16 to GPIO29)

The logic level of a GPIO pin can be read via the read–only registers GPIO_IN_L and GPIO_IN_H. Read GPIO_IN_L and GPIO_IN_H register always returns the current logical value the GPIO pin, independently whether the pins is selected as input or output.

GPIO_IN_L

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPIO15_IN	GPIO14_IN	GPIO13_IN	GPIO12_IN	GPIO11_IN	GPIO10_IN	GPIO9_IN	GPIO8_IN	GPIO7_IN	GPIO6_IN	GPIO5_IN	GPIO4_IN	GPIO3_IN	GPIO2_IN	GPIO1_IN	GPIO0_IN

GPIO_IN_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	Reserved	GPIO29_IN	GPIO28_IN	GPIO27_IN	GPIO26_IN	GPIO25_IN	GPIO24_IN	GPIO23_IN	GPIO22_IN	GPIO21_IN	GPIO20_IN	GPIO19_IN	GPIO18_IN	GPIO17_IN	GPIO16_IN

GPIOx_IN(x=0:29) variables can be accessed directly from script to read the logic level of particular pin.

GPIO_OUT_L and GPIO_OUT_H register determines the value of a digital pin when it is selected by MCEWizard as output. Writing a 0 to a bit position delivers a low level at the corresponding output pin. A high level is output when the corresponding bit is written with a 1.

GPIO_OUT_L

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GPIO15_OUT	GPIO14_OUT	GPIO13_OUT	GPIO12_OUT	GPIO11_OUT	GPIO10_OUT	GPIO9_OUT	GPIO8_OUT	GPIO7_OUT	GPIO6_OUT	GPIO5_OUT	GPIO4_OUT	GPIO3_OUT	GPIO2_OUT	GPIO1_OUT	GPIO0_OUT

GPIO_OUT_H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	Reserved	GPIO29_OUT	GPIO28_OUT	GPIO27_OUT	GPIO26_OUT	GPIO25_OUT	GPIO24_OUT	GPIO23_OUT	GPIO22_OUT	GPIO21_OUT	GPIO20_OUT	GPIO19_OUT	GPIO18_OUT	GPIO17_OUT	GPIO16_OUT

GPIOx_OUT(x=0:29) variables can be accessed directly from script to write the logic level of particular pin.

```
001        /*Task0 script function*/
002        Script_Task0()
003        {
004          /*Toggle the GPIO2 pin using bit field*/
005          if(GPIO2_IN)
006          {
007            GPIO2_OUT=0;
008          }
009          else
010          {
011            GPIO2_OUT =1;
012          }
013          /*Toggle the GPIO2 pin using variable*/
014          if(GPIO_IN_L&0x2)
015          {
016            SET_BIT(GPIO_OUT_L,2);
017          }
018          else
019          {
020            CLEAR_BIT(GPIO_OUT_L,2);
021          }
022          /*Toggle the GPIO2 pin using variable*/
023          GPIO_OUT_L = GPIO_OUT_L^0x4;
024        }
```

Code Listing 14 Digital IO Access Example

Analog pins

Analog pins available for users can be enabled using MCEWizard. All enabled analog pins values are read by MCE every 1 ms and the value can be read by script code.

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

Variable Name	Type	Description
ADC_Result0	READONLY	Holds AIN0 analog input value (12 bit value)
ADC_Result1	READONLY	Holds AIN1 analog input value (12 bit value)
ADC_Result2	READONLY	Holds AIN2 analog input value (12 bit value)
ADC_Result3	READONLY	Holds AIN3 analog input value (12 bit value)
ADC_Result4	READONLY	Holds AIN4 analog input value (12 bit value)
ADC_Result5	READONLY	Holds AIN5 analog input value (12 bit value)
ADC_Result6	READONLY	Holds AIN6 analog input value (12 bit value)
ADC_Result7	READONLY	Holds AIN7 analog input value (12 bit value)
ADC_Result8	READONLY	Holds AIN8 analog input value (12 bit value)
ADC_Result9	READONLY	Holds AIN9 analog input value (12 bit value)
ADC_Result10	READONLY	Holds AIN10 analog input value (12 bit value)
ADC_Result11	READONLY	Holds AIN11 analog input value (12 bit value)

Note: If user analog input are not enabled in MCEWizard, ADC_Result variable holds value 0

```
001       /*Task0 script function*/
002       Script_Task0()
003       {
004         /*Start the motor if ADC value is more than 100 count*/
005         if(ADC_Result10>100)
006         {
007           /*Set Target speed value based on ADC input*/
008           TargetSpeed = ADC_Result10<<2;
009           /*Motor start command*/
010           Command=1;
011         }
012         Else /*stop the motor*/
013         {
014           TargetSpeed=0;
015           /*Motor stop command*/
016           Command=0;
017         }
018       }
```

Code Listing 15 Read User Analog pin Example

2.6.11.4.2 Example Script code

A simple example script is described in this section. Example project requirements are listed below

- Run the motor in open loop mode and voltage control
- Set Target speed comments via Analog input.
- Calculate the voltage command based on target speed. Voltage = A*TargetSpeed+B.
- Start command via GPIO input, start motor if GPIO input is high and stop motor if GPIO input is low

Script Implementation

- AIN0 pin is used to read the speed command, AIN0 pin is enabled in MCEWizard.
- GPIO3 pin is for start/stop command. GPIO3 pin is configured as input pin in MCEWizard.
- Voltage = A*TargetSpeed+B, A and B are represented as (Q23.8 Q-format)in script.(A=1.5 is represented in script as 384)
- Execute the script code from Task0 for every 50ms

```
001       /*************************************************************/
002       #SET SCRIPT_USER_VERSION (1.0)
003       #SET SCRIPT_TASK0_EXECUTION_PERIOD    (50)
004       /*************************************************************/
005       int A_Const,B_Const;          /* Global variable definition */
006       /*************************************************************/
007       Script_Task0_init()           /*Task0 init function*/
008       {
009         int volt, Max_Limit;        /*Local variable definition */
010         Max_Limit = 4095;
011         A_Const = 150;
012         B_Const = 150;
013         EnableCoherentUpdate();
014           AngleSelect =0;           /*Set to open loop mode*/
015           CtrlModeSelect =0;        /*voltage control mode */
016         DoCoherentUpdate();
017       }
018       Script_Task0()                /*Task0 script function*/
019       {
020        if(GPIO3_IN==0)              /*Check GPIO3 input level*/
021        {
022          TargetSpeed=0;             /*Set Target Speed to zero*/
023          if(SpdRef<100)             /*Wait until motor rampdown */
024          {
025            Command=0;               /*Motor Stop Command*/
026          }
027        }
028        else
029        {
030          TargetSpeed = ADC_Result0;  /*Set Target Speed from ADC0*/
031          /*Calculate voltage set value*/
032          volt = ((A_Const* SpdRef) + B_Const)>>8;
033
034          if(volt> Max_Limit)         /*Limit Check*/
035          {
036            volt = Max_Limit;
037          }
038          Vd_Ext = volt;              /*Set Vd value*/
039          Command=1;                 /*Motor Start Command*/
040        }
041       }
```

Code Listing 16 Script Example

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 6th bit of 'SysTaskConfig' parameter. See respective product datasheets for detailed specification of achievable accuracy.

3 Register Description

This chapter describes the registers used in MCE. Parameters and variables are scaled within the 16 bit fixed point data range to represent floating-point quantities of the physical value (e.g.: in SI units).

There are two types of parameters used in MCE:

- STATIC : These type of parameters only can be modified/configured from MCEWizard and read from MCEDesigner
- DYNAMIC: These types of parameters can be modified/configured from MCEWizard and read/ write from MCEDesigner

Each parameter or variable can be addressed by using its unique App ID and Index. This section describes each parameter and variable in details.

The Table 26 below describes the scaling from register count values to the corresponding real variable values. The same scaling applies to limit setting or threshold level parameter registers. Notice this assumes DC bus compensation is enabled, and current sensing is centered in the middle of the measurement range. The motor Nominal flux is the motor back EMF constant in Volts per rad/s.

Table 26 Variable Scaling Table

Variable	Count to real scaling	Units	Variable registers
Motor α β current	$\frac{Maximum_I_{sense}}{2048}$	A	Iu, Iv, Iw, I_Alpha, I_Beta
DC bus voltage	$\frac{Maximum_Vdc_{sense}}{4096}$	V	VdcRaw, VdcFilt
Rotor Flux Magnitude	$\frac{Motor\ Nominal\ Flux}{2048}$	Wb	Flx_M, Flx_Q
Rotor angle	$\frac{180^o}{32768}$	°	FluxAngle, HallAngle, RotorAngle
Rotor α β Flux	$\frac{Motor\ Nominal\ Flux}{1244}$	Wb	FluxAlpha, FluxBeta
Motor α β Voltage	$\frac{1}{3}\frac{Max_Vdc_{sense}}{8191}$	V	V_Alpha, V_Beta
Motor d-q current	$\frac{Motor\ Rated\ Current}{4096}$	A rms	Id, Iq, IdFilt, IqFilt, TrqRef, MotorCurrent, TrqRef_Comp
Motor d-q voltage	$\frac{1}{3\sqrt{2}}\frac{Max_Vdc_{sense}}{4974}$	V rms	Vd, Vq, MotorVoltage
Motor Speed	$\frac{Maximum\ Speed}{16384}$	RPM	TargetSpeed, MotorSpeed, abs_MotorSpeed, SpeedError, SpdRef, HallMotorSpeed, FluxMotorSpeed
ADC voltage input	$\frac{V_{DD}}{4096}$	V	ADC_Result0..11
PFC current	$\frac{Max_I_{sense}}{4096}$	A	PFC_IpfcRaw, PFC_IpfcAvg,
PFC rms current	$\frac{Max_I_{sense}}{4096}$	A rms	PFC_IpfcRMS
PFC AC voltage	$\frac{Max_VAC_{sense}}{4096}$	V	PFC_AbsVacRaw, PFC_VacRaw

Variable	Count to real scaling	Units	Variable registers
PFC AC rms voltage	$\frac{Max_VAC_{sense}}{4096}$	V rms	PFC_VacRMS
PFC DC bus voltage	$\frac{Max_VDC_{sense}}{4096}$	V	PFC_TargetVolt, PFC_VdcRaw, PFC_VdcFilt
PFC power	$\frac{PFC_Current_Scaling \cdot PFC_VAC_Scaling}{128}$	W	PFC_ACPower

3.1 System Control Register (App ID =0)

3.1.1 ParPageConf

Index	0		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:0xFFFF	Default: 0

Scaling or Notation: Bit field defined

Description: This parameter defines parameter page selection method and default parameter page

[3:0] Parameter page selection
- 0- No Selection
- 1- Parameter page selection via UART
- 2- Parameter page selection via Analog input
- 3- Parameter page selection via digital input

[7:4] Default parameter page number

[15:8] Reserved

3.1.2 InterfaceConf0

Index	2		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:0xFFFF	Default: 0
Scaling or Notation:	Bit field defined		

Description:

This parameter defines UART0 and UART1 configuration.

[2:0] UART0 function
000: UART0 not used
001: UART0 is used for User UART
111: UART0 is used for MCEDesigner communication
Others: reserved

[3] UART0 TxD output configuration
0: push-pull output
1: open-drain output

[6:4] UART1 function
000: UART1 not used
001: UART1 is used for User UART
111: UART1 is used for MCEDesigner communication
Others: reserved

[7] UART1 TxD output configuration
0: push-pull output
1: open-drain output

[15:8] Transmission delay configuration
1 = 1 ms
Min: 0; Max: 255; Default: 0

3.1.3 InterfaceConf1

Index	3		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:0xFFFF	Default: 0
Scaling or Notation:	Bit field defined		

Description:

This parameter configures JCOM interface.

[2:0] UART0 function
000: JCOM interface is not used by MCE
011: JCOM interface is used for inter-core communication by MCE
Others: reserved

[15:3] Reserved

3.1.4 SysTaskTime

Index	62		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:1000	Default: 1
Scaling or Notation:	1 count = 1ms		
Description:	This parameter defines the execution rate of state machine. The default value is 1 and it is not recommended to be changed by the users.		

3.1.5 CPU_Load

Index	80		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max:1000	Default: 0
Scaling or Notation:	CPU load is represented in %. 1 = 0.1%		
Description:	CPU load is calculated in real-time. This parameter holds the value of CPU load value.		

3.1.6 CPU_Load_Peak

Index	84		
Size	Unsigned 16 bit		
Variable Type	Read Write		
Range	Min: 0	Max:1000	Default: 0
Scaling or Notation:	CPU load is represented in %. 1 = 0.1%		
Description:	CPU load peak is calculated in real-time. This parameter holds the peak value of CPU load value. This value can be reset to start over the tracking of CPU load peak value.		

3.1.7 FeatureID_selectH

Index	61		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:0xFFFF	Default: 0
Scaling or Notation:	Bit field defined		
Description:	This parameter defines enable or disable motor control and PFC [0] Enable PFC : 0-Disable, 1 -Enable [7:1] Reserved [8] Enable Motor control : 0-Disable, 1 -Enable [15:9] Reserved		

3.1.8 GKConf

Index	22		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:0xFFFF	Default: 0
Scaling or Notation:	Bit field defined		
Description:	This parameter defines gate kill input source for motor control and pfc. Refer section 2.1.16.3 [1:0] Comparator 0 usage : 0- used by motor, 1- used by PFC [2] Comparator 0 enable : 0-Disable, 1 -Enable [4:3] Comparator 1 usage : 0- used by motor, 1- used by PFC [5] Comparator 1 enable : 0-Disable, 1 -Enable [7:6] Comparator 2 usage : 0- used by motor, 1- used by PFC [8] Comparator 2 enable : 0-Disable, 1 -Enable [10:9] Comparator 3 usage : 0- used by motor, 1- used by PFC [11] Comparator 3 enable : 0-Disable, 1 -Enable [12] Motor control gate kill pin enable :0- Disable, 1 -Enable [15:13] Reserved		

3.1.9 SW_Version

Index	82		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max:0xFFFF	Default: 0
Scaling or Notation:	Bit field defined		
Description:	SW version scheme - 4:6:6 Bit Coding [5:0] Test version [11:6] Minor version [15:12] Major version		

3.1.10 InternalTemp

Index	81		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max: 65535	Default: 0
Scaling or Notation:	InternalTemp is represented in Kelvin. Ex: 300 = 300K.		
Description:	This parameter holds the sampled value of the internal temperature sensor. It is updated every $20 \cdot SysTaskTime\ (ms)$.		

3.1.11 SysTaskConfig

Index	63		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max: 65535	Default: Set by MCEWizard

Scaling or Notation: Bit field defined

Description: This parameter controls certain system functions as described below.

[0] Enable MCEDesigner agent
- 0: Disable
- 1: Enable

[5:1] Reserved

[6] Enable Internal Oscillator Temperature Compensation
- 0: Disable
- 1: Enable

[7] Reserved

[8] Enable background task monitor reset
- 0: Disable
- 1: Enable

[15:9] Reserved

3.1.12 GPIOs[x] (x: 0 - 29)

Index	24 - 53		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max: 65535	Default: Set by MCEWizard

Scaling or Notation: Bit field defined

Description: This parameter controls the behavior of the specific GPIO pin as described below.

- [3:0] Port IO number, configured by MCEWizard
- [6:4] Port number, configured by MCEWizard
- [7] Default output level when configured as output
  - 0: Logic low
  - 1: Logic high
- [8] Output mode when configured as output
  - 0: Push-pull
  - 1: Open drain
- [9] Hysteresis mode when configured as input (refer to product datasheet for specific input voltage thresholds for different logic levels)
  - 0: Standard hysteresis
  - 1: Large hysteresis
- [11:10] Internal pull-up / pull-down mode when configured as input
  - 0: No internal pull-up / pull-down
  - 1: Internal pull-down
  - 2: Internal pull-up
- [12] Invert mode when configured as input
  - 0: Direct input
  - 1: Inverted input
- [13] Reserved
- [14] Input / output configuration, configured by MCEWizard
  - 0: Input
  - 1: Output
- [15] Pin availability, configured by MCEWizard
  - 0: Disable
  - 1: Enabled

GPIO[0] to GPIO[6] are reserved for the implemention of certain special functions as summarized in the following Table 27.

Table 27 Special Function Summary for GPIOs

GPIO Register	Special Function	Description
GPIOs[0]	LED	This pin is used to drive an LED indicator to show fault status.
GPIOs[1]	PGOUT	This pin is used to output a pulse signal as an indication of the motor speed.

GPIO Register	Special Function	Description
GPIOs[2]	PAR0	This pin is used for multiple parameter set selection.
GPIOs[3]	PAR1	This pin is used for multiple parameter set selection.
GPIOs[4]	PAR2	This pin is used for multiple parameter set selection.
GPIOs[5]	PAR3	This pin is used for multiple parameter set selection.
GPIOs[6]	DIR	This pin is used for direction selection.

3.2 Motor Control Register (App ID =1)

Complete list of parameter and variables are listed in the Table 28 and Table 29 and find description in the following chapters.

Table 28 Motor control Parameter list

App ID	Index	Parameter Name	Type	Description
1	1	HwConfig	STATIC	Application hardware configuration parameter
1	2	SysConfig	STATIC	System configuration parameter
1	3	AngleSelect	DYNAMIC	Angle selection from flux or open loop or external
1	4	CtrlModeSelect	DYNAMIC	Control mode : Speed control, Current control or Voltage control
1	5	PwmFreq	STATIC	Motor PWM frequency
1	6	PwmDeadtimeR	STATIC	PWM dead time during leading edge (raising edge of high side switch output)
1	7	PwmDeadtimeF	STATIC	PWM dead time during trailing edge (falling edge of high side switch output)
1	8	SHDelay	DYNAMIC	Switch delay from PWM output to ADC sample time to avoid ADC sample during switching
1	9	TMinPhaseShift	DYNAMIC	Minimum time of an active vector for single shunt current measurement in phase shift PWM mode
1	10	TCntMin	DYNAMIC	Minimum time of an active vector for single shunt current measurement (minimum pulse width method)
1	11	PwmGuardBand	DYNAMIC	Minimum time of null vector for phase current measurement
1	12	FaultEnable	DYNAMIC	Enable or disable fault condition handling. When a fault bit is not set, the fault condition is ignored
1	13	VdcOvLevel	DYNAMIC	DC bus over voltage trip level
1	14	VdcUvLevel	DYNAMIC	DC bus under voltage trip level
1	15	CriticalOvLevel	DYNAMIC	DC bus critical over voltage trip level

App ID	Index	Parameter Name	Type	Description
1	16	RotorLockTime	DYNAMIC	Rotor lock fault detection time
1	18	FluxFaultTime	DYNAMIC	PLL out of synchronous fault detection time
1	19	GatekillFilterTime	STATIC	Persistence filter time for PWM gate kill input
1	20	CompRef	STATIC	Overcurrent trip level for gate kill input
1	21	BtsChargeTime	DYNAMIC	Bootstrap capacitor charging time
1	22	TCatchSpin	DYNAMIC	Catch spin synchronization time duration before engaging motor start acceleration
1	23	DirectStartThr	DYNAMIC	Catch Spin threshold speed limit for free running motor that decides whether to run directly in close loop FOC or configured startup mode
1	24	ParkTime	DYNAMIC	Total parking time of the rotor during startup
1	25	ParkAngle	DYNAMIC	Rotor alignment angle during parking
1	26	OpenloopRamp	DYNAMIC	Open loop speed acceleration rate
1	27	IS_Pulses	DYNAMIC	Number of PWM cycles for each inductor sensing pulse under nominal DC bus voltage
1	28	IS _Duty	DYNAMIC	PWM duty cycle during ANGLE_SENSING
1	29	IS_IqInit	DYNAMIC	Initial torque been applied after done ANGLE_SENSING stage and before entering MOTOR_RUN.
1	30	KpSreg	DYNAMIC	Proportional gain of the speed regulator
1	31	KxSreg	DYNAMIC	Integral gain of the speed regulator
1	32	MotorLim	DYNAMIC	Maximum allowable motor current (d axis and q axis)
1	33	RegenLim	DYNAMIC	Maximum allowable motor current (d axis and q axis)while motor is running in regenerative mode
1	34	RegenSpdThr	DYNAMIC	Switch over speed threshold between RegenLim and MotorLim motor current limits
1	35	LowSpeedLim	DYNAMIC	Maximum allowable motor current (d axis and q axis) at low speed
1	36	LowSpeedGain	STATIC	Increment rate of Motor current limit after Minspd Threshold
1	37	SpdRampRate	DYNAMIC	close loop speed acceleration rate
1	38	MinSpd	DYNAMIC	Minimum allowed drive operating speed
1	39	Rs	STATIC	Per phase winding resistance of the motor
1	40	L0	STATIC	Per phase winding inductance of the motor at rated current
1	41	LSlncy	STATIC	Saliency inductance of the motor
1	42	VoltScl	DYNAMIC	Internal scaling factor between voltage and flux
1	43	PllKp	DYNAMIC	Angle Frequency Generator tracking proportional gain
1	44	PllKi	DYNAMIC	Angle Frequency Generator tracking integral gain
1	45	PllFreqLim	DYNAMIC	Frequency limit of the PLL integral gain output
1	46	AngMTPA	DYNAMIC	Angle compensation

App ID	Index	Parameter Name	Type	Description
1	47	FlxTau	STATIC	Define by flux estimator time constant. Used to adjustment for the flux estimator bandwidth.
1	48	AtanTau	DYNAMIC	Angle compensation for the phase shift introduced by flux integration time constant
1	49	SpeedScalePsc	STATIC	Speed scale prescaler value. Used for internal scaling between rotor frequency and motor speed
1	50	SpeedScale	STATIC	Internal scaling factor between rotor frequency and motor speed. Convert rotor frequency to motor speed
1	51	SpeedScaleRcp	STATIC	Internal scaling factor between rotor frequency and motor speed. Convert motor speed to rotor frequency
1	52	SpdFiltBW	DYNAMIC	Low pass filter time constant for motor Speed estimation
1	53	PGDeltaAngle	DYNAMIC	PG output configuration, defines number of pulses per motor revolution
1	54	IfbkScl	STATIC	Internal scaling factor between alpha/beta current 1to d/q current. Current scale to represent d axis and q axis current rated motor current.
1	55	KpIreg	DYNAMIC	Proportional gain of q-axis current regulator
1	56	KpIregD	DYNAMIC	Proportional gain of d-axis current regulator
1	57	KxIreg	DYNAMIC	Integral gain of d- and q-axis current regulator
1	58	FwkLevel	DYNAMIC	Modulation threshold to start field weakening
1	59	FwkKx	DYNAMIC	Gain of field weakening control
1	60	FwkCurRatio	DYNAMIC	–Id current limit for field weakening
1	61	VdqLim	DYNAMIC	Current regulator output limit
1	62	AngDel	DYNAMIC	Gain adjustment for current angle advancement
1	63	AngLim	DYNAMIC	Maximum limit on the current angle phase advancement
1	64	IdqFiltBW	DYNAMIC	low pass filter time constant for Id and Iq
1	65	Pwm2PhThr	DYNAMIC	Switch over speed from 3 phase PWM to 2 phase PWM
1	66	TDerating	STATIC	Reserved for future use.
1	67	TShutdown	DYNAMIC	Over-temperature shutdown threshold
1	68	CmdStop	STATIC	Motor stop threshold value for Vsp/frequency/duty cycle control inputs. If the input value is less than threshold, motor will be stopped.
1	69	CmdStart	STATIC	Motor start threshold value for Vsp/frequency/duty cycle control inputs. If the input value is more than threshold, motor will be started.
1	70	CmdGain	STATIC	Slope of set speed value for Vsp/frequency/duty cycle control inputs.
1	71	AppConfig	DYNAMIC	Application configuration Parameter
1	72	NodeAddress	STATIC	Node Address

App ID	Index	Parameter Name	Type	Description
1	73	PrimaryControlLoop	DYNAMIC	Primary control loop
1	74	PhaseLossLevel	DYNAMIC	Phase loss detection current level
1	75	TrqCompGain	DYNAMIC	Torque Compensation gain factor
1	76	TrqCompAngOfst	DYNAMIC	Torque compensation angle offset value
1	77	TrqCompLim	DYNAMIC	Compensation torque reference limit value
1	78	TrqCompOnSpeed	DYNAMIC	Torque compensation ON speed threshold value
1	79	TrqCompOffSpeed	DYNAMIC	Torque compensation OFF speed threshold value
1	80	PolePair	DYNAMIC	Motor pole pair value
1	81	FaultRetryPeriod	DYNAMIC	Retry period after fault if control input is frequency, duty or VSP
1	85	HallAngleOffset	DYNAMIC	Hall sensor alignment offset value.
1	86	Hall2FluxThr	DYNAMIC	Switch-over speed threshold from using Hall angle to using flux angle.
1	87	Flux2HallThr	DYNAMIC	Switch-over speed threshold from using flux angle to using Hall angle.
1	88	HallSampleFilter	STATIC	Hall sample filter de-bounce time.
1	89	HallSpdFiltBW	DYNAMIC	Time constant of the digital low-pass filter for Hall frequency.
1	94	HallTimeoutPeriod	DYNAMIC	Hall sensor time-out fault period.
1	100	KpHallPLL	DYNAMIC	Proportional gain for Hall_FrequencyAdjust.

Table 29 Motor control Variable list

App ID	Index	Variable Name	Type	Description
1	120	Command	READWRITE	Controls the system state - Stop/ start the motor
1	121	TargetSpeed	READWRITE	Target speed of the motor, when the drive is in speed control mode.
1	122	Iu	READONLY	Reconstructed motor phase U current
1	123	Iv	READONLY	Reconstructed motor phase V current
1	124	Iw	READONLY	Reconstructed motor phase W current
1	125	MotorSpeed	READONLY	Filtered motor running speed
1	126	I_Alpha	READONLY	Ialpha current
1	127	I_Beta	READONLY	Ibeta current
1	128	IdRef_Ext	READWRITE	Current command on d axis, when the drive is in current control mode.
1	129	IqRef_Ext	READWRITE	Current command on q axis, when the drive is in current control mode.
1	130	Vd_Ext	READWRITE	Vd command when the drive is in voltage control mode.

App ID	Index	Variable Name	Type	Description
1	131	Vq_Ext	READWRITE	Vq command when the drive is in voltage control mode.
1	132	SwFaults	READONLY	Drive fault status based on fault condition and fault mask
1	133	SequencerState	READONLY	Current state of the drive
1	134	FaultClear	READWRITE	Fault clear
1	135	FaultFlags	READONLY	Drive fault status based on fault condition
1	136	VdcRaw	READONLY	DC bus voltage
1	137	VdcFilt	READONLY	DC bus filtered voltage
1	138	FluxAngle	READONLY	Estimated rotor Angle
1	139	Flx_M	READONLY	Fundamental flux amplitude
1	140	abs_MotorSpeed	READONLY	Absolute motor speed
1	141	IdFilt	READONLY	Id current filter value
1	142	IqFilt	READONLY	Iq current filter value
1	143	IdFwk	READONLY	Id field weakening current value
1	144	VTH	READONLY	NTC temperature value
1	145	FluxAlpha	READONLY	Flux alpha component value
1	146	FluxBeta	READONLY	Flux beta component value
1	147	Flx_Q	READONLY	Net Flux on Q axis value
1	148	TrqRef	READONLY	Torque Reference, Speed PI output
1	149	Id	READONLY	Id current value
1	150	Iq	READONLY	Iq current value
1	151	V_Alpha	READONLY	Voltage alpha component value
1	152	V_Beta	READONLY	Voltage beta component value
1	153	SpeedError	READONLY	Speed PI error value
1	154	MotorCurrent	READONLY	Motor current value
1	155	OpenLoopAngle	READWRITE	Open loop Angle
1	156	Vd	READONLY	Motor Vd voltage value
1	157	Vq	READONLY	Motor Vq voltage value
1	158	MotorVoltage	READONLY	Motor voltage value
1	159	TrqRef_Ext	READONLY	Desired compensation torque reference value from torque compensation function block
1	162	SpdRef	READONLY	Speed Reference output from Speed ramp function. Input to Speed PI controller
1	164	ControlFreq	READONLY	Input signal frequency measurement result
1	165	ControlDuty	READONLY	Input signal duty cycle measurement result
1	167	HallAngle	READONLY	Hall sensor-based rotor angle.
1	168	HallMotorSpeed	READONLY	Hall sensor-based motor speed.
1	169	FluxMotorSpeed	READONLY	Flux estimator based motor speed.
1	170	RotorAngle	READONLY	Rotor angle.

App ID	Index	Variable Name	Type	Description
1	171	MotorStatus	READONLY	Angle type, PWM modulation type, and phase shift PWM scheme.
1	175	PositionCounter	READONLY	Lower 16 bit of the Hall position counter.
1	176	PositionCounter_H	READONLY	Higher 16 bit of the Hall position counter.
1	181	HallStatus	READONLY	Hall driver status.
1	182	Hall_FrequencyOut	READONLY	
1	183	HallPLL_FrequencyAdjust	READONLY	
1	184	Hall_Atan_Angle	READONLY	Estimated rotor angle using Atan calculation method for analog Hall sensors
1	185	HallU	READONLY	This variable provides the subtraction result of analog Hall U+ input ADC sample value and analog Hall U- input ADC sample value.
1	186	HallV	READONLY	This variable provides the subtraction result of analog Hall V+ input ADC sample value and analog Hall V- input ADC sample value.
1	187	Ipeak	READONLY	Peak amplitude of the motor phase current.
1	188	CurrentAmpOffset0	READWRITE	Phase U current sensing offset value.
1	189	CurrentAmpOffset1	READWRITE	Phase V current sensing offset value.
1	190	TrqRef_Total	READONLY	Total torque reference value that includes TrqRef from speed PI output and TrqRef_Ext from the torque compensation function.
1	191	TrqCompBaseAngle	READONLY	Base angle value that is used in synthesizing a sinusoidal compensation torque reference.
1	194	TrqCompStatus	READONLY	Torque compensation function status.

3.2.1 Control Register Group

3.2.1.1 HwConfig

Index	1		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 0xFFFF	Default: 0x0120

Scaling or Notation: Bit field definitions are mentioned in description

Description: Motor control application hardware configuration parameter

[0] Current Shunt Type
- 0- Single Shunt current sensing
- 1- Leg Shunts current sensing (2 Phase current sensing)

[2:1] Reserved

[4:3] PWM Mode
- 0- 3 Phase PWM only
- 3- 2 Phase Type 3 PWM

[5] Minimum Pulse (single shunt only)
- 0- Use phase shift for narrow pulse
- 1- Use minimum pulse for narrow pulse

[6] Active polarity for Low side PWM outputs
- 0- Active level is high
- 1- Active level is low

[7] Active polarity for High side PWM outputs
- 0- Active level is high
- 1- Active level is low

[8:9] Internal gain for current measurement
- 0- Internal gain is 1
- 1- Internal gain is 3
- 2- Internal gain is 6
- 3- Internal gain is 12

[12] Low noise phase shift PWM enable
- 0- Disable low noise phase shift PWM
- 1- Enable low noise phase shift PWM

[15:13] CurrentOffsetCal_Psc value (Refer to section 2.1.3 for details.)

Attention: ***In MCEWizard current shunt type shall be configured as per actual hardware. Wrong configuration may leads to damage of switches due to over current.***

Attention: ***In MCEWizard active polarity of high side and low side switches shall be configured as per actual hardware. Wrong configuration may lead to short circuits in switches.***

3.2.1.2 SysConfig

Index	2		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 0xFFFF	Default: Set by MCEWizard

Scaling or Notation: Bit field definitions are mentioned in description

Description: Motor control system configuration parameter

[0] DC bus voltage compensation
- 0- Disabled
- 1- Enabled

[1] Reserved

[5:2] Execution rate for current control loop (Fast Control Rate).
- 1- Current control loop executed every PWM period
- 2- Current control loop executed every 2 PWM period
- …
- 15- Current control loop executed every 15 PWM period

[7:6] Reserved

[10:8] Hall input, hold number of hall inputs
- 000_b – No hall sensor
- 011_b - 2 Hall sensor input
- 111_b – 3 Hall sensor input

Other values are reserved.

[12:11] Hall Type, Digital or Analog hall sensor
- 0 – Digital hall interface
- 2 – Analog hall interface

[13] Enable Hall Atan angle calculation method
- 0 – Disabled
- 1 – Enabled

[15:14] Analog Hall sensor interface comparator hysteresis configuration
- 0 – 20 mV
- 1 – 15 mV
- 2 – 5 mV
- 3 – 0 mV

3.2.1.3 AngleSelect

Index	3		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 2	Default: Set by MCEWizard
Scaling or Notation:	See description		
Description:	This parameter used to select the rotor angle 0- Open loop angle. Rotating speed is configured by parameter “TargetSpeed”, if Targetspeed=0, open loop angle is fixed can be changed by writing a value to parameter “OpenLoopAngle” 1- Hall angle. Rotor angle is provided by Hall angle. 2- Flux angle. Rotor angle is provided by Flux estimator. 3- Hybrid angle. Rotor angle switches between Hall angle and flux angle. The switch-over threholds are determined by parameters named ‘Hall2FluxThr’ and ‘Flux2HallThr’.		

3.2.1.4 CtrlModeSelect

Index	4		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 2	Default: Set by MCEWizard
Scaling or Notation:	See description		
Description:	This parameter used to select one of three control modes: 0- Open loop voltage control mode, voltage command is Vd_Ext and Vq_Ext 1- Current control mode, current command is IdRef_Ext and IqRef_Ext 2- Speed control mode, speed command is TargetSpeed		

3.2.1.5 APPConfig

Index	71		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 0xFFFF	Default: 0

Scaling or Notation: Bit field definitions are mentioned in description

Description: Motor control system configuration parameter

[2:0] Control input selection (Set target speed value)
0: UART control
1: Vsp analog input
2: Frequency input
3: Duty cycle input

[3] Enable Restart after Fault: Vsp, frequency or duty cycle control input mode, if there is fault condition, motor will stop and start a 10seconds counter. After 10 seconds, the control will try to clear the fault for 10 times. If the fault condition still exists, control will stay in fault condition.
This feature is not available in UART control mode.
0: Disable restart after fault
1: Enable restart after fault

[4] Enable torque compensation
0: Disable
1: Enable

[5] Reserved

[6] Enable control input measurement
0: Disable
1: Enable

[7] Reserved

[15:8] Hall Atan angle active period: this higher 8-bit word defines the number of sectors for which Hall Atan angle is being used as rotor angle during start-up if Hall angle or hybrid angle is selected and Hall Atan angle calculation method is enabled.
0: Atan angle is not used as rotor angle during start-up if Hall angle or hybrid angle is selected and Hall Atan angle calculation method is enabled.
k = 1~254: Atan angle is used as rotor angle for k number of sectors during start-up if Hall angle or hybrid angle is selected and Hall Atan angle calculation method is enabled.
255: Atan angle is always used as rotor angle if Hall angle or hybrid angle is selected and Hall Atan angle calculation method is enabled.

3.2.1.6 PrimaryControlRate

Index	73		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 1	Max: 16	Default: 2
Scaling or Notation:	See description		
Description:	This parameter defines the execution rate of speed control loop. Speed control loop executed every Fast control rate * Primary control rate * PWM period. Speed ramp rate is calculated based on this value.		

3.2.1.7 Command

Index	120		
Size	Unsigned 16 bit		
Variable Type	Read Write		
Range	Min: 0	Max: 1	Default: 0
Scaling or Notation:	See description		
Description:	This variable controls the system state with the following values: 0- Stop the motor 1- Start the motor		

3.2.1.8 SequencerState

Index	133		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max: 9	Default: 0
Scaling or Notation:	See description		
Description:	This variable contains the current sequence state of the drive 0- Power on state 1- Stop state 2- Calculate offset current 3- Charging boot strap capacitors 4- Motor running 5- Fault state 6- Catch spin 7- Parking 8- Open loop acceleration 9- Angle Sensing (Initial rotor angle detection)		

3.2.1.9 MotorStatus

Index	171		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: -32768	Max: 32767	Default: NA
Scaling or Notation:	See description.		
Description:	This variable contains the status of angle type, PWM modulation type, and phase shift PWM scheme. [0] Angle type 0: using Hall angle 1: using flux angle [1] Phase shift PWM scheme 0: using normal phase shift PWM scheme 1: using low noise phase shift PWM scheme [3:2] PWM modulation type 0: using 3-phase PWM modulation 3: using 2-phase PWM modulation		

3.2.2 PWM Register Group

3.2.2.1 PwmFreq

Index	5		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 20	Max:800	Default:160
Scaling or Notation:	1 = 0.1 kHz F_{PWM} ; 160 = 16kHz F_{PWM}		
Description:	This parameter configures the motor PWM frequency in 0.1 kHz increment. PWM Period value = 96,000,000/(2*PWMFreq[Hz])		

3.2.2.2 PWMDeadtimeR

Index	6		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:240	Default:48
Scaling or Notation:	$1 = 20.8333ns$		
Description:	PWM dead time during leading edge (raising edge of high side switch output)		

3.2.2.3 PWMDeadtimeF

Index	7		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:240	Default:48
Scaling or Notation:	$1 = 20.8333ns$		
Description:	PWM dead time during trailing edge (falling edge of high side switch output)		

3.2.2.4 SHDelay

Index	8		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: -192	Max:960	Default: 0
Scaling or Notation:	$1 = 10.4167ns$		
Description:	SHDelay specifies the time delay from PWM output to ADC sample time for current sensing. The delay time is depending on the hardware design; usually it should consider propagation delay of gate driver circuit and turn on (turn off) delay of switching devices. In Phase Shift PWM mode, in order to avoid sample the shunt resistor signal while device is switching, SHDelay should be configured smaller than actual hardware delay. Some board design may allow bigger SHDelay value without causing much current sensing noise (bigger SHDelay value may help for a smaller TMinPhaseShift value). In Minimum Pulse PWM mode, SHDelay should be configured same as actual hardware delay.		

3.2.2.5 TMinPhaseShift

Index	9		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:960	Default: 0
Scaling or Notation:	$1 = 10.4167ns$		
Description:	In Phase Shift PWM mode, TMinPhaseShift configure the minimum time of an active vector for single shunt current sensing.		

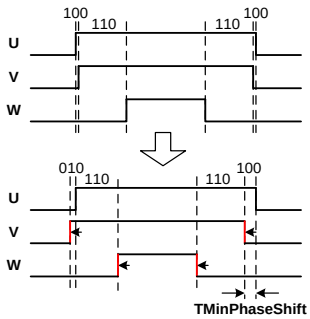


Figure 76 TminphaseShift PWM

3.2.2.6 TCntMin

Index	10		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:960	Default: 0
Scaling or Notation:	$1 = 10.4167ns$		
Description:	This parameter specifies the minimum PWM pulse width if minimum pulse width method is being used.		

3.2.2.7 PwmGuardBand

Index	11		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:960	Default: 0
Scaling or Notation:	$1 = 10.4167ns$		
Description:	In leg shunt configuration, this parameter provides a guard band such that PWM switching at high modulation cannot migrate into the beginning and end of a PWM cycle. The guard band insertion can improve feedback noise immunity for signals sampled near the beginning and end of a PWM cycle. Guard band insertion will reduce the maximum achievable inverter output voltage.		

3.2.2.8 Pwm2PhThr

Index	65		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: 0
Scaling or Notation:	16383= Motor Max RPM		
Description:	Switch over speed from 3 phase PWM to 2 phase PWM.		

3.2.3 Speed Control Register Group

3.2.3.1 KpSreg

Index	30		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: Set by MCEWizard
Scaling or Notation:	U8.8		
Description:	This parameter specifies the proportional gain of the speed regulator.		

3.2.3.2 KxSreg

Index	31		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default:12
Scaling or Notation:	U0.16		
Description:	This parameter specifies the integral gain of the speed regulator		

3.2.3.3 MotorLim

Index	32		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:16383	Default:4095
Scaling or Notation:	4095 = 100% motor rated current		
Description:	This parameter specifies the maximum allowable total motor current (d axis and q axis)		

3.2.3.4 RegenLim

Index	33		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:16383	Default:409
Scaling or Notation:	4095= 100% motor rated current		
Description:	This parameter specifies the maximum total motor current (d axis and q axis) while motor is running in regenerative mode. RegenLim should be set to a low value if the drive has no break resistor otherwise regenerative current will raise the DC bus voltage and cause fault condition.		

3.2.3.5 RegenSpdThr

Index	34		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:16383	Default:600
Scaling or Notation:	16383 = Motor Max RPM		
Description:	This parameter specifies the switch over speed threshold between RegenLim and MotorLim.		

3.2.3.6 LowSpeedLim

Index	35		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:16383	Default:2047
Scaling or Notation:	4095 = 100% motor rated current.		
Description:	This parameter specifies the maximum allowable motor current (d axis and q axis) at low speed (motor speed value less than or equal to Minspd value). Refer section 2.1.7.2.3 for more information.		

3.2.3.7 LowSpeedGain

Index	36		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	U16.0		
Description:	This parameter specifies the increment rate of Motor current limit between MinSpd and low speed threshold. Refer section 2.1.7.2.3 for more information.		

3.2.3.8 SpdRampRate

Index	37		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: 0
Scaling or Notation:	See description		
Description:	This parameter (Q11) specifies the ramp rate of target speed reference.		

$$SpdRampRate = \frac{Speed\ Ramp\ Rate(RPM/s) \cdot 16383}{Motor\ Max\ Speed(RPM)} \cdot \frac{Primary\ Control\ Rate \cdot Fast\ Control\ Rate}{F_{PWM}(Hz)} \cdot 2^{11}$$

3.2.3.9 MinSpd

Index	38		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:16383	Default: 600
Scaling or Notation:	16383 = Motor Max RPM		
Description:	This parameter configures the minimum motor speed. Motor will run at MinSpd when the target motor speed is below MinSpd.		

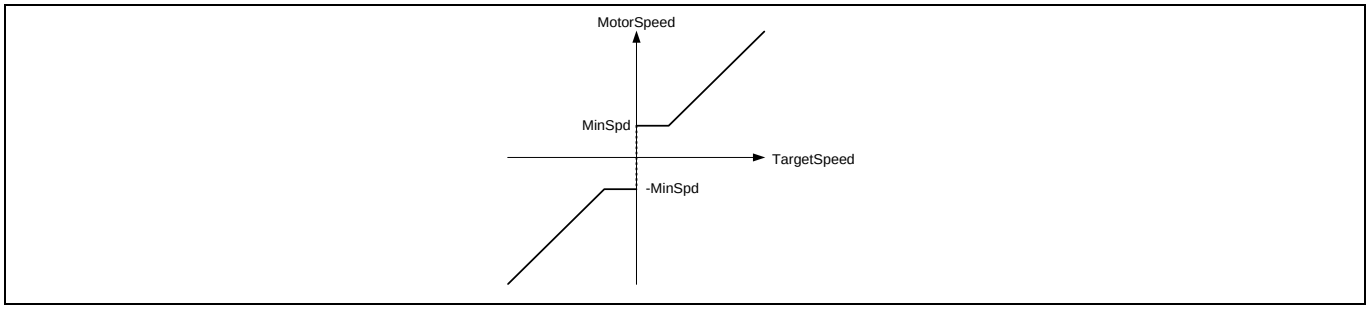


Figure 77 Minimum Speed

3.2.3.10 TargetSpeed

Index	121		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -32767	Max:32767	Default: 0
Scaling or Notation:	16383 = Motor Max RPM		
Description:	This variable sets the target speed of the motor, when the drive is in speed control mode. If the motor is running in Vsp analog input, frequency input or duty cycle control, this variable will be updated by software and writing to it has no effect.		

3.2.3.11 TrqRef

Index	148		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -4095	Max:4095	Default: 0
Scaling or Notation:	4095= 100% motor rated RMS current		
Description:	This variable holds the value of Speed PI output value.		

3.2.3.1 SpdRef

Index	162		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -32767	Max:32767	Default: 0
Scaling or Notation:	16383 = Motor Max RPM		
Description:	Speed Reference output from Speed ramp function. Input to Speed PI controller		

3.2.3.2 RotorAngle

Index	170		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -32768	Max: 32767	Default: NA
Scaling or Notation:	90° = 16384		
Description:	This variable represents the rotor angle being used by speed control loop.		

3.2.4 Hall Sensor InterfaceRegister Group

3.2.4.1 HallAngleOffset

Index	85		
Size	Signed 16 bit		
Parameter Type	DYNAMIC		
Range	Min: -32768	Max: 32767	Default: Set by MCEWizard
Scaling or Notation:	90° = 16384		
Description:	This parameter specifies the offset value that the MCE uses to compensate for the angle difference between the rotor position and the Hall sensor (DHALL1 or AHALL1) mounting position. Refer to Section 2.1.14.7 for details.		

3.2.4.2 Hall2FluxThr

Index	86		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 32767	Default: Set by MCEWizard
Scaling or Notation:	16383 = motor max. RPM		
Description:	This parameter specifies the switch-over speed threshold from using Hall angle to using flux angle in hybrid angle mode.		

3.2.4.3 Flux2HallThr

Index	87		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 16383	Default: Set by MCEWizard
Scaling or Notation:	16383 = motor max. RPM		
Description:	This parameter specifies the switch-over speed threshold from using flux angle to using Hall angle in hybrid angle mode.		

3.2.4.4 HallSampleFilter

Index	88		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 32767	Default: Set by MCEWizard
Scaling or Notation:	1 = 10.417 ns		
Description:	This parameter specifies the de-bounce time used by the Hall sample noise filter.		

3.2.4.5 HallSpdFiltBW

Index	89		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 32767	Default: Set by MCEWizard
Scaling or Notation:	See description		
Description:	This parameter specifies the time constant of the digital low-pass filter (LPF) for Hall frequency. The following pseudo code shows the LPF implementation in SW. $$filter(n) = filter(n-1) + \big(input(n) - output(n-1)\big) \cdot HallSpdFiltBW$$ $$output(n) = filter(n) \gg 16$$ Since Hall frequency is updated every motor PWM cycle, the sampling frequency F_s is the same as the motor PWM frequency. The time constant of the LPF for Hall frequency can be calculated as follows. $$T_{decay} = -\frac{Fast_Control_Rate}{F_{PWM} \cdot Ln(1 - \frac{HallSpdFiltBW}{2^{16}})}$$		

3.2.4.6 HallTimeoutPeriod

Index	94		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 65565	Default: Set by MCEWizard
Scaling or Notation:	1 = 10 ms		
Description:	This parameter specifies the Hall timeout fault detection time. If the Zero Frequency Flag bit in parameter 'HallStatus' is asserted for the duration of HallTimeoutPeriod x 10 ms, then Hall timeout fault bit in parameter 'FaultFlags' is asserted.		

3.2.4.7 KpHallPLL

Index	100		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 65535	Default: Set by MCEWizard
Scaling or Notation:	U4.12		
Description:	This variable specifies the proportional gain for parameter 'HallPLL_FrequencyAdjust'. It is recommended to set this variable to a value between 0 and 4096. 0: Hall PLL function is disabled. ≠0: Hall PLL function is enabled.		

3.2.4.8 HallAngle

Index	167		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -32768	Max: 32767	Default: NA
Scaling or Notation:	90° = 16384		
Description:	This variable represents the rotor angle based on Hall sensors.		

3.2.4.9 HallMotorSpeed

Index	168		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -32768	Max: 32767	Default: NA
Scaling or Notation:	16383 = motor max. RPM		
Description:	This variable represents the motor speed based on Hall sensors.		

3.2.4.10 PositionCounter

Index	175		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max: 65535	Default: NA
Scaling or Notation:	6 = 1 electrical revolution		
Description:	This variable represents the lower 16 bit of the Hall position counter, which is updated during each Hall event.		

3.2.4.11 PositionCounter_H

Index	176		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max: 65535	Default: NA
Scaling or Notation:	1 = PositionCounter overflows from 65535		
Description:	This variable represents the higher 16 bit of the Hall position counter, which is updated during each Hall event.		

3.2.4.12 HallStatus

Index	181		
Size	Unsigned 16 bit		
Parameter Type	Read only		
Range	Min: 0	Max: 65535	Default: NA
Scaling or Notation:	Bit field definitions are mentioned in description		
Description:	Hall sensor interface driver status parameter. [0] Actual motor direction 0: Clockwise direction 1: Counter-clockwise direction [1] Motor direction change 0: No direction change 1: direction changed [2] Zero frequency flag 0: no zero frequency fault 1: zero frequency fault [3] Reserved [6:4] Sector number 0~5: valid pattern 7: invalid pattern [7] Wide sector flag 0: The last sector just before the latest capture is a normal sector (60°). The current sector after the latest capture is a wide sector (120°). 1: The last sector just before the latest capture is a wide sector (120°). The current sector after the latest capture is a normal sector (60°). [10:8] Reserved [11] Target direction 0: Clockwise direction 1: Counter-clockwise direction [15:12] Reserved		

3.2.4.13 Hall_FrequencyOut

Index	182		
Size	Signed 16 bit		
Parameter Type	STATIC		
Range	Min: -32768	Max: 32767	Default: NA
Scaling or Notation:	1 = 0.0055° / PWM cycle		
Description:	This variable represents the low-pass filtered Hall frequency, which is defined as angle change per motor PWM cycle.		

3.2.4.14 HallPLL_FrequencyAdjust

Index	183		
Size	Signed 16 bit		
Parameter Type	STATIC		
Range	Min: -32768	Max: 32767	Default: NA
Scaling or Notation:	1 = 0.0055° / PWM cycle		
Description:	This variable represents the compensation term for Hall frequency when calculating Hall angle. It is defined as angle change per motor PWM cycle.		

3.2.4.15 Hall_Atan_Angle

Index	184		
Size	Signed 16 bit		
Parameter Type	Read only		
Range	Min: -32768	Max: 32767	Default: NA
Scaling or Notation:	90° = 16384		
Description:	This variable represents the estimated rotor angle using Atan calculation method for analog Hall sensors.		

3.2.4.16 HallU

Index	185		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -2047	Max:2047	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the subtraction result of analog Hall U+ input ADC sample value and analog Hall U- input ADC sample value.		

3.2.4.17 HallV

Index	186		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -2047	Max:2047	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the subtraction result of analog Hall V+ input ADC sample value and analog Hall V- input ADC sample value.		

3.2.5 Flux Estimator PLL Register Group

3.2.5.1 Rs

Index	39		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	See description.		
Description:	This parameter is the motor winding resistance term in the flux integrator function. MCEWizard calculates this parameter from the motor per phase resistance (motor + cable), motor flux, feedback gains and the control loop sample rate. It is proportional to but is not independently related to motor resistance.		

3.2.5.2 L0

Index	40		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	See description.		
Description:	This parameter is the motor winding inductance term in the flux integrator function. MCEWizard calculates this parameter from the average per phase inductance, motor flux, feedback gains and the control loop sample rate. It is proportional to but is not independently related to $\frac{Ld+Lq}{2}$.		

3.2.5.3 LSIncy

Index	41		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	See description.		
Description:	This parameter is the motor winding inductance saliency term in the flux integrator function. MCEWizard calculates this parameter from the var per phase inductance, motor flux, feedback gains and the control loop sample rate. It is proportional to but is not independently related to $\frac{Lq-Ld}{2}$.		

3.2.5.4 VoltScl

Index	42		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 65535	Default: 0
Scaling or Notation:	See description.		
Description:	This parameter defines the internal scaling factor between voltage and flux. The value of this parameter is calculated by MCEWizard from user input (PWM frequency, motor poles, DC bus scaling and back EMF Ke). This parameter may need to be adjusted when DC bus compensation is disabled.		

3.2.5.5 PllKp

Index	43		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: 0
Scaling or Notation:	U0.16		
Description:	This parameter specifies the Angle Frequency Generator tracking proportional gain. The Angle Frequency Generator is mainly a phase lock loop (PLL). A larger value of PllKp will increase tracking bandwidth at the expense of increasing speed or frequency ripple.		

3.2.5.6 PllKi

Index	44		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: 0
Scaling or Notation:	U0.16		
Description:	This parameter specifies the Angle Frequency Generator tracking integral gain. The Angle Frequency Generator is mainly a phase lock loop (PLL). The Figure 78 shows both a simplified and the detailed PLL architecture. PllKi relates internal PLL tracking error (q) to frequency (Rtr_Freq). A larger value of PllKi will increase tracking bandwidth at the expense of increasing speed or frequency ripple.		

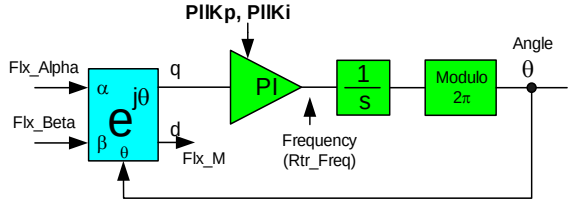


Figure 78 Simplified Block diagram of a FLUX PLL

3.2.5.7 PllFreqLim

Index	45		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 65535	Default: 0
Scaling or Notation:	See description		
Description:	This parameter specifies the frequency limit of the PLL integral gain output. The relationship between the actual frequency in Hz and this parameter is given by: $A = \frac{PllFreqLim * PwmFreq}{8192}$, *in Hz* where: A - Actual frequency in Hz PwmFreq -Inverter pwm frequency in Hz		

3.2.5.8 FlxTau

Index	47		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	See description		
Description:	Motor flux is calculated by integration of estimated voltages. Pure (ideal) integrator cannot be used due to dc offset problem. The integration is done using non-ideal integrator (low pass filter) as shown in the Figure 79. The flux integration time constant (Tau) is an entry of the MCEWizard. Typical range of non-ideal integrator time constant is in the range of 0.01 to 0.025 sec. This parameter provides the adjustment for the flux estimator bandwidth. FlxTau is inversely proportional to the “Flux estimator time constant” entered in MCEWizard. The relationship of the Flux estimator time constant and FlxTau is given by: $$Flux\ estimator\ time\ constant = \frac{2^{18} \times T_{PWM}}{FlxTau} - T_{PWM}, in\ seconds$$ where FlxTm - the Flux estimator time constant[S] T_{PWM} - 1/(PWM switching frequency) [S] This parameter is also used as low pass filter time constant for rotor frequency (Rtr_Freq, which is the output of flux PLL) as well as some internal filtering.		

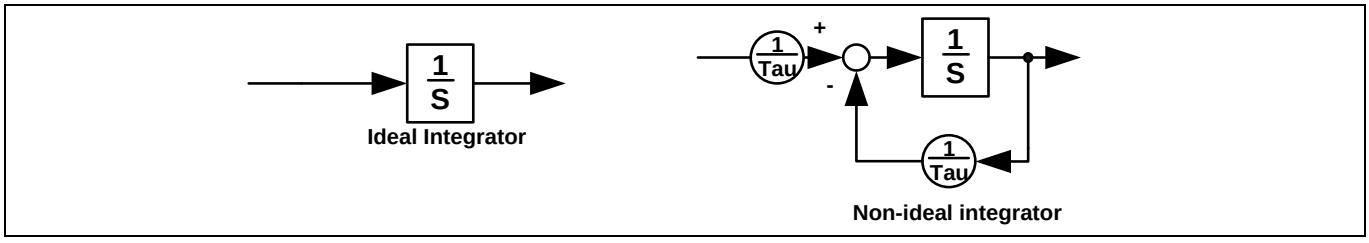


Figure 79 Ideal and Non-ideal Integrator

3.2.5.9 AtanTau

Index	48		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	See description		
Description:	This parameter provides angle compensation (frequency dependent) for the phase shift introduced by flux integration time constant. Rotor frequency (Rtr_Freq) is multiplied by a time constant (AtanTau) to form a compensating angle. This angle represents the phase shift introduced by the non-ideal flux integrators (low pass filter). Pure (ideal) integrator cannot be used due to dc offset problem. The flux integration time constant is an entry of the MCEWizard. Typical range of integrator time constant is in the range of 0.01 to 0.025 sec.		

3.2.5.10 AngMTPA

Index	46		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 16383
Scaling or Notation:	0°→+360° is represented as 0 to 65535		
Description:	This parameter defines the angle compensation for rotor angle calculation		

3.2.5.11 SpdFiltBW

Index	52		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:16383	Default: 0
Scaling or Notation:	U2.14, $\tau = \frac{16384}{SpdFiltBw}, in\ T_{PWM}$		
Description:	This parameter configures the low pass filter time constant for motor Speed. Please note that the input of motor speed calculation low pass filter is rotor frequency (Rtr_Freq), which has already been filtered by FreqBW.		

3.2.5.12 SpeedScale

Index	50		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 65535	Default: 0
Scaling or Notation:	See Description		
Description:	This parameter is the internal scaling factor between rotor frequency and motor speed. The value of this parameter is calculated by MCEWizard tool from user input (PWM frequency, motor poles and motor maximum speed).		

3.2.5.13 MotorSpeed

Index	125		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -32767	Max:32767	Default: 0
Scaling or Notation:	16383 = Motor Max RPM		
Description:	Filtered motor running speed. Filter timer constant is set by parameter SpdFiltBW. Its value will be reset to 0 when the control is not in RUN state.		

3.2.5.14 FluxAngle

Index	138		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -32768	Max:32767	Default: 0
Scaling or Notation:	90° = 16384		
Description:	This is the estimated rotor angle. It is used for the Field-Oriented control reference frame.		

3.2.5.15 Flx_M

Index	139		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:32767	Default: 0
Scaling or Notation:	2048 = 100% rated flux		
Description:	This variable represents the fundamental flux amplitude.		

3.2.5.16 Abs_MotorSpeed

Index	140		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max:32767	Default: 0
Scaling or Notation:	16383 = Motor Max RPM		
Description:	Absolute value of motor Speed.		

3.2.5.17 OpenLoopAngle

Index	155		
Size	Signed 16 bit		
Variable Type	Read Write		
Range	Min: -32768	Max:32767	Default: 0
Scaling or Notation:	90° = 16384		
Description:	If Angle Select is set to 0, use this variable to specify the internal open loop angle.		

3.2.5.18 FluxMotorSpeed

Index	169		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -32768	Max: 32767	Default: NA
Scaling or Notation:	16383 = motor max. RPM		
Description:	This variable represents the motor speed based on flux estimator.		

3.2.6 FOC Register Group

3.2.6.1 IfbkScl

Index	54		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	See description		
Description:	This parameter provides current gain such that 4095 digital counts of d-axis or q-axis current represents rated motor current. IfbkScl is calculated in MCEWizard and is a function of motor rated Amps and analog current scaling.		

$$IfbkScl = \frac{4095 * 1024}{1.647 * AiBiScale * RatedMotorCurrent * \sqrt{2}}$$

Where :
RatedMotorCurrent is in rms Amps

$$AiBiScale = \frac{CurrentInput * InternalADCGain * 4095}{Vadcref}, in\ count/Amps$$

3.2.6.2 KpIreg

Index	55		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: 0
Scaling or Notation:	U4.12		
Description:	This parameter specifies the proportional gain of the Q-axis current regulator. The value of this parameter is calculated by MCEWizard from user input (PWM frequency, motor phase inductance).		

3.2.6.3 KpIregD

Index	56		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: 0

Scaling or Notation: U4.12

Description: This parameter specifies the proportional gain of the d-axis current regulator. The value of this parameter is calculated by MCEWizard from user input (PWM frequency, motor phase inductance, Bandwidth).

3.2.6.4 KxIreg

Index	57		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: 0
Scaling or Notation:	U0.16		
Description:	This parameter specifies the integral gain of the d-axis and q-axis current regulator. The scaling depends on the current regulator execution rate which is directly related to the pwm frequency. The value of this parameter is calculated by MCEWizard from user input (PWM frequency, motor phase resistance, Bandwidth).		

3.2.6.5 FwkVoltLvl

Index	58		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:PWMPeriod	Default:32
Scaling or Notation:	PWMPeriod = 100% PWM duty cycle.		
Description:	This parameter specifies the modulation threshold to start field weakening. It must be set below PWMPeriod and it's also recommended to set this value below SVPWM linear range (PWMPeriod*0.95). Lower threshold gives more voltage margin which provides better control performance but it will enter field weakening mode earlier. Where : PWMPeriod is 96,000,000/(2*PWMFreq[Hz])		

3.2.6.6 FwkKx

Index	59		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:16383	Default:32
Scaling or Notation:	See description		
Description:	This parameter configures the gain of field weakening.		

3.2.6.7 FwkCurRatio

Index	60		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:16383	Default:32
Scaling or Notation:	4096 = 100% MotorLim.		
Description:	This parameter limits the –Id current for field weakening.		

3.2.6.8 VdqLim

Index	61		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4974	Default:32
Scaling or Notation:	4974 = 100 [% modulation]		
Description:	This parameter specifies the current regulator output limit. Refer to Section 2.1.10 for details.		

3.2.6.9 AngDel

Index	62		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	See Description		
Description:	This parameter provides gain adjustment for current angle advancement. The current angle advancement is added to a fixed defaulted phase (90 Deg) and the rotor angle to form the relative phasing of the current vector. Current angle advancement is required for Permanent Magnet motor with rotor saliency (Interior Permanent Magnet Motors). A value of zero represents zero angle advancement and therefore the current vector is placed at 90 degrees with respect to the rotor angle. Diagram below shows the implementation of the angle advancement function and the related controller parameters.		

$$Angle\ advancement = AngDel \times 0.35156 \times \frac{I_Motor}{Rated\ Motor\ Amps}, in\ degree$$

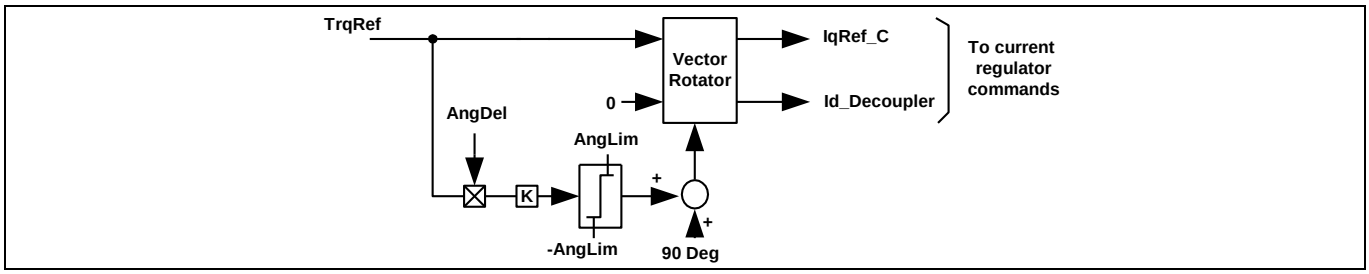


Figure 80 Angle Del

3.2.6.10 AngLim

Index	63		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	$1 = 0.17578\ degree$		
Description:	This parameter provides the maximum limit on the current angle phase advancement specified by parameter AngDel. (See also AngDel.)		

3.2.6.11 IdqFiltBw

Index	64		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:16383	Default:4096
Scaling or Notation:	U2.14 ; $\tau = \frac{16384}{\text{IdqFiltBw}}$, in T_{PWM}		
Description:	This parameter configures the low pass filter time constant for Id and Iq.		

3.2.6.12 IdRef_Ext

Index	128		
Size	Signed 16 bit		
Variable Type	Read Write		
Range	Min: -16383	Max:16383	Default: 0
Scaling or Notation:	4095 = 100% motor rated RMS current		
Description:	This is the reference input of current regulator on D axis. In speed control mode, this variable has no influence. In current control mode, this variable is used as current input.		

3.2.6.13 IqRef_Ext

Index	129		
Size	Signed 16 bit		
Variable Type	Read Write		
Range	Min: -16383	Max:16383	Default: 0
Scaling or Notation:	4095 = 100% motor rated RMS current		
Description:	This is the reference input of current regulator on Q axis. In speed control mode, this variable has no influence. In current control mode, this variable is used as current input.		

3.2.6.14 IdFilt

Index	141		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -16383	Max:16383	Default: 0
Scaling or Notation:	4096 = 100% motor rated RMS current		
Description:	Id current after low pass filter. LPF gain is set by IdqFiltBw parameter.		

3.2.6.15 IqFilt

Index	142		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -16383	Max:16383	Default: 0
Scaling or Notation:	4096 = 100% motor rated RMS current		
Description:	Iq current after low pass filter. LPF gain is set by IdqFiltBw parameter.		

3.2.6.16 IdFwk

Index	143		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -16383	Max:16383	Default: 0
Scaling or Notation:	4096 = 100% motor rated RMS current		
Description:	IdFwk represents the -Id current during field weakening.		

3.2.6.17 Id

Index	149		
Size	Signed 16 bit		
Variable Type	Read Only		
Range	Min: -16383	Max:16383	Default: 0
Scaling or Notation:	4096 = 100% motor rated RMS current		
Description:	This variable holds the motor Id component current		

3.2.6.18 Iq

Index	150		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -16383	Max:16383	Default: 0
Scaling or Notation:	4096 = 100% motor rated RMS current		
Description:	This variable holds the motor Iq component current		

3.2.6.19 MotorCurrent

Index	154		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -16383	Max:16383	Default: 0
Scaling or Notation:	4096 = 100% motor rated RMS current		
Description:	Motor current (IdqFilt) is actual motor phase RMS current. It is calculated as below: $$IdqFilt = (\sqrt{Idfilt^2 + Iqfilt^2})$$		

3.2.7 Measurement Register Group

3.2.7.1 Iu

Index	122		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -2047	Max:2047	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides reconstructed motor phase U current (offset eliminated and ADC compensated). This current is calculated from the DC bus link current feedback (single shunt configuration) or U phase shunt resistor (leg shunt configuration). It is sampled every PWM cycle.		

3.2.7.2 Iv

Index	123		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -2047	Max:2047	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides reconstructed motor phase V current (offset eliminated and ADC compensated). This current is calculated from the DC bus link current feedback (single shunt configuration) or V phase shunt resistor (leg shunt configuration). It is sampled every PWM cycle.		

3.2.7.3 IW

Index	124		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -2047	Max:2047	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides reconstructed motor phase W current (offset eliminated and ADC compensated). This current is calculated from Iu and Iv by equation$Iw = -(Iu + Iv)$. Its value is updated on every PWM cycle.		

3.2.7.4 I_Alpha

Index	126		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -2047	Max:2047	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides reconstructed motor alpha component current. Its value is updated on every PWM cycle.		

3.2.7.5 I_Beta

Index	127		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -2047	Max:2047	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides reconstructed motor beta component current. Its value is updated on every PWM cycle.		

3.2.7.6 VdcRaw

Index	136		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the measured DC bus voltage value. This value is updated every PWM cycle.		

3.2.7.7 VdcFilt

Index	137		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the filtered DC bus voltage value.		

3.2.7.8 VTH

Index	144		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides NTC temperature input value.		

3.2.7.9 Ipeak

Index	187		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: -2047	Max: 2047	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable keeps track of the peak amplitude of the motor phase current when MCE is in peak current tracking mode with single-shunt configuration by setting TminPhaseShift = 0. Its value is updated every PWM cycle.		

3.2.7.10 CurrentAmpOffset0

Index	188		
Size	Unsigned 16 bit		
Variable Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the offset value for phase U current sensing input pin. Its value is updated every PWM cycle.		

3.2.7.11 CurrentAmpOffset1

Index	189		
Size	Signed 16 bit		
Variable Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the offset value for phase V current sensing input pin. Its value is updated every PWM cycle.		

3.2.8 Protection Register Group

3.2.8.1 FaultEnable

Index	12		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:0xFFFF	Default: 0

Scaling or Notation: Bit field definitions are mentioned in description. For each bit, 0 – Ignore the associated fault; 1 – enable processing of the associated fault.

Description: This parameter specifies enable/disable of faults are mentioned below

[1:0]	Reserved
[2]	Enable DC bus overvoltage fault
[3]	Enable DC bus under voltage fault
[4]	Enable Flux PLL out of control fault
[5]	Reserved
[6]	Enable Over temperature fault
[7]	Enable rotor lock fault
[8]	Enable Phase loss fault
[9]	Reserved
[10]	Execution fault
[12:11]	Reserved
[13]	Enable UART link break fault
[14]	Enable Hall timeout fault
[15]	Enable Hall invalid fault

When a fault is disabled (bit set to “0”), the fault condition is ignored and the motor keeps running. However, even when a fault is disabled, its occurrence is reported in the FaultFlags variable, until the condition that caused the fault disappears. The exception is for UART link break fault, the FaultFlags bit[13] will not be set to 1 even when a UART link break fault occurs unless the UART link break fault is enabled by setting the FaultEnable bit[13].

Note: Phase loss fault is only detected in “PARKING” state.

3.2.8.2 VdcOvLevel

Index	13		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0

Scaling or Notation: In ADC counts.

Description: This parameter defines the dc bus over voltage trip level. A dc bus over voltage fault will be generated if dc bus voltage exceeds this threshold.

3.2.8.3 VdcUvLevel

Index	14		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts.		
Description:	This parameter defines the dc bus under voltage trip level. A dc bus under trip voltage fault will be generated if dc bus voltage falls below this threshold.		

3.2.8.4 CriticalOvLevel

Index	15		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts.		
Description:	Detection level for Critical Overvoltage. If this threshold is exceeded, all low side switches are clamped (zero-vector-braking) to protect the drive and to brake the motor. The Zero-vector is held until fault is cleared.		

3.2.8.5 RotorLockTime

Index	16		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default:1000
Scaling or Notation:	1 = 0.01 Second		
Description:	User can change the value of this parameter to customize the rotor lock detect time (default = 10 seconds). Please note if rotor lock detect time is configured too short, it may trigger the fault during acceleration or momentary high load condition.		

3.2.8.6 FluxFaultTime

Index	18		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 65535	Default:800
Scaling or Notation:	1 = 0.01 Second		
Description:	User can change the value of this parameter to customize the PLL out of synchronous detect time (default = 8 seconds).		

3.2.8.7 GateKillFilterTime

Index	19		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 4	Max:960	Default:96
Scaling or Notation:	$1 = 10.4167ns$		
Description:	Persistence filter time for PWM gate kill input (in clock cycles)		

3.2.8.8 CompRef

Index	20		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	ADC count (4095 = V_{adcref})		
Description:	This parameter value is derived from current trip level and current input offset value. $$CompRef = \frac{((iTripLevel \times CurrentScale) + Offset)}{Vadcref} * 4095$$ Example : iTripLevel = 2A, Current input scale 0.5V/A, Offset =0.55V, Vadcref =3.3V CompRef value is 1924 counts		

3.2.8.9 Tshutdown

Index	67		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts.		
Description:	This parameter defines the over-temperature shutdown threshold. If actual temperature input value is less than this threshold, trigger over temperature fault.		

3.2.8.10 PhaseLossLevel

Index	74		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: MCEWizard
Scaling or Notation:	In ADC counts.		
Description:	This parameter defines the phase current threshold value for phase loss protection function. Refer to Section 2.1.16.6 for details.		

3.2.8.11 SwFaults

Index	132		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max: 65535	Default: 0
Scaling or Notation:	See description.		
Description:	This variable is derived from FaultFlags by the following bitwise logical operation: $SwFaults = FaultFlags \cdot FaultEnable$ SwFaults is cleared by FaultClear. For bit field definition, refer to Faultflags.		

3.2.8.12 FaultClear

Index	134		
Size	Unsigned 16 bit		
Variable Type	Read Write		
Range	Min: 0	Max:1	Default: 0
Scaling or Notation:	See Description		
Description:	Writing 1 to this variable clears all faults. Once clear has been done, the variable will be cleared. If fault condition doesn't exist, fault clear will be successful and the drive will enter STOP state. If fault condition still exists, the drive will remain in fault state.		

3.2.8.13 FaultRetryPeriod

Index	81		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:0xFFF	Default:0x6403
Scaling or Notation:	Bit field definitions are mentioned in description		
Description:	This parameter enables to restart the motor after fault condition when control input signal is from frequency, duty or VSP [7:0] This field defines how many times restart MCE after fault condition. If value is 0, restart after fault is disabled. If value is 255, restart always after fault. [15:8] This field defines waiting time to restart MCE after fault condition. This value is represented in 0.1 seconds. If value is 100, MCE restarted 10S after fault.		

3.2.8.14 FaultClear

Index	134		
Size	Unsigned 16 bit		
Variable Type	Read Write		
Range	Min: 0	Max:1	Default: 0
Scaling or Notation:	See Description		
Description:	Writing 1 to this variable clears all faults. Once clear has been done, the variable will be cleared. If fault condition doesn't exist, fault clear will be successful and the drive will enter STOP state. If fault condition still exists, the drive will remain in fault state.		

3.2.8.15 FaultFlags

Index	135		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:0xFFFF	Default: 0

Scaling or Notation: Bit field definitions are mentioned in description

Description: This variable provides drive fault status. Most faults are handled by a fault handling routine operating at the PWM inverter switching frequency with the exception of Gate Kill faults. Gate Kill is handled within the Faults module and will instantly initiate inverter and regulator shutdown. The FaultFlags variable indicates currently pending fault conditions. The FaultClear variable is used to reset fault conditions.

For all bit fields defined below, a value of 1 indicates that the corresponding fault condition has occurred. The exception is for UART link break fault, the FaultFlags bit[13] will not be set to 1 even when a UART link break fault occurs unless the UART link break fault is enabled by setting the FaultEnable bit[13].

[0] Motor gatekill fault
[1] DC bus Critical overvoltage fault
[2] DC bus overvoltage fault
[3] DC bus under voltage fault
[4] Flux PLL out of control fault
[5] Reserved
[6] Over Temperature fault
[7] Rotor lock fault
[8] Phase Loss fault
[9] Reserved
[10] Execution fault (CPU load is more than 95%, or, background tasks are not executed at least once every 60s)
[11] Reserved
[12] Parameter load fault
[13] UART link break fault
[14] Hall timeout fault
[15] Hall invalid fault

Note: DC bus critical overvoltage and Gatekill fault cannot be masked by FaultEnable.

3.2.9 Start Control Register Group

3.2.9.1 BtsChargeTime

Index	21		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: 150
Scaling or Notation:	1 = one PWM period		
Description:	User can change the value of this parameter to configure the boot strap capacitor charging time (default = 150 PWM period).		

3.2.9.2 ParkAngle

Index	25		
Size	Signed 16 bit		
Parameter Type	DYNAMIC		
Range	Min: -32768	Max:32767	Default:5461
Scaling or Notation:	90° = 16384		
Description:	This parameter configures the current angle during parking state. Reset value of parking angle is set to 5461 (30°) which is the center of sector 0.		

3.2.9.3 ParkTime

Index	24		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default:1000
Scaling or Notation:	1 = 0.001 Second		
Description:	This parameter configures total parking time. During parking state, parking current increases linearly from 0 to low speed current limit. If ParkTime=0, parking state will be skipped.		

3.2.9.4 OpenLoopRamp

Index	26		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default:1000
Scaling or Notation:	See description.		
Description:	This parameter configures the open loop acceleration rate. During open loop state, motor current is regulated at low speed current limit; rotation speed accelerates linearly from 0 to MinSpd. Total duration of open loop: $T_{OpenLoop} = \frac{MinSpd*1024*10}{OpenLoopRamp}$, in 1 millisecond If OpenLoopRamp=0, open loop state will be skipped		

3.2.9.5 IS_Pulses

Index	27		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:1000	Default:14
Scaling or Notation:	1 = 1 sensing pulse under nominal DC bus voltage (DcBusVolts=2048)		
Description:	This parameter specifies the number of PWM cycles for each angle sensing pulse under nominal DC bus voltage. Actual number of PWM cycles is DC bus compensated in order to keep constant volt-second which in turns keep the same peak sensing current. Initial angle sensing function measures the current of last 2 PWM cycles, if the parameter is configured at 1, only one PWM cycle will actually carry current. So it is advised to configure this parameter ≥ 2. Write 0 to this parameter disables the initial angle sensing function.		

3.2.9.6 IS_Duty

Index	28		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:8191	Default:4095
Scaling or Notation:	8192 = 100% PWM duty cycle		
Description:	This parameter specifies the PWM duty cycle during ANGLE_SENSING state. For better current sensing quality, in single shunt current sensing, duty cycle of angle sensing should not be too low otherwise active vector will be too short to sense the current. In leg shunt current sensing, duty cycle should not be too high otherwise there will not be enough time to sense the current during zero vector.		

3.2.9.7 IS_IqInit

Index	29		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:8191	Default:14
Scaling or Notation:	4096 = 100% motor rated RMS current		
Description:	This parameter specifies the initial torque that would be applied after having completed ANGLE_SENSING state and before entering MOTOR_RUN state. Right after having completed ANGLE_SENSING state, the flux PLL has not locked onto the rotor angle. It takes some time to stabilize itself and requires the motor speed to be sufficiently high. This means at the beginning of MOTOR_RUN state, flux PLL is not working properly and it relies on initial torque to accelerate the motor in order for the PLL to lock. To achieve reliable and smooth start, some tuning to flux estimator and flux PLL is required.		

3.2.10 Catch Spin Register Group

3.2.10.1 TCatchSpin

Index	22		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default:1000
Scaling or Notation:	1 = 0.001 Second		
Description:	This parameter specifies the catch spin synchronization time duration before engaging motor start acceleration. During this period, the internal controller tries to sync rotor position with zero torque current reference.		

3.2.10.1 DirectStartThr

Index	23		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default:1000
Scaling or Notation:	16384 = Motor Max RPM		
Description:	At the end of catch spin state, this parameter is the absolute motor speed threshold that decides whether to go through Angle_Sensing + Parking + OpenLoop start or directly go to closed loop run state. If DirectStartThr=0, after catch spin, it will directly go to closed loop run state.		

3.2.11 Control Input Register Group

3.2.11.1 PGDeltaAngle

Index	53		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 65535	Default: 0
Scaling or Notation:	512 = 1 PG pulse every 360 electrical degree (every electrical cycle)		
Description:	This parameter configures the PG output. $PGDeltaAngle = 256 * \frac{Motor\ poles}{PPR}$ Write 0 to PGDeltaAngle will disable the PG output. PPR is expected PG Pulses Per Revolution. For example, 4 PPR for an 8 poles motor (1 pulse per electrical cycle), then: $PGDeltaAngle = 256 * \frac{8}{4} = 512$ 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.		

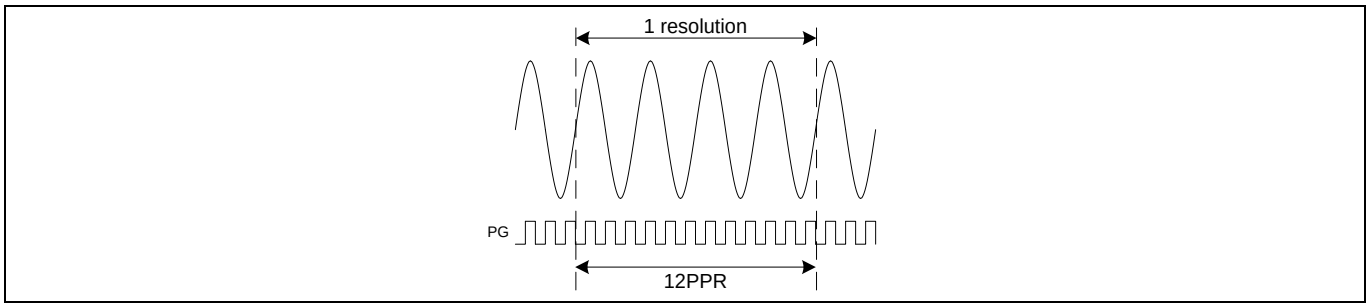


Figure 81 PG Output

Note: When PG output is disabled, PGOUT pin becomes available to be used as GPIO1 by script.

3.2.11.2 CmdStart

Index	69		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 65535	Default: 0
Scaling or Notation:	See description.		
Description:	In Vsp/Frequency/Duty Cycle input mode, this parameter specifies the input threshold for motor start.		

3.2.11.3 CmdStop

Index	68		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 65535	Default: 0
Scaling or Notation:	See description.		
Description:	In Vsp/Frequency/Duty Cycle input mode, this parameter specifies the input threshold for motor stop.		

3.2.11.4 CmdGain

Index	70		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 65535	Default: 0
Scaling or Notation:	See description.		
Description:	In Vsp/Frequency/Duty Cycle input mode, this parameter specifies the slope between the input threshold for motor start and threshold for MaxRPM.		

3.2.11.5 ControlFreq

Index	164		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max: 50000	Default: 0
Scaling or Notation:	1 count = 0.1Hz		
Description:	When control input measurement function is enabled, this variable represents the measured frequency of the input signal which is updated every 10 ms.		

3.2.11.6 ControlDuty

Index	165		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max: 1000	Default: 0
Scaling or Notation:	1 count = 0.1%		
Description:	When control input measurement function is enabled, this variable represents the measured duty cycle of the input signal which is updated every 10 ms.		

3.2.12 Voltage Control Register Group

3.2.12.1 Vd_Ext

Index	130		
Size	Unsigned 16 bit		
Variable Type	Read Write		
Range	Min: 0	Max:4974	Default: 0
Scaling or Notation:	$\text{Output duty cycle} = \frac{\text{Vd_Ext} \times 2048}{4974 \times \text{DcBusVoltsFilt}} \times 100\%$		
Description:	Vd command when the drive is working in voltage control mode.		

3.2.12.2 Vq_Ext

Index	131		
Size	Unsigned 16 bit		
Variable Type	Read Write		
Range	Min: 0	Max:4974	Default: 0
Scaling or Notation:	$Output\ duty\ cycle = \frac{Vq_Ext \times 2048}{4974 \times DcBusVoltsFilt} \times 100\%$		
Description:	Vq command when the drive is working in voltage control mode.		

3.2.12.3 V_Alpha

Index	151		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -8191	Max:8191	Default: 0
Scaling or Notation:	8191 = 100%		
Description:	This variable provides Alpha motor phase voltage.		

3.2.12.4 V_Beta

Index	152		
Size	Signed 16 bit		
Variable Type	Read only		
Range	Min: -8191	Max:8191	Default: 0
Scaling or Notation:	8191 = 100%		
Description:	This variable provides Beta motor phase voltage.		

3.2.12.5 Vd

Index	156		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4974	Default: 0
Scaling or Notation:	4974 = 100%		
Description:	Motor Vd voltage component. This variable holds the value of Id PI output value in case of speed control or current control mode.		

3.2.12.6 Vq

Index	157		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4974	Default: 0
Scaling or Notation:	4974 = 100%		
Description:	Motor Vq voltage component. This variable holds the value of Iq PI output value in case of speed control or current control mode.		

3.2.12.7 MotorVoltage

Index	158		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max: 9948	Default: 0
Scaling or Notation:	$4974 \times \frac{VdcFilt}{2048} = 100\% \times \frac{Max_Vdc_{sense}}{3}$		
Description:	This variable holds motor applied voltage. $Vdq = (\sqrt{Vd^2 + Vq^2})$		

3.2.13 Torque Compensation Register Group

3.2.13.1 TrqCompGain

Index	75		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default:N/A
Scaling or Notation:	U8.8		
Description:	This parameter specifies the gain factor that is used to adjust the amplitude of the sinusoidal compensation torque reference.		

3.2.13.2 TrqCompAngOfst

Index	76		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 65535	Default: Set by MCEWizard
Scaling or Notation:	90° = 16384		
Description:	This parameter specifies the angle offset value that the MCE uses in synthesizing a sinusoidal compensation torque reference. Refer to Section 2.1.15 for details.		

3.2.13.3 TrqCompLim

Index	77		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max: 16383	Default: 2048
Scaling or Notation:	4095 = 100% motor rated current		
Description:	This parameter specifies the maximum allowable value for internal variable ‘TrqRefFilt’ that is the low-pass filtered result of the variable ‘TrqRef’. Refer to Section 2.1.15 for details.		

3.2.13.4 TrqCompOnSpeed

Index	78		
Size	Unsigned 16 bit		
Variable Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: 0
Scaling or Notation:	16383 = Motor Max RPM		
Description:	This parameter set torque compensation ON speed threshold value. Torque compensation is active if motor speed reference value is less than this parameter value.		

3.2.13.1 TrqCompOffSpeed

Index	79		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:32767	Default: 0
Scaling or Notation:	16383 = Motor Max RPM		
Description:	This parameter set torque compensation OFF speed threshold value. Torque compensation is inactive if motor speed reference value is greater than this parameter value.		

3.2.13.2 TrqRef_Ext

Index	159		
Size	Signed 16 bit		
Parameter Type	Read only		
Range	Min: -16383	Max:16383	Default: N/A
Scaling or Notation:	4095 = 100% motor rated current		
Description:	This variable represents the synthesized sinusoidal compensation torque reference.		

3.2.13.3 TrqRef_Total

Index	190		
Size	Signed 16 bit		
Parameter Type	Read only		
Range	Min: -16383	Max:16383	Default: N/A
Scaling or Notation:	4095 = 100% motor rated current		
Description:	This variable represents the summed up torque reference that includes ‘TrqRef’ from speed PI output as well as ‘TrqRef_Ext’ from the torque compensation function.		

3.2.13.4 TrqCompBaseAngle

Index	191		
Size	Signed 16 bit		
Parameter Type	Read only		
Range	Min: -32768	Max: 32767	Default: Set by MCEWizard
Scaling or Notation:	90° = 16384		
Description:	This variable represents the base angle that is used in synthesizing a sinusoidal compensation torque reference.		

3.2.13.5 TrqCompStatus

Index	194		
Size	Unsigned 16 bit		
Parameter Type	Read only		
Range	Min: 0	Max: 65535	Default: NA
Scaling or Notation:	Bit field definitions are mentioned in description		
Description:	Torque compensation function status variable. [0] Torque compensation valid speed range status 0: SpdRef is out of the valid speed range 1: SpdRef is within the valid speed range [1] Mechanical cycle synchronization status 0: Not synchronized to mechanical cycle 1: Synchronized to mechanical cycle		

3.3 PFC Control Register (App ID =3)

Complete list of parameter and variables are listed in the Table 30 and Table 31 and find description in the following chapters.

Table 30 PFC Parameter list

App ID	Index	Parameter Name	Type	Description
3	1	PFC_HwConfig	STATIC	Application hardware configuration parameter
3	2	PFC_SysConfig	STATIC	System configuration parameter
3	3	PFC_PwmFreq	STATIC	PFC PWM frequency
3	4	PFC_TMinOff	DYNAMIC	Minimum PWM off time
3	5	PFC_Deadtime	STATIC	PWM dead time
3	6	PFC_SHDelay	DYNAMIC	Delay time from PWM output to ADC sample time
3	7	PFC_IRectLim	DYNAMIC	Rectifying current limit value
3	8	PFC_IGenLim	DYNAMIC	Generating current limit value
3	9	PFC_VdcRampRate	DYNAMIC	Voltage reference ramp up/down rate
3	10	PFC_KpVreg	DYNAMIC	Proportional gain of the voltage regulator

App ID	Index	Parameter Name	Type	Description
3	11	PFC_KxVreg	DYNAMIC	Integral gain of the voltage regulator
3	12	PFC_KpIreg	DYNAMIC	Proportional gain of the current regulator
3	13	PFC_KxIreg	DYNAMIC	Integral gain of the current regulator
3	15	PFC_TrackingCycle	DYNAMIC	Used for voltage reference in tracking mode
3	16	PFC_TrackingGain	DYNAMIC	Used for voltage reference in tracking mode
3	17	PFC_HalfCycleMin	STATIC	AC voltage minimum limit for input frequency check
3	18	PFC_HalfCycleMax	STATIC	AC voltage maximum limit for input frequency check
3	19	PFC_VacZCThr	DYNAMIC	AC voltage threshold to detect zero crossing
3	20	PFC_VacOvLevel	DYNAMIC	AC voltage input overvoltage trip level
3	21	PFC_VacUvLevel	DYNAMIC	AC voltage input under voltage trip level
3	22	PFC_VdcOvLevel	DYNAMIC	DC bus overvoltage trip level
3	23	PFC_VdcUvLevel	DYNAMIC	DC bus under voltage trip level
3	24	PFC_AcDcScale	DYNAMIC	Ratio of feedforward component added to the duty output
3	25	PFC_LFactor	DYNAMIC	Used for average current calculation
3	26	PFC_FaultEnable	DYNAMIC	Enable or disable fault condition handling. When a fault bit is not set, the fault condition is ignored
3	27	PFC_GateKillTime	STATIC	Persistence filter time for PWM gate kill input
3	28	PFC_TargetDCVolt	STATIC	Target DC bus voltage for fixed-voltage mode

Table 31 PFC Variable list

App ID	Index	Variable Name	Type	Description
3	81	PFC_SequencerState	READONLY	Current state
3	82	PFC_Command	READWRITE	Controls the system state - Stop/ start the PFC
3	85	PFC_FaultClear	READWRITE	Fault clear
3	87	PFC_SwFaults	READONLY	Fault status based on fault condition and fault mask
3	89	PFC_TargetVolt	READWRITE	Voltage set point value
3	90	PFC_VoltagePIoutput	READONLY	Voltage PI controller output value
3	92	PFC_VdcRaw	READONLY	DC bus voltage
3	93	PFC_IpfcRaw	READONLY	PFC Current
3	94	PFC_AbsVacRaw	READONLY	AC voltage absolute value
3	98	PFC_VacRMS	READONLY	AC voltage RMS value
3	99	PFC_VdcFilt	READONLY	DC bus filtered voltage
3	103	PFC_VacRaw	READONLY	AC voltage value
3	104	PFC_Fault Flag	READONLY	Fault status based on fault condition
3	105	PFC_IpfcAvg	READONLY	PFC current average value
3	106	PFC_IpfcRMS	READONLY	PFC current RMS value
3	107	PFC_ACPower	READONLY	PFC input power
3	110	PFC_CurrentPIoutput	READONLY	Current control PI output
3	113	VACPk	READONLY	AC voltage peak value during half line cycle

3.3.1 Control Register Group

3.3.1.1 PFC_HwConfig

Index	1		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 0xFFFF	Default: 0

Scaling or Notation: Bit field definitions are mentioned in description

Description: PFC application hardware configuration parameter

[0] PFC Topology
- 0- Boost Mode PFC
- 1- Totem Pole PFC

[4:1] Reserved

[5] Active polarity for Low side PWM output
- 0- Active level is high
- 1- Active level is low

[6] Active polarity for High side PWM output
- 0- Active level is high
- 1- Active level is low

[8:7] Internal gain for current measurement
- 0- Internal gain is 1
- 1- Internal gain is 3
- 2- Internal gain is 6
- 3- Internal gain is 12

[9] Current sensing polarity
- 0- Non-Inverting
- 1- Inverting

[10] AC Voltage Sensing
- 0- Single ended sensing (external op-amp required)
- 1- Differential sensing (no op-amp, use two ADC channels)

[15:11] Reserved

3.3.1.2 PFC_SysConfig

Index	2		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 0xFFFF	Default: 0
Scaling or Notation:	Bit field definitions are mentioned in description		
Description:	PFC system configuration parameter [3:0] Execution rate for current control loop. 1- Current control loop executed every PWM period 2- Current control loop executed every 2 PWM period … 15 - Current control loop executed every 15 PWM period [4] Control mode selection 0- Voltage Control Mode 1- Tracking Control Mode [5] Over-current protection (OCP) options for boost mode PFC 0: Use latch-off OCP option 1: Use cycle-by-cycle OCP option [7:6] Configure PFC OCP comparator hysteresis 0: 0 mV 1: 10 mV 2: 15 mV 3: 20 mV [15:8] Reserved		

3.3.1.3 PFC_ SequencerState

Index	81		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max: 5	Default: 0
Scaling or Notation:	See description		
Description:	This variable contains the current sequence state of the drive 0- Power on state 1- Stop state 2- Measuring offset current 4- PFC running 5- Fault state		

3.3.1.4 PFC_Command

Index	82		
Size	Unsigned 16 bit		
Variable Type	Read Write		
Range	Min: 0	Max: 1	Default: 0

Scaling or Notation: See description

Description: This variable controls the system state with the following values:

0- Stop the PFC

1- Start the PFC

3.3.2 PWM Register Group

3.3.2.1 PFC_PwmFreq

Index	3		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:1000	Default:6000
Scaling or Notation:	1 = 0.1 kHz F_{PWM} ; 1600 = 16kHz F_{PWM}		
Description:	This parameter configures the PWM frequency in 0.1 kHz increment.		

3.3.2.2 PFC_TMinOff

Index	4		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:PWMPeriod	Default: 0
Scaling or Notation:	$1 = 10.4167ns$		
Description	This parameter configures minimum PWM off time. Where : PWMPeriod is 96,000,000/(2*PWMFreq[Hz])		

3.3.2.3 PFC_Deadtime

Index	5		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max: 255	Default: 0
Scaling or Notation:	$1 = 10.4167ns$		
Description	This parameter configures PWM dead time value. This parameter is reserved for future use and should be always written 0.		

3.3.2.4 PFC_SHDelay

Index	6		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:960	Default: 0
Scaling or Notation:	$1 = 10.4167ns$		
Description:	SHDelay specifies the time delay from PWM output to ADC sample time for current sensing. The delay time is depending on the hardware design; usually it should consider propagation delay of gate driver circuit and turn on (turn off) delay of switching devices.		

3.3.3 Voltage Control Register Group

3.3.3.1 PFC_IRectLim

Index	7		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	4095 = 100% maximum measureable current		
Description:	This parameter specifies the maximum voltage PI positive output value which means allowable PFC rectifying current. Rectifying current is the energy direction from AC to DC. This limit should be set higher than maximum possible PFC current considering load condition, lowest AC voltage as well as some margin. The goal is never entering current limit unless there is hardware issue.		

3.3.3.2 PFC_IGenLim

Index	8		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	4095 = 100% maximum measureable current		
Description:	This parameter specifies the maximum voltage PI negative output value which means allowable PFC generating current. Generating current is the energy direction from DC to AC. This parameter is reserved for future PFC release and not actually working in current PFC release. Set this parameter to 0, or set to a small value (<100).		

3.3.3.3 PFC_VdcRampRate

Index	9		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	See description		
Description:	This parameter specifies the ramp rate of voltage reference. $VdcRampRate = \frac{PFC_VdcRampRate * 0.001 * VadcRef}{4095 * VdcVoltageDividerRatio * 2^{16}}$, $in\ V/s$ Where: $VdcVoltageDividerRatio = \frac{Vdc\ Sensing\ Low\ Resister}{Vdc\ Sensing\ Low\ Resister + Vdc\ Sensing\ High\ Resister}$		

3.3.3.4 PFC_KpVreg

Index	10		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	U4.12		
Description:	This parameter specifies the proportional gain of the voltage regulator		

3.3.3.5 PFC_KxVreg

Index	11		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	U4.12		
Description:	This parameter specifies the integral gain of the voltage regulator		

3.3.3.6 PFC_TargetVoltInit

Index	28		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	4095 = Maximum measureable voltage		
Description:	This parameter defines the initial DC bus target voltage for fixed voltage mode. This parameter is copied to PFC_TargetVolt variable during startup.		

3.3.3.7 PFC_TargetVolt

Index	89		
Size	Unsigned 16 bit		
Variable Type	Read Write		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	4095 = Maximum measureable voltage		
Description:	This is the reference input of voltage regulator. In tracking mode, this variable has no influence.		

$$TargetVolt = \frac{PFC_TargetVolt * VacRef * (Vdc\ Sensing\ High\ Resister + Vdc\ Sensing\ Low\ Resister)}{Vdc\ Sensing\ Low\ Resister * 4095}, V$$

3.3.3.8 PFC_VoltagePIoutput

Index	90		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	4095 = 100% maximum measureable current		
Description:	Voltage regulator output value		

3.3.4 Current Control Register Group

3.3.4.1 PFC_KpIreg

Index	12		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	U4.12		
Description:	This parameter specifies the proportional gain of the current regulator. Higher proportional gain improves PFC current waveform, but it may crease current oscillation if its value is too high, and/or AC voltage and PFC current sensing in PFC hardware has high noise.		

3.3.4.2 PFC_KxIreg

Index	13		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	U4.12		
Description:	This parameter specifies the integral gain of the current regulator		

3.3.4.3 PFC_AcDcScale

Index	24		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	See description		
Description:	This parameter defines the ratio of feedforward component added to the duty output and scaling between AC and DC voltage measurement. $PFC_{AcDCScale} = \text{AcDcScaleAdjustent} * \frac{(\text{Vac Sensing High Resister} + \text{Vac Sensing Low Resister}) * \text{Vdc Sensing Low Resister} * 2048}{\text{Vac Sensing Low Resister} * (\text{Vdc Sensing High Resister} + \text{Vdc Sensing Low Resister})}$ If high resistor and low resistor are the same value for AC voltage sensing and DC voltage sensing, PFC_AcDcScale=2048 represents 100% feedforward ratio. The ratio should be adjusted accordingly to achieve best PFC current waveform.		

3.3.4.4 PFC_LFactor

Index	25		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	See description		
Description:	This parameter is used to calculate the average current as current measurement is done at every peak current period. Average current calculation helps improve the PFC current waveform. Although the MCEWizard create the value for this parameter, due to many factors which affect the actual result, it may still need to be fine-tuned to achieve best PFC current waveform.		

3.3.4.5 PFC_CurrentPIoutput

Index	110		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max:PWM Period	Default: 0
Scaling or Notation:	PWMPeriod= 100% duty cycle		
Description:	Sum of output from current regulator and feed forward output values Where : PWMPeriod is 96,000,000/(2*PWMFreq[Hz])		

3.3.5 Protection Register Group

3.3.5.1 PFC_GateKillTime

Index	19		
Size	Unsigned 16 bit		
Parameter Type	STATIC		
Range	Min: 0	Max:960	Default:48
Scaling or Notation:	$1 = 10.4167ns$		
Description:	Persistence filter time for PWM gate kill input (in clock cycles)		

3.3.5.2 PFC_VacOvLevel

Index	20		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts.		
Description:	This parameter defines the AC over voltage trip level. AC over voltage fault will be generated if AC input voltage exceeds this threshold.		

3.3.5.3 PFC_VacLvLevel

Index	21		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts.		
Description:	This parameter defines the AC under voltage trip level. AC bus under trip voltage fault will be generated if AC input voltage falls below this threshold.		

3.3.5.4 PFC_VdcOvLevel

Index	22		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts.		
Description:	This parameter defines the dc bus over voltage trip level. A dc bus over voltage fault will be generated if dc bus voltage exceeds this threshold.		

3.3.5.5 PFC_VdcUvLevel

Index	23		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts.		
Description:	This parameter defines the dc bus under voltage trip level. A dc bus under trip voltage fault will be generated if dc bus voltage falls below this threshold.		

3.3.5.6 PFC_FaultEnable

Index	26		
Size	Unsigned 16 bit		
Parameter Type	DYNAMIC		
Range	Min: 0	Max:0xFFFF	Default: 0
Scaling or Notation:	Bit field definitions are mentioned in description. For each bit, 0 – Ignore the associated fault; 1 – enable processing of the associated fault.		
Description:	This parameter specifies enable/disable of faults are mentioned below [0] Reserved, must be set to “0” [1] Enable DC bus under voltage fault [2] Enable DC bus over voltage fault [3] Reserved, must be set to “0” [4] Enable AC under voltage fault [5] Enable AC over voltage fault [15:6] Reserved, must be set to “0” When a fault is disabled (bit set to “0”), the fault condition is ignored and the motor keeps running. However, even when a fault is disabled, its occurrence is reported in the FaultFlags variable, until the condition that caused the fault disappears.		

3.3.5.7 PFC_FaultClear

Index	85		
Size	Unsigned 16 bit		
Variable Type	Read Write		
Range	Min: 0	Max:1	Default: 0
Scaling or Notation:	See Description		
Description:	Writing 1 to this variable clears all faults. Once clear has been done, the variable will be cleared. If fault condition doesn’t exist, fault clear will be successful and the drive will enter STOP state. If fault condition still exists, the drive will remain in fault state.		

3.3.5.8 PFC_SwFaults

Index	87		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:255	Default: 0
Scaling or Notation:	See description.		
Description:	This variable is derived from FaultFlags by the following bitwise logical operation: $PFC_SwFaults = PFC_FaultFlags \cdot PFC_FaultEnable$ SwFaults is cleared by PFC_FaultClear. For bit field definition, refer to PFC_Faultflags.		

3.3.5.9 PFC_FaultFlags

Index	104		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:0xFFFF	Default: 0
Scaling or Notation:	Bit field definitions are mentioned in description		
Description:	This variable provides drive fault status. Most faults are handled by a fault handling routine operating at the PWM inverter switching frequency with the exception of Gate Kill faults. Gate Kill is handled within the Faults module and will instantly initiate inverter and regulator shutdown. The FaultFlags variable indicates currently pending fault conditions. The FaultClear variable is used to reset fault conditions. For all bit fields defined below, a value of 1 indicates that the corresponding fault condition has occurred. [0] PFC gate kill fault [1] DC bus under voltage fault [2] DC bus over voltage fault [3] Vac frequency fault [4] Vac under voltage fault [5] Vac overvoltage fault [11:6] Reserved [12] Parameter load fault [15:13] Reserved Gate kill fault and Vac Frequency fault cannot be masked by PFC_FaultEnable		

3.3.6 Measurement Register Group

3.3.6.1 PFC_VdcRaw

Index	92		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the measured DC bus voltage value. This value is updated every PWM cycle.		

3.3.6.2 PFC_VdcFilt

Index	99		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the filtered DC bus voltage value.		

3.3.6.3 PFC_IpfcRaw

Index	93		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the Ipfc current raw value.		

3.3.6.4 PFC_IpfcAvg

Index	105		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the Ipfc average current value.		

3.3.6.5 PFC_IpfcRMS

Index	106		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the Ipfc current RMS value.		

3.3.6.6 PFC_VacRaw

Index	103		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the Vac raw voltage value.		

3.3.6.7 PFC_AbsVacRaw

Index	94		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the Vac absolute voltage value.		

3.3.6.8 PFC_VacRMS

Index	98		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the Vac RMS voltage value.		

3.3.6.9 PFC_ACPower

Index	107		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:65535	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the AC input power value.		

3.3.6.10 VACPk

Index	113		
Size	Unsigned 16 bit		
Variable Type	Read only		
Range	Min: 0	Max:4095	Default: 0
Scaling or Notation:	In ADC counts		
Description:	This variable provides the peak of the Vac absolute value for each half line cycle.		

3.4 Script Register (App ID = 4)

Table 32 is a complete list of pre-defined variables for the script application.

Table 32 Script Variable list

App ID	Index	Variable Name	Type	Description
4	0	Script_UserVersion	Read Only	Holds script user version configured in script input file
4	1	Script_Command	Read Write	Controls the script state – Stop or start
4	98	ADC_Result0	READONLY	Holds AIN0 analog input value (12 bit value)
4	99	ADC_Result1	READONLY	Holds AIN1 analog input value (12 bit value)
4	100	ADC_Result2	READONLY	Holds AIN2 analog input value (12 bit value)
4	101	ADC_Result3	READONLY	Holds AIN3 analog input value (12 bit value)
4	102	ADC_Result4	READONLY	Holds AIN4 analog input value (12 bit value)
4	103	ADC_Result5	READONLY	Holds AIN5 analog input value (12 bit value)
4	104	ADC_Result6	READONLY	Holds AIN6 analog input value (12 bit value)
4	105	ADC_Result7	READONLY	Holds AIN7 analog input value (12 bit value)
4	106	ADC_Result8	READONLY	Holds AIN8 analog input value (12 bit value)
4	107	ADC_Result9	READONLY	Holds AIN9 analog input value (12 bit value)
4	108	ADC_Result10	READONLY	Holds AIN10 analog input value (12 bit value)
4	109	ADC_Result11	READONLY	Holds AIN11 analog input value (12 bit value)
4	110	GPIO_IN_L	READONLY	Holds digital input/output (GPIO0 to GPIO15) pins values.
4	111	GPIO_IN_H	READONLY	Holds digital input/output (GPIO16 to GPIO29) pins values.
4	112	GPIO_OUT_L	READWRITE	Set or reset digital output pin (GPIO0 to GPIO15)
4	113	GPIO_OUT_H	READWRITE	Set or reset digital output pin (GPIO16 to GPIO29)

Note: To access script related registers from MCEDesigner, users shall use App ID = 0, and relevant index numbers shall be added by 128. To access script related registers from script code, JCOM or user UART interfaces, users shall use App ID = 4. Besides, global variables defined in script code can also be accessed through JCOM or user UART interface. To access these global variables, users shall use App ID = 4 and

appropriate Index number. The Index number for a given global variable can be found from the script translator output file (ldf file).

3.4.1 Script_UserVersion

Index	0		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max:0xFFFF	Default: 0
Scaling or Notation:	Bit field defined		
Description:	Script user version scheme – 8:8 Bit Coding. This variable only can be read from MCEDesigner or User UART interface. [7:0] Minor version [15:8] Major version		

3.4.2 Script_Command

Index	1		
Size	Unsigned 16 bit		
Variable Type	Read Write		
Range	Min: 0	Max:3	Default: 3
Scaling or Notation:	See description		
Description:	This variable controls the system state with the following values: 0x00 : Stop Task0 and Task1 script function 0x01 : Start Task0 script function and stop Task1 script function 0x02 : Stop Task0 script function and start Task1 script function 0x03 : Start Task0 and Task1 script function 0xAE51: Reset controller from script Task0 or Task1 function		

3.4.3 ADC_Resultx [x: 0 to 11]

Index	98 – 109		
Size	Unsigned 16 bit		
Variable Type	Read Only		
Range	Min: 0	Max:0xFFF	Default: 0
Scaling or Notation:	In ADC counts, InputVoltage[v] = ADC_Result*VDD[v]/0xFFF		
Description:	These variable holds the configured user ADC pin value. These variables value are read by MCE every 1 mS		

3.4.4 GPIO_IN_L

Index	110		
Size	Unsigned 16 bit		
Parameter Type	Read Only		
Range	Min: 0	Max: 0xFFFF	Default: 0

Scaling or notation: Bitfield

Description: This variable holds the configured user GPIO value. This variable value are read by MCE every 1 mS

[0] Holds GPIO0 pin value, 0- input is low, 1- input is high
[1] Holds GPIO1 pin value, 0- input is low, 1- input is high
[2] Holds GPIO2 pin value, 0- input is low, 1- input is high
[3] Holds GPIO3 pin value, 0- input is low, 1- input is high
[4] Holds GPIO4 pin value, 0- input is low, 1- input is high
[5] Holds GPIO5 pin value, 0- input is low, 1- input is high
[6] Holds GPIO6 pin value, 0- input is low, 1- input is high
[7] Holds GPIO7 pin value, 0- input is low, 1- input is high
[8] Holds GPIO8 pin value, 0- input is low, 1- input is high
[9] Holds GPIO9 pin value, 0- input is low, 1- input is high
[10] Holds GPIO10 pin value, 0- input is low, 1- input is high
[11] Holds GPIO11 pin value, 0- input is low, 1- input is high
[12] Holds GPIO12 pin value, 0- input is low, 1- input is high
[13] Holds GPIO13 pin value, 0- input is low, 1- input is high
[14] Holds GPIO14 pin value, 0- input is low, 1- input is high
[15] Holds GPIO15 pin value, 0- input is low, 1- input is high

3.4.5 GPIO_IN_H

Index	111		
Size	Unsigned 16 bit		
Parameter Type	Read Only		
Range	Min: 0	Max: 0x3FFF	Default:0

Scaling or notation: Bitfield

Description: This variable holds the configured user GPIO value. This variable value are read by MCE every 1 mS

[0]	Holds GPIO16 pin value, 0- input is low, 1- input is high
[1]	Holds GPIO17 pin value, 0- input is low, 1- input is high
[2]	Holds GPIO18 pin value, 0- input is low, 1- input is high
[3]	Holds GPIO19 pin value, 0- input is low, 1- input is high
[4]	Holds GPIO20 pin value, 0- input is low, 1- input is high
[5]	Holds GPIO21 pin value, 0- input is low, 1- input is high
[6]	Holds GPIO22 pin value, 0- input is low, 1- input is high
[7]	Holds GPIO23 pin value, 0- input is low, 1- input is high
[8]	Holds GPIO24 pin value, 0- input is low, 1- input is high
[9]	Holds GPIO25 pin value, 0- input is low, 1- input is high
[10]	Holds GPIO26 pin value, 0- input is low, 1- input is high
[11]	Holds GPIO27 pin value, 0- input is low, 1- input is high
[12]	Holds GPIO28 pin value, 0- input is low, 1- input is high
[13]	Holds GPIO29 pin value, 0- input is low, 1- input is high
[15:14]	Reserved

3.4.6 GPIO_OUT_L

Index	112		
Size	Unsigned 16 bit		
Parameter Type	Read Write		
Range	Min: 0	Max :0xFFFF	Default: 0

Scaling or Notation: Bitfield

Description: This variable holds the configured user GPIO value. This variable value are read by MCE every 1 mS

[0]	Set or Reset GPIO0 pin, 0- Reset (low), 1- Set(High)
[1]	Set or Reset GPIO1 pin, 0- Reset (low), 1- Set(High)
[2]	Set or Reset GPIO2 pin, 0- Reset (low), 1- Set(High)
[3]	Set or Reset GPIO3 pin, 0- Reset (low), 1- Set(High)
[4]	Set or Reset GPIO4 pin, 0- Reset (low), 1- Set(High)
[5]	Set or Reset GPIO5 pin, 0- Reset (low), 1- Set(High)
[6]	Set or Reset GPIO6 pin, 0- Reset (low), 1- Set(High)
[7]	Set or Reset GPIO7 pin, 0- Reset (low), 1- Set(High)
[8]	Set or Reset GPIO8 pin, 0- Reset (low), 1- Set(High)
[9]	Set or Reset GPIO9 pin, 0- Reset (low), 1- Set(High)
[10]	Set or Reset GPIO10 pin, 0- Reset (low), 1- Set(High)
[11]	Set or Reset GPIO11 pin, 0- Reset (low), 1- Set(High)
[12]	Set or Reset GPIO12 pin, 0- Reset (low), 1- Set(High)
[13]	Set or Reset GPIO13 pin, 0- Reset (low), 1- Set(High)
[14]	Set or Reset GPIO14 pin, 0- Reset (low), 1- Set(High)
[15]	Set or Reset GPIO15 pin, 0- Reset (low), 1- Set(High)

3.4.7 GPIO_OUT_H

Index	113		
Size	Unsigned 16 bit		
Parameter Type	Read Write		
Range	Min: 0	Max: 0x3FFF	Default: 0

Scaling or Notation: Bit field

Description: This variable holds the configured user GPIO value. This variable value are read by MCE every 1 ms

[0]	Set or Reset GPIO16 pin, 0- Reset (low), 1- Set(High)
[1]	Set or Reset GPIO17 pin, 0- Reset (low), 1- Set(High)
[2]	Set or Reset GPIO18 pin, 0- Reset (low), 1- Set(High)
[3]	Set or Reset GPIO19 pin, 0- Reset (low), 1- Set(High)
[4]	Set or Reset GPIO20 pin, 0- Reset (low), 1- Set(High)
[5]	Set or Reset GPIO21 pin, 0- Reset (low), 1- Set(High)
[6]	Set or Reset GPIO22 pin, 0- Reset (low), 1- Set(High)
[7]	Set or Reset GPIO23 pin, 0- Reset (low), 1- Set(High)
[8]	Set or Reset GPIO24 pin, 0- Reset (low), 1- Set(High)
[9]	Set or Reset GPIO25 pin, 0- Reset (low), 1- Set(High)
[10]	Set or Reset GPIO26 pin, 0- Reset (low), 1- Set(High)
[11]	Set or Reset GPIO27 pin, 0- Reset (low), 1- Set(High)
[12]	Set or Reset GPIO28 pin, 0- Reset (low), 1- Set(High)
[13]	Set or Reset GPIO29 pin, 0- Reset (low), 1- Set(High)
[15:14]	Reserved

4 Motor Tuning

MCEWizard 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 MCEWizard 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:

4.1 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 MCEDesigner, output current display is usually slightly higher than measured motor current due to sensing noise. However, the values shown in MCEDesigner 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. TCntMin (single shunt only)
    4. SHDelay
    5. 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.

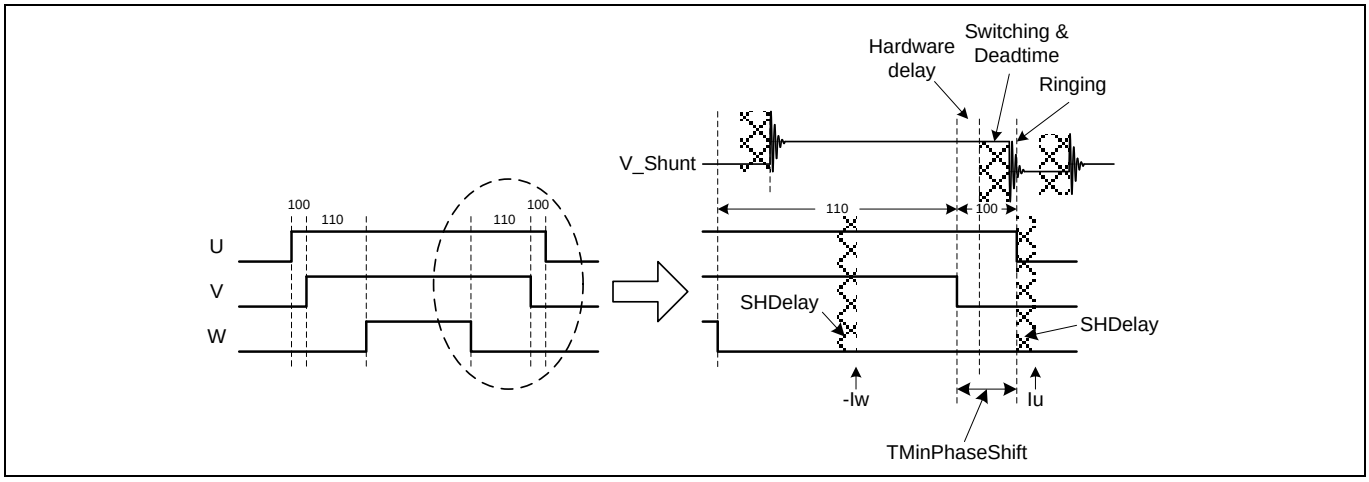


Figure 82 Single Shunt Current Sensing for Phase Shift PWM

There are three timings: Dead time, hardware delay time and ringing time. Dead time is already known since we set it in MCEWizard. 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.

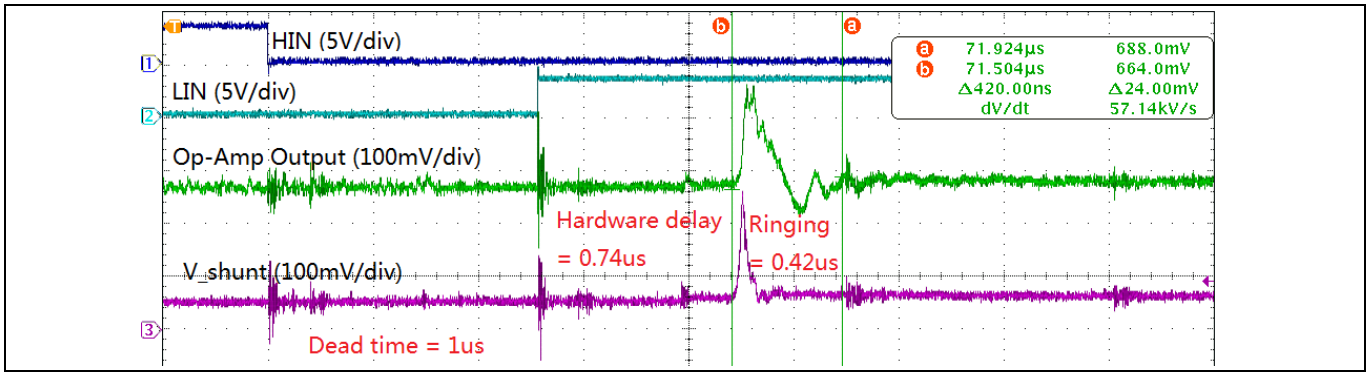


Figure 83 Measuring hardware delay and ringing time

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

$$SHDelay < 0.74us$$

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

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:

$$TMinPhaseShift = 1.6us$$

$$SHDelay = 0.6us$$

4.2 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 84. 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 84, 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 84, 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 86. The equivalent transfer function of Figure 86 is a first order lag with time constant τ_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.

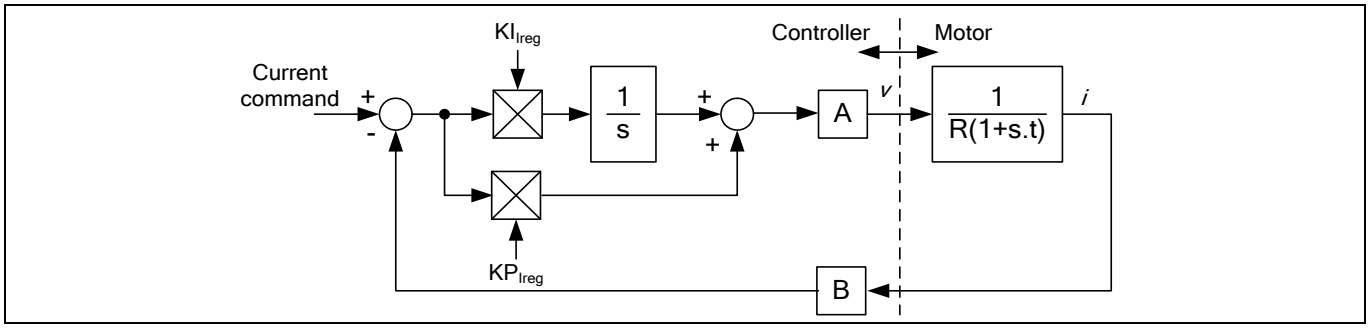


Figure 84 Current controller dynamics

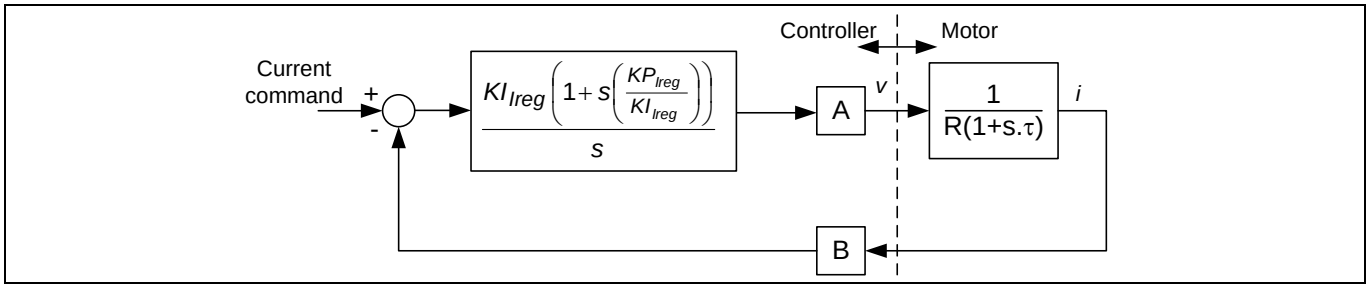


Figure 85 Pole zero cancellation

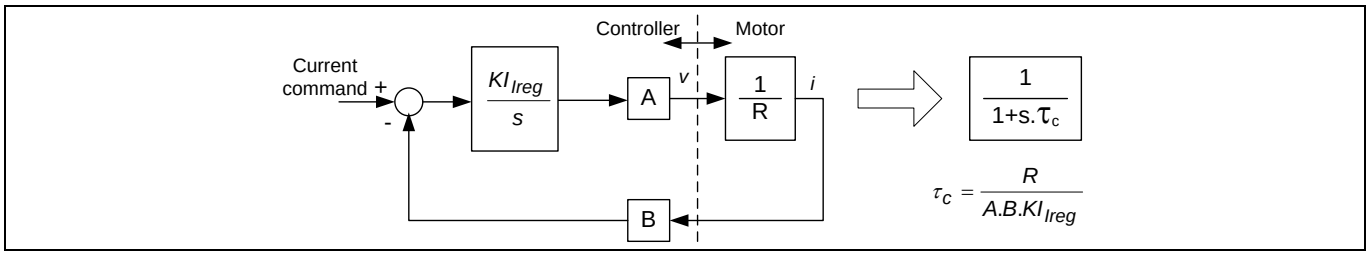


Figure 86 Simplified current control dynamics due to pole zero cancellation

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}, 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, MCEWizard 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 87 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.

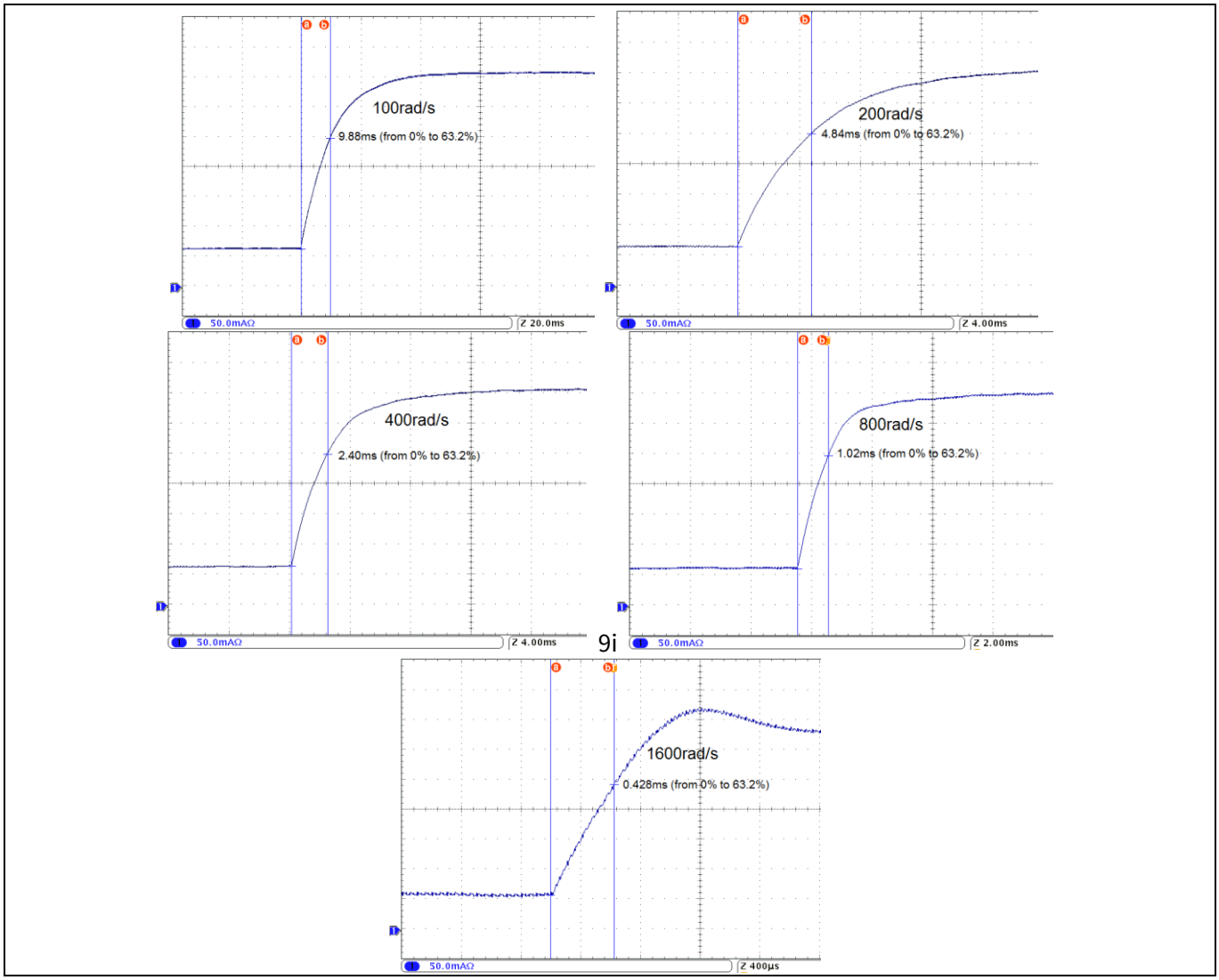


Figure 87 Current regulator step response (100/200/400/800/1600rad/s)

4.3 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

4.4 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

4.5 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.

4.6 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 minimum pulse scheme or phase shift PWM scheme (single shunt configuration). Noise caused by minimum pulse scheme can be reduced by reducing parameter value of TCntMin. 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.

5 Revision History

Document version	Date of release	Description of changes
1.35	2023-08-14	Updated to reflect MCE software revision 1.03.07 Updated Table 1 with description of Execution fault Added new Section 2.1.16.7 on Execution fault and background task monitoring Updated 3.1.11 'SysTaskConfig' bit mapping Updated 3.2.8.1 'FaultEnable' bit mapping Updated 3.2.8.15 'FaultFlags' description of Execution fault
1.34	2021-09-01	Section 2.6.11.3 (Configurable UART), Section 3.1.10 (InternalTemp), Section 2.1.14.3.1 revised
1.33	2021-06-09	Section 3.2.1.2 (SysConfig) revised.
1.32	2021-02-09	Section 3.1.12 (GPIO parameters) added. Section 2.1.16.5 revised. Section 2.6.11.3 revised.
1.31	2020-8-28	'MotorVoltage' variable description updated. Table 26 updated. Section 2.6.11.3 revised. Section 2.1.6 revised. Figure 39 revised. Parameter 'SysTaskConfig' description updated. Section 2.5.1 revised. Figure 28 and Figure 30 revised. VACPk variable (Section 3.3.6.10) added. 'PFC_SysConfig' parameter description updated. Figure 49 revised. 'PGDeltaAngle' parameter description updated. Section 2.1.15 revised. Section 2.3.7.6 revised. Figure 65 and Figure 66 revised. Section 4.2 revised.

Document version	Date of release	Description of changes
1.3	2020-4-26	'MotorStatus' 'AppConfig' parameter description updated. 'SysTaskConfig' parameter description added. Section 2.1.8.7 (Control Input Customization) added. Section 2.1.6.3 (Motor Over Current Protection) updated. Section 2.2.2.1 (PFC Over Current Protection) updated. Register group (Section 3.2.1.1) revised. Section 3.4 (Script Register) updated. Section 2.6.11.3 (Configurable UART driver methods) added. Section 2.6 (Script Engine) updated. Section 3.2.6 (Flux Estimator PLL Register Group) updated. New register group (section 3.1.3, section 3.2.8.9, section 3.2.8.10, section 3.2.8.11, section 3.2.5.12, section 3.2.5.13, section 3.2.5.14, section 3.2.5.15, section 3.2.5.16, section 3.2.5.17, and section 3.2.14) added. Section 2.1.6 (Current Measurement) revised to include current sensing timing description and section 2.1.4.2.4 (Peak Current Tracking with No Phase Shift Window). Section 'Duty Mode Control' removed. Section 'Duty Mode Control Register Group' removed. Section 2.1.13 (Hall Sensor Interface) revised to include Hall PLL and Atan angle description. Section 3.2.5.1 (HallAngleOffset) revised. Section 2.1 revised. Section 2.1.6 (DC Bus Compensation) added. Section 2.3.1 (Baud Rate) added. Section 2.1.8 (Motor Current Limit Profile) revised. Section 2.1.9 (Initial Angle Sensing) added. Section 2.1.10 (Over-Modulation) added. Section 2.1.11 (2-Phase Modulation) added. Section 2.1.16.1 revised. Section 2.1.16.2 revised. Section 2.1.16.4 revised. Section 2.1.16.5 revised. Section 2.1.16.6 revised. Section 2.1.15 (Torque Compensation) added. Section 2.1.3 (Current Sensing Offset Measurement) added. Section 2.1.1 (Variable Scaling) added. Variable scaling table added.
1.2	2019-06-05	Section 2.1.4.1.3 (Low Noise Phase Shift PWM) added. Section 2.1.9 (Duty mode control) added. Section 2.1.10 (Hall Sensor Interface) added. Section 2.4 (JCOM Inter-Chip Communication) added. New register groups (Section 3.2.4, Section 3.2.5, etc.) added.

1.1	2018-08-01	Script Engine function description added (Section 2.5 and Section 3.4) Motor control and PFC protection description is added (Section2.1.6 and Section 2.2.2) Following two parameters are added: FaultRetryPeriod and PFC_TargetDCVolt (Section 3.2 and Section 3.3) IdFwk variable type changed to ReadOnly PFC_HalfCycleMin and PFC_HalfCycleMax parameter type changed to STATIC
1.0	2018-02-09	Release version- Functional and register description on sensorless FOC and PFC

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

Edition 2021-09-08

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2023 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

ifx1

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics (“Beschaffenheitsgarantie”) .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer’s compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer’s products and any use of the product of Infineon Technologies in customer’s applications.

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

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

WARNINGS

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

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