Application Developer's Guide

iMOTION™ motor control IC with additional MCU

About this document

Scope and purpose

The IRMCx100 series motor control ICs are mixed signal devices optimized for permanent magnet motor control. They combines the iMOTION™ motion control engine (MCE) with an additional 8 Bit microcontroller (MCU) to improve application flexibility.

This Developer‘s Guide will begin with the process of initial testing with the target motor, continue with modification of the MCE design for application specific requirements and conclude with the design of motor control hardware for the final application. This guide assumes that the user is in possession of an iMOTION™ reference design kit and has already completed the activities in the Quick Start Guide. The user should also review the “MCEDesigner User‘s Guide”. This guide will refer to MCEDesigner features and actions frequently.

Section 2 starts by describing in detail how to measure the parameters of the target motor, generate the correct drive parameters, and begin spinning the motor. Next, this section gives instructions on how to tune the speed and current control loops and optimize the motor start-up parameters. Section 2 concludes with motor drive performance verification and testing methods using MCEDesigner.

Section 3 introduces the MCE processor in more detail and then gives instructions on how to modify the factory-supplied MCE design, if desired. The section finishes with some sample program modifications.

Section 4 guides the user through design, testing and optimization of application specific hardware as it relates to the IRMCx100 motor control IC.

Finally, Section 5 provides application guidance for the power factor correction (PFC) feature, which applies to IRMCx143 and IRMCx188 only and is available on the IRMCS1043 and IRMCS1188 reference design kits. It describes the topology, control loops, parameter tuning, and hardware design for PFC, with specific references to the design kits.

An additional document, the “IRMCx100 Software Developer‘s Guide”, has instructions on hardware and software requirements and the development process of the embedded 8051 code. The “Application and Software Developer‘s Guides” are designed to take the user through the design process. The reference manual, also referred to frequently in this document, has detailed information on many topics covered here, as well as full descriptions of the 8051 and MCE hardware registers.

Intended audience

This software developer’s guide is intended for customers implementing an inverterized drive.

User Guide #UG-10001 V2.2

IRMCx100 Application Developer's Guide

Version 2.2

By International Rectifier's iMotion Team

Table of Contents

1 Introduction

There are extensive application development activities which the IRMCx100 Series IC user can perform before creating the actual application code for the 8051 processor. This Developer's Guide will begin with the process of initial testing with the target motor, continue with modification of the MCE design for application specific requirements and conclude with the design of motor control hardware for the final application. This Guide assumes that the user is in possession of an iMotion Reference Design Kit and has already completed the activities in the Quick Start Guide. The user should also review the "MCEDesigner User's Guide." This Guide will refer to MCEDesigner features and actions frequently.

Section 2 starts by describing in detail how to measure the parameters of the target motor, generate the correct drive parameters, and begin spinning the motor. Next, this section gives instructions on how to tune the speed and current control loops and optimize the motor start-up parameters. Section 2 concludes with motor drive performance verification and testing methods using MCEDesigner.

Section 3 introduces the MCE processor in more detail and then gives instructions on how to modify the factory-supplied MCE design, if desired. The section finishes with some sample program modifications.

Section 4 guides the user through design, testing and optimization of application specific hardware as it relates to the IRMCx100 motor control IC.

Finally, Section 5 provides application guidance for the Power Factor Correction feature, which applies to IRMCx143 and IRMCx188 only and is available on the IRMCS1043 and IRMCS1188 Reference Design Kits. It describes the topology, control loops, parameter tuning, and hardware design for PFC, with specific references to the Design Kits.

An additional document, the IRMCx100 Software Developer's Guide, has instructions on hardware and software requirements and the development process of the embedded 8051 code. The Application and Software Developer's Guides are designed to take the user through the design process. The Reference Manual, also referred to frequently in this document, has detailed information on many topics covered here, as well as full descriptions of the 8051 and MCE hardware registers.

2 Target Motor on IR Reference Board

This section describes the process of setting up the developer's target motor for reliable operation using the IR Reference Board. Section 2.1 gives detailed instructions on measuring the motor characteristics and using MCEWizard100 to generate the correct drive parameters. The section concludes by guiding the user through importing the drive parameters into MCEDesigner and spinning the motor. Section 2.2 starts testing the target motor in application specific conditions by creating profiles in MCEDesigner. Section 2.3 covers the starting and control algorithms employed by the IRMCx100 IC. This section also covers the process of tuning the speed and current control loops, optimizing starting parameters, and troubleshooting initial drive characteristics.

Before running the motor, the designer should verify that the IR Reference Board is suited to the target motor. Verify that the power rating, continuous current rating, current sensing range, and over current protection level are appropriate to the target motor. You may not be able to safely get full performance from the motor if the hardware does not have the correct ratings. Section 2.2.2 gives some simple modifications to the Reference Board that may address this issue.

2.1 Measuring the Motor Parameters

To efficiently and effectively run a motor, the IRMCx100 motor controller requires certain motor specific parameters, in addition to a variety of hardware and application parameters which will be covered later in this Guide. Each parameter within the control IC is scaled based on the maximum speed, current, voltage, etc. (Specific information on parameter scaling can be found in Secion 2.2.3.) The MCEWizard100 tool is supplied so the designer can enter motor, hardware and application specific information in standard engineering units. When you start at the Welcome Page of MCEWizard100, verify that all of the "Custom Design Questions" are unchecked. As the developer continues through this Guide, these boxes will be checked, giving access to more input parameters. Default parameter values are specific to the motor(s) and hardware of the Reference Design Kit, which is selected on the Welcome Page.

To begin configuring the target motor, its specifications need to be entered into the appropriate sections of MCEWizard100. Often, some of these values can be found on the motor nameplate (Figure 1) and/or the motor datasheet. However, datasheets are not always clear about the motor specifications. The user should pay close attention to units and other variations such as line-line vs. line-neutral measurements and peak-peak vs. rms values.

Note: The way datasheet motor characteristics are specified for Δ-connected motors are different than for Y-connected motors. **The input values to MCEWizard100 are based on a Y-connected motor.** However, if the parameters are measured using the procedures that follow, then the correct value will be found regardless of the motor connection.

Most motor characteristics can also be easily measured, except for three values. The *rated current*, *rated speed* and *maximum speed* should be obtained from the motor manufacturer if they are not available in the datasheet or nameplate. The maximum speed entered into MCEWizard100 should be based on the application requirements and be less than or equal to the manufacturer's stated maximum speed. The controller has overspeed protection, so that a fault is generated if the motor speed exceeds the maximum speed.

There are two other values to input into MCEWizard100: *switch-over speed* and *minimum running speed*. The minimum running speed is generally set to 5 – 10% of the rated motor speed for initial testing though it may be changed for application specific requirements. The IRMCx100 controller requires a minimum motor speed to reliably perform closed-loop speed control. Set the

switch-over speed to 5% of the rated speed, or 5% less than the minimum speed. If there are start-up problems, increase it to 10% of the rated speed.

Figure 1—Motor Nameplate.

The remainder of the motor characteristics can be measured and calculated using an Ohmmeter, LCR meter and oscilloscope:

1. Motor Stator Resistance—Attach the Ohmmeter to two phases of the motor and record the resistance. Measure all three combinations of phases to check the balance of the phases (they should all be nearly the same). Average the three resistance values and then divide by two to get the single phase resistance of the motor.

2. Motor Ld & Lq Inductance—Attach the LCR meter to two phases of the motor, as shown in Figure 2. Change the position of the rotor, seeking out the maximum and minimum value of the inductance. (The rotor should be stationary and the inductance value stable to get a good measurement.) Repeat for the other combinations of phases. Average the maximum values from each phase combination and then divide by two to calculate the value of Lq. Repeat this calculation with the minimum values to get the value of Ld.

   ***Note**: The inductance does not vary with the rotor position for all SPM motors. An interior permanent magnet (IPM) motor has Lq > Ld, and can generally produce a larger torque per Amp. In a surface permanent magnet (SPM) motor, Ld = Lq. In this case, enter the same number for both.*

Figure 2—Measuring the Q phase and D phase Inductances.

3. Motor Poles— Connect two phases of the motor to an oscilloscope. Turn the motor through one revolution and record the back-emf waveform on the 'scope. Count the total number of positive and negative peaks, which should be an even number. Figure 3 shows an 8-pole motor. (It can be difficult to get exactly one revolution without extra peaks. One trick is to turn the motor through several revolutions and then divide by the number of revolutions.)

Note that the motor will still spin if this parameter is not set correctly. However, the mechanical speed of the motor will differ from the requested speed by a factor of [entered poles] / [actual poles].

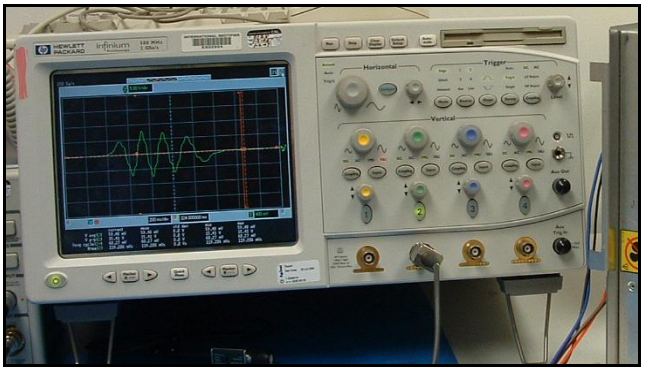


Figure 3—Counting the Number of Poles. This Motor has 8 Poles.

4. Motor Back EMF Constant (Ke)—Again connect two phases of the motor to an oscilloscope. Turn the motor at a constant rate and record the waveform as shown in Figure 4. (When the motor is turning at a constant speed, the back EMF waveform's peaks will all have the same magnitude.) To calculate the back EMF constant, begin by finding the rms voltage and the frequency (in Hz) of the waveform, then perform the following calculation:

$$Ke = 1000 * ([rms\ Voltage] / \sqrt{3}) / ([frequency] * 120/poles)$$

The factor of √3 changes the voltage from line-line to line-neutral. The final units of Ke are Vrms,line-neutral/kRPM. Generally, RPM refers to the mechanical frequency, while Hz refers to the electrical frequency in motor terminology. For accuracy, repeat this measurement at several speeds for each phase pair, and average the Ke calculated from each waveform.

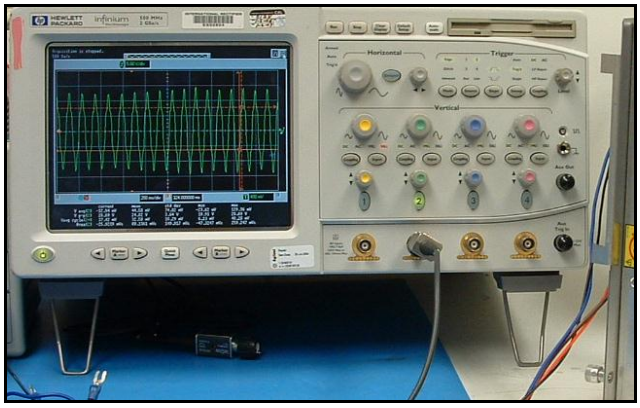


Figure 4—Measuring the Back EMF.

Note: *If the back EMF is not sinusoidal, calculate the rms voltage using a numerical method, like the rms calculation built into most oscilloscopes.*

5. Motor Torque Constant (Kt)—If this is not provided by the motor manufacturer, it can be estimated from Ke. If Ld = Lq, indicating an SPM, then

$$Kt = (9 * Ke) / (100 * \pi)$$

where the units of Ke are Vrms,line-neutral/kRPM and the units of Kt are N-m/Arms. If Lq > Ld (for an IPM), then the torque constant is current dependent. To estimate, increase the value calculated above by 5%.

6. Motor Total Shaft Inertia—This parameter is application dependent. For example, a full washer may have large load inertia while a pump has small load inertia. The inertia is used to estimate the motor speed during the open-loop period of the start-up sequence. In practice, this parameter does not need to be extremely accurate. During application testing, this can be varied to optimize the start-up performance of the motor.

The IRMCx100 Series sensorless motor controller can tolerate +/-10% motor parameter error without noticeable performance degradation. An increased parameter mismatch between the motor and controller will result in a degradation of torque per Amp capability. The degree of degradation is dependent on the operating conditions (speed, load) and motor characteristics (motor parameters and saturation).

2.1.1 Importing Drive Parameters into MCEDesigner

By selecting the correct Reference Design in the Welcome page of MCEWizard100 the default values for the rest of the inputs can be used to configure the controller; just check that the value entered for the Nominal DC bus Voltage is correct. From the Verify & Save Page (Figure 5), press “Calculate” and if there are no errors, select “Export to MCEDesigner File (.txt).” Save the file with a name that refers to the motor. Start MCEDesigner, open an .irc file and click on the “System” window. From the File menu, select “Import Drive Parameters” and select the text file you just created. Choose “Update All” from the next window and press OK.

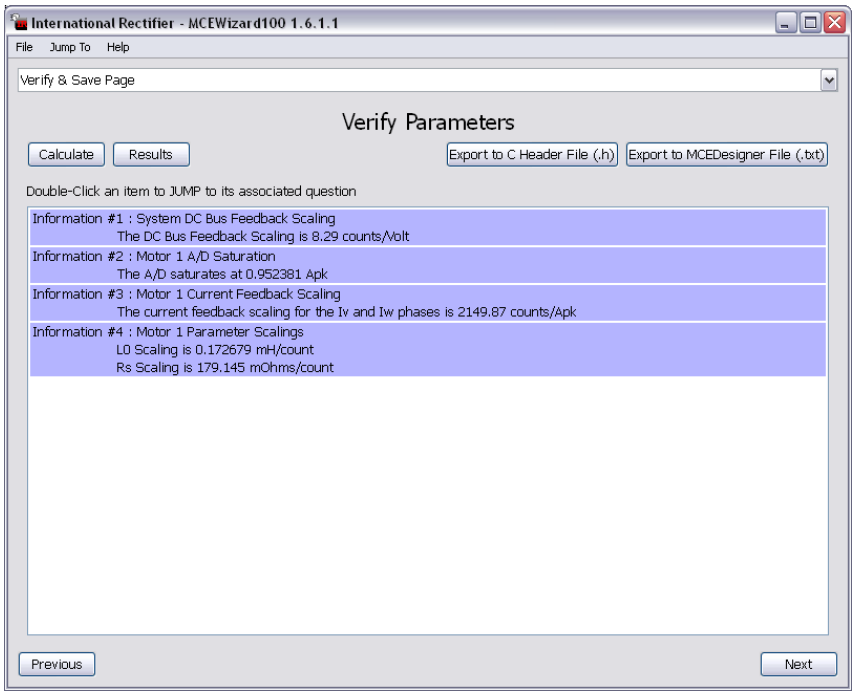


Figure 5—MCEWizard100 Verify & Save Page

Be sure that the motor is connected and the board is powered up, with the COM active. Double click the “Configure Motor” function to write the new drive parameters to the controller. Next, double click the “Start Motor” function. The motor should begin to turn! Verify that you can accurately vary the motor speed using the “Reference Speed” function. (Right-click and select “Properties” to change the speed value; double-click to write the new value to the control IC.)

Save the MCEDesigner .irc file with a descriptive name which refers to the target motor. The next time the .irc file is opened, it will already have the drive parameters saved. Simply run the "Configure Motor" function to write the values to the controller. For more information on importing drive parameters, please see the MCEDesigner User's Guide.

2.1.2 Advanced Parameter Measurement—Saturation Effects

Many motors suffer from saturation effects, where the inductances (Ld & Lq) decrease with increasing phase current. Check with the motor manufacturer for data about the saturation. To measure the saturation, apply a DC voltage (Vdc) to two phases of the windings and measure the current as a function of time. The instantaneous slope of the curve is equal to Vdc/L at the corresponding current level. The developer may have to use the saturation inductance at the rated current in order to get the maximum torque. Generally, Lq will exhibit a greater degree of saturation than Ld. Figure 6 shows a sample saturation curve.

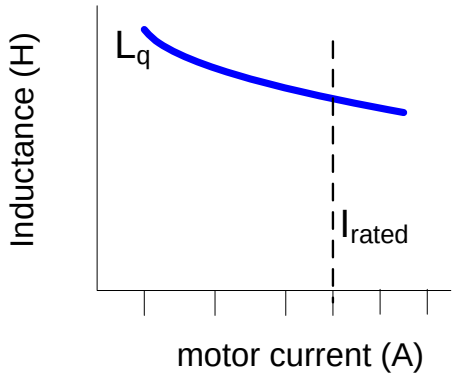


Figure 6—Saturation of Motor Inductance

2.2 Starting Motor Application-Specific Testing

Section 2.2.1 describes the process of testing the target motor in the real target application using MCEDesigner. Section 2.2.2 describes some minor modifications which can be done to the reference hardware to make it more suitable for the target motor or application. Next, Section 2.2.3 gives a list of the important internal variables and the scaling factors to relate them to physical, measurable quantities. Verifying parameter scaling is an important part of the debugging process, particularly when modifying the hardware; this process is described in Section 2.2.4.

2.2.1 MCEDesigner

The main tool used to test the motor with application-like profiles and timing is MCEDesigner. A test profile should recreate the speeds, acceleration, and timing of the real application. After tuning the drive parameters as described in Section 2.3, the motor should be tested with simulated or real loads at this stage. Both the Quick Start Guide and the MCEDesigner User's Guide have detailed instructions on creating and modifying functions.

Another MCEDesigner tool to become familiar with is the parameter Trace (Monitor) which is also covered in the Quick Start Guide and the MCEDesigner User's Guide. The parameter trace allows the designer to see the value of internal registers of the motor controller. Figure 7 shows the Trace Setup window, where the trigger settings, data sources and output file can be defined. The trace collects register values on a PWM synchronous basis with an option to down-sample to extend the trace duration; the trace length is fixed at 256 samples for each channel, with a

maximum down sampling of 255. The trigger options are Force Trigger, Trigger on Level, Trigger on Fault, and Auto Repeat Level.

Some strategies for debugging motor control problem situations using the trace:

- Use the Motor Status Flags or SequencerState register to trigger the Trace to determine at which stage the problem happened. This is particularly useful for start-up problems. (The StatusFlags and SequencerState registers are defined below.)
- Use the "Trigger on Fault" setting for unpredictable problems.
- To get an auto-repeating Force Trigger for monitoring a single value, use Auto Repeat Level, triggering on the RotorAngle register.

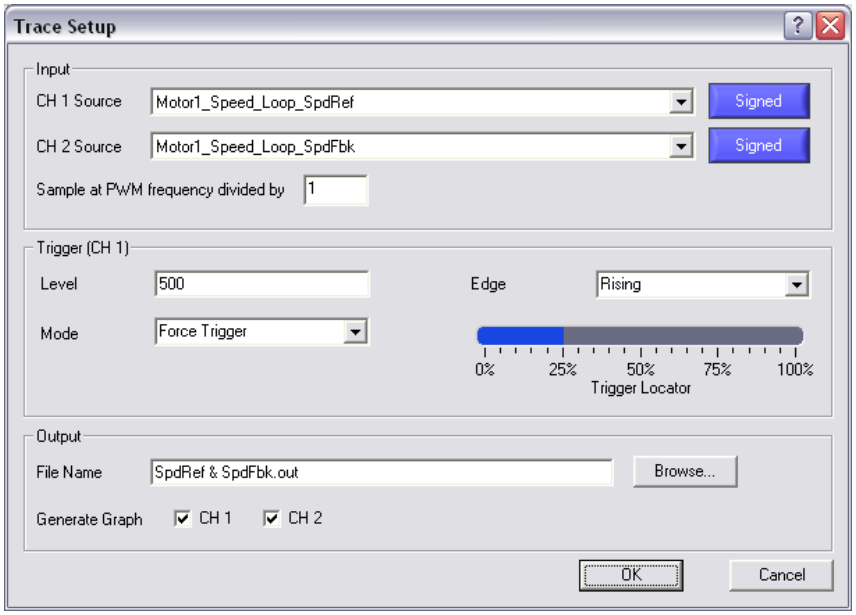


Figure 7—Trace Setup Window

2.2.1.1 Motor Control Sequencer

The firmware of the IRMCx100 includes a built-in sequencer which takes care of all state handling for starting, stopping and performing catch-spin starting. The sequencer automatically performs all of the steps required for robust control and startup. The user only needs to write to register MtrSeqCtrl—writing a value of "2" will start the drive in FOC mode, while writing a value of "4" will stop the drive by turning all the inverter switches off.

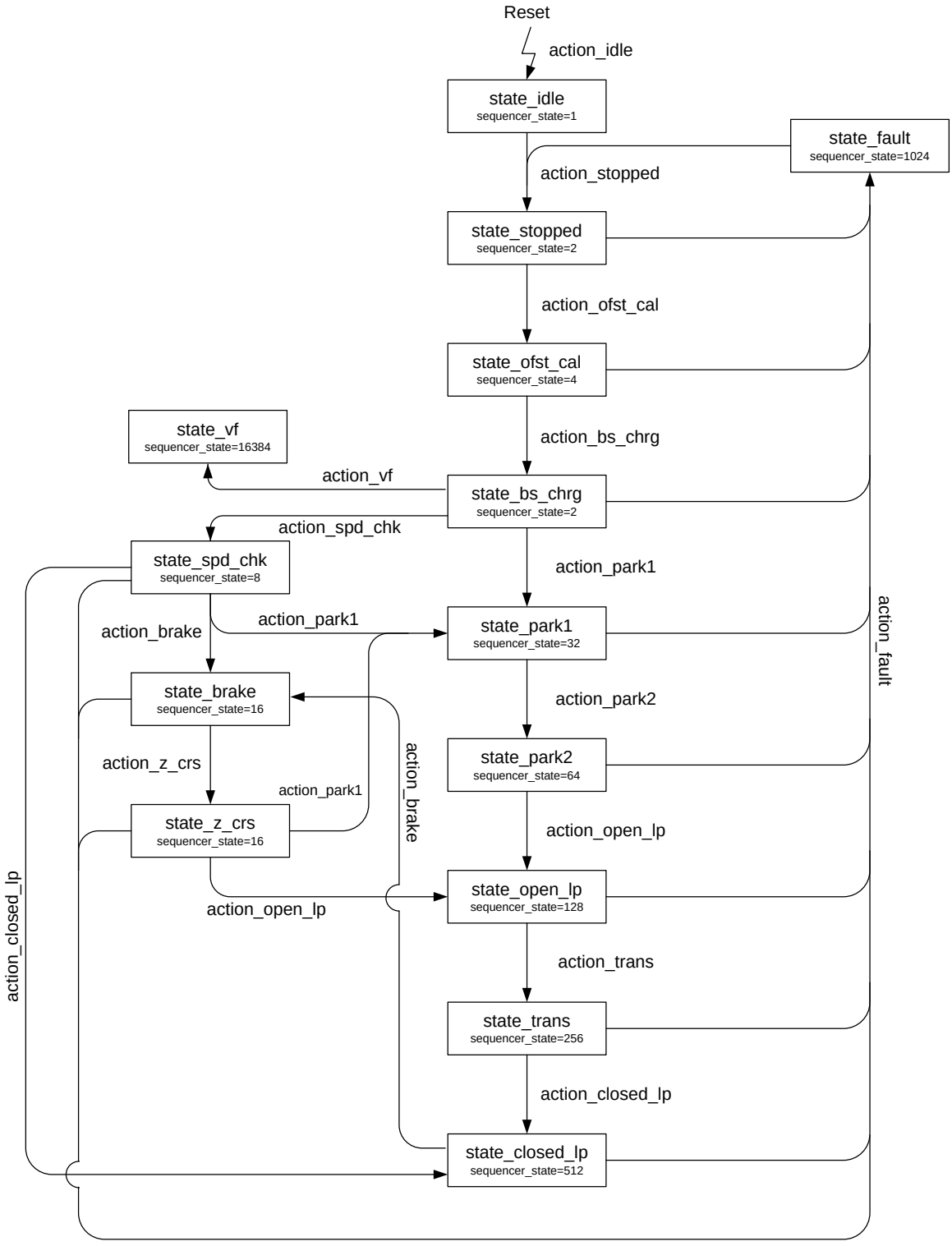


Figure 8—Sequencer State Diagram

Figure 8 above shows the detailed state diagram of the motor control sequencer. The details of each state and the transitions are described in more detail in Section 2.3. As the sequencer executes, its state can be read from the register SequencerState as defined below:

SequencerState Value	Motor Start-up Stage
1	idle state
2	bootstrap capacitor precharge
4	measure offset current
8	check speed (in catch-spin)
16	braking (in catch-spin)
32	first stage parking
64	second stage parking
128	open loop startup
256	transition from open to closed loop mode
512	closed loop mode
1024	fault pending
16384	open-loop v/f control

2.2.1.2 Motor Status Flags

There is a set of system status read registers, which are useful diagnostic registers for identifying and debugging motor control problems. It provides the motor status, particularly which stages of start-up the motor has completed. Table 1 gives the registers and the associated drive status. Please see Reference Manual for details.

Registers	**Description**
StartOk	Startup has succeeded (cleared whenever drive stops).
ClosedLoop	Closed-loop mode is enabled.
ParkingDone	Parking done; Parking stage has been completed.
ParkingOne	First stage (25% of the total park time) of Parking has been accomplished.
StartFail	Startup has failed.
TwoPhsStatus	Two phase modulation is enabled.

Table 1—Motor Status read registers

2.2.1.3 Fault Flags & Fault Handling

The FaultFlags register specifies which fault condition has occurred. In MCEDesigner, the fault status is displayed in the status bar at the bottom of the window. If the motor, for example, has a fault, then the status light will be red. Moving the pointer over the red status light will bring up a small text box which lists the faults. Table 2 gives the fault associated with each bit of FaultFlags.

Bit	Field	Description
0	GateKill	Motor gatekill fault
1	CritOVFault	Critical overvoltage fault
2	OvFault	DC bus overvoltage fault
3	LvFault	DC bus undervoltage fault
4	OverSpdFlt	Motor overspeed fault
5	ZeroSpdFlt	Zero speed fault
6	PhsLossFlt	Phase loss fault
7	StartFailFlt	Start fail fault
8	MCEFlt	The MCE has generated a fault condition
9	CsOverSpdFlt	Catch Spin overspeed fault
10	MCEExeFlt	MCE execution fault
11	PFCGateKill	PFC gatekill fault (available for 143 only)
12	ADCompFlt	A/D Compensation fault
13 – 15	*Unused*	

Table 2—FaultFlags Bit Definitions

In the case of a fault condition occurring, the fault is latched into FaultFlags, the motor drive is shut down and an interrupt to the 8051 is generated. To clear a fault condition, write “1” to the FaultClear register and then set it back to “0”. Except for Motor/PFC Gatekill and Critical Over Voltage, any of these faults can be disabled using the DisableFaults register. In the case of a disabled fault, the fault is still reported in FaultFlags and an interrupt is still generated. However, the motor drive will continue and the fault bit in FaultFlags will clear when the fault condition disappears, i.e. the fault is not latched.

2.2.2 Possible Hardware Modifications

All of the hardware modifications described below will require new configuration parameters. The designer should return to MCEWizard100 and change the appropriate input values. In the factory default setting, none of the “Custom Design Questions” on the Welcome Page are checked, which prevents modification of certain input values that do not need to be changed. For the hardware changes described below, check the box next to “I have modified the circuit board” to access the appropriate values.

2.2.2.1 Current Feedback

In some cases, the inverter system may be suitable, but the current scaling is not optimal. The current feedback scaling can easily be modified by changing the current feedback op-amp gain or by changing the shunt resistor. These modifications are useful in situations where the A/D saturation current (given in the MCEWizard100 Verify & Save page) is less than or much greater than the maximum current required by the application.

Figure 9 below shows a sample current feedback amplification circuit. The node labeled “IFB” is connected to the inverter side of the shunt resistor. In this circuit, the op-amp gain is 11.8/6.11 = 1.93. To modify the op-amp gain, the designer should change resistors in pairs (R83 & R80; R81 & R82; R77 & R79) to preserve the correct circuit biasing at AREF (0.6V reference).

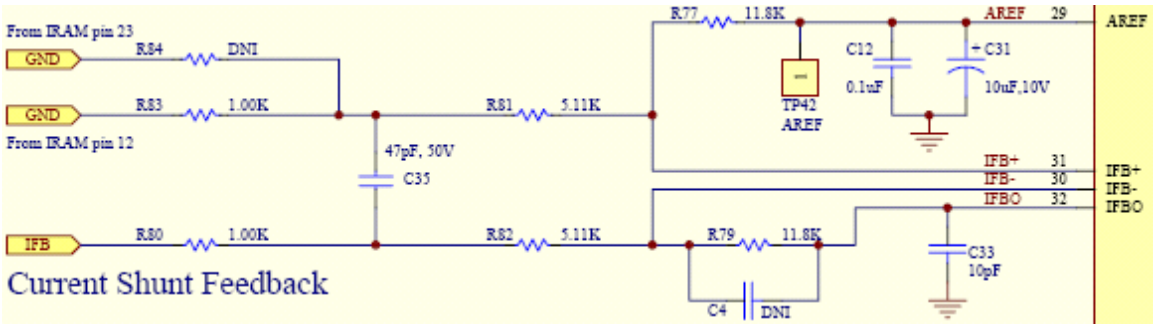


Figure 9—Current Feedback Circuit

Note: The developer may not be able to change the shunt resistor independently of the IRAM module, depending on the iMotion Reference Board in use.

In the MCEWizard100, enter the appropriate value into the Current Feedback Amplifier Gain and Current Feedback Shunt fields to get the correct drive parameters.

2.2.3 Variable Scaling

This section gives formulas to convert internal MCE variables to real, physical values corresponding to the motor operation and condition. Some variables in this section may be unfamiliar to the designer; they are explained more fully in further sections of this guide. Many of the parameter scalings are determined by values input into MCEWizard100. Determination of correct variable scaling is an important step when verifying the correct operation of the hardware.

2.2.3.1 Speed Scaling

Normal Operating Mode

TargetSpeed, **SpdRef** and **SpdFbk** scale such that

*Rotor Speed (RPM) = [TargetSpeed] / 16383 * Motor Max RPM*

where
Motor Max RPM is an entry of MCEWizard100.

Rtr_Freq, the estimated unfiltered rotor electrical frequency, scales such that

*Actual electrical frequency (Hz) = Rtr_Freq * FreqPwm * FreqScl / 2^20*

where
FreqPwm is the Motor PWM Frequency entry of MCEWizard100
FreqScl is set by bit fields of **MtrCtrlBits** and **MtrCtrlBits_S** (see IRMCx100 Reference Manual)

SpdScl is configured by MCEWizard100 as follows:

$SpdScl = 60 * 2 / poles * FreqPwm * FreqScl / 2^{10} * 16383 / Motor\ Max\ RPM$

where
Motor Max RPM an entry of MCEWizard100
FreqPwm is the Motor PWM Frequency entry of MCEWizard100
FreqScl is set by bit fields of **MtrCtrlBits** and **MtrCtrlBits_S** (see IRMCx100 Reference Manual)

In IR's released version of the MCE reference design, **SpdScl** is used to convert from the **Rtr_Freq** scaling to the **SpdFbk** scaling.

V/Hz Diagnostic Mode

When the FOC block is configured for Volts/Hz diagnostic mode (Register MtrCtrlBits), then the speed scaling of **VFFreq**, **TargetSpeed** and **SpdRef** is as follows:

*Rotor Speed Setpoint (RPM) = [SpdRef] * 0.01552583 * 120 / poles*

The **SpdFbk** is invalid in V/Hz diagnostic mode.

2.2.3.2 Torque Scaling

TrqRef is correctly evaluated in terms of current. However, for the purposes of torque estimation or torque control, the register scales as:

*Motor Torque (N-m) = Irated * Kt * TrqRef / 4095*
where
Irated is the Motor Rated Current in A_{rms} as entered into MCEWizard100
Kt is the Motor Torque Constant in N-m/A_{rms} as entered into MCEWizard100

It should be noted that this method estimates the torque assuming that Kt is constant over the speed and motor current range of operation.

2.2.3.3 Current Scaling

When the settings of MCEWizard100 are properly set, the scaling of registers **IdRef_C**, **IqRef_C**, **Di**, **Qi**, **TrqRef**, **IdRefExt**, **Id_Decoupler**, **StartLim**, **MotorLim**, **RegenLim** are all the same:

*Current (A) = Irated * √2 *[IdRef_C] / 4095*
where
Irated is the Motor Rated Current in A_{rms} as entered into MCEWizard100

This scaling is achieved by the **IfbkScl** register, one of the register values calculated in MCEWizard100. The following diagram and equation show how the value of **IfbkScl** is determined:

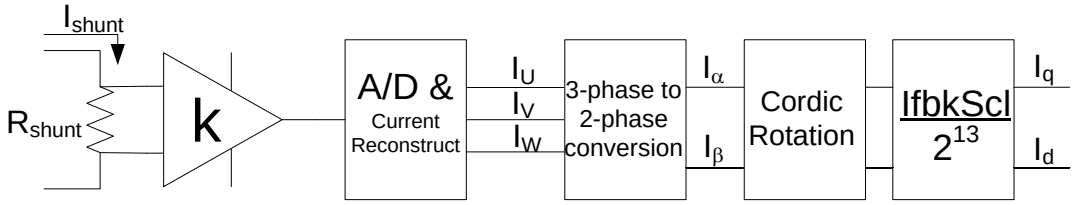


Figure 10—Current Feedback Signal Path

*4095 = Irated * √2 * Rshunt * k * A/D * (3 → 2) * (cordic) * (IfbkScl / 2^{10})*
Where:
Irated is the Motor Rated Current in A_{rms} as entered into MCEWizard100
Rshunt is the Current Feedback Shunt resistor value in Ohms as entered into MCEWizard100
k is the Current Feedback Amplifier Gain entry of MCEWizard100
A/D is the analog-to-digital converter scaling (3412 / Volt)

3 → 2 is the 3 phase to 2 phase conversion gain (1.0)
cordic is a factor introduced by the hardware vector rotator (1.64676)

Intermediate Signals

$I_{U,V,W}$—During each PWM cycle, two of the three phase currents are sampled in the shunt resistor and digitized in the A/D converter. The current feedback offset (**IfbOffset**) is subtracted from the raw A/D output. Finally, the third phase current is reconstructed using the relation U + V + W = 0. The U, V and W phase currents correspond to **IfbU**, **IfbV** and **IfbW**, respectively. The scaling for these currents can be found in the Verify & Save page of MCEWizard100 or can be calculated by:

*Current (A) = [IfbV] / (Rshunt * k * A/D)*

Where:
A/D is the analog-to-digital converter scaling (3412 / Volt)
Rshunt is the Current Feedback Shunt resistor value in Ohms as entered into MCEWizard100
k is the Current Feedback Amplifier Gain entry of MCEWizard100

$I_{\alpha,\beta}$—These currents are a 2-phase representation of the real U, V, and W phase currents. The α and β phase currents correspond to registers **I_alpha** and **I_beta**. Their scaling is the same as that of the real phase currents:

*Current (A) = [I_alpha] / (Rshunt * k * A/D * (3 → 2))*

Where:
Rshunt is the Current Feedback Shunt resistor value in Ohms as entered into MCEWizard100
k is the Current Feedback Amplifier Gain entry of MCEWizard100
A/D is the analog-to-digital converter scaling (3412 / Volt)
3 → 2 is the 3 phase to 2 phase conversion gain (1.0)

The **IScl** parameter specifies the current gain scaler for the flux estimator. MCEWizard100 calculates this parameter. Please do not tamper with this parameter without consulting the iMotion design team.

2.2.3.4 Voltage Scaling

Input DC and AC Voltages

DcBusVolts and **DcBusVoltsFilt** have the same scaling, which is the DC bus Feedback Scaling entry of MCEWizard100. (**DcBusVoltsFilt** has a 0.492 msec time constant.) This scaling is hardware dependent. In IR's Reference Design Kits, the DC bus voltage signal is reduced through a voltage divider, shown below. Then this voltage is supplied to AIN0 to go to the A/D converter.

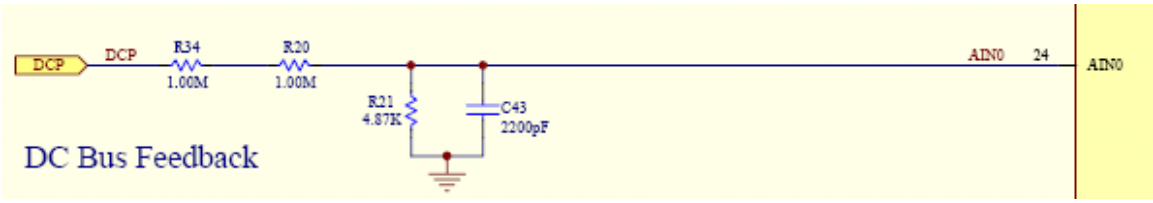


Figure 11—DC bus Feedback Circuit

The DC bus voltage can be calculated as follows:

*DC bus (V) = [DcBusVolts] / (A/D * r)*

where
A/D is the analog-to-digital converter scaling (3412 / Volt)
r is the voltage divider ratio (4.87k / (2M + 4.87k) in the figure above)

CriticalOvThr, **DcBusOvLevel**, and **DcBusLvLevel** scale as follows:

*Voltage Trip Level (V) = [CriticalOvThr]*16 / (A/D *r)*

where
A/D is the analog-to-digital converter scaling (3412 / Volt)
r is the voltage divider ratio (4.87k / (2M + 4.87k) in the figure above)

2.2.3.5 Rotor Angle Scaling

RotorAngle gives the estimated rotor electrical angle from the Rotor Angle Estimator PLL which scales as follows:

*Electrical Angle (degrees) = RotorAngle * 90 / 1024*

2.2.3.6 Parking Variables

ParkTm sets the duration of the parking stage of the start-up sequence. The parking time is calculated as follows:

Parking time (s) = ParkTm / 64

ParkI is the DC current injected into the motor during the parking stage. It defines the current in terms of the peak rated motor current. The actual parking current in a particular phase will also depend on the parking angle. The W-phase motor current can be calculated by:

$$\text{W-phase Parking Current (A)} = \text{Irated} * \sqrt{2} * \text{ParkI} / 2^{10} * \cos(\text{Parking Angle} - 60°)$$

ParkAng and **ParkAng1** specify the angles to be used in the parking stage of startup. During parking, two parking angles are used which are defined relative to the motor U-phase. The drive will first use **ParkAng1** for 25% of the total parking duration); thereafter, the parking angle will switch to **ParkAng** to complete the parking duration. These parameters scale such that

*Parking Angle (degrees) = [ParkAng] * 90 / 64*

2.2.4 Verifying Scalings

2.2.4.1 Verifying Current Scalings

When setting up a new system, it is important to verify that the current scaling is correct. There are several ways to do this, two of which are described below:

1. VF Diagnostic

The VF Diagnostic is a useful function for verifying proper operation of the power stage independently of the current feedback and angle estimation. In this function, the motor is operated in a V/Hz mode, where the ratio of voltage applied to electrical frequency is a constant determined by **VFGain**. This mode is particularly useful for driving an induction motor. The VF Diagnostic only operates in one direction. To turn an induction motor in the opposite direction, simply switch two of the motor phases.

Using the **TargetSpeed** and **VFGain** parameters, run an induction motor with the desired current level to be verified. Record the V and W phase currents using a current probe and by tracing in MCEDesigner. Use the scaling value given by MCEWizard100 to verify that the traced current matches the real current by comparing the magnitude of the sine waves.

The figures below show the induction motor currents of the V (yellow) and W (green) phases, as recorded on an oscilloscope (1A/division scale) and the MCEDesigner trace. For the hardware in this test, the current scaling for IfbV and IfbW is 219.83cts/Amp. From these figures, one can verify that the current scaling is correct as follows:

The amplitude of the current recorded on the oscilloscope is 3A, peak.
3A * 219.83cts / A = 659.49cts
The amplitude of the current as recorded on the trace is about 675cts, which matches well with the expected value.

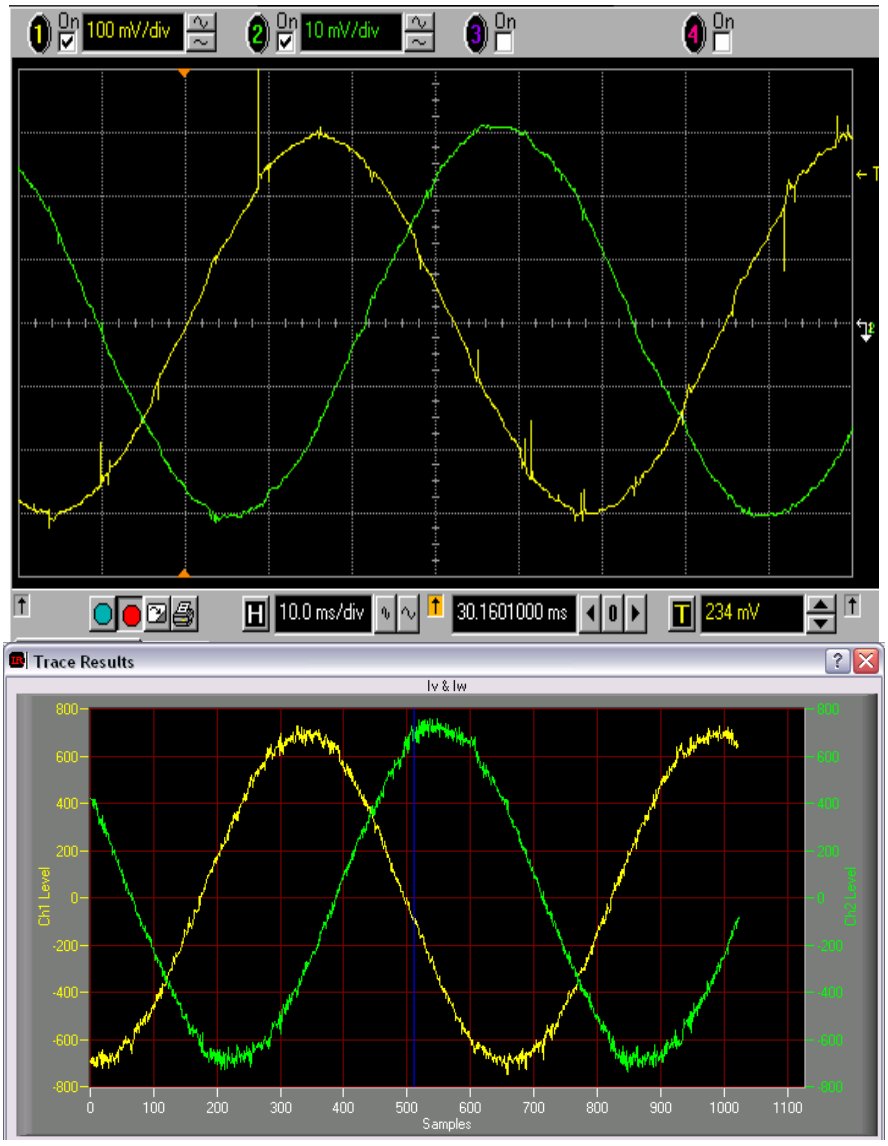


Figure 12—Verifying Current Scalings

2. Parking Diagnostic

The Parking Diagnostic function of MCEDesigner can also be used to verify the current scaling. During the parking diagnostic, DC current is supplied to the motor windings. There are several settings to the parking diagnostic including the parking angle and the parking current. For more information on the parking diagnostic, see Section 2.3.2.2.

Similar to the above procedure, run the parking diagnostic, measure the phase currents on an oscilloscope and trace the variables IfbV and IfbW in MCEDesigner. Verify IfbV and IfbW using the measured value and the scaling factor given in the Verify & Save page of MCEWizard100.

2.2.4.2 Verifying DC Bus Scaling

To verify the DC bus scaling, read the register "DCBusVoltsFilt" and then divide by the DC bus Feedback Scaling, which is an entry of MCEWizard100. Compare this value with the DC bus voltage measured using a multimeter or other instrument.

2.3 Optimizing Motor Starting and Running Parameters

This section includes descriptions of the start-up sequencing and control loops of the IRMCx100 Series ICs. Included with the descriptions are procedures and suggestions on how to tune and optimize the control parameters for your application.

2.3.1 Before Start-Up

The MCEDesigner program contains a "Start Motor" function which simply sets the motor direction and speed and then starts the sequencer (see Figure 8) using MtrSeqCtrl. The sequencer performs two important pre-startup actions: Offset Correction and Bootstrap Pre-charge.

The Offset Calibration is the first pre-start activity. The sequencer performs the current feedback Offset Calibration to compensate for offsets in the current feedback path including the A/D offset and reference voltage offset. The calibration is achieved by averaging 4096 samples from the current feedback circuit when the motor is not running. The Offset Calibration can be skipped by setting the appropriate bit in register MtrCtrlBits_S bit 6.

The Bootstrap Pre-charge is performed just before parking. The bootstrap pre-charge turns on the low-side IGBTs in sequence to charge the bootstrap capacitors of the gate drivers and can help to prevent overcurrent trips. The pre-charge stage is configured using registers GCChargePW, GCChargePD, and GCChargeT. Set GCChargeT to zero to skip the Bootstrap Pre-Charge sequence. More information on this topic can be found in the IRMCx100 Reference Manual.

2.3.2 Start-Up Tuning

2.3.2.1 Start-Up Sequence

Because the motor control relies on back EMF for position control, the MCE processor must go through a special startup sequence to provide for robust starting. The startup control block inside the Sensorless FOC block (see the Reference Manual) is used to assist drive startup with the ability to sequence the controller dynamically to three unique operating states (Parking, Open-loop or Closed-loop). These three states are illustrated in Figure 13 and described below.

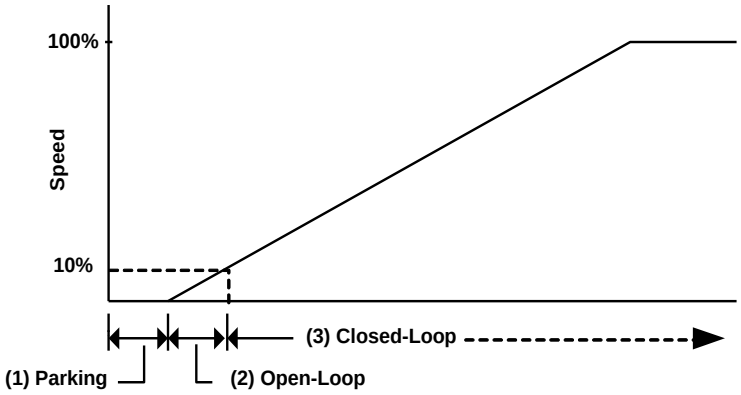


Figure 13—Drive Control Modes

State 1: Parking

The initial rotor angle is identified by forcing DC current into the motor and hence forcing the motor shaft to park at a known angle.

State 2: Open-loop angle estimation

Immediately after the Parking stage, the rotor angle is estimated with a simple motor-load mechanical model. If the mismatch between the external load characteristics and the internal motor-load model is exceedingly large, start-up performance will suffer. In this stage, the motor current is controlled and only the rotor angle is estimated.

State 3: Closed-loop angle estimation

Motor speed increases during start-up, resulting in a build-up of the motor back EMF. Useful information for rotor angle estimation can then be extracted from the motor back EMF voltage (estimated by using the PWM modulation depth and DC bus voltage). The drive will enter Closed-loop control mode as shown in Figure 13.

In addition to implementation of these three states, the startup control unit can also detect successful drive startup and signal the main Motor control sequencer of a startup failure (via the StartFail register).

2.3.2.2 Parking Parameters & Parking Diagnostic

There are several parking parameters which are used to optimize the parking state of the start sequence. These parameters are described below. More detail about each register can be found in the IRMCx100 Reference Manual.

ParkAng1 and **ParkAng**

These parameters specify the angles used in the parking stage of startup, which is defined with respect to the motor U-phase. During parking, two parking angles are used. The drive will first use ParkAng1 for 25% of the total parking duration; thereafter, the parking angle will switch to ParkAng to complete the parking duration. Two parking angles are used in order to guarantee that the rotor moves to the final park angle. If the rotor's initial position is 90° (electrically) away from the parking angle, then it will not experience any torque.
Scaling: 64 = 90 Degrees *Range*: 0 -255

ParkI

This parameter specifies the amount of dc current injection during the parking stage function of the motor rated current entered into MCEWizard100.
Scaling: See Section 2.2.3.6. *Range*: 0 - 255

ParkTm

This parameter specifies the total parking duration. The motor will park at ParkAng1 for 25% of the ParkTm and at ParkAng for the remaining time. The maximum parking duration that can be set directly using this register is four seconds, though this time can be extended using the Parking Diagnostic function described below.
Scaling: 255 = 4 sec *Range*: 0 – 255

The **Parking Diagnostic** mode of the control IC can be used to optimize the parking phase of the motor start sequence, entered by setting the appropriate bits of MtrCtrlBits. This mode drives the first park angle for 25% of ParkTm, and then park at ParkAng indefinitely, until the drive is stopped or the parking diagnostic disabled.

The following example illustrates a procedure to use an extended park time. In the example, parking time is extended to ten seconds by activating the Parking Diagnostic for ten seconds (steps 1 – 4) and then resuming normal drive operation with zero parking time (steps 5 – 7).

1. DiagSelect field of MtrCtrlBits = 1 (enable Parking Diagnostic).
2. Start drive.
3. Delay ten seconds.
4. Stop drive.
5. DiagSelect field of MtrCtrlBits = 0 (disable Parking Diagnostic).
6. ParkTm = 0 (zero parking time since parking is already established).
7. Start drive.

2.3.2.3 Parking Optimization

Correct parking is particularly important in situations where large starting torque is required. The parking stage allows the controller to match the starting current phase angle to the rotor electrical angle, maximizing the torque.

Some parking situations to beware of (with suggestions):

- The rotor is still moving at the end of the parking time. Try increasing the parking time to allow the rotor to settle down to the parking position.
- The rotor does not move to the proper angle during parking. Try increasing the parking current to provide more parking torque. A fully loaded washing machine may exhibit this behavior.
- The rotor oscillates around the parking position. Is the inertia or friction very small? Try using a smaller parking current.
- Due to cogging torque, the rotor moves away from the parking position when the parking current is removed. Experiment with parking angles to find one which is stable when the motor windings are not energized. Be aware that the parking positions are electrical angles; this means that there are poles/2 different mechanical parking positions for each park angle.
- If the application inertia is low, and the motor friction is very small, then it can be very difficult to park the rotor. In this situation, it may be more reliable to start without parking. Set the ParkTm to zero and the open loop stage will begin without parking. This can often be the case in a motor with large cogging torque.
- To soften the parking “jolt,” reduce the current regulator bandwidth during parking, as may be required in a mechanical system with a gearbox. This is also useful for reducing the overshoot in situations where high parking toque is needed, resulting in less parking oscillation.

2.3.2.4 Open-Loop angle estimation to Closing the Loop

During the open-loop stage, the motor electrical angle and speed are estimated using the load inertia and the motor torque constant supplied to MCEWizard100. The open-loop estimated acceleration rate per amp of starting current is set by the parameter KTorque.

KTorque

This parameter specifies the motor mechanical model gain used in the Open-loop startup stage. KTorque relates the torque applied by the motor to the drive acceleration. This gain plays an important role in robust startup. *At rated motor current*, the scaling is given by:

Acceleration Rate (Hz/sec) = KTorque * FreqPwm * FreqPwm / 2^29
where FreqPwm is the inverter switching frequency in Hz.

For instance: At rated motor current, 10 KHz inverter PWM frequency and KTorque = 100, the controller will estimate an 18.63 Hz/sec acceleration rate during the open-loop stage. At 50% rated motor Amps, the acceleration will be 50% of this value.

Once the controller estimates that the motor has reached the threshold speed (internal parameter WeThr) then the angle estimator PLL is started. It is important that the actual speed be large enough for the PLL to get good angle estimation (typically 5-10% of rated motor speed).

WeThr

This parameter specifies the transition level (frequency) from Open-loop to Closed-loop mode operation. The scaling of WeThr is related to internal frequency scaling of the drive by:

WeThr = SwFreq * 2^20 / FreqScl / FreqPwm

where:
SwFreq is the desired switch over frequency in Hz (typically 5 to 10% rated motor electrical frequency), called “Freq Switch-Over to Closed-loop Control” in MCEWizard100;
FreqPwm is the inverter pwm frequency in Hz; and
FreqScl is the frequency scaler, determined by bit fields of the MtrCtrlBits_S and MtrCtrlBits registers, respectively. (See the IRMCx100 Reference Manual for more information.)

In the case of KTorque and WeThr it is most convenient to set them using MCEWizard100 by setting the Load Inertia and Threshold Frequency, respectively.

2.3.2.5 Troubleshooting the Closing of the Loop

The most important factor in successfully transferring from open-loop to closed-loop control is the motor speed. The motor must generate a large enough back EMF for the angle estimator PLL to lock onto the rotor angle. There are several ways to ensure that the motor reaches this speed before the controller closes the loop.

- Reduce KTorque (by increasing the Total Shaft Inertia in MCEWizard100): With a lower value of KTorque, the controller will estimate a longer time for the motor to reach the threshold frequency, and the drive frequency will increase at a slower rate during the open-loop stage.
- Increase WeThr (by increasing the Switch-over Freq in MCEWizard100): This will also increase the duration of the open-loop period, but the drive acceleration rate will not change.
- Modify the TargetSpeed: When the loop is closed, the controller will rapidly accelerate the motor to the TargetSpeed. This can be undesirable, so set the TargetSpeed close to the threshold speed. Increase TargetSpeed to the desired value after the loop is closed, and use the speed ramp rate to control the acceleration. More information about the speed ramp can be found in Section 3.2.

- Low Voltage Fault: If the DC bus supply is not capable of supplying the start-up current, then a Low Voltage Fault may occur. If this is the case, try reducing the Start Limit in MCEWizard100, if the low speed load is small.

The selection of threshold frequency can be evaluated by running the motor at a constant speed and then checking how well the speed feedback matches the actual speed. Gradually reduce the motor speed until the speed feedback becomes inaccurate or noisy. Place the threshold frequency at a value which gives good speed feedback.

Another problem can occur during the switch-over due to the time required for the PLL to stabilize. Start the motor and trace the speed feedback (SpdFbk variable) during the start-up. The speed will rise smoothly in a parabolic curve during the open-loop stage, and then it may spike or dip before stabilizing at the running speed due to the PLL stabilizing, as shown in Figure 14. In most application situations, the PLL oscillations do not cause any starting problems. In Section 3.5.2 an example of modifying the MCE program to dampen this spike is given.

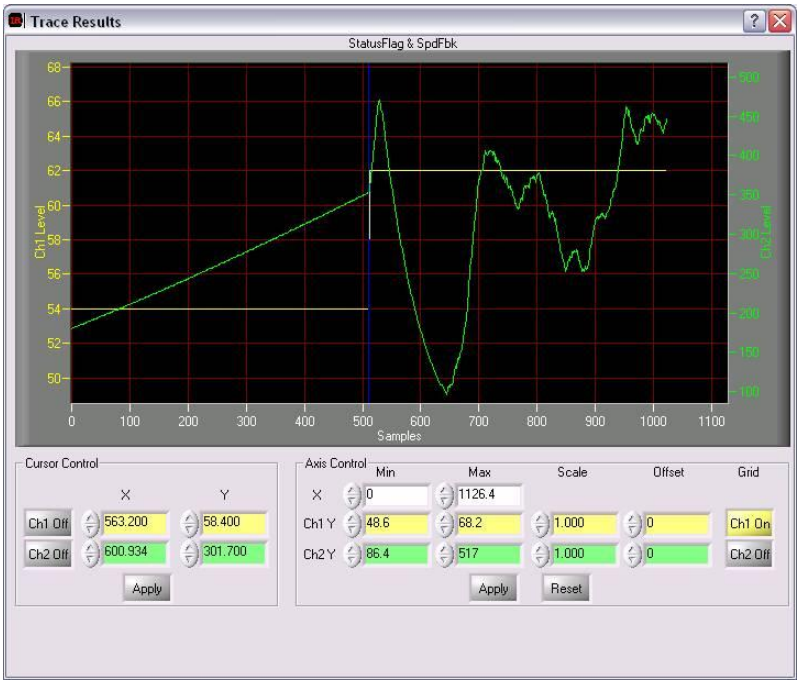


Figure 14—SpdFbk (green) when the Loop is Closed at the Blue Vertical Line

2.3.2.6 Start Fail

The control IC can detect a start failure in the motor. The developer must enable the detection of this fault mode, if desired, by clearing the correct bit in register DisableFaults (or selecting the desired option in MCEWizard100).

Start Fail—A start failure is detected by sampling the motor flux at a certain time after the controller enters Closed-loop mode (set by FlxChkT register). This error mode appears as a bit flag in both the FaultFlags and StatusFlags register. The successful-startup flux range is set by the registers FlxThrH and FlxThrL. More information about these registers can be found in the IRMCx100 Reference Manual.

2.3.2.7 Zero Speed Detection

The control IC can detect zero speed errors in the motor. The developer can disable the detection of this fault mode using DisableFaults (or MCEWizard100).

Zero Speed Fault—This fault appears as a bit flag of the FaultFlags register. The Zero Speed Fault is asserted when the motor speed falls below half the minimum speed (MinSpd) for two seconds. This allows the controller to detect problems, such as a locked rotor. In MCEDesigner, a Zero Speed Fault will shut down the drive.

2.3.2.8 Phase Loss Fault

The control IC can detect a motor phase loss. This fault can be enabled or disabled as desired using register DisableFaults (or MCEWizard100).

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. When the Phase Loss Fault is enabled, the controller detects this condition and turns on the appropriate bit flag of the FaultFlags register.

Phase loss detection is carried out at the end of each parking stage. The registers AdjPark1 and AdjPark2 have to be configured properly with respect to the parking angles; the configuration is correctly done by MCEWizard100. Sufficient parking time and current is also required for reliable Phase Loss detection. See the Reference Manual for detailed information.

2.3.3 Catch-Spin Starting

"Catch-Spin Start" is a feature designed for situations where the motor may already be turning. The catch-spin start is generally effective up to the rated speed of the motor. Catch-spin cannot be done if the motor back EMF voltage is higher than the DC bus voltage; this usually occurs when the motor is above rated speed. The catch-spin starting process is part of the sequencer and executes at start-up if MtrCtrlBits_S[0] = 1. The catch-spin sequencing (in Figure 15) is as follows:

In catch-spin, while forcing the motor currents to zero, the controller tracks the back EMF in order to determine if the motor is turning, and if so, in which direction (yellow):

- If the motor is turning in the opposite direction than that desired, zero vector braking is initiated (See Section 2.3.5.2.) to stop the motor (beige). After the motor has slowed enough, the angle is detected (at a current zero crossing) and the controller goes to open loop without parking. Or, if a zero crossing cannot be found before the CsZcTimeout, it is interpreted as a non-rotating motor and a standard start sequence is initiated.
- If the motor is turning (fast enough) in the same direction as that desired, then the controller starts in closed loop mode and then accelerates the motor to the desired speed (green).
- If the motor is stopped (or turning very slowly) or has been braked from reverse, then a normal startup sequence is initiated with parking, open-loop and closed-loop.
- If the motor is spinning too fast in either direction, then the controller generates a fault (CsOverSpdFlt) and exits the catch-spin sequence. The forward and reverse threshold speeds are set by registers CatchMaxSpeed and BrakeMaxSpeed respectively.
- Another way to enter reverse catch spin is from the closed-loop state by setting MtrSeqCtrl = 8. As shown in Figure 8, the controller can go directly from closed-loop state to the brake state of the catch spin sequence, avoiding the need to fully stop and park the motor before changing direction.
- The no-parking reversal of the motor depends on using the braking currents to locate the rotor angle, which corresponds to a value of 0 in registers ThetaStart. However, the value of this register can be changed to tune the reliability of the reverse starting.

Catch-spin is an advanced feature, but various registers for catch spin can be configured using MCEWizard100. A more detailed description of catch spin can be found in Section 3.3.1.5 of the IRMCx100 Reference Manual.

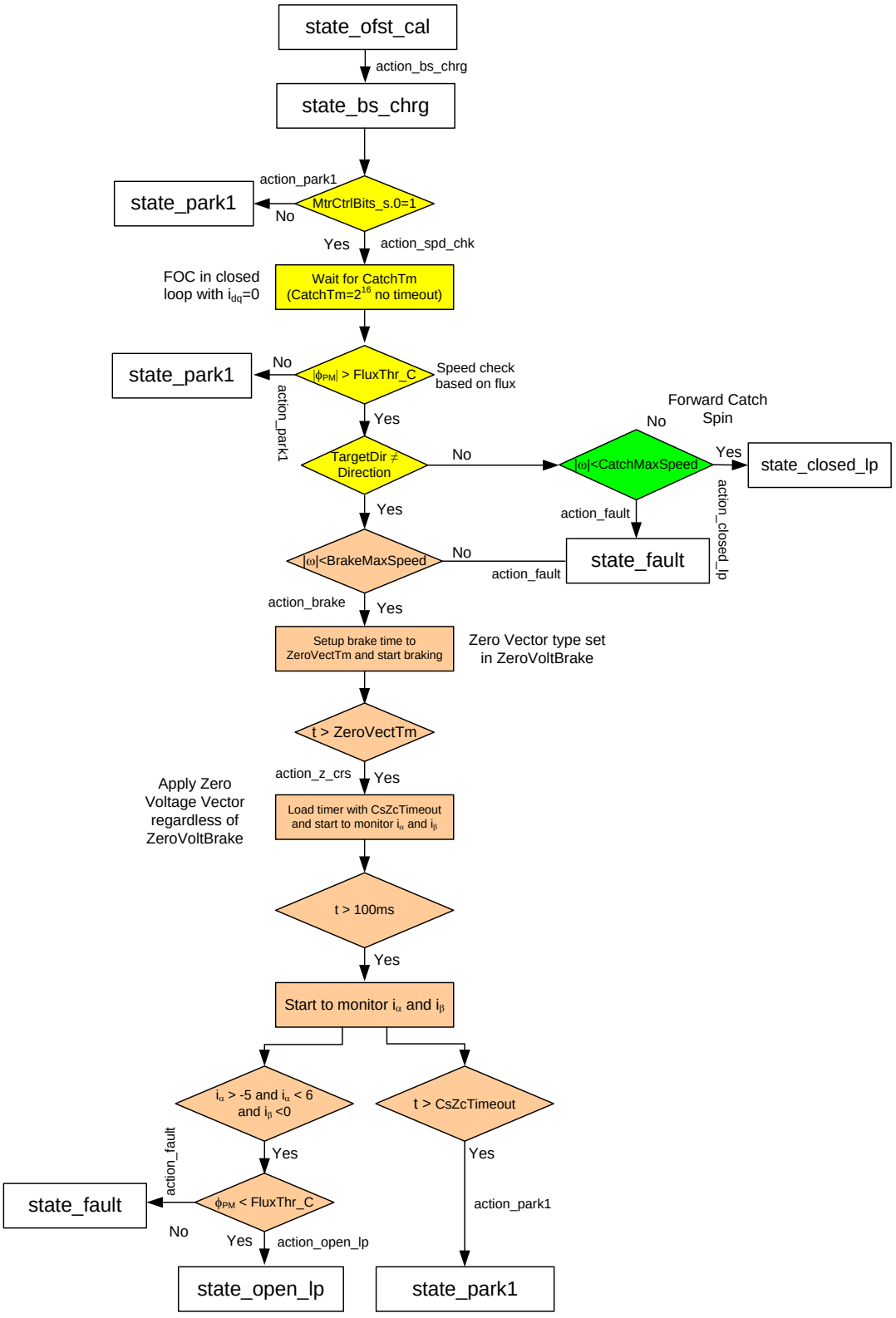


Figure 15—Catch Spin Start Sequence

2.3.4 Control Loop Structure & Tuning

There are three main control loops associated with IRMCx100 Series products. These control loops are the current control loop, speed control loop and field-weakening control loop. The following table summarizes the parameter dependence of each control loop.

Parameters	Current Controller	Speed Controller	Field-Weakening Controller
Motor Inductance	**X**		**X**
Motor Resistance	**X**		
Voltage constant (Ke)			**X**
Torque constant (Kt)		**X**	
System Inertia		**X**	

The speed loop is the outer control loop, determining the torque required based on the error between the reference speed and the speed feedback. The reference torque, which is really a (q-channel) current reference, feeds the current loop. The outputs of the current loop are the (q-channel and d-channel) voltage modulation commands, which are converted into the PWM gating times for the three phases during each PWM cycle. The field weakening loop supplies a d-channel current reference to the current loop.

2.3.4.1 Current Controller

The iMotion 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 16. The motor windings can be represented by a first order lag with a time constant $\tau = L/R$. This time constant is a function of the motor inductance and equivalent resistance (R = cable + winding). For a surface mounted permanent magnet 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 16, 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 17 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 (τ), 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 18. The equivalent transfer function of Figure 18 is a first order lag with time constant τ_c. By selecting an appropriate current regulator response (typically 0.5 to 1 msec, MCEWizard100 entry Current Regulator Bandwidth = $1/\tau_c$) 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.

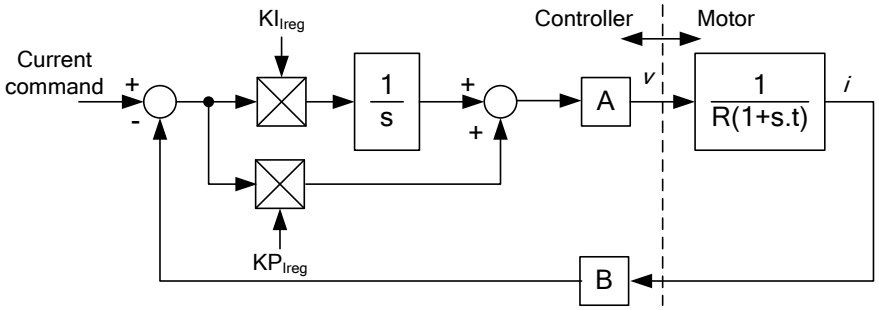


Figure 16—Current Controller Dynamics

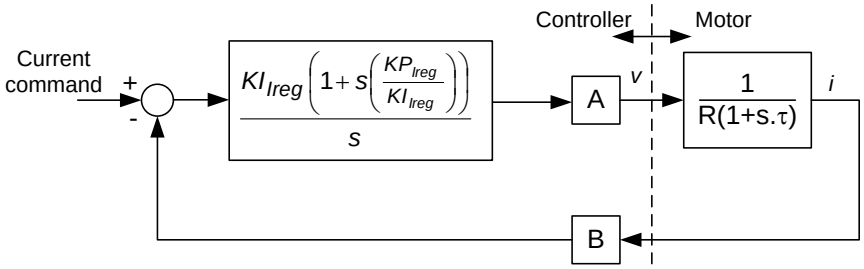


Figure 17—Pole Zero Cancellation

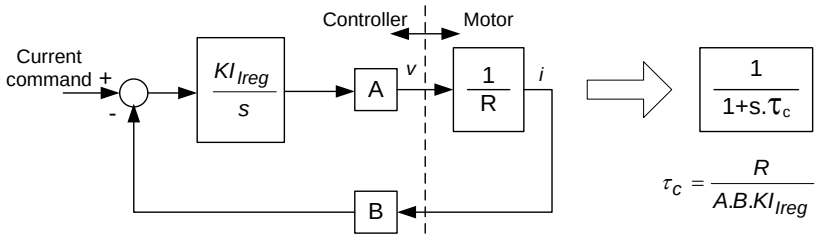


Figure 18—Simplified Current Control Dynamics Due to Pole Zero Cancellation

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 100% modulation the rms phase voltage is $V_{dc}/\sqrt{2}/\sqrt{3}$. The modulator produces 100% modulation for a digital input of 2355 while the forward rotation function has a gain of 1.646. Therefore, the current loop voltage scaling A is given by this equation:

$$A = \frac{\left(V_{dc} \big/ \sqrt{6}\right)(1.647)}{(2355)} \text{ Vrms}$$

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 register, **IfbkScl**, described in the IRMCx100 Reference Manual. However, MCEWizard100 calculates **IfbkScl** so that a count of 4095 is equivalent to the motor rated rms current. Therefore, the current loop feedback scaling is simply given by:

$$B = \frac{(4095)}{I_{RATED}} \text{ Arms}^{-1}$$

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:

$$KpIreg = \frac{L_q \cdot CurrentRegBW.2^{14}}{A \cdot B}$$

$$KxIreg = \frac{R \cdot CurrentRegBW \cdot T.2^{19}}{A \cdot B}$$

The following gain calculation illustrates a drive application setup with MCEWizard100 entries:

DC bus Voltage:		300V
Calculated voltage gain A:		0.0857 V
Rated motor current:		2.10 A
Calculated feedback gain B:		1950 A^{-1}
A_B product:	(A.B):	167.1 $V.A^{-1}$
PWM Switching Frequency:		10kHz
Calculated sampling time T:		10^{-4} s
Inductance Lq:		21mH
Inductance Ld:		21mH
Stator Resistance:		6.9 ohms/ph
Current Regulator Bandwidth:		1500 rad/sec

The current regulator gains are calculated as:

$$KpIreg = \frac{0.021 \cdot 1500 \cdot 2^{14}}{167.1} = 3089$$

$$KpIregD = \frac{0.021 \cdot 1500 \cdot 2^{14}}{167.1} = 3089$$

$$KxIreg = \frac{6.9 \cdot 1500 \cdot 10^{-4} \cdot 2^{19}}{167.1} = 3247$$

The current controller in the Sensorless FOC block module directly uses these gain values.

The sequencer provides a current loop diagnostic test function called "Current Regulator Diagnostic," which is entered into by setting the appropriate bits of MtrCtrlBits. This test provides the response of the current control loop and also the steady state accuracy. This diagnostic repeatedly provides a step current command as would be performed during parking. The current command level is set by the register ParkI.

Once the current regulator diagnostic is executed, the step current response can be observed from the trace function or current probes on the W-phase. It is recommended to use a current probe to observe the step current response. In this test, the rotor shaft should not move; if it does, it should be immobilized. Figure 19 shows the step current response (using a current probe) of the w-phase when the Current Reg diagnostic function is executed. In this figure, a 25% rated current step is commanded. The step level can be controlled by parameter ParkI (inside the Current Reg Diagnostic function). Figure 20 shows the expanded version of Figure 19. The measured current loop response is critically damped with a 0.65 msec time constant (0 – 63% of the final steady-state value), which is approximately equal to the anticipated current regulator bandwidth response (1/1500 = 0.667msec).

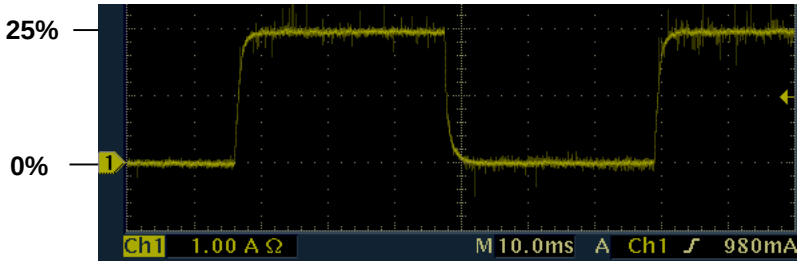


Figure 19—Step Current Response

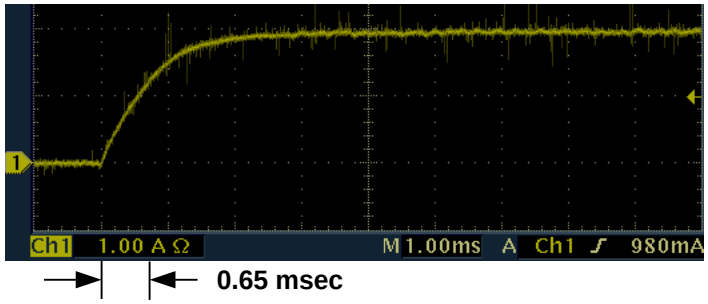


Figure 20—Step Current Response (Expanded Time Scale)

2.3.4.2 Speed Controller

After tuning the current controller, proceed to tuning the speed controller. The speed controller is the most outer-loop controller in the cascaded speed drive system, so the inner loop must be tuned first. Figure 21 shows the cascaded control dynamics of the speed control loop. In practice, the inner current loop has a much higher control bandwidth than the speed controller; therefore for speed control dynamic purposes, the inner current loop can be ignored as shown in Figure 22. Parameter M (Figure 22) relates the command current digital counts to the actual current in Amps. The motor mechanical dynamic is a first order function with mechanical time constant equal to J/F (Inertia/Friction). The pole-zero cancellation technique (outlined in Section 2.3.4.1) can be used to simplify tuning of the speed controller proportional and integral gains (KpSreg, KxSreg). In practice, information on mechanical friction (F) is difficult to obtain, therefore it is not modeled here. In addition, a temperature dependent friction characteristic is present in some applications. Therefore, manual speed tuning may be required to achieve optimal speed response. Some applications cannot tolerate high speed regulator bandwidth due to mechanical resonances present in the mechanical system.

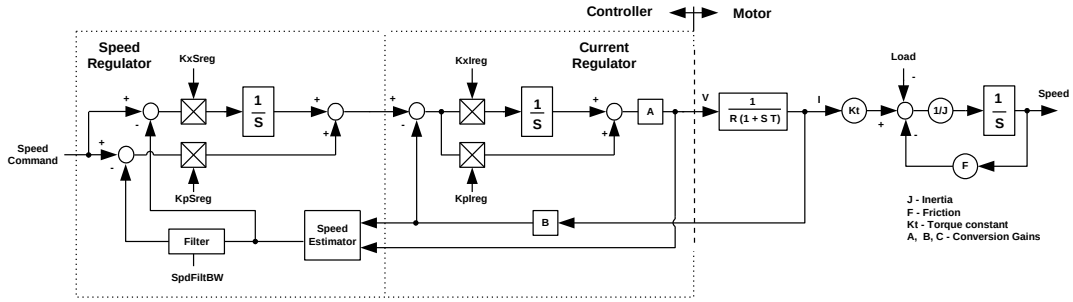


Figure 21—Cascaded Control Dynamical Model

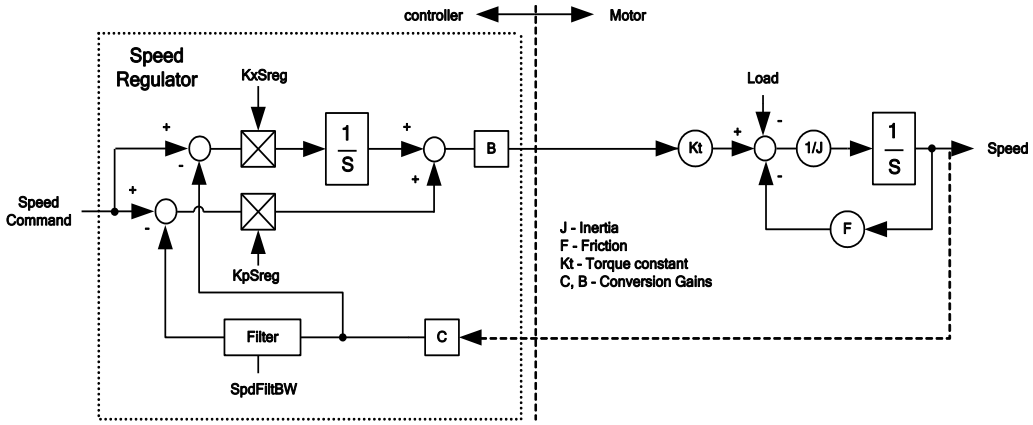


Figure 22—Simplified Speed Control Loop Dynamics

As mentioned earlier, information on mechanical parameters (*e.g.*, Inertia) may be inaccurate, causing MCEWizard100 to output less than optimal gains for the controller. Manual tuning of the speed regulator can be used to optimize the speed performance. Figure 23 shows the speed response (trace buffer speed feedback signal) of a high inertia fan under speed ramping. As can be seen, the speed response shows oscillatory behavior due to non-optimized gain values.

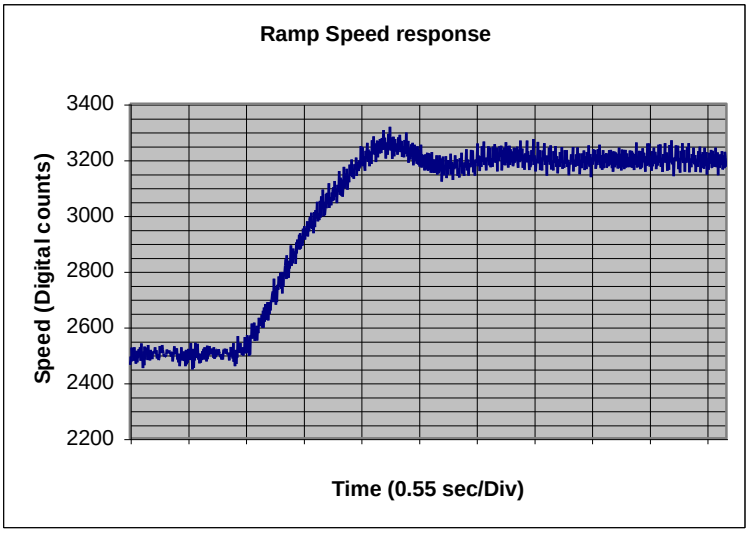


Figure 23—Ramp Speed Response

There are many different approaches to tuning a PI regulator for various applications. The following steps provide an example guide line of speed regulator tuning for fan applications.

1) **Tuning of KpSreg**. Run the drive at a convenient speed; say 30% of the rated rpm. Perform a small step speed change (step size of 5 - 10% of rated speed) with **KxSreg set to zero**. The step response can be achieved by setting a fast speed ramp in MCEWizard100. Under such conditions, the first order speed response is expected as shown in Figure 24. This figure shows the speed responses using three different proportional gains (KpSreg = Kp1, Kp2, Kp3).

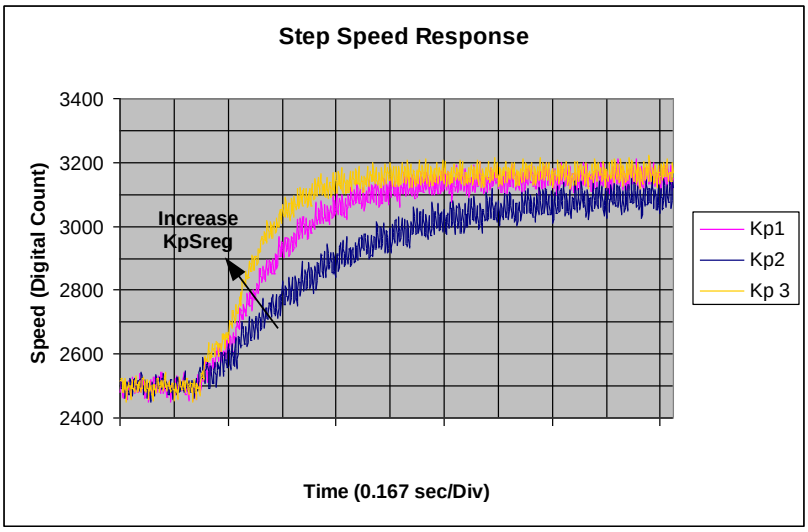


Figure 24—Step Speed Response under Different KpSreg Gains

Adjust KpSreg until the desired transient response (Speed regulator bandwidth) is obtained. For this fan application with a high inertia to friction ratio, Kp3 is selected to yield approximately 0.2 sec first order time constant.

2) **Tuning of KxSreg**. After the desired proportional gain (KpSreg) is selected (Step 1), resume the desired ramp rate and speed regulator integral gain (KxSreg). Under such circumstance (with KpSreg = Kp3 and KxSreg = Kx1), issue a ramp speed command over the same speed range as illustrated in Step 1. Figure 25 shows the ramp speed responses under three different integral gains (Kx1, Kx2 and Kx3). The response with the original integral gain (Kx1) exhibits oscillatory behavior. The integral gain is being reduced (Kx2 and Kx3) just enough to remove speed oscillation. For this fan application, the response obtained is acceptable with KxSreg = Kx3.

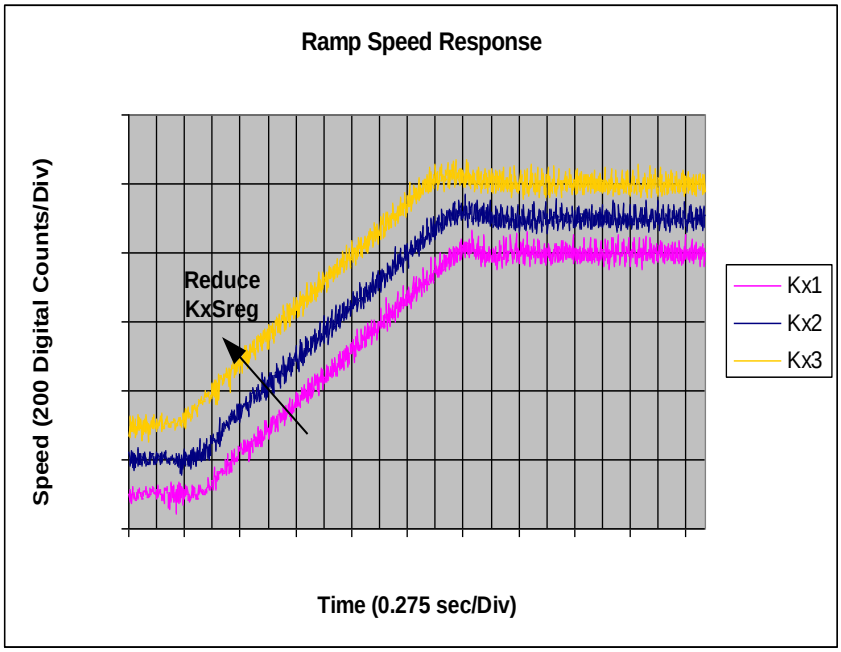


Figure 25—Ramp Speed Response under Different KxSreg Gains

3) Figure 26 shows ramp speed response with non-optimized (KpSreg = Kp1, KxSreg = Kx1) and optimized (KpSreg = Kp3, KxSreg = Kx3) speed regulator gains. A tighter control response is exhibited due to the gain optimization.

4) It may be noticed that there is still slight overshoot on the optimized Ramp Speed response (Figure 26). Most applications can tolerate a slight overshoot (<10%).

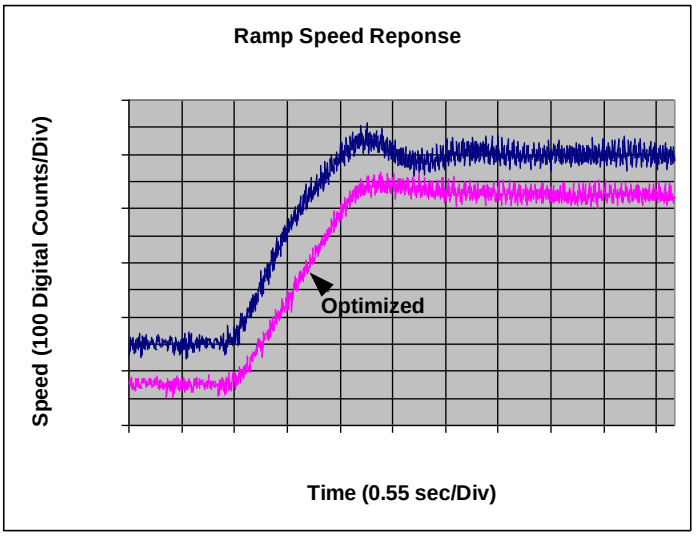


Figure 26—Comparison of Optimized and Non-optimized Speed Response

Increasing KpSreg or reducing the speed ramp rate as shown in Figure 27 can further reduce speed overshoot. It is recommended to keep overshoot to the minimal possible for applications (*e.g.*, washer spin mode), which require a Field-Weakening range of more than 1.5 (150% of the rated speed).

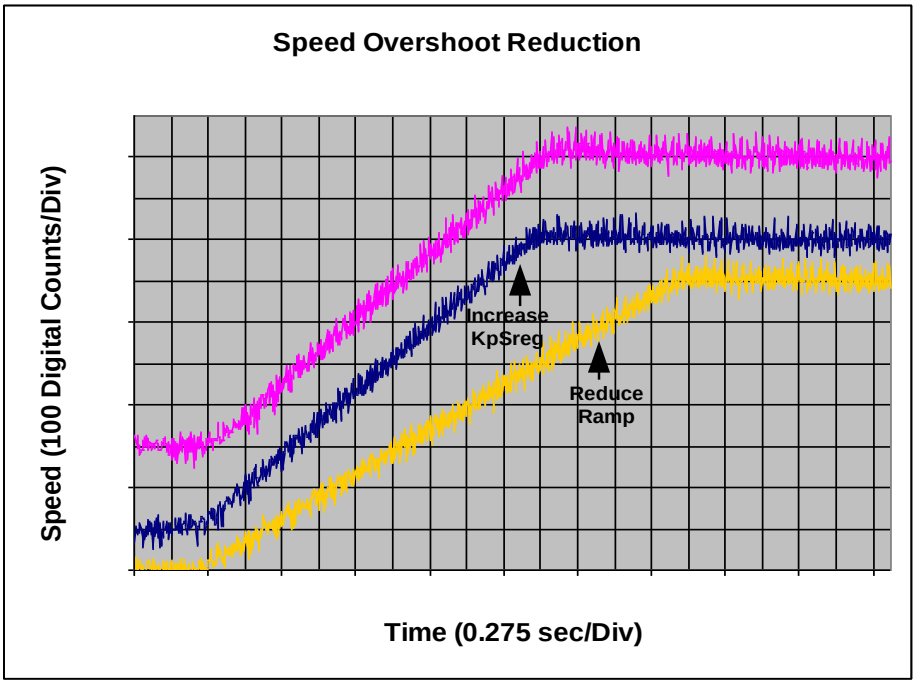


Figure 27—Speed Overshoot Reduction

2.3.4.3 Interior Permanent Magnet Motor Control

The motor torque developed by a permanent magnet motor is given by:

$$\text{Torque} = \frac{P}{2} \cdot \underbrace{\left(\text{FluxM} \cdot I_q\right)}_{\text{Cylindrical}} + \underbrace{\left(L_d - L_q\right) \cdot I_d \cdot I_q}_{\text{Reluctance}}$$

Where:

P	number of rotor poles
L_d, L_q	d and q-axis inductance (d axis aligns to rotor magnet)
I_d, I_q	d and q-axis current components.
FluxM	Flux linkage of the permanent magnets

There are two torque components associated with the motor torque equation. The first component (cylindrical torque) is due to interaction between the rotor magnet flux and the stator q-axis current. The second component (reluctance torque) is due to the motor saliency (difference in d and q inductance). This saliency term is negligible (Ld = Lq) in Surface Mounted Permanent magnet (SPM) motors. In the case of an Interior Permanent Magnet Motor (IPM) where Lq is not equal to Ld, the torque per ampere rating is boosted by the saliency torque term. In motoring operation, a negative Id injection will contribute to the increase in reluctance torque.

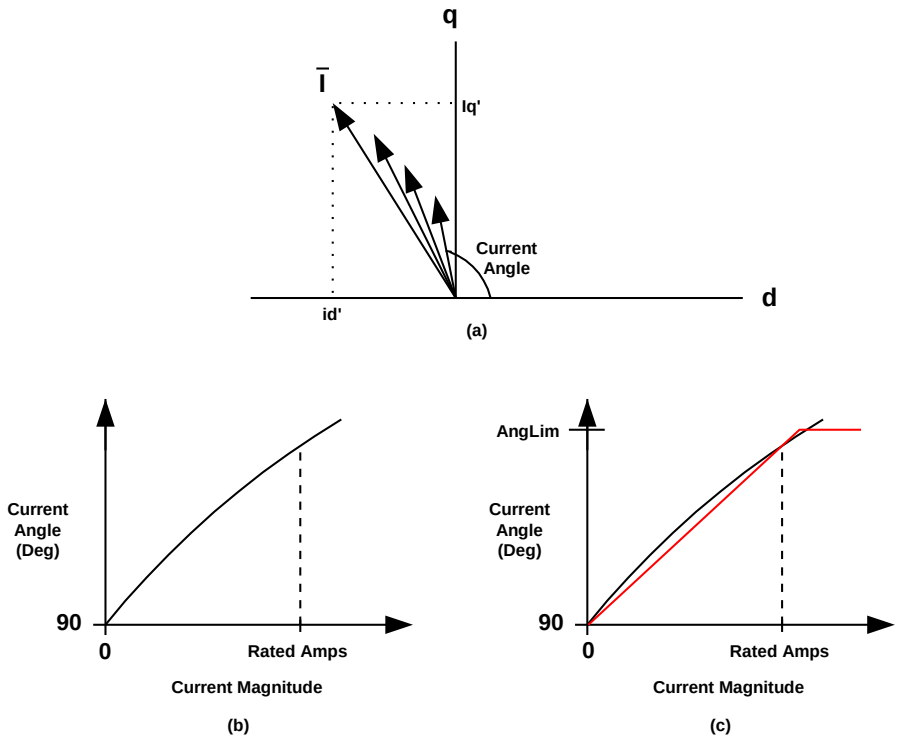


Figure 28—Current Angle at Maximum Torque per Ampere

Figure 28a shows the current vector trajectory for optimal torque per ampere generation of an IPM motor. As the current magnitude increases, the current angle advancement also increases, which indicates an increase in negative d-axis current demand. The required current angle for optimal torque per ampere generation is depicted in Figure 28b. In the iMotion control IC, this optimal current characteristic is approximated by a linear fit as shown in Figure 28b. Two parameters (AngDel and AngLim) are used to characterize the behavior of the optimal current angle for generating maximum torque per ampere. Parameter AngDel fixes the slope of the line and parameter AngLim limits the maximum allowable angle advancement. MCEWizard100 computes AngDel with two points (zero and rated current point). The implementation of this linear approximation and the calculation of the commanded d-q current are shown in Figure 29.

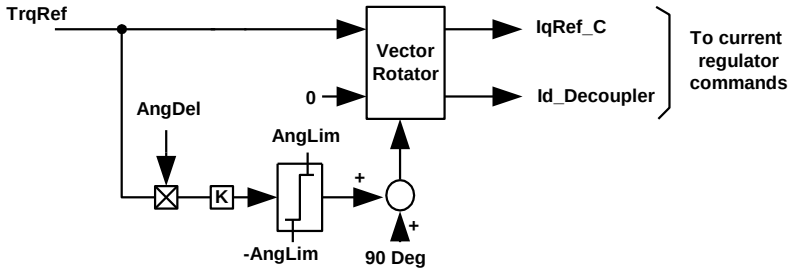


Figure 29—Current Decoupler for Optimal Torque per Ampere Operation

2.3.4.4 Field-Weakening Controller

Field weakening is required to extend the motor operating point beyond the rated speed. The back EMF (BEMF) of a motor increases with speed up to the dc bus voltage. A further increase in the motor speed requires flux weakening to maintain the motor terminal voltage at its maximum possible level as shown in Figure 30.

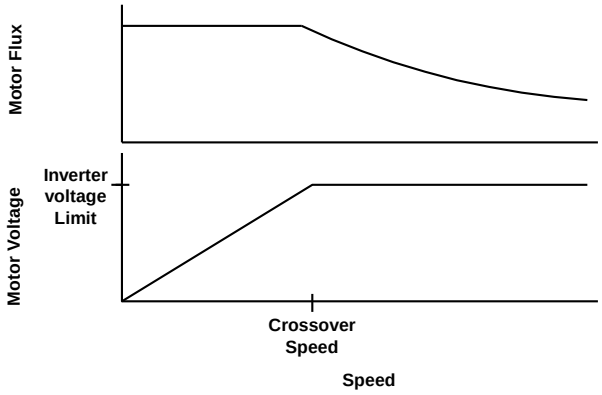


Figure 30—Field-Weakening Characteristics

Figure 31 shows a block representation of the Field-Weakening Controller. The output of the controller (Fwk_Id) is the d-axis current component, which opposes the rotor magnet flux (Flux). By injecting a negative d-axis current, the resultant flux can be reduced and hence the motor voltage can be limited to stay within the ceiling voltage of the inverter output. The control loop gain increases with motor frequency as shown in Figure 31. Inside the Field-Weakening controller, gain modulation is used to decouple the variation of loop gain due to motor frequency.

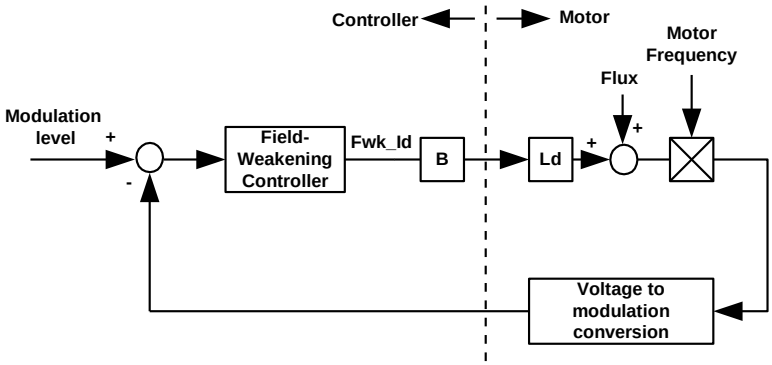


Figure 31—Field-Weakening Control Model

Figure 32 shows the Field-weakening controller. As can be seen from this figure, when the modulation exceeds a prescribed level, specified by FwkLvl, a negative d-axis current (Fwk_Id) is commanded and reduces the main flux. The Field-Weakening controller acts as a modulation index limiter. The gain modulation block serves to compensate the increase in loop gain due to an increase in the motor frequency, as mentioned earlier.

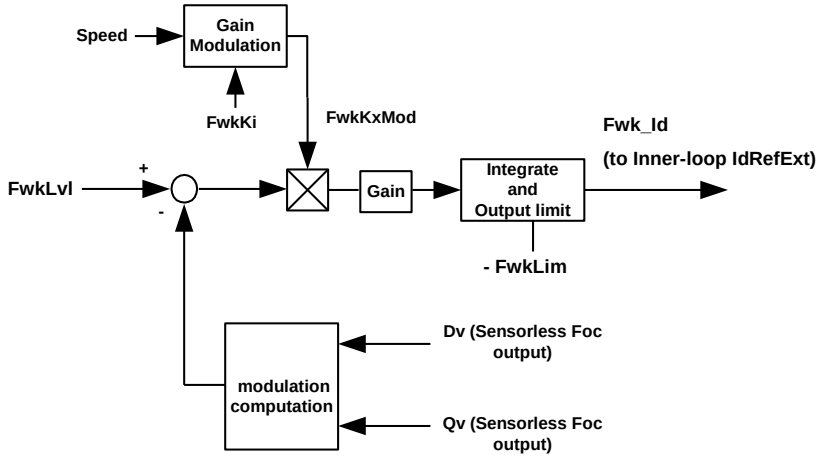


Figure 32—Field-Weakening Controller Block

With gain modulation incorporated, the control loop gain is decoupled from the motor frequency. Figure 33 shows a simplified representation of the Field-weakening controller. The equivalent transfer function of Figure 33 is a first order lag system. Parameter M in Figure 33 is a function of motor inductance, nominal dc bus voltage and the motor crossover frequency (function of motor Ke). MCEWizard100 sets the field-weakening controller gain (FwkKi) based on parameter M and a prescribed field-weakening loop response (0.25 sec).

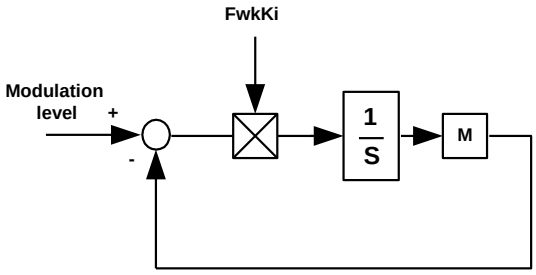


Figure 33—Simplified Field-Weakening Control Model

The response of the Field-weakening loop can be observed from the command d-axis current (using the MCEDesigner trace function). Under light Field-weakening conditions (-10% rated motor current on field-weakening controller output current), a step change (-2%) in the modulation limit level (FwkLvl) is issued. When the modulation limit reduces, the Field-Weakening controller increases the d-axis motor current (with negative sign) in order to reduce the motor flux and satisfy the modulation limit (voltage limit). Figure 34 shows a step change in FwkLvl and the response of the command d-axis current. The first order response is exemplified in Figure 34. The tuning of the Field-Weakening controller is straightforward since it only involves one controller gain. The response of the Field-Weakening controller should be high enough to catch up with speed changes. In practice, most appliance applications do not require high dynamic speed changes in the Field-Weakening region. Therefore the response of Field-weakening can be relaxed (typically: 0.1 to 0.4 sec response time. The MCEWizard100 is preset to 0.25sec).

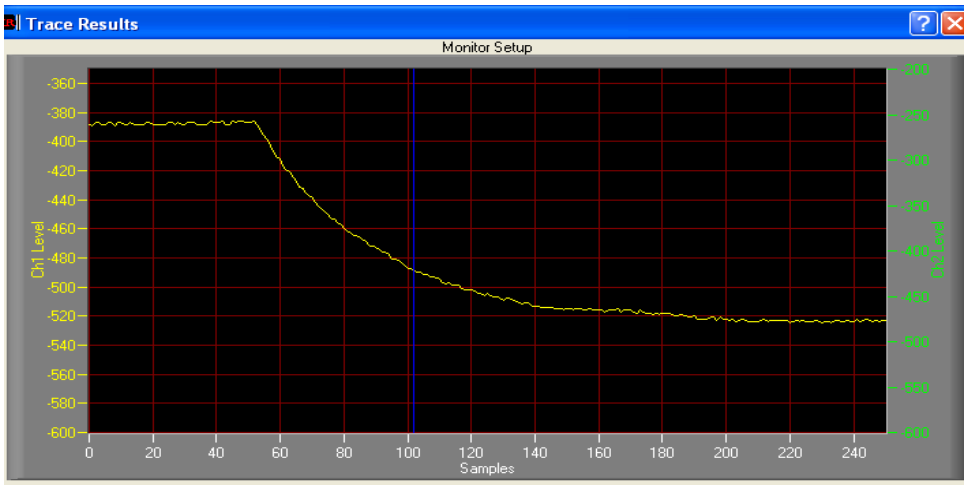


Figure 34—Field-Weakening Control Responses

Vertical: Id command (digital counts) Horizontal: Time (0.1 sec/ div)

Some control hints for high speed operation:

- Reduce the speed loop gain.
- The maximum torque may not be available during field weakening.
- Also, try reducing the values of register MotorLim to increase the voltage available for BEMF reduction.
- Reducing the FWLevel register will begin the field weakening earlier to provide more voltage margin for control.

2.3.5 Braking the Motor

Naturally, in almost all applications, the motor will need to be stopped as well as started. The simplest way to stop the motor is to simply stop the drive. The PWM switching will stop and the motor terminals will no longer be energized. The rotor will coast down to a stop.

However, many applications will require a less passive and more effective braking method. One such method is to park the motor (DC injection). This may be useful in situations where the motor is moving slowly and needs to be started again immediately. This section describes two more braking methods and then concludes by describing a fault condition which protects the hardware by braking.

2.3.5.1 Regenerative Braking

The velocity controller is capable of applying a torque in either the same or opposite direction of the motor motion. This reverse torque can be used to slow the rotor in a technique known as Regenerative Braking.

In normal operation, a current is applied to the motor winding. During active braking, the phase of the current is inverted, so that current is now *removed* from the motor windings. The rotor experiences a torque, proportional to the current, opposite the direction of motion, and the DC bus becomes charged from the motor current. Be careful of overcharging the DC bus during active braking. Be sure to set the overvoltage fault level at a safe value. One way to dissipate the energy is to use a brake resistor, which discharges the DC bus in an overvoltage situation.

Implementing active braking is relatively simple. Change the “Regen Limit” input of MCEWizard100 to a non-zero number and then set a low speed reference. The Regen Limit is defined as a percentage of the rated current, so a larger Regen Limit will result in a higher braking current and a higher reverse torque. The (negative of) Regen Limit acts as a lower limit

to the Torque Reference output of the speed loop. The result is that the Torque Reference can be inverted, indicating torque in the opposite direction of the motion and resulting in an inversion of the motor current phase. The Torque Reference can become inverted if the Speed Feedback is greater than the Speed Reference. The speed loop is covered in detail in Section 3.2.

2.3.5.2 Zero Vector Braking

In Zero Vector Braking, energy is removed from the mechanical system by the back EMF of the rotor and is dissipated by the resistance of the motor windings. At an electrical frequency where the winding impedance is dominated by the inductance, zero vector braking will produce a constant current in the motor windings. When using this braking method, verify that the motor can withstand the short circuit current.

There are two methods implemented in the IRMCx100 for this type of braking: Zero Voltage and Zero Vector. In Zero Voltage Braking, the three motor terminals are shorted together by alternately turning on all the low-side and then all high-side transistors of the motor drive inverter, as shown in Figure 35 below.

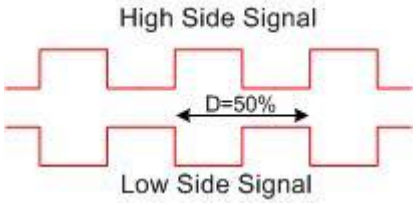


Figure 35—Zero Voltage Braking

Note that the minimum pulse width and dead time are applied in this braking mode. The advantage of this method is that the heating of the power switches is shared between the high-side and low-side. Another advantage of Zero Voltage Braking is that the motor currents can still be observed—do not change the minimum pulse width in this case. **The disadvantage is that, due to dead time and the minimum pulse width, the DC bus can be charged up if the motor is spinning at field-weakening speeds.** Set the minimum pulse width (TCntMin3Ph, TCntMin2Ph) to zero to minimize this effect.

In Zero Vector, the low-side power switches are turned on continuously. Since there is no deadtime necessary, there is no chance of charging the DC bus.

To request Zero Voltage Braking, write ‘1’ to register Zero_Vec_Req. To brake using the Zero Vector, trigger a Critical Overvoltage Fault (see Section 2.3.5.3) by writing a low value to register CriticalOvLevel.

In a reverse Catch-Spin situation, the user may select the type of braking used by writing to the register ZeroVoltBrake.

2.3.5.3 Critical Over Voltage Protection

In order to achieve high-speed operation under limited voltage capability, the motor flux is suppressed by injection of a negative d-axis current (Field-weakening) to the motor. In case of an inverter shunt down at high speeds, all inverter devices will be disabled and the negative d-axis field forcing will be lost. Under such circumstances, the motor flux will resume and the motor voltage will build up according to the motor Ke as shown in Figure 36. This is a critical over voltage condition in which the motor BEMF builds up and charges the dc bus capacitor voltage to an exceedingly large value.

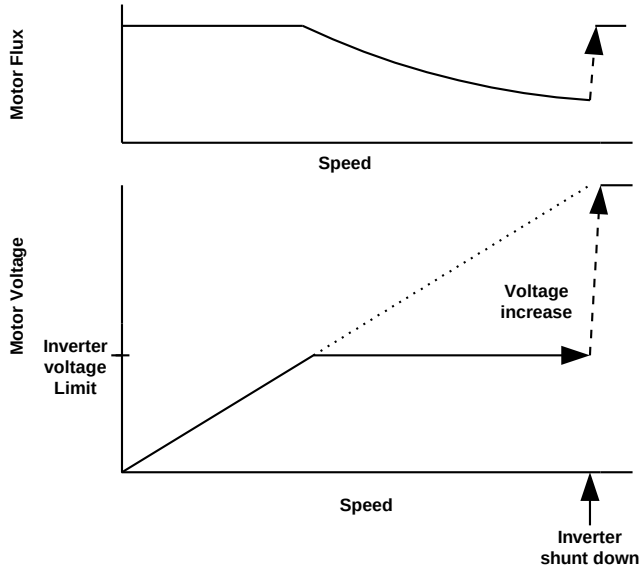


Figure 36—Inverter Shunt Down during Field-Weakening

In the iMotion control IC (IRMCx100 series), the critical over voltage protection is implemented in the MCE firmware. The critical over voltage condition is detected by comparing the dc bus voltage feedback to a configurable voltage level. When a critical over voltage is detected a zero vector (low side devices turn-on) PWM state is entered, independent of any condition (including faults). The use of a zero vector forces short circuit to the motor terminal and hence prohibits charging of dc bus capacitors. Figure 37 illustrates the critical over voltage condition and the engagement of zero vector protection. Upon application of the zero vector, the motor current will circulate within the motor windings and the rotational energy of the motor will be dissipated inside the motor (copper and core losses). The inverter is held in the zero vector state until the Critical Overvoltage Fault is cleared.

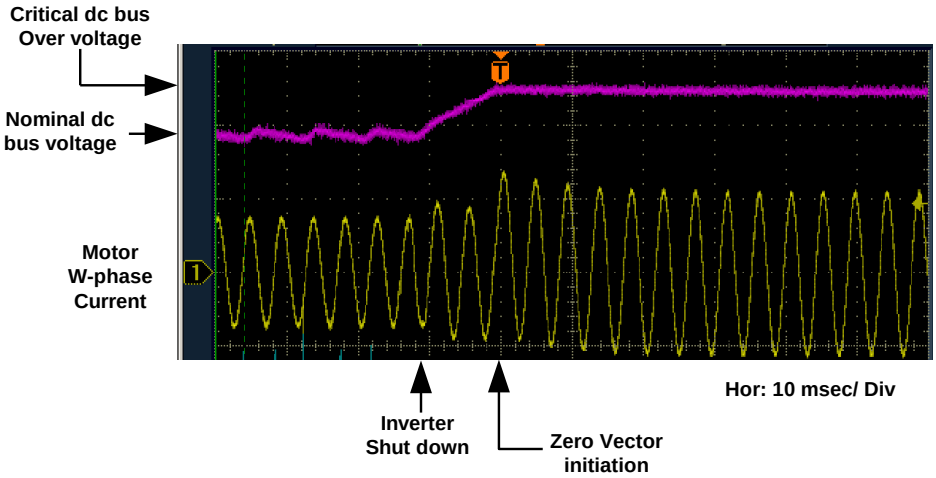


Figure 37—Critical DC bus Over Voltage
(top: dc bus voltage, bottom: motor current)

3 MCE Program Customization

This section will begin by describing the Motion Control Engine (MCE), which is the processor where the main motor control loops and protection logic are implemented. The motor control algorithm is realized through a combination of user configurable software elements (*i.e.*, Speed loop) and fixed firmware elements (Current loop) executed within the MCE on a PWM synchronous basis. The software control sequence (MCE program) is available for modification by the designer for application-specific functionality, using MATLAB/Simulink as a graphical editing tool. Section 3.2 explains the factory-installed MCE program. The following section explains the general properties of an MCE design created in MATLAB/Simulink. Section 3.4 gives instructions on how to configure the MATLAB software and modify, compile and download the new MCE program. Section 3.5 concludes with two examples of MCE program modifications: "Torque Mode" and "Limiting the Speed Feedback Input Variance".

The MCE program development environment consists of the following components:

- A library of graphically-represented Simulink control blocks to be used in the design of a motor control system.
- The MCE compiler, a web-based tool which analyzes the Simulink design and generates a corresponding program file that is executed by the MCE processor on the IRMCx100 IC.
- MCEDesigner, which provides a graphical user interface to the IRCMx100 to allow download of the MCE executable file, control of MCE operation, and analysis of system function and performance.

The MCE development tools software distribution is organized beneath a main directory named MCE Compiler, within the iMotion directory (normally found at C:\Program Files\iMotion\MCE Compiler). The main MCE Compiler directory contains subdirectories "Simulink Library100", which contains the Simulink library block definitions; and "Matlab100", which contains the MATLAB scripts that implement the graphical blocks.

The MCE development tools are designed to operate with MATLAB version 7.4 and Simulink version 6.6. They should operate properly with newer versions, but may not function correctly with older versions of MATLAB and Simulink.

Note: MCEWizard100 may produce incorrect register values if the MCE program is changed. The designer should carefully consider whether the MCEWizard100 output is still correct after modification of the MCE program.

3.1 The Motion Control Engine

The Motion Control Engine carries out the main motor control computations within the IRMCx100 Series IC. Figure 38 is a block diagram of the MCE showing the main components and main communication bus.

The primary component of the MCE is the **MCE processor** which runs the MCE firmware and MATLAB design, including sequencing some hardware computations according to the MCE program instructions.

The **MCE firmware** and **MCE program** are contained in a configurable block of RAM. The program is loaded to the RAM during the boot process of the MCE. For more information about the boot process, see the Reference Manual.

In addition to the MCE program RAM, there is a **shared** (data) **RAM** which is accessible to the 8051 processor. The registers contained in this section of RAM are used by the 8051 processor to control the settings and actions of the MCE. MCEDesigner sends instructions to the 8051 processor, which then starts, stops and changes the speed of the motor by writing to registers within the shared RAM.

Designed to facilitate and speed the motor control computations, the **MCE firmware** is a set of fixed software modules to do specific functions. Included in these modules are blocks which regulate the current, produce the gate drive PWM signals for the motor drive and reconstruct the motor phase currents from the single shunt current sampling. Also, over-current shut down signals (GATEKILL) go directly to the MCE to get a fast shut down of the drive. Closely integrated with the MCE is the **Analog Signal Engine**, which contains op-amps, sample-and-hold circuitry, analog multiplexing and an A/D converter.

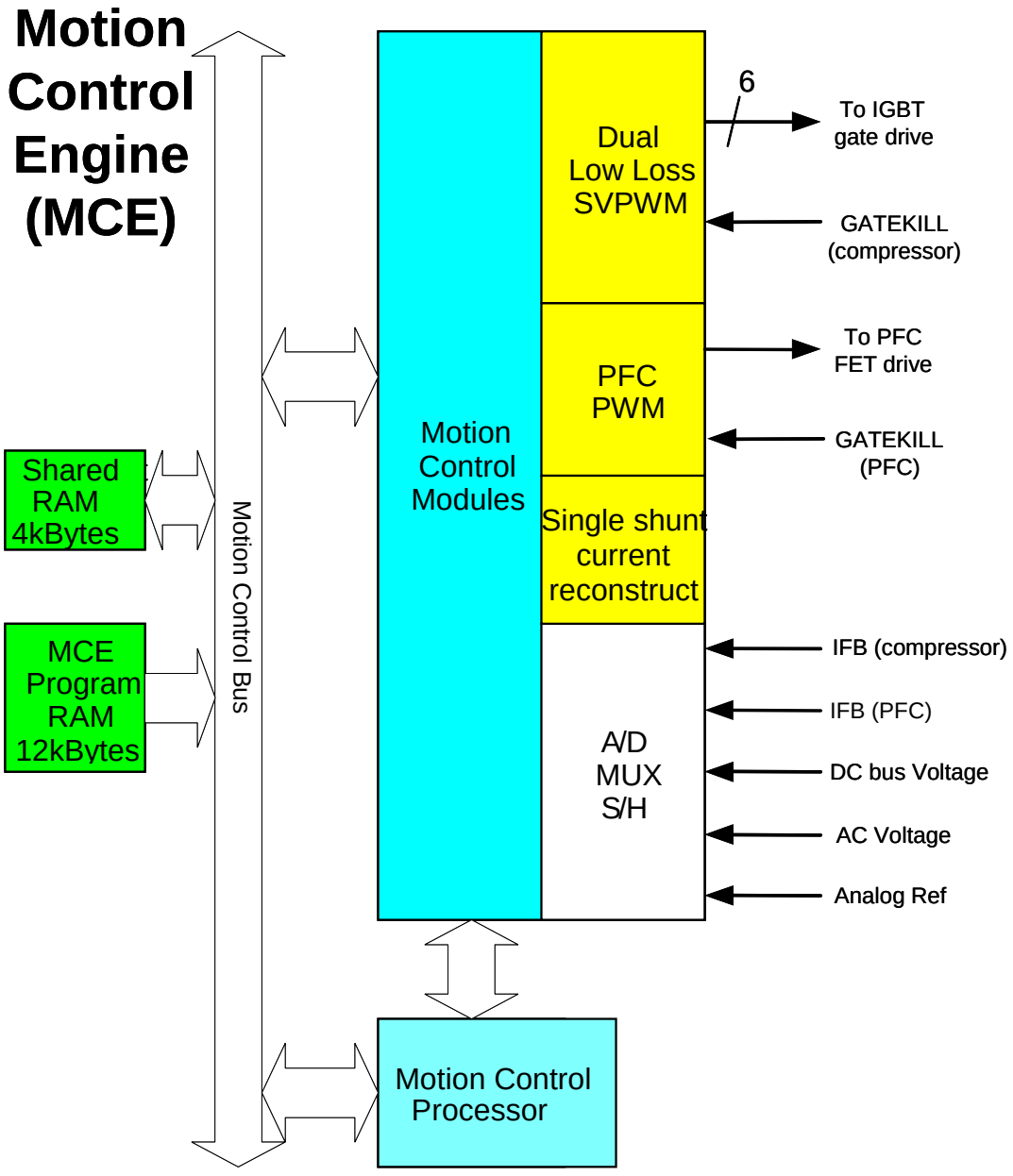


Figure 38—Block Diagram of the MCE

3.2 IR Standard MCE Program

3.2.1 Block Diagram

Figure 40 is a block diagram of the MCE design of the motor control loop. The Simulink model file (.mdl) and a PDF file of the block diagram can be found within the reference design kit installation in a sub-folder specific to the kit type (normally \My Documents\iMotion\IRMCS1271 or \My Documents\iMotion\IRMCS1043). The block diagram has several elements:

- Motion Peripherals—The yellow colored blocks represent the Motion Peripherals, a subset of the Motion Control Modules. These are controlled by a large number of registers, which are represented as input and output signals of the yellow blocks. The yellow block registers provide the interface from the MCE program and MCE firmware.
- Other Hardware and Software Blocks—The other colored blocks in the diagram are I/O blocks or software and hardware elements which are performed (or called) by the MCE processor.
- Standard Simulink library components—The MCE Compiler recognizes a subset of the standard Simulink library components.

Section 3.3 describes these elements of the block diagram in more detail.

3.2.2 Motor Speed control Loop MCE Program

The primary control feature of the MCE program is the *speed loop*, introduced schematically in Section 2.3.4.2. The block diagram contains the full speed loop control with protections, limits and other logic. To follow the speed loop, begin by locating the TargetSpeed register, located near the upper left corner of the diagram, as shown in Figure 39. After minimum speed protection and sign conversion logic (based on direction of rotation), the TargetSpeed becomes the input to the Speed Ramp block. The other inputs to this block (AccelRate, DecelRate & RampScaler) control the ramp rate of SpdRef (reference speed). The input to the PI (proportional + integral) block is the difference of SpdRef and SpdFbk. SpdFbk is the actual motor speed, which is computed in the FOC motion peripheral block and then scaled in the block diagram. The output of the PI is limited for protection, and then fed back to the FOC block as the TrqRef, closing the loop. The TrqRef (torque reference) actually serves as the current command to the controller, because torque is assumed to be proportional to current.

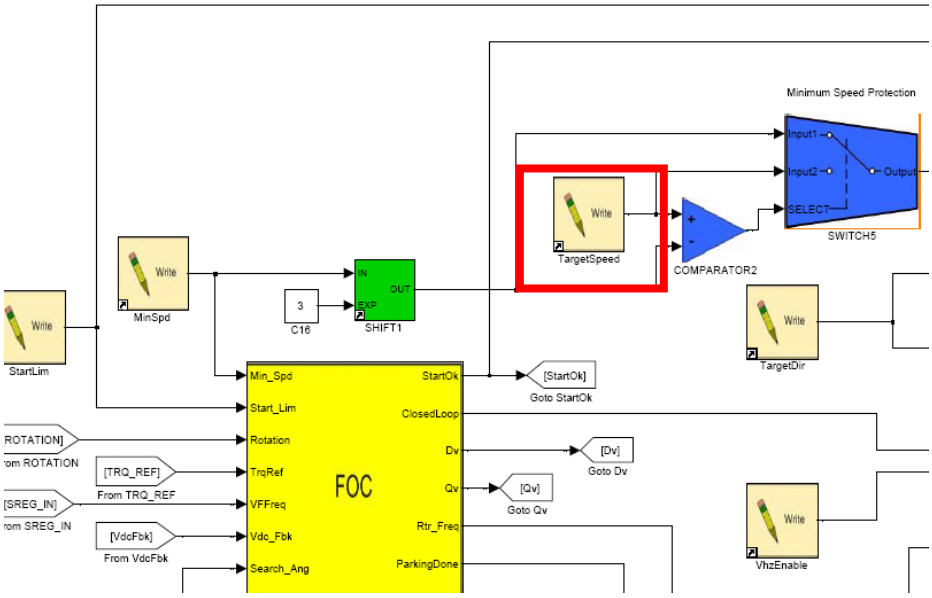


Figure 39—TargetSpeed Register

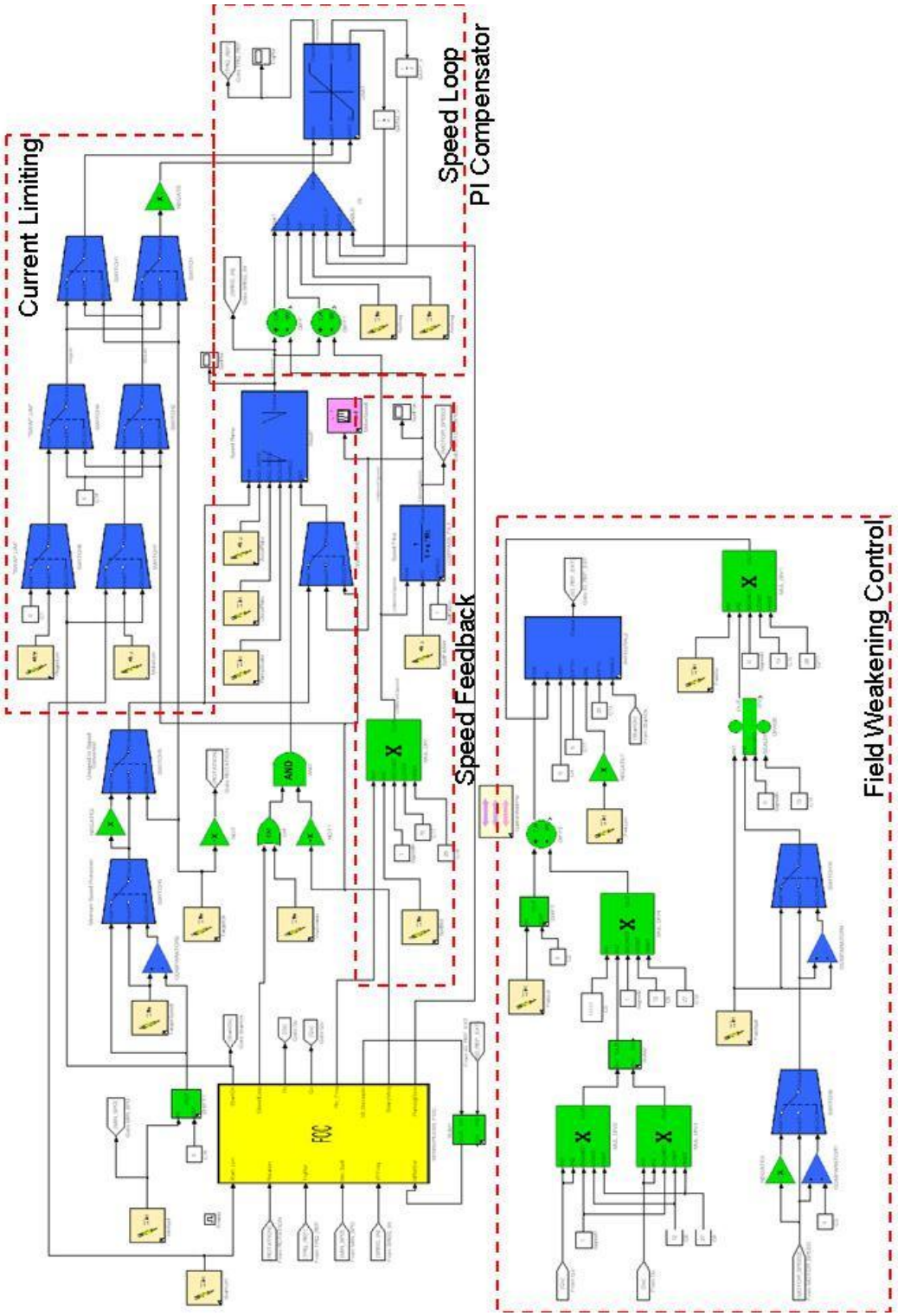


Figure 40—MCE Simulink Design of the Motor Control Loop

3.2.3 Other Features of the Speed Loop MCE Program

Here are descriptions of the other control features of the MCE program, as shown in Figure 40.

- The FOC block contains the current loop (described in Section 2.3.4.1) and the angle estimator. This hardware block also contains reluctance torque control for IPMs (Section 2.3.4.3).
- The SVPWM block is another firmware block which converts the voltage modulation command to PWM gating signals for the 3-phase inverter. The default source of the voltage modulation command is the FOC block, connected together in the MCE firmware.
- The Field Weakening Control logic implements field weakening control as described in Section 2.3.4.4.

Two of the Motion Peripherals, the FOC block and the SVPWM block, are particularly important to the motor control. Each one has a large number of input and output registers, which the designer may add to the block diagram as required (Section 3.4.2.1). Complete block diagrams and descriptions of the input and output registers can be found in the Reference Manual. It is recommended that the designer become familiar with these blocks.

Note that the Speed Loop is slightly different for IRMCx143 where there is a speed reduction function which is designed to limit the output power. See Section 5.1.2 for more information.

3.2.4 Input and Output Registers of the Speed Loop

Torque Limit Registers: **StartLim, RegenLim, MotorLim**
Scaling or Notation: Range: 0 – 4095
Scaling: Current (A) = [StartLim] * I_{rated}/4095
Description: These registers limit the value of TorqueRef before it becomes the (torque-producing) current command to the FOC block.
StartLim applies only during the open-loop stage of the start-up sequence, while **MotorLim** applies during normal closed-loop operation.
A non-zero **RegenLim** allows the TorqueRef variable to become negative, resulting in active braking (see Section 2.3.5.1). These registers are configured by MCEWizard100.

Speed Control Registers:
TargetSpeed
Scaling or Notation: Range: 0 – 16383
Scaling: Scaling: Speed (RPM) = TargetSpeed * (Maximum motor speed) / 16383
Description: This register sets the target motor speed and is the value to which the speed command will ramp. The rate at which the speed command ramps to TargetSpeed is set by the Ramp Rate Registers.
TargetDir
Scaling or Notation: Range: 0 -1
Description: Set the motor direction with this register. A value of 0 will result in a negative speed, and a value of 1 will result in a positive speed.
MinSpd
Scaling or Notation: Range: 0 – 2048
Scaling: Speed (RPM) = MinSpeed * (Maximum motor speed) / 2048
Description: This register sets the minimum motor speed. This register is configured by MCEWizard100.

Speed Regulator Registers: **KpSreg, KxSreg**
Scaling or Notation: Range: 0 – 32767

Description: These registers set the speed regulator PI block proportional (KpSreg) and integral (KxSreg) gains. The MCEWizard100 calculates values for these registers. For more detail on the speed regulator tuning, see Section 2.3.4.2.

Speed Feedback Registers:
SpdScl
Description: The variable Rtr_Freq, an output of the FOC block, is scaled to the same scaling as TargetSpeed using register SpdScl. For details about how this register is calculated by MCEWizard100, see Section 2.2.3.1.

SpdFiltBW
Scaling or Notation: Range: 0 – 8192
Scaling: Cutoff Freq (Hz) = PWMFreq * SpdFiltBW / 8192
Description: This register sets the cutoff frequency of the digital lowpass filter for the speed feedback. To send the signal through unfiltered, set SpdFiltBW to 8192.

Field Weakening Registers: **FwkLvl, FwkLim, FwkSpd, FwkKx**
Description: These registers control the generation of the field-weakening current command and are set by MCEWizard100. More information on the calculation of these registers can be found in Section 2.3.4.4.

Ramp Rate Registers: **RampScaler, AccelRate, DecelRate**
Scaling or Notation: Range: 0 – 31 (RampScaler); 0 – 32767 (AccelRate, DecelRate)
Scaling: Motor Acceleration Rate (RPM/s) = (Maximum Motor Speed / 16383) * ([AccelRate] / 2^RampScaler) * PWM Frequency
Description: The ramp rate of the speed command toward the target speed is controlled by these registers. Do not change the value of RampScaler while the motor is running; it can cause the speed command to change abruptly. These registers are configured by MCEWizard100.

Miscellaneous Control Registers: **VhzEnable**
Description: This register is a logic input (0 or 1) for the speed loop when the V/Hz diagnostic mode is used. This register alone does not enable the V/Hz diagnostic (see Section 2.2.4.1) .

Read Registers: **MotorSpeed**
Scaling or Notation: Range: 0 – 16383
Scaling: Speed (RPM) = TargetSpeed * (Maximum motor speed) / 16383
Description: This register gives the filtered motor speed.

Traceable Parameters:
SpdFbk, SpdRef
Scaling or Notation: Range: 0 – 16383
Scaling: Speed (RPM) = TargetSpeed * (Maximum motor speed) / 16383
Description: These are the measured and commanded speeds, respectively.

TrqRef
Scaling or Notation: Range: 0 – 4095
Scaling: Current (A) = TrqRef * I_{rated} / 4095

Description: This is torque command, which is an input to the FOC block. This becomes the torque current command.

3.3 Simulink MCE Design Components

This section describes the components of an MCE Simulink design. Most of your design components will be taken from the MCE library, but some components of the standard Simulink library are also used.

3.3.1 MCE Design Hierarchical Format

This section describes the hierarchical structure of a complete MCE system and provides instructions for creating a new MCE model template

The MCE design hierarchy has the structure shown in Figure 41.

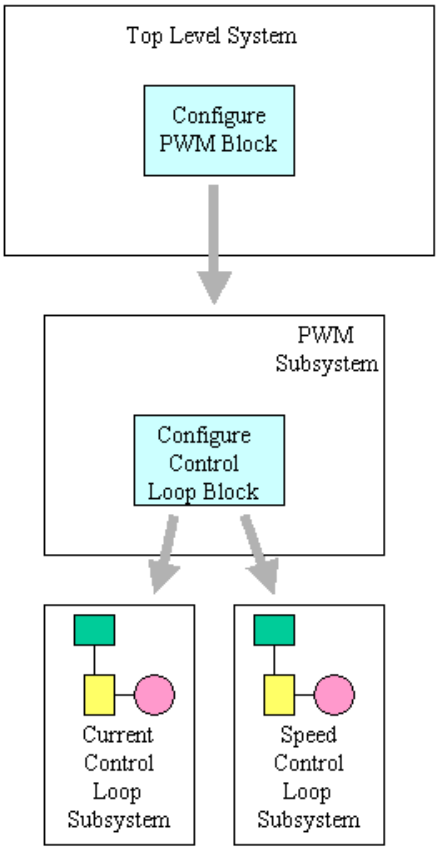


Figure 41—MCE Design Hierarchy

The top level of the system design contains a ***Configure PWM*** block and a PWM subsystem block, which is implemented using a standard Simulink ***Enabled Subsystem*** block. The Configure PWM block has one output, labeled ***PWM***. The PWM subsystem is identified by connecting the Configure PWM block output to the Enable input of a PWM subsystem block. There are no other blocks or connections at the top level of the design.

The PWM subsystem contains a ***Configure Control Loop*** block and two control loop subsystem blocks, which are implemented using standard Simulink ***Enabled Subsystem*** blocks. The Configure Control Loop block has two outputs, labeled ***Speed*** and ***Misc***. Each

control loop subsystem is identified by connecting the appropriate Configure Control Loop block output to the Enable input of a Configure Control Loop subsystem block. There are no other blocks or connections at the top level of the PWM subsystems.

The procedure described below can be used to create an empty MCE design in the correct hierarchical format. However, it is generally easier to modify the Standard design, as in the examples at the end of the Section.

Step 1
Create a new (empty) Simulink model. (From the MATLAB ***File*** menu, select ***New*** and then ***Model***.) Right click in the new window and select ***Model Properties***. On the ***Summary*** tab of the ***Model Properties*** dialog, you can enter a text description of the design and save your name as its creator.

Step 2
From the Configuration group of the MCE library, drag a ***Configure PWM*** block into the model. From the standard Simulink library's Subsystems group, drag an ***Enabled Subsystem*** block into the model. Connect the output of the Configure PWM block to the Enable input of the subsystem block. Double click the label under the subsystem block to enter a name of your choice for the PWM subsystem.

Step 3
Double click the Motor PWM subsystem to open it. Delete the default input and output ports and the line that connects them. From the Configuration group of the MCE library, drag a ***Configure Control Loop*** block into the model. From the standard Simulink library's Subsystems group, drag ***Enabled Subsystem*** blocks into the model. Connect the outputs of the Configure Control Loop block to the Enable input of each of the subsystem blocks. Double click the label under each subsystem block to enter a name of your choice for the control loop subsystem.

Step 4
The hierarchical structure is now complete, and you can begin designing your motion control algorithms by adding and connecting MCE library blocks in each of the control loop subsystems.

3.3.2 The MCE Library

The modules of the Simulink library are grouped into seven main categories, with a library model file in the Simulink Library directory for each category. This section gives a brief description of each library. Detailed descriptions of each of the blocks are provided in the Reference Manual. These are:

- Configuration
- Registers
- Control
- Math
- Tools
- Motion Peripherals
- Designs

Simulink library files have a .mdl filename extension (same as Simulink model files). For example, the Math library file is named Math.mdl.

The MCE library contains various control block modules specific to motor control applications as well as a number of general-purpose modules for miscellaneous operations and support functions. The main window of MCE Simulink library is shown in Figure 42. By connecting library blocks in the MATLAB/Simulink™ environment, the user can design a custom control algorithm based on

application requirements. A graphic compiler analyzes the completed design and automatically translates it into a sequence of MCE-specific machine code for integration with the IRMCx100. The two basic types of hardware resources available on the IRMCx100 are Motion Peripherals and Control blocks.

Motion peripherals process analog and digital signals and interface to external hardware; for example, the Low Loss Space Vector PWM module (SVPWM), Sensorless Field-Oriented Control (FOC) module, and single shunt current reconstruction module. These modules are colored yellow throughout this document and in Matlab/Simulink™ to distinguish them from other elements. Each motion peripheral module is used only once in an application design since it corresponds to a single hardware resource.

MCEControl Blocks are the math, control, and logic elements implemented in hardware. These modules can be used in an application design as many times as needed. MCEControl block signals can be connected to another MCEControl Block or to a Motion Peripheral module. MCEControl Blocks are colored green (for math) or blue (all others) throughout this document and in MATLAB/Simulink™. There are no pre-defined registers for control block configuration and monitoring as there are for the motion peripherals.

Additional blocks are provided for support functions such as data initialization and monitoring, signal delays and page-to-page connections. Some support functions are implemented using standard Simulink library components.

All blocks are based on 16-bit signed or unsigned integer input and output.

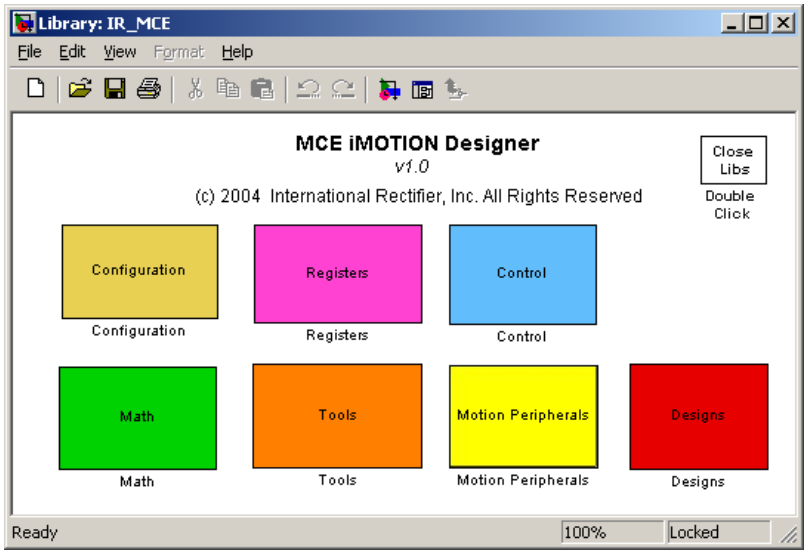


Figure 42—MCE Simulink Library

The seven library groups are described below.

Configuration
The Configuration group contains the Configure PWM and Configure Control Loop blocks that are used in the formation of the MCE hierarchical design for a complete system. If you create your system design using the MCE design template file template.mdl or by modifying the standard design, these blocks are already included at the appropriate locations in the subsystem hierarchy. (See Section 3.3.1 for more information.)

Registers
The Registers group contains read and write register blocks, which you can use in any of your control loop subsystems. If you want to define a configurable parameter that can be set from the MCEDesigner tool or from an 8051 application, drag a write register block into your design and connect its output to the input of the appropriate module(s) that will use the parameter. If you want to monitor a module output from MCEDesigner or an 8051 application, drag a read register block into your design and connect the module output to it.

Control
The Control group contains the special-function motion control blocks that are used to implement your motion control algorithms. You can drag these blocks into any of your control loop subsystems.

Math
The Math group contains general-purpose math and logic blocks that you can use in any of your control loop subsystems.

Tools
The Tools group contains the MCE Compiler block, which you can add to your design to simplify access to the MCE compiler (see Section 3.4.3 for more information). The Tools group also includes a Host Register Summary block, which you can add to your design and use to view and modify the read and write host register blocks you've included in your design (see Section 3.4.2.2) and a tool that allows you to customize the inputs and outputs of certain motion peripheral blocks (CustomMotPer, described in Section 3.4.2.1).

Motion Peripherals
The Motion Peripherals group contains the special-function motion peripheral blocks that can be included in your control loop subsystems. The blocks in this group have input and output registers that are described in the Reference Manual. Because some of the blocks have a large number of inputs and outputs, the designer has the ability to enable only the inputs and outputs needed for the control loops as described in Section 3.4.2.1.

Designs
The Designs group contains sample designs shipped with the product, as well as the system template design that you can copy and use as a basis for your system designs. You can add your custom system designs to this library group if you wish.

3.3.3 Standard Simulink Library Components

The standard Simulink library components described below can be included in your design. Enter "simulink" in the MATLAB command window to open the Simulink library.

Enabled Subsystem
Use this block to create PWM and control loop subsystems for your system design. If you start with the MCE design template file template.mdl, the appropriate subsystem blocks are already present in the design. Refer to Section 3.3.1 for more information about the use of the Enabled Subsystem block in the MCE design hierarchy.

Constant
Use this block to define a constant value as an input to a block in any of your control loop subsystems or macro block definition. Double click the constant block to set a value for the constant.

Scope
If you want a module output in a control loop subsystem to have the capability of being traced (using MCEDesigner's trace monitor feature), drag a Scope block into your design and connect the module output to it. The name you assign to the Scope block will be used in MCEDesigner so you can recognize the trace item.

Goto and From
If you need to connect elements in two different subsystems of your design, you can use a Goto block at the source of the signal and a From block at the destination. To avoid cluttering your diagram with long and roundabout lines, you can also use Goto and From blocks to connect elements at distant points within the same subsystem.

After dragging a Goto into your design, double click it to set its parameters. Set the tag field to a unique name, which is used to match the Goto with one or more From blocks. Set tag visibility to "global" if any matching From blocks are in other subsystems or "local" if all matching From blocks are in the same subsystem as the Goto. (Visibility type "scoped" is not used.) Double click each From block to set its goto tag. This tag identifies the matching Goto block and must match the tag you specified in the Goto block.

Note: When a block's input is obtained from another subsystem (using a global Goto and From), The compiler cannot guarantee that the block providing the data will execute before the block using the data. Depending on the order of execution of the subsystems (which can depend on product type and configuration), the input data may be current (if the providing subsystem runs first) or one cycle old (if the providing subsystem has not yet run during the current PWM period).

Unit Delay
You can use the Unit Delay block to introduce a signal delay of one or more PWM cycles. In certain situations, a delay is required to identify a feedback signal (an input data value obtained from a previous cycle). For example, suppose an output of block A is used as an input to block B and an output from block B is used as an input to block A. Both inputs cannot be generated on the current cycle since one block must execute before the other. A Unit Delay block must be inserted in one of the two paths (between block A's output and block B's input or between block B's output and block A's input) to identify which signal is obtained from a previous cycle. The compiler uses this information to sequence the blocks correctly.

After dragging a Unit Delay block into your design, double click it to set its parameters. The initial condition defines the value of the signal used for the initial cycles until stored values (from previous cycles) are available. The sample time defines the number of cycles to delay. (Note that the MCE Compiler's use of the sample time parameter differs from Simulink's definition.)

Model Info
You can include a Model Info block at the top level of your design to provide a description of the design and keep track of modifications and version numbers. The Model Info block allows you to enter text and special variables that Simulink automatically replaces with model property information such as creator, version and modification date. In addition, the MCE Compiler looks in the Model Info block to find information about compatible firmware options. See Section 3.4.2.3 for more information on this feature.

3.4 New MCE Design—Start to Finish

This section describes how to create, compile and download MCE designs in the MATLAB/Simulink environment.

A Simulink model (.mdl) file defines a graphical Simulink model, or design, using a proprietary syntax in text format. The basic elements of the definition syntax are Systems, Blocks, Ports and Lines. A System is a functional collection of Blocks and Lines. A Block is an individual design component or a representation of a subsystem. Ports define the inputs and outputs of a Block or a System, and Lines are the connections between Blocks. Using a Block to represent a subsystem enables the creation of a hierarchical design.

The MCE compiler analyzes the graphical elements defined in a model file to generate the MCE program to implement the represented design on the Motion Control Engine processor. The MCE compiler analyzes a Simulink model file and uses information in the database to determine inputs and outputs for each Block and an execution sequence for the Blocks. It then creates an MCE executable (.bin) file for a complete system build. The compiler also creates the following optional output files:

- A register map file that can be imported into MCEDesigner so host read and write registers defined in the design can be accessed through MCEDesigner at runtime.
- A header file in C source code format that defines the host read and write registers so they can be accessed from an 8051 application resident on the IRMCx100. (See Software Developer's Guide.)
- A listing file that shows the block sequence and the block input/output connections.

3.4.1 Setting up Matlab/Simulink

Before You Start

The very first time you use the MCE design tools with MATLAB, you need to create a MATLAB search path for MCE so that MATLAB knows where to find the MCE Libraries and utilities. To set the search path, you'll need to know the location of the main MCE directory within your iMOTION software installation. (The default path is C:\Program Files\iMOTION\MCE Compiler, but a different location can be selected during installation.) If you're not sure where the software is installed on your computer, open an MS-DOS command prompt window and type the following command:

echo %MCEBASE%

This command displays the full pathname of the MCE base directory.

To set the search path, start MATLAB and select *Set Path…* from the *File* menu. In the *Set Path* dialog box, click the *Add Folder…* button and browse for the main MCE directory. Click *OK* in the *Browse for Folder* dialog box and then click *Save* in the *Set Path* dialog box. Click *Close* to close the dialog box. (If you don't click *Save* before you click *Close*, you'll need to add the search path again next time you run MATLAB.)

3.4.2 Creating a Complete System Design

This section describes how to create a complete system design for execution on the IRMCx100.

Step 1

Start MATLAB, and in the MATLAB command window, type "mceinit100" to open the MCE Simulink Libraries. Open the standard libraries supplied with Simulink by typing simulink in the command window.

Step 2

Create a new Simulink model file with the appropriate MCE subsystem hierarchy. The easiest way to do this is to make a copy of the reference design model file and open it in MATLAB. If you want to create your own MCE model template, refer to the description in Section 3.3.1.

Step 3
Compose the design of each control loop subsystem within your model. You can drag and drop blocks from the MCE libraries into the control loop subsystems. (Do not add blocks to the top level or the PWM subsystems.) Use Simulink's graphical design features to arrange, resize and connect the blocks appropriately. To document your design you can add annotations and, if you wish, assign a descriptive name to each line and block.

Step 4
Customize your read and write register blocks. Write register blocks define parameters that you want to be able to set through the host interface at runtime. Read register blocks define output values that you want to be able to view through the host interface. To customize a register block, double click it. In the ***Parameters*** section of the ***Mask Parameters*** dialog box, follow the prompts to enter the desired values. This information is exported to MCEDesigner in the register map file.

Step 5
When you are satisfied with your Simulink design, it's time to run the compiler. This procedure is detailed in Section 3.4.3.

3.4.2.1 Customizing Motion Peripheral Library Blocks

The CustomMotPer tool allows you to modify the inputs and outputs of certain motion peripheral library blocks. You can add and remove inputs and outputs by selecting from lists of available signals. A full description of the inputs and outputs is available in the Reference Manual.

To customize a motion peripheral block, first drag it from the library into your design. Then drag the CustomMotPer block from the Tools library into your design and double-click it.

When you double-click the CustomMotPer block, it starts the Customize Motion Peripheral Block GUI, as shown in Figure 43. The GUI has a single screen, at the top of which is a pull-down list of the customizable blocks in your design. Once you've selected the block you want to customize, the currently defined inputs for the block are shown in the list on the left-hand side of the window and the currently defined outputs are shown on the right.

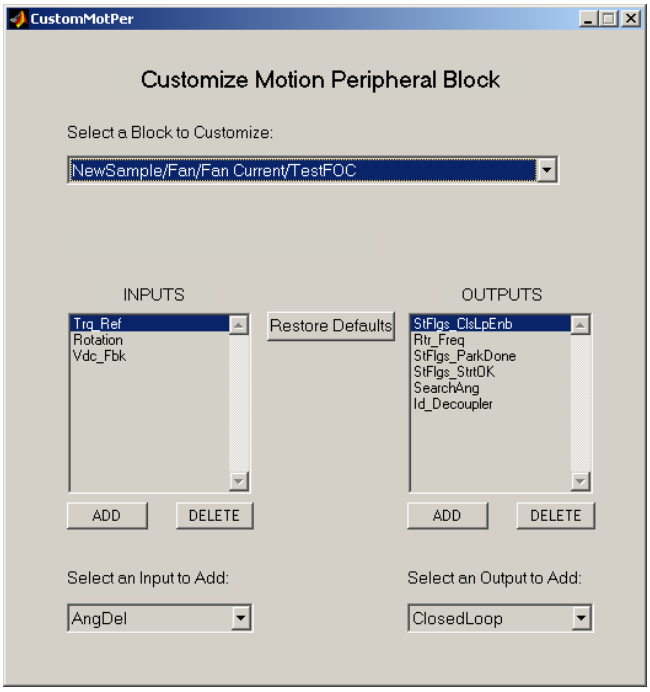


Figure 43—The CustomMotPer Utility

Summary of the display:

- The pull-down list labeled “Select a Block to Customize” lets you choose any one of the customizable blocks in the design that’s currently open in Simulink.
- The Inputs and Outputs list boxes show the inputs and outputs (respectively) that are currently defined for the selected block.
- The pull-down list labeled “Select an Input to Add” lets you choose from a list of inputs available for addition to the selected block.
- The pull-down list labeled “Select an Output to Add” lets you choose from a list of outputs available for addition to the selected block.
- Click the ADD button after selecting an input or output from the appropriate “available” list.
- Click the DELETE button after selecting an existing input or output.
- Click the Restore Defaults button to restore the entire block (inputs and outputs) to the standard default settings (as defined in the Motion Peripherals library).
- When you click DELETE or Restore Defaults, a confirmation message with CANCEL and OK buttons is displayed in red in the upper portion of the window. Click the CANCEL button to abort the operation or OK to proceed.

To delete an existing input or output:
In the Inputs or Outputs list box, click on the item you want to delete and then click the DELETE button. In the upper part of the window, click the red OK button to confirm the operation.

To add a new input or output:
Select an available input or output from the appropriate pull-down list. Click the ADD button to add the new input/output.

To restore the default inputs and outputs:
Click the Restore Defaults button. In the upper part of the window, click the red OK button to confirm the operation. This restores all inputs and outputs to the default configuration. (You can’t restore only inputs or only outputs.)

Once you've customized a block in your design, you can copy it to another location in the design (if the block is allowed to be used more than once) or drag it into another design. For blocks that can be used once, you can customize each usage of the block with different inputs and outputs.

3.4.2.2 The Host Register Summary Utility

The Tools group of the MCE Simulink library contains a block called "Host Register Summary". This utility allows you to view a list of the host read and write registers in your design. To use it, you must first drag the Host Register Summary block into your design. If you start with the MCE design template file template.mdl, the Host Register Summary block is already present at the top level.

Once you have added the block to your design, double-click the block to display a summary of your host read and write registers. If you click on a register in the list, you can view and modify the register settings.

The main window of the Host Register Summary utility is shown in Figure 44.

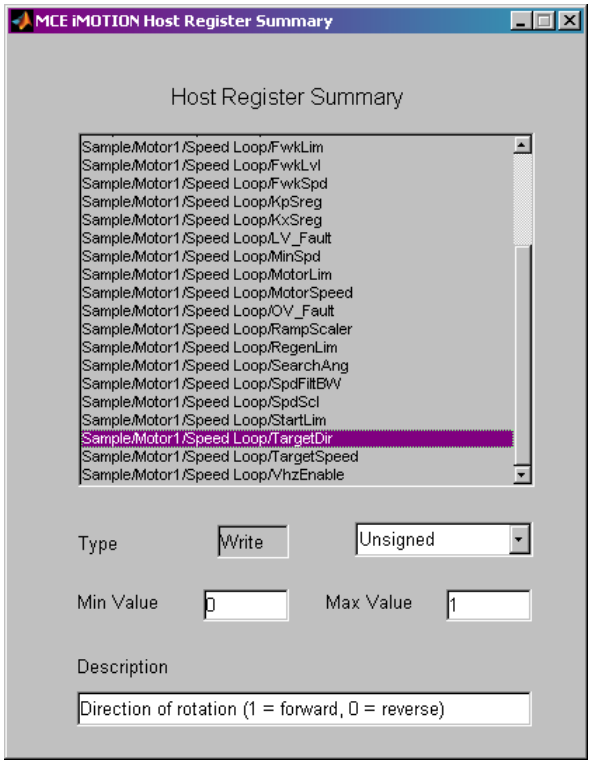


Figure 44—The Host Register Summary Utility

The list box in the top section of the main window lists the full "path" of all the registers in your design. The path identifies the model name and the subsystem in which the register is defined in addition to the register name. In the example, the path of the selected register is "Sample/Motor1/Speed Loop/TargetDir". This means that the model name is "Sample," the register is defined in PWM subsystem "Motor1" and control loop subsystem "Speed Loop." The register name is "TargetDir."

The detailed information in the lower section of the window shows the settings defined for the register that's selected in the list box. (Just click on a register to select it.) You can modify any of

the settings except the register type (read or write). Changes take effect as soon as they are entered.

3.4.2.3 Providing Information about Compatible Firmware Options

By default, the MCE Compiler links your system design with the standard MCE firmware for the product type you specify when you compile. If your design interfaces with a different version of the firmware or can support more than one firmware version, you can provide a list of compatible firmware options in a Model Info block (a component of the standard Simulink library) and the compiler uses the option information to link with the appropriate firmware at compile time.

How to Specify Firmware Options in the Model Info Block:
The Model Info block allows you to enter descriptive text in a free format. If you want to add one or more firmware options to the Model Info block for the compiler's use, you must use a defined format for the list so the compiler can recognize the information. Enter the firmware option information as follows: Below any descriptive information and variables you would like to include, enter a line that reads "Compatible Firmware:" (be sure to include the colon). Then on the following lines, list the names of the compatible firmware options, one option on each line. The firmware name is the object filename without the file extension. For example, the standard firmware object file for the IRMCx143 is svpwm143.hex, so the firmware name would be entered as "svpwm143". An example model info block listing two firmware options is shown in Figure 45.

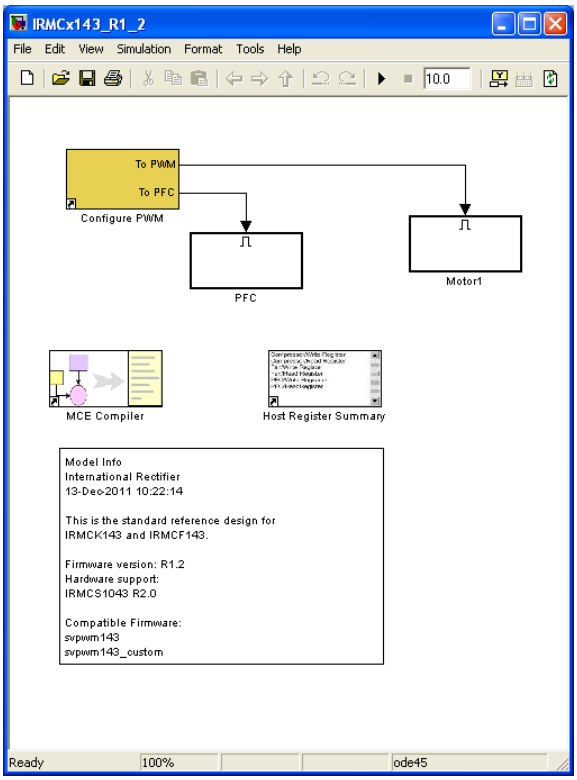


Figure 45—Example Model Info Block Showing Two Firmware Options

Where to Find the Firmware Files:
The MCE Compiler is located within the installation (typically Program Files\iMotion) in the folder MCE Compiler\bin. The firmware object files are located in device subfolders underneath MCE Compiler\bin, where each subfolder is named according to the last three digits of an IRMCx1xx part number (171, 143, etc.). When you run the MCE Compiler (see Section 3.4.3 below) you specify a product type and the compiler uses firmware only from the corresponding device subfolder.

How the MCE Compiler Chooses a Firmware Option:
After you click the Compile button (see Section 3.4.3 below for details) the MCE Compiler reads the specified model file and selects a firmware option as follows:

- The compiler uses the default firmware for the specified product type if:
  - There is no Model Info block in the design;
  - There is a Model Info block but it does not contain a "Compatible Firmware:" list; or
  - There is a "Compatible Firmware:" list but no matching firmware files are found in the device subfolder for the selected product type.
- If the "Compatible Firmware:" list contains only one entry and an object file matching that entry is found in the appropriate device subfolder, the compiler uses the firmware option specified in the list.
- If the "Compatible Firmware:" list contains multiple entries with matching object files in the appropriate device subfolder, the compiler displays a new window listing the valid firmware options and allows the user to select the desired option.
- If the saved compilation history (docompile.bat file in the MATLAB working directory) specifies a firmware option, the compiler uses the previously-selected option unless the user un-checks the "Use previously selected firmware option" checkbox.

3.4.3 The MCE Compiler

Before You Start

The MCE compiler uses the Simulink model file as input. If your design is open in Simulink when you run the compiler, be sure to save your changes before running the compiler.

The Tools group of the MCE Simulink library contains a block called "MCE Compiler". You can access the compiler by copying that block into your design and double-clicking it. If you start with one of the reference designs or the MCE design template file template.mdl, the MCE Compiler block is already present at the top level.

When you double-click the MCE Compiler block, the input screen appears as shown in Figure 46. (The "Use previously selected firmware option" checkbox is not displayed the first time you compile a design.)

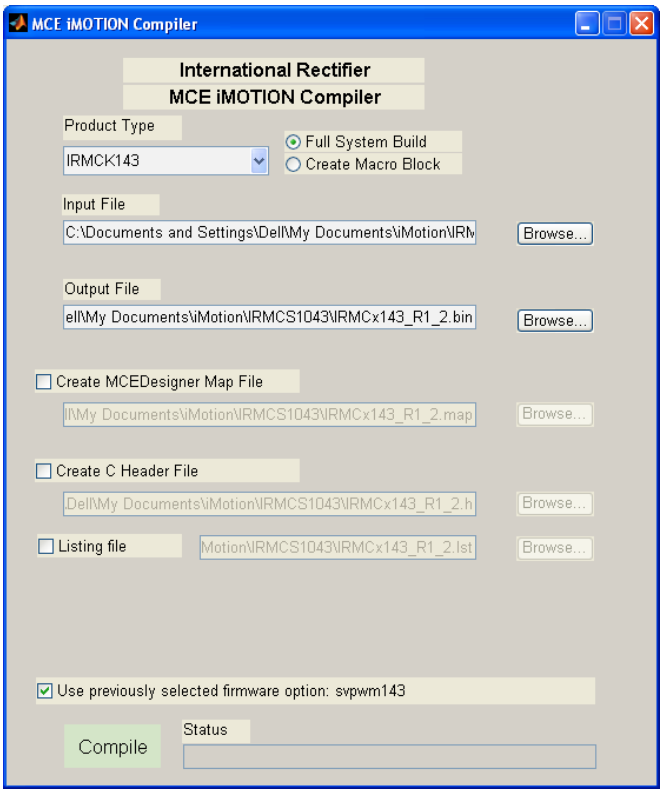


Figure 46—MCE Compiler Input Screen

Step 1
To compile a complete system design, click the “Full System Build” radio button. You can select optional output files:

- If you want the compiler to generate a register map file for use with MCEDesigner, check “create MCEDesigner Map File”.
- If you want the compiler to generate C-language register definitions in a header file for use with your 8051 application, check “create C Header File”.

Step 2
Select your product type from the pulldown menu.

Step 3
Enter the pathname of your Simulink model file in the ”Upload Design file (.mdl)” edit box, or browse for the file by clicking the browse button to the right of the edit box. The location where the output files are saved can also be modified.

Step 4
Check the “Generate listing?” checkbox if you want the compiler to generate an output text file that lists the order of block execution and all the block connections within your design. This file can be generated for either a full system or macro block compilation. You can use it as an aid in testing and verifying your design.

Step 5
If the “Use previously selected firmware option” checkbox is displayed, leave it checked unless you want the compiler to link with a different firmware version than you used the last time you compiled.

Step 6

When you're ready, click the **Compile** button to run the compiler. The compiler checks your model file for a Model Info block and selects a firmware option as described in Section 3.4.2.3. If there are multiple options are available and the "Use previously selected firmware option" is not checked (or not displayed), the compiler displays the Firmware Selection window similar to the example shown in Figure 47. Select the firmware option you want to use and click the **Continue** button.

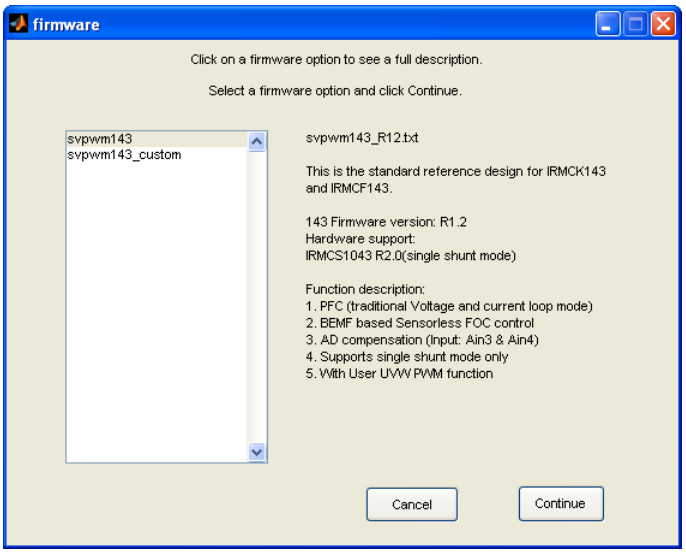


Figure 47—Example Firmware Selection Window

When compilation is complete, the MCE Compiler results window appears. An example is shown in Figure 48. To verify that the compiler linked with the firmware option you intended, look for the message "Opening system code object file…" in the results window and verify that the firmware name is correct.

If the results window is not displayed, check the "Status" information at the bottom of the main window for an error message. Many errors are caused by a mismatch between selected product type and firmware option. Check to be sure you selected the proper product type for your design.

Time Estimates

The compiler output includes execution time estimates (in system clock cycles) for each control loop as well as the total size of the MCE program and data. You should review this information carefully. The compiler displays a warning message if your code and/or data are too large to fit in the available memory. However, the compiler cannot warn you if the execution time of your control loops is too long, because the time available for control loop execution depends on the PWM frequencies configured at run time. These time estimates can be input into MCEWizard100, which will give the total MCE processor usage based on the PWM frequencies and the clock frequency. (Check the "I have modified the MCE Application Program" box on the Welcome page to modify the Motor 1 Cycles on the Options page.) For more information about setting the clock frequency, see the Software Developer's Guide or the Reference Manual.

Note: The compiler produces worst-case time estimates based on cycle counts for all MCE instructions it generates, including those that may be executed only under certain conditions. The execution time estimates documented for each block in the Reference Manual are more accurate and provide a range of cycle counts when execution time varies depending on conditions. For this reason, the compiler's execution time estimate will generally exceed the estimate you would obtain by summing the documented execution times for each block in the design.

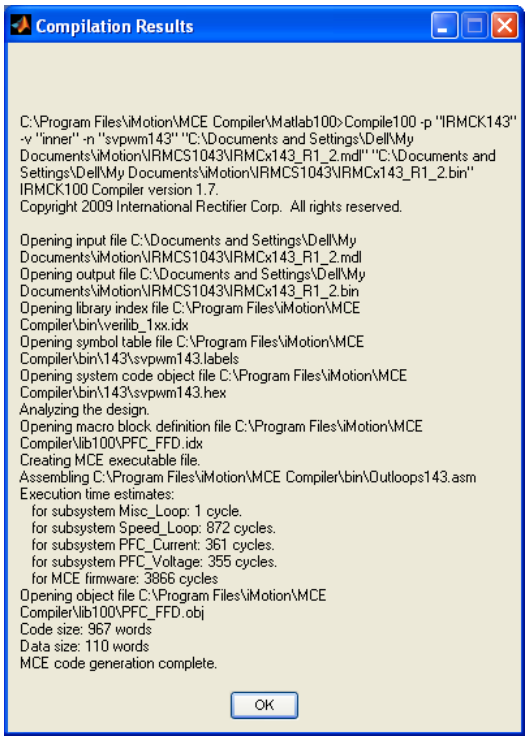


Figure 48—MCE Compiler Results Example

3.4.4 Downloading to the Reference Board

There are two different choices for downloading to the reference board, one by downloading to MCE program RAM using MCEDesigner and one by programming the OTP or flash memory of an IRMCx100 part. The option "Download to RAM" will load the MCE program (.bin file) output by the MCECompiler directly to the MCE program RAM; if the power to the control IC is removed, then the program will be lost. This method is described in detail below. Programming OTP or flash memory is described in detail in the Software Developer's Guide, including setting up of all hardware and software.

3.4.4.1 Download to RAM

If you've recompiled your MCE design and want to do a quick test without programming a new IC, you can download it directly to RAM and execute it without restarting the target platform. You can download the code and start execution in a single step, or as separate operations.

Use the following procedure to download MCE code to MCE program RAM on the IRMCx100 IC:

1. Start MCEDesigner and open a configuration (.irc) file.
2. Wait for the status bar to show that the connection is "Up". If the link comes up, but an error message is displayed showing that there is a mismatch between the MCE program and register map, you can ignore the error and proceed with the download.
3. Click on the System window and then select Load Target from the Tools menu. The Load Target dialog appears.
4. In the Load box, click the "MCE to RAM only" radio button as shown in Figure 49. In the Boot box, check the "Start MCE" checkbox if you want MCEDesigner to start execution of the MCE code after it's loaded. If you want to start execution as a separate operation, leave the checkbox unchecked.
5. In the Files box, enter or browse for the pathname of your MCE download file. The MCE executable is generated by the MCE Compiler and has the filename extension ".bin". In this mode, the 8051 download file entry box is disabled (grayed out). You cannot load 8051 code directly to RAM using MCEDesigner.
6. When you have selected a valid MCE download file, click the OK button.
7. Wait while the file is transferred to RAM over the serial link. This takes only a few seconds. When download is complete, the message "Load complete; remote boot not selected" is displayed if you didn't check the "Start MCE" checkbox or "Load and remote boot complete" if you did. (If errors occurred during download, the message "Load complete but checksums don't match" is displayed instead.)

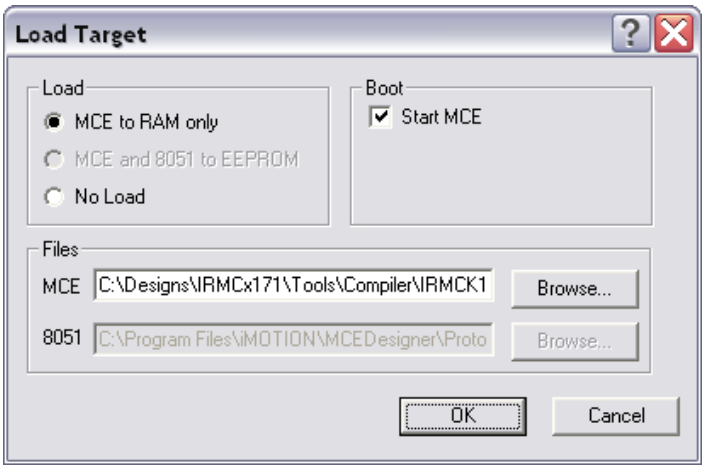


Figure 49—Load MCE to RAM

If you don't check the "Start MCE" checkbox when you download your MCE code, you need to perform a separate operation to start execution of the code. To do this, open the Load Target dialog again. This time, click the "No Load" radio button in the Load box and check the "Start MCE" checkbox in the Boot box. Then click OK.

3.4.4.2 Design ID and Revision Level Monitoring

MCEDesigner's design ID and version monitoring is a safety feature designed to prevent hardware damage caused by using the incorrect register map with an MCE program. This could cause bad values to be written to critical registers during drive configuration.

A Register Map design ID and version number are stored in every configuration file (.irc file). These values identify the MCE program that the file was created to support and the version of the design's register map that was last imported to the .irc file. (See the MCEDesigner User's Guide for more information.)

If the design ID or version number of the MCE program currently loaded to the IRMCx100 IC does not match the ID and version specified for the current Register Map (or if no MCE image is currently loaded), an error message similar to the one shown in Figure 50 is displayed. You can continue using MCEDesigner in an "offline" mode, but you cannot read or write any registers or execute any functions until the problem is corrected.

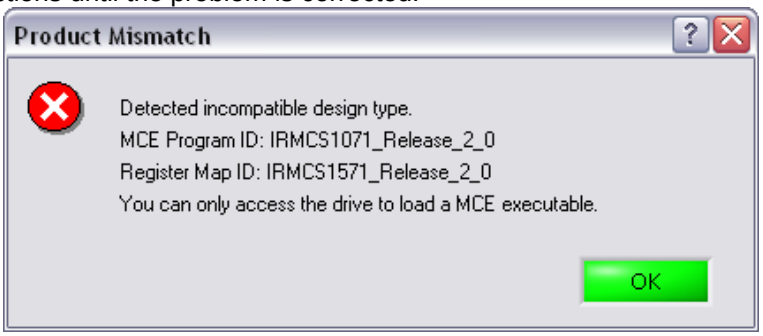


Figure 50—MCE Program/Register Map Mismatch Dialog

You can see details about the MCE Program and Register Map in the Connection dialog (Click the System window in MCEDesigner, then select Preferences→Connection.), as shown in Figure 51.

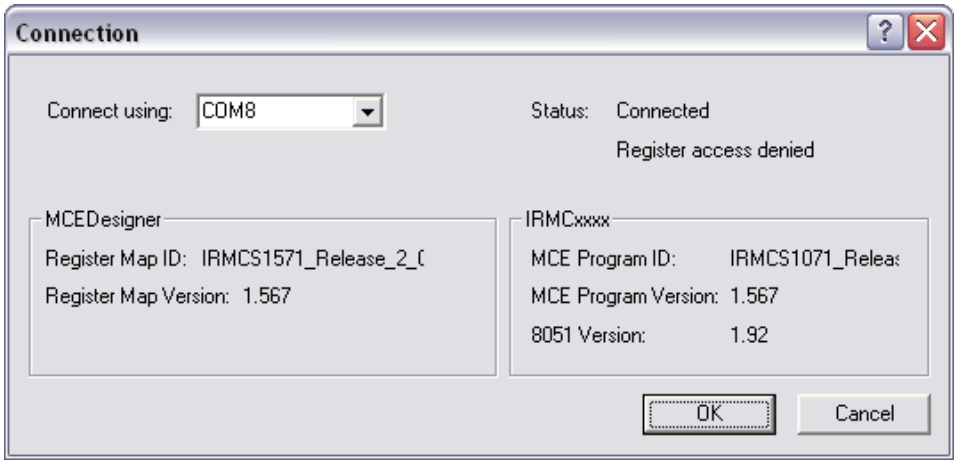


Figure 51—Design ID and Version Details

3.4.4.3 Importing an MCE Register Map

While testing the reference design, there is no need to modify MCEDesigner's register definitions, since the configuration file shipped with the release exactly matches the reference MCE program. However, when testing a custom MCE program, be sure to import the modified MCE register map into the configuration file. MCEDesigner is completely configurable, so it can be used with new designs as long as the configuration file is updated with the correct register map for the MCE program.

When making changes to the register definitions, be sure to save the changes in the .irc file before exiting MCEDesigner. If you select Save from the File menu or answer "Yes" when asked if you want to save your changes on exit, the current definitions are saved to the file that's currently open. To save your changes to a different file, select Save As… from the File menu.

When you import a new MCE register map, MCEDesigner modifies the Register Structure Definition section of the database as follows:

- If there is a new register in the MCE design that doesn't exist in your database, the register is added to the Default register group under Write Registers or Read Registers (depending on the register type). If the name of the register conflicts with an existing fixed or 8051 register, the MCE register name is modified by appending the characters "_USER" to avoid a conflict. (For example, if your MCE design contains a register named "MtrCtrlBits", the name will be changed to "MtrCtrlBits_USER" when it's added to the database.)
- If the definition of an existing MCE register has been modified, the register definition is updated to match the definition in the map file.
- If there is an MCE register in your database that is no longer defined in your MCE design, MCEDesigner displays a message box asking if you would like the register to be deleted from the register definitions or retained with "obsolete" status. You cannot write to or read from an obsolete register, and MCEDesigner ignores it when it appears within a function or subfunction.

If you use MCE registers inside functions and subfunctions, MCEDesigner updates your functions and subfunctions as follows:

- If there is a new register in your MCE design that doesn't exist in the database, it is not automatically added to any functions or subfunctions.
- If a register definition has been modified, the register is updated wherever it's used inside functions and subfunctions. This includes changes to the register description, but not the Notes field. (The Notes field is your own personal "scratch area" and is never automatically updated.)
- If a register is no longer defined in your MCE design and you choose to have it deleted from the register definitions, it's deleted from all functions and subfunctions that use it.
- If a register is no longer defined in your MCE design and you choose to retain it with "obsolete" status, it remains in any functions and subfunctions that use it, but it is ignored when the function is executed.

Use the following procedure to import a register map file into your database:

1. Open the database you want to update.
2. Click on the System window and then select File→Import Register Map from the menu.
3. When the Load Register Map window (Figure 52) appears, you can enter the pathname of your map file or click the Browse button to browse for the map file.
4. Click OK twice to open the map file and update the configuration file. Be sure to save the .irc file before exiting MCEDesigner so your register map updates are not lost.

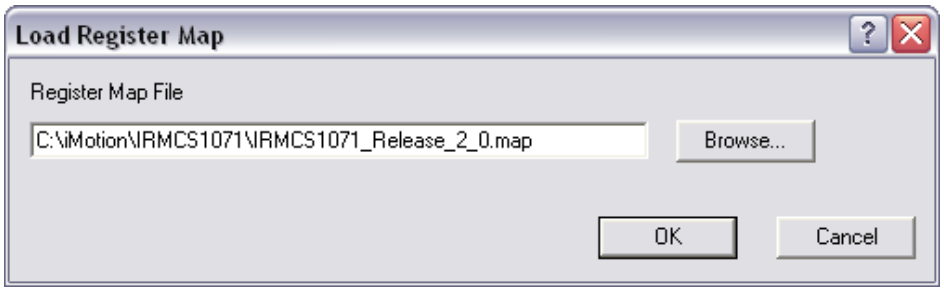


Figure 52–The Import Register Map Dialog

3.5 Example Modifications

This section will give two examples of simple modifications to the MCE program. They are presented here to demonstrate application related issues or requirements which can be satisfied by modifying the MCE program and also to present other features of the Library blocks.

3.5.1 Torque Mode

Some motor control situations require that the motor provide a steady torque, rather than a constant speed. The MCE program can easily be changed to provide such a mode. Figure 53 shows the modified section of the block diagram where two write registers have been added, ModeSelect and TorqueReq. They are both inputs to a new SWITCH block added between the PI of the speed loop and the LIMIT. Setting ModeSelect to 0 enables Torque Mode, where the requested torque is supplied directly, while 1 enables the normal speed control mode.

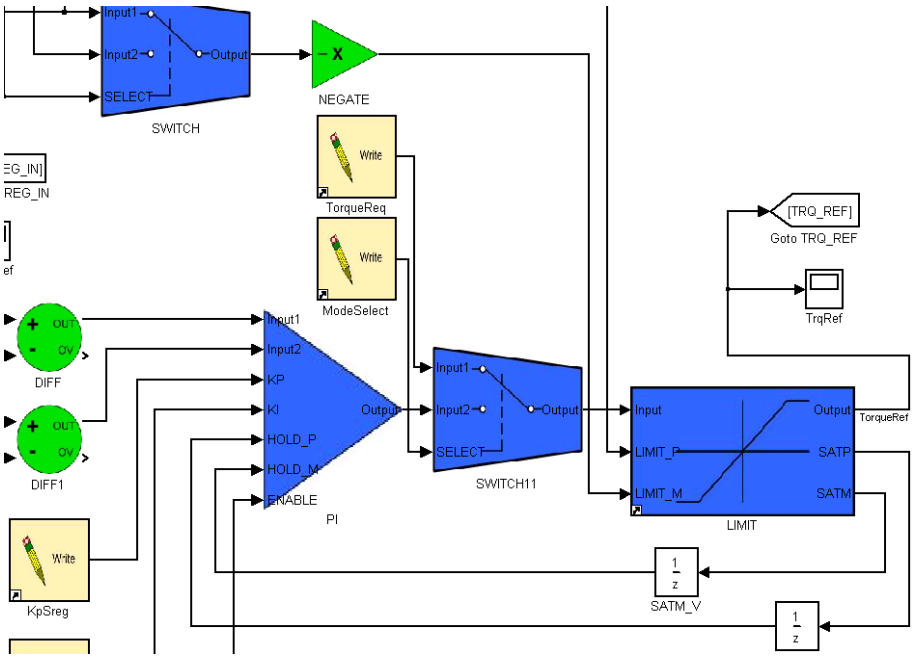


Figure 53—Torque Mode Block Diagram

As a further design example, below are listed some application-related issues which the designer should consider when implementing Torque Mode. These issues may best be solved with extra logic (for example) in the block diagram or in the 8051 application code; this choice is determined by application and design specifications.

- Scaling—The scaling of TorqueReq (which is really a current command) is same as that of the limit registers. To interpret it in terms of torque: Torque Requested (N-m) = (TorqueReq / 4095) * I_{rated} * k_T, where the last two terms are the rated current and the motor torque constant as entered into MCEWizard100.

- Starting—The starting performance may change, depending on the load.

- Current Limit—Note that the SWITCH is placed before the LIMIT block, so that the StartLimit, RegenLimit, and MotorLimit will still provide protection.

- Speed Limit—There are no speed limiting features in Torqe Mode as implemented. The application may require some protection logic.

- Changing Modes while running—Application testing can determine whether it is safe to change modes while the motor is running. The application may require a smooth change over. (Hint: This can be accomplished using a RAMP block which initializes with the output of the PI block.)

Naturally, there are other ways to produce a Torque Mode. For example, set the speed loop gains (KpSreg and KxSreg) to zero so that the output of the PI block is always zero. Then set the TorqueRef by setting the lower limit of the LIMIT block, LIMIT_M.

Note that MCEWizard100 still properly configures the registers in this modified program, though the designer must determine the proper values for the two new registers.

3.5.2 Limiting the Speed Feedback Input Variance

In some start-up situations, the speed feedback can exhibit large spikes at the transition from open-loop to closed-loop due to the time required for the Angle Estimator PLL to stabilize. These spikes can reduce the reliability of the start-up, depending on the load.

One solution is to limit the input variance of the Speed Feedback filter. Figure 54 below shows the location in the block diagram. Double Click on the LOWPASS_FILT block and a dialog box will open which allows the designer to limit the difference between successive inputs to the LOWPASS_FILT block. The limit value is valid from 0 – 15, and is interpreted as 2^{LIMIT}; for example, a value of 8 will limit the difference between an input and the previous input to 256. There are several blocks with hidden configurable parameters such as this; see the complete block descriptions in the Reference Manual. In this scheme, the unfiltered speed change is not limited, but the effect of an integral controller is to filter this feedback component as well.

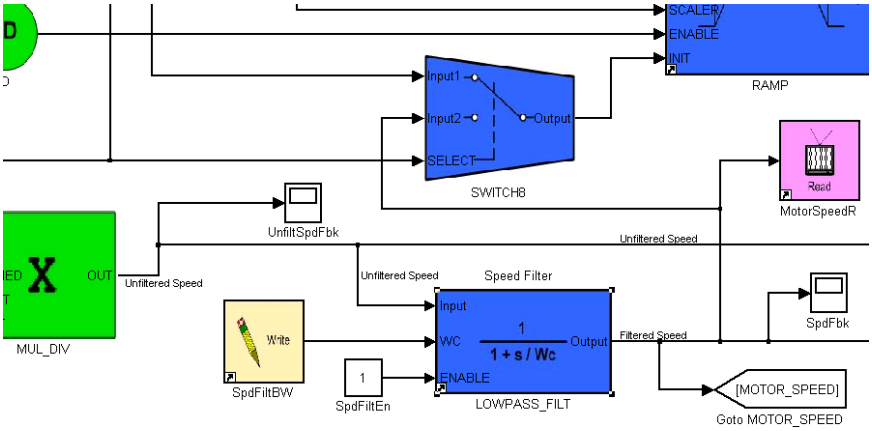


Figure 54—Speed Feedback Section

Note that this block diagram also has an additional scope, UnfiltSpdFbk, as an example of adding a value which can be monitored with the Trace function of MCEDesigner. This information is imported into the configuration file as part of the register map file.

4 Motor Application Hardware Design

This chapter of the Application Developer's Guide discusses hardware design issues specific to the IRMCx100 series ICs. Section 4.1 discusses issues related to the development of the hardware schematic including the power supply, oscillator, and communication requirements for component selection, setting the feedback scaling and over current protection. Section 4.2 gives layout recommendations relating to the current feedback circuit, and Section 4.3 finishes with some suggestions for testing the board and optimizing the settings. This chapter references MCEWizard100's hardware-dependant settings throughout; to modify these settings, be sure to check the box next to "I have modified my hardware" in the Welcome page.

4.1 Schematic Elements

This section begins by giving specifications and tips related to component selection for various in-circuit functions. Next, it reviews the feedback scaling for current and voltage as related to entering values into MCEWizard100. This section also discusses techniques for A/D offset compensation and proper overcurrent protection.

4.1.1 Component Selection

The information in this section aids the designer in selecting some of the components of a custom hardware design.

Power Supplies—The IRMCx100 series IC only requires a 3.3V (VDD1) input with **max 60mA** current required. There is an internal 1.8V regulator to provide for the IC logic; capacitors should be connected between the pins VDDCAP and VSS.

Input Clock—The control IC requires an external clock input for proper operation. This clock may be generated by a quartz crystal oscillator or ceramic chip resonator. The input clock is connected to a configurable phase-locked loop (PLL) to generate the internal clock for the IC. The input clock frequency must be between 3.2MHz and 60MHz, with a resulting internal clock frequency of 32MHz to 128MHz, which is configured by the user. The Reference Design Kits use a 4MHz crystal and is configured for a 100MHz clock frequency. Details of configuring the PLL to generate the internal clock are found in the Software Developer's Guide and the Reference Manual.

JTAG Interface— It is important to provide galvanic isolation to the JTAG interface to protect any external JTAG interface hardware. If this is not done, then the JTAG hardware ground should be connected to the ground of the DC bus and the power ground of the IC.

The JTAG has a configurable clock frequency (TCK). Be sure to verify that the isolation circuit is capable of the data speed. The FS2 JTAG pod has a TCK frequency of 500kHz ~1MHz.

The IRMCS-ISO V3.0 or higher version isolation box contains all of the JTAG isolation related circuits, nearly identical as the IRMCS1271/1043 isolation circuits and supports both 10-pin and 20-pin FS2 JTAG pod. The isolation box can interface to debug tools for custom hardware designs which do not include isolation.

RS232 Interface and Driver—Like the JTAG interface, if the RS232 hardware (i.e. the designer's computer) is not at the same ground as the IC and DC bus then isolation should be provided for protection.

The RS232 interface speed is limited by 8051 clock speed. For full range of options, design RS232 data circuitry to run at a baud rate of up to 115,200 bps.

The IRMCS-ISO V3.0 (or higher version) isolation box provides two interfaces to the UART isolation circuit, one supporting RS232 and another USB for connecting to the PC.

Reset Circuit—Besides galvanic isolation as part of the JTAG interface, the designer may require bidirectional signal flow for the reset signal. The specific circuit required for a bidirectional signal will depend on the specifics of the JTAG interface and hardware. Another consideration is if a hard (physical switch) reset is required, or if a software reset is adequate.

In-Circuit OTP Programming—The designer may want the ability to program the OTP in-circuit for IRMCK100 ICs. The main interface to perform this function is the JTAG. In addition to the JTAG interface, there should be a provision to connect the programming voltage (6.75V) to pin VPP/P1.5. More information on OTP programming can be found in the Software Developer's Guide.
The IRMCS-ISO V3.0 or higher version isolation box contains OTP programming related circuits, which include 6.75V VPP and JTAG isolation circuits. It can provide an interface for in-circuit programming of the final application board.

Decoupling Capacitors—The IRMCx100 IC places power supply pins next to ground pins in order to make it easy for the designer to place decoupling capacitors between these pins; they should be placed as close as possible to the pins and have values of 0.1uF and 0.01uF, depending on power supply type and its voltage ripple.

Current Feedback Op-Amp Capacitors—The Reference Designs place capacitors at the output of the current feedback op-amp, as shown in Figure 56. These capacitors stabilize the op-amp output, preventing oscillation. The op-amp oscillation varies between production lots and is also influenced by the layout. The value recommended below will solve this problem provided that the designer follows the layout guidelines of Section 4.2.1.

We recommend that a 47pF capacitor be placed between the op-amp output and the analog ground (C33 of Figure 56). The capacitor choice will influence the minimum pulse width and the sampling delay required because it slows down the response of the op-amp output. A larger minimum pulse width can result in greater acoustic noise, particularly at low speeds. See Section 4.3.3.2 for more discussion on choosing the minimum pulse width and the sampling delay.

4.1.2 A/D Feedback Scaling

4.1.2.1 Motor related AD

There are two important signals that the IC uses in the control of the motor and to protect hardware: shunt current and DC bus voltage. This section describes how to determine the correct values to input into MCEWizard100. MCEWizard100 calculates the correct scaled values to configure the controller based on the user hardware.

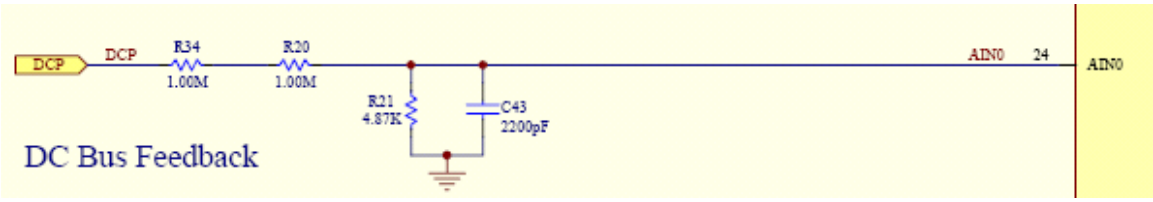


Figure 55—DC bus Feedback Voltage Divider

DC bus feedback Scaling—This input to MCEWizard100 is the internal scaling of the DC bus in cts/V. The DC bus is sensed at AIN0 through a voltage divider. To calculate the proper value for

this cell, one needs to know that 1.2V input at AIN0 is equal to 4095 digital counts. As an example, for the DC bus feedback circuit shown in Figure 55, the value for the DC bus feedback scaling is calculated as follows:

*DC bus Scaling (cts/V) = 4095 * (4.87k / (1.00M + 1.00M + 4.87k)) / 1.2 = 8.29 cts/V*

Current Feedback Amplifier Gain—This MCEWizard100 input is the gain of the current feedback amplifier. The user can configure this gain as desired by changing the resistors of the input circuit to IFB+, IFB- and IFBO. An example is shown in Figure 56,

Motor Current Feedback Amplifier Gain = 11.8k / (5.11k + 1.00k) = 1.93

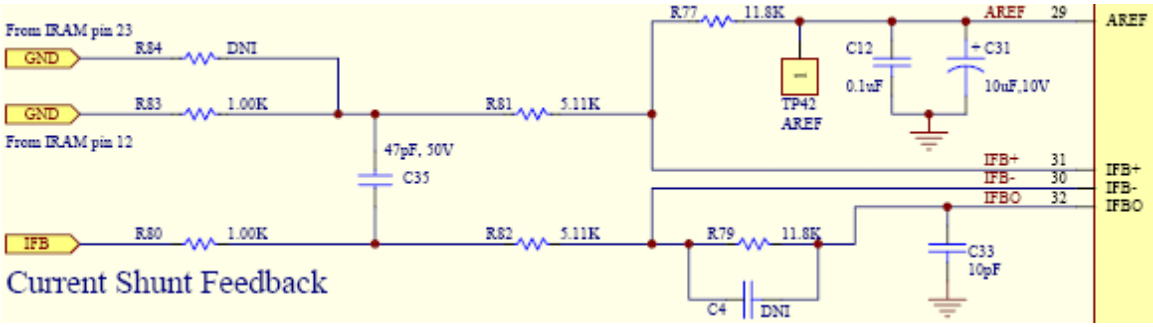


Figure 56—Current Feedback Circuit

Motor Current Feedback Shunt—Enter the shunt current resistor value into this MCEWizard100 field.

The Verify & Save page of MCEWizard100 displays the current at which the A/D converter saturates. This is calculated based on the shunt resistor value and the current feedback amplifier gain. It is important to keep the motor drive current less than this value to keep the currents in control. For example, if the A/D converter saturates at 10Apk, then the motor current should not be set higher than 6.4Arms, which leaves ~10% margin before the A/D saturates. On the other hand, leaving too high a margin will result in low current resolution. If the peak rated current is less than 25% of the A/D saturation current, then MCEWizard100 will display a warning on the Verify & Save page.

4.1.3 Gate Drive Signals

The primary outputs of the control IC are the six PWM gating signals. When the PWM gating is disabled, the output pins of the control IC enter a high impedance state where they are weakly pulled up, to about 2V. These PWM gating pins should be pulled up or down, depending on the gate driver IC logic, to prevent unwanted turn on of the IGBTs. The recommended pull-up or pull-down resistor value is 4.7kOhms.

4.1.4 A/D Converter Offset Compensation

The IRMCx100 series motor controllers utilize an internal Analog-to-Digital converter for feedback of important system parameters. This ADC, like all others, experiences offset in its measurements. These offsets can be amplified in measurement situations such as monitoring the DC bus voltage. With a small ADC input range of 0-1.2V potentially amplified into the 300-400V range, a small 10mV offset could result in a DC bus measurement error of 10V. Such an offset may not only affect ADC performance, but rotor angle estimation and field-weakening capability as well. These effects are minimized by a firmware function.

The firmware contains an automatic A/D compensation function. This function requires the use of two analog channels to provide the reference voltages for the compensation, with input pins for

each part and recommended voltages as listed in the table below. Additionally, the MCEWizard calculates values for registers AdRefH and AdRefL which correspond to the ideal A/D converted value of each reference voltage.

Reference Input	Recommended Voltage	IRMCx171, IRMCx143
High Reference	about 1100mV	AIN3
Low Reference	about 100mV	AIN4

After the first MCE configuration (MtrSeqCtrl=1), the firmware will accumulate the raw ADC result of AIN3 & Ain4 for 4096 times (Motor PWM cycles), then take the average value to filter the A/D result.

By using AdRefH & AdRefL (ideal AD results of AIN3 & Ain4 inputs) the firmware can calculate the real AD gain and offset information, which is stored as AdGainx1024 & AdOfst. These values become the K & B, respectively for 2-point AD compensation.

Figure 57 below shows how the AD correction parameters are calculated by the firmware based on the AD converted result and the ideal AD converted value of the reference voltages. The correction factors can be read by the designer in registers AdGainx1024 and AdOfst, described in detail in Section 3.4.25 of the Reference Manual.

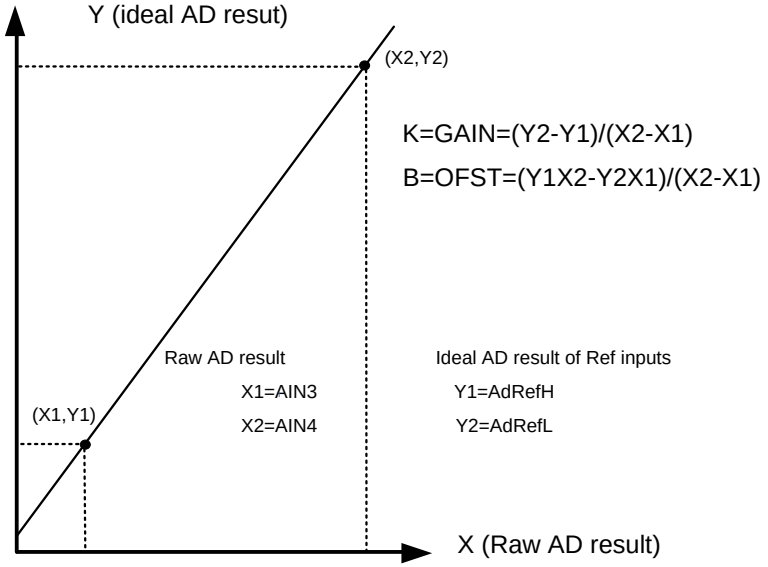


Figure 57. Calculation of A/D correction parameters

The AD compensation is actually is the computation of Y=K*X + B, where Y is the compensated AD result output and X is the raw AD result input.

Please note that AD compensation is only avaibale 4096 PWM cyles after the first MCE configuration (8051 set to MtrSeqCtrl=1 after power on). Before setting MtrSeqCtrl=1, please make sure AdRefH & AdRefL as well as all of other MCE registers are correctly configured. During the 4096 motor PWM cycles, when MCE is calculating the AD offset and gain parameters, it is better for the 8051 not to read any AD results. It is recommended to wait longer than PWM period * 4096 before reading AD channels from 8051.

When this function is enabled, all of the outputs of the the A/D blocks are corrected using the formula below. The A/D compensation function is enabled by default. To disable A/D compensation, set bit 9 of the MtrCtrlBits_S register to 1.

There is also an A/D compensation fault which indicates that the correction values are outside of a reasonable range. If this fault occures, the reference voltages or the values of registers AdRefH and AdRefL are incorrect, or that the IC has an abnormally large AD conversion error. More information about the A/D compensation fault can be found in Section 3.3.6 of the Reference Manual.

4.1.5 Overcurrent Protection

Overcurrent protection circuitry is required by most applications to prevent damage to the hardware and the motor. The IRMCx100 IC contains an input pin, GATEKILL, which is designed to provide the overcurrent shutdown signal to the controller, though it can also be used to signal an arbitrary hardware fault condition. Upon assertion, all PWM gating is halted and the IC latches a fault. One method of implementing overcurrent protection is to compare the voltage across the shunt resistor to a voltage set by a resistor network. The output of the comparator is connected to the GATEKILL pin of the IC through a resistor. To prevent unwanted overcurrent trips due to switching noise, connect a capacitor between the comparator input and ground. Additionally, the register GkillFiltCnt sets a minimum GATEKILL signal time, effectively ignoring any GATEKILL trip signal shorter than this time. Be sure that the inverter stage is compatible with the overcurrent shutdown latency.

In some types of IRAM modules that include a shunt resistor, the gate drive IC already has overcurrent protection built in, with a pin assigned to signal the shutdown to the controller. The gate drive IC shuts down the PWM switching very quickly to protect the power stage. In this case, connect the pin to GATEKILL so that the control IC is aware of the over current shut down, and stops the PWM signals.

4.2 Layout Recommendations

4.2.1 Current Feedback Circuit with IRMCx100

The IRMCx100 series IC has the necessary circuitry to implement PFC current (applies to 188 &143), motor single shunt or 2-shunt current feedback including built-in operational amplifiers, sample & hold hardware, and multiplexers. Sample timing is determined by the PWM logic automatically. The only things the designer needs to do is add resistors and capacitors in order to configure the internal operational amplifier as a differential amplifier and then adjust the minimum pulse width and sampling instances to optimal ones by setting parameters such as TcntMin3Phs and SHDelay. This process of optimization these settings is described in Section 4.3.

PFC current sampling, and motor single shunt current feedback are similar in that both are very sensitive to the PCB layout. The guidelines for motor current feedback layout also apply to PFC current feedback.

Figure 58 shows an example of this differential amplifier circuit. Rsh is the shunt resistor in the negative DC link current path. The voltage across this shunt resistor (displayed as IFB in Figure 65) is used as the input to the amplifier, the gain of which [R5 / (R1 + R3)] should be set appropriately to cover the operating range with maximum resolution, as described in Section

4.1.2. The capacitor C5 is to stabilize Cmext, which is an un-buffered 0.6V reference, and C4 is for Aref, which is the buffered 0.6V reference voltage. C2 and C3 may also be required to stabilize the operational amplifier output, IFBO. Check your Reference Design Kit for appropriate component values. Feedback resistor R5 (=R6) needs to be in the range of 5K to 20K Ohm. AVDD and AVSS are power pins for the IC's analog circuitry and require decoupling capacitors of 0.1 µF and 0.01 µF in parallel.

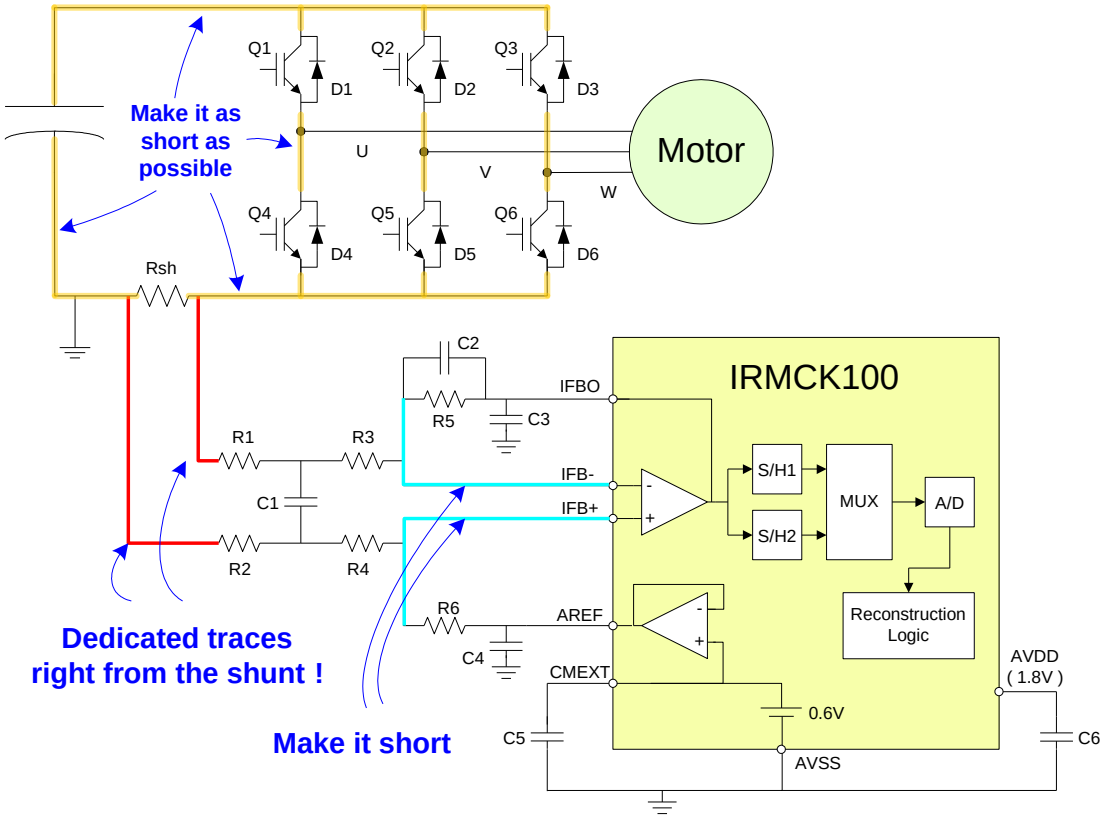
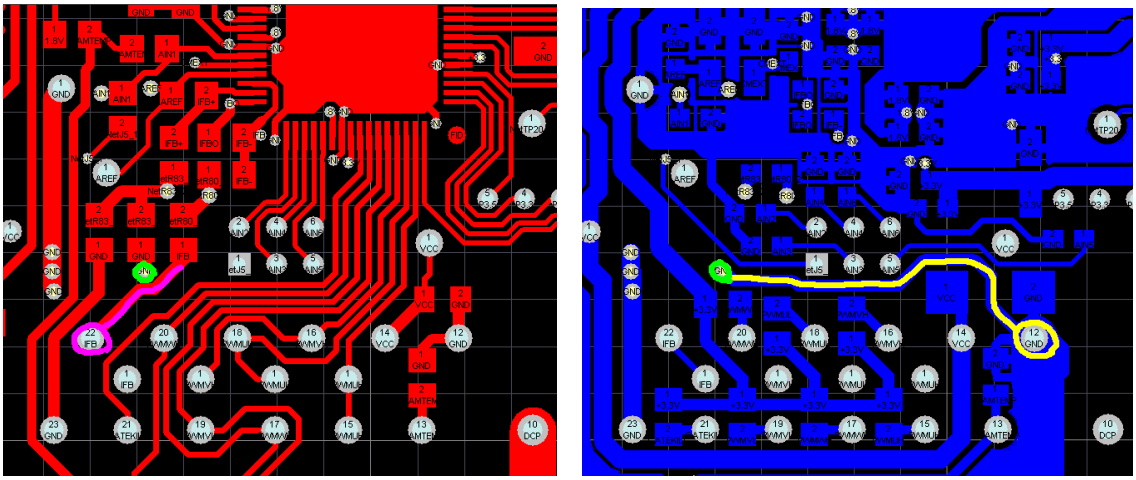


Figure 58—Current Feedback Circuit for IRMCx100

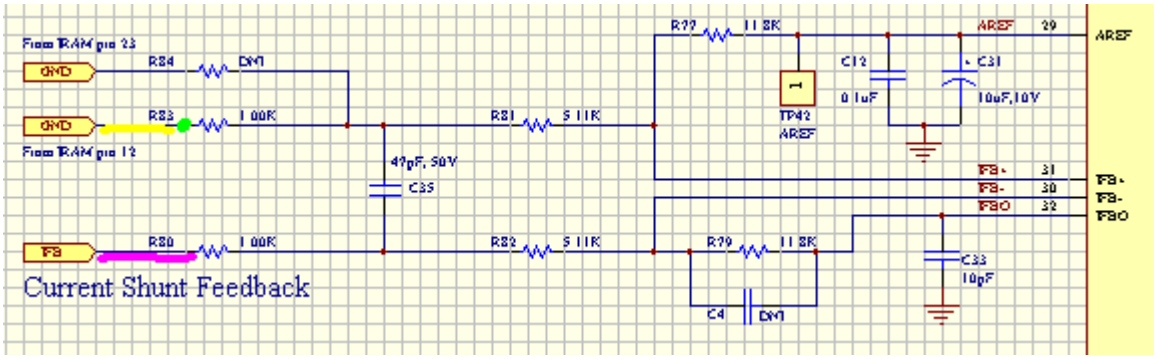
Layout for the single shunt current feedback should be done very carefully. The most important thing is to use dedicated traces right from the shunt resistor to the resistors of amplifier. Traces must not be shared with ground planes. Another important consideration is to make the power traces among the IGBTs and DC bus capacitors as short as possible. The stray inductances on these traces increase the size of the voltage spike at the switching instances. Figure 59 is a layout example from the IRMCS3041 Reference Design Kit. On the bottom layer, a trace starts right from pin 12 of IRAMS10UP60B (3-phase inverter module) separate from ground, *i.e.*, the negative DC bus.

Another very important issue is that noise from a switching power supply may significantly influence the current feedback. It is recommended to separate the IRMCx100 ground not only from the main power ground but also from the power supply primary side ground.

The internal operational amplifiers are specifically designed for this application. They have high gain, bandwidth, and slew rate to respond to the rapid rise of current through the shunt resistor. A sample & hold circuit actually tracks the signal and then holds it to reduce the sampling time. For more information regarding characteristics of operational amplifiers, sample & hold and A/D converter, please refer to the datasheet of the control IC.



(a) Top Layer (b) Bottom Layer



(c) Schematic

Figure 59—IRMCS3041 Reference Board Layout

4.2.2 Overcurrent Protection Layout

In a similar way to the current feedback layout, the overcurrent protection circuit should have a dedicated trace from the shunt resistor to the comparator. Additionally, try to route the traces away from high-current switching nodes to prevent noise induced overcurrent trips.

4.3 Testing and Optimization

Once new hardware is ready for testing, the IRMCx100 IC has registers that can be configured to help optimize the performance of the motor drive system. This section discusses the techniques to test critical aspects of the system and then set register values (or MCEWizard100 fields) based on the tests. The IRMCx100 provides both single and two shunt current feedback schemes, which are selected by the user configuration. Detailed descriptions of two shunt and single shunt current reconstruction are presented in Sections 4.3.1 and 4.3.2 respectively.

In addition to the tests described in this section, it is also important to verify the current and voltage feedback scaling as described in Section 2.2.4.

4.3.1 Space Vector PWM and Leg Shunt Current Sampling

IRMCx100 has a provision for leg shunt current measurement with the following properties.

- Sampling occurs in the middle of the zero vector V000.
- Full current information is obtained by sampling each current once per PWM period.
- A guard band is required to guarantee current samples at high modulation.
- It is not necessary to have a minimum pulse width.
- The voltage across the shunt resistor is inverted compared to single shunt measurement.

Circuit diagrams of two shunt current measurement are shown in Figure 60.

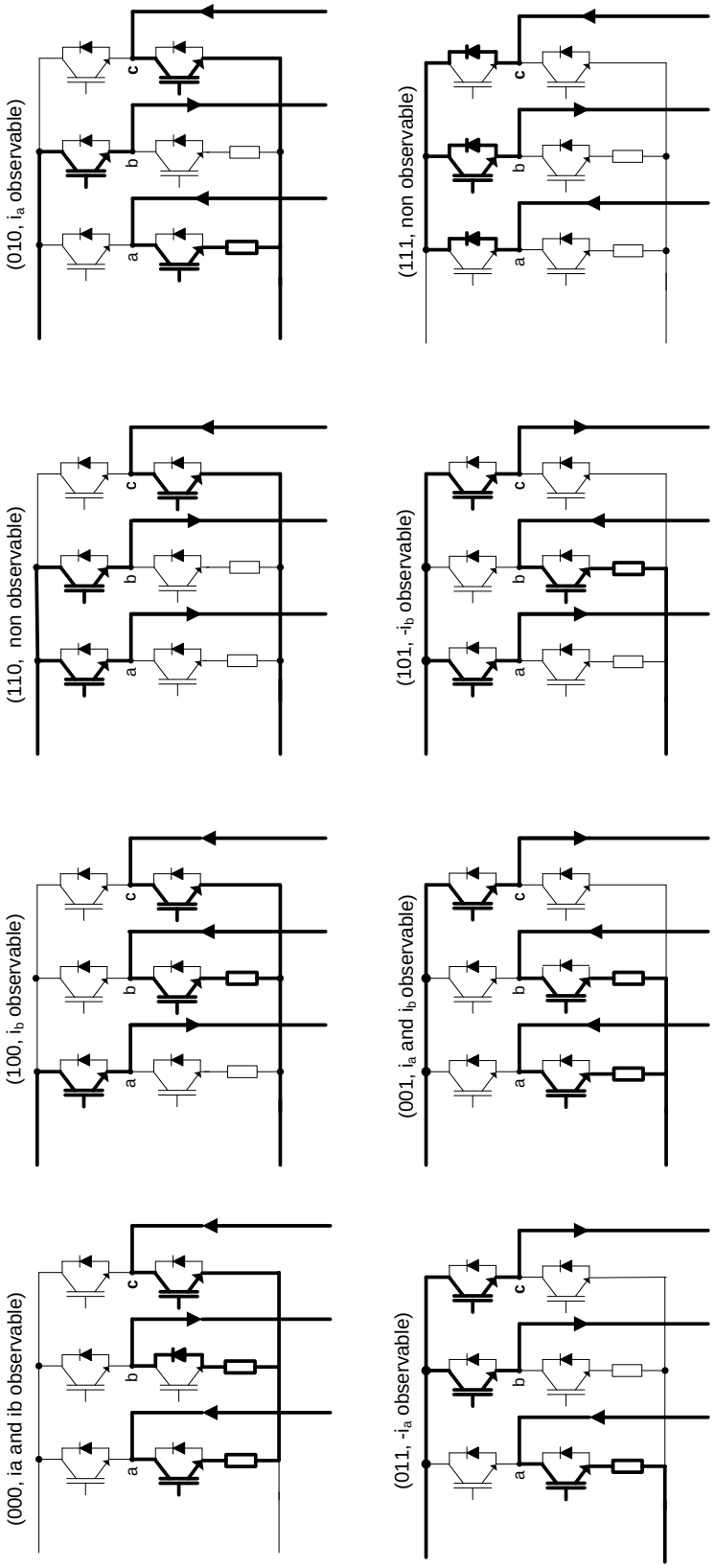


Figure 60—Two Shunt Current Measurement

As mentioned above, sampling of both currents must happen in the middle of the zero vector V000 as shown in Figure 61. By sampling at this instant, the true average current is sampled and unless the modulation index is approaching 1 there is plenty of time to sample. Because of the propagation delay of the HVIC driver and power device's turn on/off delay in the real hardware, the current sample instant should also include this delay. In MCEWizard100, input the delay at "Gating Propagating delay", normally set to 0.6us in IRMCS1043 and IRMCS1271.

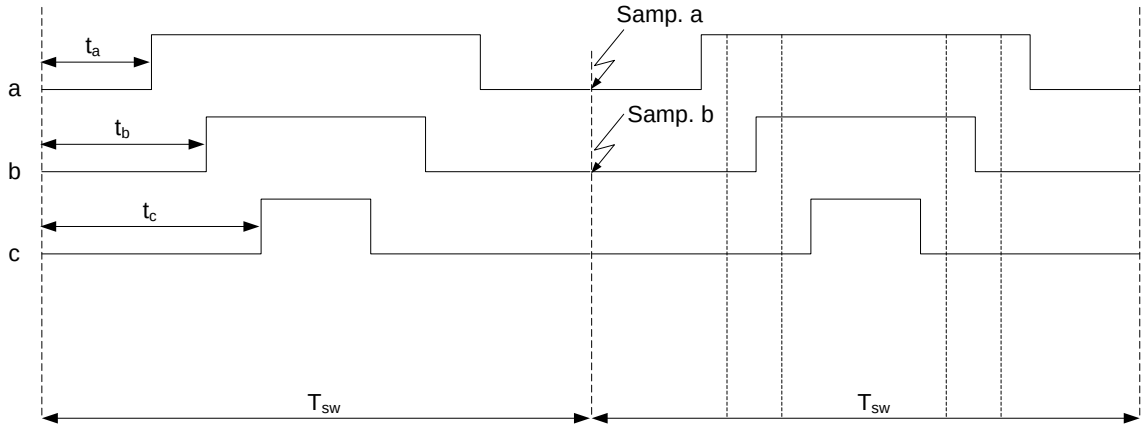


Figure 61—Sampling of Leg Shunt Currents

When the modulation index is close to unity, V000 is very short, leaving little or no time to sample leg currents. A consequence is that the non-observable region is as shown in the figure below.

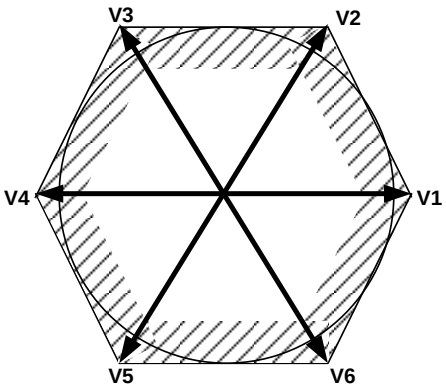


Figure 62—Non-observable Region with Leg Shunt.

To ensure that the currents can be observed, set a Guard Band in MCEWizard100. Figure 63 below shows how the Guard Band is defined. Note that the Guard band is applied at the beginning and end of the PWM cycle. This is normally set from 0 – 2us, where 0us is used only where there is some DC bus headroom and no field weakening operation.

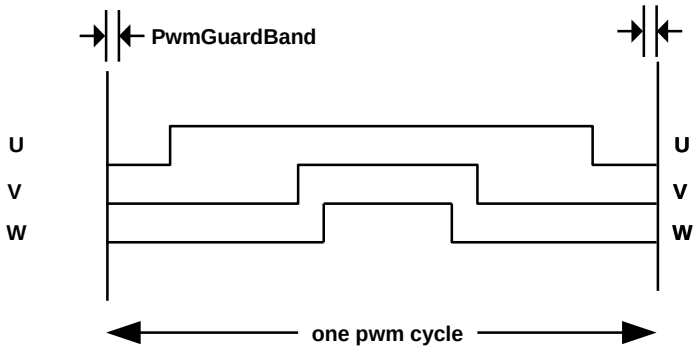


Figure 63—Guard Band Insertion

4.3.2 Space Vector PWM and Single Shunt Current Reconstruction

The IRMCx100 Series IC can use a single shunt current reconstruction circuit and methodology to minimize external analog and digital circuitry. In order to implement sensorless field oriented control, it is crucial to measure the motor winding currents precisely. The single shunt current reconstruction method derives all necessary current feedback by sampling the currents in the shunt resistor, thus eliminating the need for isolation circuits or magnetic current sensors. The space vector modulator generates sample timing signals based on the power inverter state. The IC integrates the A/D converter and amplifier to sample the voltage across the shunt resistor. Under certain operating conditions the DC link current pulses may become too narrow to guarantee reliable extraction of winding current data.

Space vector modulation is a technique to generate the three-phase power inverter switching signals based on the desired three phase voltage output. Each leg of the power inverter can connect the load to either the positive or negative DC bus. In one active inverter state, the switches connect one winding to the positive rail and the other two windings to the negative rail. In the example presented here, 2/3 of the bus voltage is across one winding and 1/3 of the voltage is across the other phase windings. In another active state, the switches connect two windings to the positive rail and the other winding to the negative rail. In the zero vector states, the switches connect all three windings to either the positive or the negative rail.

Figure 64 shows the six active vectors and two zero vectors (V0 – V7) available using three inverter switches. It also shows how switching between two active inverter states can produce any specified inverter voltage. For example, to produce voltage V* in the sector 1, the inverter is in state V1 for time Ta and in state V2 for time Tb. The inverter is in a zero vector state for the time remaining in the switching period. Typically half of this time (T0) is in the V0 state at the beginning of the cycle and the other half of the time is in the V7 state at the end of the cycle. Figure 65 shows the resultant inverter switching signals where voltage vectors V0, V1, V2 and V7 are applied for time periods T0, Ta, Tb and T7. Applying these voltage vectors in the inverse sequence in the second half of the PWM cycle generates symmetrical PWM signals. Since V* is closer to V1 (which is aligned with U phase), V1 is applied for a longer time than V2 (Ta>Tb).

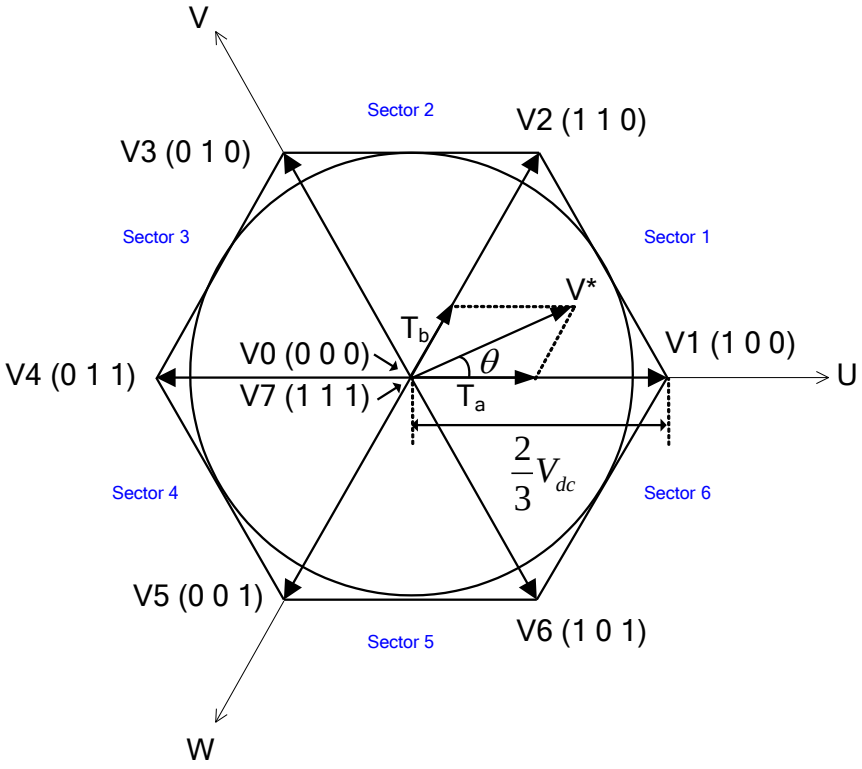


Figure 64—Inverter Output Voltage Space Vectors

A real 3-phase inverter uses a combination of transistors (IGBTs or MOSFETs) and anti-parallel diodes as the power switches, as shown in Figure 58. A high voltage integrated circuit provides level shifting between the logic level signal from the digital control IC and the transistors, which switch between the positive and negative DC bus. The polarity of the 'on' signal may be active high or active low depending on the design of the gate drive HVIC. There must be a delay, or "dead time," between the high side turn-off signal and the low side turn-on signal. This allows the high side power transistor to turn off completely before the low side transistor turns on (or vice versa) to avoid a shoot-through condition that can damage the power devices. The actual gate drive signals from the control IC include the dead time between all inverter state transitions, so there are six inverter switching signals: PWMUH through to PWMWL in Figure 65; in this case, the gate drive circuit accepts active low logic inputs. The modulation circuit typically inserts the dead time but the gate drive circuit can also provide this function. Active high/low gate logic selection is available through a control register, activepol, on the IRMCx100.

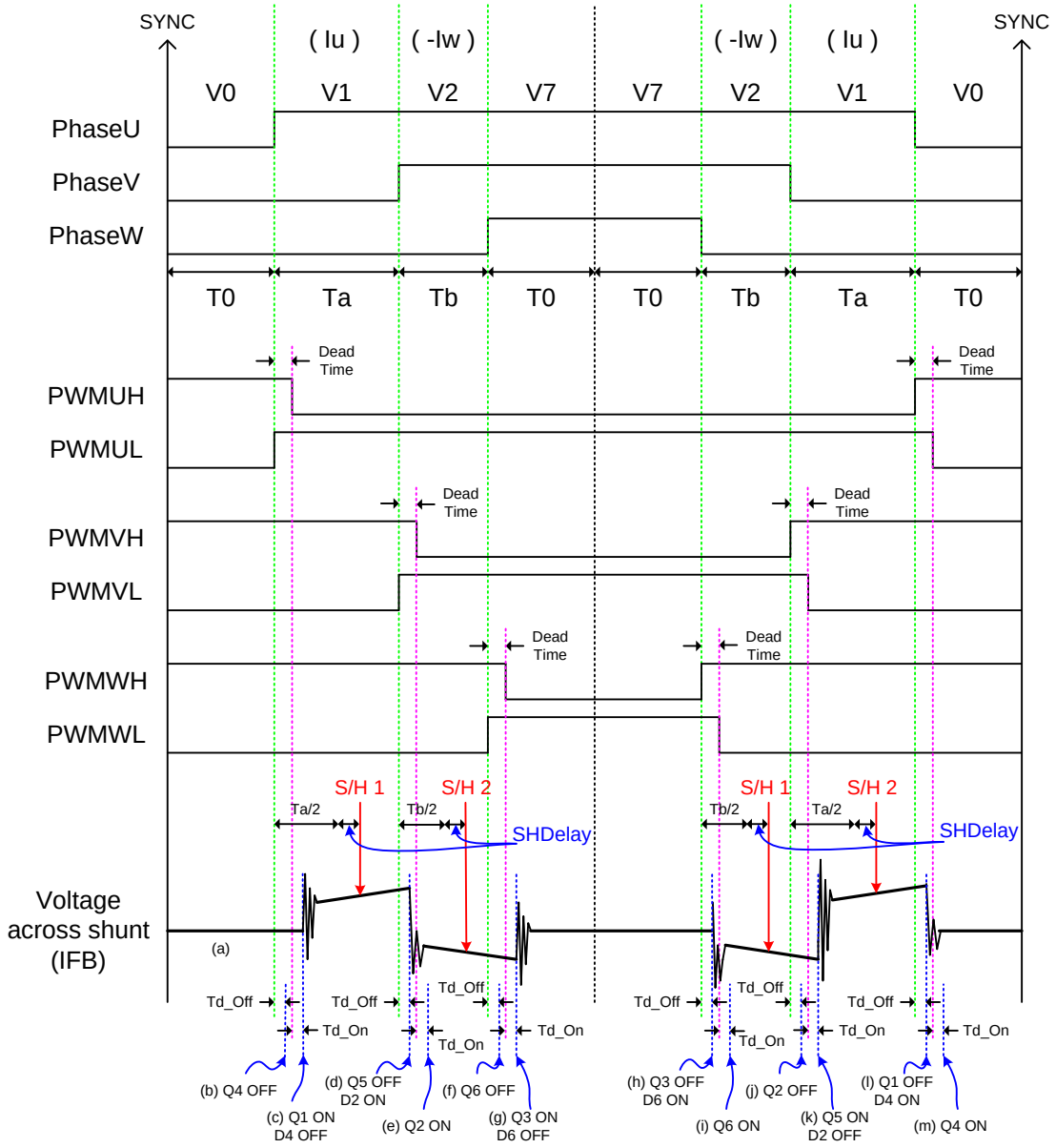


Figure 65—PWM Gate Signals in Sector 1

The motor current reconstruction circuit measures the DC link current in the shunt resistor during the active vectors of the PWM cycle. When the voltage vector V1 is applied, current flows from the positive rails into the phase U winding and returns to the negative rail through the V and W phase windings. In this instance, the DC link current flowing from the positive rail equals the U phase current. When the voltage vector V2 is applied, the DC link current returning to the negative rail equals the W phase current. Therefore, in each sector, two phase current measurements are available. The calculation of the third phase current value is possible because the three winding currents sum to zero.

4.3.3 Inverter-Related Testing and MCEWizard100 Settings

The 3-phase inverter is the major subsystem of the motor drive hardware. It is important to configure the controller with the correct values for inverter dead time and current sample timing

(for example) to get the best performance from the inverter hardware. This section describes registers and settings to align the controller with the inverter.

4.3.3.1 Miscellaneous MCEWizard100 Settings

Gate Drive Hardware section—Check the specifications of the gate driver IC to set these MCEWizard100 fields correctly. These parameters set the logic sense for the PWM gating signals. **The GATEKILL input is always active low (low true) in IRMCx100 ICs**.

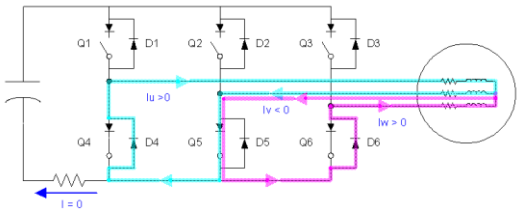
Inverter Dead Time—Dead time is used to prevent shoot-through, a condition where both the high side and low side IGBTs are on at the same time, which can damage the inverter components. To choose the dead time, carefully check the turn-on and turn-off times of the IGBTs. Also, the gate driver IC delay matching should be taken into account.

Bootstrap Capacitor Charge Pulse Width and Delay—These parameters govern the bootstrap pre-charging process for the gate driver IC. For more information, see the Reference Manual.

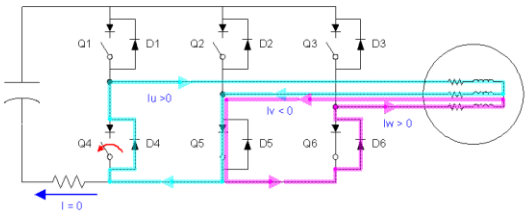
4.3.3.2 Current Feedback Sample Timing

The current sampling instant should be at the midpoint of the active space vector state to sample the average current. This sample instant for the first current sample is at time Ta/2 after the start of the first active vector V1, as shown in Figure 65. The space vector modulator calculates this timing when it calculates the timing for the gate drive signals. In a symmetrical PWM scheme, there are also two active vectors in the second half of the cycle and so two sets of current measurements are available. Averaging of the two sets of measurement improves the reliability of the current feedback.

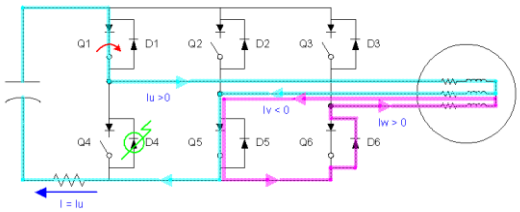
Successful implementation of motor current reconstruction requires detailed knowledge of power inverter operation to account for circuit delays that can result in incorrect current sampling. The error introduced by sampling delays depends on the magnitude of the motor current ripple, which depends on the bus voltage, switching frequency winding inductance and motor back emf. The IRMCx100 includes a sampling delay register, SHDelay that allows the system designer to compensate circuit delays to ensure accurate current measurement. The voltage across the DC link shunt resistor, IFB, in Figure 65 illustrates how to calculate the sampling delay compensation. Figure 66 illustrates current flow associated with the dead time and each switching instance to display the change of current path and reverse recovery current from the diode. In this example, Iu > Iw > 0 > Iv and the IGBT is modeled as a switch with a diode. Depending on the current direction, sometimes turning on or off the switch doesn't change the current flow. A thunder mark on a diode indicates the reverse recovery action of the diode.



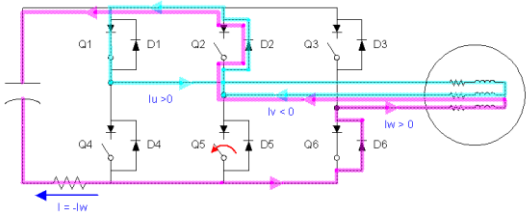
(a) Zero Vector V0



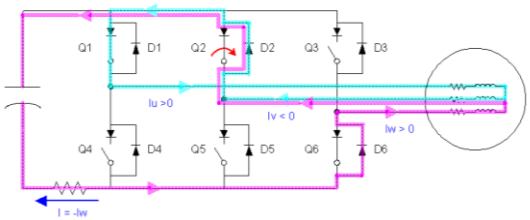
(b) Q4 OFF



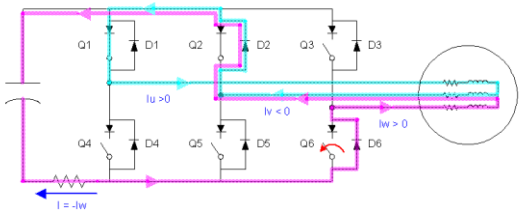
(c) Q1 ON, D4 OFF



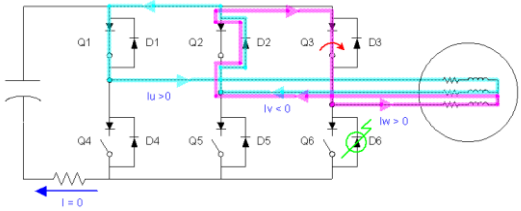
(d) Q5 OFF, D2 ON



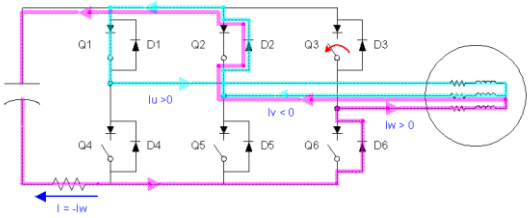
(e) Q2 ON



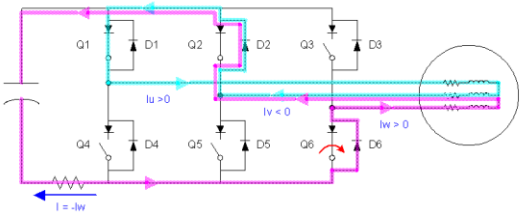
(f) Q6 OFF



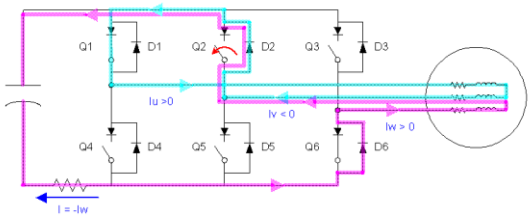
(g) Q3 ON, D6 OFF



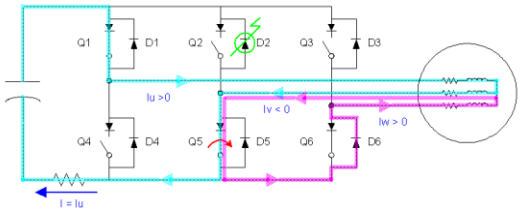
(h) Q3 OFF, D6 ON



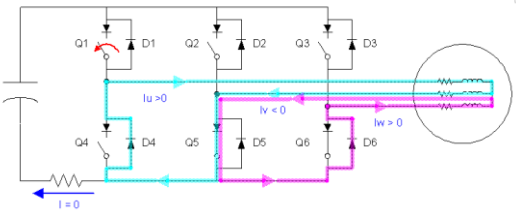
(i) Q6 ON



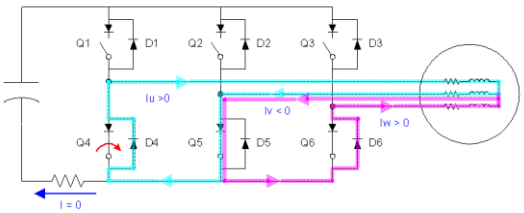
(j) Q2 OFF



(k) Q5 ON, D2 OFF



(l) Q1 OFF, D4 ON



(m) Q4 ON

Figure 66—Current Flow Example in Sector 1

There is a delay between gate driver IC input and output, and another delay from gate driver output to real switching instance of the device such as IGBT. This is a function of gate charge and gate impedance. Td_On and Td_Off in Figure 65 are the sum of these two delays respectively. For example if the gate driver delay is 400ns and the IGBT turn on and off delays are 190ns and 700ns respectively, then:

Td_On = gate driver delay + transistor turn on delay = 400 ns + 190 ns = 590 ns
Td_Off = gate driver delay + transistor turn off delay = 400 ns + 300 ns = 700 ns

There is a limitation such that an active vector must exist for a minimum time to ensure a reliable sampling of the DC link current. This minimum time is set by the MCE registers TcntMin3Phs for three-phase modulation and TcntMin2Phs for two-phase modulation. This lower bound on the minimum time results in a limitation when the modulation index is small (small voltage) or the voltage vector passes an active vector. The areas where problems exist are highlighted in Figure 67.

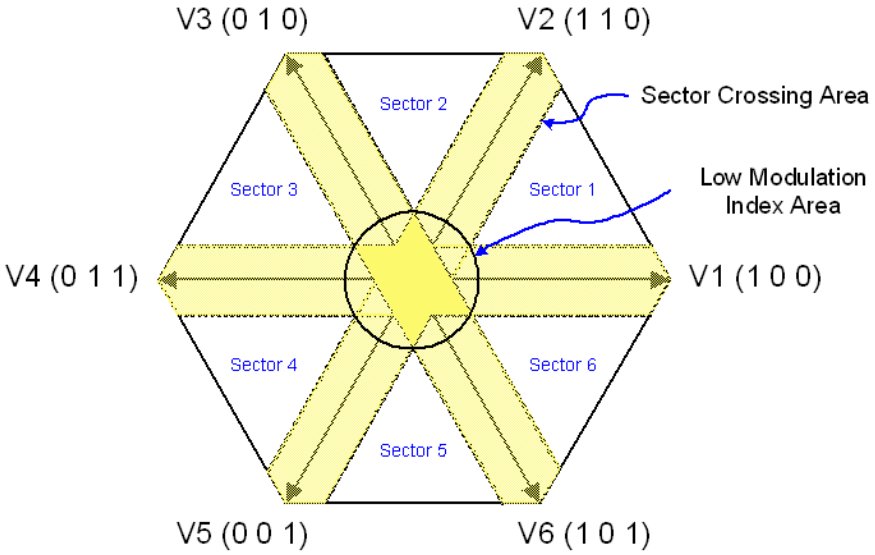


Figure 67—Areas Where Reliable Sampling is Difficult

The minimum time required for reliable current sampling adds an undesired voltage distortion, which may cause audible noise especially in low speed operation. In order to minimize this time, it is important to understand when sampling occurs. Ideally, the current sample should occur at the center of the active vector, which results in an average value of the current regardless of the slope related to motor inductance. However, as discussed previously, actual switching happens after a certain period of time from the edges of PhaseU, PhaseV, and PhaseW. This delay can

be as short as Td_Off and as long as dead time plus Td_On. Sampling timings can be adjusted using the SHDelay register such that sampling occurs at one half of the active vector time plus SHDelay after the edges of PhaseU, PhaseV, and PhaseW. For example, in Figure 65, the first sampling instance is Ta/2 plus SHDelay after the rising edge of PhaseU. The real switching instance occurs either Td_Off or Td_On plus dead time after the edge of PhaseX. Thus, SHDelay can be set to cover worse case as follows:

SHDelay = Td_On + dead time **(1)**

Since sampling should be done after the ringing settles down even in the case of minimum pulse, a condition for sampling delay from the PhaseX edge can be derived as below.

minimum pulse /2 + SHDelay > dead time + Td_On + ringing time **(2)**

The left hand side is the sampling delay in the case of an active vector with minimum pulse and the right hand side is the actual delay time required to sample without noise.

From (1) and (2), the minimum pulse can be derived to be:

minimum pulse > 2 * ringing time **(3)**

Remember that (1) to (3) are the mid-point sampling case. If the slope of the current is not steep, delaying the sample instance further to the end of the active vector can reduce the necessary minimum pulse. Because the switching of the next "PhaseX" edge also has at least Td_Off (or sometimes even dead time plus Td_On) to have a real switching instance, the minimum pulse can be as small as the following equation:

minimum pulse = dead time + Td_On + ringing time - Td_Off **(4)**

This can be put into (2) to get the proper SHDelay.

(dead time + Td_On + ringing time - Td_Off) / 2 + SHDelay
> dead time + Td_On + ringing time **(5)**

SHDelay = (dead time + Td_On + ringing time + Td_Off) / 2 **(6)**

If the motor inductance is small and the sampling should be done at the center, Equation (3) and (1) should be used to get the minimum pulse and SHDelay. If the application requires a shorter minimum pulse and slope of the shunt voltage is not steep due to a relatively high inductance of the motor or small DC bus voltage, then equation (4) and (6) can be used. Keep in mind that Td_On and Td_Off can vary depending on the operating condition.

Note that in MCEWizard100 there are two input parameters the sum of which determines the value of SHDelay—"Inverter Dead Time" and "Gating Propagation Delay." Set the Inverter Dead Time to the desired dead time, and then set Gating Propagation Delay according to equation (1) or (6) depending on the application conditions.

4.3.3.3 An Example of Optimizing the Current Feedback

Figure 68 shows the real waveforms for V* in sector 1 of Figure 64. Channel 1 is voltage across the shunt resistor and the others are low side gate signals (active low case). U phase current (Iu) is positive during vector V1 and negate of W phase current (-Iw) is negative during vector V2, which means W phase current is positive.

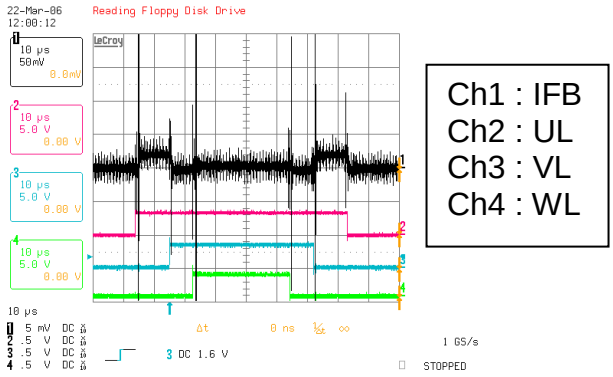
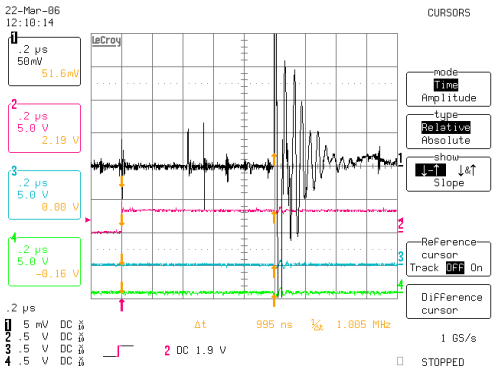
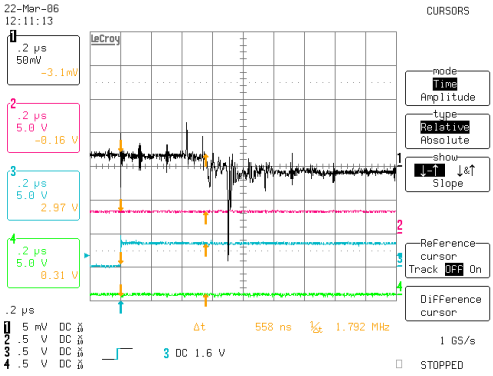


Figure 68—Waveforms in Sector 1

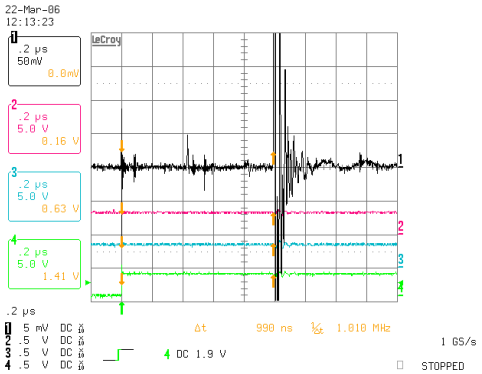
Figure 69 is a collection of waveforms when active vector changes. This figure can be better understood together with Figure and Figure 66. It can be observed that ringing is most severe at the transition from V2 to V1, in which case the largest amount of current is flowing through D2 and therefore reverse recovery is also most significant.



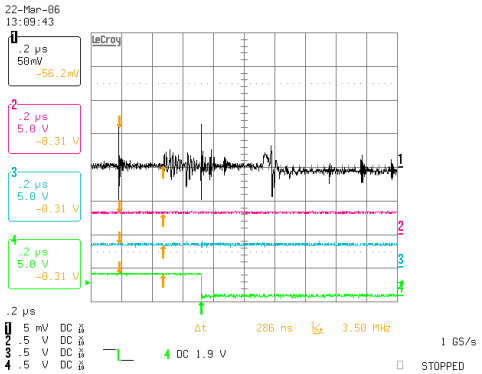
(a) V0 to V1



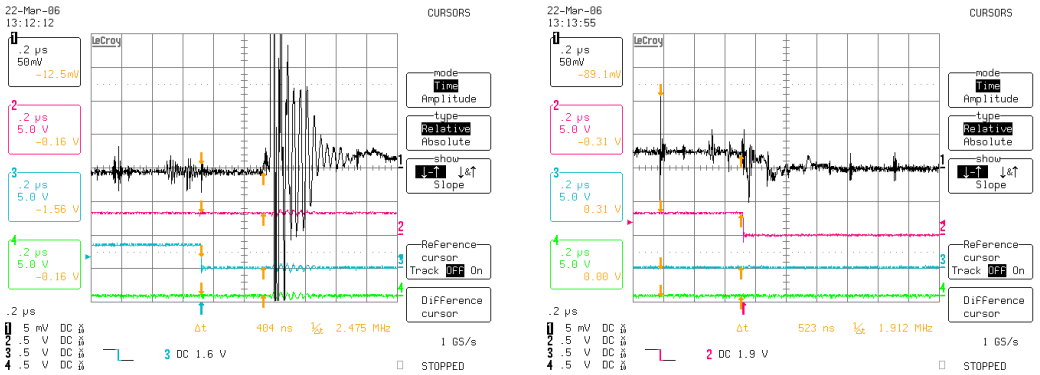
(b) V1 to V2



(c) V2 to V7



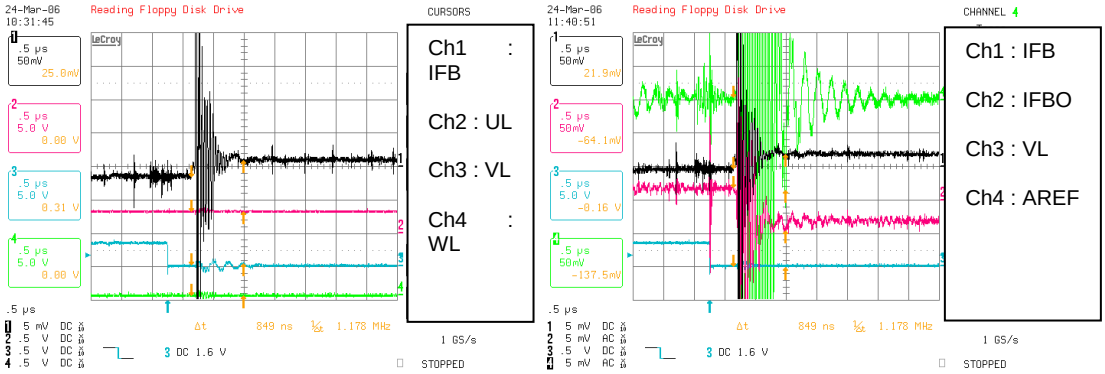
(d) V7 to V2



(e) V2 to V1 (f) V1 to V0

Figure 69—Vector Transition Waveforms in Sector 1

The waveform in Figure 70(a) is captured to measure the longest ringing time at the transition from V2 to V1. High frequency noise stops within 0.6 µsec, but there is also slow component which ends in 0.85 µsec. However, the operational amplifier output (IFBO) is the one that needs attention here because this is the input to the sample & hold. Figure 70(b) shows IFBO and AREF together with IFB. Some slow noise components in AREF are reflected into IFBO. This ripple ends in 1.25 µsec.



(a) IFB at V2 to V1 (b) IFBO and AREF

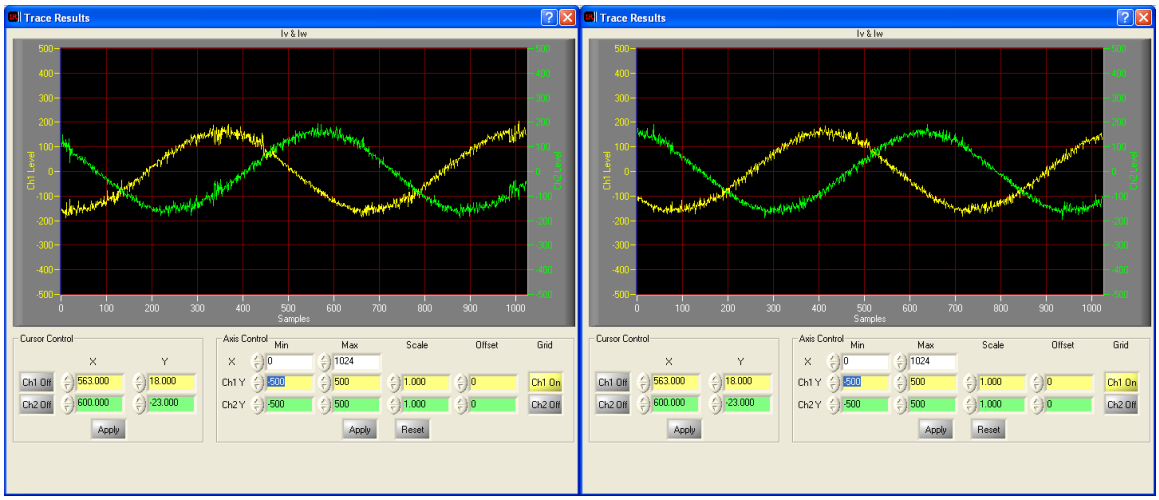
Figure 70—Ringing at Transition from V2 to V1

When dead time is set to 500 ns, equation (1) and (3) give:

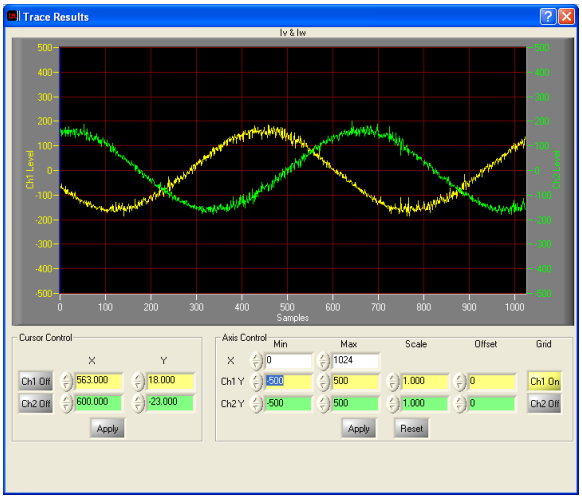
SHDelay = Td_On + dead time = 590 ns + 500 ns = 1.1 µsec

minimum pulse = 2 * ringing time = 2 * 1.25 µs = 2.5 µsec

Figure 71(a) is a trace buffer plot from MCEDesigner for this case. Figure 71(b) is a plot when SHDelay is 1.1 µsec and minimum pulse is 1.5 µsec. Some glitches exist due to slow ripple component.



(a) SHDelay 1.1µs and MinPulse 2.5 µsec (b) SHDelay 1.1 µs and MinPulse 1.5 µsec



(c) SHDelay 1.5 µs and MinPulse 1.7 µsec

Figure 71—Phase Current Plot from Trace Buffer

From equation (4) and (6),

minimum pulse = dead time + Td_On + ringing time - Td_Off
= 0.5 + 0.59 + 1.25 - 0.7
= 1.64 µsec

SHDelay = (dead time + Td_On + ringing time + Td_Off) / 2
= (0.5 + 0.59 + 1.25 + 0.7) / 2
= 1.52 µsec

Figure 71(c) is a plot for this case and seems as good as 62(a). These plots in Figure 71 verify equations (1) to (6).

4.3.4 Overcurrent Protection

The overcurrent protection circuit prevents damage to the motor and inverter by shutting down the PWM switching outputs of the control IC when the current across the DC link shunt resistor reaches some threshold level. When verifying new hardware, the overcurrent protection circuit should be tested early in the process.

To test the overcurrent protection circuit, begin by verifying that the comparator input voltages are at the expected values when the hardware is powered up. Also calculate the expected trip current. If possible, use a DC current source to run current through the shunt resistor and test the overcurrent trip level.

Attach current probes to the motor phases. Connect an induction motor to the hardware and run the “VF Diagnostic” function. To test the trip level, increase the motor current by gradually increasing the value of VFGain register until a GATEKILL fault occurs. To capture the current waveform at moment of the fault, trigger on the GATEKILL signal.

As a final, potentially destructive test, short circuit the motor phases; be sure to have current probes on each phase and to be triggering on the GATEKILL signal with the oscilloscope. Start the motor, and an overcurrent trip should occur immediately. Check the delay time from the current reaching the threshold to the trip occurring. Verify that this shut down time is fast enough to protect the transistors of the 3-phase inverter. Modify the GATEKILL circuit capacitances and/or the value of GkillFiltCnt to adjust the shutdown time.

5 PFC Application Development

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 situations. 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 effectiveness of the PFC.

The IRMCx143 & IRMCx188 includes op-amps for sensing the AC input voltage and PFC current; along with the DC bus, this sensing allows the MCE to perform digital PFC control. The Reference Design Kit for this part number includes an IR-supplied MCE program to perform the PFC, described in Section 5.1. Figure 72 below shows the simplified topology of the boost PFC control employed in the reference design kits, with the necessary sensing parameters labeled.

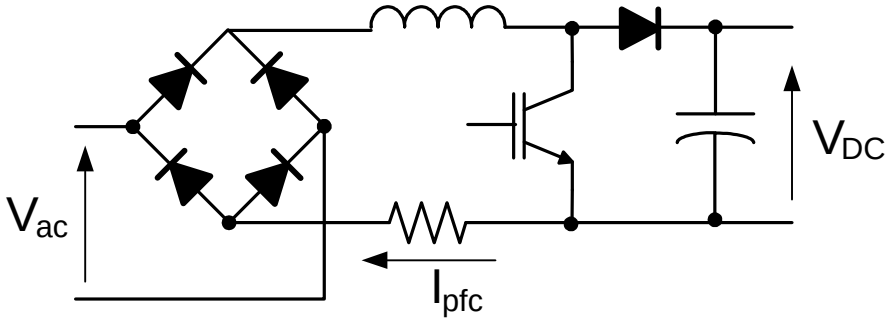


Figure 72—Basic PFC Circuit

The first section of this chapter describes the MCE program which does the digital PFC control including the structure of the control loops and the configuration parameters. Section 5.2 gives guidelines for PFC inductor measurement. Next, Section 5.3 gives basic information and instructions for using the PFC as implemented in the IRMCS1088 Reference Design Kit. It also describes some hardware modifications, tuning and optimization techniques and advanced capabilities of the PFC control. Section 5.4 continues with guidelines for hardware design, optimization and testing of the PFC as related to the IRMCx143 & IRMCx188 IC. Finally Section 5.5 describes full mode PFC and partial PFC application.

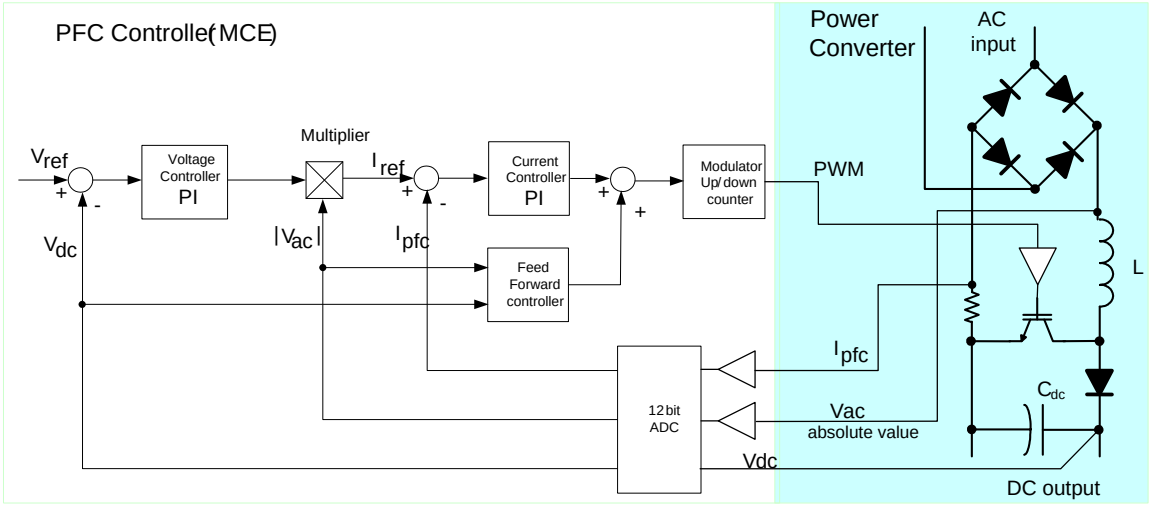


Figure 73—PFC Control System

Figure 73 provides a high-level block diagram of the PFC control scheme, as implemented in the IRMS1043 Reference Design Kit. The digital control portion on the left is implemented in the MCE program, while the portion on the right with the blue shading represents the PFC and DC bus hardware components.

There are two control loops in Figure 73, an inner current loop and an outer voltage loop, along with a feedforward (FFW) 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 PFC current, AC line voltage and DC bus voltage.

To view parts of the PFC controller in detail, a PDF version of the Simulink Model file is part of the files installed with the reference design kit (in My Documents\iMotion\IRMCS1043). Each component of the controller is described fully in Section 5.1.

5.1 PFC MCE Program

5.1.1 Current Loop

A simplified block diagram of the current loop is shown in Figure 74.

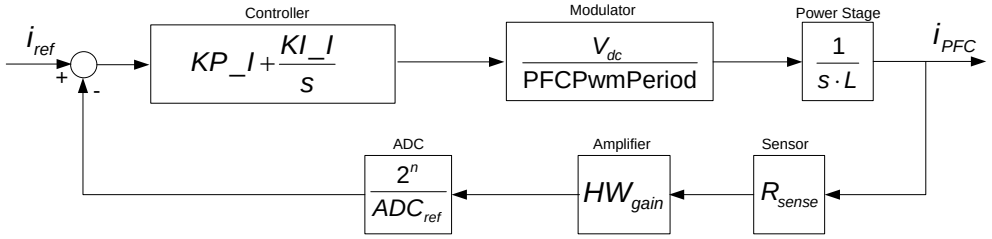


Figure 74—Simplified Current Loop

The controller gains, KP_I & KI_I, are automatically configured by MCEWizard100 for a bandwidth of 1400Hz. This bandwidth is chosen because it satisfies the control requirements without excessively amplifying current feedback noise. Also, the achievable bandwidth of the loop is limited by the control delay of 0.5 to 1 PWM cycle and the PFC PWM frequency. In order to keep a constant loop gain, based on the diagram above, the current controller PI gains scale as:

$$KP\_I \propto \frac{L \cdot}{R_{sence} \cdot A_{IPFC} \cdot PFCPwmFrequency}$$

$$KI\_I \propto \frac{L \cdot}{R_{sence} \cdot A_{IPFC} \cdot PFCPwmFrequency}$$

where A_{Ipfc} is the current sense feedback gain which includes the resistor divider and A/D converter gain and R_{sence} is the current sense resistor.

The complete current loop implementation in the MCE program is shown in Figure 75. The PFC current amplitude, which is output from the PFC voltage loop, is a quasi-DC signal at steady-state power. The current amplitude is multiplied by the instant AC voltage and then rescaled to obtain the PFC current reference (I_Ref), containing both amplitude and phase information for the desired PFC input current. I_Ref is compared with the real PFC input current to get the current error (I_error) which is sent to a PI regulator. The output of the PI regulator generates only a part

of the PWM command, since the PI current regulator BW (about 1.4KHz BW is setup by MCEWizard100) is not high enough for perfect current tracking. The PI regulator BW is limited by the PFC frequency (for example, 18KHz supports a maximum BW of 1.8KHz) and the PFC current feedback noise (which depends on the PCB layout).

The remainder of the PFC PWM command is generated by a feedforward block, lowering the burden on the PI regulator. The sum of FFD and current PI output is limited to [0 ~ Limit_P_I, MAX range 0~16383] for protection, then a two-bits shift (4x) to fit the input range of PFC PWM block ([0~65535] corresponding to 0~100% duration).

Inside the PFC_PWM block, there is a PFC state machine, current & voltage and minimum pulse blanking. For more information, please refer to the IRMCx100 Reference Manual.

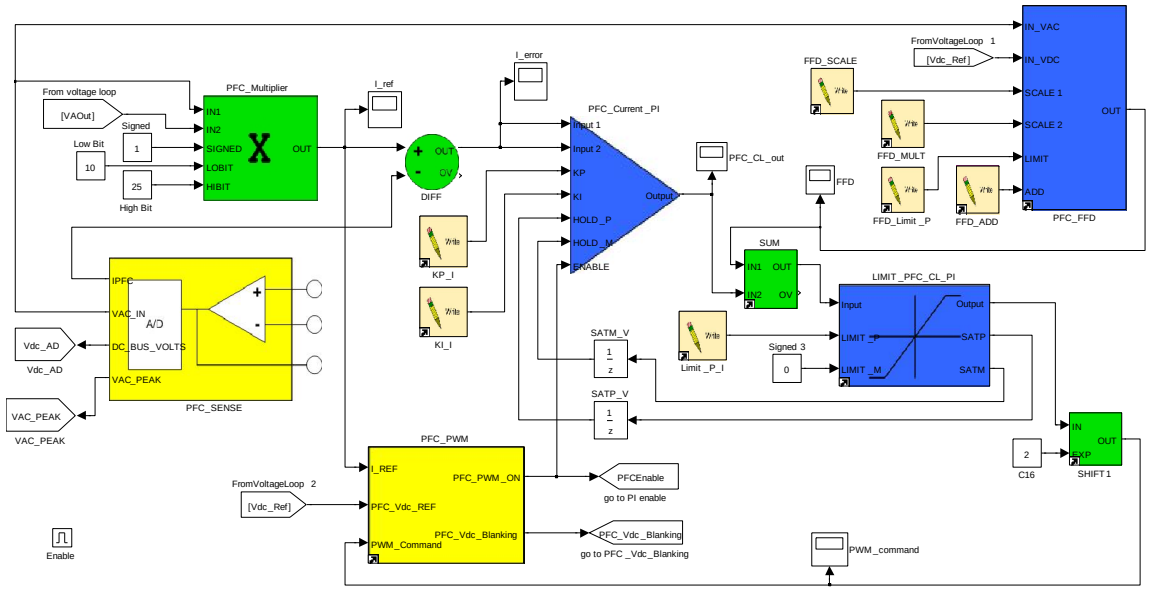


Figure 75— MCE Implementation of the PFC Current Control Loop

5.1.2 Voltage Loop

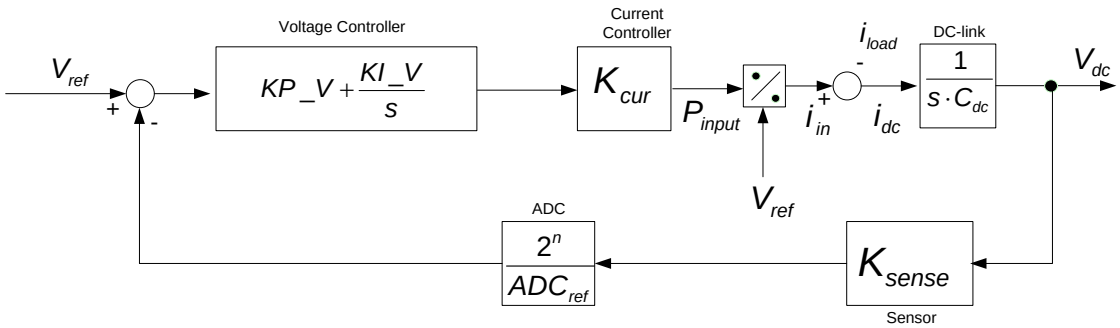


Figure 76—Simplified PFC Voltage Loop

Figure 76 gives a simplified block diagram of the voltage loop in the s-domain. The current controller is represented as a gain component; this can be done because the current regulator bandwidth is much larger than the bandwidth of the voltage loop. MCEWizard100 configures the voltage controller gains, KP_V & KI_V, in order to achieve a bandwidth of about 10 Hz. Note that

the PFC voltage loop is executed at the motor PWM frequency as defined in the options page of MCEWizard100. In order to keep a constant loop gain, from the diagram above, the voltage controller PI gains scale as:

$$KP\_V \propto \frac{C_{dc}}{A_{Vdc} \cdot A_{Vac}}$$

$$KI\_V \propto \frac{C_{dc}}{A_{Vdc} \cdot A_{Vac}}$$

where A_{Vdc} and A_{Vac} are the dc bus and ac input voltage feedback gains, respectively, which include the resistor divider and A/D converter gain. The ac voltage feedback gain appears here because the output of the voltage regulator is multiplied by the rectified ac voltage.

The MCE program implementation of the PFC voltage loop is shown in Figure 77. To follow the PFC voltage loop, locate the Vdc_Ref register in the upper left corner of the diagram. SWITCH5 selects between DC bus reference signals for full and partial PFC modes—Vdc_Ref or (Vac_PEAK – VdcHyst), respectively, selected by the write register PartialPFC_ON. A RAMP block prevents the DC bus reference from changing discontinuously before it passes to the PFC Voltage Loop as input Real_PfcDCRef. The DC bus voltage should normally ramp for 100 – 200ms to rise to the steady-state value. The input to the PI regulator is the difference between Real_PfcDCRef and Vdc_AD while the output of the PI is limited to [0 – Limit_P_V] for protection, and then sent to the PFC current loop.

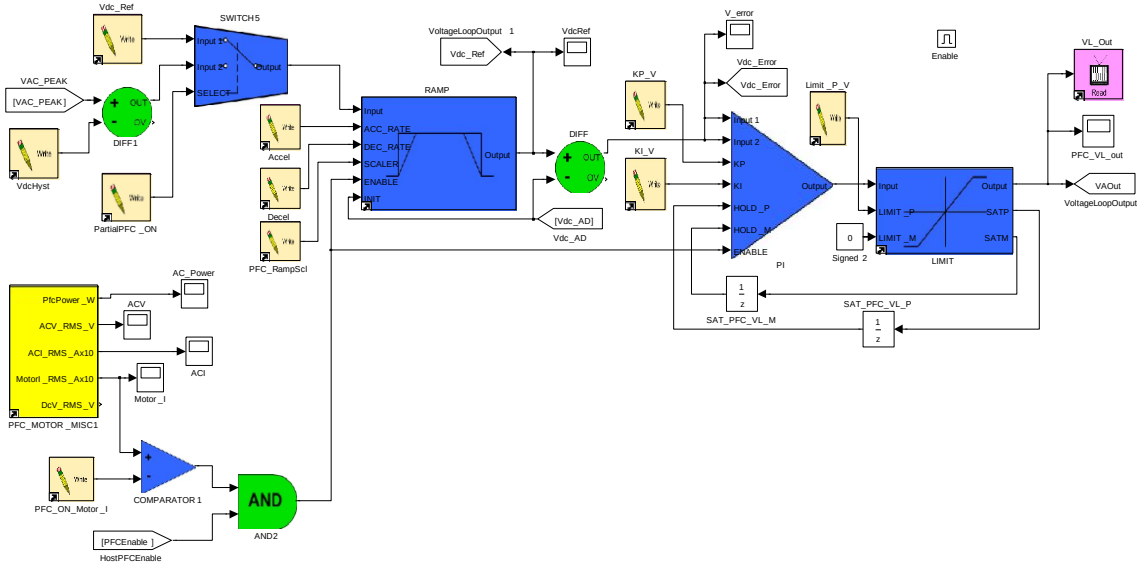


Figure 77—MCE Implementation of the PFC Voltage Control Loop

The PFC voltage loop is enabled mainly by the PFCEnable write register and PFC state machine. However, the PFC_ON_Motor_I input register and COMPARATOR at the bottom left corner of the diagram serve to disable the PFC voltage loop when the motor current is lower than PFC_ON_Motor_I. MCEWizard100 configures PFC_ON_Motor_I = 0 by default so that the PFC can run independently of the motor status.

5.1.3 Feedforward

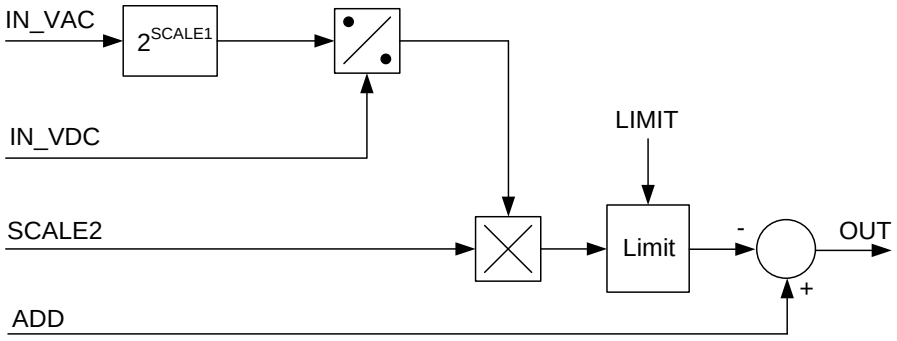


Figure 78—FFD Block Implementation

Figure 78 provides a block diagram of the internal operation of the PFC_FFD block. The input AC voltage is divided by the DC bus voltage, then scaled and limited. The result is subtracted from a dc component to produce the FFD signal. Figure 79 shows an example of the FFD output and the current regulator output (note that the traces have been scaled up by 4x to correspond to the modulation scaling where 65535 = 100% modulation). The FFD provides a large duty cycle command during the AC voltage zero crossing which reduces the demand on the current controller.

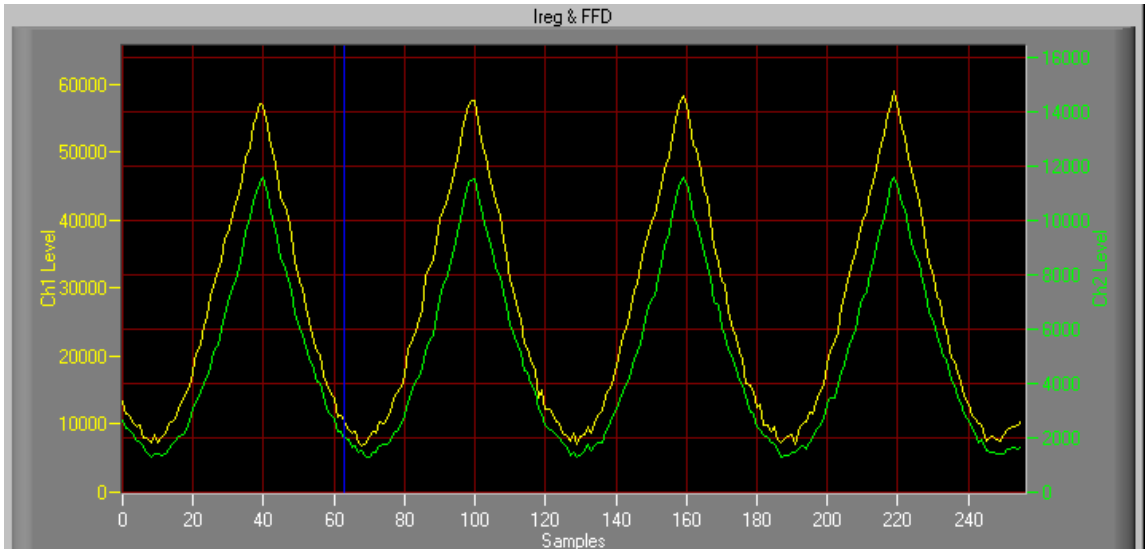


Figure 79—Current Regulator Output (yellow) and Feedforward Output (green)

Depending on the DC bus and AC Input voltage scalings, the FFD output can exhibit significant quantization if SCALE1 is set to too low a number. The quantization can lead to degradation of the PFC performance. This problem is avoided when using the IR default MCE program and configuring with MCEWizard100.

5.1.4 PFC State Machine

A simple state machine manages the PFC related input and outputs which deal with the following items:

1. Disable/Enable AcV&AcI RMS and PFC power calculation according to MtrCtrlBits_S bit7.
2. Disable/Enable AC status handling (AC OV & UV detect) according to MtrCtrlBits_S bit8.
3. Check for completion of the IPFC offset correction.
4. Check for motor or PFC faults.
5. Vac peak voltage detection.
6. Vdc blanking signal generation if Vacpeak > VdcBus.
7. Check PfcEnable command and decide whether to set PfcPwmOn flag to enable PFC PWM.

PfcPwmOn indicates the final PFC PWM status:
PfcPwmOn = 1 when PFCEnable = 1 and none of the conditions below are true.
PfcPwmOn = 0 when PFCEnable = 0. Or any one of following conditions is true:

1. PFCEnable = 0.
2. The PWM is disabled by the PFC voltage blanking (Vacpeak > VdcBus) function.
3. The PWM Command is less than the PFC Minimum pulse (PfcMinPulse).
4. AC input over or under voltage condition.
5. Motor or PFC fault is active.

The PFC_PWM_ON signal can be used in an MCE design or 8051 application code to enable, disable or reset the PFC control loop regulators (voltage, current, etc.), as is done in the reference design.

5.1.5 Input and Output Registers of the PFC

PFC Voltage Loop control Write Registers:

PartialPFC_ON
Scaling or Notation: 1 = select partial PFC mode
0 = select full PFC mode
Description: This parameter selects the PFC mode. Full PFC mode uses the Vdc_Ref register as the PFC DC bus reference, while Partial PFC mode uses (VAC_PEAK – VdcHyst). VAC_PEAK is updated every 100ms according to VAC voltage feedback. Please see section 5.5 for full PFC and partial PFC setup.

Vdc_Ref
Scaling or Notation: DC bus reference voltage = Vdc_Ref / DC bus feedback scaling
Range: 0 – 4095,
In IRMCS1043 hardware, DC bus feedback scaling = 8.27 cts/V
Description: This register provides DC bus voltage reference for full PFC mode when PartialPFC_ON = 0. Please ensure that Vdc_Ref is higher than Vac Peak for good PFC performance.

VdcHyst
Scaling or Notation: Voltage reference adjustment = VdcHyst / DC bus feedback scaling
Range: -512 – 512
In IRMCS1043 hardware, DC bus feedback scaling = 8.27 cts/V
Description: This register provides DC bus voltage reference adjustment for partial PFC mode when PartialPFC_ON = 1. It is subtracted from the AC voltage peak voltage to get the DC bus reference (a value of 10 – 20V is recommended).

Accel, Decel and **PFC_RampScl**
Scaling or Notation: **Accdel and Decel** Range: 0 – 32767
PFC_RampScl Range: 0 – 31.
Description: The ramp rate of the PFC Voltage command to the target DC bus voltage is controlled by these registers. Note that the ramp rate is also proportional to the voltage loop execution rate (which is the motor PWM frequency). MCEWizard100 configures these registers to set the Voltage ramp to ~900V/s. Do not change the value of PFC_RampScl while the PFC is running because it can cause the voltage command to change abruptly.

KP_V, KI_V
Scaling or Notation: Range: 0 – 32767
Description: These registers set the PFC Voltage loop PI block proportional and integral gains, respectively. MCEWizard100 calculates the values so that PFC voltage loop have a BW of about 10Hz.

Limit_P_V
Scaling or Notation: Range: 0 – 4095
Description: This register limits the value of the PFC voltage PI output (VL_Out) before it becomes the PFC current loop command. It depends on the MAX PFC Power, Min AC voltage and the PFC current feedback scaling.

PFC_ON_Motor_I
Scaling or Notation: Motor current threshold (A) = PFC_ON_Motor_I /10
Description: This register sets the Motor current threshold for PFC operation. The PFC will turn on only when motor current is larger than this threshold. Some applications may not need PFC at very low motor loads in order to reduce the switching loss and noise. If it is set to 0 the PFC will run regardless of the motor status.

PFC Voltage Loop control Read Registers:

VL_Out
Scaling or Notation: Range: 0 – Limit_P_V
Description: This register is the PFC voltage PI output, which is the current amplitude reference for the PFC current loop.

PFC Voltage Loop Traceable Parameters:

VdcRef
Scaling or Notation: DC bus reference voltage = DC bus feedback scaling
Description: This parameter is the PFC voltage RAMP output, which is the real voltage reference for the PFC voltage loop.

V_error
Scaling or Notation: Range: -4096 – 4095
Description: This parameter is the PFC voltage error, which is the difference between Real_PfcDCRef and the DC bus feedback.

PFC_VL_out
Scaling or Notation: Range: 0 – Limit_P_V
Description: This parameter is the PFC voltage PI output, which is the current reference for the PFC current loop and the same as read register VL_Out.

ACV, ACI, Motor_I, AC_Power
Scaling or Notation: AC input voltage (Vrms) = ACV
AC input current (Arms) = ACI
AC input Power (W) = AC_Power
Motor phase current (Arms) = Motor_I / 10
Description: These parameters are outputs of PFC_MOTOR_MISC: AC voltage, AC current, Power and motor current, respectively.

PFC Current Loop control Write Registers:

KP_I, KI_I
Scaling or Notation: Range: 0 - 32767
Description: These registers set the PFC current loop PI block proportional and integral gains, respectively. MCEWizard100 configures the PFC current loop values to give a BW of about 1.4KHz.

Limit_P_I
Scaling or Notation: Range: 0 – 16383

Description: This register limits the value of PFC current PI output (PWM_Command before it becomes the PFC PWM modulation.

FFD_SCALE, FFD_MULT, FFD_Limit_P, FFD_ADD

Description: These registers configure the PFC current loop feedforward block. The FFD improves PFC performance by reducing the sensitivity to PFC current sampling noise and the burden on the current loop. MCEWizard100 calculates the values based on the input of “FFD_Gain”.

PFC Current Loop Traceable Parameters:

I_ref

Scaling or Notation: Range: 0 – 4095

Description: This parameter is the real PFC Current Reference, which is the rescaled product of PFC voltage loop PI input and instantaneous AC voltage.

I_error

Scaling or Notation: Range: -4096 – 4095

Description: This parameters is the PFC current error, which is the difference between PFC Current reference and PFC current feedback.

PFC_CL_out

Scaling or Notation: Range: 0 – 16383

Description: This parameter is the PFC current loop PI output, which is added to the FFD output to produce the PFC PWM command.

PWM_command

Scaling or Notation: Range: 0 – 65535

Description: This parameter is the PFC PWM command to PWM block.

5.2 PFC Inductor Measurement

Similar to measuring the Motor Lq and Ld, the designer can measure the PFC inductance with an LCR meter when using an inductor other than the one supplied with the IRMCS1043 Reference Kit. However, measuring the correct PFC inductance is not straightforward because the result can vary depending on the measurement condition. Table 3 gives an example of an iron-core inductor measured at different voltages and frequencies.

Test Frequency	100Hz	1KHz	10KHz	20KHz	30Khz
0.1V test	3.76mH	3.69mH	2.5mH	1.91mH	1.30mH
1V test	3.89mH	3.79mH	2.53mH	1.96mH	1.31mH

Table 3—An Iron-core Inductor's Measured Value Varies with Test Condition

This effect is due to the nature of the magnetic core material. Eddy-current losses in an iron-core will vary with voltage and frequency, influencing the measurement result. Based on the data above, input an inductance value of 1.8 – 2mH in MCEWizard100 for 20KHz PFC operation.

To correctly measure the inductor value for the PFC application, follow the following procedure:

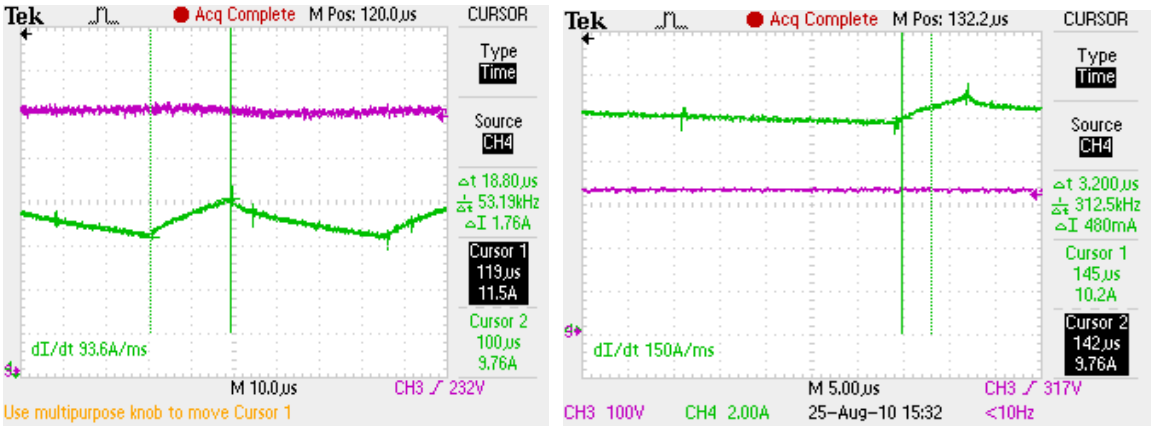
1. Begin by measuring the inductor with an LCR meter at 0.1V, 100Hz. Take 40% of this value for an iron-core inductor, 60% for a ferrite inductor, and input it into MCEWizard100.
2. Use the resulting drive parameters to run the PFC at nominal power and AC input voltage, setting the DC bus voltage slightly higher than the value to be used in the application.
3. Monitor the AC input voltage and inductor current on an oscilloscope, triggering near the peak of the AC input voltage.
4. Measure the AC voltage and di/dt of the inductor current to calculate the inductor value.

The value calculated using this method depends on the phase of the AC voltage, so be sure to trigger on the peak of the AC input voltage and then use the smallest value observed.
Figure 80 shows test results for a system run at a PWM frequency of 18kHz and input power of 1.5kW. The calculation below illustrates how the measured inductance varies with the instantaneous input voltage.

$$L_{test1} = U \cdot \frac{dt}{di} = 232V \cdot \frac{1}{93.6A/ms} = 2.47mH$$

$$L_{test2} = U \cdot \frac{dt}{di} = 317V \cdot \frac{1}{150A/ms} = 2.1mH$$

With this measurement, input a value of 1.6 – 2.1mH into MCEWizard100 in order to provide some design margin as required. A lower value makes the PFC current regulator less sensitive to hardware noise, but the PFC performance may be reduced since then the real bandwidth of the current regulator would be lower than 1.4KHz.



a. Trigger at 232V b. Trigger at 317V

Figure 80—Iron-core Inductor di/dt Test at 1.5kW Input Power,18kHz PWM Frequency

5.3 Using PFC on the IR Reference Board

At the initial power-up, the IRMCS1043 Reference Design Kit is already loaded with an MCE program that includes PFC control. Additionally, the MCEDesigner .irc file for the Reference Kit has the correct parameters to configure the PFC for the supplied hardware.

5.3.1 Using the Wizard to Create the Configuration Parameters

MCEWizard100 already contains the correct input values for the Reference Design Kit hardware selected on the Welcome Page. There are just a few parameters needed to configure the PFC controller for the specific application. The regulation voltage for the boost PFC is set with the Wizard parameter DC bus Voltage Reference. The **System DC Bus** section of questions also sets the over- and under-voltage trip levels as well as the scaling for DC bus sensing. Also, set the parameters in the **PFC Application** section of questions. These are all the PFC specific parameters which should be set if the Reference Design Kit is not modified.

If the designer chooses to modify the hardware, or create a custom board, then MCEWizard100 can generate the configuration parameters for the new hardware. Be sure to select “I have modified the circuit board” on the Welcome page of MCEWizard100 to enable the hardware dependent questions. See Section 5.3.6 for examples of simple modifications which can be made to the Reference hardware.

As with the motor, configure the PFC and then start PFC in MCEDesigner. IRMCx100’s PFC can start any time after configuration, regardless of motor status, and it supports both full PFC and partial PFC mode. Full PFC is the default mode. At light loads, the power factor will be low, but will rise to > 0.9 by the time the output power exceeds 150W.

PFC PWM can be set up at first MCE configuration after power on, and updated at the end of PWM Pre-charge step. The PFC PWM rate is not updated at any other time. In other words, the PFC PWM rate can be changed by configuring MtrCtrlBits in Motor stop state, but the real PFC frequency change only happens at motor start (updated at end of PWM Pre-charge step).

5.3.2 Overcurrent Protection Circuit

The control IC includes an input pin to shut off the PFC switch in case of an over-current event. In the Reference Design Kits, a PFC over-current situation is recognized, essentially, by comparing the voltage across the PFC shunt resistor with a reference voltage; Figure 81 shows the circuit from the IRMCS1088. The output of the comparator triggers the PFC gatekill fault in the control IC.

In the event of a gatekill fault, the PFC PWM pin will be immediately changed to the off state by hardware in the IRMCx188 IC.

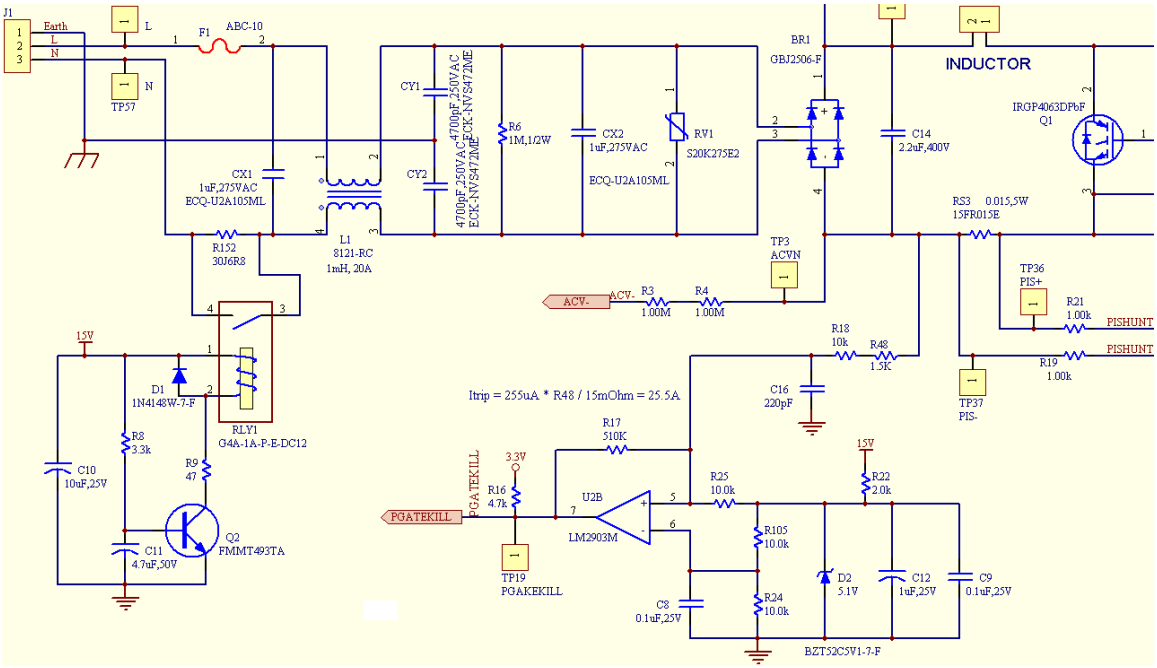


Figure 81—PFC Gatekill Circuit

5.3.3 PFC Variable Scaling

Voltage Loop:
The DC bus voltages, **Vdc_Ref** and **Vdc_AD**, have the same scaling as defined in Section 2.2.3.4:

*DC bus (V) = [Vdc_AD] / (A/D * r)*

Where:
A/D is the analog-to-digital converter scaling (3412/Volt, 4095→1.2V)
r is the voltage divider ratio used in the voltage sense circuit

AC Voltage Feedback—There are two parameters in the MCE program which can be traced—**VAC**, which is the rectified AC line voltage and **VacVoltsFilt**, which is the filted VAC. The scaling is calculated similarly to the DC bus feedback:

*AC Voltage = [VAC] / (A/D * k)*

Where:
A/D is the analog-to-digital converter scaling (3412/Volt, 4095→1.2V)
k is the op-amp gain of the voltage sense circuit, In the sensing circuit show in Figure 83, k = 4.87k / (4.87k*2 + 1.00M + 1.00M) = 2.423 x 10^{-3}.

Accel, **Decel**, and **PFC_RampScl** combine to set the voltage ramp rate as follows:

*Voltage Setpoint Ramp Rate (V/s) = ([AccelRate] / 2^RampScaler) * PWM Freq / (A/D * r)*

where
PWMFreq is the PWM switching frequency in *Hz*
A/D is the analog-to-digital converter scaling (3412/Volt, 4095→1.2V)
r is the voltage divider ratio used in the voltage sense circuit

Current Loop:

IPFC (PFC current feedback)

*PFC Current (A) = IPFC / (Rshunt * k * A/D)*

where
A/D is the analog-to-digital converter scaling (3412/Volt, 4095→1.2V)
k is the op-amp gain of the PFC current sense circuit
Rshunt is the PFC current feedback resistor in Ohms

I_REF (PFC current reference

*PFC Current Reference (A) = I_REF / (Rshunt * k * A/D)*

where
A/D is the analog-to-digital converter scaling (3412/Volt, 4095→1.2V)
k is the op-amp gain of the PFC current sense circuit
Rshunt is the PFC current feedback resistor in Ohms

PWM_Command (**PFC** PWM modulation): The modulation scaling is different than in the motors; it is defined in terms of the PFCPWMPeriod register:

*Modulation (%) = 100 ** **PWM_Command** */ 65535*

5.3.4 Optimizing Starting and Running

The operation of the PFC switching can be understood by the waveforms of Figure 82. A PWM SyncPulse occurs at regular intervals according to the motor PwmPeriodConfig register and the PWM ratio between motor and PFC. The PFC PWM frequency can be set to 2, 3, 4 or 5 times the motor PWM frequency, defined by MtrCtrlBits bits 6 – 8.

The SyncPulse initiates the execution of the PFC control loop, updating the PFC PWM Duty Cycle Command. The duty cycle gets latched in at the next SyncPulse and then the latched **PWM_Command** is compared to the PFC PWM carrier up counter. The PFC switch is turned on when the **PWM_Command** is greater than the PFC PWM carrier. The inductor current is sampled at the center of the PFC PWM pulse.

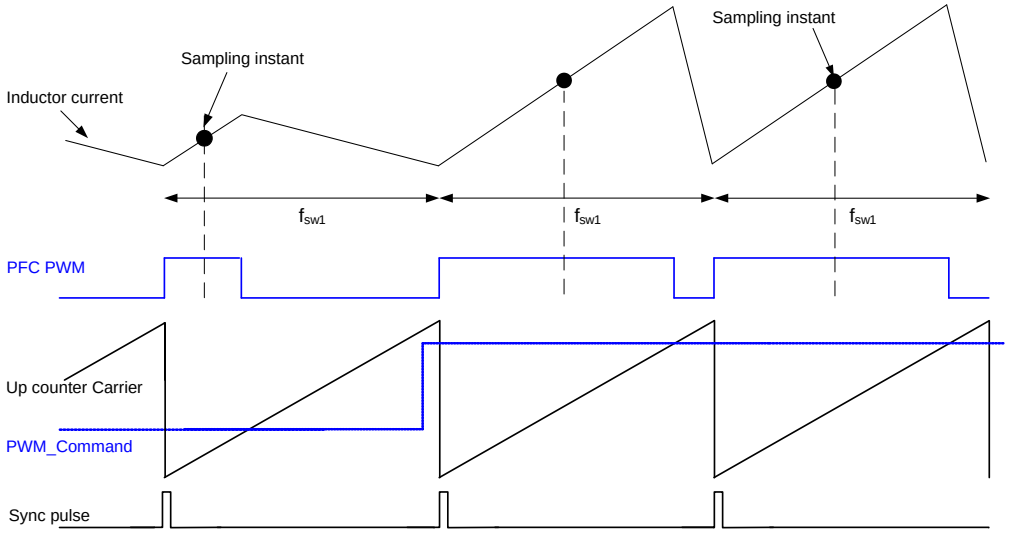


Figure 82—PFC PWM Cycle Timing

5.3.4.1 Varying Switching Frequency

The PFC PWM switching frequency is an important parameter in the design of PFC systems. A higher switching frequency can allow for a lower PFC inductor value or for a reduction of the current ripple at the expense of higher switching losses. A lower PFC frequency (for example 18kHz) allows the use of a low cost iron-core inductor; however, the PFC performance might also be reduced. The trade-offs from increasing the PWM frequency are total system cost and performance, increased heating of the PFC switch due to switching losses, and increased use of the MCE processor resources. If you increase the PWM frequency be sure to check the MCE usage on the 'Options' page of MCEWizard100 and then create new drive parameters based on the new frequency. Many gains will change with a change of PWM frequency. It is recommended that the utilization not exceed 85% to ensure that none of the calculations are missed; MCEWizard100 will give a warning if utilization exceeds 90%.

5.3.4.2 PFC Start-up/Voltage Ramp Rate

MCEWizard100 configures the system registers to set the voltage ramp to ~900V/s. See the equation in Section 5.3.3 to calculate the ramp rate. Note that changing the switching frequency or the voltage feedback gain will change the ramp rate. When increasing the ramp rate, verify that the resulting input current during PFC start-up is not too large.

5.3.4.3 Voltage Loop Tuning

The voltage loop requires a much lower bandwidth than the current loop. The Reference Design Kits automatically set the voltage loop bandwidth to ~10 Hz, but the optimal bandwidth depends on the application requirements. For example, some situations may specify a high voltage ramp rate with little or no overshoot. This may be achieved by increasing the integral gain (KI_V). This will also reduced the voltage rise after a sudden drop in load power. However, increasing the gain too much will reduce the damping (which can make the voltage regulator unstable) and increase the third harmonic. It is up to the designer to verify that the voltage loop performance meets the application requirements.

5.3.4.4 Current Loop Tuning

In general, the bandwidth of the current loop should be as high as possible. However, there are several factors which limit the achievable bandwidth, including the control delay from the current sampling to the update in the modulation command or the amount of noise in the current and voltage feedback signals. MCEWizard100 configures the current loop to have a bandwidth of ~1400 Hz, which is optimized for most applications.

A very simple way to adjust the PFC current loop bandwidth is to keep the same inductor while changing the inductance value input into MCEWizard100. A higher value than the real inductance will increase the real PFC current regulator bandwidth and vice versa. For a more stable system, use the lowest bandwidth which meets the power factor requirement, particularly when the IPFC feedback noise is significant.

5.3.5 Other PFC Features

There are several other features of the PFC which are available to the designer for improving the PFC operation, reliability, and ability to handle short line disturbances. Please refer to the IRMCx100 Reference Manual for information on PFC PWM Blanking and the PFC state machine.

5.3.6 Possible Hardware Modifications

This section describes several simple hardware modifications which the designer can do to adjust or extend the capabilities of the Reference Kit to optimize it for the application. In each case, be sure to generate new configuration parameters using MCEWizard100.

5.3.6.1 PFC Current Feedback Shunt Resistor

The IRMCS1043 Reference Design Kit has a 15mΩ shunt resistor for PFC current sensing. Increasing the shunt resistor value will reduce the current range (for the same op-amp gain), but increase the resolution. Always make sure that the shunt resistor is rated for the current and power which will be dissipated in it.

Changing the PFC shunt resistor can change the range of currents which the hardware can sense or change the GK current trip level. Note that another way to change the GK current trip level is to modify the comparator voltage. In Figure 81, change R105 and R24 to set the voltage at the inverting input to the comparator, or only change R48 to change the GK current trip level.

The Verify & Save page of MCEWizard100 displays the current at which the A/D converter saturates. This is calculated based on the PFC shunt resistor value and the PFC current feedback amplifier gain. It is important to keep the PFC drive current less than this value to keep the currents in control. For example, if the IPFC input offset is set to 100mV (R34 and R46 divider get the input offset), the expected input signal range is 100mV~1100mV, the PFC shunt resister is 15mΩ and op-amp gain is 4.1. The PFC instantaneous current should not exceed 1100mV/4.1/15mΩ = 17.89A or 12.6Arms to ensure PFC current is controlled.

5.3.6.2 PFC Current Feedback Amplifier Gain

Modifying the PFC current feedback gain is another way to change the range of current the controller can sense. Figure 83 shows the PFC current sensing op-amp circuit in IRMCS1043. The user can configure this gain as desired by changing the resistors of the input circuit to IPFC+, IPFC- and IPFCO.

PFC Current Feedback Amplifier Gain = 8.2k / (1.00k + 1.00k) = 4.1

Unlike IRMCx300's PFC current feedback circuit, A 50 – 100mV offset is added at the op-amp circuit, so that the inductor current can be sensed at very low load conditions. The PFC current op-amp has a higher gain and larger range, with input range of 50 – 100mV to 1.2V.

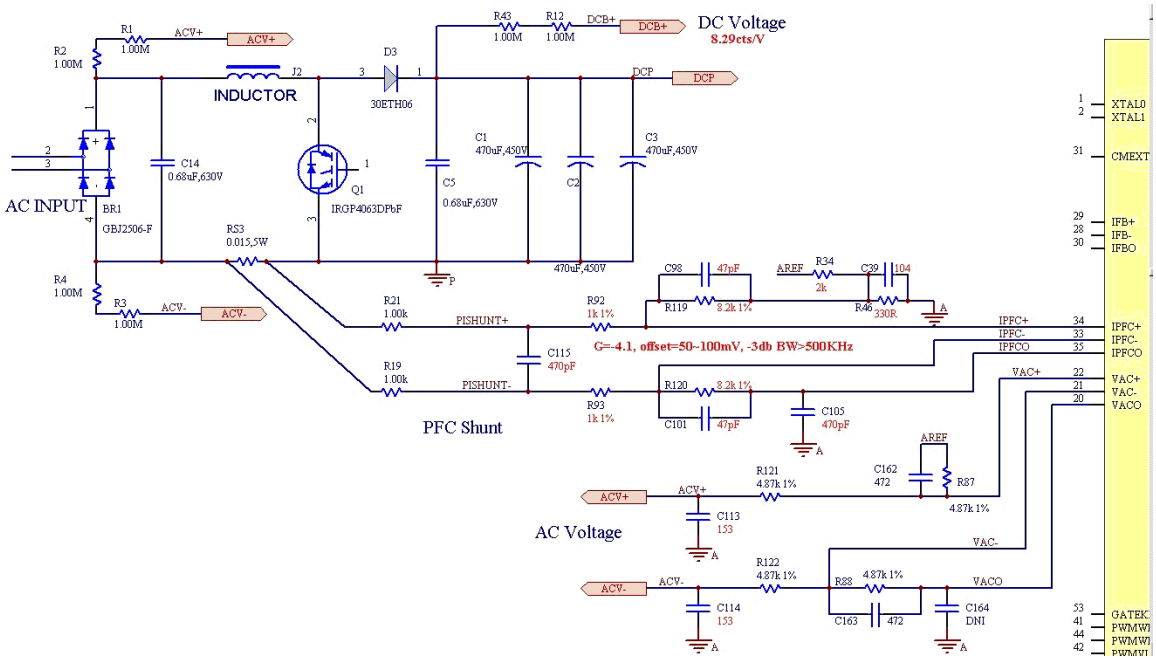


Figure 83. PFC Current Feedback and AC Voltage Feedback Circuit

In the same way as for motor current feedback, the gain of this circuit is 8.2.0 / (1.00 + 1.00) = 4.1. To modify the op-amp gain, the designer should change resistors in pairs (R92 & R93; R119 & R120).

5.3.6.3 AC Line Feedback Scaling

The AC voltage is sensed at the output of the rectifier bridge through an op-amp (VAC+, VAC- and VACO). As an example, for the AC voltage feedback circuit shown in Figure 83, the value for the AC Voltage feedback scaling is calculated as follows:

*AC Line Feedback Scaling (cts/V) = (4.87k / (1.00M + 1.00M + 4.87k*2)) * 4095 / 1.2 = 8.27 cts/V*
(where the A/D gain is 4095cts/1.2V)

Be sure to change the PFC AC voltage feedback circuit together with DC bus voltage feedback circuit if partial PFC mode is used. For partial PFC mode, the AC voltage scaling must be same as the DC BUS voltage scaling because the DC bus reference is set by the AC peak voltage every 100ms and the PFC firmware calculates the reference by assuming the AC voltage feedback scaling is 100% of the DC bus feedback scaling. MCEWizard100 will display a warning on the Verify & Save page if the hardware design does not satisfy the requirements for partial PFC.

5.3.6.4 DC Bus Capacitor

The DC bus capacitor sizing can be determined by the hold-up requirements and ripple tolerance of the application. MCEWizard100 estimates the DC bus ripple and provides a warning if the ripple exceeds 20V. The voltage ripple can be calculated as follows:

$$V_{ripple} = \frac{P}{4\pi f_{AC} \cdot C \cdot V_{DC}}$$

Where:

V_{ripple} is the peak-to-peak voltage ripple
f_{AC} is the AC line frequency
C is the DC bus capacitance
V_{DC} is the DC bus voltage
P is the input power

5.3.6.5 A/D Converter Offset Compensation

In power factor correction, the accuracy of the DC bus voltage feedback is important. The designer can use one of the schemes described in Section 4.1.4 to compensate for any A/D converter and reference voltage offset.

5.3.6.6 PFC Inductor

The inductor may be changed to a different value based on the application requirements. For example, increasing the inductor value will reduce the PFC current ripple, while decreasing the inductor will reduce the system cost. In order to ensure correct PFC operation, it is required that the inductor value exceeds **20 / f_{PFC} mH** (f_{PFC} is the PFC PWM frequency in kHz).

The inductor is easily changed by connecting the new inductor in place of the old one at the connector (J2 in the IRMCS1043 Reference Kit). Be sure to generate new configuration parameters using MCEWizard100. Note that the size of the inductor is not the only factor in selection—for example, in some applications the saturation characteristics may be important, especially when using iron-core inductors.

Remember that inputting a value lower than the real inductance into MCEWizard100 will configure a reduced PFC current loop bandwidth, which stabilizes the system by filtering noise, but may reduce the PFC performance.

For example, in the IRMCS1043 kit, 1.6mH is input into MCEWizard100 even though it includes a 2mH inductor. Changing the Wizard input to 3.2mH (and using the same inductor) will result in a current loop bandwidth much higher than 1.4kHz, which may not be achievable at the PWM frequency—the system may be unstable or PFC gate kill or audible noise occurs. On the other hand, if 0.8mH is input to MCEWizard, the actual bandwidth will be much lower and the PFC performance will be reduced.

5.4 PFC Hardware Design

5.4.1 Schematic Elements

Section 5.3.6 gives some detailed schematic recommendations concerning the PFC current feedback, VAC voltage feedback, DC capacitors, PFC inductor and over current protection circuits. Those guidelines are also applicable to the user application board design. In each case, be sure to generate new configuration parameters using MCEWizard100 if the design is different from the IRMCS1043 Reference Design.

5.4.2 EMI Filter

The Reference Design Kits contain an EMI Filter which can be used as a starting point for the filter required for the application. Figure 84 below shows the schematic of the EMI Filter in the IRMCS1043. CX1 and CX2 act to filter the differential mode noise, while CY1 and CY2 filter the common mode noise. Additionally, L1 acts as a common mode choke to attenuate the common mode noise, while the leakage inductance of L1 also provides filtering of the differential mode noise.

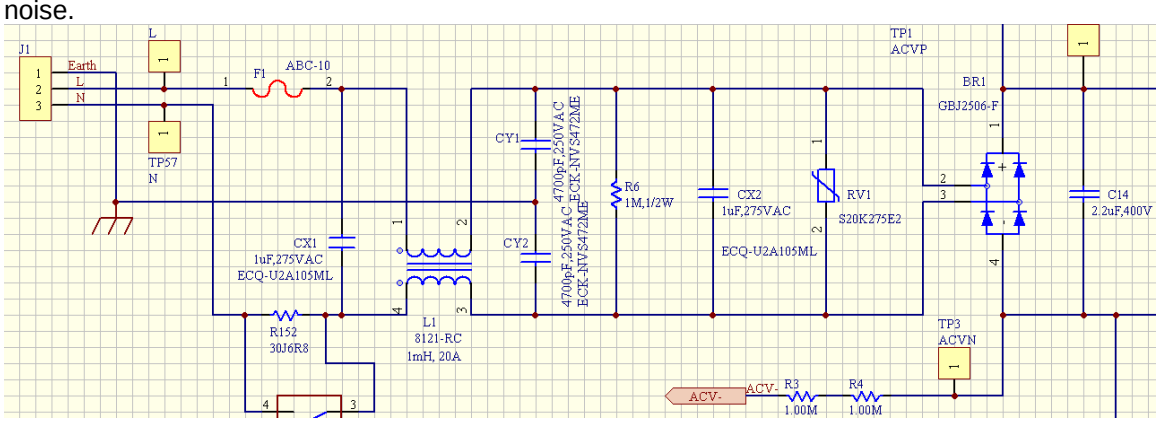


Figure 84—EMI Filter

However, additional Y-Caps should be placed between the PFC shunt resistor and the motor drive inverter. These components will prevent common mode EMI current from flowing through the PFC shunt resistor, reducing current feedback noise and increasing the achievable control bandwidth. For an example, see CY3 and CY4 in the schematic of the IRMCS1043 Reference Design Kit.

5.4.3 Layout Recommendations

Section 4.2 gives some detailed layout recommendations concerning the current feedback and overcurrent protection circuits. Those guidelines are also applicable to the layout for the PFC current feedback and over current protection.

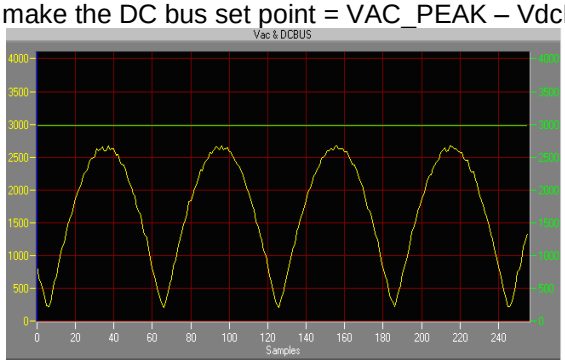
5.5 PFC Starting and Optimizing

IRMCx100 supports both partial PFC and full PFC modes. Figure 85 shows the difference between partial PFC and full mode PFC.

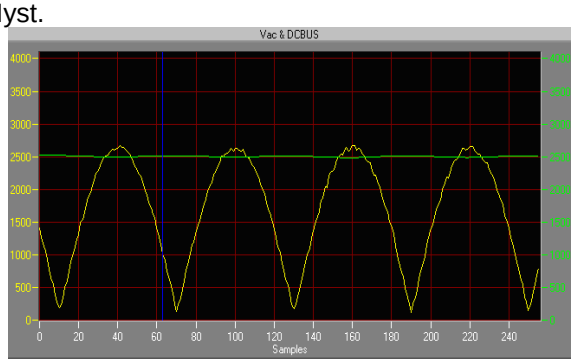
In Full PFC mode the PWM is active 100% of the time, which requires that the DC bus set point is at least 5% higher than the AC peak voltage. An insufficient voltage gap between DC bus and VAC voltage may reduce the PFC performance or even cause system instabilities. To operate

PFC in full mode set the nominal DC bus higher than the peak AC peak voltage by at least 5%. In each case, be sure to have enough voltage margin for the DC bus capacitor. Set register PartialPFC_ON = 0 to select full PFC mode.

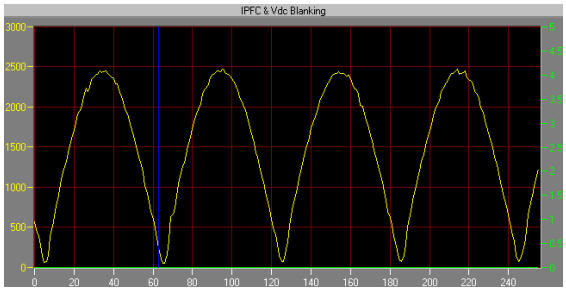
Partial PFC is a tradeoff solution between PFC performance and switching loss where the PWM is only active part of the time (say 60 – 70% of the total grid cycle) mainly near the AC zero crossing. The IRMCx100 automatically configures the DC bus set point 10~30V lower than the AC peak voltage. A larger gap between AC peak voltage and DC bus set point results in lower switching losses at the cost of lower PFC performance. Setting register PartialPFC_ON = 1 will make the DC bus set point = VAC_PEAK – VdcHyst.



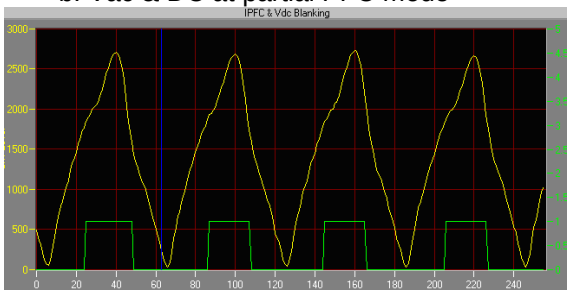
a. Vac & DC at Full PFC mode



b. Vac & DC at partial PFC mode



c. IPFC & Vdc_blanking at Full PFC mode
Full PFC mode: PF=1.000



d. IPFC & Vdc_blanking at partial PFC mode
Partial PFC mode: PF =0.988
(Vdc_Blanking=1 means Vac>Vdc and PWM is OFF)

Figure 85—Partial PFC Vs Full mode PFC

VdcHyst is an important register for partial PFC setup. It sets the gap between the peak AC voltage and DC bus. A gap of 10 – 20V is recommended. This register has the same scaling as Vdc_Ref.

The AC peak voltage (VacPeak) is generated by the IRMCx100 by filtering the AC voltage signal and then extracting the peak voltage. The bandwidth of the low pass filter is ¼ of the motor PWM frequency.

IR WORLD HEADQUARTERS: 233 Kansas St., El Segundo, California 90245, Tel: (310) 252-7105
http://www.irf.com

Edition 2014-02-06

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

IRMCx100_AppDevGuide

IMPORTANT NOTICE

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

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

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

WARNINGS

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

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