# Application Developer's Guide

## iMOTION™ motor control IC with additional MCU

## About this document

### Scope and purpose

The IRMCx300 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 IRMCx300 motor control IC.

Finally, Section 5 provides application guidance for the power factor correction (PFC) features which are available on the IRMCS3012 and IRMCS3043 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 "IRMCx300 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.

International
IOR Rectifier

USER GUIDE

# User Guide #0608

## IRMCx300 Application Developer's Guide

**Version 1.3**

*By International Rectifier's iMotion Team*

Table of Contents

Paragraph annotation of the contents of this User Guide.

# 1 Introduction

There are extensive application development activities which the IRMCx300 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 Guide 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.

The final design step, before 8051 code development, is covered in Section 4. This section guides the user through design, testing and optimization of application specific hardware as it relates to the IRMCx300 motor control IC.

Finally, Section 5 provides application guidance for the Power Factor Correction features which come with the IRMCS3012 and IRMCS3043 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 IRMCx300 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 MCEWizard 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 will cover the starting and control algorithms employed by the IRMCx300 IC. This section will also cover 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 overcurrent 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 2.2.1.2 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 IRMCx300 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 MCEWizard 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 the MCEWizard, 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 the MCEWizard. 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:** They way datasheet motor characteristics are specified for Δ-connected motors are different than for Y-connected motors. The inputs values to MCEWizard are based on a Y-connected motor. However, if the parameters are measured using the procedures which 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 the MCEWizard 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 the MCEWizard: *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 IRMCx300 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.

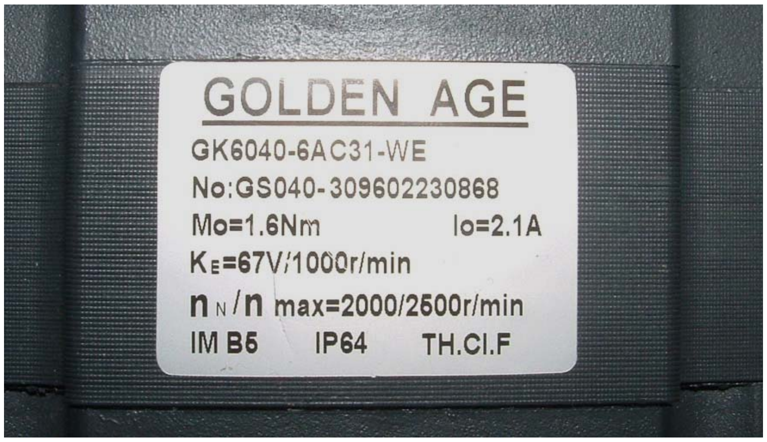


Figure 1—Motor Nameplate.

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 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.*

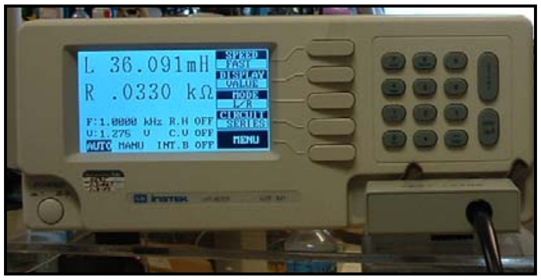


Figure 2—Measuring the Q phase and D phase inductances.

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].

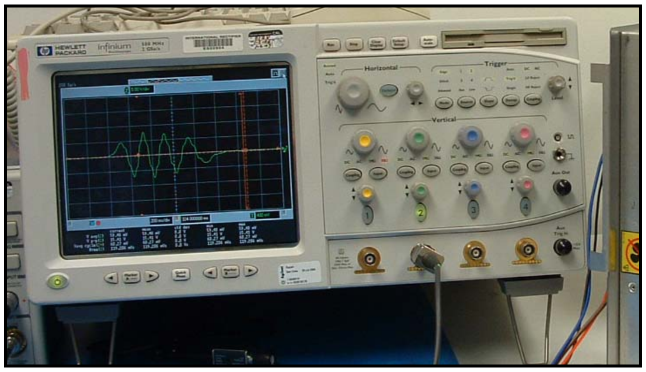


Figure 3—Counting the number of poles. This motor has 8 poles.

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.

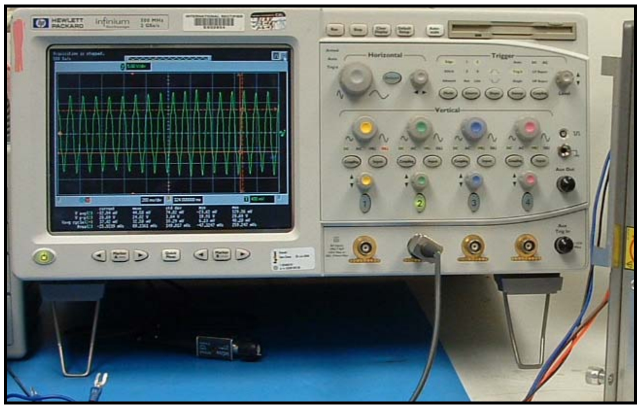


Figure 4—Measuring the back EMF.

**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

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

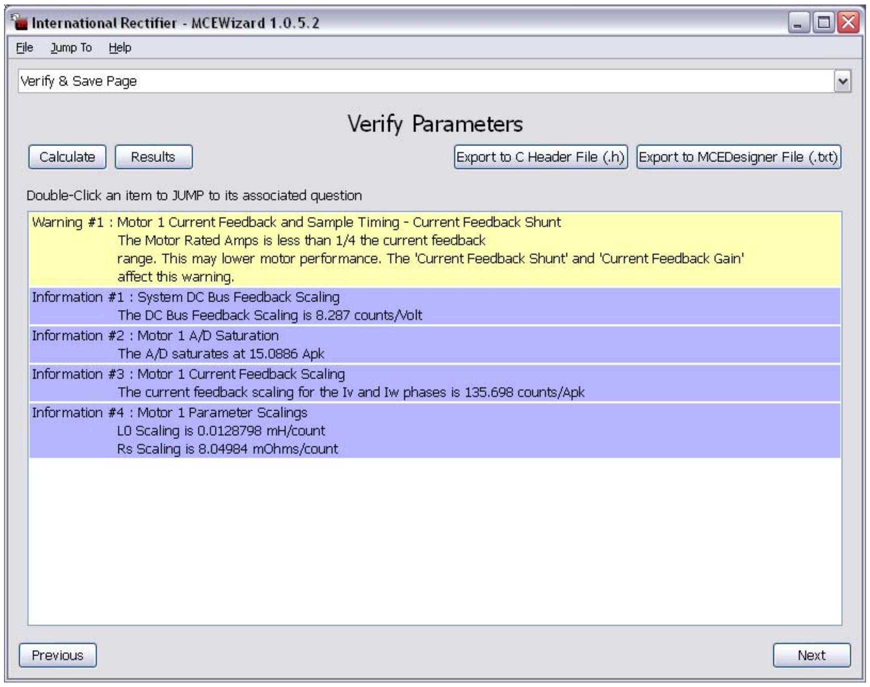
where the units of Ke are Vrms,line-neutral/kRPM and the units of Kt are N-m/Arms. If Lq > Ld (for an IPM), then the torque constant is current dependent. To estimate, increase the value calculated above by 5%.

6. Motor Total Shaft Inertia—This parameter is application dependent. For example, a full washer may have large load inertia while a fan has small load inertia at low speeds. The inertia is used to estimate the motor speed during the open-loop period of the start-up sequence. Therefore, for a fan, the low speed inertia is the appropriate value to use. 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 IRMCx300 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 the MCEWizard 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 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.


International Rectifier - MCEWizard 1.0.5.2
File
Jump To
Help
Verify & Save Page
Verify Parameters
Calculate
Results
Export to C Header File (.h)
Export to MCEDesigner File (.txt)
Double-Click an item to JUMP to its associated question
Previous
Next

Warning #1 : Motor 1 Current Feedback and Sample Timing - Current Feedback Shunt
The Motor Rated Amps is less than 1/4 the current feedback range. This may lower motor performance. The 'Current Feedback Shunt' and 'Current Feedback Gain' affect this warning.

Information #1 : System DC Bus Feedback Scaling
The DC Bus Feedback Scaling is 8.287 counts/Volt

Information #2 : Motor 1 A/D Saturation
The A/D saturates at 15.0886 Apk

Information #3 : Motor 1 Current Feedback Scaling
The current feedback scaling for the Iv and Iw phases is 135.698 counts/Apk

Information #4 : Motor 1 Parameter Scalings
L0 Scaling is 0.0128798 mH/count
Rs Scaling is 8.04984 mOhms/count

Figure 5—MCEWizard 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.

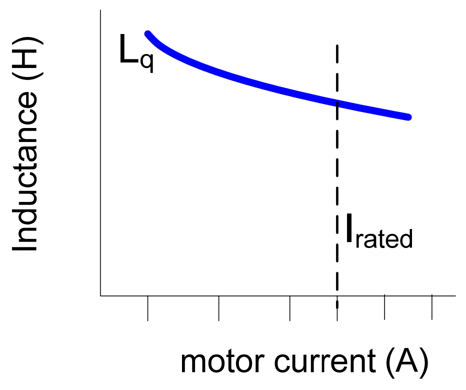


Figure 6—Saturation of Motor Inductance

## 2.2 Starting Application-Specific Testing

Section 2.2.1 will describe the process of testing the target motor in the real target application using MCEDesigner. Section 0 will then describe 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.

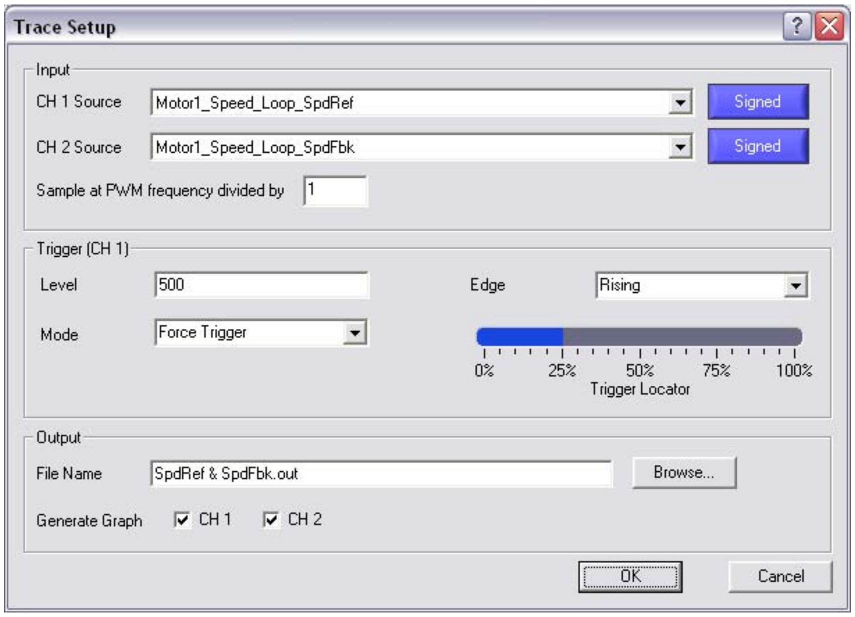
### 2.2.1 MCEDesigner

The main tool 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 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 1024 samples for each channel. 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 StatusFlags register to trigger the Trace to determine at which stage the problem happened. This is particularly useful for start-up problems. (The StatusFlags register is 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.


Trace Setup

Input

CH 1 Source: Motor1_Speed_Loop_SpdRef — Signed

CH 2 Source: Motor1_Speed_Loop_SpdFbk — Signed

Sample at PWM frequency divided by: 1

Trigger (CH 1)

Level: 500

Edge: Rising

Mode: Force Trigger

Output

File Name: SpdRef & SpdFbk.out

Generate Graph: [✓ CH 1] [✓ CH 2]

Figure 7—Trace Setup Window

2.2.1.1 Motor Control Sequencer

To simplify the execution of certain control operations, an embedded sequencer resides on the 8051 side (in the 8051 code known as the MCEDesigner Agent) where predefined functions can be initiated by MCEDesigner through special 8051 registers. The sequencer automatically performs all of the steps required for robust control and startup. An explanation of the available commands is shown below. More information on the sequencer and MCEDesigner Agent can be found in the MCEDesigner User's Guide, including the series of actions performed for each command.

The sequencer has its own group of read and write registers that can be accessed in MCEDesigner. These registers should only be used when sequencer operation is desired. A description of these registers is shown below. Examples of the use of these registers can be seen by looking at the default MCEDesigner functions.

Write Registers

- SeqEnable

  This registers defines if the motor sequencer should be enabled.

  0 – Sequencing disabled
  1 – Sequencing enabled

- SeqFaults

  This register defines what command, with system scope, is to be performed. This register will revert to 0 after a command is completed.

  Bit 0 – No Command
  Bit 1 – TRUE value performs Fault Clear
  Bit 2 – TRUE value performs Emergency Stop

- SeqCatchEn_1 & SeqCatchEn_2

  These registers are control registers for Motor 1 and Motor 2. They define how the sequencer should handle starting the motor.

  Bit 0 – TRUE value enables Catch-Spin Start, FALSE disables

- SeqCmd_1 & Seq_Cmd_2

  These registers control the specific command that will be sent to the sequencer.

Value 1 – performs Motor Start command
Value 2 – performs Motor Stop command
Value 3 – performs Zero Vector Brake command

- FlxThrC_1 and FlxThrC_2
  These register's values are generated by the MCEWizard. This value determines the level of Flux required to detect if the motor is spinning when performing Catch-Spin Start. If current motor Flux is above this threshold the motor is considered to be turning, if it is below then a normal startup is performed. This value does not need modification.

- ZeroVectTm_1 & ZeroVectTm_2
  The values of these registers are generated by the MCEWizard. This value determines how long the motor should brake the motor before changing its direction during Catch-Spin Start. This value does not need modification.

Read Registers

- SeqFaultStatus
  This registers is as copy of the FaultFlags register in MCEDesigner as seen by the sequencer.
- SeqState_1 & SeqState_2
  These registers indicate what state the motors are in.
  Value is 0 – Not Enabled
  Value is 1 – Motor Stopped
  Value is 2 – Motor Running
- MrotStatus
  Indicates if register values have been properly transferred to the sequencer from MCEDesigner.
  Value is 0 = No Errors
  Value is 1 – Motor 1 register transfer failed
  Value is 2 – Motor 2 register transfer failed
- MrotDiag
  Indicates which register is not transferring to the sequencer. This value should be given to an IR FAE if such an error occurs.

Some restrictions are placed on register names and functionality if the Motor Control Sequencer is to be used in MCEDesigner: register names and functionality cannot be changed for the registers listed below. The 8051 Sequencer relies on these names and functions to control the system. If these names are changed in the Matlab/Simulink MCE program the MrotStatus and MrotDiag registers will indicate that register transfer to the sequencer is not functioning and the sequencer will not enable. If the functionality of these registers is modified then undesired operation may result.

Restricted Registers

- FltClr_Tmp
- TargetSpeed
- SearchAng
- MotorSpeed
- MotorSpeedR
- TargetDir

#### 2.2.1.2 Status Flags

The StatusFlags register is an important and useful tool for identifying and debugging motor control problems. It provides the motor status, particularly which stages of start-up the motor has completed. The list below gives the register's bits and the associated drive status.

| | |
|---|---|
| Bit 0 | **TwoPhsEnable—**Two phase modulation is enabled |
| Bit 1 | **FocEnable—**Field-Oriented Control regulators are enabled |
| Bit 2 | **PwmEnable—**PWM gatings are enabled |
| Bit 3 | **ClosedLoop—**Closed-loop mode is enabled |
| Bit 4 | **ParkingDone—**Parking done; Parking stage has been completed. |
| Bit 5 | **ParkingOne—**First stage (25% of the total park time) of Parking has been accomplished |
| Bit 6 | **StartFail—**Startup has failed (latched until drive restart occurs). |
| Bit 7 | **StartOk—**Startup has succeeded (cleared whenever drive stops) |
| Bits 8 – 15 | Unused |

The drive status is cumulative, so that in a normal, successful start-up progression, the StatusFlags register will return the following (decimal) values:

| | |
|---|---|
| *6* | PWM and FOC enabled |
| *38* | 1st Parking Stage Completed (Sequence is currently in 2nd Parking Stage) |
| *54* | Parking Stage Completed (Sequence is currently in Open-loop stage) |
| *62* | Sequence is currently in Closed-loop stage |
| *190* | Closed-loop, with successful start (StartOK) |

Figure 8 shows the steps which the StatusFlags register (green trace) goes through during a successful start-up, with the scale in green to the right. The yellow trace shows the speed feedback

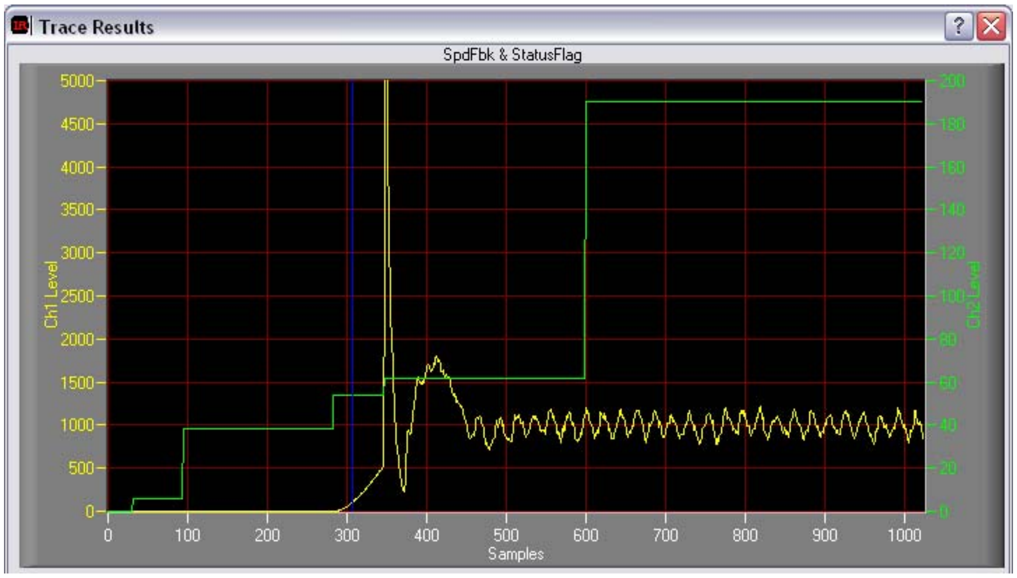


Figure 8—StatusFlags steps (green) and Speed Feedback (yellow) of a successful startup.

### 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 Motor1, 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. All faults may not be valid, depending on the IRMCx300 version in use. The list below gives the faults associated with each bit of FaultFlags:

| | |
|---|---|
| Bit 0 | **OvFault—**DC bus over voltage trip fault. |
| Bit 1 | **LvFault—**DC bus under voltage trip fault. |
| Bit 2 | **PwmSyncErr**—Pwm synchronization error fault. This fault indicates that Motor 1, 2 and PFC are out of synchronization. (See the Reference Manual for more information.) |
| Bit 3 | **PFCGateKill**—PFC Gate Kill fault. |
| Bit 4 | **GateKill_2**—Motor 2 Gate Kill fault. |
| Bit 5 | Unused (reserved) |
| Bit 6 | **PhsLossFlt_2**—Motor 2 phase loss fault. |
| Bit 7 | **ZeroSpdFlt_2**—Motor 2 zero speed fault. |
| Bit 8 | **GateKill_1**—Motor 1 Gate Kill fault. |
| Bit 9 | Unused (reserved) |
| Bit 10 | **PhsLossFlt_1**—Motor 1 phase loss fault. |
| Bit 11 | **ZeroSpdFlt_1**—Motor 1 zero speed fault. |
| Bit 12 | **MCEFlt**—The MCE has generated a fault condition. |
| Bits 13 – 15 | Unused (reserved) |

Fault handling is performed in the MCEDesigner Agent by a timer interrupt every 2 ms. This interrupt will shut down Motor 1 in the case of Motor 1, voltage or MCE fault. Similarly, the interrupt will shut down Motor 2 in the case of Motor 2, voltage, or MCE fault.

### 2.2.1.4 Using GPIO from MCEDesigner

In some cases, the user may want to control external devices, switches, or even gate driver ICs using the general purpose digital I/O (GPIO) during initial testing phases, before writing embedded 8051 code. This capability is available in MCEDesigner from two register groups, one write (GPIO Control) and one read (GPIO Status), contained in the .irc file. The tables below list the registers and their address offsets.

| GPIO Control Write Registers | | |
|---|---|---|
| Name | Offset | Usage |
| IoPortAlloc | 16 | Configuration |
| P1DIR | 17 | Port Direction |
| P2DIR | 18 | |
| P3DIR | 19 | |
| P5DIR | 21 | |
| P1WR | 22 | Port Data Write |
| P2WR | 23 | |
| P3WR | 24 | |
| P5WR | 26 | |

| GPIO Control Write Registers | | |
|---|---|---|
| Name | Offset | Usage |
| P1RD | 27 | Port Data Read |
| P2RD | 28 | |
| P3RD | 29 | |
| P5RD | 31 | |

The procedure for using the GPIO registers is as follows:
**Step 1.**
Write to the IoPortAlloc register to allocate the I/O ports that you intend to use.

In the IoPortAlloc register, a bit is defined to configure MCEDesigner control of each I/O port. Setting a bit to 1 informs the 8051 software that you intend to control that port's I/O pins from MCEDesigner. Setting a bit to 0 frees the port to be used for other purposes locally on the 8051. The bits are defined as follows:

| | |
|---|---|
| Bit 0: | Port 1 |
| Bit 1: | Port 2 |
| Bit 2: | Port 3 |
| Bit 4: | Port 5 |

The other bits are unused and should be set to zero.

**Step 2.**
For each I/O port to be used, write to the appropriate port direction register to configure each bit as an input or an output. Bits 0 – 7 of each port direction register are defined exactly the same as the corresponding 8051 special function register (SFR) of the same name. (Setting a bit to 0 configures it as an input; setting it to 1 configures it as an output.) Bits 8 – 15 are not used.

**Step 3.**
To set the value of bits configured as outputs, write the value to the appropriate port data write register. Bits 0 – 7 of each data write register are defined exactly the same as the corresponding 8051 SFR (P1 – P3, P5). The user must always write the entire data register (i.e. individual bits cannot be written).

**Step 4.**
Read the port data input register to get the current value of the corresponding 8051 SFR (P1 – P3, P5). Reading the data read register shows the value of the pins defined as inputs and also the value last written to pins defined as outputs.

**Note:**
MCEDesigner's interface to the I/O pins is not integrated with other use of the pins directly from the 8051 software. On the IRMCF341 Rev. C reference platform, the 8051 software uses I/O

pins P1.5 and P1.6 to control the bi-color LED. When using the reference platform, do not use the Port 1 I/O pins from MCEDesigner, otherwise the LED will not operate as intended.

## 2.2.2 Possible Hardware Modifications

All of the hardware modifications described below will require new configuration parameters. The designer should return to MCEWizard 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 Inverter Module

The motor drive inverter of the Reference Board is an IR Appliance Module (IRAM), which is a multi-die package of IGBTs, a gate drive IC, diodes and other components. This component may be replaced by a pin-compatible part with a different current rating. For example, if the development kit has a 10A rated IRAM (IRAMS10UP60B), it may be replaced by a 6A or 16A IRAM; both of which have the same pin assignments and footprint as the 10A IRAM. Generally, the module would be changed if the factory default IRAM is not suitable for the application, i.e. being undersized for the current required.

In this case, there are several adjustments which should be made to the motor configuration parameters. First, the shunt resistor, which is included in some modules, may have changed in value. Input the new value into the MCEWizard (Current Feedback Shunt) to get the proper current feedback scaling. Next, follow the instructions in Section 4.3.2 to optimize the current feedback signal. Finally, evaluate the heatsink's capability to dissipate the heat from the new hardware.

### 2.2.2.2 Current Feedback

In some cases, the inverter system maybe 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 MCEWizard 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).

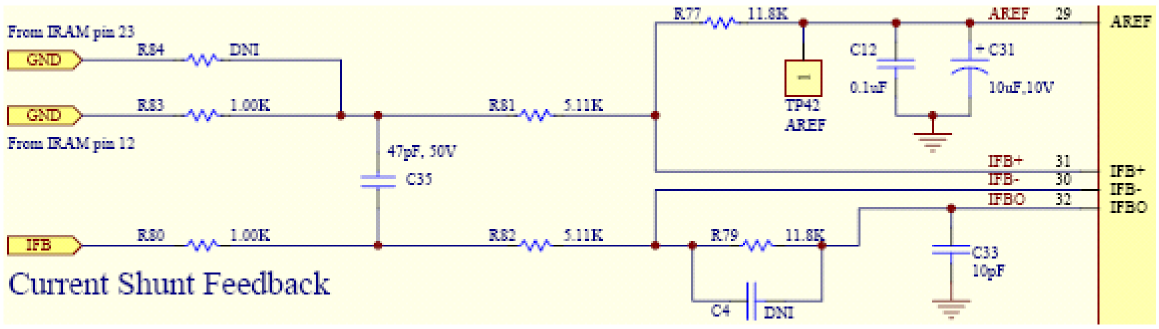


Figure 9—Current feeback circuit

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

In the MCEWizard, 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 variable in this section may be unfamiliar to the designer; they will be explained more fully in further sections of this guide. Many of the parameter scalings are determined by values input into the MCEWizard. Verifying that the variable scaling is correct 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 the MCEWizard

---

**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 1 Frequency entry of the MCEWizard

FreqScl is set by bit fields of **MtrCtrlBits** and **MtrCtrlBits_S** (See IRMCx300 Reference Manual.)

---

**SpdScl** is configured by the MCEWizard as follows:

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

where

Motor Max RPM an entry of the MCEWizard

FreqPwm is the Motor 1 Frequency entry of the MCEWizard

FreqScl is set by bit fields of **MtrCtrlBits** and **MtrCtrlBits_S** (See IRMCx300 Reference Manual.)

In IR's release version of the MCE program, **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 the MCEWizard
Kt is the Motor Torque Constant in N-m/$A_{rms}$ as entered into the MCEWizard

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 the MCEWizard 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 the MCEWizard

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

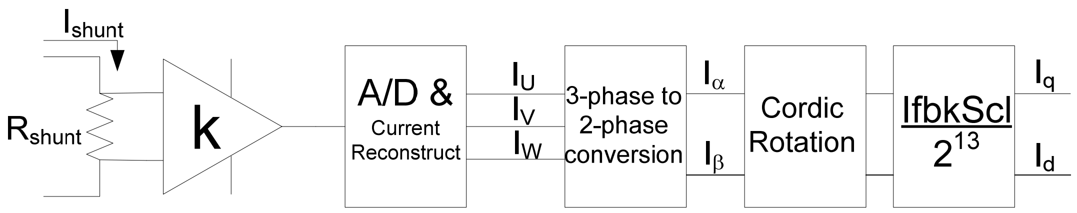


Figure 10—Current feedback signal path

*4095 = Irated * √2 * Rshunt * k * A/D * (3 → 2) * (cordic) * (IfbkScl / $2^{13}$)*
where
Irated is the Motor Rated Current in $A_{rms}$ as entered into the MCEWizard
Rshunt is the Current Feedback Shunt resistor value in Ohms as entered into the MCEWizard
k is the Current Feedback Amplifier Gain entry of the MCEWizard
A/D is the analog-to-digital converter scaling (3412 / Volt)
3 → 2 is the 3 phase to 2 phase conversion gain (8.0)
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 V and W phase currents correspond to **IfbV** and **IfbW**. The scaling for these currents can be found in the Verify & Save page of the MCEWizard 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 the MCEWizard
k is the Current Feedback Amplifier Gain entry of the MCEWizard

---

$I_{\alpha,\beta}$—These currents are a 2-phase representation of the real U, V, and W phase currents. The $\alpha$ and $\beta$ phase currents correspond to registers **I_alpha** and **I_beta**. Their scaling is 8 times 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 the MCEWizard
k is the Current Feedback Amplifier Gain entry of the MCEWizard
A/D is the analog-to-digital converter scaling (3412 / Volt)

---

The **IScl** parameter specifies the current gain scaler for the flux estimator. The MCEWizard 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 the MCEWizard. (**DcBusVoltsFilt** has a 0.492 msec time constant.) This scaling is hardware dependent. In IR's Reference Design Kits, the DCBus voltage signal is reduced through a voltage divider, shown below. Then this voltage is supplied to AIN0 to go to the A/D converter.

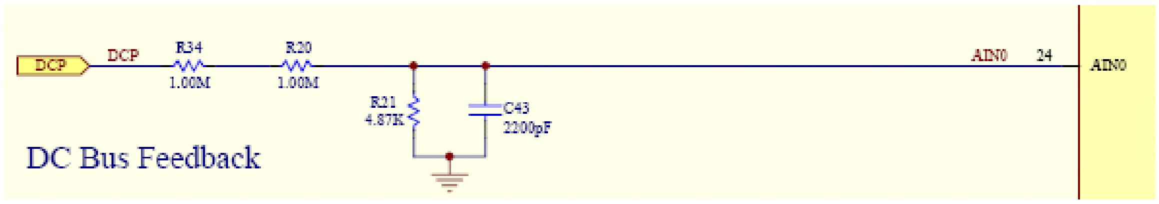


Figure 11—DC Bus feedback circuit

The DCBus voltage can be calculated as follows:

*DCBus (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

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

*Electrical Angle (degrees) = RotatorAngle * 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.

*Parking Current (A) = Irated * √2 * 0.3399 * ParkI / 100*

**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:

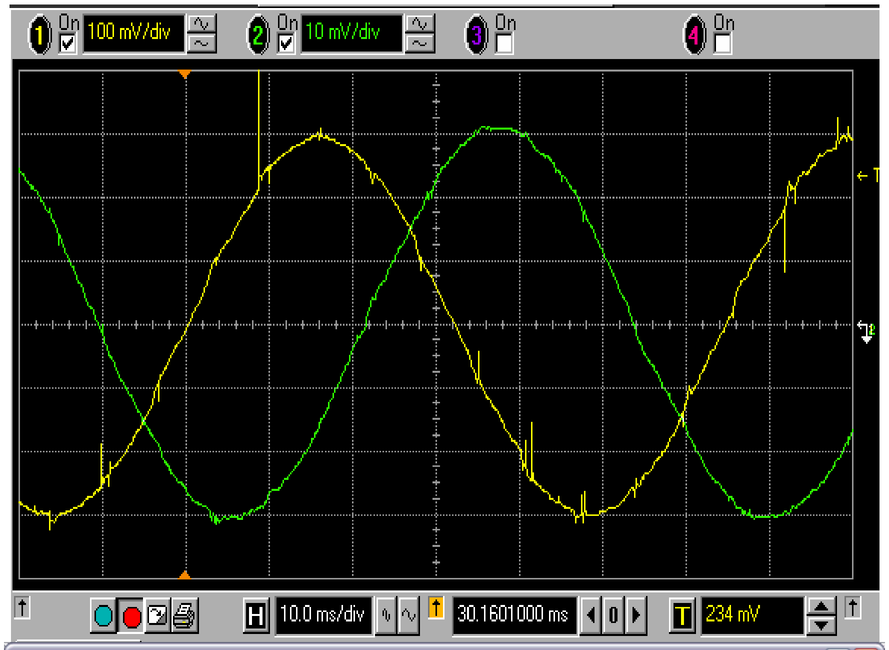
1. VF Diagnostic

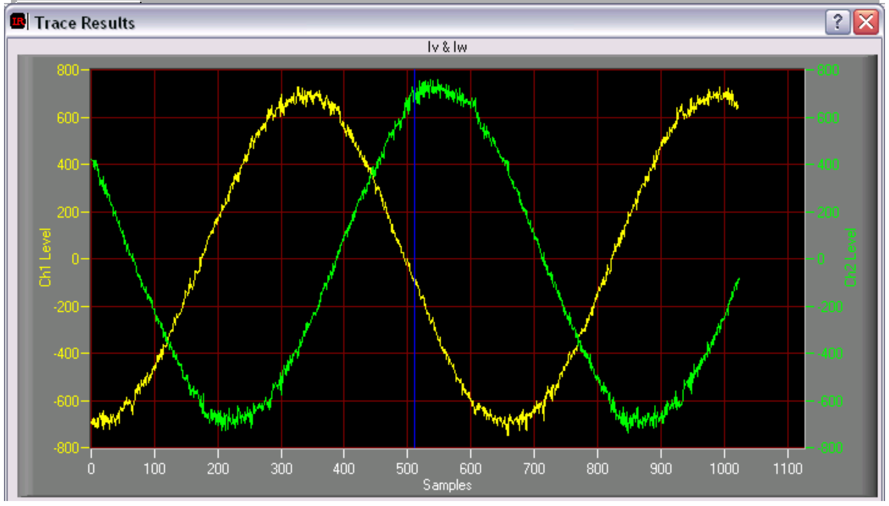
The VF Diagnostic is a useful function for verifying proper operation of the power stage independently of the current feedback and angle estimation. In this function, the motor is operated in a V/Hz mode, where the ratio of voltage applied to electrical frequency is a constant determined by **VFGain**. 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 in MCEWizard 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.





2. Parking Diagnostic

The Parking Diagnostic function of MCEDesigner can also be used to verify the current scaling. The parking diagnostic function simulates the parking stage of the start-up routine of the IRMCx300 Series IC. 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 MCEWizard.

### 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 the MCEWizard. Compare this value with the DC bus voltage measured using a multimeter or other instrument.

## 2.3 Optimizing Starting and Running Parameters

This section will include descriptions of the start-up sequencing and control loops of the IRMCx300 Series ICs. Included with the descriptions will be 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 performs two important pre-startup actions: Offset Correction and Bootstrap Pre-charge. These actions are not automatically sequenced by the MCE processor, but are performed in the MCEDesigner Agent sequencer. For more information, see the Software Developer's Guide.

MCEDesigner performs the current feedback Offset Correction by writing a 1 to the register IfbOffsetCalc, waiting 500ms (at minimum 4095 pwm cycles), and then writing 0 back to the same register. This compensates for offsets in the current feedback path including the A/D offset and reference voltage offset.

The Bootstrap Pre-charge is performed when 12 (0x0C) is written to register pwmctrl. The bootstrap pre-charge turns on the low-side IGBTs in sequence to charge the bootstrap capacitors of the gate drivers. The command to begin the start-up sequence (pwmctrl = 11) follows the pre-charge after a 1ms delay. The precharge action helps to prevent overcurrent trips. More information on this topic can be found in the IRMCx300 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 12 and described below.

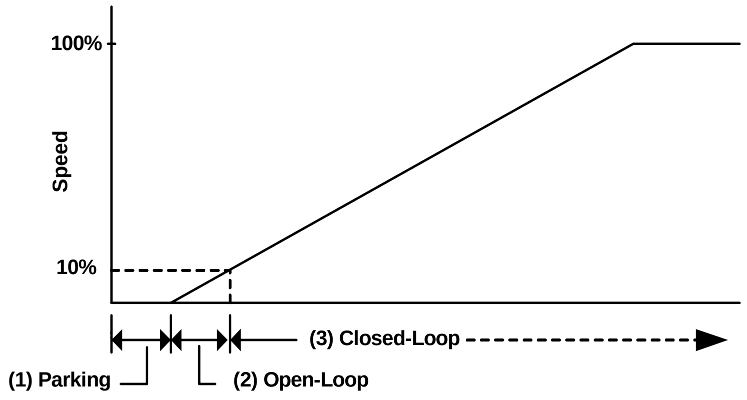


Figure 12—Drive Control Modes

**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 12.

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 Statusflags 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 IRMCx300 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 as a percentage of the motor rated current entered into the MCEWizard.
*Scaling:* 1 = 0.3399% of rated motor current *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. One of the functions pre-installed into MCEDesigner is the "Parking Diagnostic," which can be activated by double-clicking the function. This function will perform parking for 10 seconds and then turn off without starting the motor.

It is possible to manually configure an extended parking duration by forcing the drive into parking mode. This is done by enabling the Parking Diagnostic through the bit fields of the MtrCtrlBits register. The Parking Diagnostic overrides control of parking duration using the ParkTm register. (ParkTm is still used to determine the duration of ParkAng1.)

The following example illustrates this procedure. 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, as in a fan at low speed? 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 (at start-up) 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 fan motor with 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 the MCEWizard. 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 the MCEWizard;
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 IRMCx300 Reference Manual for more information.)

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

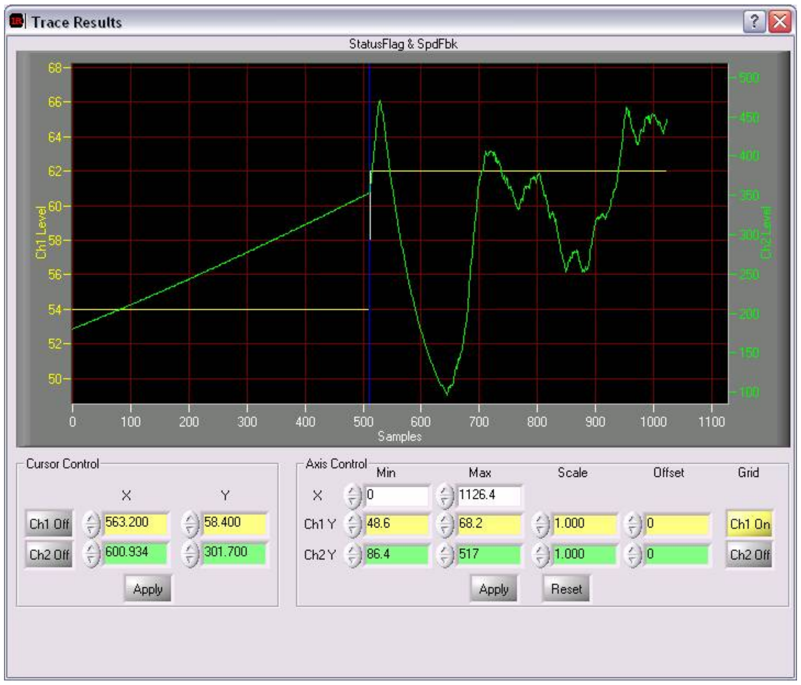
#### 2.3.2.5 Troubleshooting the Closing of the Loop

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 MCEWizard): 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 MCEWizard): 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 Low Voltage Fault may occur. If this is the case, try reducing the Start Limit in MCEWizard, 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. You will see the speed 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 13. In Section 3.5.2 an example of modifying the MCE program to dampen this spike is given.



Cursor Control

| | X | Y |
|---|---|---|
| Ch1 Off | 563.200 | 58.400 |
| Ch2 Off | 600.934 | 301.700 |

Axis Control

| | Min | Max | Scale | Offset | Grid |
|---|---|---|---|---|---|
| X | 0 | 1126.4 | | | |
| Ch1 Y | 48.6 | 68.2 | 1.000 | 0 | Ch1 On |
| Ch2 Y | 86.4 | 517 | 1.000 | 0 | Ch2 Off |

Figure 13—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.

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 RetryTm register). This error mode appears as a bit flag of the StatusFlags register. Start Fail detection can be enabled by setting the NumRetries register non-zero. The successful-startup flux range is set by the registers FlxThrH and FlxThrL. More information about these registers can be found in the IRMCx300 Reference Manual. Note that to attempt multiple start-up retries requires supporting application software in the 8051 processor.

### 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 if desired.

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. This fault can be disabled by setting a bit in register MtrCtrlBits.

### 2.3.2.8 Phase Loss Fault

The control IC can detect a motor phase loss. This fault can be enable or disabled as desired.

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 will detect this condition and turn on the appropriate bit flag of the FaultFlags register. This fault can be disabled by setting a bit in register MtrCtrlBits. The registers AdjPark1 and AdjPark2 have to be configured properly with respect to the parking angles; the configuration is correctly done by the MCEWizard. See the Reference Manual for detailed information.

### 2.3.3 Catch-Spin Starting

"Catch-Spin Start" is a sub-function available in MCEDesigner 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 process (in Figure 14 below) is as follows:

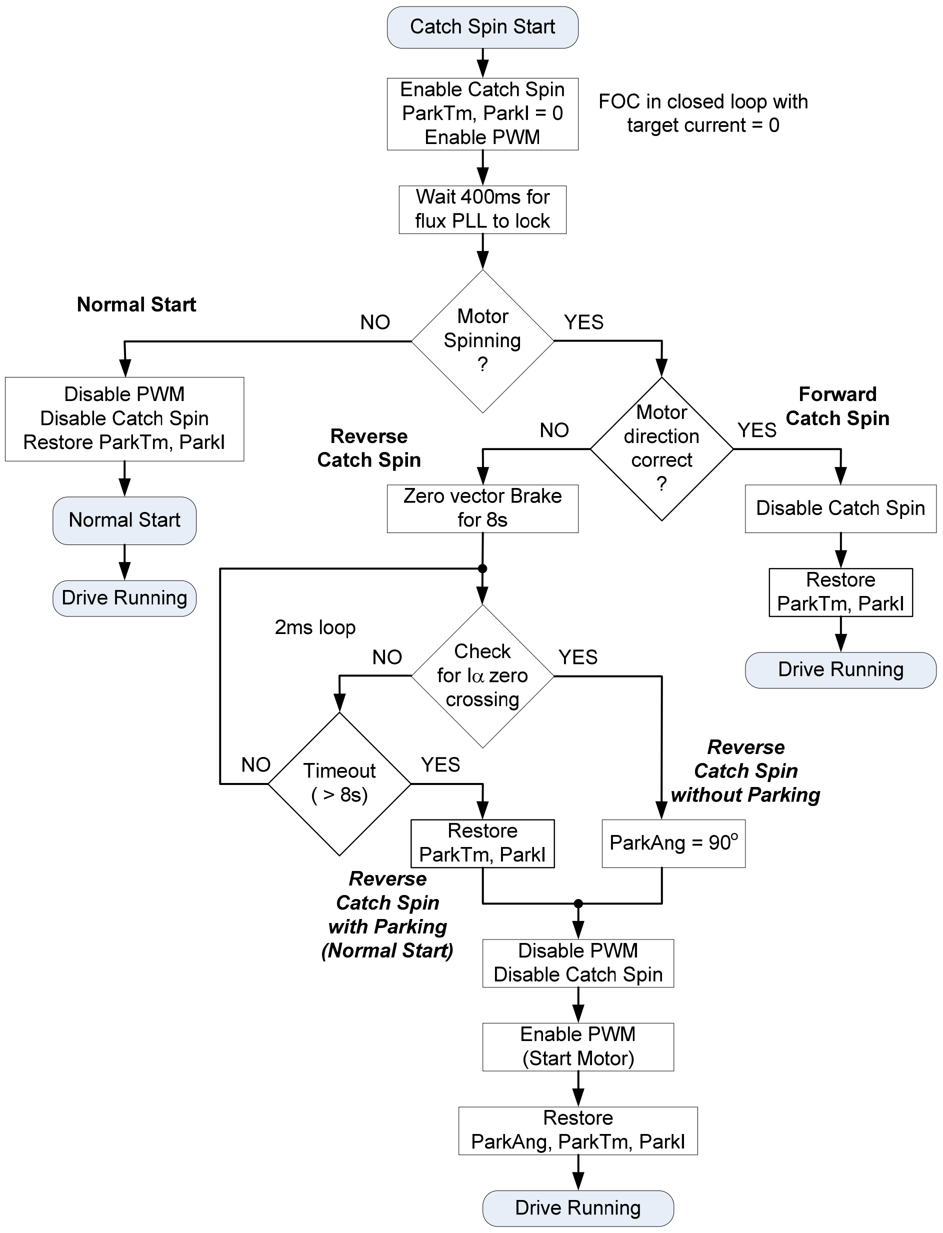

Catch Spin Start
Enable Catch Spin ParkTm, ParkI = 0 Enable PWM
FOC in closed loop with target current = 0
Wait 400ms for flux PLL to lock
Normal Start
NO
Motor Spinning ?
YES
Disable PWM Disable Catch Spin Restore ParkTm, ParkI
Normal Start
Drive Running
Reverse Catch Spin
NO
Motor direction correct ?
YES
Forward Catch Spin
Zero vector Brake for 8s
Disable Catch Spin
Restore ParkTm, ParkI
Drive Running
2ms loop
NO
Check for Iα zero crossing
YES
NO
Timeout ( > 8s)
YES
Reverse Catch Spin without Parking
Restore ParkTm, ParkI
ParkAng = 90°
Reverse Catch Spin with Parking (Normal Start)
Disable PWM Disable Catch Spin
Enable PWM (Start Motor)
Restore ParkAng, ParkTm, ParkI
Drive Running

Figure 14—Catch Spin Start Sequence

The first step of the catch-spin algorithm is to track the back EMF of the motor while forcing the motor phase currents to zero. In addition, the parking time and parking current are set to zero to avoid a phase loss fault (if the fault is enabled). By tracking the back EMF, the speed and direction of the motor can be determined:

1) If the motor is turning is the opposite direction than that desired (Reverse Catch Spin), zero vector braking is initiated (Section 2.3.5.2) to stop the motor. After the motor has slowed enough, the angle is detected by checking for the I$\alpha$ zero crossing. By knowing the motor electrical angle, the controller can goes into open loop without parking, and then closed loop. If the angle detection times out, the motor is assumed to be stopped or turning very slowly and a standard start sequence (with parking) is initiated.
2) If the motor is turning (fast enough) in the same direction as desired, the controller starts in closed loop mode and then accelerates the motor to the desired speed.
3) If the motor is stopped or turning very slowly, then a normal startup sequence will be initiated with parking, open-loop and closed-loop.

Catch-Spin is an advanced function. Tuning and debugging of the catch-spin start is best done from the application (8051) code level of development. The detailed logic and timing for catch-spin can be found in the sample 8051 code, IRSamples.

### 2.3.4 Control Loop Structure & Tuning

There are 3 main control loops associated with IRMCx300 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 15. 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 15, 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 16 where the current controller is rearranged to give transfer function block C(s). Setting $Kp_{Ireg\_D}/KI_{Ireg}$ of C(s) equal to the time constant of the motor ($\tau$), 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 17. The equivalent transfer function of Figure 17 is a first order lag with time constant $\tau_C$. By selecting an appropriate current regulator response (typically 0.5 to 1 msec, MCEWizard 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.

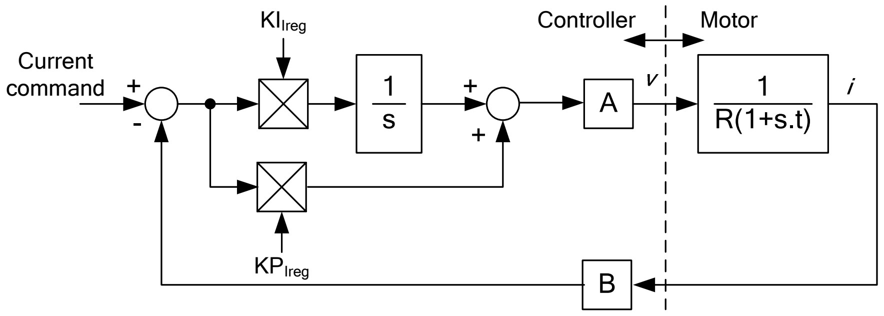


Figure 15—Current Controller Dynamics

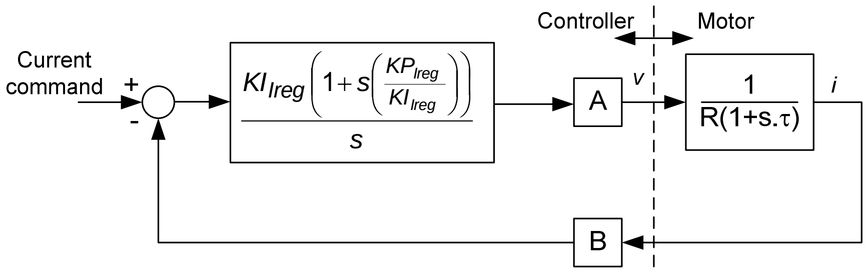

Controller
Motor
Current command
+
-
$\frac{KI_{Ireg}\left(1+s\left(\frac{KP_{Ireg}}{KI_{Ireg}}\right)\right)}{s}$
A
v
$\frac{1}{R(1+s.\tau)}$
i
B

Figure 16—Pole Zero Cancellation

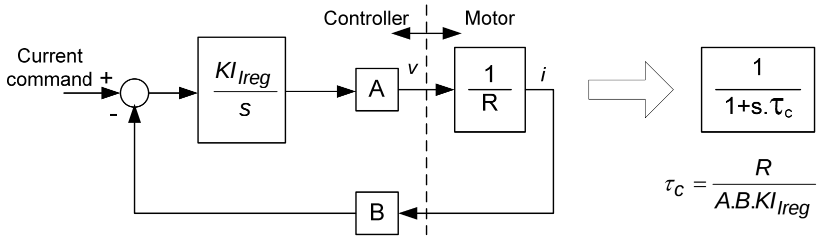


Figure 17—Simplified Current Control Dynamics Due to Pole Zero Cancellation

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

$$Kp_{Ireg} = \frac{L_q \cdot Current\,\mathrm{Re}\,gBW}{A \cdot B}$$

$$KI_{Ireg} = \frac{R \cdot Current\mathrm{Re}\,gBW}{A \cdot B}$$

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:

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

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 section 4.4 of the IRMCx300 Reference Manual. However, the MCEWizard 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:

$$KpIreg = \frac{L_q \cdot Current\,\mathrm{Re}\,gBW .2^{14}}{A \cdot B}$$

$$KxIreg = \frac{R \cdot Current\,\mathrm{Re}\,gBW \cdot T .2^{19}}{A \cdot B}$$

The following gain calculation illustrates a drive application setup with MCEWizard 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:

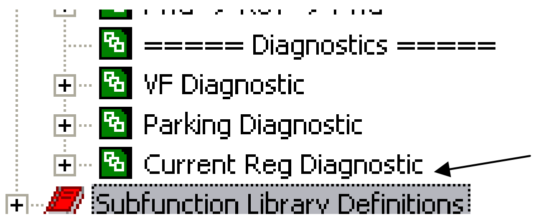
$$KpIreg = \frac{0.021 \cdot 1500 \cdot 2^{14}}{167.1} = 3089$$

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

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

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

MCEDesigner provides a current loop diagnostic test function called “Current Reg Diagnostic.” This test provides the response of the current control loop and also the steady state accuracy.



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 18 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 19 shows the expanded version of Figure 18. The measured current loop response is critically damped with a 0.65 msec time constant (0 – 66.3% of the final steady-state value), which is approximately equal to the anticipated current regulator bandwidth response (1/1500 = 0.667msec).

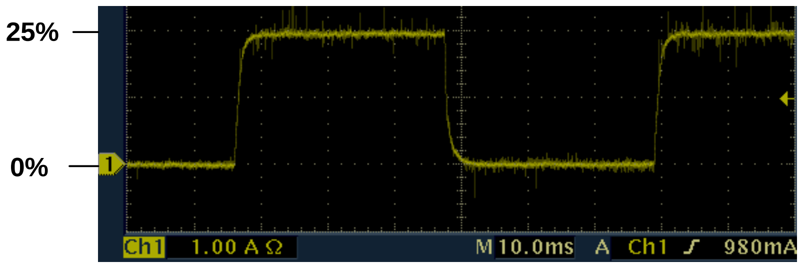


Figure 18—Step Current Response

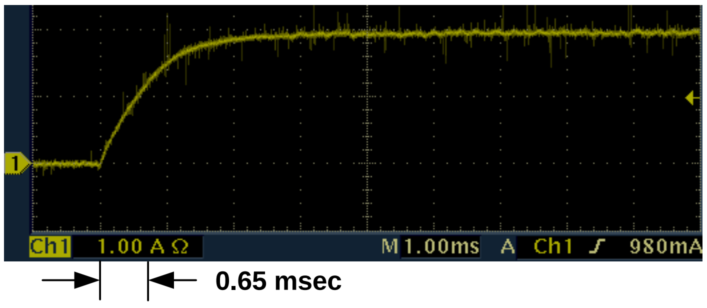


Figure 19—Step Current Response (Expanded Time Scale)

## 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 20 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 21. Parameter M (Figure 21) 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 current regulator tuning section) 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.

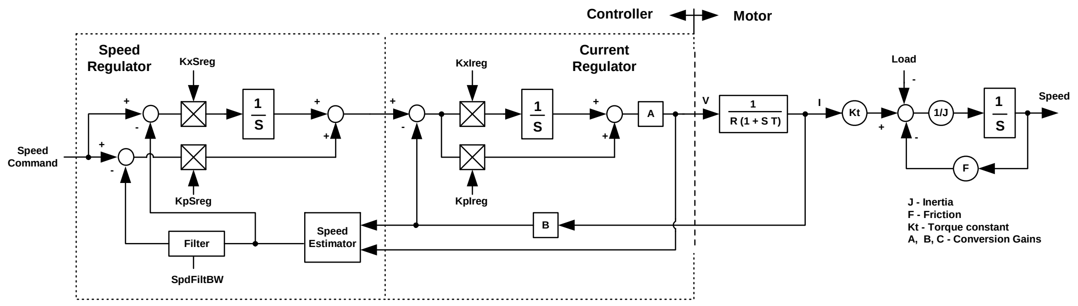


Figure 20—Cascaded Control Dynamical Model

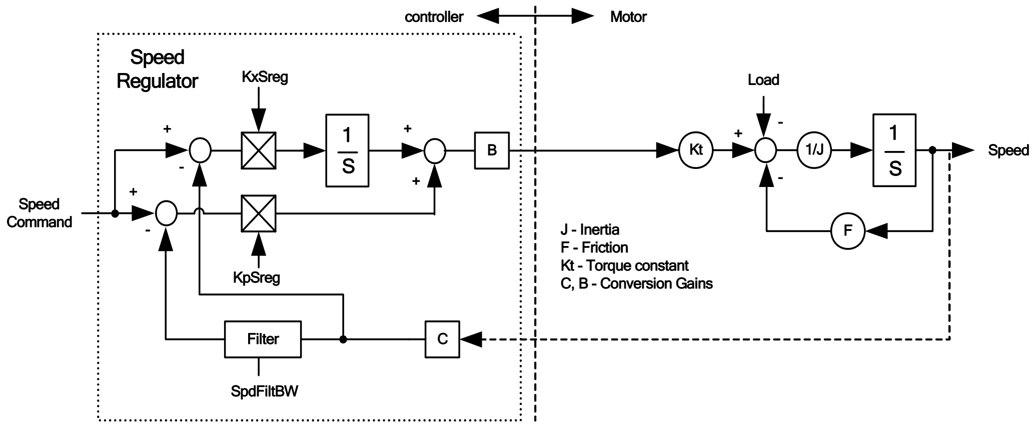


Figure 21—Simplified Speed Control Loop Dynamics

As mentioned earlier, information on mechanical parameters (i.e. Inertia) may be inaccurate, causing MCEWizard to output less than optimal gains for the controller. Manual tuning of the speed regulator can be used to optimize the speed performance. Figure 22 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.

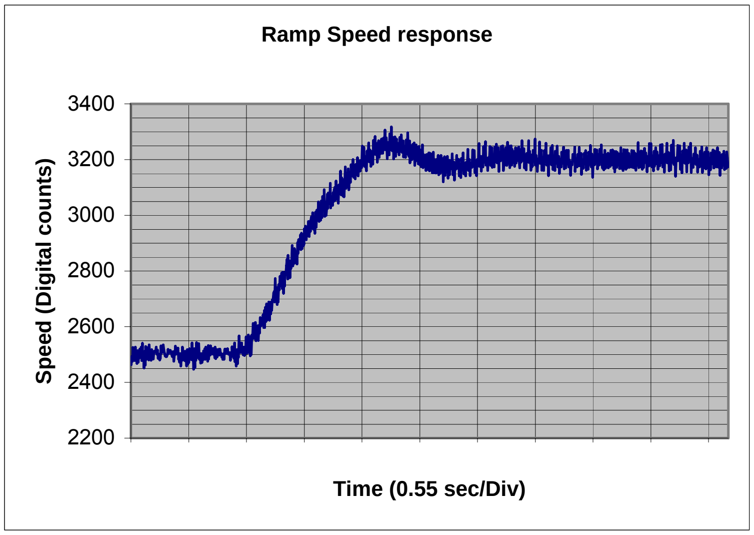


Figure 22—Ramp Speed Response

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 the MCEWizard. Under such conditions, the first order speed response is expected as shown in Figure 23. This figure shows the speed responses using 3 different proportional gains (KpSreg = Kp1, Kp2, Kp3).

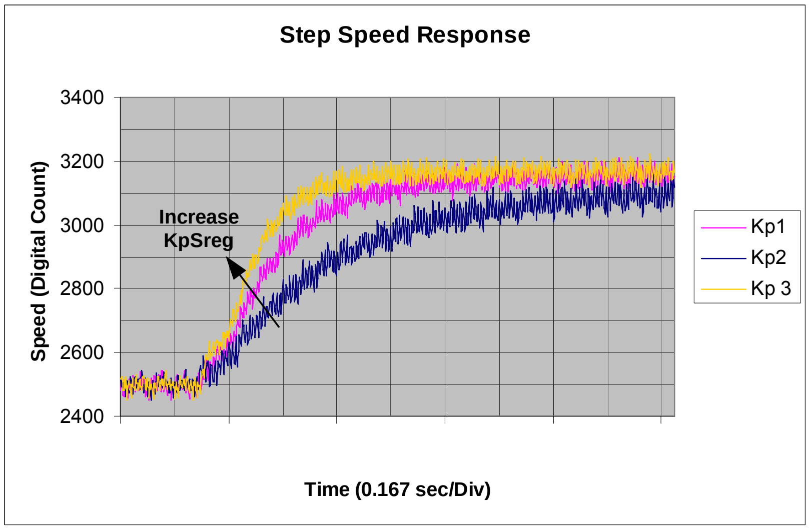


Figure 23—Step Speed Response under Different KpSreg Gains

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), please 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 24 shows the ramp speed responses under 3 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.

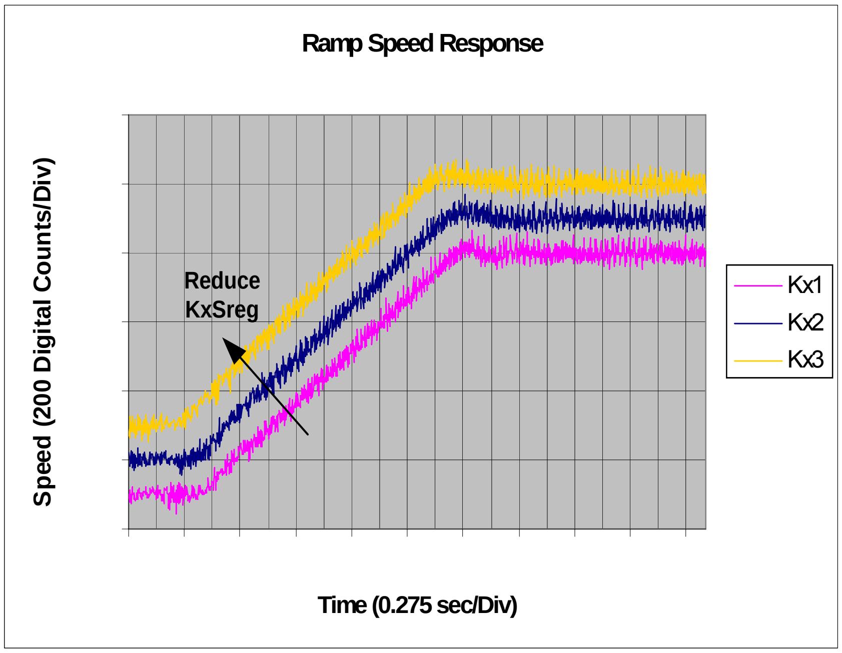


Figure 24—Ramp Speed Response under Different KxSreg Gains

3) Figure 25 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.

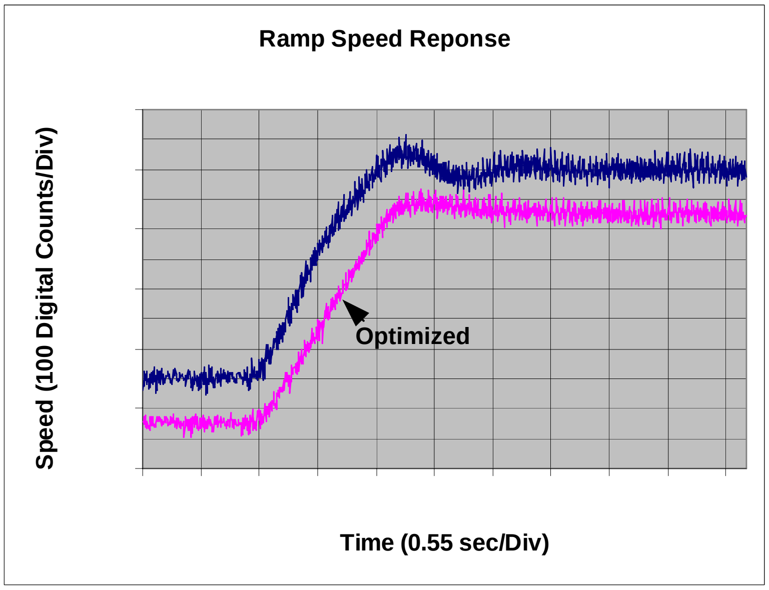


Figure 25—Comparison of Optimized and Non-optimized Speed Response

4) It may be noticed that there is still slight overshoot on the optimized Ramp Speed response (Figure 25). Most applications can tolerate a slight overshoot (<10%).

Increasing KpSreg or reducing the speed ramp rate as shown in Figure 26 can further reduce speed overshoot. It is recommended to keep overshoot to the minimal possible for applications (i.e. Washer Spin Mode), which require a Field-Weakening range of more than 1.5 (150% of the rated speed).

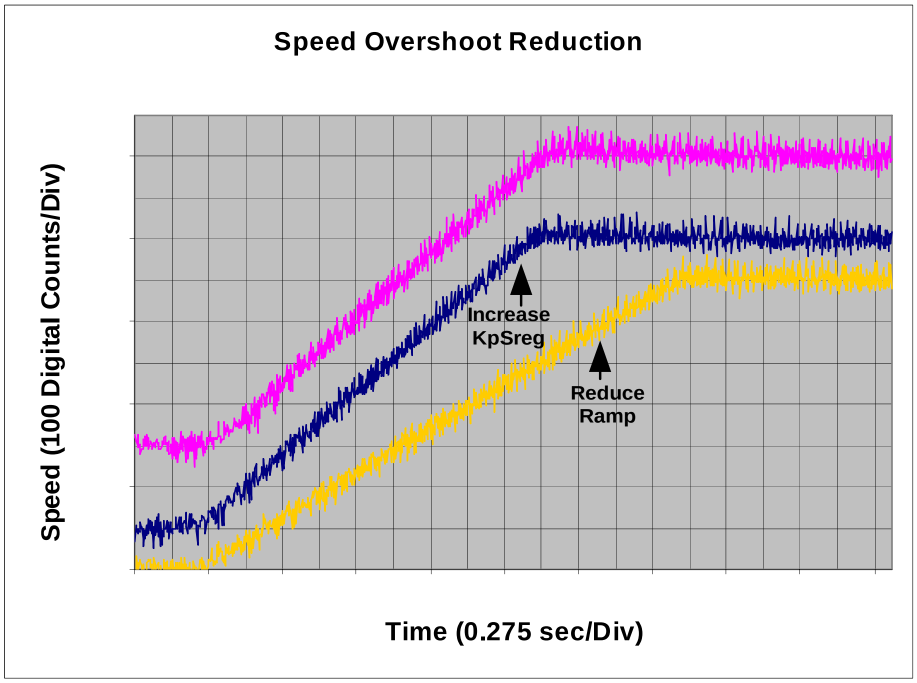


Figure 26—Speed Overshoot Reduction

2.3.4.3 Interior Permanent Magnet Motor Control

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

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

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 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.

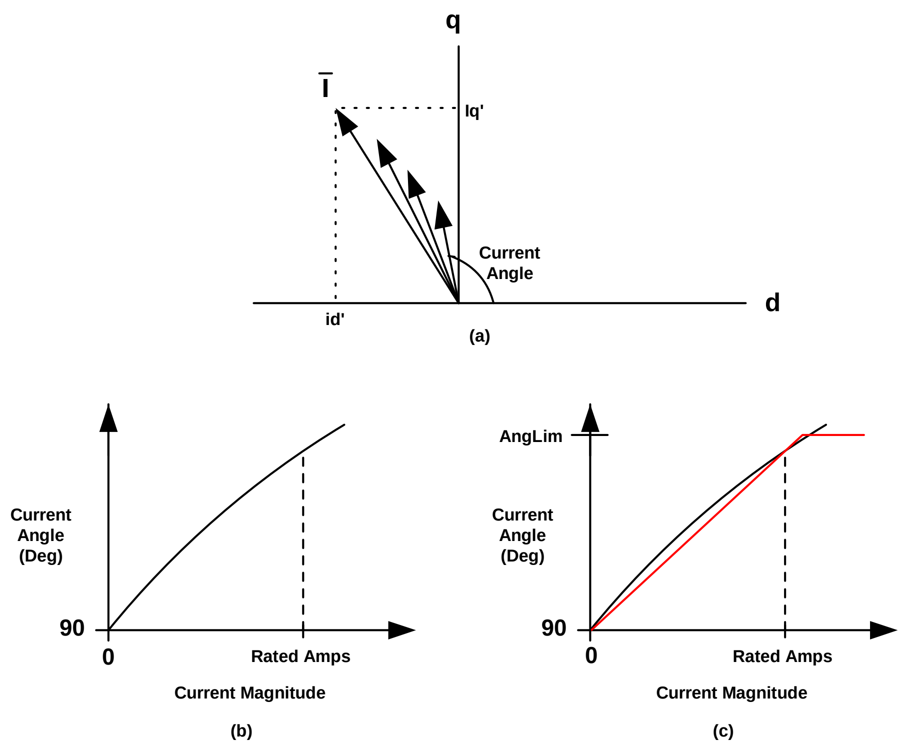


Figure 27—Current Angle at Maximum Torque per Ampere

Figure 27a 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 27b. In the iMotion control IC, this optimal current characteristic is approximated by a linear fit as shown in Figure 27b. 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. The MCEWizard computes AngDel with 2 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 28.

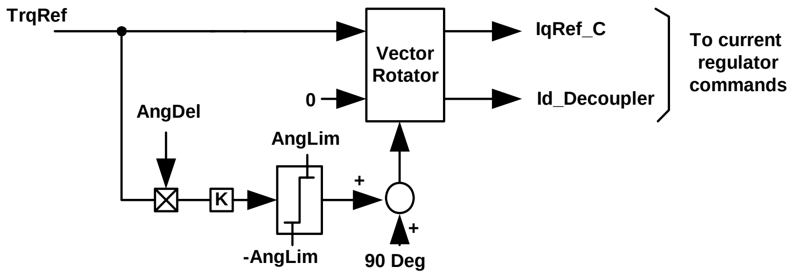


Figure 28—Current Decoupler for Optimal Torque per Ampere Operation

### 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 29.

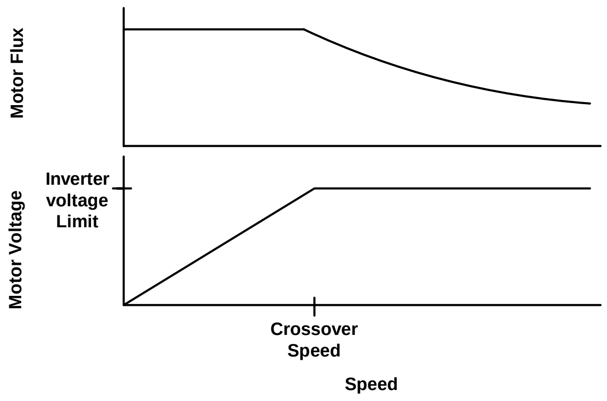


Figure 29—Field-Weakening Characteristics

Figure 30 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 30. Inside the Field-Weakening controller, gain modulation is used to decouple the variation of loop gain due to motor frequency.

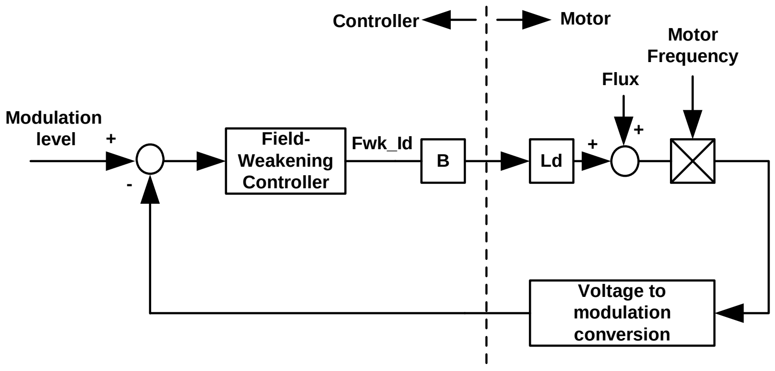


Figure 30—Field-Weakening Control Model

Figure 31 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) will be commanded and reduce 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.

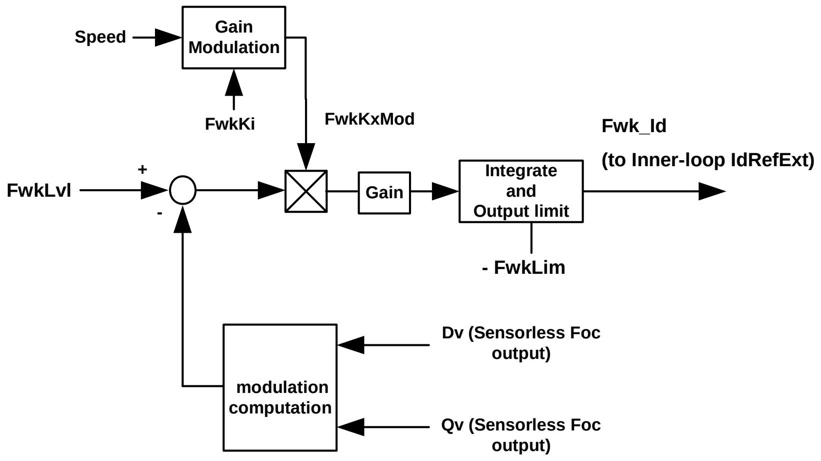


Figure 31—Field-Weakening Controller Block

With gain modulation incorporated, the control loop gain is decoupled from the motor frequency. Figure 32 shows a simplified representation of the Field-weakening controller. The equivalent transfer function of Figure 32 is a first order lag system. Parameter M in Figure 32 is a function of motor inductance, nominal dc bus voltage and the motor crossover frequency (function of motor Ke). The MCEWizard sets the field-weakening controller gain (FwkKi) based on parameter M and a prescribed field-weakening loop response (0.25 sec).

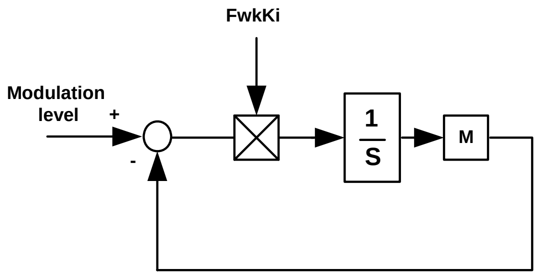


Figure 32—Simplified Field-Weakening Control Model

The response of the Field-weakening loop can be observed from the command d-axis current (using MCEDesigner trace function). Under light Field-weakening condition (-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 33 shows a step change in FwkLvl and the response of the command d-axis current. The first order response is exemplified in Figure 33. 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 MCEWizard is preset to 0.25sec).

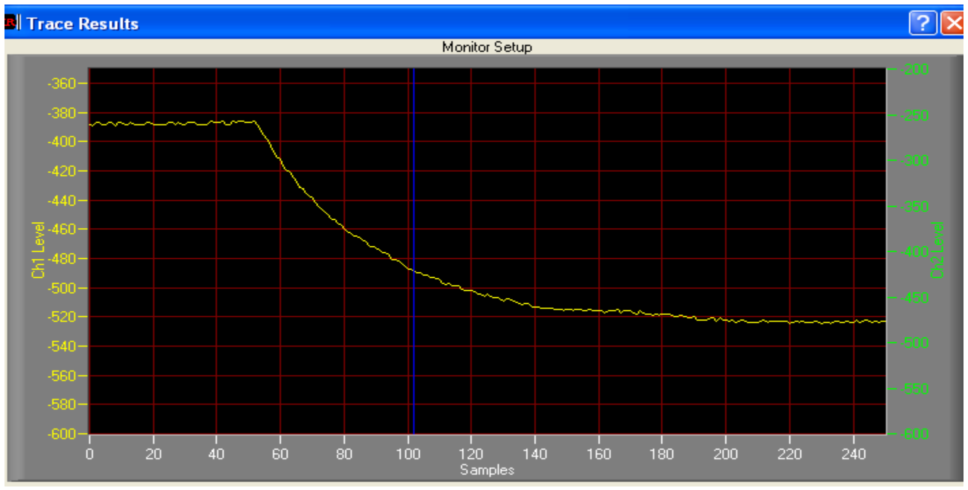


Figure 33—Field-Weakening Control Responses,

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

Some control hints for high speed operation:

- Reduce the speed loop gain to reduce the torque demanded. The maximum torque may not be available during field weakening. Otherwise the controller may go into overmodulation.
- 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 know 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 the MCEWizard to a non-zero number. The Regen Limit is defined as a percentage of the rated current, so a larger Regen Limit will result in a higher 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 become negative, indicating torque in the opposite direction of the motion and resulting in an inversion of the motor current phase. The Torque Reference can become negative 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

Another braking method is Zero Vector Braking. In this case, the three motor terminals are shorted together by alternately turning on all the low-side transistors and then the high-side transistors of the motor drive inverter. 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. To enter Zero Vector Braking, write ‘1’ to register Zero_Vec_Req or use the sequencer by writing ‘3’ to SeqCmd. If the motor speed is high enough (field weakening regime), then the DC Bus may charge up due to the PWM deadtime.

### 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 34. 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.

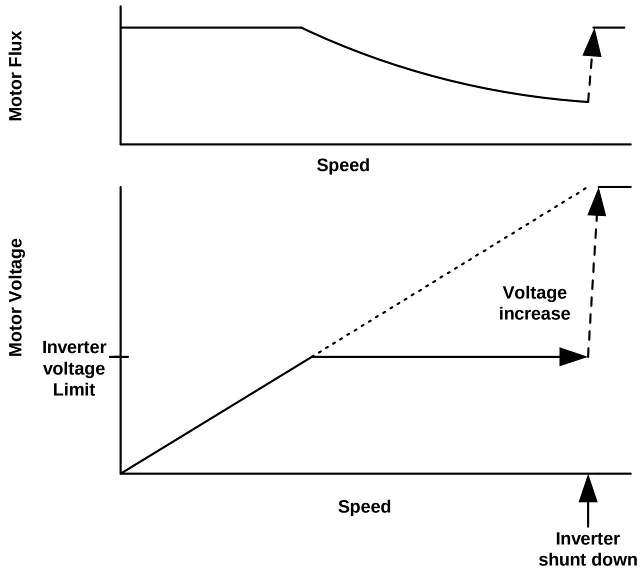


Figure 34—Inverter Shunt Down during Field-Weakening

In the iMotion control IC (IRMCx300 series), the critical over voltage protection is implemented in the MCE application software. The critical over voltage condition is detected by comparing the dc bus voltage feedback to a configurable voltage level (configurable by MCEWizard: DC Bus Critical Voltage Level). When a critical over voltage is detected, the command bit CriticalOv (inputs of MCE module Space Vector PWM) will be set. This will trigger a zero vector (low side devices turn-on) PWM state 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 35 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).

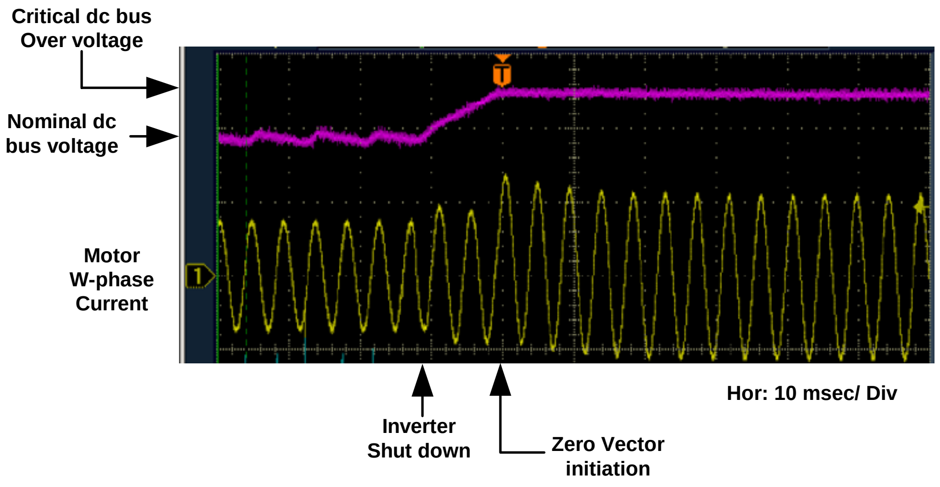


Figure 35—Critical DC Bus Over Voltage
(top: dc bus voltage, bottom: motor current)

The interface between the critical over voltage detection and the Space Vector PWM module is shown in Figure 36. The full description of the SVPWM module is available in the Reference Manual.

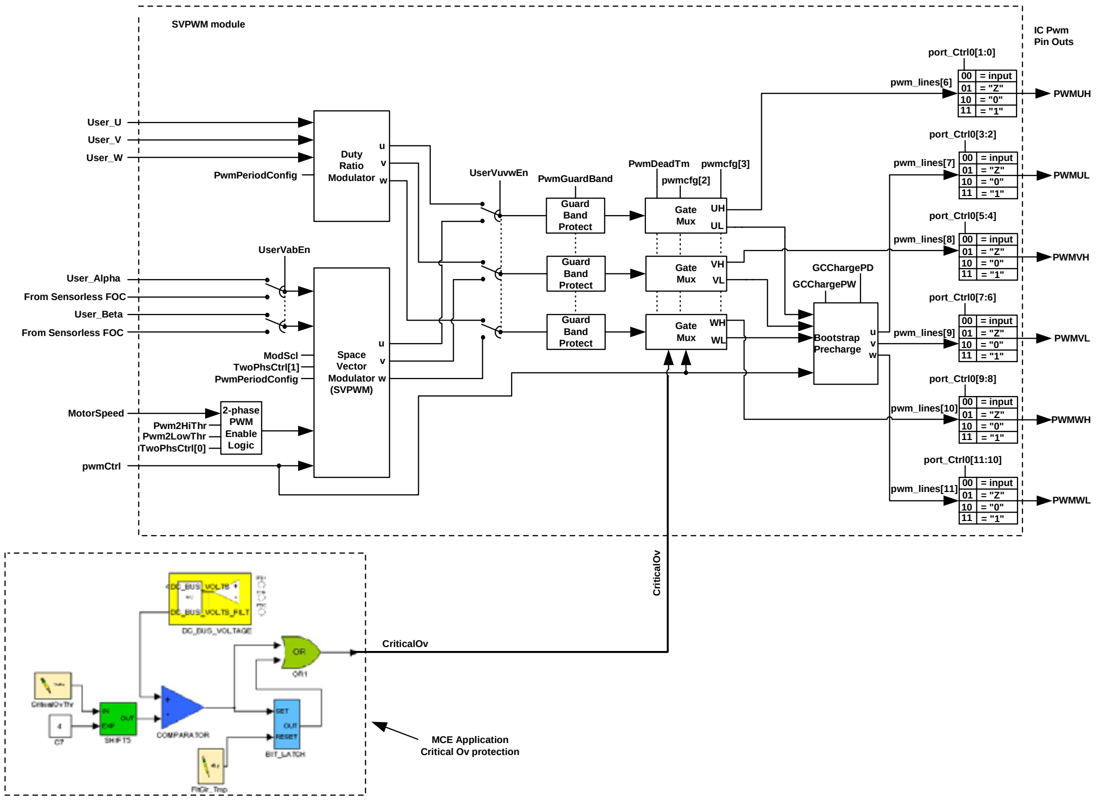


Figure 36—Critical Over Voltage Interface

## 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 contained. The motor control algorithm is realized through a combination of software elements (i.e. Speed loop) and hardware 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 will explain the general properties of a MCE design created in Matlab/Simulink. Section 3.4 will give instructions of how to configure the Matlab software and modify, compile and download the new MCE program. Section 3.5 will conclude 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 IRMCx300 IC.

- MCEDesigner, which provides a graphical user interface to the IRCMx300 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. The main MCE Compiler directory contains three subdirectories: “Simulink Library”, which contains the Simulink library blocks; “Matlab”, which contains the MATLAB scripts that implement the graphical blocks; and “bin”, which contains the executable files and linkable object files for the MCE compiler.

The MCE development tools are designed to operate with MATLAB version 6.1 and later. They may not function correctly with older versions of MATLAB.

**Note:** MCEWizard may produce incorrect register values if the MCE program is changed. The designer should carefully consider whether the MCEWizard 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 IRMCx300 Series IC. Figure 37 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 handles some computation and also sequences the Motion Control Modules according to the MCE program instructions.

The **MCE program** is 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 **Dual Port** (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 Dual Port RAM.

Designed to facilitate and speed the motor control computations, the **Motion Control Modules** are a group of hardware computation blocks to do specific functions. Included in these modules are blocks which produce the gate drive PWM signals for the motor drive or PFC. Also, over-current shut down signals (GATEKILL) go directly to the MCE blocks to get a fast shut down of the drive. Another important module reconstructs the motor phase currents from the single shunt current sampling. More information about single shunt current sampling can be found in Section 4.3.2. 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.

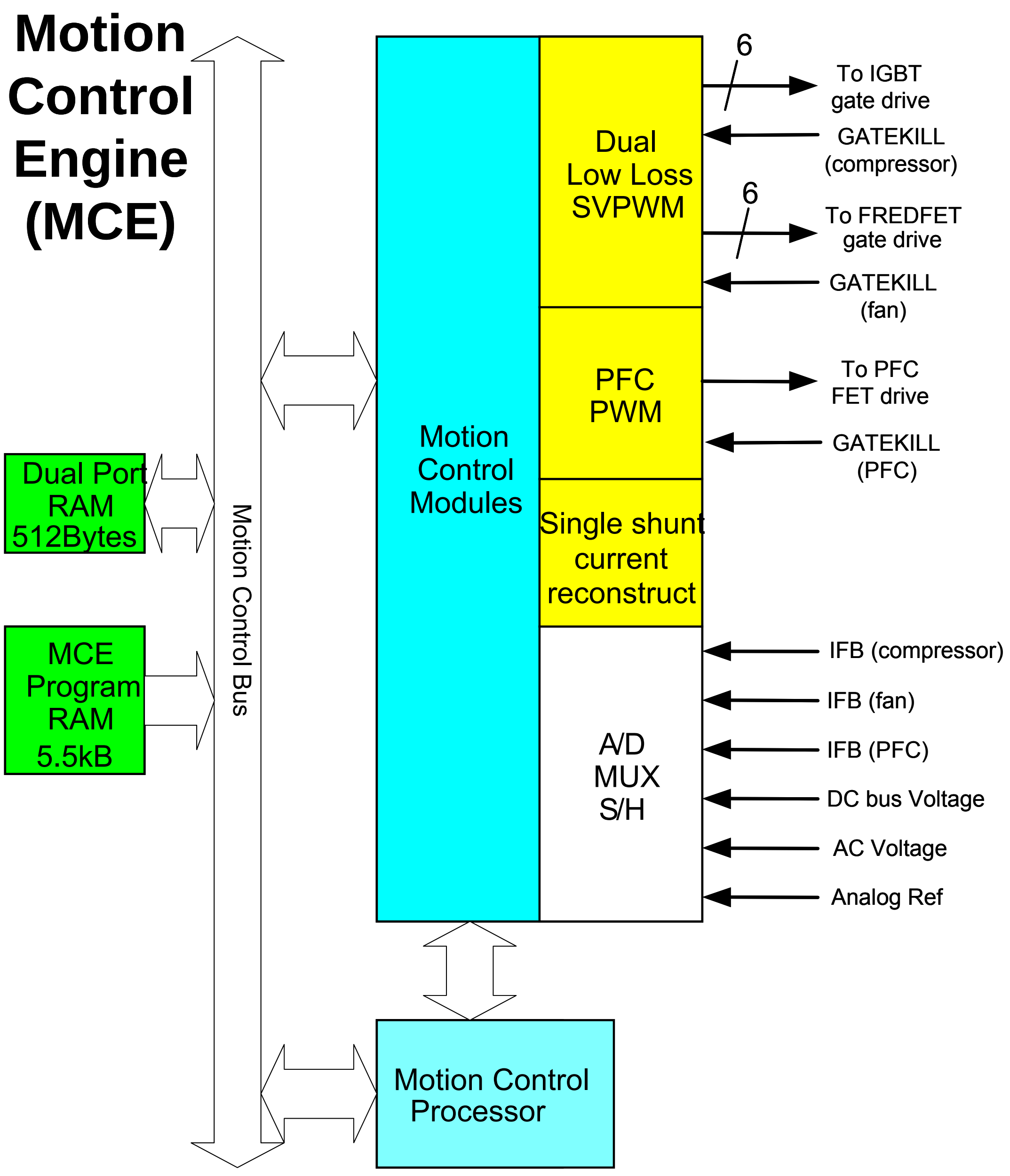


Figure 37—Block Diagram of the MCE

## 3.2 IR Standard MCE Program

### 3.2.1 Block Diagram

Figure 39 is a block diagram of the MCE design of the motor 1 control loop. The Simulink model file (.mdl) and a PDF file of the block diagram can be found within the program installation. 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.
- 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 few of the standard Simulink library components.

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

#### 3.2.1.1 Speed Loop

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 38. After minimum speed protection and sign conversion (based on direction of rotation) logic, 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 yellow 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.

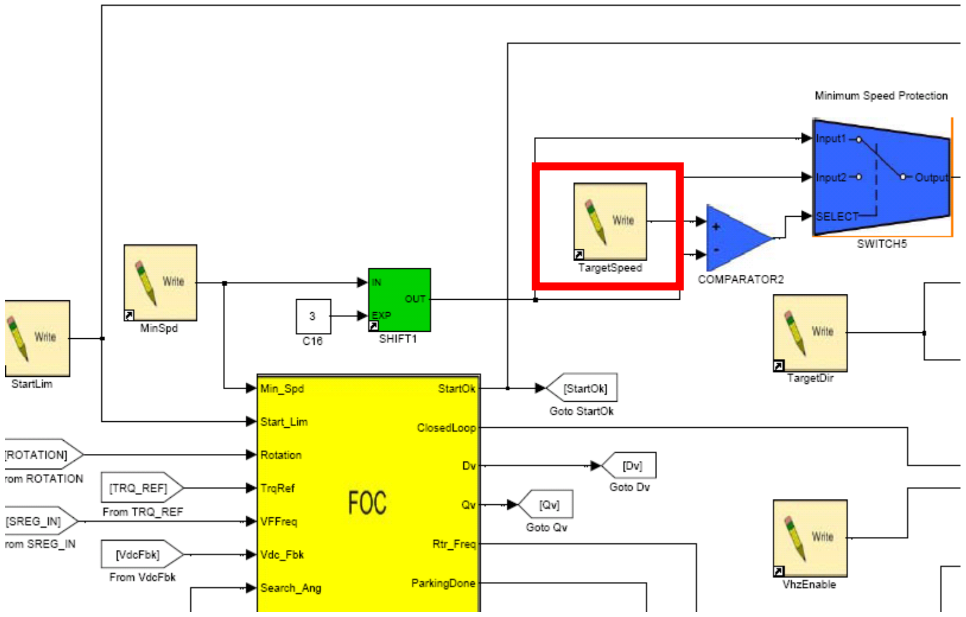


Figure 38—TargetSpeed Register

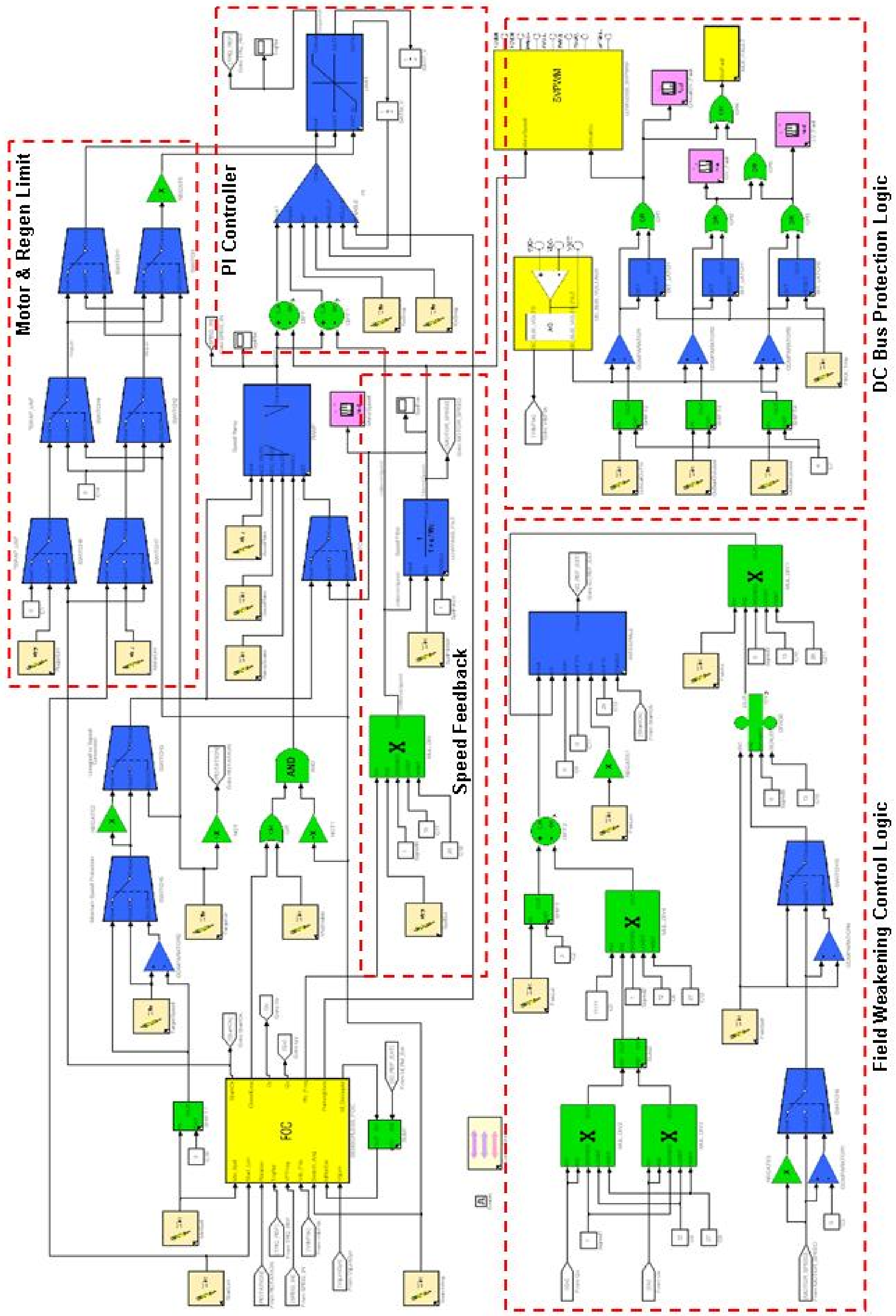


Figure 39—MCE Simulink Design of the Motor 1 Control Loop

### 3.2.1.2 Misc Loop: Dynamic Vq Limit

The MCE program contains a Misc Loop, shown in Figure 40 below, which performs dynamic limiting of the q-axis voltage (Vq). In this subsystem, the VqLimit is calculated by taking the maximum modulation, ModLim, and removing the d-axis voltage (as a vector calculation) to find the modulation available for the q-axis. The result is filtered and then written to VqLim register in the FOC block.

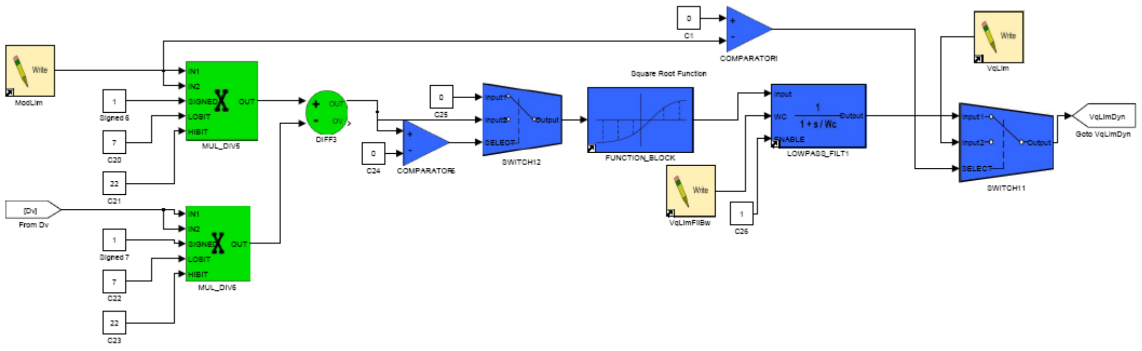


Figure 40—MCE Simulink Design of the Motor 1 Mics Loop

### 3.2.1.3 Other Features (of the MCE program)

Here are descriptions of the other control features of the MCE program, as shown in Figure 39.

- 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 hardware 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.
- The Field Weakening Control logic implements field weakening control as described in Section 2.3.4.4.
- The DC Bus Protection Logic carries out under-voltage, over-voltage and critical over-voltage fault detection. It compares DC bus voltage feedback with trip levels and generates MCE Fault and latch signals for each trip. An over-voltage or under-voltage fault will appear in FaultFlags as an MCE Fault with the OVFault or LVFault register reading a 1. There is a further, hardware layer of DC bus protection at 400V and 120V for over- and under-voltage faults, respectively, for the current feedback scaling of the Reference Board. In digital counts, DC_BUS_VOLTS has hardware over- and under-voltage fault levels of 3360 and 976, respectively. These hardware faults will show up in the bit fields of FaultFlags (See Section 0), but are disabled in the Reference Design, so that only the software voltage fault detection is active.

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.

## 3.2.2 Input and Output Registers of the MCE Program

*Torque Limit Registers:* **StartLim, RegenLim, MotorLim**
Range: 0 – 4095  Scaling: Current (A) = [StartLim] * $I_{rated}$/4095
Function: 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 the MCEWizard.

*Speed Control Registers:*
**TargetSpeed**—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, described below.
Range: 0 – 16383 Scaling: Speed (RPM) = TargetSpeed * (Maximum motor speed) / 16383
**TargetDir**—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**—This register sets the minimum motor speed. The maximum value for this register represents 12.5% of the maximum speed. This register is configured by MCEWizard.
Range: 0 – 255 Scaling: Speed (RPM) = MinSpeed * (Maximum motor speed) / 2048

*Speed Regulator Registers:* **KpSreg, KxSreg**
Range: 0 – 32767
Function: These registers set the speed regulator PI block proportional (KpSreg) and integral (KxSreg) gains. The MCEWizard calculates values for these registers. For more detail on the speed regulator tuning, see Section 2.3.4.2.

*Speed Feedback Registers:*
**SpdScl**—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 the MCEWizard, see Section 2.2.3.1.
**SpdFiltBW**—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.
Range: 0 – 8192 Scaling: Cutoff Freq (Hz) = PWMFreq * SpdFiltBW / 8192

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

*Ramp Rate Registers:* **RampScaler, AccelRate, DecelRate**
Range: 0 – 31 (RampScaler); 0 – 32767 (AccelRate, DecelRate)
Scaling: Motor Acceleration Rate (RPM/s) = (Maximum Motor Speed / 16383) * ([AccelRate] / 2^RampScaler) * PWM Frequency
Function: 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 MCEWizard.

*DC Bus Protection Registers:*
**CriticalOvThr, DcBusOvLevel, DcBusLvLevel**—These registers set the DC bus protection levels for Critical Overvoltage, Overvoltage and Low Voltage faults, respectively. MCEWizard calculated values for these registers.
Range: 0 – 255 Scaling:Trip Level (V) = [CriticalOvThr] * 16 / (DC bus feedback scaling)
**FltClr_Tmp**—Set this register to 1 for at least one PWM cycle to ensure that any DC bus protection faults are cleared. Set FltClr_Tmp to 0 for normal operation.

*Miscellaneous Control Registers:*
**VhzEnable**—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).
**SearchAng**—This logic register (0 or 1) is only used for the catch-spin function. See Section 2.3.3 for more on catch-spin.

*Dynamic Vq Limiting Registers:*

**ModLim**—This register defines the maximum modulation. The Wizard sets this to 1430 = 100% modulation. Set this register to zero to turn off the dynamic Vq limiting.
**VqLimFilBw**—This register sets the filter time constant for the dynamic Vq limiting. The Wizard sets it to 4ms.
**VqLim**—If ModLim = 0, then the Vq limit will be fixed at the value of this register. This register is different than the VqLim hardware register defined in the Reference Manual.

*Read Registers:*
**MotorSpeedR**—This register gives the instantaneous measured motor speed.
Range: 0 – 16383 Scaling: Speed (RPM) = TargetSpeed * (Maximum motor speed) / 16383
**CriticalOV_Fault, OV_Fault, LV_Fault**—These flag registers indicate the presence (1) or absence (0) of a critical overvoltage, overvoltage and low voltage fault, respectively.

*Traceable Parameters:*
**SpdFbk, SpdRef**—These are the measured and commanded speeds, respectively
Range: 0 – 16383 Scaling: Speed (RPM) = TargetSpeed * (Maximum motor speed) / 16383
**TrqRef**—This is torque command, which is an input to the FOC block. This becomes the torque current command.
Range: 0 – 4095 Scaling: Current (A) = TrqRef * $I_{rated}$ / 4095

## 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.

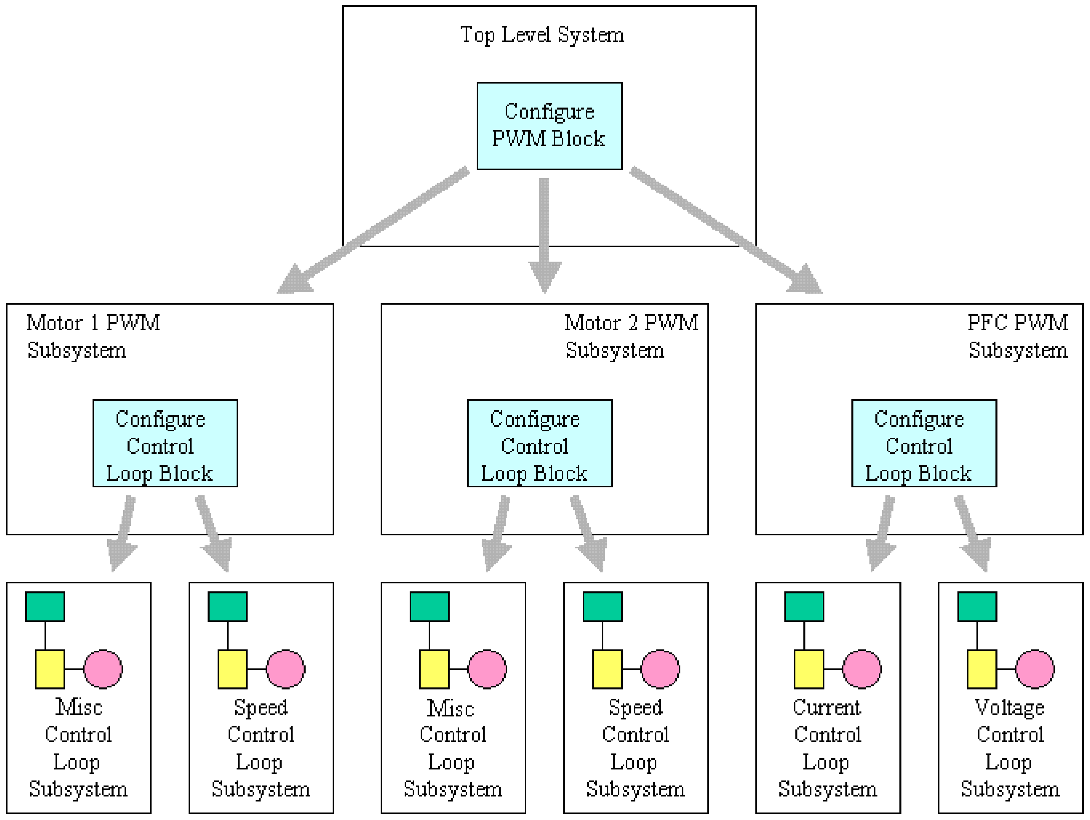

Top Level System
Configure PWM Block
Motor 1 PWM Subsystem
Motor 2 PWM Subsystem
PFC PWM Subsystem
Configure Control Loop Block
Configure Control Loop Block
Configure Control Loop Block
Misc Control Loop Subsystem
Speed Control Loop Subsystem
Misc Control Loop Subsystem
Speed Control Loop Subsystem
Current Control Loop Subsystem
Voltage Control Loop Subsystem

Figure 41—MCE Design Hierarchy

The top level of the system design contains a ***Configure PWM*** block and up to three PWM subsystem blocks, which are implemented using standard Simulink ***Enabled Subsystem*** blocks. The Configure PWM block has up to three outputs, labeled ***Motor 1***, ***Motor 2*** and ***PFC***. Each PWM subsystem is identified by connecting the appropriate 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.

Each of the PWM subsystems 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 of the Motor 1 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. The PFC subsystem has two outputs, ***Current*** and ***Voltage***. 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 up to ***Enabled Subsystem*** blocks into the model. Connect each output of the Configure PWM block to the Enable input of one of the subsystem blocks. Double click the label under each subsystem block to enter a name of your choice for the PWM subsystem.

**Step 3**
Double click the Motor 1 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**
Repeat Step 3 to create control loop subsystems for the Motor 2 and PFC PWM subsystems if required. In the PFC subsystem, remember to connect the Voltage output of the Configure Control Loop block instead of the Speed output.

**Step 5**
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$^{TM}$ 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 IRMCx300. The two basic types of hardware resources available on the IRMCx300 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$^{TM}$ to distinguish them from other elements. Each motion peripheral module is used only once in an application design (or once for each motor) 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$^{TM}$. 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.

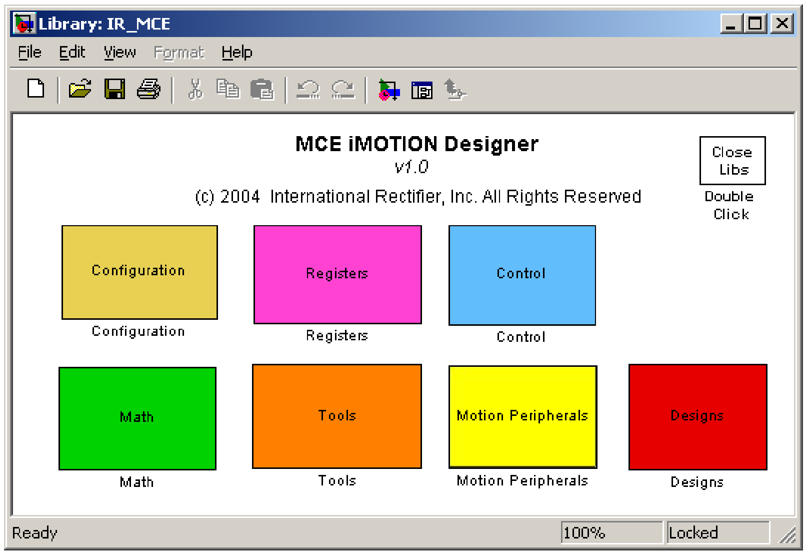


Figure 42—MCE Simulink Library

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 3.4.2.3) 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 which 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.

**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.)

## 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 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 IRMCx300. (See MCE Software Developer's Guide.)

### 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 IRMCx300.

**Step 1**
Start MATLAB, and in the MATLAB command window, type "mceinit" 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 IR Standard model file in the main MCE directory 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, size 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 Creating a Macro Block Definition

This section describes how to create a macro block, or subsystem block, that you can use in your system designs. If you want to create a complete system design for execution on the IRMCx3xx, refer to Section 3.4.2.

**Step 1**

Start MATLAB, and in the MATLAB command window, type `mceinit` 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 (empty) Simulink model file. Macro block definitions do not use the MCE subsystem hierarchy required for complete system designs.

**Step 3.**
Compose the design of your macro block. You can drag and drop blocks from the MCE libraries into the model. Use Simulink's graphical design features to arrange, size 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.

To define inputs and outputs for your macro block, use the standard Simulink input and output port elements. (Refer to Section 3.3.3 for details.)

Macro blocks **may not** include the following MCE and Simulink design elements:

- "Configure PWM" and "Configure Control Loop" blocks
- "Read Register" and "Write Register" blocks
- Other macro blocks
- Simulink Scope blocks
- Simulink Unit Delay blocks
- Subsystems

**Step 4.**
Encapsulate your macro block design elements in a masked subsystem. To create a subsystem, select all the components of the design and then select "Create subsystem" from the Simulink Edit menu. Simulink creates a subsystem block with input and output ports connected to it. **Delete the input and output ports and the lines that connect them to the subsystem block** so that only the subsystem block itself remains. The components of your design are inside the subsystem block and can be accessed by double clicking it. **Do not delete the input and output ports inside the subsystem.**

To mask the subsystem, click on the subsystem block and then select "Mask subsystem…" from the Edit menu. In the Mask Editor window, enter the name of your macro block as the "Mask type" and then click OK.

Once you have created a masked subsystem for your design, you can edit the components of the design by double-clicking the subsystem or by right-clicking on the subsystem and selecting "Look under mask" from the menu.

**Step 5.**
Enter the string `IR_MACRO` in the Tag field of your masked subsystem's block properties. To access the Block Properties window, right click on the subsystem block and select "Block Properties" from the menu. When you use the macro block in a system design, the compiler uses the `IR_MACRO` string to recognize the block as a macro, which requires special processing.

**Step 6.**
When you are satisfied with your Simulink design, it's time to run the compiler. This procedure is detailed in Section 3.4.3.

**Step 7.**

When your macro block is successfully compiled and ready to use, you can add it to the MCE "Designs" library group (see Section 3.3.2) so it's easy to drag the macro block into your system designs.

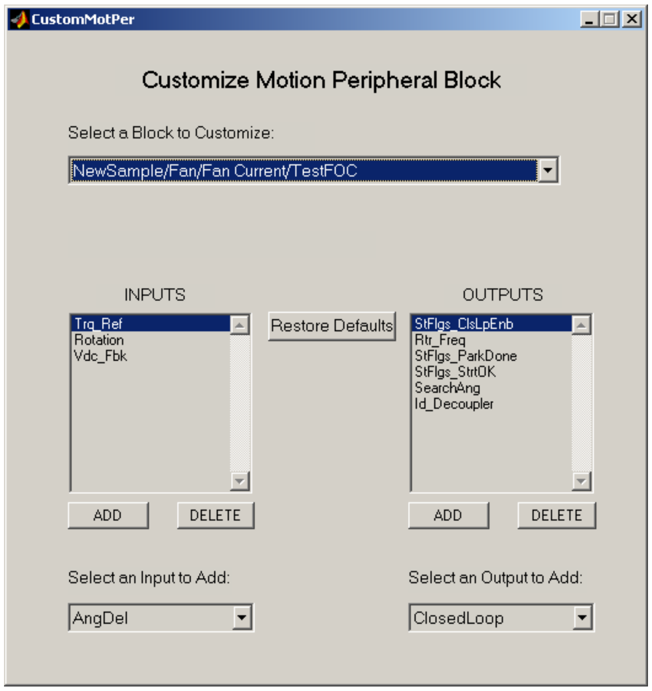
**Note:** When you add your macro block to the Designs library, the block definition is copied into the library model file, `Designs.mdl`. The library does not simply reference the original macro block model file. If you make changes to the original macro block model file, the macro block definition in the library file is not affected. To modify your macro block after you've added it to the Designs library, you should do one of the following: either edit the macro block by opening it directly from the Designs library; or edit the original macro block model file, then delete the old macro block from the Designs library and drag the newly modified block back into the library.

3.4.2.2 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.



NewSample/Fan/Fan Current/TestFOC

INPUTS: Trq_Ref, Rotation, Vdc_Fbk

OUTPUTS: StFlgs_ClsLpEnb, Rtr_Freq, StFlgs_ParkDone, StFlgs_StrtOK, SearchAng, Id_Decoupler

Select an Input to Add: AngDel

Select an Output to Add: ClosedLoop

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.)

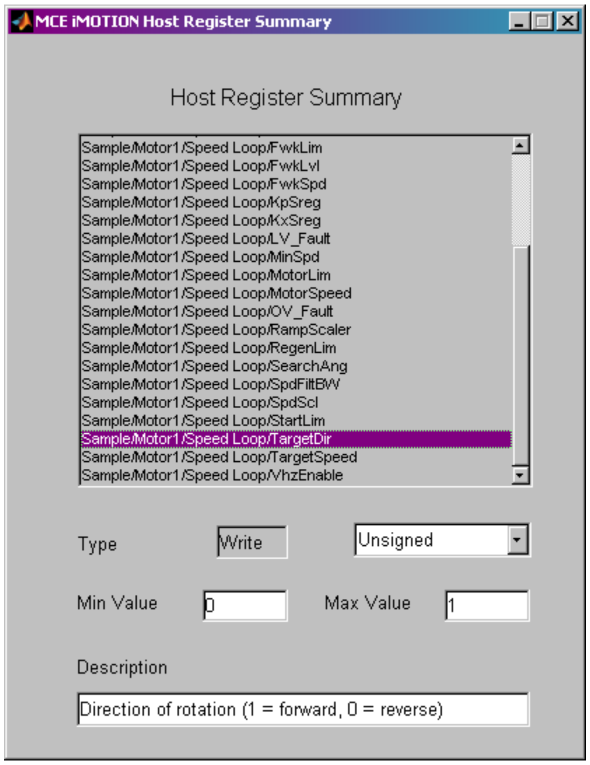
Once you've customized a block in your design, you can copy it to another location in the design (if the block is intended to be used once for each motor) or drag it into another design. For blocks that can be used once for each motor, you can customize each usage of the block with different inputs and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

#### 3.4.2.3 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.


MCE iMOTION Host Register Summary

Host Register Summary

Sample/Motor1/Speed Loop/FwkLim
Sample/Motor1/Speed Loop/FwkLvl
Sample/Motor1/Speed Loop/FwkSpd
Sample/Motor1/Speed Loop/KpSreg
Sample/Motor1/Speed Loop/KxSreg
Sample/Motor1/Speed Loop/LV_Fault
Sample/Motor1/Speed Loop/MinSpd
Sample/Motor1/Speed Loop/MotorLim
Sample/Motor1/Speed Loop/MotorSpeed
Sample/Motor1/Speed Loop/OV_Fault
Sample/Motor1/Speed Loop/RampScaler
Sample/Motor1/Speed Loop/RegenLim
Sample/Motor1/Speed Loop/SearchAng
Sample/Motor1/Speed Loop/SpdFiltBW
Sample/Motor1/Speed Loop/SpdScl
Sample/Motor1/Speed Loop/StartLim
Sample/Motor1/Speed Loop/TargetDir
Sample/Motor1/Speed Loop/TargetSpeed
Sample/Motor1/Speed Loop/VhzEnable

Type: Write, Unsigned

Min Value: 0 Max Value: 1

Description: Direction of rotation (1 = forward, 0 = reverse)

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.

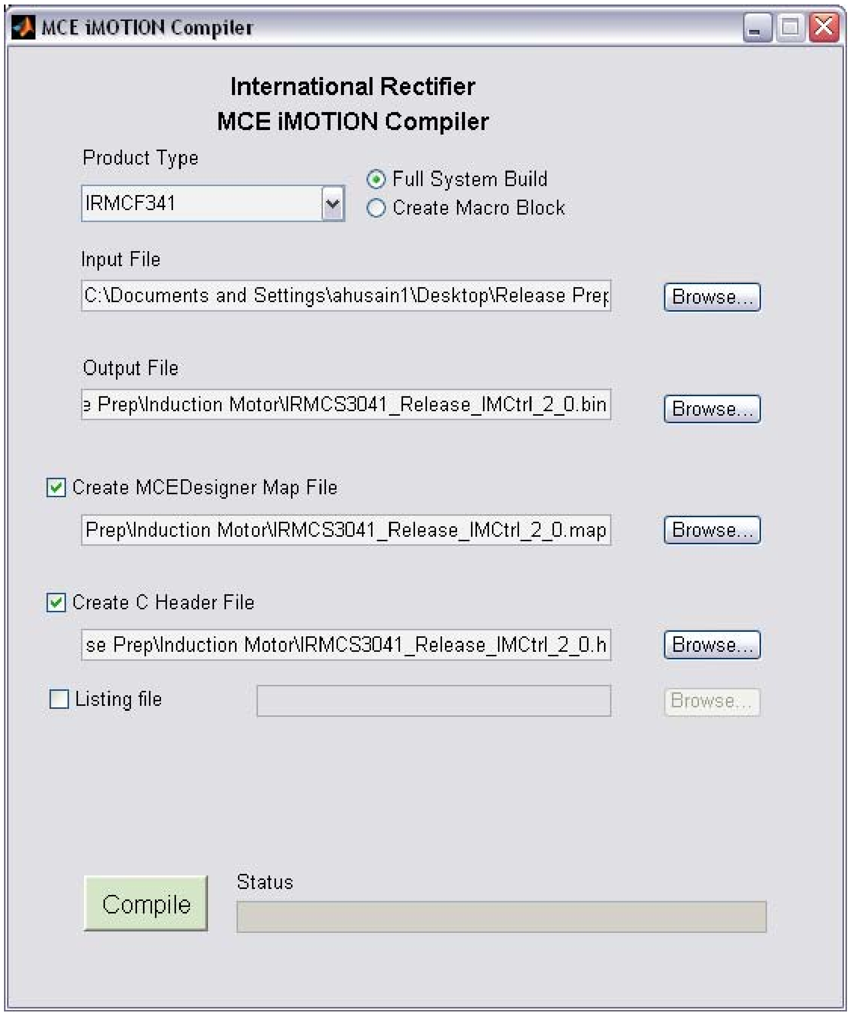
### 3.4.3 The MCE Compiler

**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 also access the compiler by copying that block into your design and double-clicking it. If you start with 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 MCE Compiler input screen appears as shown in Figure 45.


MCE iMOTION Compiler

International Rectifier
MCE iMOTION Compiler

Product Type: IRMCF341

[✓ Full System Build] Create Macro Block

Input File: C:\Documents and Settings\ahusain1\Desktop\Release Prep

Output File: e Prep\Induction Motor\IRMCS3041_Release_IMCtrl_2_0.bin

[✓] Create MCEDesigner Map File: Prep\Induction Motor\IRMCS3041_Release_IMCtrl_2_0.map

[✓] Create C Header File: se Prep\Induction Motor\IRMCS3041_Release_IMCtrl_2_0.h

[ ] Listing file

Figure 45—MCE Compiler Input Screen

**Step 1**
To compile a complete system design, click the "Full System Build" radio button. To compile a macro block definition, click "Create Macro Block" instead. For a complete system, 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".

The optional output files don't apply to a macro block compilation.

**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.

**Step 4**
Check the "Listing file" 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

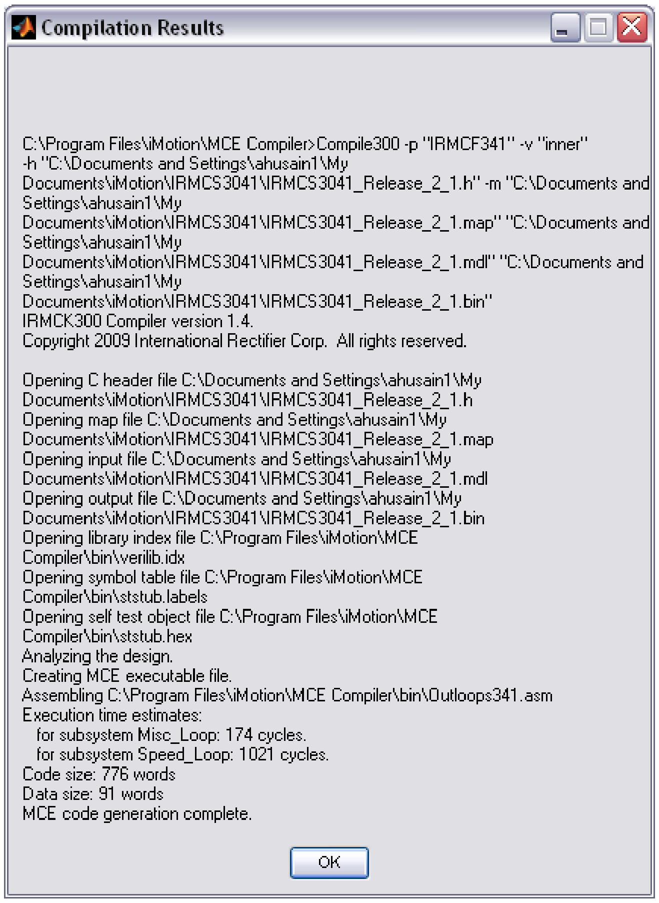
generated for either a full system or macro block compilation. You can use it as an aid in testing and verifying your design.

**Step 5**
Select the compiler version you want to use. The most recent version is selected by default.

**Step 6**
When you're ready, click the **compile** button to run the compiler. When compilation is complete, the MCE Compiler input screen is redrawn and you can scroll to the bottom of the window to see the output messages from the compiler. An example is shown in Figure 46.

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 the MCEWizard, 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.



```
C:\Program Files\iMotion\MCE Compiler>Compile300 -p "IRMCF341" -v "inner"
-h "C:\Documents and Settings\ahusain1\My
Documents\iMotion\IRMCS3041\IRMCS3041_Release_2_1.h" -m "C:\Documents and
Settings\ahusain1\My
Documents\iMotion\IRMCS3041\IRMCS3041_Release_2_1.map" "C:\Documents and
Settings\ahusain1\My
Documents\iMotion\IRMCS3041\IRMCS3041_Release_2_1.mdl" "C:\Documents and
Settings\ahusain1\My
Documents\iMotion\IRMCS3041\IRMCS3041_Release_2_1.bin"
IRMCK300 Compiler version 1.4.
Copyright 2009 International Rectifier Corp.  All rights reserved.

Opening C header file C:\Documents and Settings\ahusain1\My
Documents\iMotion\IRMCS3041\IRMCS3041_Release_2_1.h
Opening map file C:\Documents and Settings\ahusain1\My
Documents\iMotion\IRMCS3041\IRMCS3041_Release_2_1.map
Opening input file C:\Documents and Settings\ahusain1\My
Documents\iMotion\IRMCS3041\IRMCS3041_Release_2_1.mdl
Opening output file C:\Documents and Settings\ahusain1\My
Documents\iMotion\IRMCS3041\IRMCS3041_Release_2_1.bin
Opening library index file C:\Program Files\iMotion\MCE
Compiler\bin\verilib.idx
Opening symbol table file C:\Program Files\iMotion\MCE
Compiler\bin\ststub.labels
Opening self test object file C:\Program Files\iMotion\MCE
Compiler\bin\ststub.hex
Analyzing the design.
Creating MCE executable file.
Assembling C:\Program Files\iMotion\MCE Compiler\bin\Outloops341.asm
Execution time estimates:
  for subsystem Misc_Loop: 174 cycles.
  for subsystem Speed_Loop: 1021 cycles.
Code size: 776 words
Data size: 91 words
MCE code generation complete.
```

Figure 46—MCE Compiler Results Example

### 3.4.4 Downloading to the Reference Board

There are two different choices for downloading to the reference board, both available through MCEDesigner. 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. "Download to EEPROM" allows the designer to reprogram the boot EEPROM with the new MCE program. Once the code is stored in EEPROM, the IC automatically loads and executes it whenever the target platform is powered up or reset.

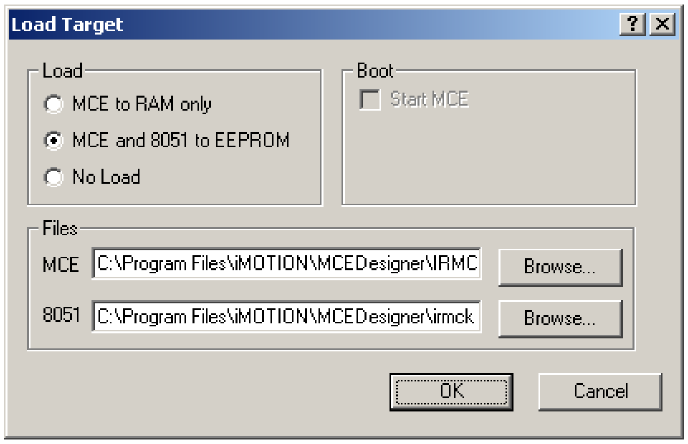
#### 3.4.4.1 Download to EEPROM

The code for the 8051 microprocessor and the MCE are stored in EEPROM together as a single boot image. Therefore, when you program EEPROM, you must always download code for both processors at the same time.

Use the following procedure to download code and store it in EEPROM on the target platform:

1. Start MCEDesigner and open a configuration 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. (See Section 3.4.4.3)
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 and 8051 to EEPROM" radio button as shown in Figure 47. The Boot box is not used when loading to EEPROM.

5. In the Files box, enter or browse for the pathnames of your MCE and 8051 program files. The MCE download file is generated by the MCE Compiler and has the filename extension ".bin". The 8051 download file must be in Intel hex format, which can be generated from the Keil uVision3 IDE. This file has the extension ".hex." (See the Software Developer's Guide for more information.) MCE Designer Agent is the .hex file which came with the Reference Design Board.
6. When you have selected valid download files for *both* processors (MCE and 8051), click the OK button in the Load Target dialog.
7. A status bar will show the progress of transferring files to the IC over the serial link and programmed to EEPROM. When programming is complete, the message "Load complete." is displayed.
   — If errors occurred during EEPROM programming, the message "Load complete but checksums don't match" is displayed instead. *If this occurs, do not power cycle or reset the board!* Attempt the download process again. If an error occurs again contact an IR FAE.
8. The downloaded code does not execute until a power cycle or reset is performed on the reference board.



Load: ( ) MCE to RAM only [✓ MCE and 8051 to EEPROM] ( ) No Load

Boot: ☐ Start MCE

Files:
MCE C:\Program Files\iMOTION\MCEDesigner\IRMC
8051 C:\Program Files\iMOTION\MCEDesigner\irmck

Figure 47—Load MCE and 8051 to EEPROM

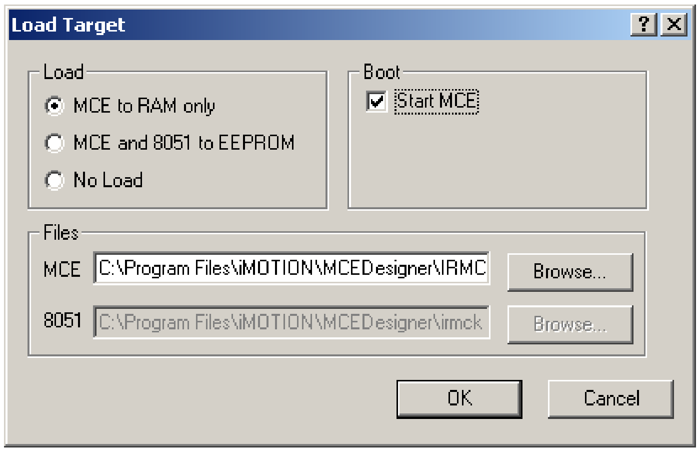
### 3.4.4.2 Download to RAM

If you've recompiled your MCE design and want to do a quick test without programming the code to EEPROM, 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 shared RAM on the IRMCx IC:
1. Start MCEDesigner and open a configuration 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 48. 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.)


Load Target

Load: [● MCE to RAM only] ○ MCE and 8051 to EEPROM ○ No Load

Boot: [✓ Start MCE]

Files

MCE: C:\Program Files\iMOTION\MCEDesigner\IRMC

8051: C:\Program Files\iMOTION\MCEDesigner\irmck

Figure 48—Load MCE to RAM

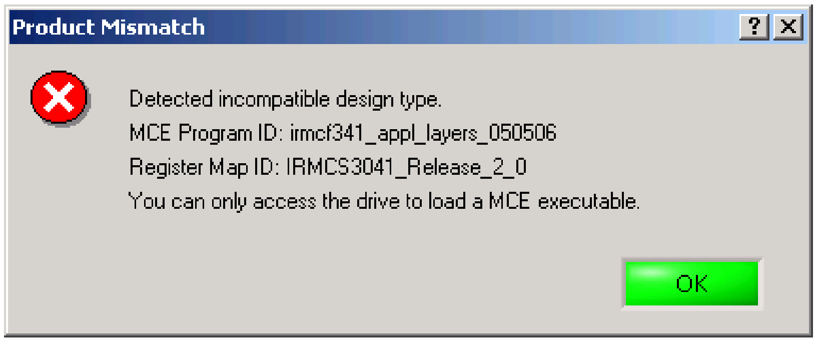
If you don’t check the “Start MCE” checkbox when you download your MCE code, you need to perform a separate operation to start execution of the code. To do this, open the Load Target dialog again. This time, click the “No Load” radio button in the Load box and check the “Start MCE” checkbox in the Boot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

#### 3.4.4.3 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 a 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. 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 configuration file. (See the MCEDesigner User’s Guide for more information.)

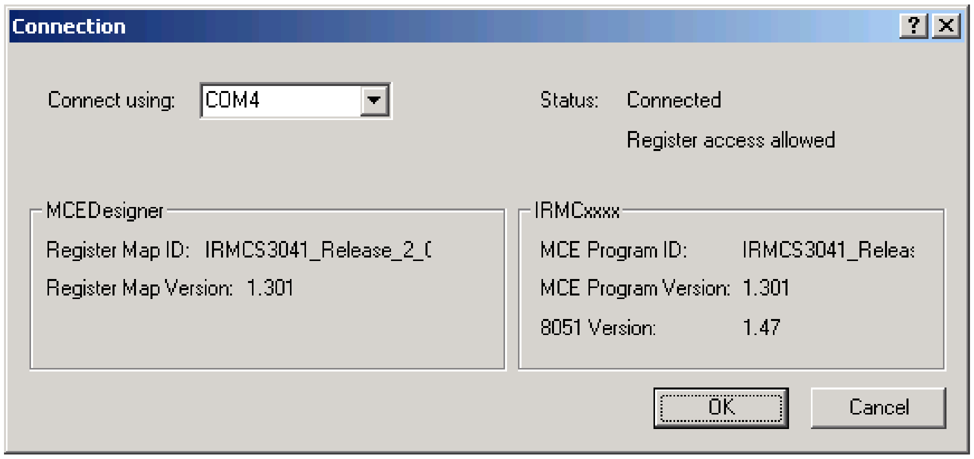
If the design ID or version number of the MCE program currently loaded to the IRMCx300 IC does not match the ID and version specified for the current Register Map (or if no MCE image is currently loaded), an error message similar to the one shown in Figure 49 is displayed. You can continue using MCEDesigner in an “offline” mode, but you cannot read or write any registers or execute any functions until the problem is corrected.



Detected incompatible design type.
MCE Program ID: irmcf341_appl_layers_050506
Register Map ID: IRMCS3041_Release_2_0
You can only access the drive to load a MCE executable.

Figure 49—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 50.



Connect using: COM4
Status: Connected
Register access allowed

Register Map ID: IRMCS3041_Release_2_(
Register Map Version: 1.301

MCE Program ID: IRMCS3041_Releas
MCE Program Version: 1.301
8051 Version: 1.47

Figure 50—Design ID and Version Details

### 3.4.4.4 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.

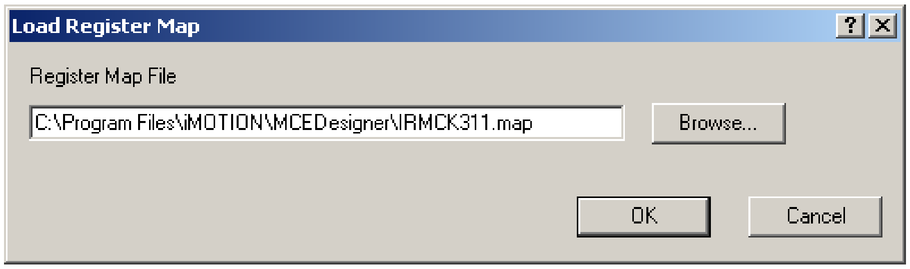
When you import a new MCE register map, MCEDesigner modifies the Register Structure Definition section of the database as follows:

- 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

"pwmcfg", the name will be changed to "pwmcfg_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 51) 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 file.



C:\Program Files\iMOTION\MCEDesigner\IRMCK311.map

Figure 51. 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 52 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.

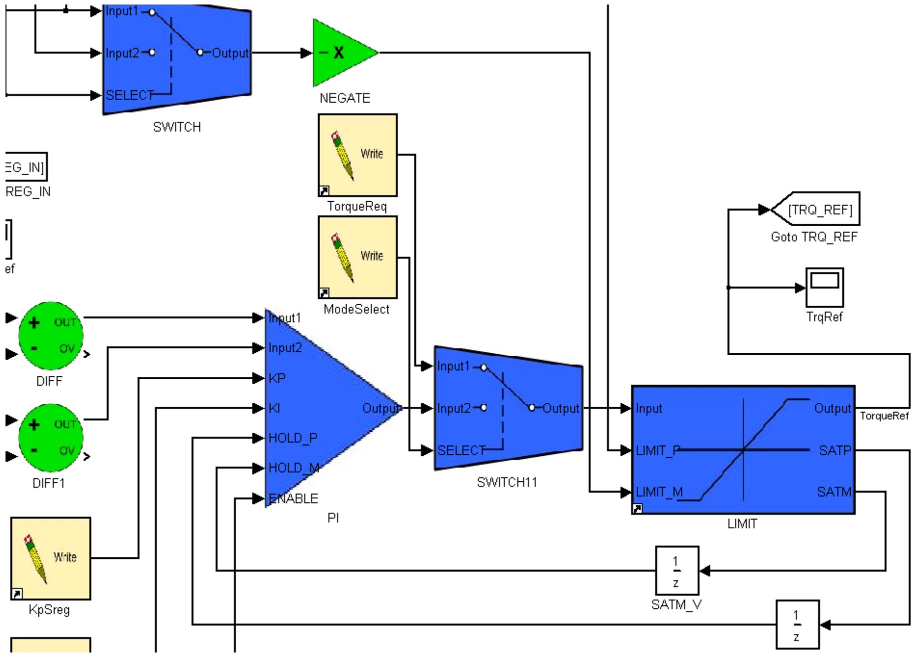


Figure 52—Torque Mode Block Diagram

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 the MCEWizard.

- 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 MCEWizard 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 53 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.

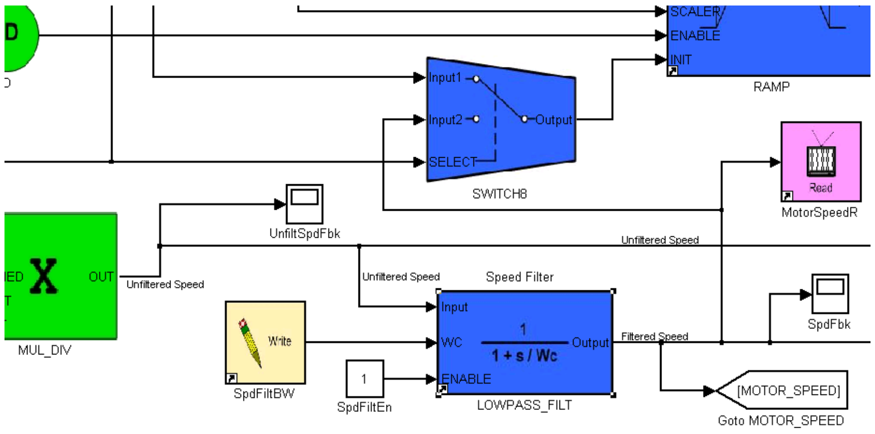


Figure 53—Speed Feedback Section

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 Application Hardware Design

This final chapter of the Application Developer's Guide will discuss hardware design issues specific to the IRMCx300 series ICs. Section 4.1 will discuss 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 overcurrent protection. Section 4.2 will give layout recommendations relating to the current feedback circuit, and Section 4.3 will finish with some suggestions for testing the board and optimizing the settings. This chapter will reference MCEWizards 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 will begin by giving specifications and tips related to component selection for various in-circuit functions. Next, it will review the feedback scaling for current and voltage as related to entering values into MCEWizard. This section will also discuss techniques for A/D offset compensation and proper overcurrent protection.

### 4.1.1 Component Selection

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

**Power Supplies**—Figure 54 below plots the supply currents required to provide power to the control IC. The 1.8V digital supply current (VDD2) is strongly dependent on the clock frequency used in the IRMCx300. For more information on setting the clock frequency, see the Software Developer's Guide or the Reference Manual.

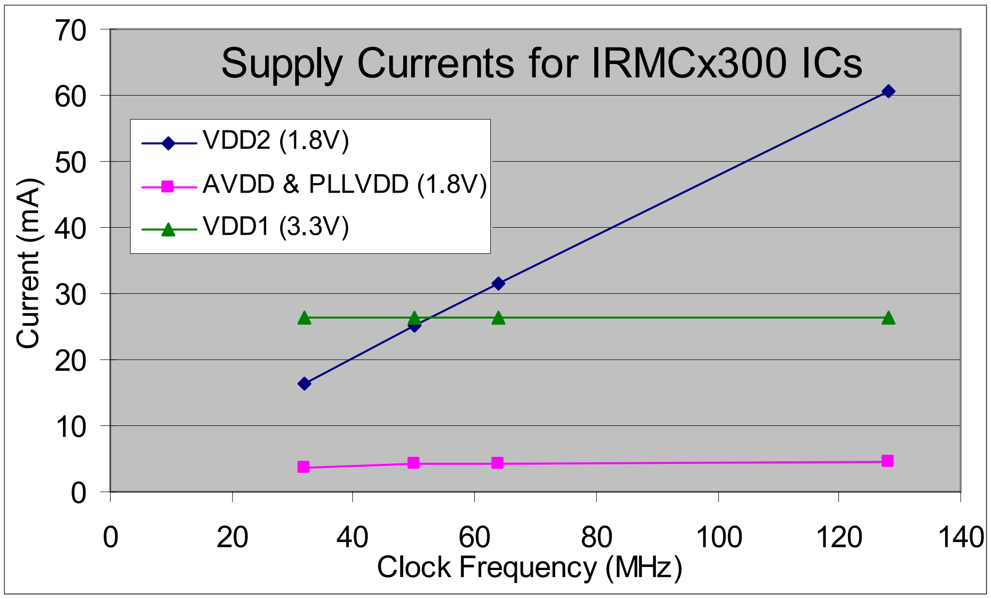


Figure 54—Supply Currents for the IRMCx300

**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 chosen by the user. Details of configuring the PLL to generate the internal clock are found in the Software Developer's Guide and the Reference Manual. Note that the IRMCS3041 and IRMCS3012 utilize a quartz crystal oscillator while the IRMCS3043 uses a ceramic chip resonator.

**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 has a TCK frequency of 500kHz.

**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 go up to 155,200 baud rate.

**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.

**EEPROM**—If the RAM version (IRMCF3xx) of the IC is used, an external EEPROM is required to provide boot code, which is automatically loaded at power-up. The EEPROM should be capable of a maximum data rate of 400kHz for the purpose of boot loading, but the data rate is configurable for other purposes. For specific information about the EEPROM data format for boot loading, see the Software Developer's Guide.

At boot up, the EEPROM communication protocol is selected by pin P1.3. Pull this pin up to VDD1 (3.3V) for I2C boot and down to VSS for SPI.

**Decoupling Capacitors**—The IRMCx300 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 33pF capacitor be placed between the op-amp output and the analog ground (C33 of Figure 56).*** The capacitor choice will influences 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 4.3.2.2 for more discussion on choosing the minimum pulse width and the sampling delay.

### 4.1.2 A/D Feedback Scaling

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

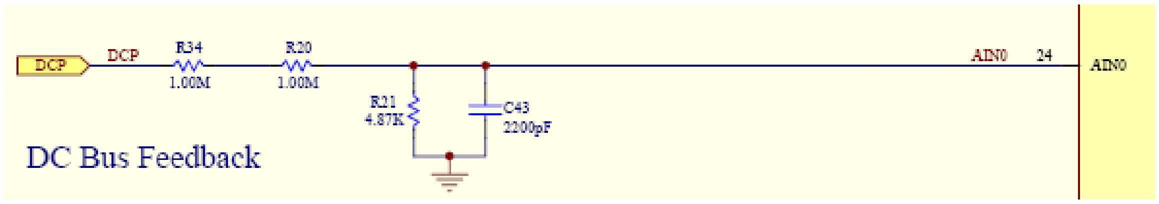


Figure 55—DC Bus feedback voltage divider

DC Bus feedback Scaling—This MCEWizard input 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:

$$\text{Scaling (cts/V)} = 4095 * ( 4.87k / ( 1.00M + 1.00M + 4.87k )) / 1.2 = 8.3 \text{ cts/V}$$

Current Feedback Amplifier Gain—This MCEWizard 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. For example, in Figure 56, *Current Feedback Amplifier Gain = 11.8k / ( 5.11k + 1.00k ) = 1.93*

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

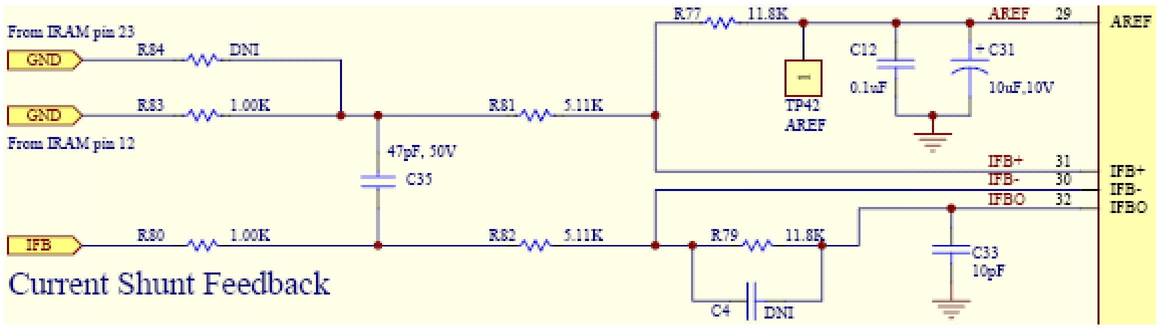


Figure 56—Current Feedback Circuit

The Verify & Save page of the MCEWizard 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 the MCEWizard 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 IRMCx300 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, PFC, and field-weakening capability as well. These effects can be minimized through the use of an external reference for offset compensation.

IRMCx300 series controllers have a single Analog-to-Digital converter, which is multiplexed for motor current feedback, as well as DC bus measurements. A second multiplexer is employed to provide customer configurable ADC measurements. One of these configurable ‘AINx’ pins can be used for offset compensation through an external reference. A separate technique employs the differential amplifiers used for current feedback to reference an external source to compensate ADC measurements. Because these inputs route to the same ADC, an offset measurement made on one channel can be applied across the system.

| Compensation Type | Advantage | Disadvantage |
|---|---|---|
| AINx Input (Recommended Method) | Low cost, no performance degradation, can use already present 1.8V supply | Requires one AINx pin |
| Current Feedback (Ifb Op-Amps) | Does not require an AINx pin | Higher cost for external non-switch mode reference, possible noise coupling to Ifb with switch mode reference |

In the case of non-PFC equipped models there is no real danger created by inaccurate DC bus measurements. The worst-case situation would be the controller detecting an under or over-voltage condition on the DC bus and turning off the gating signals to the 3-phase inverter. In the case of PFC equipped controllers, an offset in DC bus measurement will cause the system to regulate it incorrectly. This regulation offset, if left unchecked, could exceed the limits of the bus capacitors and cause a catastrophic failure. It is also important to note that the techniques outlined here will directly affect the PFC regulation (in PFC equipped systems), and if improper compensation is applied the bus voltage may be significantly different than expected.

This technique utilizes an external reference to compensate for any offset present in the ADC. This offset can be realized by sampling an accurately known reference source with the ADC. The difference between the measured and actual voltages will result in the offset of the ADC. This value can be then subtracted from the DC bus to yield a considerably more accurate measurement and regulation point for PFC equipped systems.

Two methods are described to use an external reference as compensation. The first uses an available AINx pin and the 1.8V voltage supply rail as a reference. The benefit of using the already present 1.8V supply is that a dedicated external reference is not required, but the accuracy will not be a good as with an external reference chip. The second utilizes a differential op-amp used for current feedback, where an external reference is supplied by a dedicated source. Both of these techniques are suitable for a situation where high accuracy is required over a small range of input voltages, as in PFC DC bus sensing. Depending on the application, more elaborate compensation techniques may be required.

Note that the feedback offset is not an issue for the current feedback sensing because the IC hardware already has an offset compensation feature for current feedback.

#### 4.1.4.1 External Reference for AINx

A good place to acquire a moderately accurate voltage reference is directly from the output of a switch-mode power supply. The IRMCx300 series controllers require a 1.8V supply, which can be used as a reference for DC bus measurements. This compensation will rely on the accuracy of the chosen power supply, but any 2% regulator will be sufficient. An external reference should be used if even greater accuracy is required.

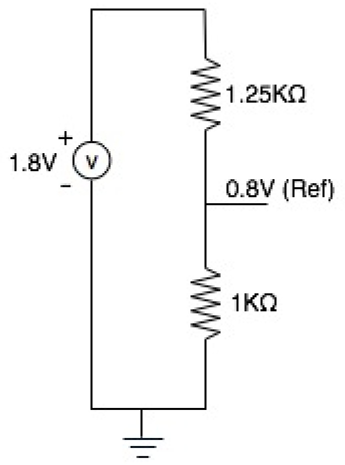


Figure 57—Voltage divider to create reference voltage

In its simplest form a voltage divider can be implemented to bring the 1.8V supply within the 1.2V measurement range (Figure 57). This voltage reference point is important in achieving an accurate DC Bus measurement. The reference point should be equivalent to the voltage the ADC will receive when the DC Bus is at its target operating point, although a 0.6V generic reference is also acceptable.

For example, take the gain between the DC bus voltage and ADC as 0.00243 (as it is in IRMCx300 Reference Design Kits). If the nominal DC bus voltage were 350V, then a voltage reference of 0.851V should be used, as in the calculation below. The algorithm to implement this compensation method is covered in the next section.

*'Nominal Bus' * 'Bus to ADC Gain' = Target Reference*
*350V * 0.00243 = 0.851V*

4.1.4.2 External Reference for Ifbk Op-Amps

This voltage source can then be used for the differential op-amp reference already in place for motor current feedback (Figure 58). Using an external reference connected to R2 will allow for the output voltage of the differential op-amp to be 0.6V when no current is present. The ADC would then measure this output, where an offset from the ideal reference (0.6V) can be calculated. This offset can then be applied as compensation to DC bus measurements. The algorithm to provide this offset calculation can be implemented in the MCE.

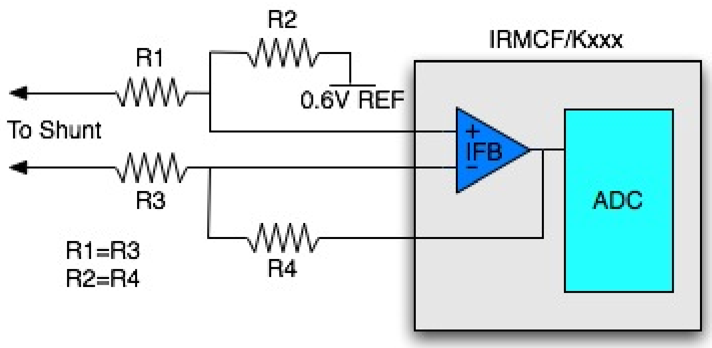


Figure 58—External reference to current feedback circuit

### 4.1.4.3 Performing Compensation Calculations

Now that a reference voltage is supplied to the ADC, the controller must calculate the real DC Bus voltage. This process can be implemented in either the MCE or in the 8051 side of the controller. This application note covers implementing these offset calculations in the MCE. Figure 59 illustrates how the entire solution can be realized within the MCE for the AINx based compensation method.

In this example ADC channel 6 is used as the compensation input. Its value will first go through a low-pass filter to remove any system noise. Next a compensation value will be subtracted from this measurement. The compensation value is the ADC reference target value in digital counts, as calculated below. This subtraction will yield the offset value of the ADC. Then by subtracting this offset from the raw DC Bus voltage measurement the compensated bus voltage is created. This compensated voltage can then be connected to anywhere the raw bus voltage would be.

*( 'Reference Voltage' / 1.2V ) * 4095 = 'Compensation'*
*( 0.851 / 1.2 ) * 4095 = 2904*

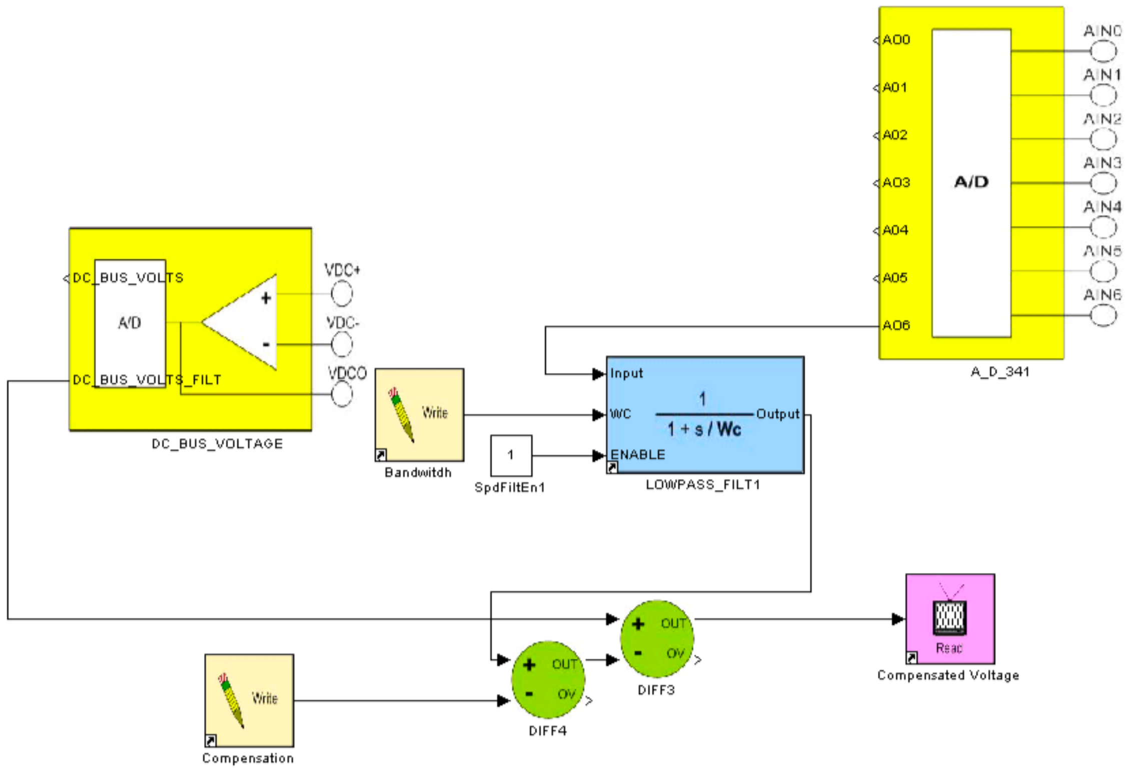


Figure 59—AINx based MCE Compensation

### 4.1.4.4 External Voltage Protection

If a variable DC Bus voltage is not an issue in a particular application, or you do not have an available ADC input for compensation, external over-voltage protection could be employed instead. Because the over-voltage protection in the controller relies on the ADC for accuracy, the system may not properly shut down in the event of an over voltage. By designing an external comparator to signal the controller when an over voltage has occurred (either through general digital input or interrupt) potential safety issues can be mitigated. The same can be implemented for under-voltage protection if necessary. This method, though, will only prevent system damage and does not improve the ability to read DC Bus voltages.

### 4.1.5 Overcurrent Protection

Overcurrent protection circuitry is required by most applications to prevent damage to the hardware and the motor. The IRMCx300 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. Also, there is an 8051 special function register, STOPS, which should be configured properly for the GATEKILL function. See the Reference Manual for details on using setting STOPS.

In some types of IRAM modules which include a shunt resistor, the gate drive IC already has overcurrent protection built in, with a pin assigned to signal the shut down 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 away of the overcurrent shut down, and stops the PWM signals.

## 4.2 Layout Recommendations

### 4.2.1 Current Feedback Circuit with IRMCF300

The IRMCx300 series IC has the necessary circuitry to implement single 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 4.3.2.2.

Figure 60 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 63) is used as the input to the amplifier, whose gain [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.

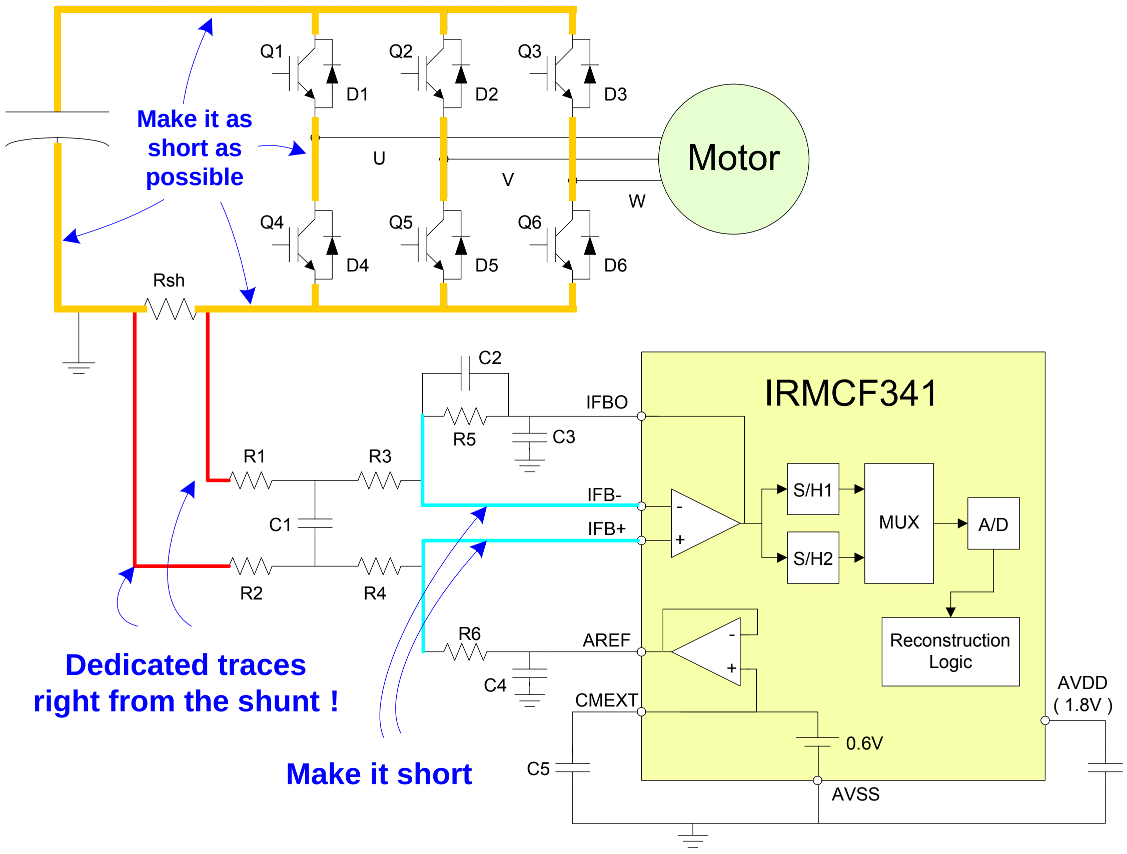
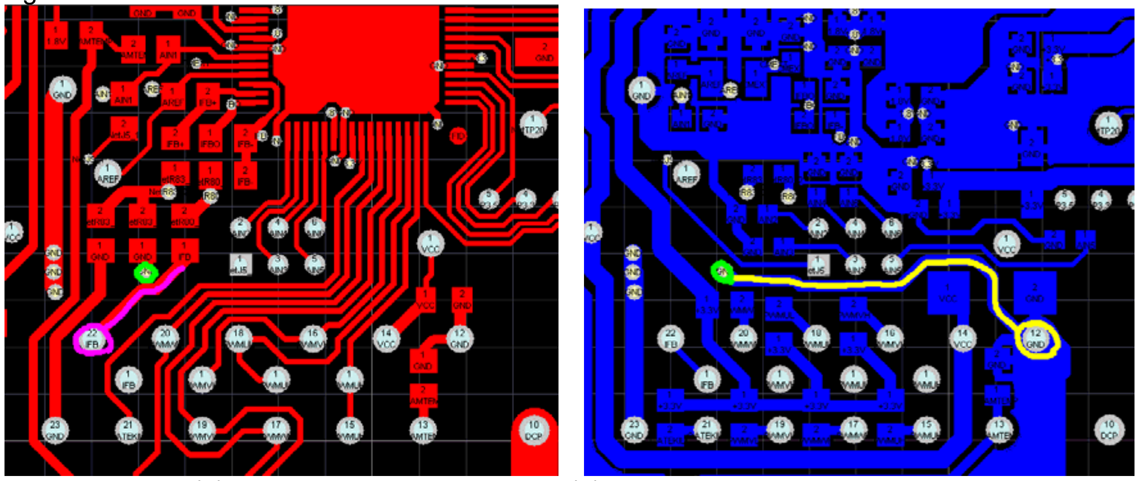
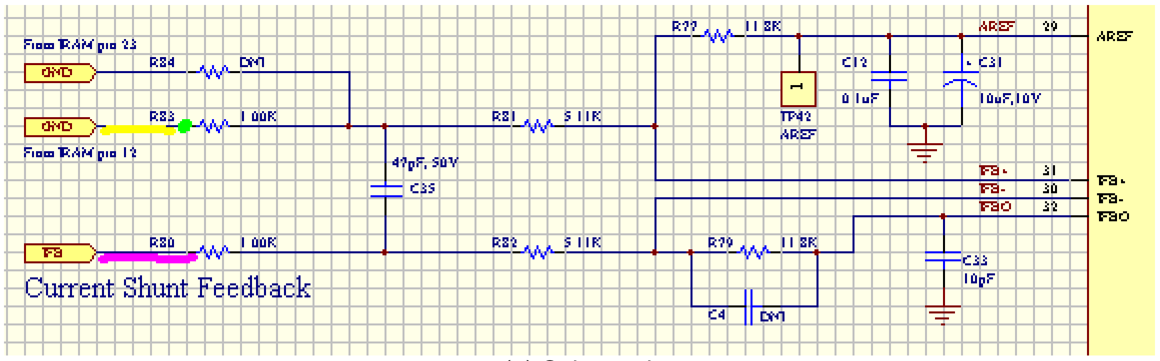


Figure 60—Current Feedback Circuit for IRMCF341

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 61 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.



(a) Top Layer (b) Bottom Layer



(c) Schematic

Figure 61— IRMCS3041 Reference Board Layout

Another very important issue is that noise from a switching power supply may significantly influence the current feedback. It is recommended to separate the IRMCF300 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.

### 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 IRMCx300 IC has registers which can be configured to help optimize the performance of the motor drive system. This section will discuss the techniques to test critical aspects of the system and then set register values (or MCEWizard fields) based on the tests. To begin, a detailed description of single shunt current reconstruction is presented in the next section.

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 Single Shunt Current Reconstruction

The IRMCx300 Series IC employs 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 3-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 62 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 63 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).

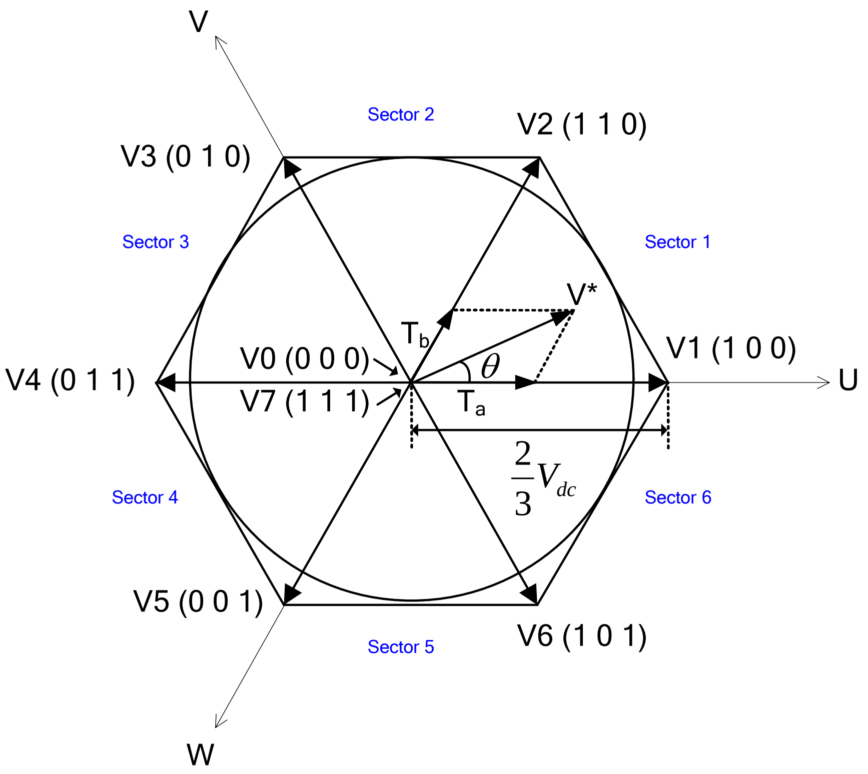


Figure 62—Inverter Output Voltage Space Vectors

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 60. 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 63; 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, pwmcfg, on the IRMCF300.

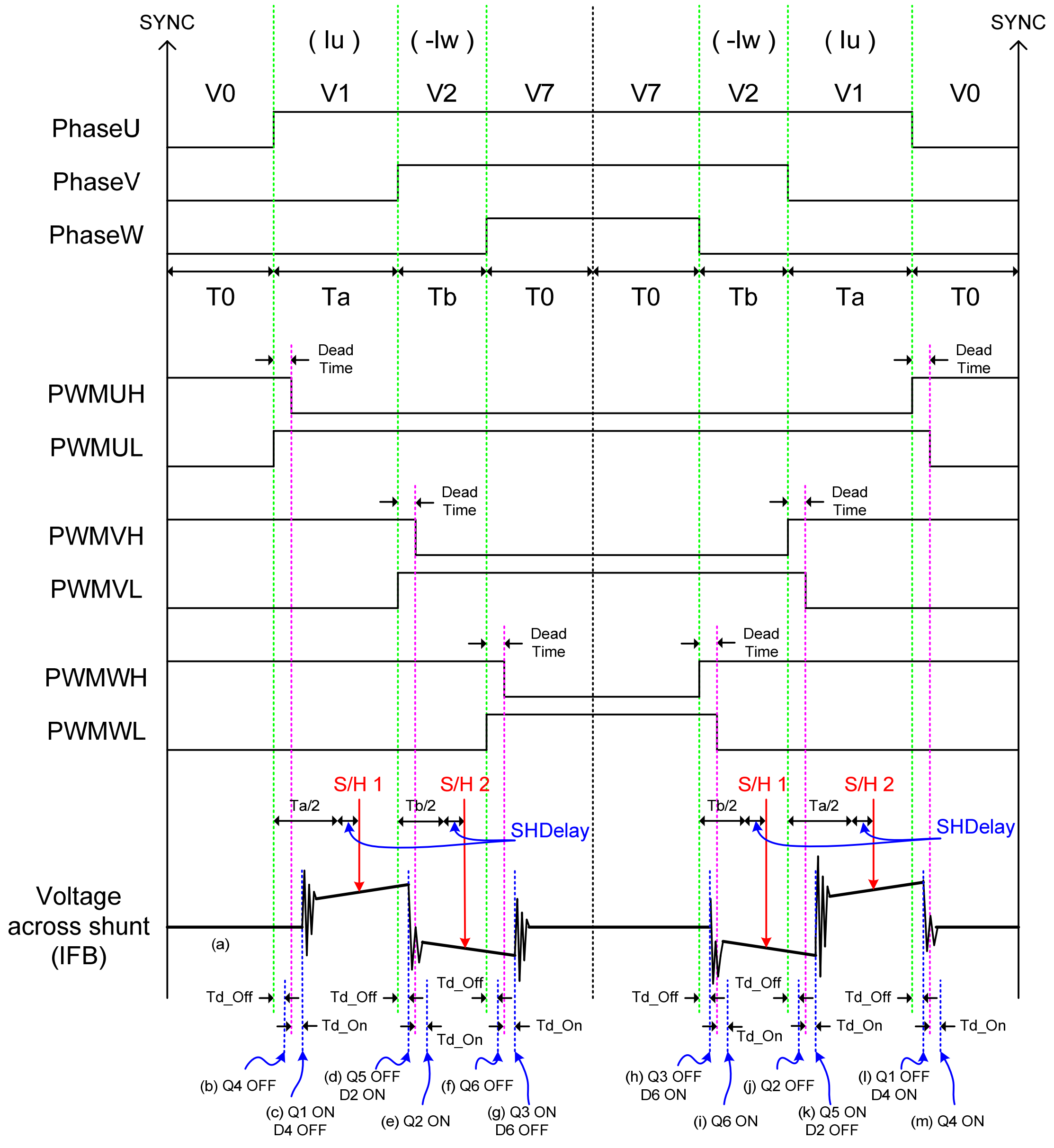


Figure 63—PWM Gate Signals in Sector1

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.

### 4.3.2 Inverter-Related Testing and MCEWizard 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 will describe registers and settings to align the controller with the inverter.

#### 4.3.2.1 Miscellaneous MCEWizard Settings

**Disable Second PWM and/or PFC**—Select the IRMCx300 series product in use in the MCEWizard Welcome Page and the Wizard will automatically set several registers to properly configure the drive.

**Gate Sense section**—Check the specifications of the gate driver IC to set these MCEWizard fields correctly. These parameters will set the logic sense for the PWM gating signals as well as the GATEKILL signal. The logic level to assert GATEKILL is dependent on the hardware implementation.

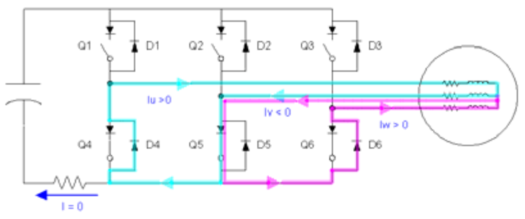
**Inverter Dead Time**—Dead time is used to prevent shoot-through, a condition where both the high side and low side IGBTs are on at the same time, which can damage the inverter components. To choose the dead time, carefully check the turn-on and turn-off times of the IGBTs. Also, the gate driver IC 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 Section 4.3.5.5 of the IRMCx300 Reference Manual.

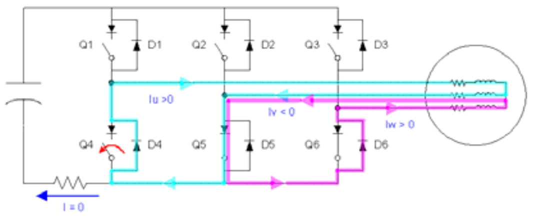
#### 4.3.2.2 Current Feedback Sample Timing

The current sampling instant should be at the mid point 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 63. 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.

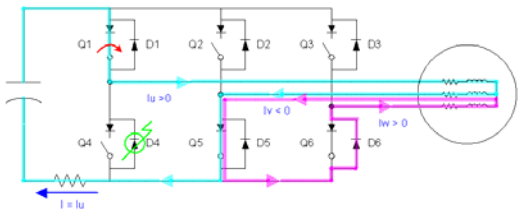
Successful implementation of motor current reconstruction requires detailed knowledge of power inverter operation to account for circuit delays that can result in incorrect current sampling. The error introduced by sampling delays depends on the magnitude of the motor current ripple, which depends on the bus voltage, switching frequency winding inductance and motor back emf. The IRMCx300 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 63 illustrates how to calculate the sampling delay compensation. Figure 64 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.



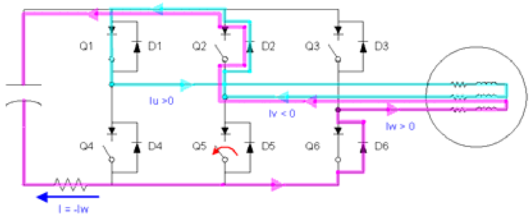
(a) Zero Vector V0



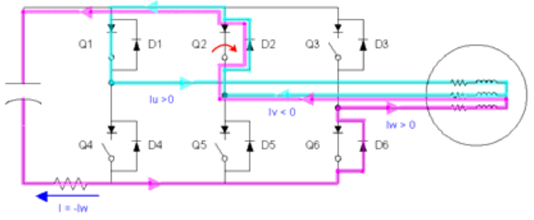
(b) Q4 OFF



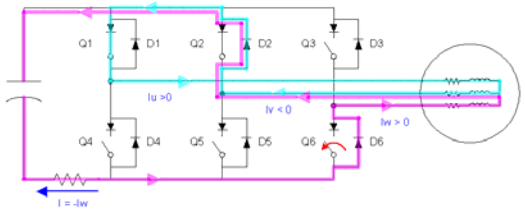
(c) Q1 ON, D4 OFF



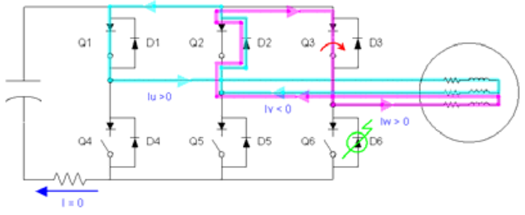
(d) Q5 OFF, D2 ON



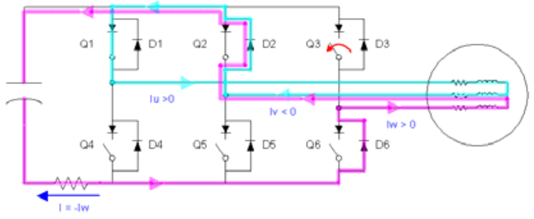
(e) Q2 ON



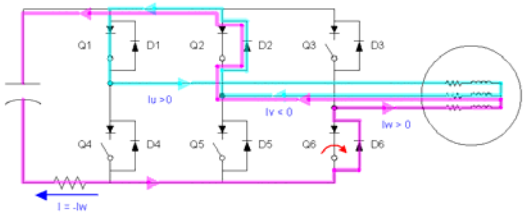
(f) Q6 OFF



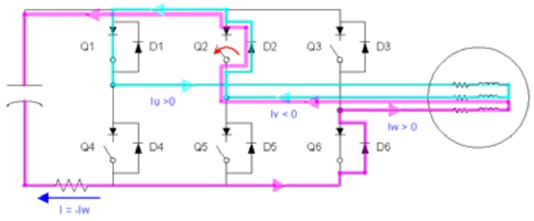
(g) Q3 ON, D6 OFF



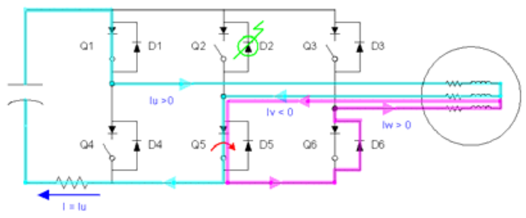
(h) Q3 OFF, D6 ON



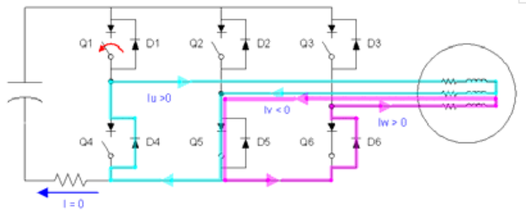
(i) Q6 ON



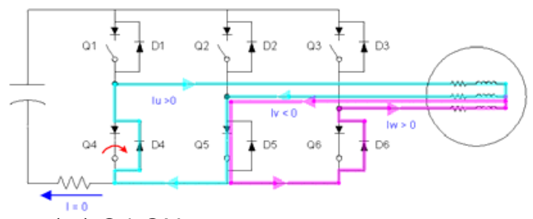
(j) Q2 OFF



(k) Q5 ON, D2 OFF



(l) Q1 OFF, D4 ON



(m) Q4 ON

Figure 64—Current Flow Example in Sector 1

There's 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 63 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 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 65.

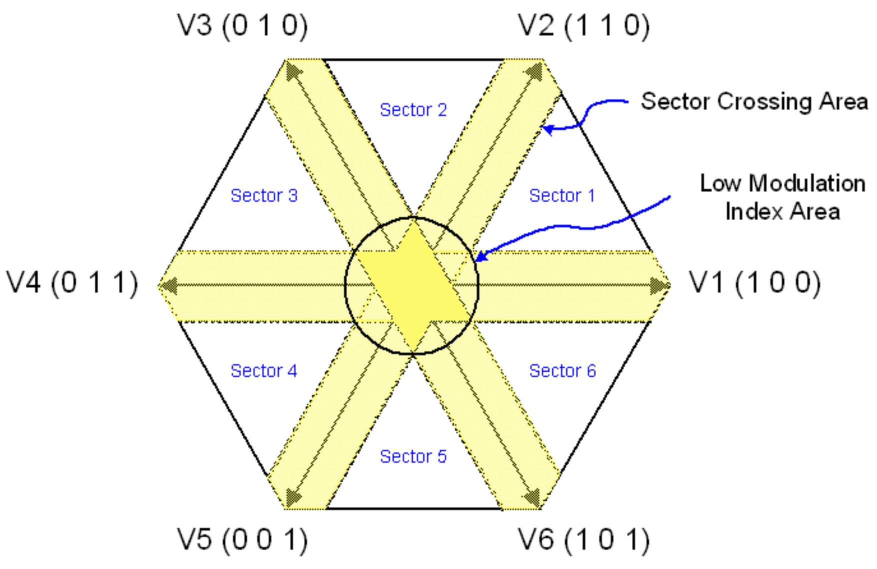


Figure 65—Areas Where Reliable Sampling is Difficult

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 63, 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 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 the MCEWizard there are two input parameters whose sum 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.

#### 4.3.2.3 An Example of Optimizing the Current Feedback

Figure 66 shows the real waveforms for V* in sector 1 of Figure 62. 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.

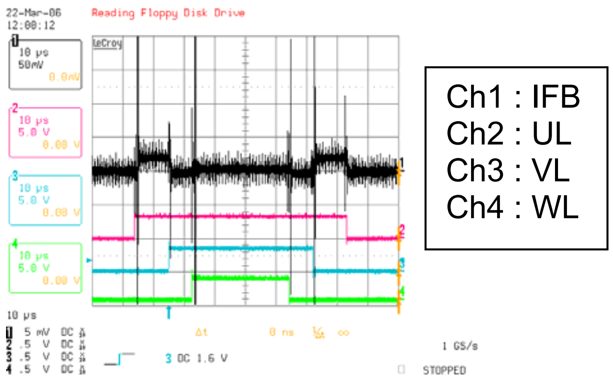
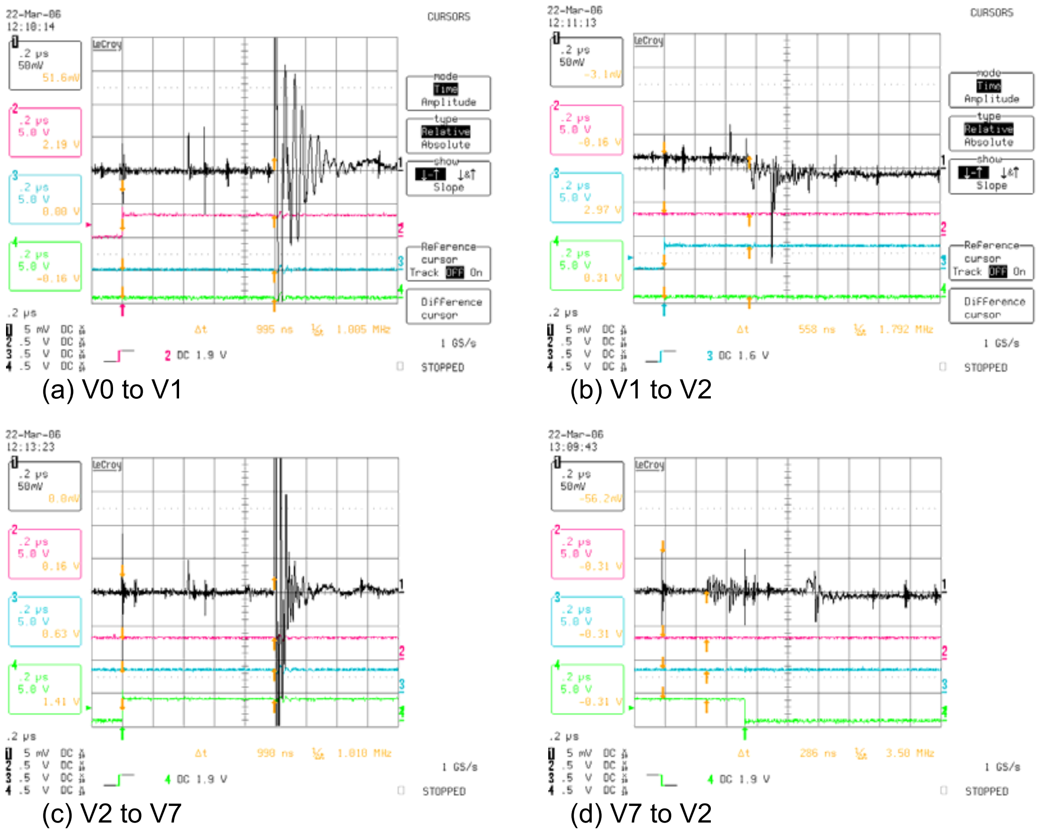
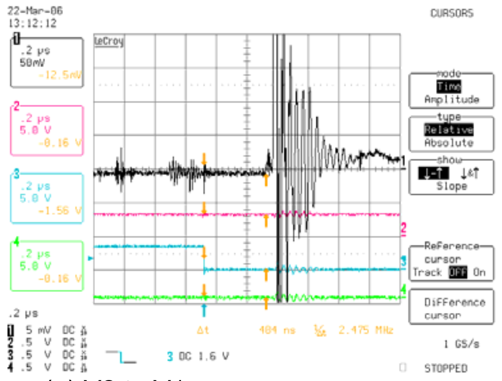


Figure 66—Waveforms in Sector 1

Figure 67 is a collection of waveforms when active vector changes. It will be better to understand together with Figure 63 and Figure 64. 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.




(e) V2 to V1

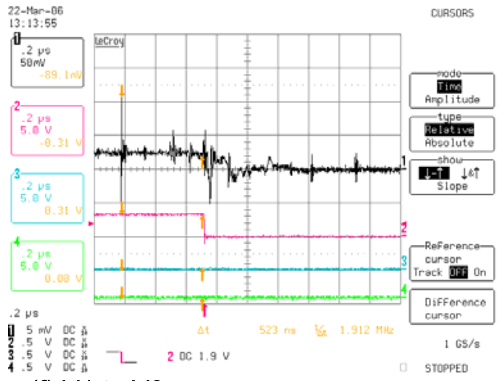
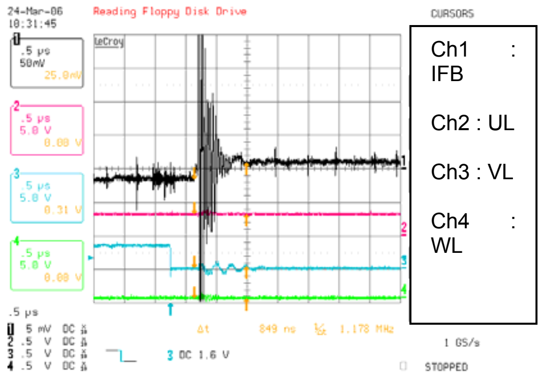
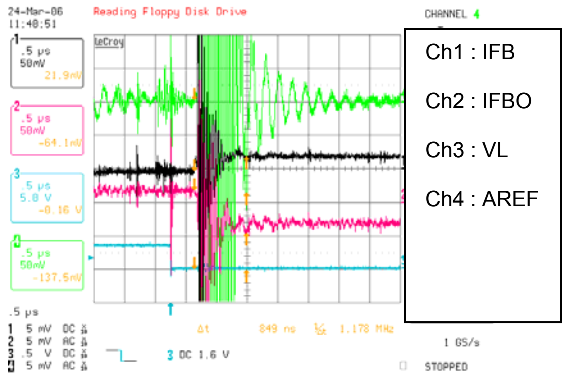

(f) V1 to V0

Figure 67—Vector Transition Waveforms in Sector 1

The waveform in Figure 68(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's also slow component which ends in 0.85 µsec. However, the operational amplifier output (IFBO) is the one which needs attention here because this is the input to the sample&hold. Figure 68(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

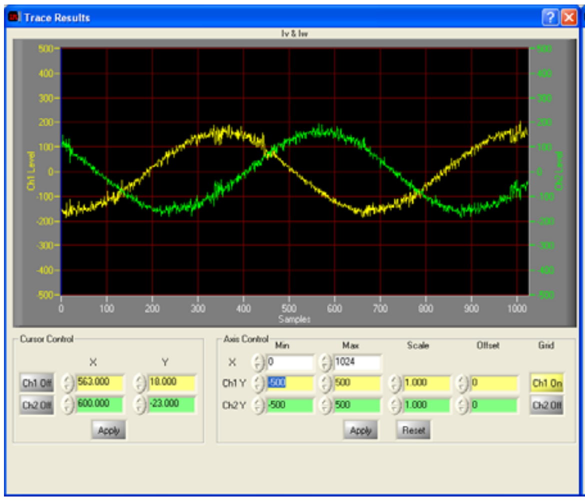
Figure 68—Ringing at Transition from V2 to V1

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

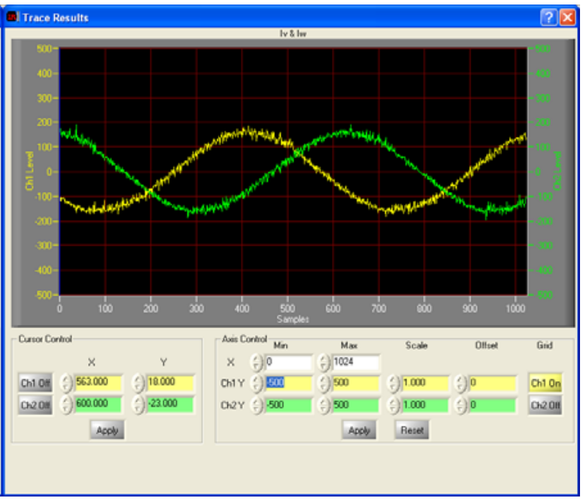
SHDelay = Td_On + dead time = 590 ns + 500 ns = 1.1 µsec

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

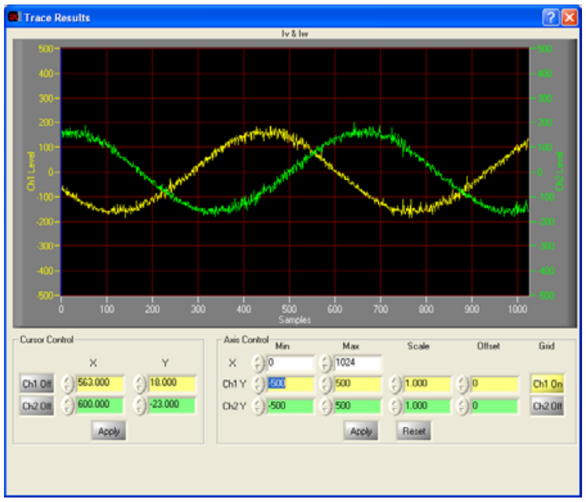
Figure 69(a) is a trace buffer plot from MCE Designer for this case. Figure 10(b) is a plot when SHDelay is 1.1 µsec and minimum pulse is 1.5 µsec. Some glitches exist due to slow ripple component.



(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 69—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 69(c) is a plot for this case and seems as good as 62(a). These plots in Figure 69 verify equations (1) to (6).

### 4.3.3 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 shut down 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 IRMCx300 series includes three part numbers which provide I/O to support active PFC control: IRMCx343, IRMCx311 and IRMCx312. These ICs include 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 Kits for these part numbers include an IR-supplied MCE program to perform the PFC, described in Section 5.1. Figure 70 below shows the simplified topology of the boost PFC control employed in the reference design kits, with the necessary sensing parameters labeled.

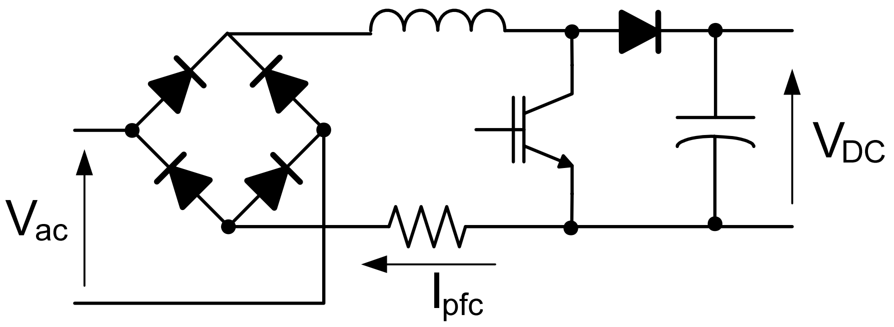


Figure 70—Basic PFC circuit

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. Next, Section 5.2

gives basic information and instructions for using the PFC as implemented in the Reference Kits. It also describes some hardware modifications, tuning and optimization techniques and advanced capabilities of the PFC control. Finally, in Section 5.3, this chapter concludes with guidelines for hardware design, optimization and testing of the PFC as related to the IRMCx300 series IC.

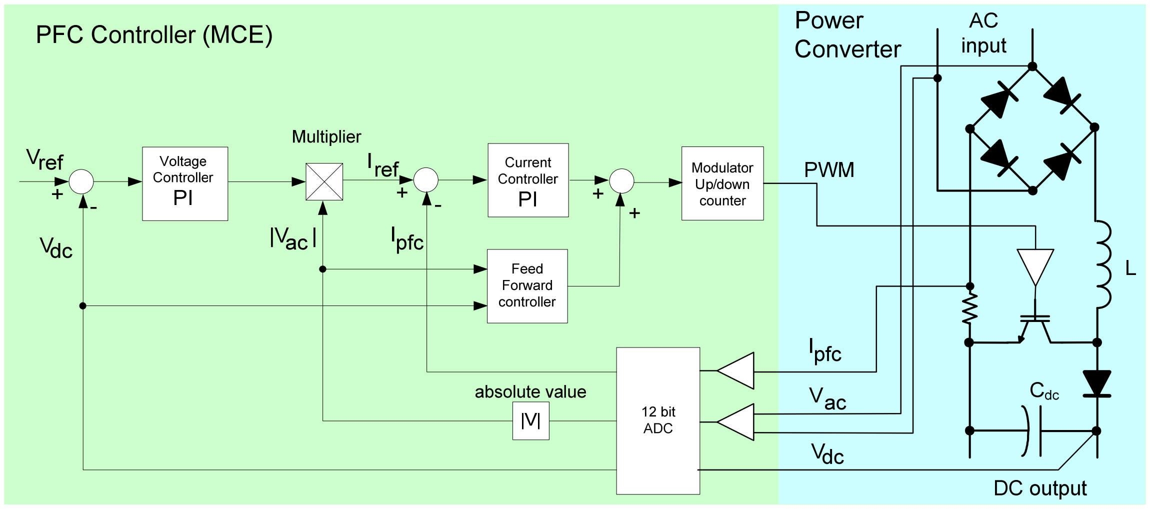


Figure 71—PFC Control System

Figure 71 above provides a high-level block diagram of the PFC control scheme, as implemented in the IRMS30xx Reference Design Kits. The digital control portion on the left, in the green shading, 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 71, 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). Each component of the controller will be described fully in Section 5.1.

## 5.1 MCE Program

### 5.1.1 Current Loop

A simplified block diagram of the current loop is shown in Figure 72, below.

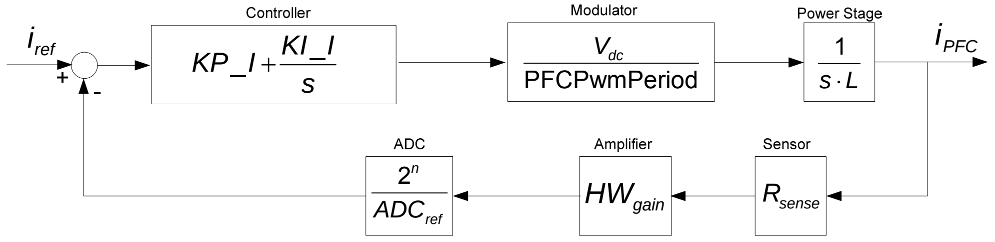


Figure 72—Simplified current loop

The controller gains, KP_I & KI_I, are automatically configured by the MCEWizard 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. In order to keep a constant loop gain, based on the diagram above, the current controller PI gains scale as:

$$KP\_I \propto \frac{L \cdot PFCPwmPeriod}{R_{sense} \cdot A_{Ipfc}}$$

$$KI\_I \propto \frac{L \cdot PFCPwmPeriod}{R_{sense} \cdot A_{Ipfc}} \cdot \frac{(1 + PFC\_sync\_divider)}{f_{PWM}}$$

where $A_{Ipfc}$ is the current sense feedback gain which includes the resistor divider and A/D converter gain. The PFCPwmPeriod appears because the modulator scaling is defined as 100% = PFCPwmPeriod. The value $f_{PWM}$ / (1 + PFC_sync_divider) gives the control loop computation rate. See Section 5.2.5.2 for details on the sync divider.

The complete current loop implementation in the MCE program is shown on the next page in Figure 73. Here, the output of the voltage loop, VAout, is multiplied by the input AC voltage, V_IN, to produce a current reference shaped by the line voltage. The difference between the reference and the current feedback feeds the input of a PI block. The output of the PI block is added to the feedforward component (Section 5.1.2), then limited and scaled to produce the modulation command for the PFC PWM output. The limit block also provides anti-windup functionality.

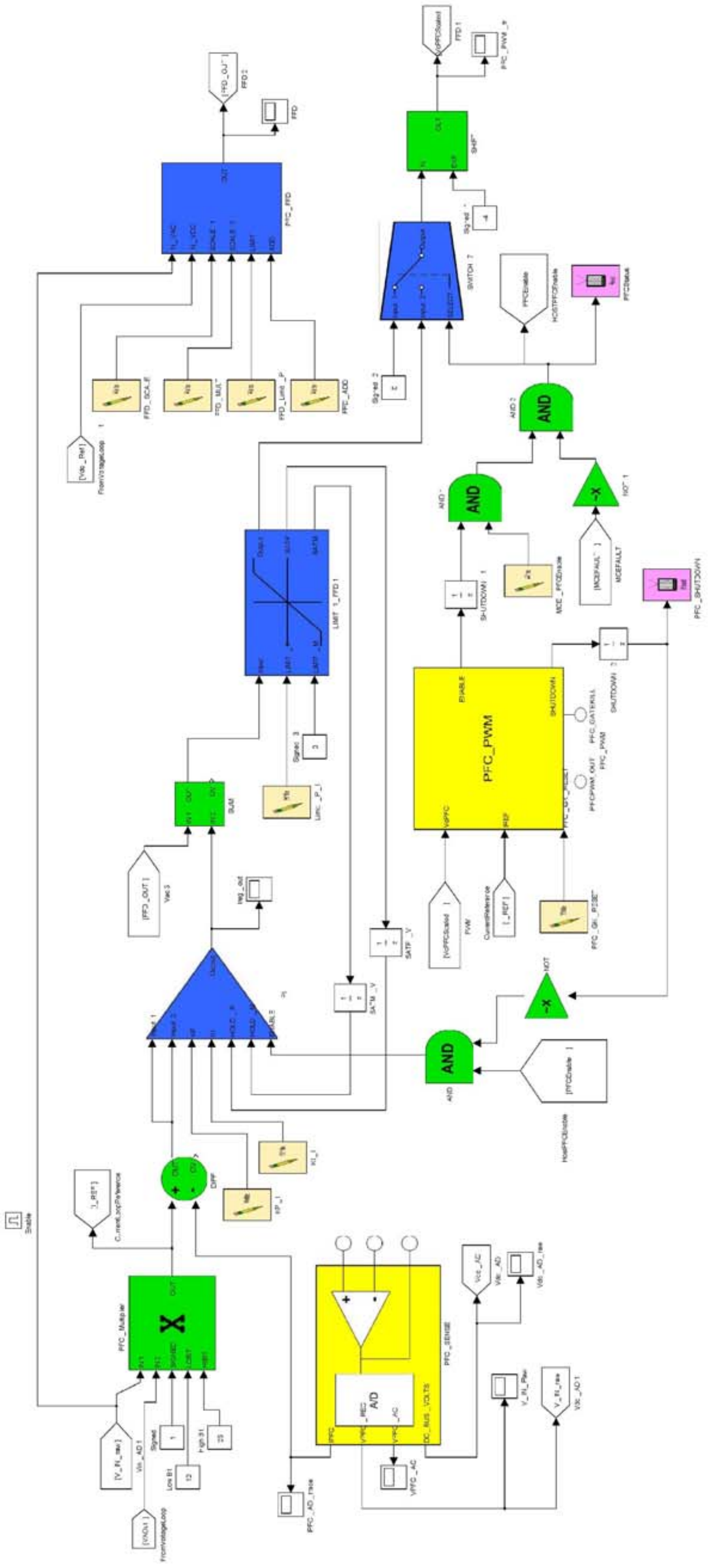


Figure 73— MCE implementation of the PFC Current Control Loop

### 5.1.2 Voltage Loop

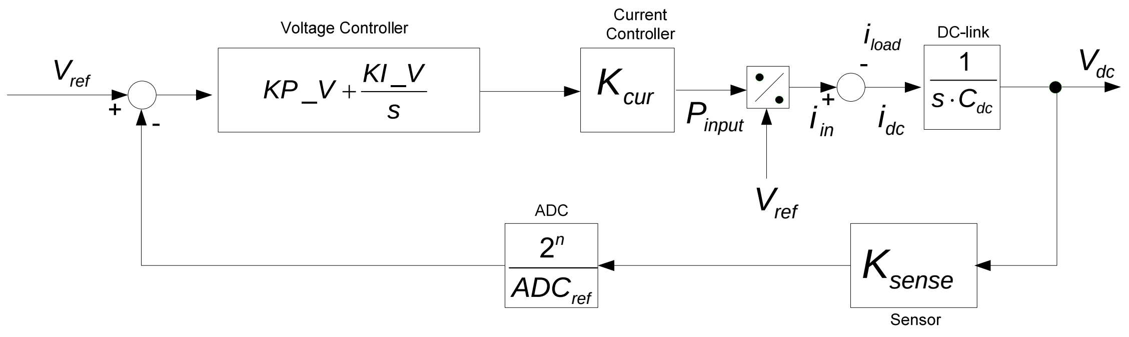


Figure 74—Simplified voltage loop.

Figure 74 gives a simplified block diagram of the voltage loop in the s-domain. The current controller represented as a gain component; this can be done because the current regulator bandwidth is much larger than the bandwidth of the voltage loop. The MCEWizard configures the voltage controller gains, KP_V & KI_V, in order to achieve a bandwidth of 3.7 Hz. In order to keep a constant loop gain, from the diagram above, the voltage controller PI gains scale as:

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

$$KI\_V \propto \frac{C_{dc}}{A_{Vdc} \cdot A_{Vac}} \frac{(1 + PFC\_sync\_divider)}{f_{PWM}}$$

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 mutiplied by the rectified ac voltage. The value $f_{PWM}$ / (1 + PFC_sync_divider) gives the control loop computation rate. See Section 5.2.5.2 for details on the sync divider.

The MCE program implementation of the PFC voltage loop is shown in Figure 75. Starting from the left side, a ramp block ensures that the PFC voltage is not changed discontinuously. Next, the reference voltage is compared to the filtered voltage feedback and the difference is input to the PI block. A LIMIT block keeps the PI output within a specified range and also provides anti-windup functionality.

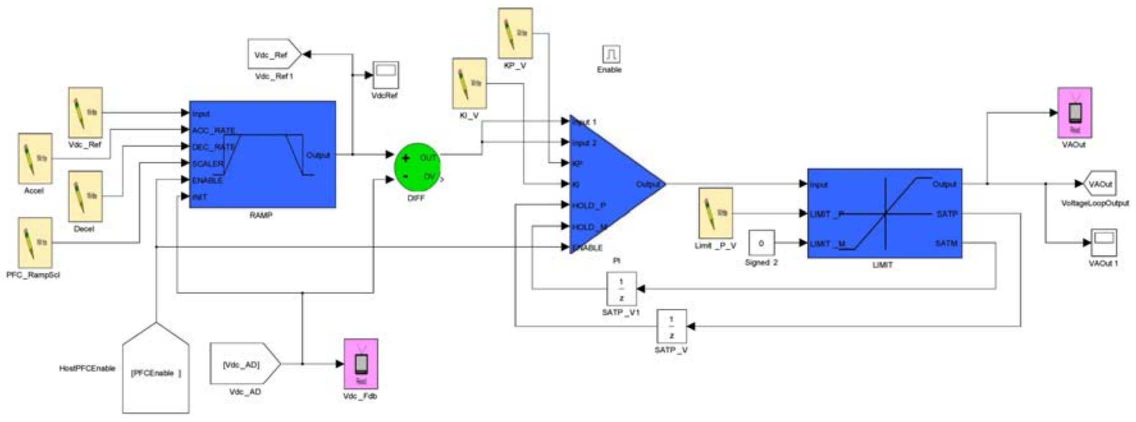


Figure 75—MCE implementation of the PFC Voltage Control Loop

### 5.1.3 Feedforward

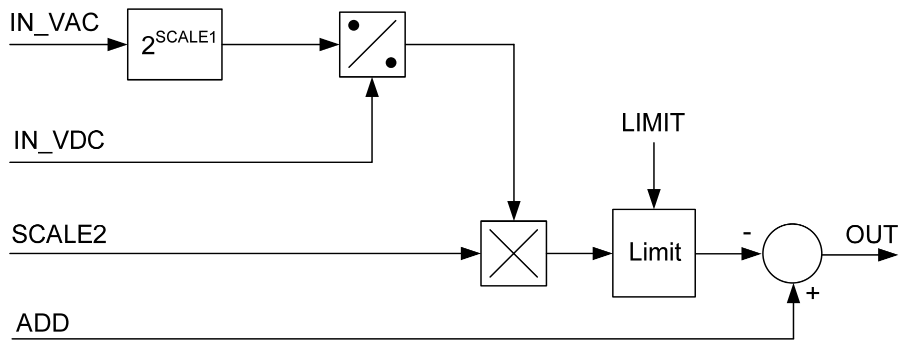


Figure 76—FFD Block implementation

Figure 76 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 77 shows an example of the FFD output and the current regulator output. The FFD provides a large duty cycle command during the AC voltage zero crossing which reduces the demand on the current controller.

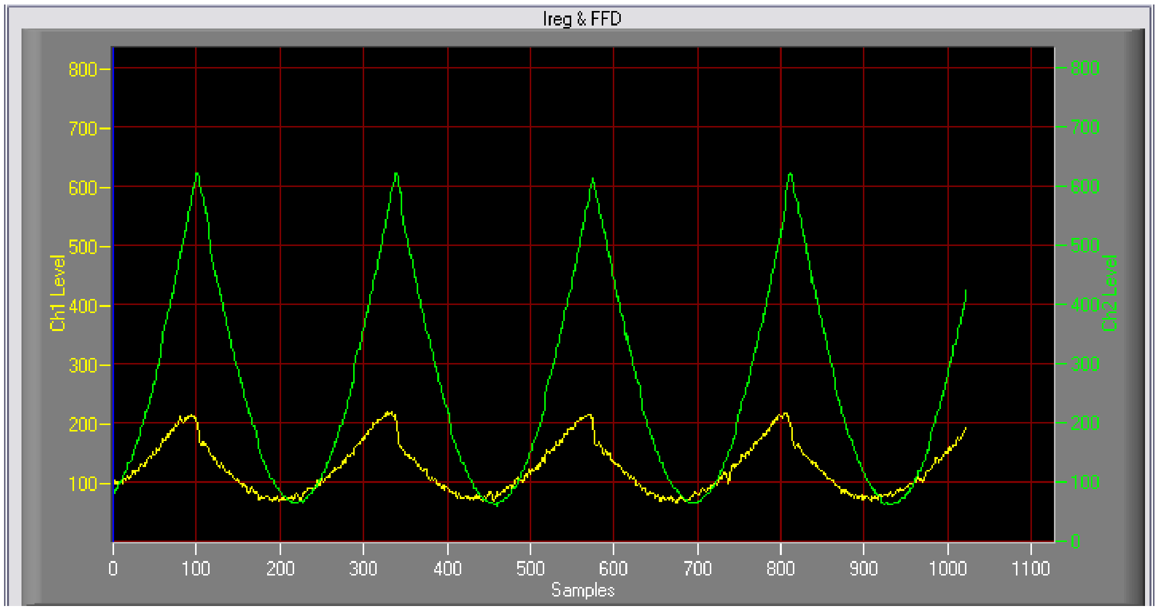


Figure 77—Current Regulator Output (yellow) and Feedforward Output (green); 833 corresponds to 100% modulation

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 the MCEWizard.

### 5.1.4 Enable and Shutdown

One thing to note about the PFC control program is that PFCEnable alone does not enable the PWM output. The register MCE_PFCEnable must be set to 1, otherwise the PFC modulation command will be zero. The operation of MCE_PFCEnable can be traced in the MCE program diagram of the current regulator shown in Figure 73. This register provides a means of disabling the PFC controller without recalculating the current offset.

Another feature of the PFC implementation is PFC Shutdown. The SHUTDOWN signal of the PFC_PWM block becomes '1' when the DC Bus voltage is lower than the AC input voltage. When there is a PFC Shutdown event, then the PFC PWM output is blanked and the PI regulator of the current controller is disabled and reset. For more information on PFC Shutdown, see Section 5.2.5.1.

### 5.1.5 Input and Output Registers

This section lists all the input and output registers of the MCE program for PFC. The scaling for many of the registers is given in Section 5.2.3. Additionally, the hardware registers which control the PFC can be found in the Reference Manual.

**Voltage Loop**
Input:
Vdc_Ref—Sets the boost PFC voltage. This voltage should be at least 5% higher than the peak input AC voltage

Accel, Decel, PFC_RampScl—Together set the voltage ramp rates. The default value for the IRMCS30xx Reference Design Kits is ~225V/s.

KP_V, KI_V—Proportional and integral gains, respectively, of the PI block of the voltage regulator.

Limit_P_V—The upper limit is set such that the maximum possible current command (in the current regulator) is within the A/D sensing range of the control IC. The upper limit is calculated using the current & voltage feedback scaling and the maximum AC input voltage. This register also sets the level at which the integrator is halted for anti-windup. The lower limit of the voltage regulator is always set to zero.

Output:
Vdc_Fdb—DC bus voltage feedback

VAOut—Output of the voltage loop. When this value is multiplied by the AC input voltage, the result is the current command.

**Current Loop**
Input:
KP_I, KI_I—Proportional and integral gains, respectively, of the PI block of the current regulator

Limit_P_I—The upper limit corresponds to 100% modulation. This register also sets the level at which the integrator is halted for anti-windup. The lower limit of the current regulator is always set to zero.

**Feedforward**
Input:
FFD_SCALE, FFD_MULT, FFD_Limit_P, FFD_ADD—Set the limits and parameters of the PFC_FFD block. The FFD block is configured to provide a maximum modulation command of 75% at zero crossings of the AC input voltage. With reference to Figure 76, FFD_SCALE

corresponds to SCALE1, FFD_MULT corresponds to SCALE2, FFD_Limit_P corresponds to LIMIT and FFD_ADD corresponds to ADD.

**Other**
Output:
PFC_SHUTDOWN—Indicates when PFC blanking occurs.

PFCStatus—Indicates the PFC run status.

## 5.2 Using PFC on the IR Reference Board

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

One of the features of the digital PFC control is automatic offset correction of the PFC shunt current. The offset is continuously measured when the PFCEnable register is set to 0. When the PFCEnable register is set to 1, the offset is latched in and automatically applied to the PFC current measurements. **It is important to set the PFCEnable register to 1 while there is no load on the system otherwise it may affect the offset measurement.** For example, set PFCEnable to 1 before starting any motors, because the motor currents will flow through the PFC sense resistor. The "Start PFC" function in MCEDesigner sets PFCEnable to 1.

### 5.2.1 Using the Wizard to create the configuration parameters

The MCEWizard 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. This group of questions, System DC Bus, 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 the MCEWizard can generate the configuration parameters for the new hardware. Be sure to select "I have modified the circuit board" on the Welcome page of MCEWizard to enable the hardware dependent questions. See Section 5.2.6 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. Remember to start the PFC before starting any motors. At light loads, the power factor will be low, but will rise to higher than 0.9 by the time the output power reaches 150W.

### 5.2.2 Overcurrent Protection Circuit

The control IC includes an input pin to shut off the PFC switch in the 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 78 below shows the circuit from the IRMCS3012. 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 immediately change to the off state by hardware in the IRMCx300 IC. The soonest that the PFC can be re-enabled is the next PWM cycle by setting the PFC_GK_RESET register to 1.

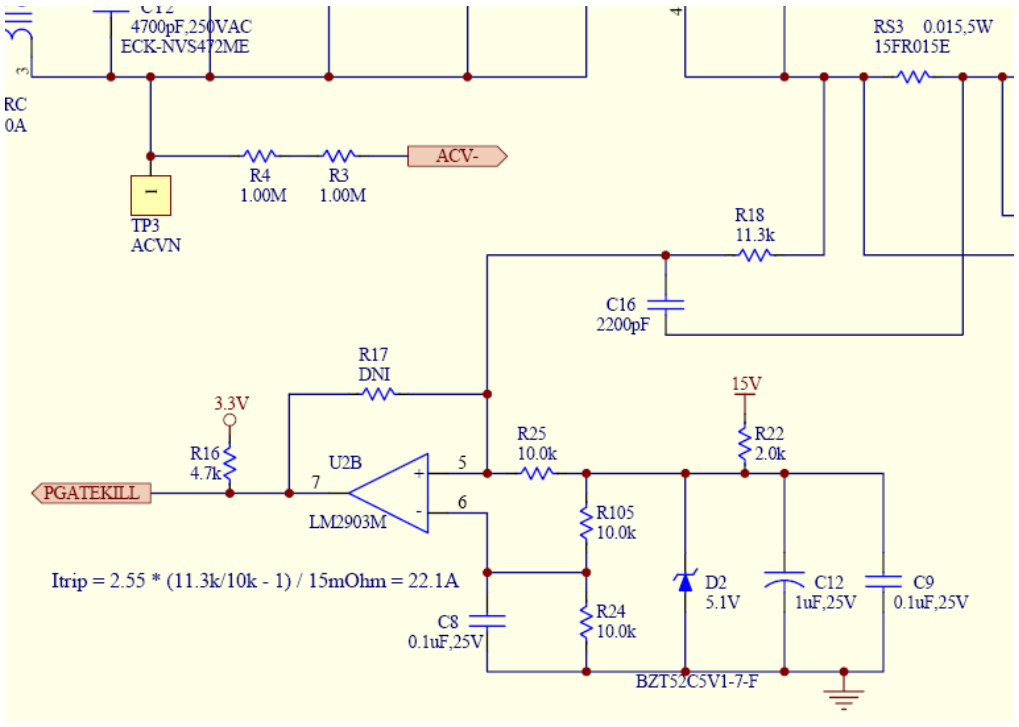


Figure 78—PFC Gatekill circuit

The gatekill fault is useful for protecting the PFC hardware components, but in some situations it can become a “nuisance trip.” The developer may implement an automatic gatekill reset either in the MCE program or in the 8051 code. However, be sure to provide some means of distinguishing a nuisance trip from a real overcurrent event which threatens the PFC hardware. To reset the PFC gatekill, write ‘1’ briefly to the register PFC_GK_RESET. However, if this register is set to ‘1’ while the PFC is running, then any PFC gatekill will be automatically cleared at the *next* PWM cycle, and the PFC will continue operating.

### 5.2.3 PFC Variable Scaling

**Voltage Loop**:
The DCBus voltages, **Vdc_Ref** and **Vdc_AD**, have the same scaling as defined in Section 2.2.3.4:
*DCBus (V) = [Vdc_AD] / (A/D * r)*
where
A/D is the analog-to-digital converter scaling (3412/Volt)
r is the voltage divider ratio used in the voltage sense circuit

AC Voltage Feedback—The figure below shows the AC line voltage feedback path to the terminal so of the op-amp. There are two parameters in the MCE program which can be traced—**VPFC_AC**, which is the AC line voltage and **V_IN_Raw**, which is the rectified AC line voltage. The scaling is calculated similarly to the DC Bus feedback:
*AC Voltage = [V_IN_Raw] / (A/D * k)*
where
A/D is the analog-to-digital converter scaling (3412/Volt)
k is the op-amp gain of the voltage sense circuit

In the sensing circuit show in Figure 79 below, k = 2.43k / (4.87k + 1.00M + 1.00M) = 1.21 x $10^{-3}$.

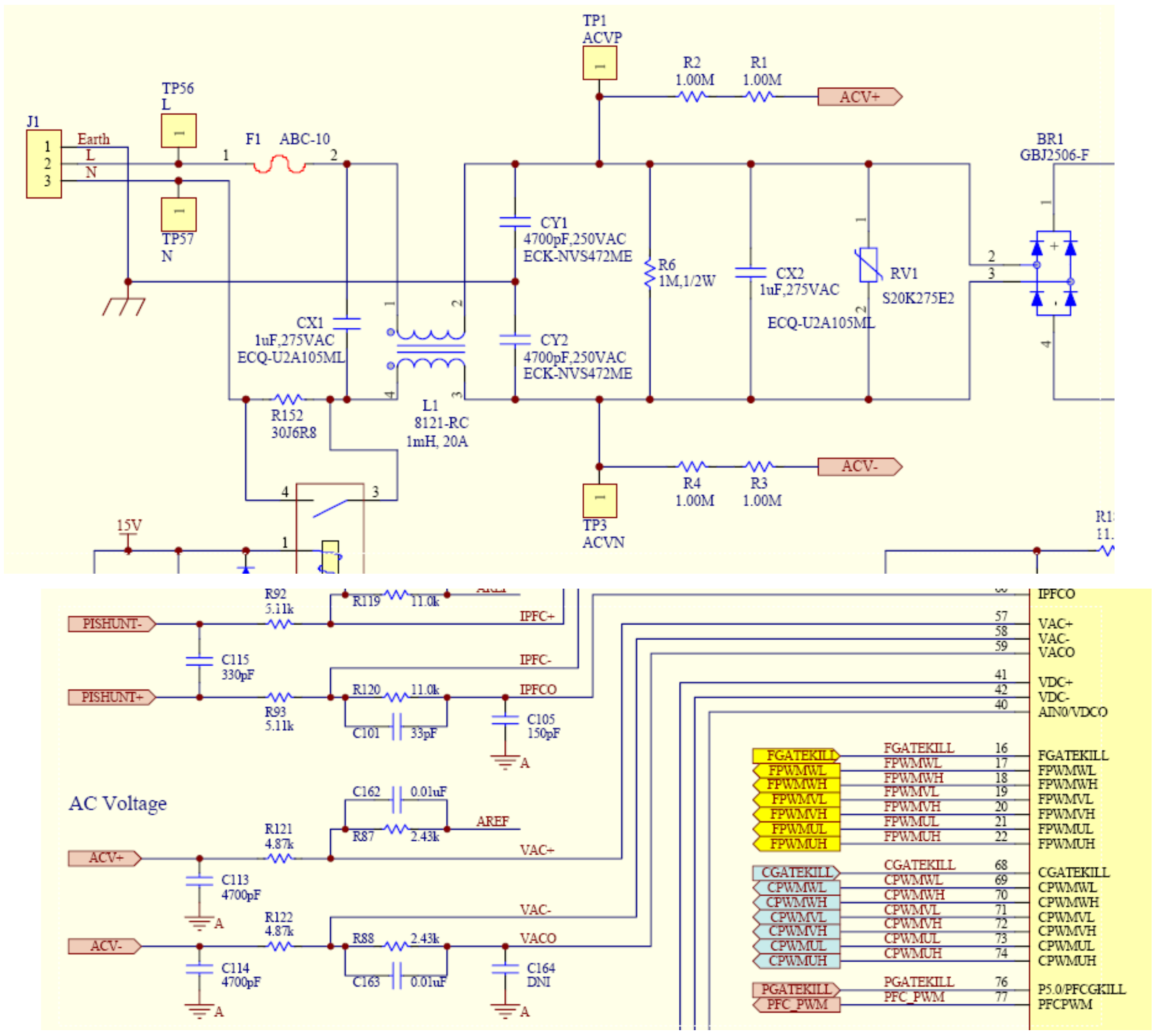


Figure 79—AC Voltage Feedback Circuit

**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)
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)
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)

k is the op-amp gain of the PFC current sense circuit
Rshunt is the PFC current feedback resistor in Ohms

**VcPFC** (PWM modulation): The modulation scaling is different than in the motors; it is defined in terms of the PFCPWMPeriod register:
*Modulation (%) = 100 * VcPFC / PFCPWMPeriod*

### 5.2.4 Optimizing Starting and Running

The operation of the PFC switching can be understood by the waveforms of Figure 80. A PFCPWM_SyncPulse occurs at regular intervals according to the PFCPWMPeriod register, at the minimum of the up-down counter. Unless another control loop is still running, a PFCControl_SyncPulse initiates the A/D conversion of the PFC Current and Voltage. After the A/D conversion, the PFC control loop runs, updating the Duty Cycle Command. The duty cycle gets latched in at the next PFCPWM_SyncPulse. The latched VcPFC is compared to the PFC PWM carrier up-down counter. The PFC switch is turned on when VcPFC is less than the PFC PWM carrier.

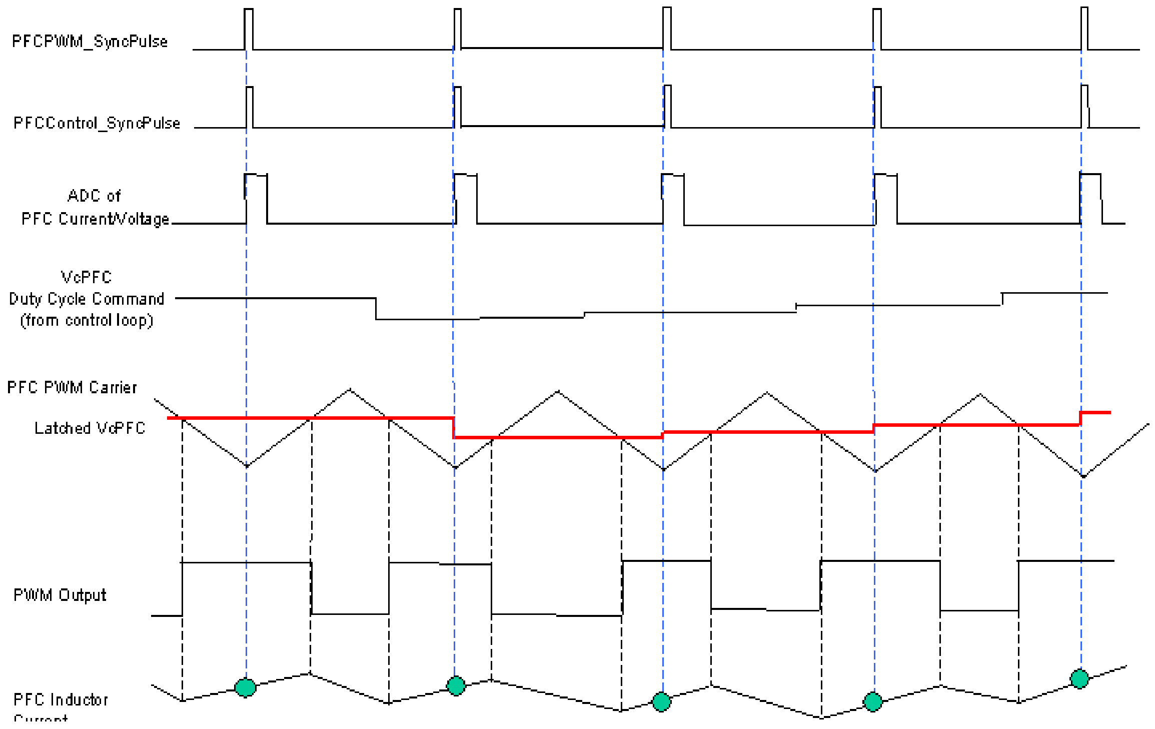


Figure 80—PFC PWM cycle timing

#### 5.2.4.1 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. The trade-offs from increasing the PWM frequency are 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 the MCEWizard 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.

To increase the switching frequency without increasing the MCE usage, set up a non-zero sync divider value as described in Section 5.2.5.2.

#### 5.2.4.2 PFC Start-up/Voltage Ramp Rate

The PFC is automatically configured with a voltage ramp rate of ~225V/s. See the equation in Section 5.2.3 to calculate the ramp rate. Note that changing the switching frequency, sync divider ratio 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.

#### 5.2.4.3 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 at 3.7 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 make the voltage regulator unstable. It is up to the designer to verify the voltage loop performance meets the application requirements.

#### 5.2.4.4 Current Loop Tuning

In general, the current loop should have as high a bandwidth as possible. However, there are several factors which will 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 feedback. The MCEWizard configures the current loop to have a bandwidth of about 1400 Hz.

### 5.2.5 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.

#### 5.2.5.1 Blanking

PFC blanking is a function which disables the PFC PWM switching in situations where the DC Bus voltage is lower than the instantaneous input voltage. The full-mode boost PFC operation requires that the DC bus voltage be higher than the peak of the AC input voltage. However, during input voltage transients or a sudden large increase in load, the peak of the AC input voltage can be higher than the DC bus voltage. If the PWM switching continues, the boost inductor may go to saturation because its volt-seconds can not be balanced. Consequently, large currents can be generated, causing a nuisance over-current fault. The PWM_Blanking feature provides protection by generating an output ShutDown signal to blank the PFC PWM output.

The Blanking function compares the instantaneous DC Bus and AC input voltages to determine if the PWM switching should be blanked. The registers PFC_OffsetDC, PFC_OffsetVin and Blanking_Gap provide offset and adjustment for the comparison, and are used to calculate signals Vdc_Compare and Vin_Compare as described below:

*Vdc_Compare* = Vdc_ADC + PFC_OffsetDC

*Vin_Compare* = 2 * Vin_ADC + PFC_OffsetVin
**Note**: The computation above implies that the DC Bus voltage scaling (in counts/volt) must be twice the AC input voltage scaling. This is true in all Reference Design Kits.

*Blanking_Gap* provides hysteresis for the Blanking function. For a detailed description of how this register is used, see Section 4.3.8.2 of the Reference Manual.

Basically, if Vdc_Compare > Vin_Compare then Shutdown = 1 and the PWM output is blanked. True operation is more complex than just this comparison; it is describe fully in the Reference Manual.

The Shutdown signal will stay on for minimum time of CountOneSet / PFC_PWMFrequency, which corresponds to CountOneSet number of PFC PWM cycles. Once the CountOneSet duration has expired, then, based on the circuit operational status (Vin, Vdc, IPFC and Iref), ShutDown will transit from 1 to 0, allowing the PWM Output to be sent out. This provides fast and smooth transitions between the PWM enable and disable modes. CountTwoSet is a register which provides additional filtering of the current feedback comparison. For detailed information about ShutDown turn-off criteria and the use of CountTwoSet, see Section 4.3.8.2 of the Reference Manual. Note also that the integral component of the PFC current controller is reset on Shutdown.

The Wizard provides typical values for the PFC Blanking registers, but the optimal configuration should be found by testing anticipated AC input voltage disturbances and possible step changes in the load.

#### 5.2.5.2 SyncDivider

The SyncDivider feature of the PFC can allow the designer to use a higher PFC switching frequency without increasing the computational load on the MCE. When using the SyncDivider, the PFC switches at the frequency defined by the PFCPWMPeriod register. However, the PFC current and AC input voltage are only sampled at some fraction of the PFC frequency, as defined by the PFCSyncRatio, the lowest four bits of the register PFC_sync_divider. Also, the PFC component of the MCE program will only run during the PWM cycles when the PFC current is sampled, reducing the computational load on the MCE. The A/D sampling and control loop frequency is equal to PFC PWM Frequency / (PFCSyncRatio + 1)

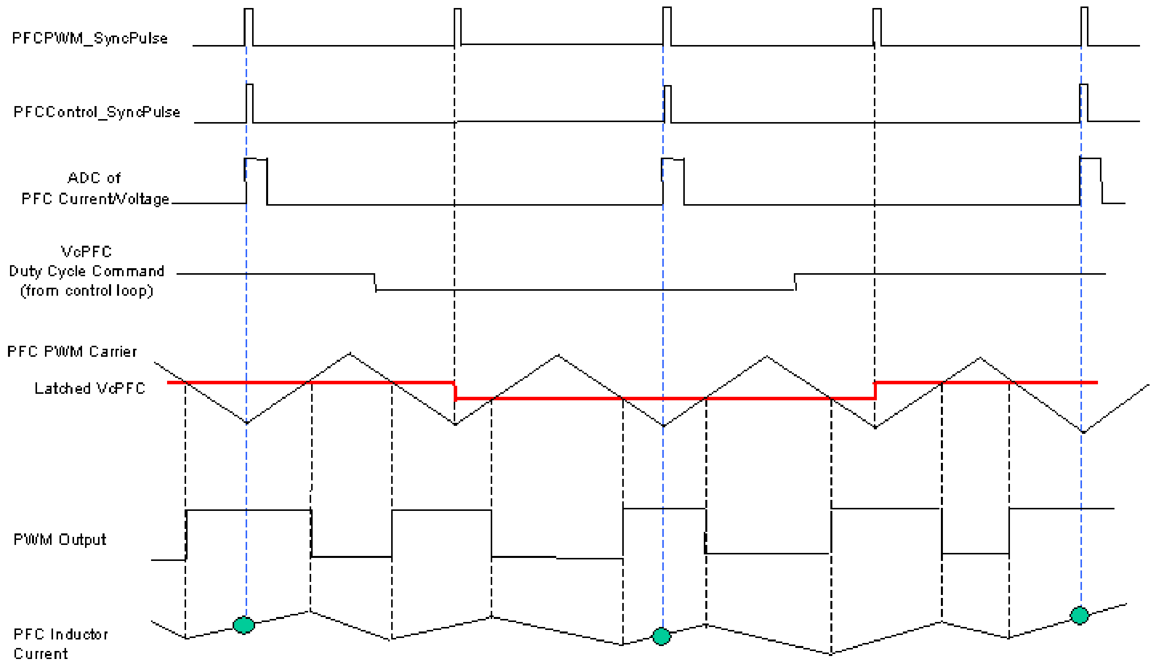


Figure 81—PFC PWM timing when PFCSyncRatio = 1

Figure 81 shows the PFC operation with a PFCSyncRatio = 1. Compare this diagram with the normal situation (PFCSyncRatio = 0) shown previously in Figure 80. In the figure above, the current and voltage are sampled only every other PWM cycle, and the Duty Cycle Command is also updated at this reduced frequency.

There is one important restriction on the selection of the SyncDivider usage: The PFC A/D Frequency (= PFC PWM Frequency / (PFCSyncRatio + 1)) must be an integer multiple of the master PWM frequency (usually Motor 1). If this constraint is not met, then the actual frequency of the A/D sampling and control loop will vary between [PFC PWM Frequency / (PFCSyncRatio + 1)] and the PFC PWM Frequency over the course of one master PWM cycle.

#### 5.2.5.3 PFC Phasing

PFC Phasing is an extension to the SyncDivider feature. The PFC phasing offsets the PFC Current & Voltage sampling from the Master PWM Sync pulse (usually Motor 1, but user configurable). The value of PFCPhasing (PFC_sync_divider[4:7]) specifies the number of PWM cycles the PFC A/D sampling is delayed following the master motor control sync pulse. The PFC control loop runs right after the A/D sampling. Figure 82 depicts the PFC phasing in an example with the following settings:

• PwmMasterSel is set to 0, selecting Motor 1 as the master.

• Motor 1 PWM frequency is 5 KHz and PFC PWM frequency is 40 KHz, so that a Motor 1 sync pulse occurs on every eighth PFC PWM cycle.

• PFCSyncRatio is set to 3, configuring PFC A/D sampling on every fourth PFC PWM cycle (10 KHz) and PFCPhasing is set to 2.

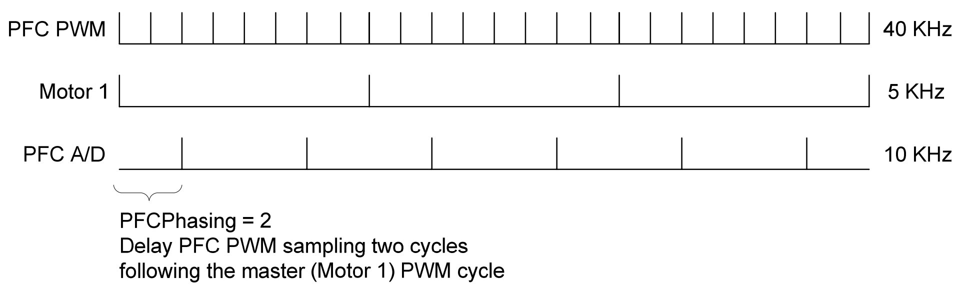

PFC PWM
40 KHz
Motor 1
5 KHz
PFC A/D
10 KHz
PFCPhasing = 2
Delay PFC PWM sampling two cycles
following the master (Motor 1) PWM cycle

Figure 82—PFC Phasing Example

Again, the PFC A/D frequency should be a multiple of the Motor 1 frequency. If it is not, then the actual A/D frequency will not be correct and the phase relative to Motor 1 will not be predictable. One final constraint when using the phasing is that the value of PFCPhasing < (PFCSyncRatio + 1), or else the PFC will not run at all.

### 5.2.6 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 MCEWizard.

#### 5.2.6.1 Shunt Resistor

Changing the PFC shunt resistor can change the range of currents which the hardware can sense or change the GK current trip level. For example, the IRMCS3012 Reference Design Kit has a 15mΩ shunt resistor for sensing the PFC current. 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.

Note that another way to change the GK current trip level is to modify the comparator voltage. In Figure 78, change R105 and R24 to set the voltage at the inverting input to the comparator.

### 5.2.6.2 Feedback Gain

Modifying the PFC current feedback gain is another way to change the range of current which the controller can sense. Figure 83 shows an example of the op-amp circuit involved in the PFC current sensing.

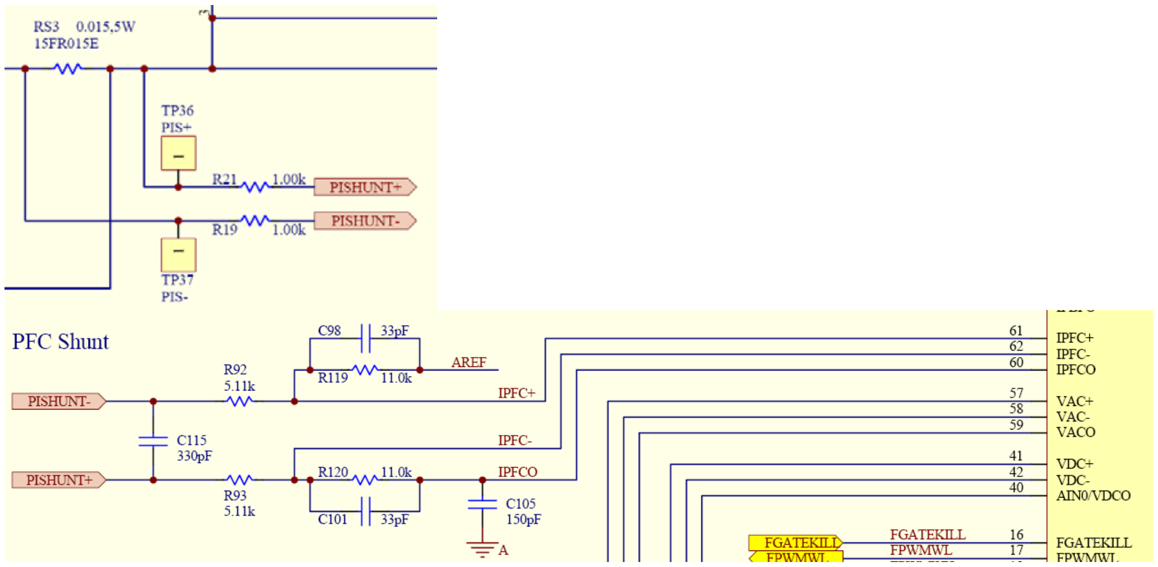


Figure 83—PFC Current Feedback Circuit

In the same way as for motor current feedback, the gain of this circuit is 11.0 / (5.11 + 1.00) = 1.8. To modify the op-amp gain, the designer should change resistors in pairs (R92 & R93; R119 & R120) to preserve the correct circuit biasing at AREF (0.6V reference).

### 5.2.6.3 DC Bus Sensing

In the default configuration, the DC bus sensing is accomplished through a voltage divider and then feeding to the A/D converter through pin AIN0/VDCO. This pin is the output of an operational amplifier which is disabled. To use the op-amp for sensing instead, make the following modifications:

1) Install a 4.87kΩ resistor at R44 and a 0Ω resistor (jumper) at R85.
2) Remove R47 and R125.
3) In the 8051 code set sfr HWCFG to 0xdF. (See Software Developer's Guide.)

With these modifications, the DC bus voltage scaling will not change.

### 5.2.6.4 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
where $f_{PFC}$ 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 IRMCS3012 Reference Kit). Be sure to generate new configuration parameters using MCEWizard. 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.

## 5.3 PFC Hardware Design

### 5.3.1 Schematic Elements

#### 5.3.1.1 DC Bus and AC Input Voltages

The DC Bus capacitor sizing can be determined by the hold-up requirements and ripple tolerance of the application. The MCEWizard 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

An additional consideration is that in order for PFC Blanking to operate correctly, the DC Bus scaling (in counts/V) must be twice the AC Input Voltage scaling. This is already true in the Reference Design Kits.

#### 5.3.1.2 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. Note that this compensation is automatically achieved for the PFC current feedback. When PFCEnable is set to 0 the IC measures the PFC current offset and then latches this value when PFCEnable is set to 1. For this reason it is important to set PFCEnable to 1 only when there is a very small or no load on the DC Bus.

The IRMCS3012 Reference Design Kit hardware has a provision to provide voltage references of 1.1 V at AIN3 and 0.1V at AIN4 by dividing down the 1.8V supply. The designer may uses these analog inputs to perform offset correction.

#### 5.3.1.3 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. The Figure 84 below shows the schematic of the EMI Filter in the IRMCS3012. 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.

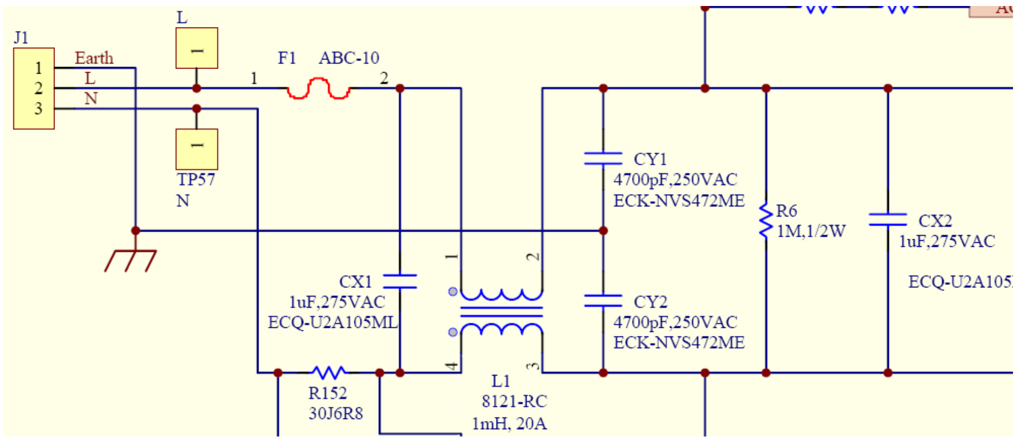


Figure 84—EMI Filter

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 Compressor Inverter schematic of the IRMCS3012 Reference Design Kit.

### 5.3.2 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 overcurrent protection.

## 5.4 Advanced Topics

**Automatic Gatekill Handling**
In some applications, noisy, unstable or unreliable AC input voltages can cause nuisance trips, even with the aid of the PFC Blanking. However, these trips can be handled automatically within the MCE program using the PFC_GK_RESET register of the PFC_PWM block. Use the FAUTLS block to provide the signal that a PFC Gatekill has occurred, then construct some logic to reset the fault depending on the conditions.

**Moving voltage loop to execute at lower frequency**
Though the voltage loop executes at the same frequency as the current loop, the voltage changes much more slowly than the current and requires a much lower bandwidth. This would allow the voltage loop to be moved to a component with a lower PWM frequency, such as motor 1 or motor 2. The benefit of moving these blocks would be that the current loop could execute at a higher frequency, since more MCE computing resources would be available.

The gains of the voltage regulator should be modified if it is moved to a lower switching frequency.

**Using PFC PWM output for other functions**
Note that the PWM output pin could be used for control of any switching desired by the user. The MCE program for the PFC could be replaced by a custom designed control algorithm as required.

**Rectified Input Voltage Sensing**
In the reference design, the AC input voltage is sensed before the diode bridge. If the rectified line is sensed, then use the RAW_VIN_SENSE signal from the PFC_SENSE block instead of VPFC_REC. (For details about RAW_VIN_SENSE, see the Reference Manual.)

# 6 Induction Motor (IM) Control

## 6.1 Introduction

The IRMCx300 Series motor control ICs are designed to perform sensorless field oriented control (FOC) of a permanent magnet motor. However, the MCE program can be extended to allow sensorless FOC of an induction motor without any hardware modification. The general architecture of the IRMCx300 controller is preserved, containing a speed loop and current loop, with modified MCE code to accommodate the IM control function. Section 6.2 describes this MCE program in detail. Also, measurement and configuration of the motor parameters are covered in Section 6.3.

The resources for IM control listed below are installed at "My Documents\iMotion\Induction Motor."

- IRMCS3041_Release_IMCtrl_2_0.mdl (Simulink model file)
  - IRMCS3041_Release_IMCtrl_2_0.pdf (PDF of model file)
- IRMCS3041_Release_IMCtrl_2_0.bin (binary file for program download)
  - IRMCS3041_Release_IMCtrl_2_0.map (register map)
- IRMCS3041_Release_IMCtrl_2_0.irc (MCEDesigner file for drive interface)

These files are designed for the IRMCS3041 Reference Design Kit only, though IM control could be implemented in any of the controller versions. Contact an IR FAE for support for other Reference Design Kits. Additionally, MCEWizard supports configuration of IM control.

The IRMCS3041 Reference Design Kit is initially configured to run the supplied PM motor. Before running an induction motor, the MCE program needs to be replaced with the Induction motor control version (the binary [.bin] file mentioned above). Section 3.4.4 describes the process of downloading a new program in detail.

## 6.2 IM Control Program

The induction motor control components are implemented in the MCE Misc loop subsystem as shown in Figure 85. The interface from the Misc loop to the existing PM controller is implemented in the Speed Loop subsystem; the interface specifics are shown in Figure 86.

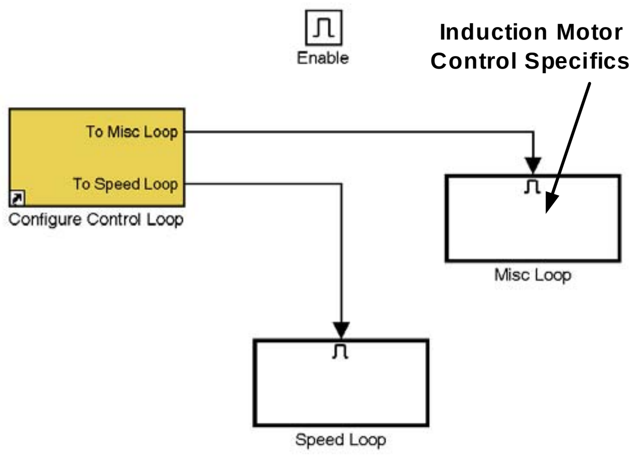


Figure 85—IM Control Implementation Location in MCE Program

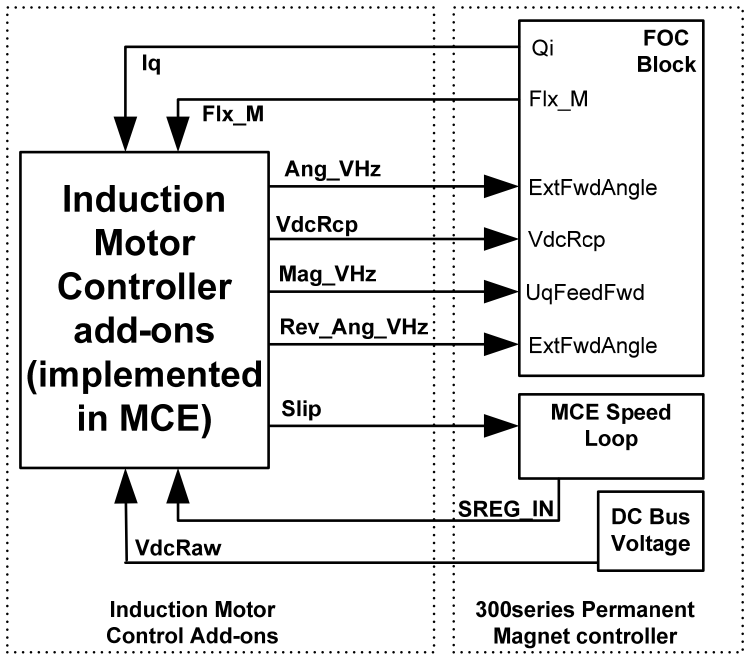


Figure 86—Interface between IM Control Add-on and 300 Series PM Controller

The IM control implementation fully utilizes the existing PM motor controller with a few add-ons to complete the entire IM drive control. By setting the appropriate drive parameters; two IM control modes—Field-Oriented Control (FOC) and Volts per Hertz (VHz)—can be obtained.

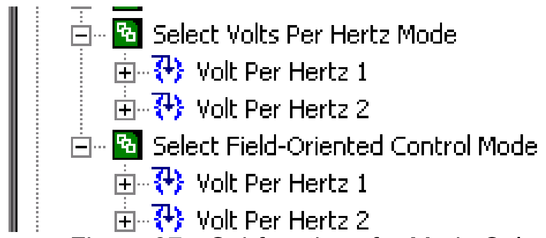
There are several major and minor components inside the IM Controller add-ons of Figure 86, each applicable to one or both of the two IM control modes. These aspects of IM control will be described in the sections below:

- Volts per Hertz Mode
  - Voltage magnitude and Angle calculation
  - Voltage Boost
  - Pre-Flux for VHz

- Field-Oriented Control Mode
  - Slip speed compensation

- Magnetization current injection
- Continuous Flux Checking
- Pre-Flux for FOC

- Common to both modes
  - DC Bus Compensation
  - Non-Regenerative Braking



Select Volts Per Hertz Mode
- Volt Per Hertz 1
- Volt Per Hertz 2

Select Field-Oriented Control Mode
- Volt Per Hertz 1
- Volt Per Hertz 2

Figure 87—Subfunctions for Mode Selection

The user can select between the two IM control modes—Field-Oriented control (FOC) and Volts per Hertz (VHz)—by setting the appropriate parameters. Two sub functions (Volt Per Hertz 1 and Volt Per Hertz 2) are provided in IRMCS3041_Release_IMCtrl_2_0.irc for setting these parameters, as shown in Figure 87. Selection functions for FOC and VHz mode are built based on these two sub functions. The specific parameter settings to select between the VHz or FOC modes are described in the appropriate sections below. FOC is the default mode after execution of the "Configure Motor" function in MCEDesigner.

It should be noted that the Critical Overvoltage protection was removed. This applies a zero vector when the DC bus rises above a threshold value. It is not useful for induction motors because there is no back-EMF, so the zero vector has no effect.

### 6.2.1 Volts per Hertz (VHz) Mode

The "Volts per Hertz" mode supports a Volts-Hertz profile with voltage boosting. This mode is simply an open-loop system, which provides a configurable voltage to frequency relationship, with uncontrolled motor current. The VHz control block is shown schematically in Figure 88 below along with its transfer characteristic (Figure 90).

VHz mode is open loop in nature and does not require current feedback. Due to the bypassing of the speed and current loops, there is no need for speed or current regulator tuning. Therefore, drive setup is extremely simple. In VHz mode, the voltage command is generated from a prescribed voltage-frequency profile. Because of the open loop nature of the control, VHz mode can provide better voltage utilization in field-weakening (high speed) operation as compared to current controlled systems (for instance, FOC), which limit the DCBUS voltage utilization when applying current control.

The VHz mode provides additional features, such as automatic field-weakening, voltage boosting and reverse-frequency operation, which the existing diagnostic mode (VF Diagnostic) does not support. The VF Diagnostic function is a scaled down version of the VHz mode described here. Its original intent is for diagnostic purposes only (mainly for checking current feedback integrity).

#### 6.2.1.1 Angle and Voltage Calculation (with Voltage Boost)

The input SREG_IN of Figure 88 below is a speed command signal that is driven by the speed ramp block (in the MCE speed loop subsystem) of the existing PM control algorithm. The outputs are the voltage modulation index (Mag_VHz) and electrical angle (Ang_VHz). These signals feed the external interface handles UqFeedForward and ExtFwdAngle, respectively, of the existing

FOC library block as shown in Figure 89. Details on these handles are given in the IRMCx300 Reference Manual (Figure 47, SENSORLESS_FOC block diagram).

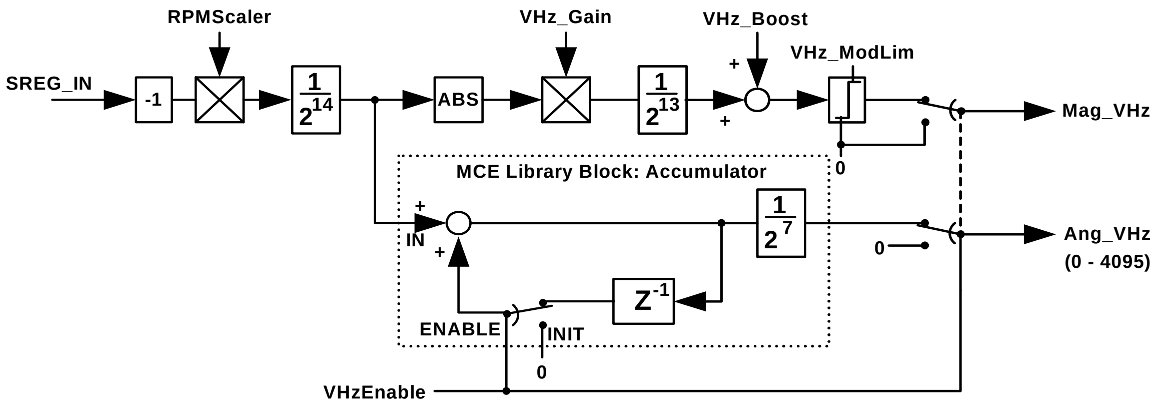


Figure 88—Implementation of Volts per Hertz with Voltage Boost

The actual Simulink implementation of the diagram in Figure 88 is shown in Figure 91. Note that the calculation path is also used to generate voltage modulation and angle for Non-regenerative Braking, as described in Section 6.2.3.2, below.

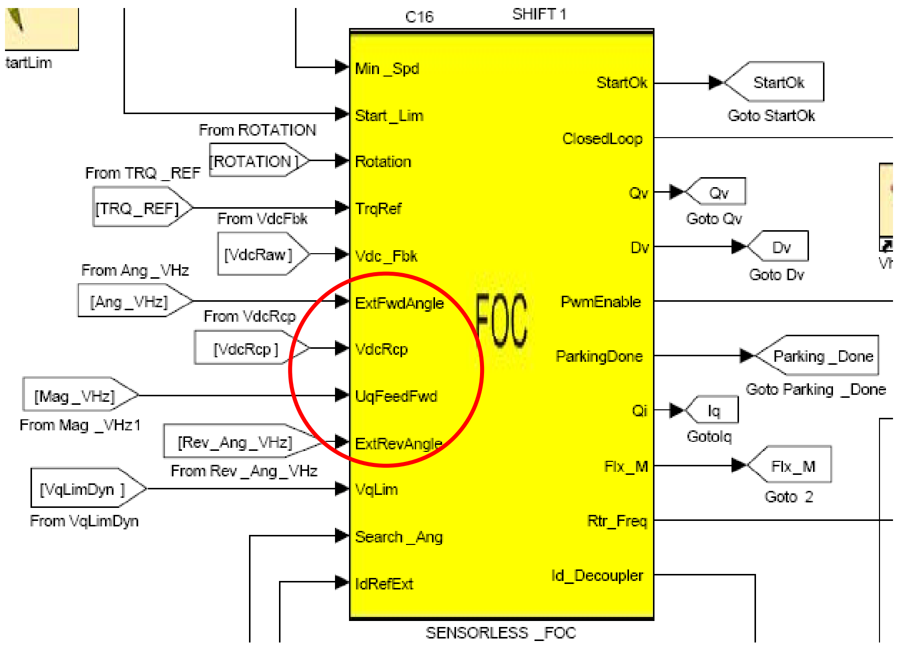


Figure 89—Interface to FOC Block

There are five parameters (VHz_Gain, VHz_Boost, VHz_ModLim, VhzEnable and RPMScaler) associated with VHz mode. These parameters are calculated by the MCEWizard based on motor data entries. The definition and scaling of these parameters are given below.

**VHzEnable**—This register provides the switch for enabling and disabling the outputs (Mag_VHz, Ang_VHz) of the Volts per Hertz function.
Range: 0 or 1 Scaling: 1 = enable, 0 = disable

***Angle and Voltage Generation Registers:***

**RPMScaler**—This parameter translates TargetSpeed to internal frequency counts for integration to generate the correct angle each PWM cycle such that the actual motor drive frequency matches the commanded speed (TargetSpeed: 16383 = Motor Max RPM from MCEWizard).
Range: 0 – 32767
Scaling: RPMScaler = $2^{19}$ * Motor Max RPM * Poles / (120 * PWMFreq)
**VHz_Gain**—This parameter adjusts the inverter output voltage-to-frequency ratio (Flux), which determines the operating flux level of the motor at the rated speed condition. The calculation in MCEWizard provides a VHz profile such that the rated motor voltage is obtained at the rated frequency.
Range: 0 – 65535 Scaling: Flux = VHz_Gain * DC Bus Voltage / PWMFreq / 31.6 [Volt-sec]
**VHz_Boost**—This parameter provides a constant motor voltage boost, which allows higher motor starting current. Applying an exceeding large VHz_Boost value will cause an over-current trip. Reducing the VHz_Boost value will reduce the motor starting torque. The MCEWizard sets this parameter (using stator resistance and rated current information) to target for the highest possible starting torque without exceeding the motor rated current.
Range: 0 – 32767
Scaling: Rms line-to-line voltage = VHz_Boost / 1430 * DC Bus Voltage / √(2) [Vrms]
**VHz_ModLim—**This is the maximum modulation output of the VHz function. Normally, it is set lower than 100% to allow a margin for DC bus ripple voltage compensation.
Range: 0 – 32767 Scaling: % modulation index = VHz_ModLim/1430 * 100 %

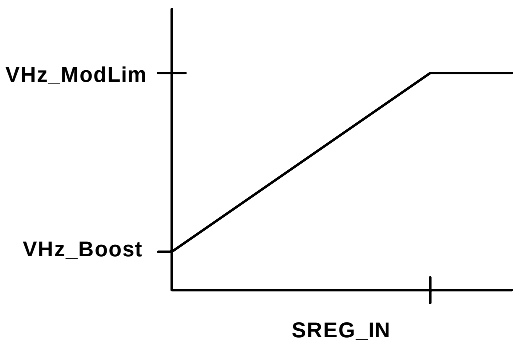
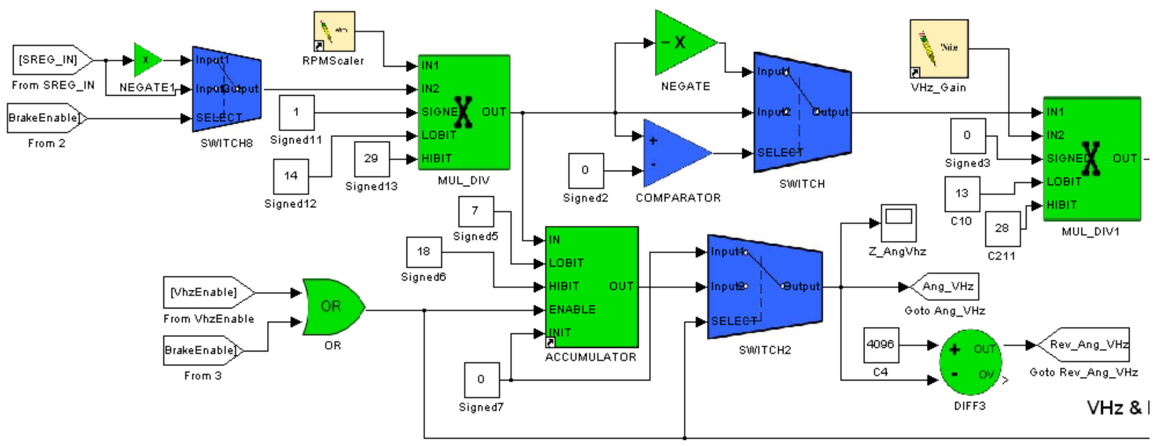


Figure 90—Transfer Characteristics of Volts per Hertz Function

The three parameters, VHz_Gain, VHz_Boost, and VHz_ModLim, define the transfer characteristic of the VHz mode, shown in Figure 90. VHz_Boost provides for higher starting current, resulting in a higher starting torque. VHz_Gain sets the slope of the transfer characteristic, and VHz_ModLim sets an upper limit to the voltage modulation.



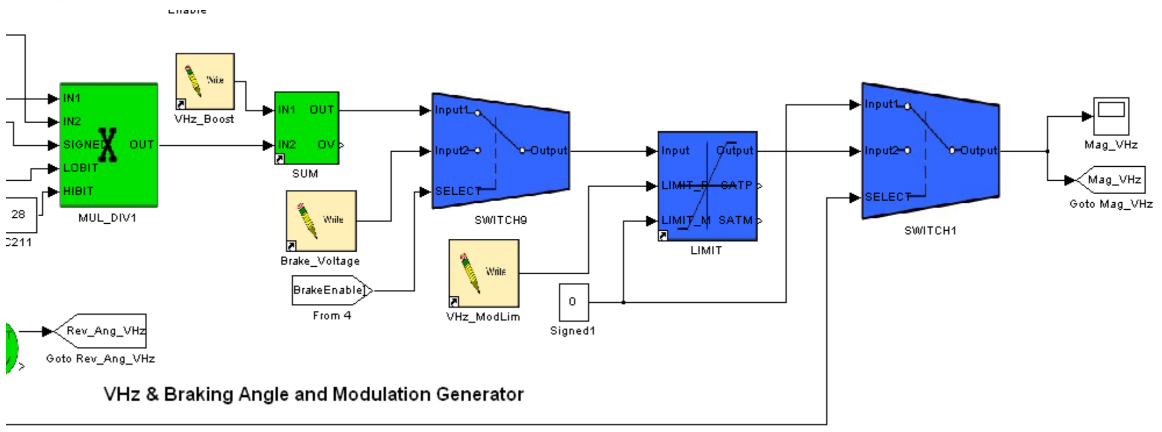


Figure 91—MCE Implementation of VHz (and Braking) Components (split into two parts)

### 6.2.1.2 Preflux for VHz Mode

In VHz mode, motor preflux is done in open loop (no closed-loop current control) fashion. Preflux is achieved by applying a DC boost voltage during the parking time. The voltage is set by VHz_Boost and the parking duration is specified by parameter ParkTm. In order to achieve preflux, a modification is made in the speed loop subsystem of the MCE design. The speed reference (TargetSpeed) is held at zero for the parking duration as shown in Figure 92. After the parking is done the speed reference is allowed to ramp to TargetSpeed. ParkTm is configured by the MCEWizard.

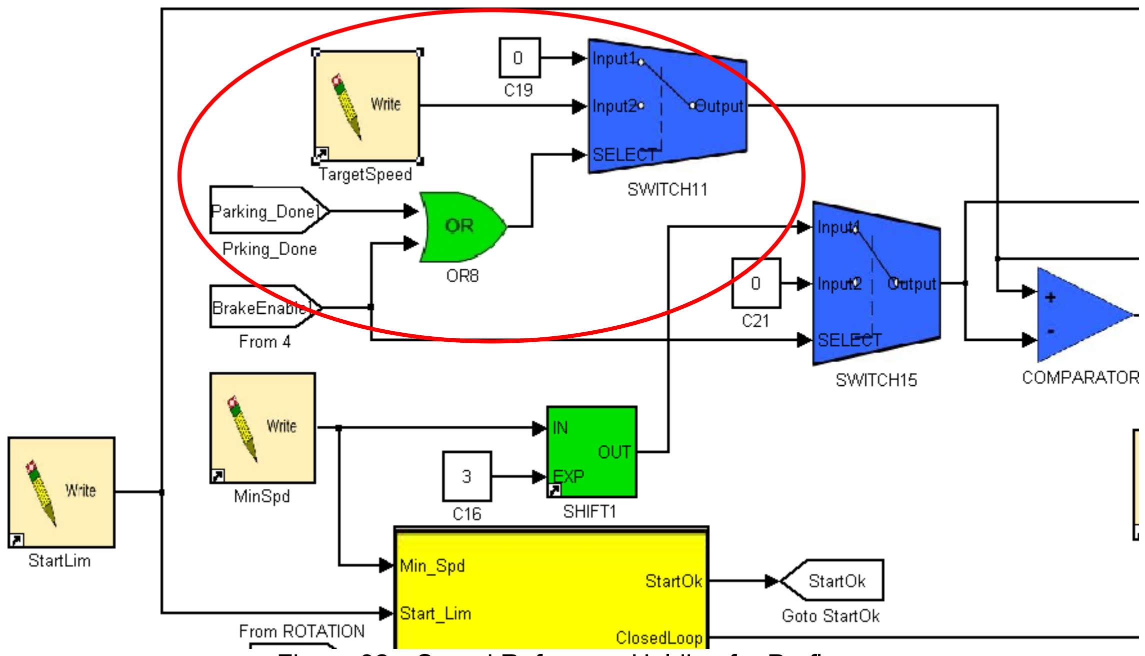


Figure 92—Speed Reference Holding for Preflux

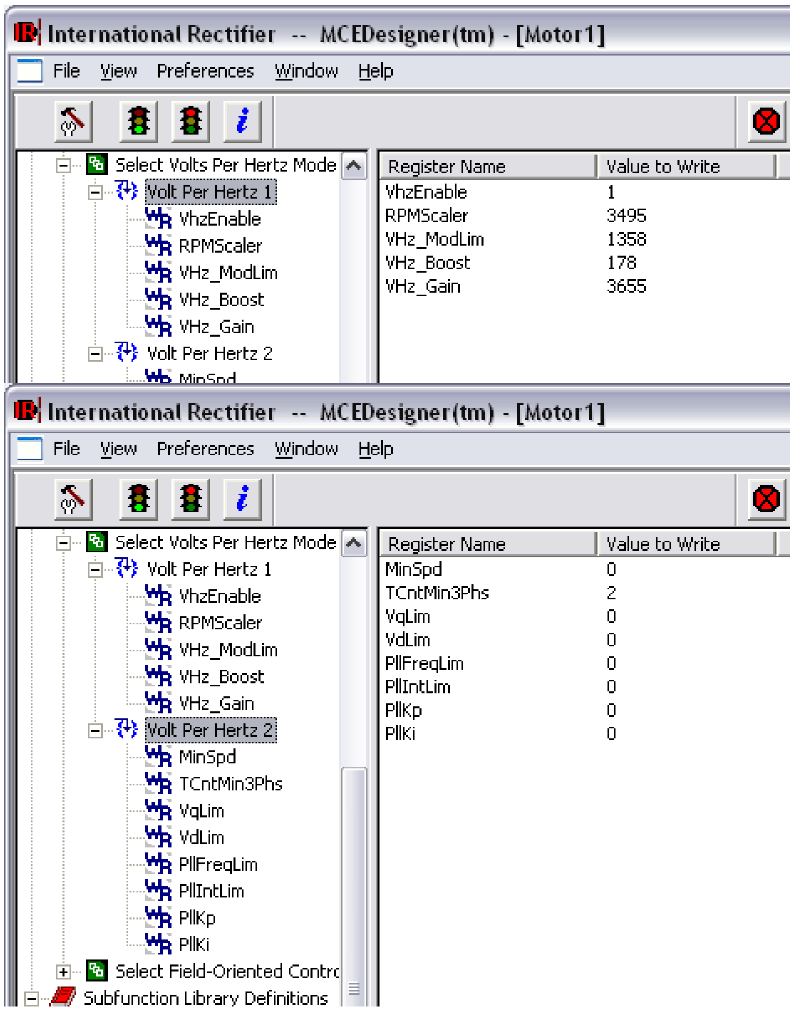
### 6.2.1.3 VHz Mode Selection

As mentioned earlier, certain parameters need to be set to select between VHz mode and FOC mode. These parameters can be set using the subfunctions *Volt Per Hertz 1* and *Volt Per Hertz 2*, as displayed in Figure 93. The upper part of the Figure shows the registers written by Volt Per

Hertz 1, which sets the parameters to configure and enable VHz mode. Except for VhzEnable, these values are calculated by the MCEWizard.

The lower part of Figure 93 shows the registers written to by the Volt Per Hertz 2 subfunction, which prevents certain FOC actions from interfering with the VHz mode:

- MinSpd = 0—This allows VHz operation to low motor speeds and prevents zero speed faults.
- TCntMin3Phs = 2—This reduces the minimum pulse width since current feedback is unnecessary in VHz mode.
- VqLim, VdLim = 0—These registers set the output of the current regulator to zero, so that the modulation is only set by writing to the UqFeedFwd register.
- PllFreqLim, PllIntLim, PllKp, PllKi = 0—Setting these registers to zero forces the angle estimator output to zero so that the ExtFwdAngle determines the motor electrical angle.


International Rectifier -- MCEDesigner(tm) - [Motor1]
File View Preferences Window Help
Select Volts Per Hertz Mode
Volt Per Hertz 1
VhzEnable
RPMScaler
VHz_ModLim
VHz_Boost
VHz_Gain
Volt Per Hertz 2
MinSpd
TCntMin3Phs
VqLim
VdLim
PllFreqLim
PllIntLim
PllKp
PllKi
Select Field-Oriented Contro
Subfunction Library Definitions

| Register Name | Value to Write |
|---|---|
| VhzEnable | 1 |
| RPMScaler | 3495 |
| VHz_ModLim | 1358 |
| VHz_Boost | 178 |
| VHz_Gain | 3655 |

| Register Name | Value to Write |
|---|---|
| MinSpd | 0 |
| TCntMin3Phs | 2 |
| VqLim | 0 |
| VdLim | 0 |
| PllFreqLim | 0 |
| PllIntLim | 0 |
| PllKp | 0 |
| PllKi | 0 |

Figure 93—VHz Mode Selection Parameters

### 6.2.1.4 Startup Characteristics of VHz Mode

A typical VHz startup current trace of an induction motor washing machine is shown in Figure 94. The current consumption during startup is higher than in the case of FOC control (Figure 101). This washer requires high (100%) startup torque, therefore a large voltage boost must be applied to ensure proper startup. In applications (for instance: fan, pump) where low startup torque can be tolerated, the voltage boost (VHz_Boost) can be reduced to decrease startup current. VHz can operate at lower speeds than the FOC control if high torque demand is not required for the application. In addition, VHz will provide better voltage utilization than FOC under deep field-weakening operation.

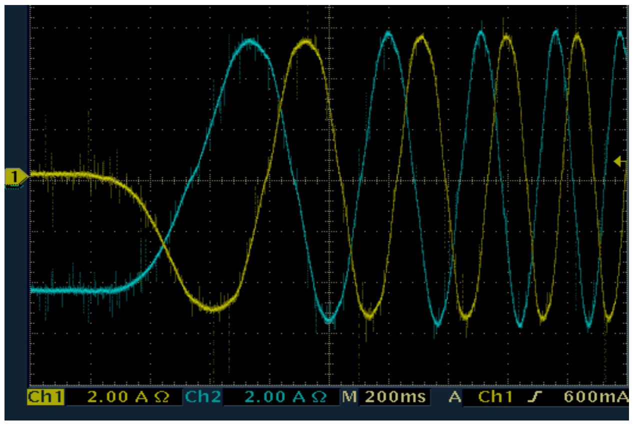


Figure 94—VHz Startup Currents
(Iw – yellow, Iu – Blue, horiz. 0.2sec/div, vert. 2A/div)

### 6.2.2 FOC Mode

FOC mode utilizes current and speed feedback to feed the current and speed loops. This mode of control provides the highest torque per ampere current rating, which is crucial for high starting torque applications. With proper parameter settings (using the MCEWizard), the existing PM FOC controller can be easily adapted for IM control. The adaptation requires two new add-ons: Slip compensation and Magnetization current injection.

#### 6.2.2.1 Slip Compensation

In a permanent magnet motor drive, the speed and motor frequency are equal (no slip). However, in the case of an induction motor, the frequency and speed are related by slip. The slip compensation addition is necessary to compensate for the speed inaccuracy due to the inherent slip speed of Induction motors. A simplified block diagram of the MCE slip compensation is shown in Figure 95. The input is torque current (Iq) and the output is Slip speed in digital counts. Figure 96 shows the implementation of slip compensation in Simulink, while Figure 97 shows how the Slip speed signal is used in the MCE speed loop.

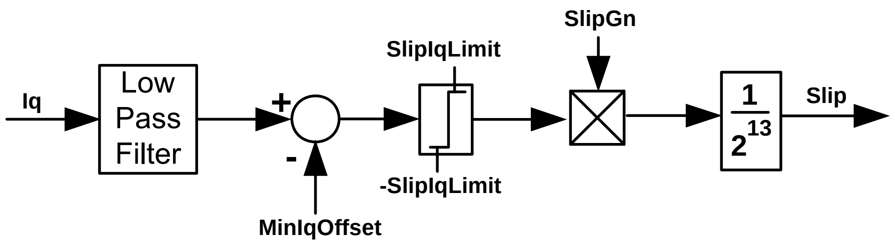


Figure 95—Slip Speed Compensation

The Slip speed compensator corrects the frequency (speed) output of the FOC block in order to account for motor slip. The torque current, Iq is first filtered, and then the MinIqOffset is subtracted from the filtered current. MinIqOffset accounts for core loss effects and is calculated by the MCEWizard from the core loss resistance. The SlipGn is a configurable parameter for the slip compensation. This parameter is calculated by the MCEWizard based on the motor rated speed and frequency information. Finally, the slip compensation is enabled only after the StartOk signal in order to enhance startup reliability.

***Slip Compensation Registers:***

**SlipGn**—In an induction motor, the shaft speed decreases as load increases (slipping). A slip gain adjustment (SlipGn) is provided to model this slip behavior.
Range: 0 – 32767 Scaling: Slip = Iq * SlipGn * Maximum Motor Speed / 2^27 [rpm]
**MinIqOffset—**This parameter models the effect of the core loss resistance in an induction motor. It is subtracted from the torque current feedback (Iq).
Range: 0 – 32767 Scaling: Offset Current = MinIqOffset / 4095 * Irated [Arms]
**SlipIqLimit—**This parameter sets a maximum torque current for slip compensation.
Range: 0 – 32767 Scaling: Limit Current = SlipIqLimit / 4095 * Irated [Arms]

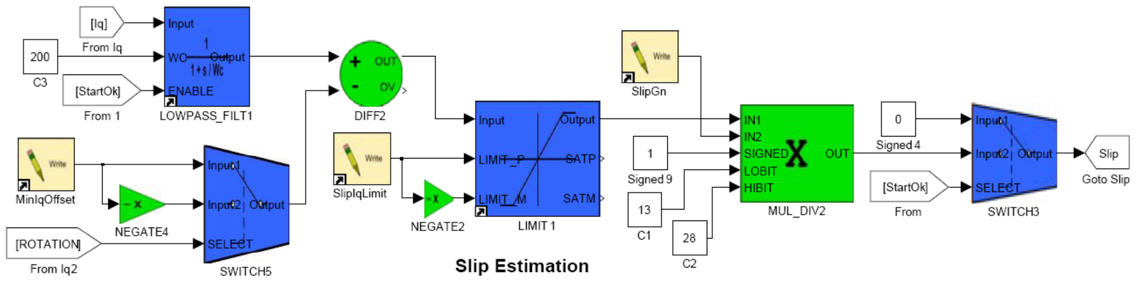


Figure 96—Slip Compensation Implementation

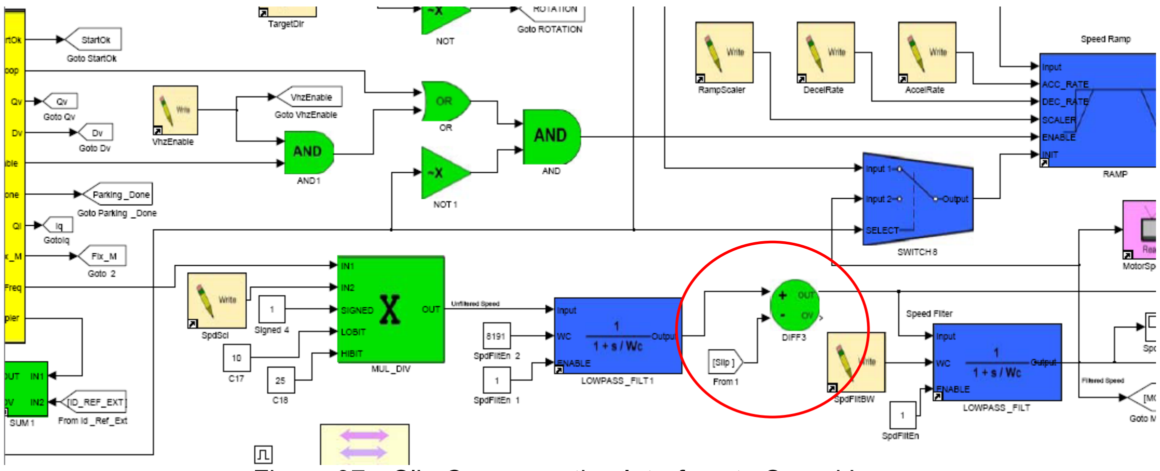


Figure 97—Slip Compensation Interface to Speed Loop

### 6.2.2.2 Magnetization Current Injection

A magnetization current parameter (Id_Mag) is summed into the d-axis current reference input (IdRefExt) of the Sensorless FOC block as shown in Figure 98 to support magnetizing current control of the motor. This parameter (Id_Ref: 4095 = rated current) is configured by the MCEWizard and sets the magnetization current level in FOC mode.

**Id_Mag**—This parameter sets the magnetization current level for the induction motor. It is summed with the Field Weakening current to give the Id current.
Range: -4096 – 4095 Scaling: Magnetization Current = Id_Mag * Irated / 4095 [Arms]

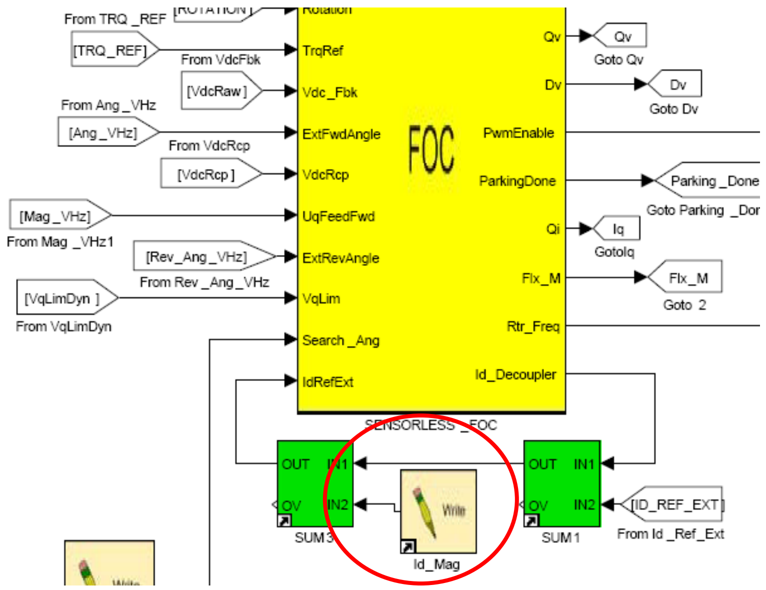


Figure 98—Magnetization Current Injection

### 6.2.2.3 Stopped Rotor Detection

The estimated flux is low pass filtered and then compared to a threshold, Flx_MThr, as shown in Figure 99. If the flux exceeds this level, then an OverLoad_Fault is latched in and also an MCE Fault is flagged in register FaultFlags. The MCEWizard sets the Flx_MThr to a fixed value of 950. The designer may modify this value to tune for the application. Note that the stopped rotor detection only applies to FOC mode.

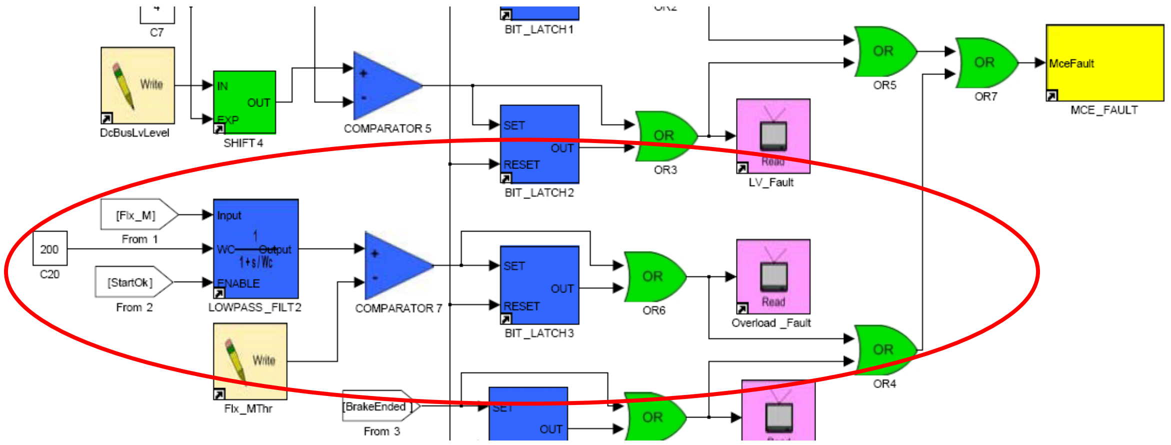


Figure 99—Stopped Rotor Detection

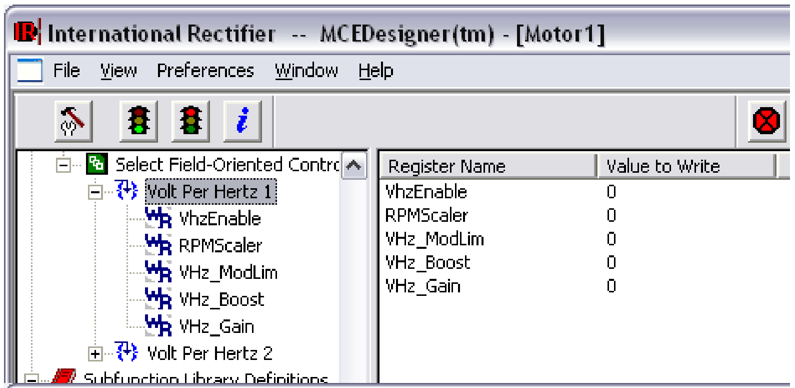
***Stopped Rotor Detection Registers:***
**Flx_MThr**—This parameter sets the flux level above which the rotor will be interpreted as stopped. This is signaled as a '1' in the Overload_Fault register.
Range: 0 – 4095 Scaling: Flux Threshold = Flx_MThr / 500 [% of Rated Flux]
**Overload_Fault**—This read register signals that the rotor flux has exceeded the Flx_MThr, indicating a stopped rotor condition. When this fault occurs, it is also signaled as an MCE Fault.
Range: 0 or 1 Scaling: 0 = No fault, 1 = Overload Fault

### 6.2.2.4 Preflux for FOC

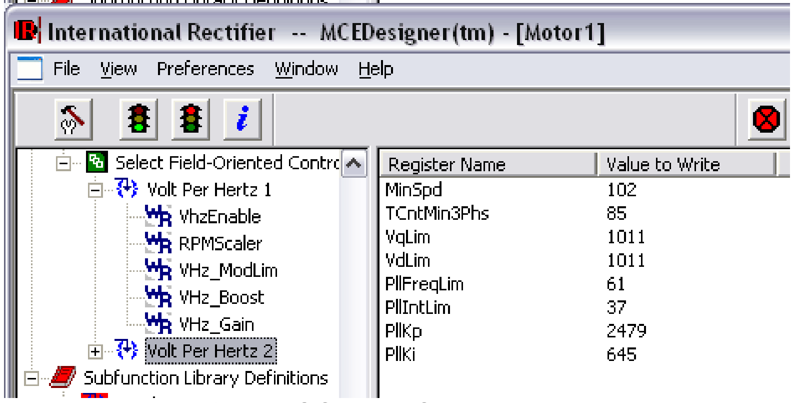
DC current injection applied in the parking time is used to preflux the induction motor before torque is applied. ParkTm and ParkI control the DC injection current level and duration. In the case of a PM motor these parameters are user entries in MCEWizard. However, for IM control, these parameters are calculated instead. Parking current is calculated based on the motor magnetization current level and parking time is calculated based on the IM rotor time constant.

### 6.2.2.5 FOC Mode Selection

In a similar manner to VHz mode, certain registers must be correctly configured to enable proper operation of the FOC mode. The upper image of Figure 100 shows that the VHz registers are all set to zero, while the lower image shows that the FOC registers are set to their appropriate values, calculated by MCEWizard. The parameter settings written by the Configure Motor function in MCEDesigner select the FOC mode by default.


International Rectifier -- MCEDesigner(tm) - [Motor1]
File View Preferences Window Help
Select Field-Oriented Contr
Volt Per Hertz 1
VhzEnable
RPMScaler
VHz_ModLim
VHz_Boost
VHz_Gain
Volt Per Hertz 2
Subfunction Library Definitions

| Register Name | Value to Write |
|---|---|
| VhzEnable | 0 |
| RPMScaler | 0 |
| VHz_ModLim | 0 |
| VHz_Boost | 0 |
| VHz_Gain | 0 |


International Rectifier -- MCEDesigner(tm) - [Motor1]
File View Preferences Window Help
Select Field-Oriented Contr
Volt Per Hertz 1
VhzEnable
RPMScaler
VHz_ModLim
VHz_Boost
VHz_Gain
Volt Per Hertz 2
Subfunction Library Definitions

| Register Name | Value to Write |
|---|---|
| MinSpd | 102 |
| TCntMin3Phs | 85 |
| VqLim | 1011 |
| VdLim | 1011 |
| PllFreqLim | 61 |
| PllIntLim | 37 |
| PllKp | 2479 |
| PllKi | 645 |

Figure 100—FOC Mode Selection Parameters

### 6.2.2.6 Startup Characteristic of FOC mode

A typical FOC startup current trace of an IM washing machine is shown in Figure 101. The initial DC current injection is the preflux (parking) period. If high starting torque is not required, the preflux period can be reduced.

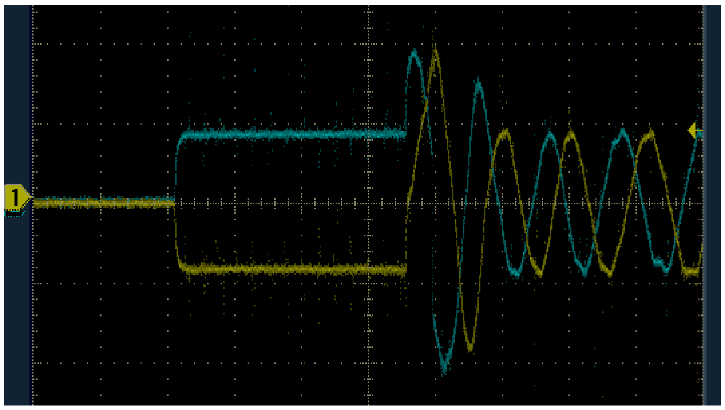


Figure 101—Startup Currents
(Iw – yellow, Iu – Blue, hor. 0.1sec/div, ver. 2A/div)

### 6.2.3 Other Features

#### 6.2.3.1 DC Bus Voltage Ripple Compensation

DC bus voltage ripple compensation is an additional feature of the IM control program. The function provides improved current stability especially if open-loop control (VHz mode) is employed. This compensation function is shown in Figure 102. The input is raw DC bus feedback (VdcRaw), which is obtained from the DC bus feedback block of the MCE speed loop subsystem (Figure 86). The compensation function generates an output (VdcRcp), which represents the inverse of the DC bus ripple. This signal is being fed to the FOC block (Figure 89) to dynamically adjust the modulation index in order to attenuate the effect of DC bus ripple on the motor current. Details on modulation adjustment using VdcRcp are given in the IRMCx300 Reference Manual

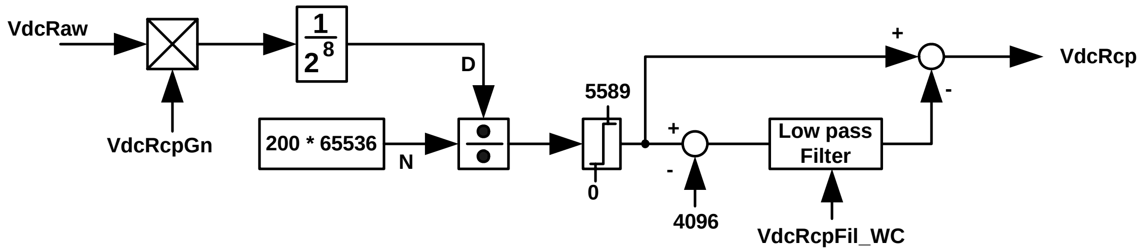


Figure 102—DC Bus Voltage Ripple Compensation

**Note:** The FOC block allows DC bus compensation only if the IregCompEnb bit is set to 1 (MtrCtrlBits_S[4], section 4.4.9 of IRMCx300 Reference Manual). This is automatically set by MCEWizard when the motor type is set to “Induction Motor.”

Normally, DC bus compensation is done using the direct inverse of the DC bus voltage feedback to compensate the inputs of the PWM modulator. When the average value of the DC bus falls (for example: low ac line), the PWM modulator will enter overmodulation (at high speeds) and the effectiveness of the DC bus ripple compensation will be significantly reduced. Here, DC bus compensation is done such that overmodulation can be avoided. The average value of the output (VdcRcp) is maintained at 4096 digital counts by a low pass filter (Figure 102). This is done to ensure that the PWM modulator operation can stay within the linear range (no overmodulation) even during a low ac line condition.

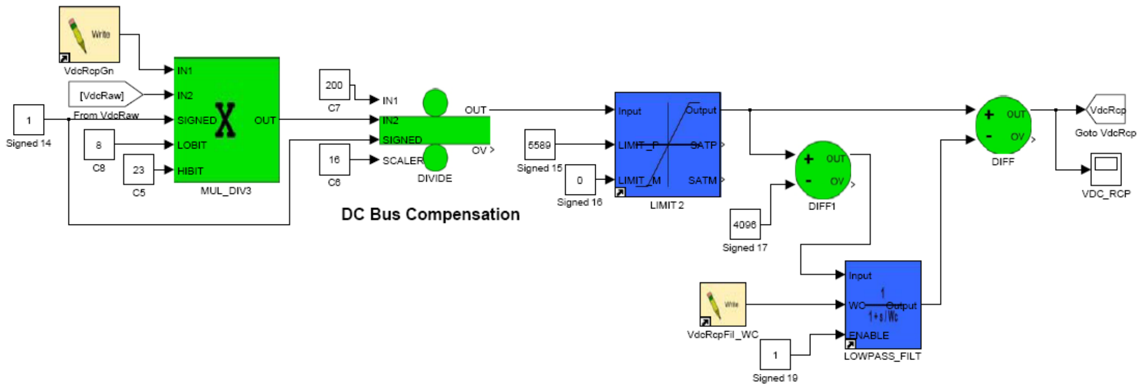
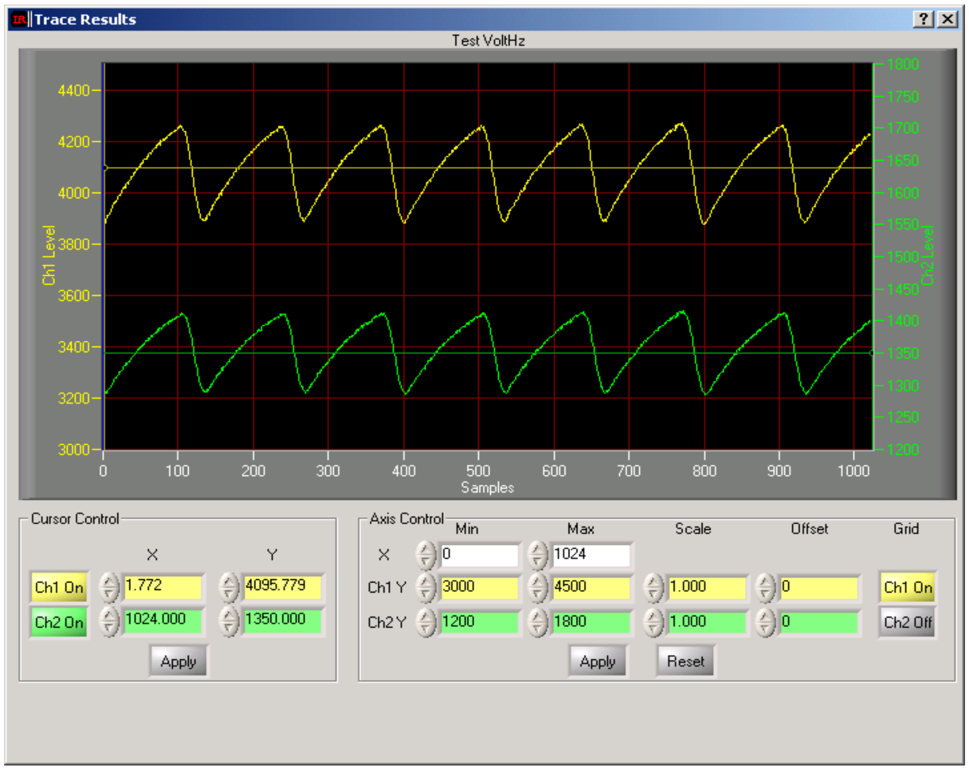


Figure 103—MCE Program Implementation of DC Bus Compensation

Figure 103 shows the implementation of the compensation function in the MCE program. Figure 104 illustrates the output signal (VdcRcp) and the actual modulation command (Qv). As can be observed from this figure, the average values of VdcRcp and Qv are fixed at 4096 (nominal VdcRcp) and 1358 (1430 = full modulation), respectively. In VHz mode, the limit of the average value of Qv can be adjusted using parameter VHz_ModLim. The MCEWizard calculates VHz_ModLim allowing a 5% modulation index margin to accommodate the DC bus ripple.



Cursor Control

| | X | Y |
|---|---|---|
| Ch1 On | 1.772 | 4095.779 |
| Ch2 On | 1024.000 | 1350.000 |

Apply

Axis Control

| | Min | Max | Scale | Offset | Grid |
|---|---|---|---|---|---|
| X | 0 | 1024 | | | |
| Ch1 Y | 3000 | 4500 | 1.000 | 0 | Ch1 On |
| Ch2 Y | 1200 | 1800 | 1.000 | 0 | Ch2 Off |

Apply Reset

Figure 104—VdcRcp (yellow) and Qv (green) Characteristics

***DC Bus Compensation Registers:***
**VdcRcpGn**—This parameter provides proper scaling for DC bus compensation.
Range: 0 – 32767
Scaling: VdcRcpGn = 819200 / (Nominal DC bus Voltage * DC bus feedback scaling)

**VdcRcpFil_WC**—This parameter sets the low pass filter bandwidth. This filter is used for maintaining the average value of VdcRcp at nominal digital counts (4096). The MCEWizard sets this filter time constant to 5ms.
Range: 0 – 8192  Scaling: Filter time constant = 2^13 * PwmPeriod / (VdcRcpFil_WC) [sec]

### 6.2.3.2 Non-Regenerative Braking

A non-regenerative braking function with brake ended detection is implemented in the IM controller. In this mode, the motor is driven at a low frequency in the opposite direction as the motion. This is an inefficient state of the motor and kinetic energy is dissipated in the motor windings.

Regenerative braking is still available in FOC mode; simply set the RegenLim register to a non-zero value and then command a decrease in speed. The motor kinetic energy will be regenerated to the DC bus, causing it to rise.

The non-regenerative braking function makes use of the same calculation path as the VHz mode to generate the angle and voltage parameters to the FOC block. Write to the TargetSpeed register to choose the braking frequency. Note that the MinSpd check is disabled during braking to ensure that the correct drive frequency is used. When BrakeEnabled is set high, the speed is automatically set to be in the opposite direction. The braking voltage modulation level is set by register Brake_Voltage. The implementation of the braking is shown in Figure 91 above.

To activate non-regenerative braking, the correct sequence of register writes must be performed, as listed below and also implemented in the function “Brake Function” in the MCEDesigner file IRMCS3041_Release_IMCtrl_2_0.irc:

1) Stop the PWM
2) Set TargetSpeed to desired (low) frequency
3) After charging the bootstrap capacitors, enable the PWM (not the FOC) by setting pwmctrl = 9
4) Set BrakeEnable = 1
5) Stop the PWM when BrakeEnded_Signal = 1. (MCEDesigner will do this automatically.)
6) Set BrakeEnable = 0
7) Clear the fault to start the drive again.

An additional feature of the braking is automatic shut-off; the MCE program implementation is shown in Figure 105 below. This shutoff is achieved by comparing the (filtered) torque current (Iq) with a threshold set by register BrakeThr. When the current falls below the threshold level the end of the braking is signaled by the register BrakeEnded_Signal. Also, the MCE Fault (in register FaultFlags) is tripped to signal that the drive should be stopped. For the brake end detection to work properly, the drive should be configured to FOC mode.

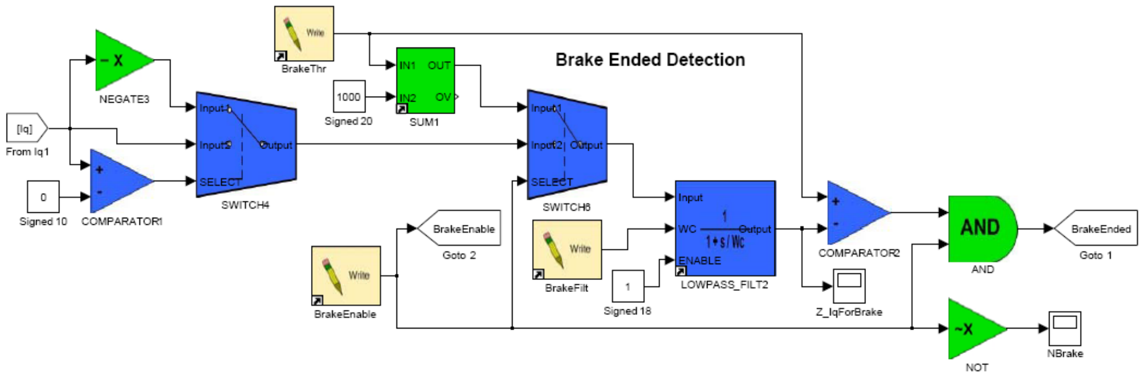


Figure 105—Brake Ended Detection

The tuning of the braking parameters (brake frequency, Brake_Voltage & BrakeThr) is best done by testing the desired induction motor. The MCEWizard gives parameters which will work for a wide variety of motors, but optimization must be done empirically. The trace parameter, Z_IqForBrake, is provided to aid in tuning the braking.

In order to tune the braking, use the NBrake signal as the trigger to trace the Z_IqForBrake and collect traces as shown below. Higher frequency braking is somewhat slower, but has a more distinct threshold for turning off the drive, as shown in Figure 106 below. If the braking voltage and frequency are not turned off at the right time, then the motor could start turning in the opposite direction.

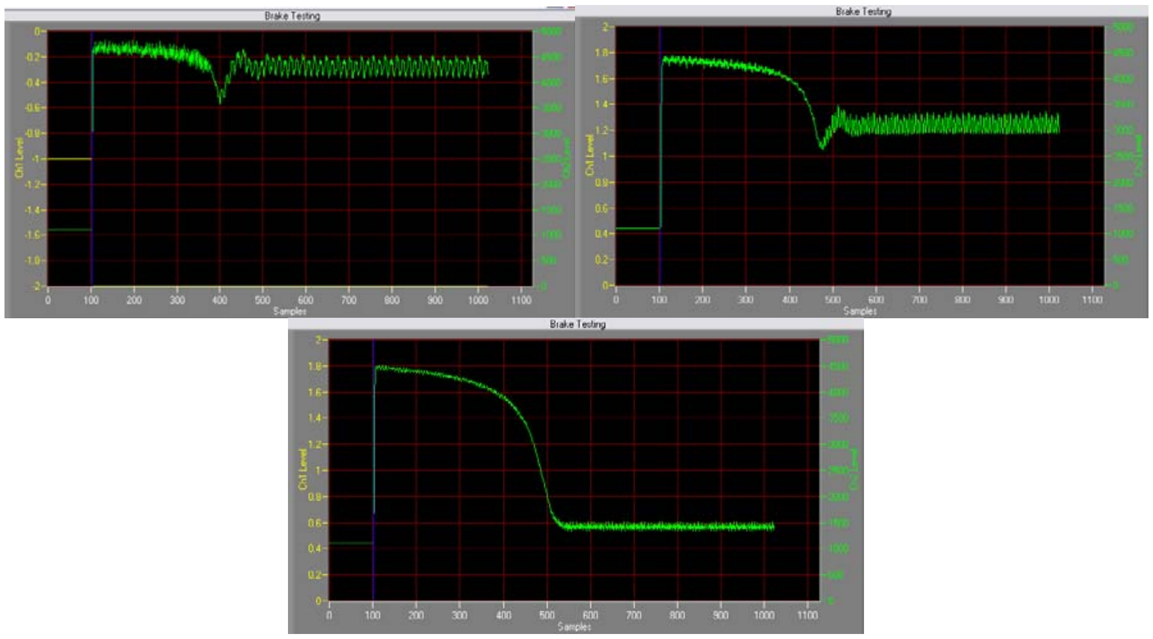


Figure 106—Z_IqForBrake signals for braking at several frequencies, with constant BrakeVoltage. TargetSpeed = 180 (upper left) 360 (upper right) and 720 (lower) in these plots.

***Non-regenerative Braking Registers:***
*Inputs*
**BrakeEnable**—Enables non-regenerative braking.
Range: 0 or 1 Scaling: 0 = No Braking, 1 = Braking Enabled
**Brake_Voltage**—This parameter configures the magnitude of the braking voltage as used in Figure 91. The MCEWizard sets this parameter to 100.
Range: 0 – 1430
Scaling: Rms line-to-line brake voltage = Brake_Voltage / 1430 * DC Bus Voltage / √(2) [Vrms]
**BrakeFilt**—This register sets the cutoff frequency of the lowpass filter of Iq for brake ended detection. The output of the filter is the Z_IqForBrake signal. MCEWizard sets this parameter to 100.
Range: 0 – 8192 Scaling: Filter time constant = 2^13 * PwmPeriod / BrakeFilt [sec]
**BrakeThr**—This parameter defines the Iq current level below which the braking should stop. The MCEWizard sets this parameter to 2500.
Range: 0 – 4095 Scaling: Braking Current Threshold = BrakeThr * Irated / 4095 [Arms]
*Outputs*
**Z_IqForBrake**—This is the signal which is compared to BrakeThr to determine the end of braking. It can be traced for the purpose of setting the braking frequency, voltage and threshold.
Range: 0 – 4095 Scaling: Braking Current = Z_IqForBrake * Irated / 4095 [Arms]

**BrakeOn**—This trace parameter mirrors BrakeEnable and is provided as a signal to trigger on for the purpose of tuning the braking registers.
Range: 0 or 1 Scaling: 0 = Braking Disabled, 1 = Braking Enabled
**BrakeEnded_Signal**—This read register signals that the braking should be turned off. When the brake ending is detected, it is also signaled as an MCE Fault.
Range: 0 or 1 Scaling: 0 = No Brake Ended, 1 = Brake End Detected

## 6.3 Parameter Configuration

The IRMCX300 motor controller requires certain motor parameters and other hardware specific parameters to run the motor. These values are entered into the MCEWizard parameter configuration utility as real numbers and then converted to scaled, digital versions that can be used by the controller. The induction motor can be configured to run in open loop using the basic parameters which are normally found on the motor nameplate. The full induction motor equivalent circuit parameters are required to run the motor in closed loop with FOC control. Figure 108 below shows the per-phase equivalent circuit for an induction motor. The equivalent circuit parameters should be supplied by the motor manufacturer in the datasheet but they can be measured experimentally if the information is not available. The measurement process is detailed in the following sections.

Most of the application-related parameter entries (in MCEWizard) for setting up the Permanent Magnet motor controller also apply for the Induction motor controller.

The majority of IM manufacturers specify the frequency and rpm value corresponding to rated flux operation (Point A of Figure 107). The frequency and rpm values that MCEWizard expects are based on the rated magnetization current level (Point A). Point B specifies the maximum frequency of the motor. This point corresponds to reduced flux operation (and therefore reduced torque). Please be aware that some manufacturers may specify point B in the motor nameplate instead of point A.

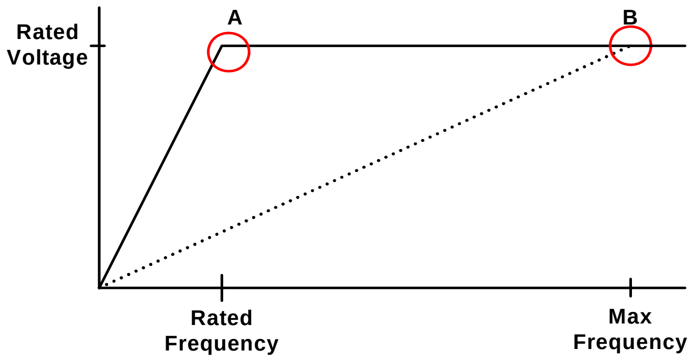


Figure 107—Motor Nameplate Operating Points

### 6.3.1 Configuring for testing in VHz Mode

The procedure for measuring the motor circuit parameters begins by using MCEDesigner to run the motor in open loop (VHz) mode. Using the No-Load test and the Blocked Rotor test, the circuit parameters can be derived and entered into MCEWizard for configuring the drive.

The following motor parameters are needed to configure the induction motor for VHz mode:

Motor Rated Line Volts ($V_{rated}$)
Motor Rated Amps ($I_{rated}$)
Motor Rated Frequency ($f_{rated}$)

Motor Rated Speed ($w_{rated}$)
Motor Poles (p)
Motor Stator Resistance ($R_1$)
Motor Max RPM

The first four parameters are typically listed on the motor nameplate. The motor pole number, p, can be calculated from the ratio of the rated frequency, $f_{RATED}$ in Hz and speed, $w_{RATED}$ in RPM:

$$\frac{p}{2} = \frac{f_{rated}}{\left(w_{rated}/60\right)}$$

(Round down to the nearest even integer to account for slip)
The stator winding resistance is measured directly using an ohmmeter. Measure the line to line resistance for all three phase pairs 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 per phase resistance of the stator: $R_1$. Finally, the Motor Max RPM should be less than or equal to point B of Figure 107.

Enter the values for the parameters listed above into MCEWizard. When starting the MCEWIzard program select the appropriate reference design kit from the Welcome page, then on the “Base Configuration Options” page, select “Induction Motor” from the “Motor Type” selection menu to enable induction motor questions. Enter the six parameters listed above and leave the remaining default values. From the “Verify & Save” page, export the drive parameters to a .txt file. Import the .txt file into MCEDesigner to prepare to run the motor in VHz mode.

Next, use MCEDesigner to enter VHz mode by first running the “Configure Motor” function and then the “Select VHz Mode” function. Finally, run the “Start Motor” function and the motor should begin to turn.

## 6.3.2 Parameter Measurement

The parameter tests recommended by the IEEE require the measurement of the motor input voltage, current and power (or power factor) when driven from a fixed frequency variable ac supply. The measurements are made when motor is unloaded (No Load Test) and when the rotor is blocked (Blocked Rotor Test).

Figure 108 shows the IM per phase equivalent circuit model, where the $R_1$ and $L_1$ refer to the *stator* resistance and leakage inductance, respectively, and $R_2$ and $L_2$ refer to the corresponding *rotor* values. $L_m$ and $I_m$ are the magnetization inductance and current, while $R_{core}$ models the core losses including eddy currents and no-load friction. The tests and calculations described in the next sections will provide the parameters of this model. Note that in the tests to follow, $R_{core}$ will be assumed to be large enough to neglect, though it will be estimated.

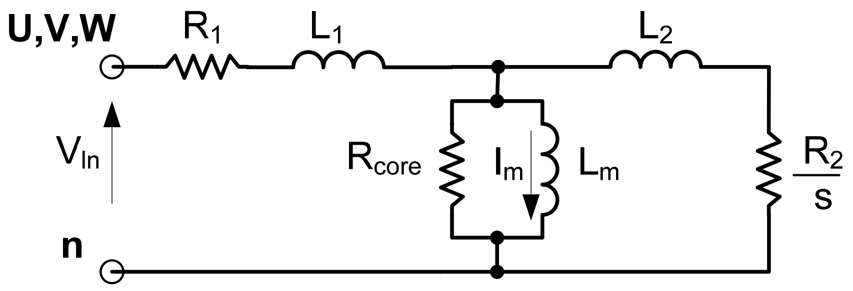


Figure 108—Induction motor per phase equivalent circuit

### 6.3.2.1 Measurement Setup

Figure 109 illustrates the power measurement circuit, showing the phase current $I_L$ and line-to-line voltage $V_L$. A 3-phase variable voltage supply could be achieved using a 3-phase AC line and a 3-phase Variac, but the procedure below will utilize the Reference Design Kit to drive the motor. A power meter measures the current in two phases and the voltage in the two phases relative to the third phase. In general when testing motors, it is also a good practice to monitor the phase current, for example with a current probe and oscilloscope.

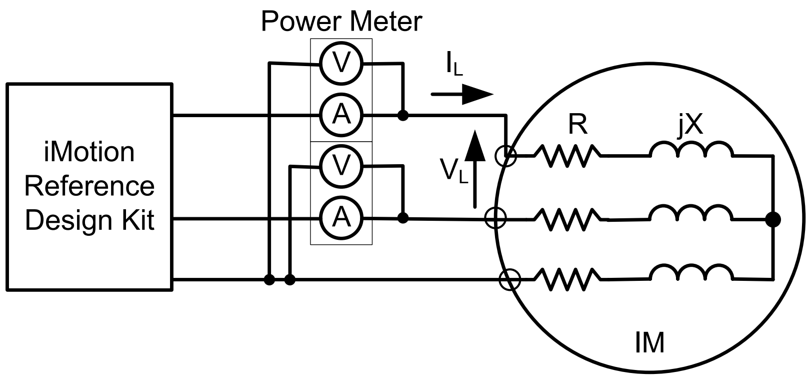


Figure 109—Parameter measurement circuit

### 6.3.2.2 No Load Test

The motor should be driven at rated voltage and rated frequency without any shaft load. Once the motor has been started in VHz mode, use the “Reference Speed” function to run the motor at rated voltage and frequency. (Enter [$f_{rated}$ * 60 * 2 / p] in Reference Speed to command the correct speed.)

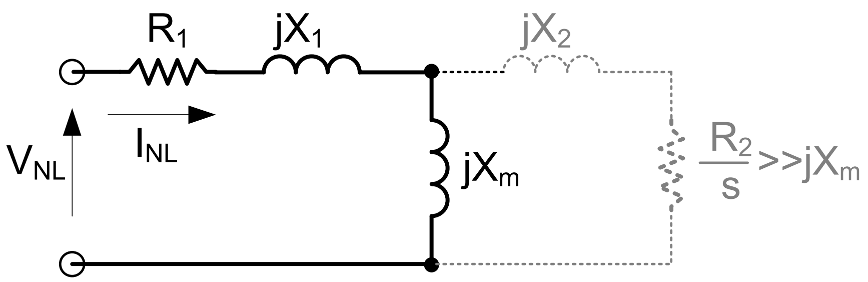


Figure 110—No Load circuit

When the motor is unloaded the slip is close to zero and the rotor branch of the equivalent circuit can be ignored (Figure 110). The no-load input power ($P_{NL}$), current ($I_{NL}$) and line voltage ($V_{NL}$) should be noted. Referring to the measurement circuit in Figure 109, the line voltage and input current are calculated as follows:

$$V_{NL} = \frac{V_L}{\sqrt{3}}, I_{NL} = I_L$$

Then the no-load impedance ($Z_{NL}$) and resistance ($R_{NL}$) can be calculated:

$$Z_{NL} = \frac{V_{NL}}{I_{NL}}, R_{NL} = \frac{P_{NL}}{3{I_{NL}}^2}$$

$$Z_{NL} = R_{NL} + j(X_1 + X_m)$$

where $X_i = 2\pi f_{rated} L_i$

Note that $R_{NL}$ includes the stator winding resistance as well as rotational losses. The no load input reactance can be calculated from:

$$X_{NL} = \sqrt{\left(Z_{NL}{}^2 - R_{NL}{}^2\right)}$$

The no load input reactance is the sum of the stator leakage reactance and the magnetizing reactance. These parameters will be separated in the next test.

$$X_{NL} = X_1 + X_m$$

The core and rotational losses ($P_{core}$) are derived from the no load power by subtracting the no load stator winding losses, and then the core loss resistance ($R_{core}$) can be approximated:

$$P_{core} = P_{NL} - 3I_{NL}^2 R_1$$

$$R_{core} \approx \frac{V_{NL}^2}{P_{core}/3}$$

Finally, at no load the phase current equals the magnetizing current:

$$I_m = I_{NL}$$

6.3.2.3 Blocked Rotor Test

For this test, the motor is run at rated current and rated frequency while the rotor is fixed so that it cannot rotate. Before starting the motor, reduce VHz_Gain to 1/10 of the configuration value given by MCEWizard. Next, start the motor and use Reference Speed to achieve the rated frequency. Gradually increase the VHz_Gain until the phase current achieves the rated level. The phase current can be monitored using a current probe or by tracing $I_q$ and $I_d$. When $\sqrt{(I_q^2 + I_d^2)} = 4096$, then the phase current has reached the rated current.

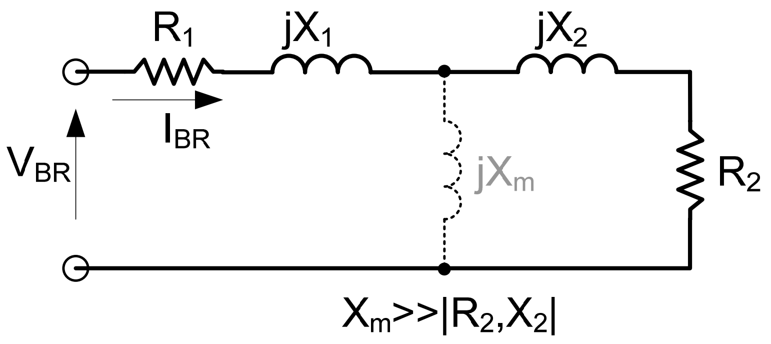


Figure 111—Blocked Rotor circuit

When the rotor is blocked, the rotor circuit impedance is very low and the magnetizing branch of the equivalent circuit may be ignored (Figure 111) so that:

$$Z_{BR} = R_1 + R_2 + j\left(X_1 + X_2\right)$$

The blocked-rotor input power ($P_{BR}$), current ($I_{BR}$) and line voltage ($V_{BR}$) should be noted. Recall that, according the measurement circuit in Figure 109:

$$V_{NL} = \frac{V_L}{\sqrt{3}}, I_{NL} = I_L$$

The input resistance ($R_{BR}$) and reactance ($X_{BR}$) can now be calculated.

$$R_{BR} = \frac{P_{BR}}{3I_{BR}^2}$$

$$Z_{BR} = \frac{V_{BR}}{I_{BR}}$$

$$X_{BR} = \sqrt{\left(Z_{BR}{}^2 - R_{BR}{}^2\right)}$$

The blocked rotor resistance is the sum of the stator and rotor resistance so the rotor resistance can now be determined:

$$R_2 = R_{BR} - R_1$$

The blocked rotor reactance is the sum of the stator and rotor leakage reactance. Assume the stator and rotor reactance are the same and so that the leakage inductances can be determined:

$$X_1 = X_2 = \frac{X_{BR}}{2}$$

$$L_1 = L_2 = \frac{X_1}{2\pi f_{rated}}$$

The magnetizing inductance can now be determined:

$$X_m = X_{NL} - X_1$$

$$L_m = \frac{X_m}{2\pi f_{rated}}$$

Note that setting $X_1 = X_2$ is an approximation and may be different for different classes of motors. The table below gives the distribution of leakage reactance for several types of induction motors.

| **Motor Type** | **Blocked Rotor Leakage Reactance Distribution** | |
|---|---|---|
| | $X_1$ | $X_2$ |
| Squirrel-cage Class A | $0.5X_{BR}$ | $0.5X_{BR}$ |
| Squirrel-cage Class B | $0.4X_{BR}$ | $0.6X_{BR}$ |
| Squirrel-cage Class C | $0.3X_{BR}$ | $0.7X_{BR}$ |
| Squirrel-cage Class D | $0.5X_{BR}$ | $0.5X_{BR}$ |
| Wound rotor | $0.5X_{BR}$ | $0.5X_{BR}$ |

### 6.3.3 Configuring for FOC Mode

Using the motor characteristics measured and calculated in the previous section, the full FOC configuration parameters can be obtained. Return to MCEWizard and fill in the following fields, in addition to the values from VHz Mode:

Motor Stator Leakage Inductance ($L_1$)
Motor Rotor Resistance ($R_2$)
Motor Rotor Leakage Inductance ($L_2$)
Motor Core Loss Inductance ($R_{core}$)
Motor Magnetization Inductance ($L_m$)
Motor Magnetization Current ($I_m$)
Minimum Running Speed
Closed-Loop Switch-Over Speed

Set the Minimum Running Speed to 15% of the Motor Rated Speed and the Closed-Loop Switch-Over Speed to 10% of the Motor Rated Speed. (This ensures that the speed command is higher than the closed loop speed at startup, due to slip.)

Go to the "Verify & Save" page, export the drive parameters and import them into MCEDesigner. Configure and then run "Start Motor" to control the motor in FOC mode. Tips for starting the motor are given in the next section.

### 6.3.4 Parameter Estimation by Saturation Curve

For some applications it may not be possible to block the rotor without damage, or a power meter may not be available. In these situations, the motor parameters may be estimated using the saturation curve method described below.

Begin by configuring the motor to run in V/Hz mode as described in Section 6.3.1. Before starting, reduce the VHz_Gain to 1/10 of the configured value and then run the motor unloaded at the rated frequency (or the frequency at which the motor will operate). Gradually increase the VHz_Gain, measuring the phase current $I_L$ and line-to-line voltage $V_L$. Plot the phase current $I_L$ vs phase voltage $V_L/\sqrt{3}$ as shown in Figure 112.

Leakage Inductance ($L_1 + L_2$)—Measure the line-line inductance at each phase pair. Average the values and then divide by two to get an estimate of the total leakage inductance at each phase. This assumes that the magnetization inductance is much larger than the rotor leakage inductance.

Magnetization Amps ($I_m$)—The point at which the curve begins to deviate from a straight line defines the magnetization current. Figure 112 below shows an example saturation curve and the approximation of the magnetization current of 0.5 A at the dashed line.

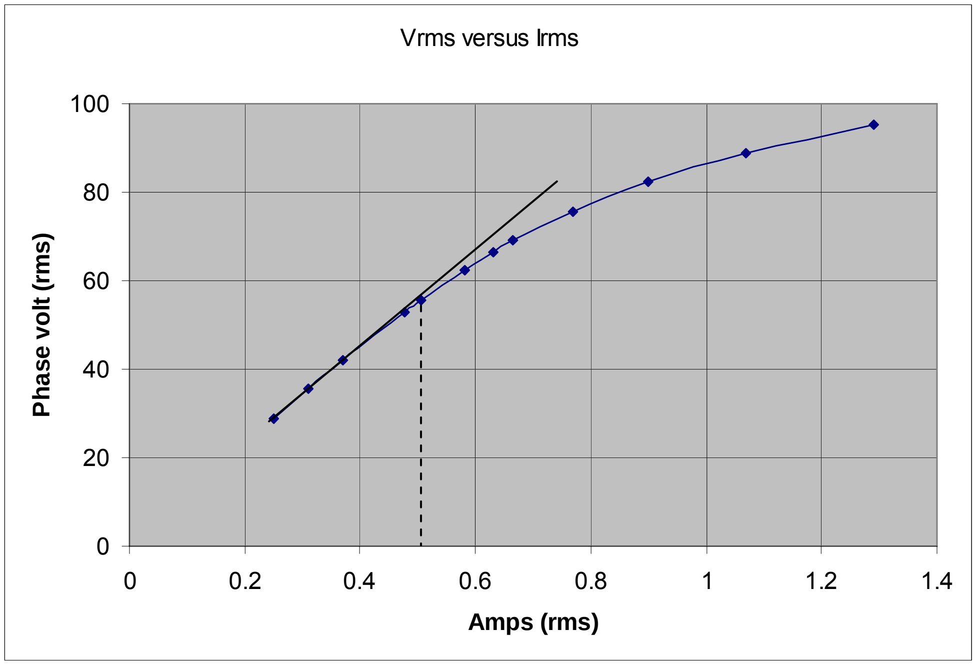


Figure 112—Saturation Curve

Magnetization Inductance ($L_m$)—At the magnetization current, calculate the magnetization inductance as follows

$$L_m = \frac{V_L / \sqrt{3}}{2\pi f_{rated} I_m} - L_2$$

Rotor Resistance ($R_2$)—At the magnetization current, calculate the rotor resistance as follows

$$R_2 = \frac{L_m \cdot I_m \cdot w_{slip} \cdot (2\pi/60)(p/2)}{\sqrt{\left(I_{rated}{}^2 - I_m{}^2\right)}}$$

where $w_{slip}$ is the slip mechanical frequency at the rated conditions:

$$w_{slip} = f_{rated} \cdot 60 \cdot 2/p - w_{rated}$$

## 6.4 General Tips

**Starting**

In FOC mode if the motor is unloaded, then at start-up the rotor can begin turning and then the current drops to zero, resulting in a Fault and a failed start. Sometimes the rotor may turn, stop, turn and stop rapidly before finally triggering a Fault. This happens because the motor speed rises quickly past the speed reference, causing the torque reference to fall to zero and thereby eliminating the torque current. The fix is to increase the integral gain of the speed loop PI block.

The other starting tips from Chapter 2 also apply: increase the closed loop frequency or reduce the value of register KTorque, but do not modify the parking parameters. The purpose of the parking parameters is to provide a preflux of the induction motor.

**Magnetization Current**

The magnetization current, as measured above, will allow the motor to deliver the rated torque. However, if the rated torque is not needed in the application a reduced magnetization current may be used. This can reduce motor and inverter heating during operation. Other configuration parameters may need to change if the magnetization current is reduced, for example, the closed loop speed may need to be increased to in order to achieve good angle estimation.

**Slip Tuning**

The slip estimation parameters may require some tuning, particularly when the load torque is much smaller (or larger) than the rated torque.

**Speed Range**

Note that some induction motors are designed to operate only at 60Hz line frequency. Attempting to run the motor at speeds significantly different the rated frequency may have problems such as increased acoustic noise or instability, which cannot be improved by tuning the drive parameters.

**Trademarks of Infineon Technologies AG**
µHVIC™, µIPM™, µPFC™, AU-ConvertIR™, AURIX™, C166™, CanPAK™, CIPOS™, CIPURSE™, CoolDP™, CoolGaN™, COOLiR™, CoolMOS™, CoolSET™, CoolSiC™, DAVE™, DI-POL™, DirectFET™, DrBlade™, EasyPIM™, EconoBRIDGE™, EconoDUAL™, EconoPACK™, EconoPIM™, EiceDRIVER™, eupec™, FCOS™, GaNpowIR™, HEXFET™, HITFET™, HybridPACK™, iMOTION™, IRAM™, ISOFACE™, IsoPACK™, LEDrivIR™, LITIX™, MIPAQ™, ModSTACK™, my-d™, NovalithIC™, OPTIGA™, OptiMOS™, ORIGA™, PowIRaudio™, PowIRStage™, PrimePACK™, PrimeSTACK™, PROFET™, PRO-SIL™, RASIC™, REAL3™, SmartLEWIS™, SOLID FLASH™, SPOC™, StrongIRFET™, SupIRBuck™, TEMPFET™, TRENCHSTOP™, TriCore™, UHVIC™, XHP™, XMC™

Trademarks updated November 2015

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

Edition 2009-09-22

Published by
Infineon Technologies AG
81726 Munich, Germany

© 2016 Infineon Technologies AG.
All Rights Reserved.

Do you have a question about this document?
Email: erratum@infineon.com

Document reference
IRMCx300_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.