International
IOR Rectifier

USER GUIDE

User Guide #0609

IRMCx300
Reference Manual

23 October 2009
Version 2.11

International
IOR Rectifier

TABLE OF CONTENTS

User Guide #0609 1
1. Overview 9
1.1 System Components 11
1.2 Memory Map 12
1.3 Byte Ordering 13
2 Reset Mechanism and Boot Process 14
2.1 Power Sequencing at Startup 16
2.2 EEPROM Data Format 18
3 8051 Microcontroller 20
3.1 Instruction Set 20
3.2 Special Function Registers 24
3.2.1 Processor Registers 27
3.2.2 General Purpose I/O 29
3.2.3 Clock Selection and PLL Frequency Configuration 33
3.2.4 Miscellaneous Functions 36
3.3 Interrupts 39
3.3.1 Standard Interrupts 39
3.3.2 Extended Interrupts 40
3.3.3 P4 Interrupts 40
3.3.4 Enabling Interrupts 40
3.3.5 Interrupt Priority 42
3.3.6 Service Order 43
3.3.7 Interrupt Latency 44
3.3.8 Interrupt Vectors 44
3.4 Timers 45
3.4.1 Timer Prescaler 45
3.4.2 General-Purpose Timer/Counters 45
3.4.2.1 Modes of Operation 48
3.4.2.2 Configuring the Timers 48
3.4.2.3 Using the Timers to Measure a Time Interval 48
3.4.2.4 Using the Timers to Signal When a Defined Period Has Elapsed 49
3.4.2.5 Using the Timers as Event Counters 49
3.4.2.6 Reading the Timers 49
3.4.3 Periodic Timer 49
3.4.4 Watchdog Timer 50
3.4.5 Capture Timer 51
3.5 UARTs 53
3.6 D/A PWM 56
3.7 I^2C / SPI Serial Interface 57
3.7.1 Command Descriptions for the I^2C Interface 59
3.7.1.1 Reset Command 59
3.7.1.2 Read and Write Commands 59
3.7.2 Command Descriptions for the SPI Interface 60
3.7.2.1 Read Instructions 60
3.7.2.2 Write Instructions 61
4 Motion Control Engine 62
4.1 Rotating Frame Notation and Conventions 64
4.2 Control Blocks 65
4.2.1 Frequency Domain Blocks 65
4.2.1.1 PI – Proportional Plus Integral 65
4.2.1.2 LOWPASS_FILT – First Order Low Pass Filter 68
4.2.1.3 HIGHPASS_FILT – First Order High Pass Filter 70
4.2.2 Coordinate Transformation Blocks 72

4.2.2.1 VECROT – Vector Rotation 72
4.2.2.2 CLARK – Inverse Clark Transformation 73
4.2.3 Utility Blocks 75
4.2.3.1 LIMIT 75
4.2.3.2 RAMP – Linear Ramp 76
4.2.3.3 ATAN – Arc Tangent block 78
4.2.3.4 FUNCTION_BLOCK 79
4.2.3.5 COMPARATOR 81
4.2.3.6 SWITCH 81
4.2.3.7 BIT_LATCH 82
4.2.3.8 PEAK_DETECT 83
4.2.3.9 TRANSITION – One Shot Pulse Generator 84
4.2.3.10 INTEGRAL2 – Integral with Limit 85
4.2.3.11 PFC_FFD 87
4.2.4 Math 88
4.2.4.1 DIFF – Subtraction 88
4.2.4.2 SUM – Addition 89
4.2.4.3 ACCUMULATOR 89
4.2.4.4 COUNTER 91
4.2.4.5 SHIFT – Multiply by a Power of Two 91
4.2.4.6 MUL_DIV – Signed / Unsigned Multiplier with Extraction 92
4.2.4.7 DIVIDE 94
4.2.4.8 NOT – Bitwise Inversion 95
4.2.4.9 NEGATE – Two's Complement 96
4.2.4.10 AND – Bitwise Logical AND 96
4.2.4.11 OR – Bitwise Logical OR 97
4.2.4.12 XOR – Bitwise Logical Exclusive OR 97
4.3 Motion Peripherals 98
4.3.1 SENSORLESS_FOC 98
4.3.1.1 Current Control 101
4.3.1.2 Rotor Position Estimation 102
4.3.1.3 Startup Control 104
4.3.2 SINGLE_I_SHUNT 106
4.3.3 DC_BUS_VOLTAGE 109
4.3.4 A_D – A/D Converter 110
4.3.5 Low Loss Space Vector PWM 112
4.3.5.1 SVPWM Transfer Characteristics 115
4.3.5.2 Deadtime Insertion Logic 116
4.3.5.3 Three-Phase and Two-Phase Modulation 117
4.3.5.4 Guard Band 117
4.3.5.5 PWM Pre-Charge Control 118
4.3.5.6 Duty Ratio Control Mode 119
4.3.6 FAULTS Block 120
4.3.7 MCE_FAULT Generator 121
4.3.8 PFC_PWM 122
4.3.8.1 PFC PWM Generation 124
4.3.8.2 PFC PWM Blanking 125
4.3.9 PFC_SENSE 127
4.4 Motion Peripheral Registers 129
4.4.1 System Write Register Group 129
4.4.2 PWM Configuration Write Register Group 132
4.4.3 Current Feedback Configuration Write Register Group 136
4.4.4 System Control Write Register Group 137
4.4.5 Torque Loop Configuration Write Register Group 138
4.4.6 Velocity Control Write Register Group 140

4.4.7 Closed Loop Angle Estimator Write Register Group 142
4.4.8 Open Loop Angle Estimator Write Register Group 147
4.4.9 Startup Angle Estimator Write Register Group 148
4.4.10 Startup Retrial Write Register Group 151
4.4.11 Phase Loss Detect Write Register Group 153
4.4.12 Single Shunt Write Register Group 154
4.4.13 Start/Stop Sequencing Write Register Group 156
4.4.14 Catch Spin Write Register Group 157
4.4.15 User Control Write Register Group 160
4.4.16 Field Weakening Control Write Register Group 162
4.4.17 Protection Write Register Group 163
4.4.18 External Signals Write Register Group 165
4.4.19 PFC Control Write Register Group 167
4.4.20 System Read Register Group 172
4.4.21 System Status Read Register Group 174
4.4.22 DC Bus Voltage Read Register Group 175
4.4.23 FOC Diagnostic Data Read Register Group 176
4.4.24 Velocity Status Read Register Group 179
4.4.25 Current Feedback Offset Read Register Group 180
4.4.26 PFC Status Read Register Group 181
5 8051 / MCE Interface 183
5.1 The Shared RAM 183
5.1.1 Reading and Writing Shared RAM 183
5.1.2 Arbitration 184
5.2 Motion Peripheral Register Interface 185
5.3 Interrupts from the MCE to the 8051 186
6 The MCE Development Process 188
6.1 MCE Design Generation 189
6.2 Creating an MCE Design Using Simulink 190
6.2.1 Creating a Complete System Design 190
6.2.2 Creating a Macro Block Definition 191
6.3 Simulink MCE Design Components 193
6.3.1 The MCE Library 193
6.3.1.1 Configuration 193
6.3.1.2 Registers 193
6.3.1.3 Control 193
6.3.1.4 Math 193
6.3.1.5 Tools 194
6.3.1.6 Motion Peripherals 194
6.3.1.7 Designs 194
6.3.2 Standard Simulink Library Components 194
6.3.2.1 Enabled Subsystem 194
6.3.2.2 Constant 194
6.3.2.3 Scope 194
6.3.2.4 Input Port 194
6.3.2.5 Output Port 194
6.3.2.6 Goto and From 194
6.3.2.7 Unit Delay 195
6.4 The MCE Compiler 196
6.5 The Host Register Summary Utility 199
6.6 Customizing Motion Peripheral Library Blocks 200
6.7 MCE Design Hierarchical Format 202
7 The 8051 Development Process 204
7.1 Source Code Samples 204
7.1.1 EEPROM Programming 204

7.1.2 Register Interface 204
7.1.3 UART Driver 204
7.1.4 MCE Initialization 205
7.1.5 Motor Control 206
7.1.6 Other Operations 207
7.2 The Keil and FS2 Tools 208
7.2.1 Software Installation 208
7.2.2 Software Setup 208
7.2.3 The Keil Compiler 208
7.2.4 Debugging 209
7.3 Storing 8051 and MCE Code in EEPROM 210

LIST OF FIGURES

Figure 1. Typical Application Block Diagram Using IRMCF312 9
Figure 2. IRMCF312 Internal Block Diagram 11
Figure 3. Memory Map of IRMCF300 12
Figure 4. Reset Module 14
Figure 5. Reset and Boot Process for F-version 15
Figure 6. Reset and Boot Process for K-version. 16
Figure 7. Exact power sequence for IRMCF341 – Obtained in IRMCS3041 reference demo board 17
Figure 8. Detail of power supply circuit in IRMCS3041 reference demo board 17
Figure 9. Port Structure of IRMCF300 30
Figure 10. PWM Frequency Limit 35
Figure 11. Periodic Timer 50
Figure 12. Watchdog Timer 51
Figure 13. Capture Timer 52
Figure 14. D/A PWM Output 56
Figure 15. I^2C Pin Structure 60
Figure 16. PI Block 65
Figure 17. LOWPASS_FILT Block 68
Figure 18. LOWPASS_FILT frequency response 69
Figure 19. HIGHPASS_FILT Block 70
Figure 20. HIGHPASS_FILT frequency response 71
Figure 21. VECROT Block 72
Figure 22. VECROT vector interpretation 72
Figure 23. CLARK Block 73
Figure 24. CLARK vector interpretation 74
Figure 25. LIMIT Block 75
Figure 26. RAMP Block 76
Figure 27. ATAN Block 78
Figure 28. FUNCTION_BLOCK Block 79
Figure 29. FUNCTION_BLOCK Example 80
Figure 30. COMPARATOR Block 81
Figure 31. SWITCH Block 81
Figure 32. BIT_LATCH Block 82
Figure 33. PEAK_DETECT Block 83
Figure 34. TRANSITION Block 84
Figure 35. INTEGRAL2 Block 85
Figure 36. PFC_FFD Block 87
Figure 37. Block Diagram of PFC_FFD 87
Figure 38. DIFF Block 88

Figure 39. SUM Block 89
Figure 40. ACCUMULATOR Block 89
Figure 41. COUNTER Block 91
Figure 42. SHIFT Block 91
Figure 43. MUL_DIV Block 92
Figure 44. DIVIDE Block 94
Figure 45. NOT Block 95
Figure 46. NEGATE Block 96
Figure 47. AND Block 96
Figure 48. OR Block 97
Figure 49. XOR Block 97
Figure 50. SENSORLESS_FOC Block 98
Figure 51. SENSORLESS_FOC Block Diagram 103
Figure 52. Drive Control Modes 104
Figure 53. Flux and Current vector displacement 104
Figure 54. 2-Stage Parking 105
Figure 55. SINGLE_I_SHUNT Block 106
Figure 56. Single Shunt Current Sense Timing 107
Figure 57. Single Current Shunt Registers (TCntMin2Phs, TCntMin3Phs) 108
Figure 58. DC_BUS_VOLTAGE Block 109
Figure 59. A/D Interface Blocks 110
Figure 60. IRMCx312 A/D Converter Structure 111
Figure 61. Low Loss SVPWM Block 112
Figure 62. SVPWM Internal Block Diagram 114
Figure 63. Space Vector Diagram 115
Figure 64. Voltage Vector Rescaling 116
Figure 65. Deadtime Insertion 116
Figure 66. Three-Phase and Two-Phase Modulation 117
Figure 67. Types of Space Vector PWM 117
Figure 68. Guard Band Insertion 118
Figure 69. Bootstrap Pre-Charge Sequence 118
Figure 70. Timing of Bootstrap Capacitor Charging 119
Figure 71. Duty Ratio Control 119
Figure 72. FAULTS Block 120
Figure 73. MCE_FAULT Block 121
Figure 74. PFC_PWM Block 122
Figure 75. PFC_PWM Internal Block Diagram 123
Figure 76. Generation of PFC PWM output and ADC timing (PFC_sync_divider = 0) 124
Figure 77. Generation of PFC PWM output and ADC timing (PFC_sync_divider = 1) 125
Figure 78. PFC_SENSE Block 127
Figure 79. Detail scaling of PLL 145
Figure 80. Calculation of VdcRcp 163
Figure 81. PFCPhasing Example 171
Figure 82. Timing of Sync and MCE Computation 186
Figure 83. MCE Simulink Library 193
Figure 84. MCE Compiler Input Screen 196
Figure 85. MCE Compiler Results Example 198
Figure 86. The Host Register Summary Utility 199
Figure 87. The CustomMotPer Utility 200
Figure 88. MCE Design Hierarchy 202

LIST OF TABLES

Table 1. IRMCF300 Comparison Table 10
Table 2. Boot EEPROM Memory Map 18
Table 3. 8051 Instructions 23
Table 4. Special Function Register Memory Map 24
Table 5. Special Function Registers 26
Table 6. Discrete I/Os from Ports 1 – 5 29
Table 7. Interrupt Source Summary 39
Table 8. Interrupt Service Order 44
Table 9. SFRs for I^2C 57
Table 10. SFRs for SPI 57
Table 11. MCE Library Elements 63
Table 12. PI User Inputs and Outputs 67
Table 13. PI System Inputs and Outputs 67
Table 14. PI Execution Time 67
Table 15. LOWPASS_FILT User Inputs and Outputs 69
Table 16. LOWPASS_FILT Execution Time 69
Table 17. HIGHPASS_FILT User Inputs and Outputs 71
Table 18. HIGHPASS_FILT Execution Time 71
Table 19. VECROT Inputs and Outputs 72
Table 20. VECROT Execution Time 73
Table 21. CLARK Inputs and Outputs 74
Table 22. CLARK Execution Time 74
Table 23. LIMIT Inputs and Outputs 75
Table 24. LIMIT Execution Time 75
Table 25. RAMP User Inputs and Outputs 77
Table 26. RAMP Execution Time 77
Table 27. ATAN Inputs and Outputs 78
Table 28. ATAN Execution Time 78
Table 29. FUNCTION_BLOCK Inputs and Outputs 80
Table 30. FUNCTION_BLOCK Execution Time 80
Table 31. COMPARATOR Inputs and Outputs 81
Table 32. COMPARATOR Execution Time 81
Table 33. SWITCH Inputs and Outputs 82
Table 34. SWITCH Execution Time 82
Table 35. BIT_LATCH User Inputs and Outputs 83
Table 36. BIT_LATCH Execution Time 83
Table 37. PEAK_DETECT User Inputs and Outputs 83
Table 38. PEAK_DETECT Execution Time 84
Table 39. TRANSITION User Inputs and Outputs 85
Table 40. TRANSITION Execution Time 85
Table 41. INTEGRAL2 User Inputs and Outputs 86
Table 42. INTEGRAL2 Execution Time 86
Table 43. PFC_FFD Inputs and Outputs 87
Table 44. PFC_FFD Execution Time 87
Table 45. DIFF Inputs and Outputs 88
Table 46. DIFF Execution Time 88
Table 47. SUM Inputs and Outputs 89
Table 48. SUM Execution Time 89
Table 49. ACCUMULATOR User Inputs and Outputs 90
Table 50. ACCUMULATOR Execution Time 90
Table 51. COUNTER User Inputs and Outputs 91
Table 52. COUNTER Execution Time 91

Table 53. SHIFT Inputs and Outputs 92
Table 54. SHIFT Execution Time 92
Table 55. MUL_DIV Inputs and Outputs 93
Table 56. MUL_DIV Execution Time 93
Table 57. DIVIDE Inputs and Outputs 94
Table 58. DIVIDE Execution Time 95
Table 59. NOT Inputs and Outputs 95
Table 60. NOT Execution Time 95
Table 61. NEGATE Inputs and Outputs 96
Table 62. NEGATE Execution Time 96
Table 63. AND Inputs and Outputs 96
Table 64. AND Execution Time 97
Table 65. OR Inputs and Outputs 97
Table 66. OR Execution Time 97
Table 67. XOR Inputs and Outputs 97
Table 68. XOR Execution Time 97
Table 69. SENSORLESS_FOC Available Inputs and Outputs 100
Table 70. SINGLE_I_SHUNT Available Inputs and Outputs 106
Table 71. DC_BUS_VOLTAGE Outputs 109
Table 72. A_D Outputs 110
Table 73. SVPWM Available Inputs and Outputs 113
Table 74. FAULTS Block Outputs 120
Table 75. MCE_FAULT Block Inputs 121
Table 76. PFC_PWM Inputs and Outputs 122
Table 77. PFC_SENSE Outputs 127

1. Overview

The IRMCF300 series is a new family of International Rectifier integrated circuit devices primarily designed as one-chip solutions for inverter controlled appliance motor control applications. This includes the IRMCF312, IRMCF311, IRMCF343, IRMCF341 and IRMCF371. Throughout this document, the IRMCF300 will refer to any one of the five digital control IC's in the IRMCF300 family.

Unlike a traditional microcontroller or DSP, the IRMCF300 provides a built-in closed loop sensorless control algorithm using the unique Motion Control Engine (MCE™) for permanent magnet motors. The MCE™ consists of control elements, motion peripherals, a dedicated motion control sequencer and a dual port RAM to map internal signal nodes. The IRMCF300 also employs a unique single shunt current reconstruction circuit to eliminate additional analog/digital circuitry and enables a direct shunt resistor interface to the IC. Some of the IRMCF300 has a digital PFC control for both boost and bridgeless mode. Motion control programming is achieved by using a dedicated graphical compiler integrated into the MATLAB/Simulink™ development environment. Sequencing, user interface, host communication, and upper layer control tasks can be implemented in the built-in 8051 high-speed 8-bit microcontroller. The 8051 microcontroller is equipped with a JTAG port to facilitate emulation and debugging tools. Figure 1 shows a typical application schematic using the IRMCF312. Two permanent magnet motors and PFC can be controlled by a single chip without requiring motor position sensors.

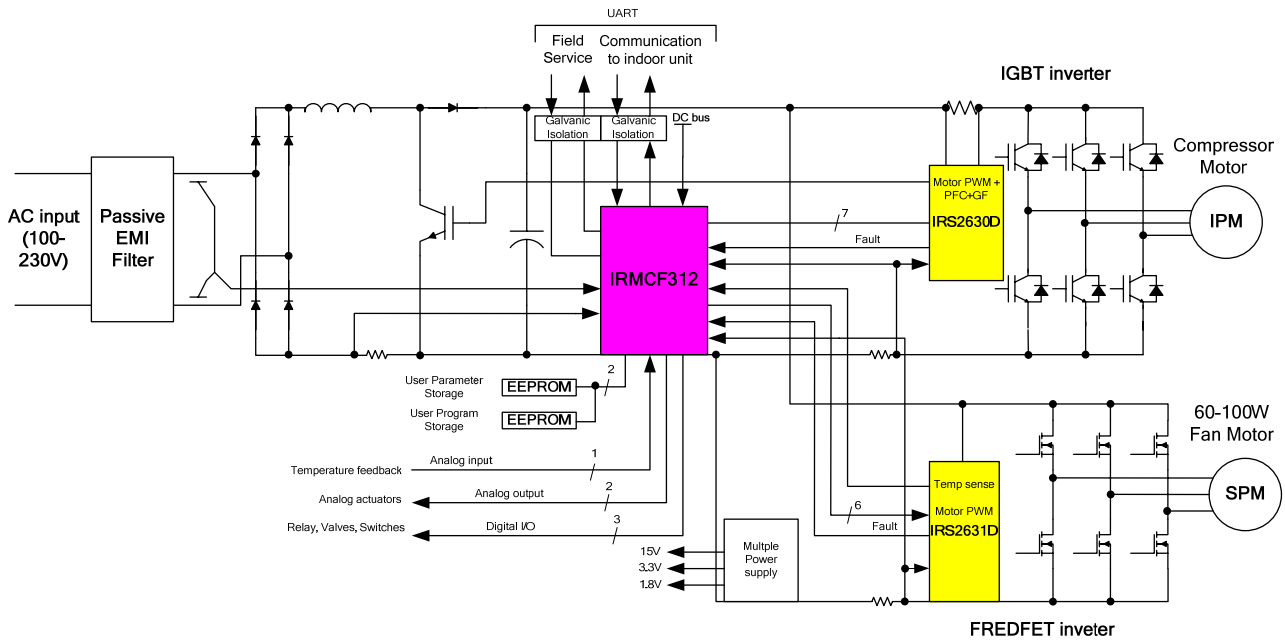


Figure 1. Typical Application Block Diagram Using IRMCF312

IRMCF300 contains 48K bytes of RAM for instruction storage, which can be loaded from external EEPROM for 8051 program execution. The IRMCF300 is intended for development purposes. For high volume production, the IRMCK300 contains one-time-programmable (OTP) memory in place of program RAM. IRMCK300 also includes 8K bytes of data RAM. The "F" and "K" versions of each part have identical pin configurations to facilitate PC board layout and transition to mass production.
Table 1 shows differences among IRMCF300 IC's.

	312	**311**	**343**	**341**	**371**
Motor Control	2	2	1	1	1
PFC	Yes	Yes	Yes	No	No
Package	QFP100	QFP64	QFP64	QFP64	QFP48
8051 Program Memory	48KByte	48KByte	48KByte	48KByte	48KByte
RAM	8KByte	8KByte	8KByte	8KByte	8KByte
I/O (max)	36	14	23	24	13
A/D Channels Total /Buffered	11 / 5	6 / 4	5 / 3	8 / 1	4 / 1
D/A Channels	3	2	3	3	1
UART	2	2	1	1	1
General Purpose Counter Pin (T0, T1, T2)	3	-	1	2	1

Table 1. IRMCF300 Comparison Table

A block diagram of the IRMCF312, a superset of other IRMCF300 IC, is shown in Figure 2. For details of the pin out and typical application connections, please refer to the datasheets of each IC's.

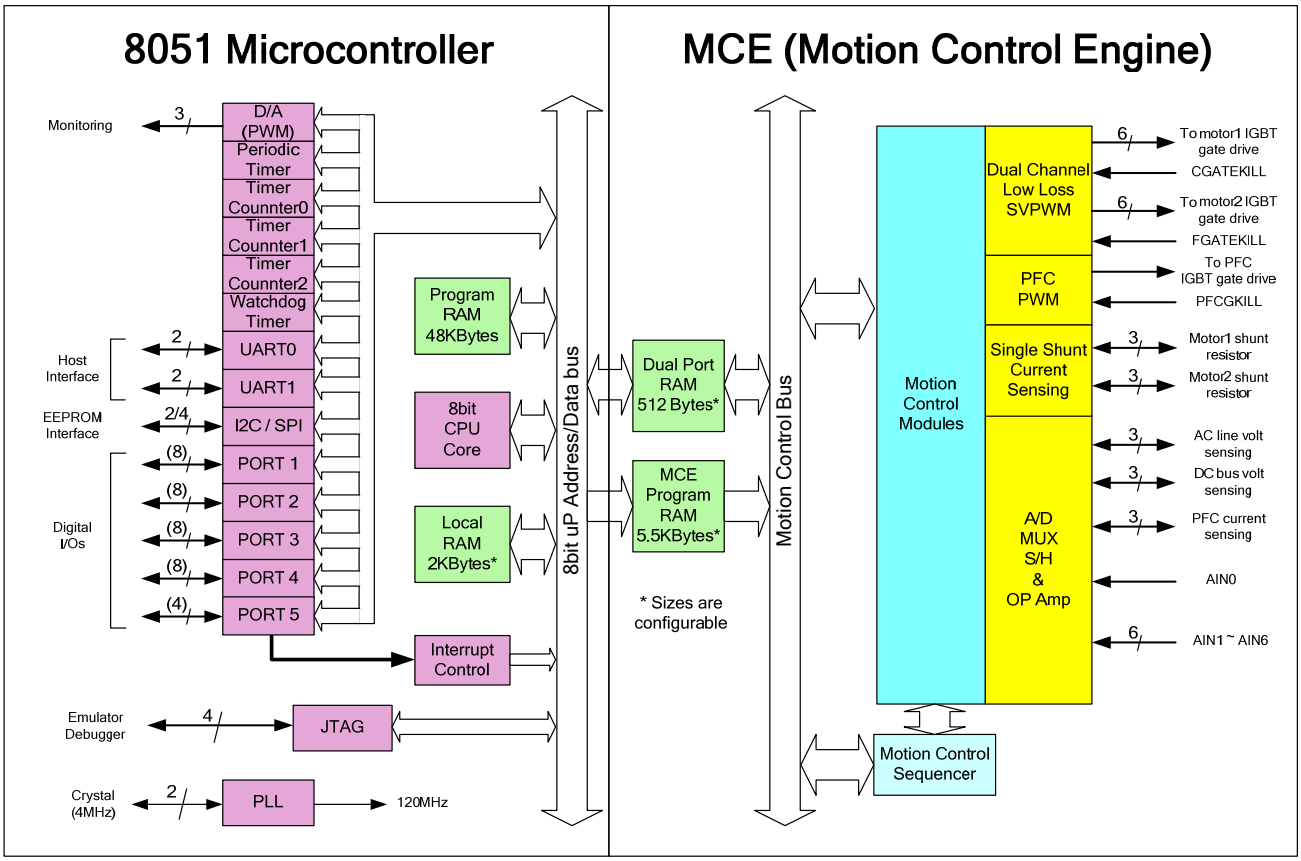


Figure 2. IRMCF312 Internal Block Diagram

1.1 System Components

The IRMCF300 can be divided into four main components, shown color-coded in Figure 2. The components are:

- The 8051 microcontroller, shown at the far left in purple
- RAM for program and data storage, shown at the center of the diagram in green
- The Motion Control Engine (MCE), shown at right in blue
- Hardware interface or "motion peripheral" modules, shown at the far right in yellow

1.2 Memory Map

Figure 3 shows the memory map of the IRMCF300 RAM. The 48K bytes of program RAM shown in the figure (at 8051 external RAM addresses 0x0000 – 0xBFFF) are present only in the IRMCF300 for software development purposes. In the IRMC**K**300, the RAM is replaced with one-time-programmable (OTP) memory in the same address range, except with 56K available for the 8051 program. Data RAM for the 8051 is located at external RAM addresses 0xF800 – 0xFFFF. The shared RAM (at 8051 external RAM addresses 0xE000 – 0xF7FF) is described more fully in Section 5.1.

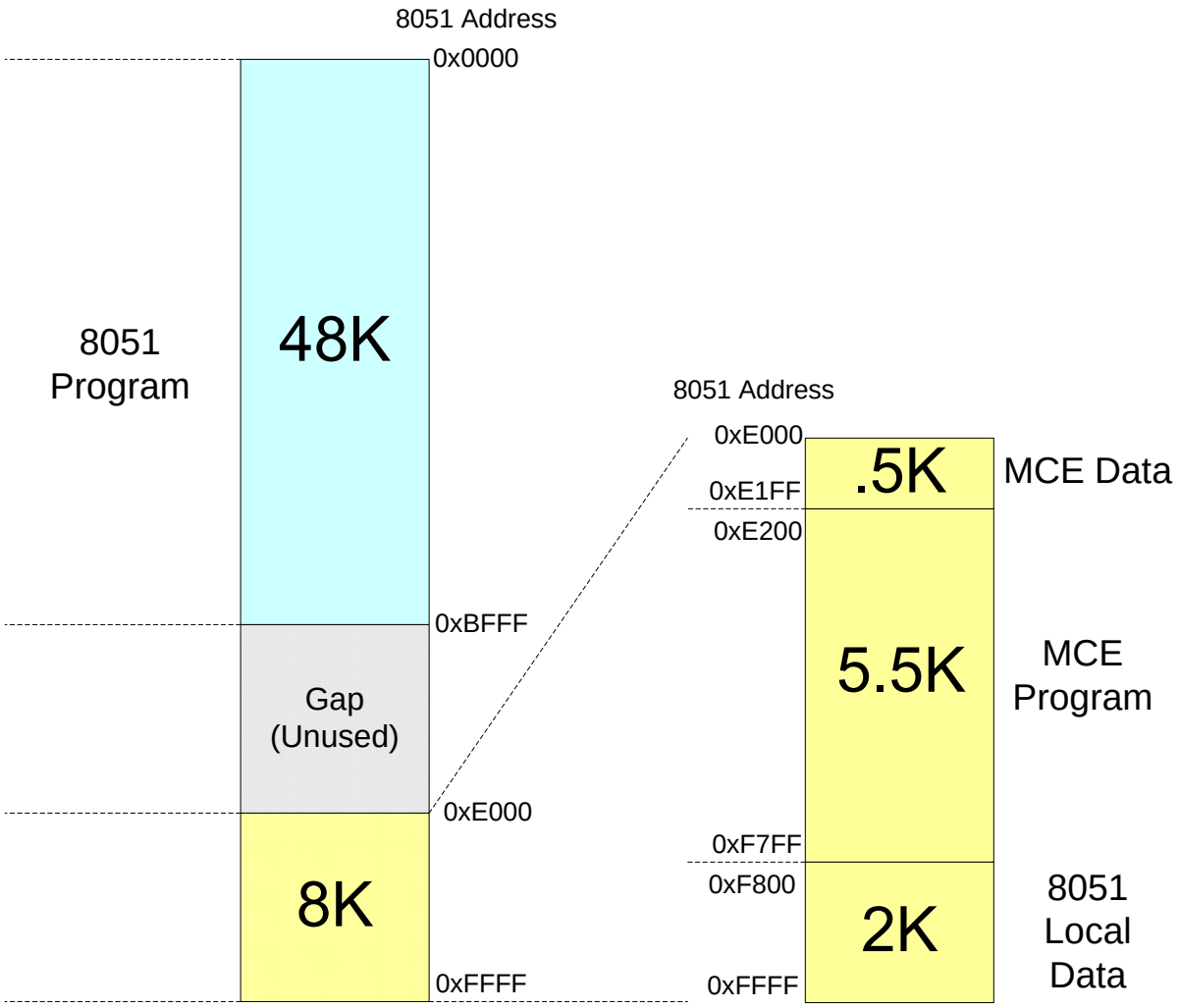


Figure 3. Memory Map of IRMCF300

1.3 Byte Ordering

Byte ordering refers to the convention used to store 16-bit and 32-bit values in memory using a processor, such as the 8051, that has a native addressing mode of 8 bits. The two standard byte ordering conventions are "big endian" or "Motorola" byte ordering and "little endian" or "Intel" byte ordering.

In big endian byte ordering, the "big end" of a value is stored first. That is, the high order byte is stored at the lowest memory address and the low-order byte is stored at the highest memory address. In little endian byte ordering, the "little end" is stored first, with the low-order byte at the lowest memory address.

For example, suppose the 16-bit value 0x2345 is to be stored in memory at address 0x1000. Using big endian byte ordering, 0x23 is stored at address 0x1000 and 0x45 is stored at address 0x1001. Using little endian byte ordering, 0x45 is stored at address 0x1000 and 0x23 is stored at address 0x1001.

The table below shows how the value 0x456789AB would be stored at address 0x1000 using each of the byte ordering conventions.

Address	Big Endian	Little Endian
0x1000	0x45	0xAB
0x1001	0x67	0x89
0x1002	0x89	0x67
0x1003	0xAB	0x45

The Keil compiler used for 8051 software development generates code that uses big endian byte ordering to store 16-bit and 32-bit values in memory.

The MCE is a 16-bit processor and uses little endian byte ordering for data storage. The smallest unit of data storage on the MCE processor is 16 bits (it cannot access a single byte in memory). The shared RAM used to exchange information between the 8051 and MCE processors is 8-bit addressable to the 8051, but 16-bit addressable to the MCE.

All data shared between the 8051 and MCE processors is expected to be in little endian byte ordering. This means that the 8051 must swap bytes before writing to shared RAM and swap bytes after reading from shared RAM. The table below shows how the value 0x456789AB would be stored at address 0xE200 in 8051 RAM, which corresponds to address 0x0100 in MCE RAM. Note that the 8051 reads and writes a byte at a time, but the MCE always accesses the memory a word (16 bits) at a time.

8051 Address	8051 Bytes	MCE Address	MCE Words
0xE200	0xAB	0x0100	0x89AB
0xE201	0x89		
0xE202	0x67	0x0101	0x4567
0xE203	0x45		

2 Reset Mechanism and Boot Process

A power on reset does not require any assertion of reset signal from an external circuit. The IRMCF300 contains an analog power on reset (POR) circuit which issues reset to all internal circuits. Therefore no filter other than a pull-up resistor is required at the reset pin. The reset pin is bidirectional and becomes output when one of the following conditions occurs.

1) The watchdog timer times out.
2) Under voltage lockout (UVCC) circuit detects low voltage on AVDD (1.8V).
3) Power on reset (POR) circuit becomes alive at the power up.

Among these, cases (2) and (3) pull down RESET for 2048 periods of crystal clock (XCLK). When the watchdog timeout occurs, RESET low assertion time becomes about 31 µsec.

When the RESET input is asserted from an external source, minimum 10 µsec low level assertion pulse is required to ensure the internal reset.
Figure 4 shows the Reset Module, which contains two analog circuits, namely Under Voltage Lockout (UVCC) and Power-On Reset (POR), and a 12-bit ripple counter to stretch the Reset pulse width. This module is one of the blocks in Figure 5 and Figure 6.

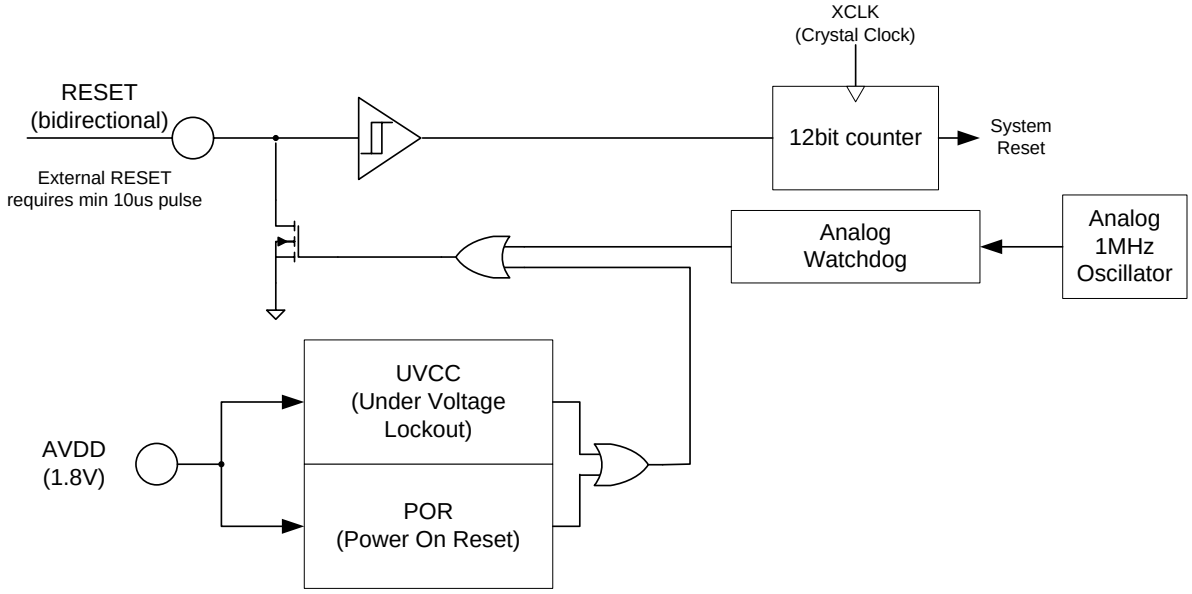


Figure 4. Reset Module

The reset and boot processes are closely tied together in the IRMCF300. The boot process is automatically accomplished following a proper reset sequence, which is triggered by power on or external RESET. Therefore, a user application cannot intervene during the reset and boot process. The main task of the boot process is to copy the user application program stored in an external serial EEPROM to program RAM, initialize the program counter and transfer control to the 8051 CPU. The block diagram of the reset and boot process is shown in Figure 5.

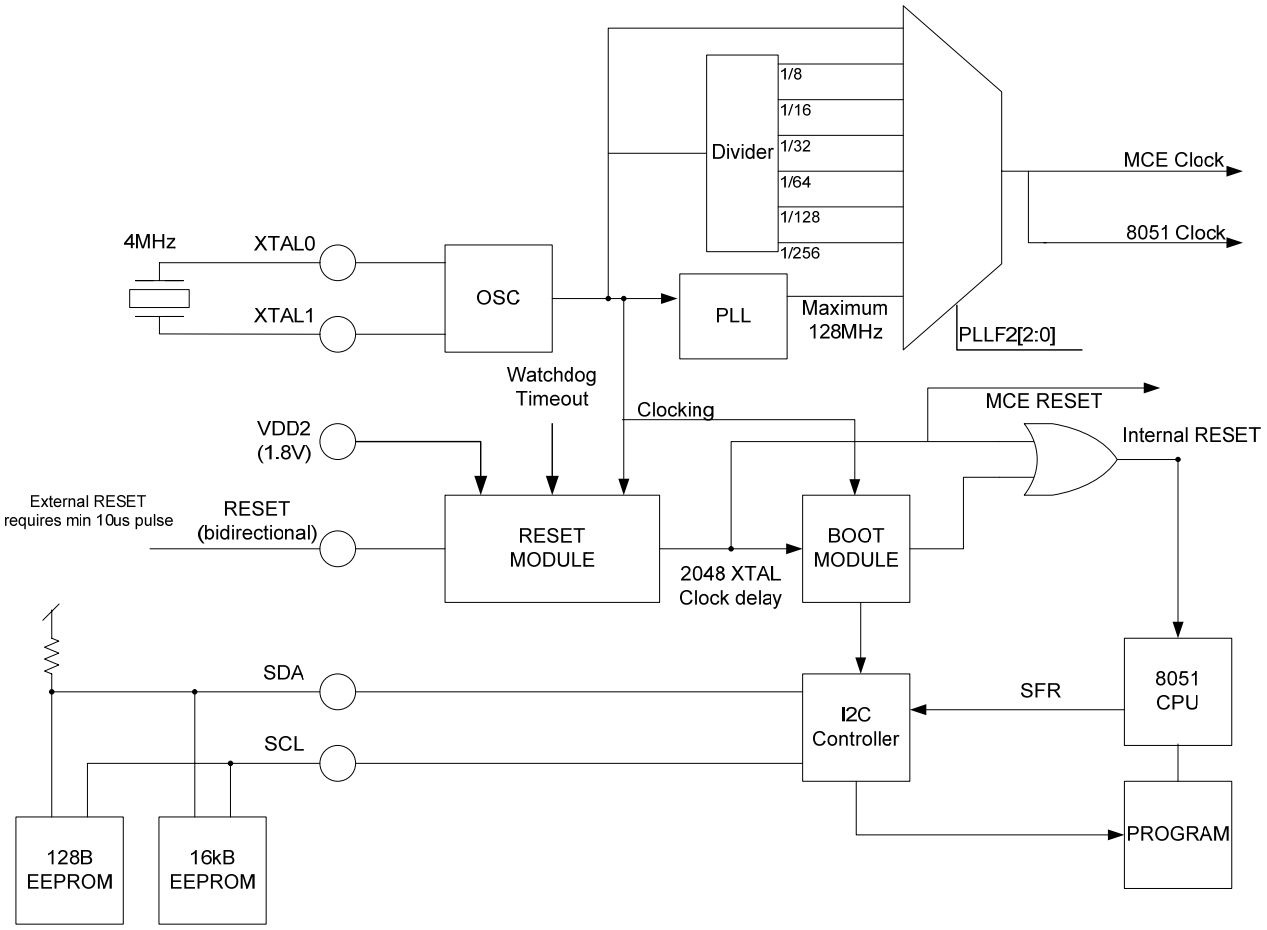


Figure 5. Reset and Boot Process for F-version

A reset of the IRMCF300 is accomplished by asserting low on the RESET input pin for a minimum of 10 µsec. The RESET is a Schmitt trigger type input with hysteresis to avoid any multiple triggers to the system. Once the reset is recognized in the system, the reset module counts up to 2048 clocks at the crystal frequency (i.e. 4 MHz, 512 µsec) to ensure that the internal PLL becomes stable for generation of the internal system clock. When this waiting period is complete, the boot module begins copying the user program from external EEPROM to program RAM via the I^2C port. The boot module is clocked off of the external oscillator (which is 4Mhz in IR's Reference Kits). The time to complete the copy process depends on the size of the user program. The following example shows the time required to copy 48Kbytes of 8051 program and 6Kbytes of MCE program.

Total user program to be copied: (48 + 6) * 1024 bytes
Number of bit periods to transfer one byte: 9 bits
I^2C clock speed: 333 kHz, for a transfer time of 3 µsec per bit

Total transfer time = 54 * 1024 * 9 * 3 µsec = 1.5 sec

The boot module holds the internal Reset active while data transfer takes place, then it releases the internal Reset and I^2C port control upon completion of the copy process. Immediately after the copy process completes, the 8051 application program begins execution. The IC pin P1.3 determines whether the controller performs an I2C or SPI boot load. Pull up the pin to VDD1 (3.3V) for I2C boot load.

The time to complete the copy process varies depending on the actual size of the application program and the I^2C clock speed, which can be modified to accommodate various types of I^2C EEPROM devices.

The boot process and clock generation are somewhat different in the K-version, as shown in Figure 6. Since the K-version contains OTP memory, it does not download from the EEPROM. Instead, the highest address 8Kbytes (0xE000 to 0xFFFF) of the OTP are copied into the IRMCK300 IC's RAM. Also, the K-version provides for the 8051 and MCE to have different clock rates since the OTP program memory is limited to 33MHz.

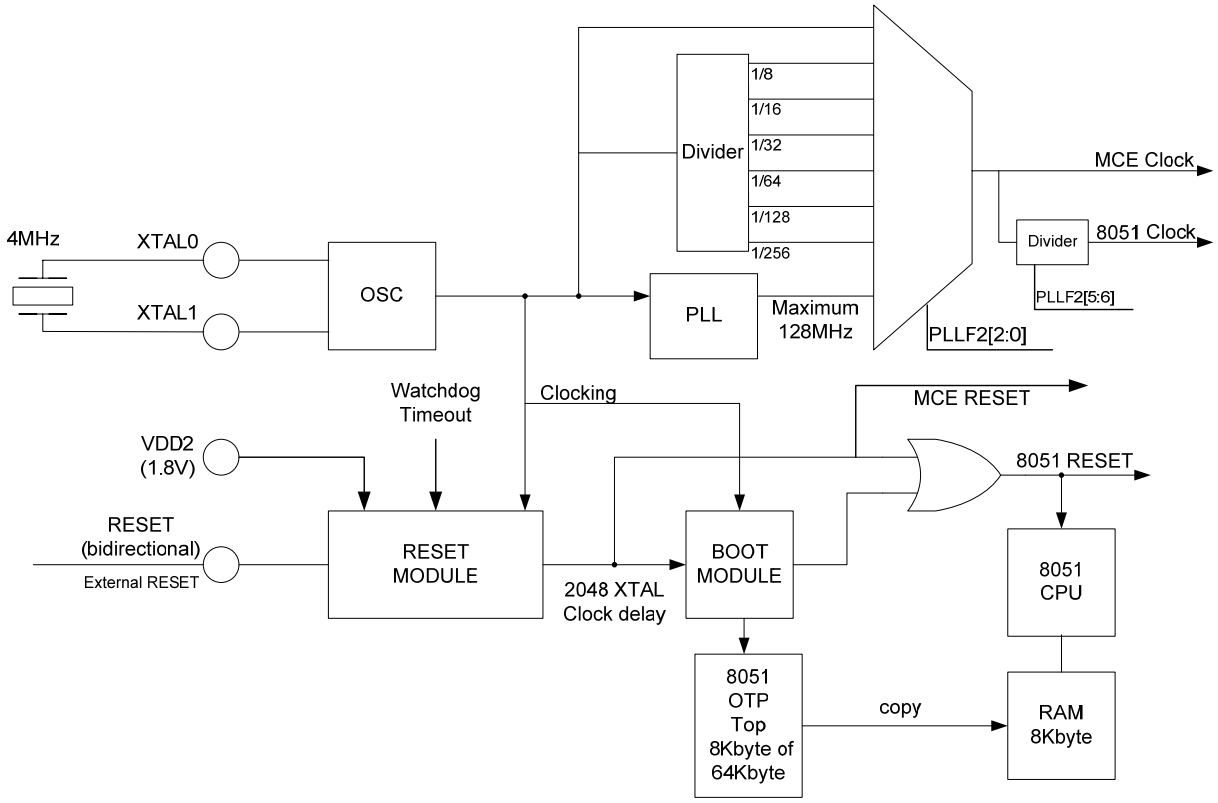


Figure 6. Reset and Boot Process for K-version.

In the K-version, the boot loading from the OTP to the RAM takes 10 clock cycles per byte, with the clock provide by the external oscillator. From this the boot time can be calculated (assuming a 4Mhz clock):

8192 bytes * 10 clocks/byte / 4Mhz clock freq = 20.48 ms

2.1 Power Sequencing at Startup

This section defines the sequence of power supply voltage ramps that must be followed when starting up the IR IRMCx3xx controller ICs. The IRMCS3041 (IRMCF341 reference demo board) is presented as reference for the schematic and the scope pictures, but the same applies to the whole family of ICs (IRMCF/K371, 341, 343, 311 and 312).

The IRMCF341 needs two different voltages power supplies:

- 1.8V used for the PLL, A/D converter and digital core;
- 3.3V used for Input/Output pins.

The correct sequence to have good startup of the IRMCx3xx is obtained when:
1.8V power supply line voltage lags the 3.3V power supply line voltage and the actual value of the voltage of the 1.8V line is always lower than the actual value of the 3.3V line during ramp up.

An example of the correct power sequencing is shown in Figure 7. The green trace is the 1.8V power supply voltage and the yellow trace is the 3.3V power supply voltage.

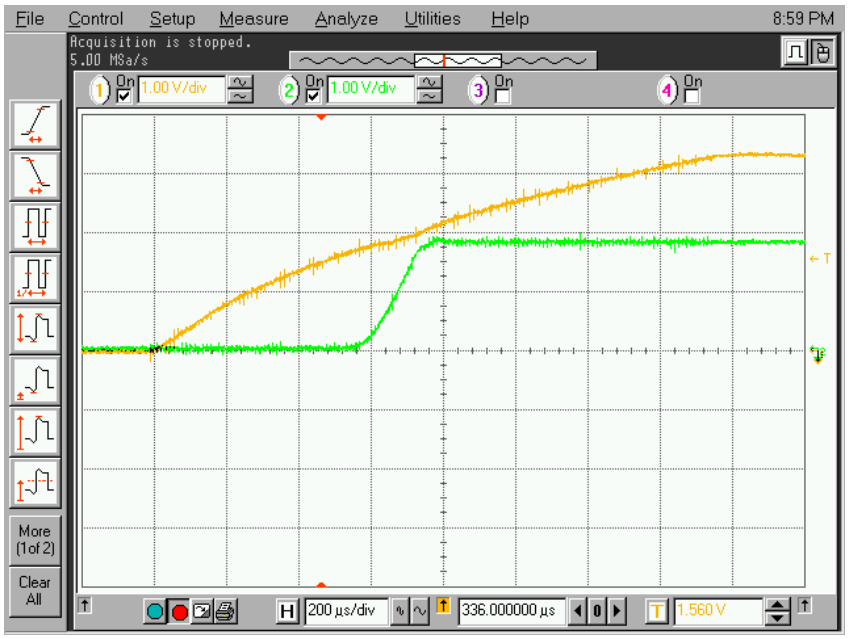


Figure 7. Exact power sequence for IRMCF341 – Obtained in IRMCS3041 reference demo board

The traces in Fig.1 have been obtained using the IRMCS3041 reference demo board, where the 1.8V power supply is obtained directly from 3.3V power supply with a low dropout linear regulator. The detail of the IRMCS3041 schematic is shown in Fig.2.

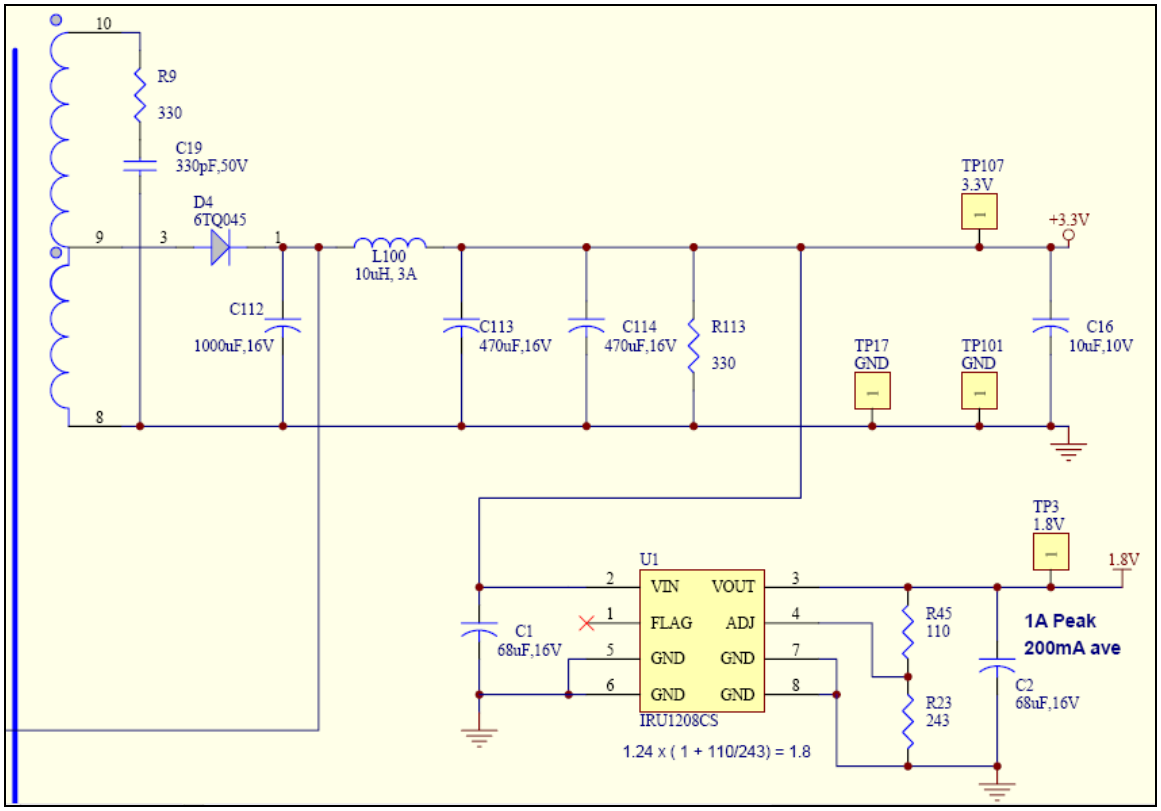


Figure 8. Detail of power supply circuit in IRMCS3041 reference demo board

If the power sequencing is guaranteed by design, the IRMCx3xx start up is successful and the proper operation is obtained.

If the sequence is not followed, a proper IRMCx3xx startup is not guaranteed. However in case of an incorrect power supply ramp up sequence, the IRMCx3xx functionality can be recovered by asserting the RESET line and then releasing it. If the right power sequence cannot be guaranteed by design, a dual voltage supervisor IC from a third party can easily solve the problem.

2.2 EEPROM Data Format

To support variable-length application programs and variable speed of the I^2C EEPROM transfer, the first four bytes of EEPROM and the last two bytes in EEPROM are dedicated for the boot process. The EEPROM data is divided into 256-byte pages with an address byte (page number) for each page. This allows the EEPROM to store any number of program segments. The boot module copies each page to its specified destination address in RAM independently. There is no requirement for pages to be stored in sequential or ascending order.

The page number is the high-order byte of the 16-bit RAM destination address. For example, page number 0x34 is copied to address 0x3400. All pages except the last must contain exactly 256 bytes of data. The last page may contain fewer than 256 bytes, with the total byte count in the EEPROM header determining the number of bytes in that page.

If the application program consists of n pages and *len* is the number of bytes of data in the last page, the total byte count stored in the first two bytes of EEPROM can be calculated as follows:

$$\text{Byte count} = 4 + (257 * (n - 1)) + 1 + len + 2$$

The EEPROM header is four bytes. Each page before the last is 257 bytes (one byte page number and 256 bytes of data). The last page has a 1-byte page number and *len* data bytes, and the 2-byte checksum is included at the end.

Table 2 shows the EEPROM data format. The four-byte header, two-byte checksum and the page number preceding each 256-byte page are not copied to RAM.

Offset byte address	Data	Note
0	Byte Count Low	Total byte count including the four header bytes, all of the 8051 and MCE program, and two checksum bytes
1	Byte Count High	
2	Baud Rate Low	12-bit baud rate time constant
3	Baud Rate High	
4	First page number	8051 and MCE program
5	First page starting data	
…	…	
260	First page ending data	
261	Second page number	
262	Second page starting data	
…	…	
517	Second page ending data	
…	…	
Last – 1	Checksum Low	16-bit checksum
Last	Checksum High	

Table 2. Boot EEPROM Memory Map

The first two bytes of the header are a 16-bit count value representing the total number of bytes in EEPROM including the four header bytes, the application program pages and the last two bytes of checksum. The remaining two bytes of the header have the I^2C baud rate value in the lower 12 bits. Given the crystal clock (XClk) and the desired baud rate (f_{SCL}), use the formula below to set the baud rate time constant to go into the EEPROM header. MCEDesigner sets this value to 0x04. (Note that the power-up value of the baud rate time constant is 0x50, so that the first four bytes have a baud rate of 24.7 kHz.)

$$BaudRateConst = \left(\frac{XClk}{2 \cdot f_{SCL}} - 1 \right)$$

The 16-bit checksum follows the last byte of application program data, low-order byte first. The checksum is calculated on all preceding bytes in EEPROM, including the four-byte header. To calculate the checksum, sum all the bytes (a byte at a time, ignoring any overflow beyond 16 bits) and then negate (two's complement) the resulting value.

When the IRMCF300 validates the checksum by adding all bytes including the first four bytes and the last two bytes of checksum itself, the 16-bit result should be zero. If it is not zero, the system halts with a checksum error and does not transfer control to the 8051 application program. There is no external indication of the error and no further operations are performed until the device is reset. To diagnose such an issue, check the I2C bus during boot (right after a reset). The SCL pin should show clock pulses, as data is transferred from the EEPROM to the control IC, which should cease after the transfer. This indicates that the IC is not damaged and is working properly. Next, use the JTAG interface to reprogram the EEPROM with a known good image. If this image operates correctly, then there is strong likelihood that there is a checksum error with the original image.

3 8051 Microcontroller

This section describes IRMCF300 features and functions that are specific to the 8051 microcontroller. The interface between the 8051 and the MCE is covered in Section 5.

The instruction set and basic operation of the IRMCF300 8051 microcontroller is consistent with the standard Intel 8051 processor. A number of peripheral devices and special functions have been added to customize the operation for the intended application.

3.1 Instruction Set

The instructions of the 8051 microcontroller are 1, 2 or 3 bytes long as listed in the 'Bytes' column below. Each instruction takes either 1, 2 or 4 machine cycles to execute as listed in the following table. 1 machine cycle comprises 2 SYSCLK cycles.

Table 3 lists the 8051 instructions. In the table, an entry such as E8-EF indicates a continuous block of hex opcodes used for 8 different registers, the register numbers of which are defined by the lowest three bits of the corresponding code. Non-continuous blocks of codes, shown as 11→F1 (for example), are used for absolute jumps and calls, with the top 3 bits of the code being used to store the top three bits of the destination address.

The CJNE instructions use the abbreviation #d for immediate data; other instructions use #data.

ARITHMETIC				
Mnemonic	**Description**	**Bytes**	**Cycles**	**Hex code**
ADD A,Rn	Add register to A	1	1	28-2F
ADD A,dir	Add direct byte to A	2	1	25
ADD A,@Ri	Add indirect memory to A	1	1	26-27
ADD A,#data	Add immediate to A	2	1	24
ADDC A,Rn	Add register to A with carry	1	1	38-3F
ADDC A,dir	Add direct byte to A with carry	2	1	35
ADDC A,@Ri	Add indirect memory to A with carry	1	1	36-37
ADDC A,#data	Add immediate to A with carry	2	1	34
SUBB A,Rn	Subtract register from A with borrow	1	1	98-9F
SUBB A,dir	Subtract direct byte from A with borrow	2	1	95
SUBB A,@Ri	Subtract indirect memory from A with borrow	1	1	96-97
SUBB A,#data	Subtract immediate from A with borrow	2	1	94
INC A	Increment A	1	1	04
INC Rn	Increment register	1	1	08-0F
INC dir	Increment direct byte	2	1	05
INC @Ri	Increment indirect memory	1	1	06-07
DEC A	Decrement A	1	1	14
DEC Rn	Decrement register	1	1	18-1F
DEC dir	Decrement direct byte	2	1	15
DEC @Ri	Decrement indirect memory	1	1	16-17
INC DPTR	Increment data pointer	1	2	A3
MUL AB	Multiply A by B	1	4	A4
DIV AB	Divide A by B	1	4	84
DA A	Decimal Adjust A	1	1	D4

LOGICAL				
Mnemonic	**Description**	**Bytes**	**Cycles**	**Hex code**
ANL A,Rn	AND register to A	1	1	58-5F
ANL A,dir	AND direct byte to A	2	1	55
ANL A,@Ri	AND indirect memory to A	1	1	56-57
ANL A,#data	AND immediate to A	2	1	54
ANL dir,A	AND A to direct byte	2	1	52
ANL dir,#data	AND immediate to direct byte	3	2	53
ORL A,Rn	OR register to A	1	1	48-4F
ORL A,dir	OR direct byte to A	2	1	45
ORL A,@Ri	OR indirect memory to A	1	1	46-47
ORL A,#data	OR immediate to A	2	1	44
ORL dir,A	OR A to direct byte	2	1	42
ORL dir,#data	OR immediate to direct byte	3	2	43
XRL A,Rn	Exclusive-OR register to A	1	1	68-6F
XRL A,dir	Exclusive-OR direct byte to A	2	1	65
XRL A,@Ri	Exclusive-OR indirect memory to A	1	1	66-67
XRL A,#data	Exclusive-OR immediate to A	2	1	64
XRL dir,A	Exclusive-OR A to direct byte	2	1	62
XRL dir,#data	Exclusive-OR immediate to direct byte	3	2	63
CLR A	Clear A	1	1	E4
CPL A	Complement A	1	1	F4
SWAP A	Swap Nibbles of A	1	1	C4
RL A	Rotate A left	1	1	23
RLC A	Rotate A left through carry	1	1	33
RR A	Rotate A right	1	1	03
RRC A	Rotate A right through carry	1	1	13

DATA TRANSFER				
Mnemonic	**Description**	**Bytes**	**Cycles**	**Hex code**
MOV A,Rn	Move register to A	1	1	E8-EF
MOV A,dir	Move direct byte to A	2	1	E5
MOV A,@Ri	Move indirect memory to A	1	1	E6-E7
MOV A,#data	Move immediate to A	2	1	74
MOV Rn,A	Move A to register	1	1	F8-FF
MOV Rn,dir	Move direct byte to register	2	2	A8-AF
MOV Rn,#data	Move immediate to register	2	1	78-7F
MOV dir,A	Move A to direct byte	2	1	F5
MOV dir,Rn	Move register to direct byte	2	2	88-8F
MOV dir,dir	Move direct byte to direct byte	3	2	85
MOV dir,@Ri	Move indirect memory to direct byte	2	2	86-87
MOV dir,#data	Move immediate to direct byte	3	2	75
MOV @Ri,A	Move A to indirect memory	1	1	F6-F7
MOV @Ri,dir	Move direct byte to indirect memory	2	2	A6-A7
MOV @Ri,#data	Move immediate to indirect memory	2	1	76-77
MOV DPTR,#data	Move immediate to data pointer	3	2	90
MOVC A,@A+DPTR	Move code byte relative DPTR to A	1	2	93
MOVC A,@A+PC	Move code byte relative PC to A	1	2	83
MOVX A,@Ri	Move external data(A8) to A	1	2	E2-E3
MOVX A,@DPTR	Move external data(A16) to A	1	2	E0

DATA TRANSFER				
MOVX @Ri,A	Move A to external data(A8)	1	2	F2-F3
MOVX @DPTR,A	Move A to external data(A16)	1	2	F0
PUSH dir	Push direct byte onto stack	2	2	C0
POP dir	Pop direct byte from stack	2	2	D0
XCH A,Rn	Exchange A and register	1	1	C8-CF
XCH A,dir	Exchange A and direct byte	2	1	C5
XCH A,@Ri	Exchange A and indirect memory	1	1	C6-C7
XCHD A,@Ri	Exchange A and indirect memory nibble	1	1	D6-D7

BOOLEAN				
Mnemonic	**Description**	**Bytes**	**Cycles**	**Hex code**
CLR C	Clear carry	1	1	C3
CLR bit	Clear direct bit	2	1	C2
SETB C	Set carry	1	1	D3
SETB bit	Set direct bit	2	1	D2
CPL C	Complement carry	1	1	B3
CPL bit	Complement direct bit	2	1	B2
ANL C,bit	AND direct bit to carry	2	2	82
ANL C,/bit	AND direct bit inverse to carry	2	2	B0
ORL C,bit	OR direct bit to carry	2	2	72
ORL C,/bit	OR direct bit inverse to carry	2	2	A0
MOV C,bit	Move direct bit to carry	2	1	A2
MOV bit,C	Move carry to direct bit	2	2	92

BRANCHING				
Mnemonic	**Description**	**Bytes**	**Cycles**	**Hex code**
ACALL addr 11	Absolute jump to subroutine	2	2	11→F1
LCALL addr 16	Long jump to subroutine	3	2	12
RET	Return from subroutine	1	2	22
RETI	Return from interrupt	1	2	32
AJMP addr 11	Absolute jump unconditional	2	2	01→E1
LJMP addr 16	Long jump unconditional	3	2	02
SJMP rel	Short jump (relative address)	2	2	80
JC rel	Jump on carry = 1	2	2	40
JNC rel	Jump on carry = 0	2	2	50
JB bit,rel	Jump on direct bit = 1	3	2	20
JNB bit,rel	Jump on direct bit = 0	3	2	30
JBC bit,rel	Jump on direct bit = 1 and clear	3	2	10
JMP @A+DPTR	Jump indirect relative DPTR	1	2	73
JZ rel	Jump on accumulator = 0	2	2	60
JNZ rel	Jump on accumulator ≠ 0	2	2	70
CJNE A,dir,rel	Compare A,direct jne relative	3	2	B5
CJNE A,#d,rel	Compare A,immediate jne relative	3	2	B4
CJNE Rn,#d,rel	Compare register, immediate jne relative	3	2	B8-BF
CJNE @Ri,#d,rel	Compare indirect, immediate jne relative	3	2	B6-B7
DJNZ Rn,rel	Decrement register, jnz relative	2	2	D8-DF
DJNZ dir,rel	Decrement direct byte, jnz relative	3	2	D5

MISCELLANEOUS				
Mnemonic	**Description**	**Bytes**	**Cycles**	**Hex code**
NOP	No operation	1	1	00

ADDITIONAL INSTRUCTIONS (selected through EO[4])				
Mnemonic	**Description**	**Bytes**	**Cycles**	**Hex code**
MOVC @(DPTR++),A	IRMCF300-specific instruction supporting software download into program memory (see Section 3.2.4)	1	2	A5
TRAP	Software break command (IRMCF300-specific : see Section 3.2.4)	1	1	A5

Table 3. 8051 Instructions

3.2 Special Function Registers

All I/O, timer/counter, D/A PWM, UARTs and some MCE operations are accessed via Special Function Registers (SFRs). These registers occupy direct Internal Data Memory space locations in the range 80h to FFh.

Table 4 shows a summary of the SFR memory map and identifies the bit-addressable registers. Table 5 lists each register individually and shows its value on reset.

Some general SFRs are described in detail later in this section. Others associated with specific functions such as UARTs, timers and the MCE interface are defined later in the document.

	0(8)	1(9)	2(A)	3(B)	4(C)	5(D)	6(E)	7(F)
80	P0	SP	DPL	DPH				
88	TCON	TMOD	TL0	TL1	TH0	TH1		
90	P1	P1DIR			U0CTL	U0STAT	U0BUF	U0BR
98					U1CTL	U1STAT	U1BUF	U1BR
A0	P2	P2DIR	EO	DAD3	WDTL	WDTH	TL3	TH3
A8	IE	PSCL	IOCON0	IOCON1		DAD0	DAD1	DAD2
B0	P3	P3DIR	HWCFG	RSTRSN	CLASTL	CLASTH	CPREVL	CPREVH
B8	IP	I2CAL	I2CAH	I2CTD	I2CCD	I2CDA	I2CBC	I2CNF
C0	IS	P4IS	P4IM0	P4IM1				
C8	T2CON		RCP2L	RCP2H	TL2	TH2		
D0	PSW	MCEAD0	MCEAD1	MCEAD2	MCEAD3	PLLF0	PLLF1	PLLF2
D8		MCEBL	MCEBH	PLLF3	HWREV	MCESS	SYNCS	
E0	ACC	P4DIR	P4	MCESL	MCESH	STOPS	P5DIR	P5
E8	IE1	MCECD0	MCECD1	MCECD2	MCECD3		MCEAAH	MCEAAL
F0	B							
F8	IP1							
	Bit Addressable							

Table 4. Special Function Register Memory Map

Address (hex)	Label	Description	Reset Value (hex)	Bit Address-able	Notes	Document Section
80	P0	Port 0	FF	*	Not used	
81	SP	Stack Pointer	07			3.2.1
82	DPL	Data Pointer Low Byte	00			3.2.1
83	DPH	Data Pointer High Byte	00		3.2.1	3.2.1
88	TCON	Timer/Counter Control	00	*		3.4.2
89	TMOD	Timer/Counter Mode Control	00			3.4.2
8A	TL0	Timer/Counter 0 Low Byte	00			3.4.2
8B	TL1	Timer/Counter 1 Low Byte	00			3.4.2
8C	TH0	Timer/Counter 0 High Byte	00			3.4.2
8D	TH1	Timer/Counter 1 High Byte	00			3.4.2
90	P1	Port 1	FF	*		3.2.2
91	P1DIR	Port 1 Direction Register	00			3.2.2
94	U0CTL	UART 0 Control Register	00			3.5
95	U0STAT	UART 0 Status Register	00			3.5
96	U0BUF	UART 0 Buffer Register	Undefined			3.5
97	U0BR	UART 0 Buadrate	30			3.5

Address (hex)	Label	Description	Reset Value (hex)	Bit Address-able	Notes	Document Section
9C	U1CTL	UART 1 Control Register	00			3.5
9D	U1STAT	UART 1 Status Register	00			3.5
9E	U1BUF	UART 1 Buffer Register	Undefined			3.5
9F	U1BR	UART 1 Buadrate	30			3.5
A0	P2	Port 2	FF	*		3.2.2
A1	P2DIR	Port 2 Direction Register	00			3.2.2
A2	EO	Extended Operation Register	00			3.2.4
A3	DAD3	D/A PWM 3 Data	00		Internal use	
A4	WDTL	Watchdog Timer Limit Low	00			3.4.4
A5	WDTH	Watchdog Timer Limit High	00			3.4.4
A6	TL3	Timer 3 Low	00			3.4.3
A7	TH3	Timer 3 High	00			3.4.3
A8	IE	Interrupt Enable Register 0	00	*		3.3.4
A9	PSCL	Prescaler Control	00			3.4.1
AA	IOCON0	I/O Control Register 0	00			3.2.2
AB	IOCON1	I/O Control Register 1	00			3.2.2
AD	DAD0	D/A PWM 0 Data	00			3.6
AE	DAD1	D/A PWM 1 Data	00			3.6
AF	DAD2	D/A PWM 2 Data	00			3.6
B0	P3	Port 3	FF	*		3.2.2
B1	P3DIR	Port 3 Direction Register	00			3.2.2
B2	HWCFG	Hardware Configuration	C1			3.2.4
B3	RSTRSN	Auto-Reset Reason Code	00		Read only	3.2.4
B4	CLASTL	Capture Last Time Low	undefined		Read only	3.4.5
B5	CLASTH	Capture Last Time High	undefined		Read only	3.4.5
B6	CPREVL	Capture Prev. Time Low	undefined		Read only	3.4.5
B7	CPREVH	Capture Prev. Time High	undefined		Read only	3.4.5
B8	IP	Interrupt Priority Register 0	00	*		3.3.5
B9	I2CAL	I^2C Transaction Address Low	00			3.7
BA	I2CAH	I^2C Transaction Address High	00			3.7
BB	I2CTD	I^2C Transaction Data	00			3.7
BC	I2CCD	I^2C Command Data	00			3.7
BD	I2CDA	I^2C Device Address	A6			3.7
BE	I2CBC	I^2C Baudrate Control	C8		Default 320kHz with 4MHz crystal	3.7
BF	I2CNF	I^2C Noise Filter	14			3.7
C0	IS	Interrupt Status & Acknowledge	00	*		3.3.4
C1	P4IS	Port 4 Interrupt Status & Acknowledge	00			3.3.4
C2	P4IM0	Port 4 Interrupt Mode 0	FF			3.3.4
C3	P4IM1	Port 4 Interrupt Mode 1	FF			3.3.4
C8	T2CON	Timer /Counter 2 Control	00	*		3.4.2
CA	RCP2L	Timer 2 Capture Register Low	00			3.4.2
CB	RCP2H	Timer 2 Capture Register	00			3.4.2

Address (hex)	Label	Description	Reset Value (hex)	Bit Address-able	Notes	Document Section
		High				
CC	TL2	Timer /Counter 2 Low Byte	00			3.4.2
CD	TH2	Timer /Counter 2 High Byte	00			3.4.2
D0	PSW	Program Status Word	00	*		3.2.1
D1	MCEAD0	MCE Access Data 0	00			5.2
D2	MCEAD1	MCE Access Data 1	00			5.2
D3	MCEAD2	MCE Access Data 2	00			5.2
D4	MCEAD3	MCE Access Data 3	00			5.2
D5	PLLF0	PLL Frequency Factor bits 0 – 7	7E			3.2.3
D6	PLLF1	PLL Frequency Factor bits 8 – 15	C0			3.2.3
D7	PLLF2	PLL Frequency Factor bits 16 – 23	00			3.2.3
D9	MCEBL	MCE Sequencer Breakpoint Address Low	00		Internal use	
DA	MCEBH	MCE Sequencer Breakpoint Address High	00		Internal use	
DB	PLLF3	PLL Frequency Factor bits 24 – 31	00			3.2.3
DC	HWREV	Hardware Model and Revision	Device dependent		Read only	3.2.4
DD	MCESS	MCE Sequencer Status Word	00		Read only	5.2
DE	SYNCS	Sync Status Register	00			5.3
E0	ACC	Accumulator	00	*		3.2.1
E1	P4DIR	Port 4 Direction Register	00		IRMCx312 only	3.2.2
E2	P4	Port 4	00		IRMCx312 only	3.2.2
E3	MCESL	MCE Sequencer Stack Low	00		Read only Internal use	
E4	MCESH	MCE Sequencer Stack High	00		Read only Internal use	
E5	STOPS	GATEKILL Configuration	00			3.2.4
E6	P5DIR	Port 5 Direction Register	00			3.2.2
E7	P5	Port 5	00			3.2.2
E8	IE1	Interrupt Enable Register 1	00	*		3.3.4
E9	MCECD0	MCE Coherent Data 0	00			5.1.1
EA	MCECD1	MCE Coherent Data 1	00			5.1.1
EB	MCECD2	MCE Coherent Data 2	00			5.1.1
EC	MCECD3	MCE Coherent Data 3	00			5.1.1
EE	MCEAAH	MCE Access Address Low	00			5.2
EF	MCEAAL	MCE Access Address Low	00			5.2
F0	B	B Register	00	*		3.2.1
F8	IP1	Interrupt Priority Register 1	00	*		3.3.5

Table 5. Special Function Registers

3.2.1 Processor Registers

Some of the 8051 processor registers can be accessed as SFRs. These include the stack pointer (SP), data pointer (DPTR), program status word (PSW), accumulator (A or ACC) and the B register.

STACK POINTER (SP)

Address: 81h *Not Bit Addressable* *Reset value: 07h*

The SP register contains the Stack Pointer. The Stack Pointer is used to load the program counter into Internal Data Memory during LCALL and ACALL instructions and to retrieve the program counter from memory during RET and RETI instructions.

Data may also be saved on or retrieved from the stack using PUSH and POP instructions. Instructions that use the stack automatically pre-increment or post-decrement the Stack Pointer so that the Stack Pointer always points to the last byte written to the stack, i.e. the top of the stack. On reset the Stack Pointer is set to 07h.

It falls to the programmer to ensure that the location of the stack in Internal Data Memory does not interfere with other data stored therein.

DATA POINTER (DPTR)

Address: 82h (DPL), 83h (DPH) *Not Bit Addressable* *Reset value: 0000h*

The Data Pointer (DPTR) is a 16-bit register that is used to form 16-bit addresses for External Data Memory accesses (MOVX A,@DPTR and MOVX @DPTR,A), for program byte moves (MOVC A,@A+DPTR) and for indirect program jumps (JMP @A+DPTR).
Two true 16-bit operations are allowed on the Data Pointer – load immediate (MOV DPTR,#data) and increment (INC DPTR).

On reset all data pointers are set to 00h.

PROGRAM STATUS WORD (PSW)

Address: D0h *Bit Addressable* *Reset value: 00000000b*

PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	**AC**	**F0**	**RS1**	**RS0**	**OV**	**F1**	**P**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R

This register contains status information resulting from CPU and ALU operation. The bit definitions are given below:

PSW.7	**CY**	ALU carry flag.
PSW.6	**AC**	ALU auxiliary carry flag.
PSW.5	**F0**	General-purpose user-definable flag.
PSW.4	**RS1**	Register Bank Select bit 1.
PSW.3	**RS0**	Register Bank Select bit 0.
PSW.2	**OV**	ALU overflow flag.
PSW.1	**F1**	User-definable flag.
PSW.0	**P**	Parity flag. Set each instruction cycle to indicate odd/even parity in the accumulator.

The Register Bank Select bits PSW[4:3] operate as follows.

RS1	RS0	Register Bank Select
0	0	RB0. Registers from 00 – 07 hex.
0	1	RB1. Registers from 08 – 0F hex.
1	0	RB2. Registers from 10 – 17 hex.
1	1	RB3. Registers from 18 – 1F hex.

On reset this register returns 00h.

ACCUMULATOR (ACC)

Address: E0h Bit Addressable Reset value: 00h

This register provides one of the operands for most ALU operations. It is denoted as 'A' in the instruction set summary (Table 3).

On reset this register returns 00h.

B REGISTER (B)

Address: F0h Bit Addressable Reset value: 00h

This register provides the second operand for multiply or divide instructions. Otherwise, it may be used as a scratch pad register.

On reset this register returns 00h.

3.2.2 General Purpose I/O

I/O PORTS (P1, P2, P3, P4, P5)

Address: 90h (P1), A0h (P2), B0h (P3), E2h (P4), E7h (P5) *Bit Addressable* *Reset value: FFh (P0 – P3), 00h (P4 – P5)*

P1 – P5 are latches used to drive the quasi-bidirectional I/O lines. On reset, P1 – P3 are set to the value FF hex, and P4 – P5 are set to 00 hex. Some of the ports have dual functions as shown in the following list. Port P0 (at address 0x80) is not used. The external pin number associated with each port is dependent on the pinout for the specific IC, and some ports are not available on all ICs.

Port Name	Pin Number					Function
	311	312	341	343	371	
P1.0/T2	-	3	3	3	3	Discrete I/O, or Timer/Counter 2 input
P1.1/RXD	3	4	4	4	4	Discrete I/O, or UART0 RXD input
P1.2/TXD	4	5	5	5	5	Discrete I/O, or UART0 TXD output
P1.3/ SYNC/SCK	5	6	6	6	6	Discrete I/O, or SYNC output, or SPI clock output
P1.4/CAP	6	7	7	7	7	Discrete I/O, or Capture Timer input
P1.5	-	8	8	8	-	Discrete I/O
P1.6	-	9	9	9	-	Discrete I/O
P1.7	-	10	10	10	-	Discrete I/O
P2.0/NMI	-	23	14	14	11	Discrete I/O, or Non-maskable interrupt input
P2.1	-	24	15	15	12	Discrete I/O
P2.2	-	25	16	16	-	Discrete I/O
P2.3	-	26	17	17	-	Discrete I/O
P2.4	-	27	18	18	-	Discrete I/O
P2.5	-	28	19	19	-	Discrete I/O
P2.6/AOPWM0	17	29	20	20	13	Discrete I/O, or AOPWM0 output
P2.7/AOPWM1	18	30	21	21	-	Discrete I/O, or AOPWM1 output
P3.0/INT2/CS1	48	75	48	48	36	Discrete I/O, or INT2 input, or SPI chip select 0
P3.1/AOPWM2	-	78	49	51	-	Discrete I/O, or AOPWM2 output
P3.2/INT0	51	79	50	52	37	Discrete I/O, or INT0 input
P3.3/INT1	-	80	51	-	-	Discrete I/O, or INT1 input
P3.4/T0	-	81	-	-	-	Discrete I/O, or Timer/Counter 0 input
P3.5/T1	-	82	52	-	-	Discrete I/O, or Timer/Counter 1 input
P3.6/RXD1	52	83	-	-	-	Discrete I/O, or UART1 RXD input
P3.7/TXD1	53	84	-	-	-	Discrete I/O, or UART1 TXD output
P4.0/INT3	-	35	-	-	-	Discrete I/O, or INT3 input
P4.1/INT4	-	64	-	-	-	Discrete I/O, or INT4 input
P4.2/INT5	-	88	-	-	-	Discrete I/O, or INT5 input
P4.3/INT6	-	11	-	-	-	Discrete I/O, or INT6 input
P4.4/INT7	-	36	-	-	-	Discrete I/O, or INT7 input
P4.5/INT8	-	63	-	-	-	Discrete I/O, or INT8 input
P4.6/INT9	-	89	-	-	-	Discrete I/O, or INT9 input
P4.7/INT10	-	12	-	-	-	Discrete I/O, or INT10 input
P5.0/PFCGKILL	49	76	-	49	-	Discrete I/O, or PFC GATEKILL
P5.1/TDI	59	94	59	59	43	Discrete I/O, or JTAG test data input **In K-version, P5.1 is input only.**
P5.2/TMS	57	92	57	57	41	Discrete I/O, or JTAG test mode select **In K-version, P5.2 is input only.**
P5.3/TDO	58	93	58	58	42	Discrete I/O, or JTAG test data output **P5.3 not available in K-version.**

Table 6. Discrete I/Os from Ports 1 – 5

If the discrete I/O function is selected, any port has a bi-directional I/O shown in Figure 9. The input buffer has Schmitt trigger type input with hysteresis and a 70k ohm pull-up resistor.

Port Direction registers (P1DIR – P5DIR) configure the direction (input or output).

PORT 1 DIRECTION (P1DIR)

Address: 91h *Not Bit Addressable* *Reset value: 00000000b*

P1DIR.7	P1DIR.6	P1DIR.5	P1DIR.4	P1DIR.3	P1DIR.2	P1DIR.1	P1DIR.0
P1.7D	**P1.6D**	**P1.5D**	**P1.4D**	**P1.3D**	**P1.2D**	**P1.1D**	**P1.0D**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

P1DIR.7 – P1DIR.0	**P1.xD**	0: input, 1: output

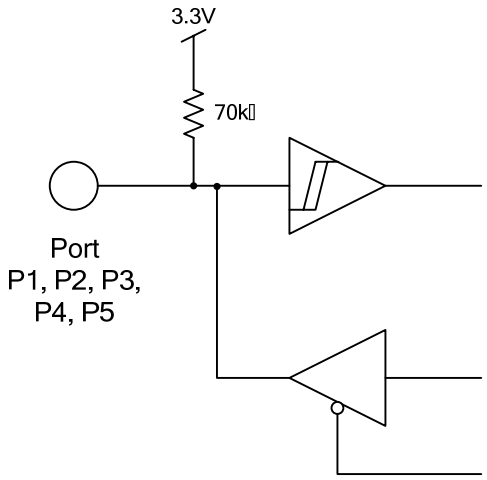


Figure 9. Port Structure of IRMCF300

PORT 2 DIRECTION (P2DIR)

Address: A1h *Not Bit Addressable* *Reset value: 00000000b*

P2DIR.7	P2DIR.6	P2DIR.5	P2DIR.4	P2DIR.3	P2DIR.2	P2DIR.1	P2DIR.0
P2.7D	**P2.6D**	**P2.5D**	**P2.4D**	**P2.3D**	**P2.2D**	**P2.1D**	**P2.0D**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

P2DIR.7 - P2DIR.0	**P2.xD**	0: input, 1: output

PORT 3 DIRECTION (P3DIR)

Address: B1h *Not Bit Addressable* *Reset value: 00000000b*

P3DIR.7	P3DIR.6	P3DIR.5	P3DIR.4	P3DIR.3	P3DIR.2	P3DIR.1	P3DIR.0
P3.7D	**P3.6D**	**P3.5D**	**P3.4D**	**P3.3D**	**P3.2D**	**P3.1D**	**P3.0D**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

P3DIR.7 - P3DIR.0	**P3.xD**	0: input, 1: output

PORT 4 DIRECTION (P4DIR)

Address: E6h *Not Bit Addressable* *Reset value: 00000000b*

P4DIR.7	P4DIR.6	P4DIR.5	P4DIR.4	P4DIR.3	P4DIR.2	P4DIR.1	P4DIR.0
P4.7D	**P4.6D**	**P4.5D**	**P4.4D**	**P4.3D**	**P4.2D**	**P4.1D**	**P4.0D**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

P4DIR.7 – P4DIR.0	**P4.xD**	0: input, 1: output

PORT 5 DIRECTION (P5DIR)

Address: E6h *Not Bit Addressable* *Reset value: 00000000b*

P5DIR.7	P5DIR.6	P5DIR.5	P5DIR.4	P5DIR.3	P5DIR.2	P5DIR.1	P5DIR.0
-	**-**	**-**	**-**	**P5.3D**	**P5.2D**	**P5.1D**	**P5.0D**
				R/W	R/W	R/W	R/W

P5DIR.7 – P5DIR.4	-	Unused
P5DIR.3	**P5.3D**	0: input, 1: output; **In K-version, Unused**
P5DIR.2 – P5DIR.1	**P5.xD**	0: input, 1: output; **In K-version, P5.1, P5.2 are always input**
P5DIR.0	**P5.0D**	0: input, 1: output

I/O CONTROL 0 (IOCON0)

Address: AAh *Not Bit Addressable* *Reset value: 00000000b*

IOCON0.7	IOCON0.6	IOCON0.5	IOCON0.4	IOCON0.3	IOCON0.2	IOCON0.1	IOCON0.0
UART1E	**UART0E**	**AOPWM2**	**AOPWM1**	**AOPWM0**	**AOPWMF**	**SYNC**	**NMI**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The bit definitions for this register are as follows:

IOCON0.7	**UART1E**	UART1 Enable. 1: P3.6 is RXD1 input and P3.7 is TXD1 output. 0: P3.6 and P3.7 are general purpose I/Os.
IOCON0.6	**UART0E**	UART0 Enable. 1: P1.1 is RXD input and P1.2 is TXD output. 0: P1.1 and P1.2 are general purpose I/Os.
IOCON0.5	**AOPWM2**	AOPWM2 Enable. 1: P3.1 is AOPWM2 output, 0: P3.1 is a general purpose I/O
IOCON0.4	**AOPWM1**	AOPWM1 Enable. 1: P2.7 is AOPWM1 output, 0: P2.7 is a general purpose I/O
IOCON0.3	**AOPWM0**	AOPWM0 Enable. 1: P2.6 is AOPWM0 output, 0: P2.6 is a general purpose I/O
IOCON0.2	**AOPWMF**	Frequency selection for AOPWM output. 1: PCLK/4096, 0: PCLK/1048
IOCON0.1	**SYNC**	SYNC output select. 1: P1.3 is SYNC output, 0: P1.3 is a general purpose I/O
IOCON0.0	**NMI**	NMI input select. 1: P2.0 is NMI input, 0: P2.0 is a general purpose I/O

I/O CONTROL 1 (IOCON1)

Address: ABh *Not Bit Addressable* *Reset value: 00000000b*

IOCON1.7	IOCON1.6	IOCON1.5	IOCON1.4	IOCON1.3	IOCON1.2	IOCON1.1	IOCON1.0
CAPR	**CAPM**	**CAPE**	**SYNCSEL**	**INT2M1**	**INT2M0**	**SYNCE**	**IOMODE**
R/W	R/W	R/W	R	R/W	R/W	R	R/W

The bit definitions for this register are as follows:

IOCON1.7	**CAPR**	Capture Timer (T4) Resolution. 1: PCLK, 0: PCLK/1024
IOCON1.6	**CAPM**	Capture Timer (T4) Interrupt Mode select. 1: Rising edge, 0: Falling edge
IOCON1.5	**CAPE**	Capture Timer input select. 1: P1.4 is Capture input, 0: P1.4 is a general purpose I/O
IOCON1.4	**SYNCSEL**	Sync signal select. 1: P1.3 is motor 1 sync output, 0: P1.3 is motor 2 sync output
IOCON1.3	**INT2M1**	Interrupt 2 Mode selection bit 1.
IOCON1.2	**INT2M0**	Interrupt 2 Mode selection bit 0.
IOCON1.1	**SYNCE**	Sync Status edge select. 1: SYNCS(DEh) is set at the rising edge of each signal, 0: SYNCS is set at the falling edge.
IOCON1.0	**IOMODE**	Controls the direction mode of all general purpose I/Os. 1: PxDIR determines the direction. 0: 8051 compatibility mode. In 8051 compatibility mode (default at power-up), all I/Os are open drain. Writing 0 to a bit drives the output to 0. Writing 1 causes the output to float.

Interrupt 2 mode selection (INT2M1, IN2M0) is encoded as follows:

INT2M1	INT2M0	Operating Mode
0	0	Interrupt 2 is level sensitive (low signal generates the interrupt)
0	1	Positive edge sensitive
1	0	Negative edge sensitive
1	1	Both Positive and Negative edge sensitive

3.2.3 Clock Selection and PLL Frequency Configuration

PLL FREQUENCY CONFIGURATION – PLLF3

Address: DBh *Not Bit Addressable* *Reset value: 00000000b*

PLLF3.7	PLLF3.6	PLLF3.5	PLLF3.4	PLLF3.3	PLLF3.2	PLLF3.1	PLLF3.0
LT7	**LT6**	**LT5**	**LT4**	**LT3**	**LT2**	**LT1**	**LT0**
W	W	W	W	W	W	W	W

PLL FREQUENCY CONFIGURATION – PLLF2

Address: D7h *Not Bit Addressable* *Reset value: 00000000b*

PLLF2.7	PLLF2.6	PLLF2.5	PLLF2.4	PLLF2.3	PLLF2.2	PLLF2.1	PLLF2.0
-	**CD1**	**CD0**	**-**	**-**	**SC2**	**SC1**	**SC0**
R	R/W	R/W	R	R	R/W	R/W	R/W

PLL FREQUENCY CONFIGURATION – PLLF1

Address: D6h *Not Bit Addressable* *Reset value: 11000000b*

PLLF1.7	PLLF1.6	PLLF1.5	PLLF1.4	PLLF1.3	PLLF1.2	PLLF1.1	PLLF1.0
OR1	**OR0**	**PS4**	**PS3**	**PS2**	**PS1**	**PS0**	**FM8**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

PLL FREQUENCY CONFIGURATION – PLLF0

Address: D5h *Not Bit Addressable* *Reset value: 01111110b*

PLLF0.7	PLLF0.6	PLLF0.5	PLLF0.4	PLLF0.3	PLLF0.2	PLLF0.1	PLLF0.0
FM7	**FM6**	**FM5**	**FM4**	**FM3**	**FM2**	**FM1**	**FM0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The bit definitions for registers PLLF0 – PLLF3 are as follows:

PLLF3.7 – PLLF3.0	**LT7 – LT0**	A write of any value to this register latches the values written to PLLF2 – PLLF0. Values written to those registers do not take effect until PLLF3 is written.
PLLF2.7		*Not implemented. Returns 0 when read.*
PLLF2.6 – PLLF2.5	**CD1 – CD0**	Clock divider bits (see table below) ***These bits are only valid in the K-version of the IC. In the F-version, do no write.***
PLLF2.4 – PLLF2.3		*Not implemented. Returns 0 when read.*
PLLF2.2 – PLLF2.0	**SC2 – SC0**	System clock configuration bits (see table below).
PLLF1.7 – PLLF1.6	**OR1 – OR0**	PLL output reduction (see table below).
PLLF1.5 – PLLF1.1	**PS4 – PS0**	PLL prescaler (a 5-bit unsigned value in the range 0 – 31).
PLLF1.0 – PLLF0.0	**FM8 – FM0**	PLL frequency multiplier (a 9-bit unsigned value in the range 0 – 511).

The clock select bits SC2 – SC0 are used to configure the system clock, as follows:

SC2	SC1	SC0	System Clock
0	0	0	Input clock
0	0	1	PLL clock
0	1	0	Input clock / 8
0	1	1	Input clock / 16
1	0	0	Input clock / 32
1	0	1	Input clock / 64
1	1	0	Input clock / 128
1	1	1	Input clock / 256

The PLL output reduction bits OR1 – OR0 are used to select the output divider value NO, as follows:

OR1	OR0	Output Divider
0	0	NO = 1
0	1	NO = 2
1	0	NO = 2
1	1	NO = 4

The values in the PLL frequency configuration registers are used to calculate the output frequency of the PLL clock generator, using the following formula:

FOUT = [FIN * (NF * 2)] / [(NR * 2) * NO]

where:

FIN = input clock frequency
NF = FM(frequency multiplier) + 2
NR = PS(prescaler) + 2
NO = output divider
FOUT = PLL output frequency (system clock)

For example, with a 4 MHz input clock, the default power-up values for the PLL frequency configuration registers produce the following result:

FIN = 4,000,000
NF = 126 + 2 = 128
NR = 0 + 2 = 2
NO = 4 (power-up value of OR1 – OR0 = 11b)

FOUT = [4,000,000 * (128 * 2)] / [(2 * 2) * 4] = [4,000,000 * 256] / [4 * 4] = 64 MHz system clock

The PLL clock generator requires that the values of FIN, NF and NR satisfy the following restrictions:

- FIN must be greater than 3.2 MHz
- FIN / (NR * 2) must be greater than 800 KHz and less than 8 MHz
- FIN * (NF * 2) / (NR * 2) must be greater than 128 MHz and less than 500 MHz

In the K-version (OTP) of the IC, the OTP memory limits the 8051 clock speed to 33MHz. Two bits (PLLF2[6:5]) control a clock divider which creates a reduced frequency clock signal from the output of the PLL. The PLL output becomes the MCE clock, while the output of the divider is the 8051 clock. The clock divider ratio can be set as described in the following table:

CD1	CD0	MCE Clock Frequency / 8051 Clock Frequency
0	0	1
0	1	2
1	0	3
1	1	4

The PWM frequency required by the application must also be considered when configuring the system clock rate (FOUT). The relationship is shown in Figure 10. The intersection of system clock rate and PWM frequency must fall *above* the diagonal line shown in the figure. This limitation is due to a counter overflow in the F-version of the IC. **This problem is not present in the K-version of the IC.**

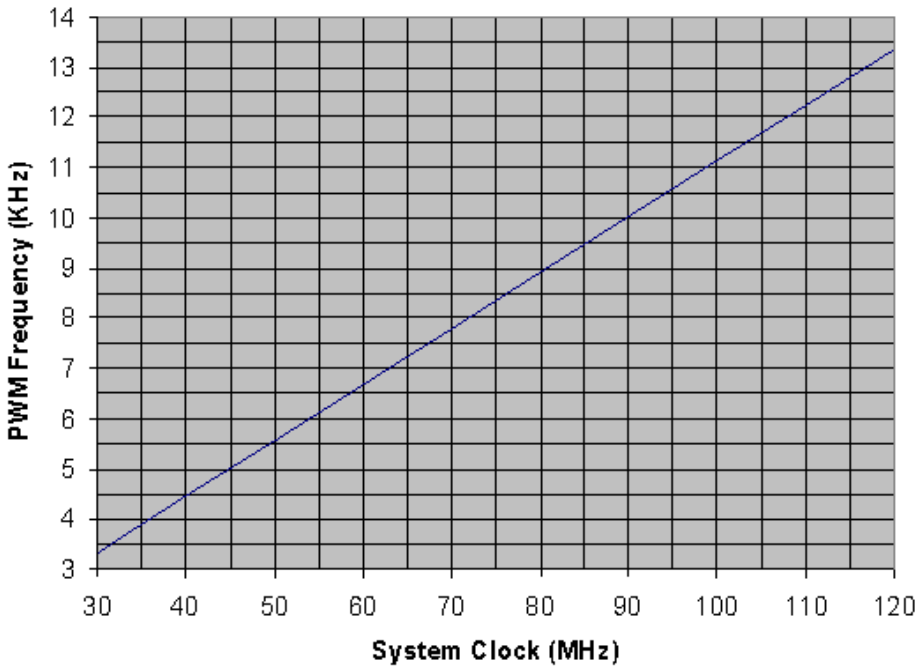


Figure 10. PWM Frequency Limit

3.2.4 Miscellaneous Functions

EXTENDED OPERATION (EO)

Address: A2h *Not Bit Addressable* *Reset value: 00000000b*

EO.7	EO.6	EO.5	EO.4	EO.3	EO.2	EO.1	EO.0
-	**-**	**-**	**TRAP_EN**	**0**	**-**	**-**	**-**
R	R	R	R/W	R	R	R	R

EO.4 is used to select the instruction executed with the opcode A5h (which is unused in the standard 8051), with bits 0 – 2 and bits 5 – 7 reserved for future expansion of this feature.

Bit definitions for this register are as follows:

EO.7 – EO.5	-	*Reserved for future use.*
EO.4	**TRAP_EN**	Selects the instruction to be executed by opcode A5h as follows: 1 Selects software TRAP instruction. 0 Selects MOVC @(DPTR++),A – a specific instruction that supports software download into program memory implemented as RAM (see Section 3.1).
EO.3	-	*Always returns 0.*
EO.2 – EO.0	-	*Reserved for future use.*

HARDWARE CONFIGURATION (HWCFG)

Address: B2h *Not Bit Addressable* *Reset value: 11000001b*

HWCFG.7	HWCFG.6	HWCFG.5	HWCFG.4	HWCFG.3	HWCFG.2	HWCFG.1	HWCFG.0
ADCP1	**ADCP0**	**VFSSEL**	**OP5**	**OP4**	**OP3**	**OP2**	**OP1**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The bit definitions for this register are as follows:

HWCFG.7	**ADCP1**	ADC power control bit 1.
HWCFG.6	**ADCP0**	ADC power control bit 0.
HWCFG.5	**VFSSEL**	*For internal use only. Always write 0 to this bit.*
HWCFG.4	**OP5**	OP amp 5 control. If 1, OP amp 5 (VAC) is enabled.
HWCFG.3	**OP4**	OP amp 4 control (IRMCx312 only). If 1, OP amp 4 (VDC) is enabled.
HWCFG.2	**OP3**	OP amp 3 control. If 1, OP amp 3 (PFC current) is enabled.
HWCFG.1	**OP2**	OP amp 2 control. If 1, OP amp 2 (Motor 2 current) is enabled.
HWCFG.0	**OP1**	OP amp 1 control. If 1, OP amp 1 (Motor 1 current) is enabled.

The ADC power control bits are defined as follows:

ADCP1	ADCP0	ADC Power Mode
0	0	Power down: lowest current
0	1	Sleep: low power, fast wakeup
1	0	Standby: less current than active
1	1	Active: full current, full functionality

For normal operation, the ADC power control bits should be set to 11b (active). The ADC should be disabled before selecting one of the low power modes.

RESET REASON CODE (RSTRSN)

Address: B3h *Not Bit Addressable* *Reset value: 00000000b*

RSTRSN.7	RSTRSN.6	RSTRSN.5	RSTRSN.4	RSTRSN.3	RSTRSN.2	RSTRSN.1	RSTRSN.0
-	**-**	**-**	**WDACT**	**WDEXP**	**RSN2**	**RSN1**	**RSN0**
R	R	R	R	R	R	R	R

The bit definitions for this register are as follows:

RSTRSN.7	**-**	*Not implemented. Returns zero when read.*
RSTRSN.6	-	*Not implemented. Returns zero when read.*
RSTRSN.5	-	*Not implemented. Returns zero when read.*
RSTRSN.4	**WDACT**	Watchdog active status. If 1, watchdog timer is enabled.
RSTRSN.3	**WDEXP**	Watchdog expiration status. If 1, watchdog timer has expired.
RSTRSN.2	**RSN2**	Reset reason code bit 2.
RSTRSN.1	**RSN1**	Reset reason code bit 1.
RSTRSN.0	**RSN0**	Reset reason code bit 0.

The reset reason code bits are defined as follows:

RSN2	RSN1	RSN0	Reset Reason
0	0	0	Power-on reset
0	0	1	Checksum error on boot
0	1	0	Undervoltage detection
0	1	1	Watchdog timer expiration
1	0	0	External reset

HARDWARE REVISION (HWREV)

Address: DCh *Not Bit Addressable* *Reset value: Revision code*

The hardware revision register identifies the hardware revision level. The value is fixed and the register is read-only.

HWREV	Revision Level
1	A
3	C

GATEKILL CONFIGURATION (STOPS)

Address: E5h *Not Bit Addressable* *Reset value: 00000000b*

STOPS.7	STOPS.6	STOPS.5	STOPS.4	STOPS.3	STOPS.2	STOPS.1	STOPS.0
-	**-**	**-**	**-**	**-**	**KP**	**KF**	**KC**
R	R	R	R	R	R/W	R/W	R/W

The bit definitions for this register are as follows:

STOPS.7 – STOPS.3	**-**	*Not implemented. Returns zero when read.*
STOPS.2	**KP**	PFC Gatekill source. If 0, PFC pwm shutdown signal comes from the PFCGKILL pin. If 1, PFC pwm shutdown signal comes from the CGATEKILL pin and the PFCGKILL pin is configured as general purpose I/O (P5.0).
STOPS.1	**KF**	**Always set to 1.** Otherwise, FGATEKILL pin will be driven internally.
STOPS.0	**KC**	Motor 1 Gatekill source. If 0, CGATEKILL or GATEKILL is driven to 0. If 1, motor 1 pwm shutdown signal comes from the CGATEKILL or GATEKILL pin.

3.3 Interrupts

The IRMCF300 supports various interrupt sources. These comprise the standard 8051/8052 internal interrupts and an additional eight interrupts. The standard and extended interrupts each have separate SFR enable bits associated with them, allowing full software control. There are two levels of interrupt priority. The non-maskable interrupt source is always enabled as long as the NMI bit is set (IOCON0.0). NMI has a higher priority than any other interrupt source, and is not controllable by software.

Table 7 provides a summary of all the supported interrupts. The first column shows the interrupt number and the second column shows the 8051 interrupt vector address at the base of 8051 internal data RAM. The third column identifies the interrupt source. The forth and fifth columns indicate which interrupt enable bit, in either register IE or IE1, is associated with the interrupt. (See Section 3.3.4 for more information about the interrupt enable registers.)

Interrupt Number	Vector Address (hex)	Interrupt Source	IE Bit Number	IE1 Bit Number
0	0003	INT0	0	
1	000B	Timer 0	1	
2	0013	INT1	2	
3	001B	Timer 1	3	
4	0023	UART 0	4	
5	002B	Timer 2	5	
6	0033	INT2		0
7	003B	Sync		1
8	0043	Timer 3		2
9	004B	UART1		3
10	0053	MCE		4
11	005B	Capture		5
12	0063	INT3 – INT10 (IRMCx312 only)	-	-
13	006B	*Unused*	-	-
14	0073	NMI	-	-

Note: Interrupts INT3 – INT10 are all tied to the same vector and are enabled using the P4IM0 and P4IM1 registers.

Table 7. Interrupt Source Summary

3.3.1 Standard Interrupts

The standard interrupts comprise three timer (Timer/Counter 0, 1 and 2) overflow interrupts, an interrupt associated with the core's built-in serial interface, and two maskable external interrupts (INT0 and INT1).

The Timer overflow interrupts, TF0, TF1, and TF2, are set whenever Timers 0, 1, and 2, respectively, rollover to zero. The states of these interrupts are stored in the TCON and T2CON registers. TF0 and TF1 (but not TF2) are automatically cleared by hardware on entry to the corresponding interrupt service routine.

The legacy external interrupts, INT0 and INT1, are driven from inputs P3.2 and P3.3 respectively. These interrupts may be either edge or level sensitive, depending on settings within the TCON register. Two further TCON register bits, IE0 and IE1, act as interrupt flags. If the external interrupt is set to edge-triggered, the corresponding register bit IE0/1 is set by a falling edge on INT0/1 and cleared by hardware on entry to the corresponding interrupt service routine. If the interrupt is set to be level sensitive, IE0/1 reflects the logic level on INT0/1.

Note: All events on INT0 and INT1, whether level-triggered or edge-triggered, are detected by sampling the relevant interrupt line on the rising edge of SCLK at the end of Phase 1 of every machine cycle. Where INT0/INT1 is level-triggered, a response is made to the signal being sampled low and, to ensure detection, the external source needs to hold the line low until the resulting interrupt is generated. (It also needs to ensure that the request is deactivated before the end of the associated service routine.) Where INT0/INT1 is edge-triggered, the response is made to a transition on

the signal from high to low between successive samples. This means that, to ensure detection, INT0/INT1 needs to have been high for at least two clocks before it goes low and then needs to be held low for at least two clocks after this transition.

3.3.2 Extended Interrupts

The extended interrupts include external interrupt 2 (INT2), the SYNC interrupt, Timer 3 (Periodic Timer), UART1, MCE interrupt, Timer 4 (Capture Timer). These interrupts are level-sensitive (low true) except External Interrupt 2 (INT2), which is configured using register IOCON1 (Section 3.2.2).

The internal interrupt line is sampled on the rising edge of PCLK at the beginning of Phase 2 of the last cycle of the current instruction.

See Section 5.3 for a description of the SYNC and MCE interrupts, which are generated by the MCE.

3.3.3 P4 Interrupts

The P4 general-purpose I/O pins can optionally generate external interrupts (INT4.0 – INT4.7). These eight interrupts are all tied to the same interrupt vector (interrupt number 12, shown in Table 7). The P4 interrupts are configured and serviced using the P4IM0, P4IM1 and P4IS registers.

3.3.4 Enabling Interrupts

The Non-Maskable Interrupt is always enabled. The maskable interrupts are enabled through a pair of bit-addressable Interrupt Enable registers (IE and IE1).

For the standard and extended interrupts, bits 0 to 5 of the IE register and bits 0 to 7 of the IE1 register each individually enable/disable a particular interrupt source. For the eight P4 interrupts, mode registers P4IM0 and P4IM1 are used to enable and select an operational mode for each interrupt individually.

Overall control is provided by bit 7 of IE (EA). When EA is set to ‘0’, all interrupts (except the NMI) are disabled: when EA is set to ‘1’, interrupts are individually enabled or disabled through the other bits of the Interrupt Enable and P4 mode Registers.

Both IE and IE1 are bit-addressable. The details of the registers are given below.

INTERRUPT ENABLE 0 (IE)

Address: A8h *Bit Addressable* *Reset value: 00000000b*

IE.7	IE.6	IE.5	IE.4	IE.3	IE.2	IE.1	IE.0
EA	**-**	**ET2**	**EU0**	**ET1**	**EX1**	**ET0**	**EX0**
R/W	R	R/W	R/W	R/W	R/W	R/W	R/W

For each bit in this register, “1” enables the corresponding interrupt and “0” disables it. The allocation of interrupts to bits is as follows:

IE.7	**EA**	Enable or disable all interrupt bits.
IE.6	-	*Not implemented. Returns zero when read.*
IE.5	**ET2**	Enable Timer 2 overflow interrupt.
IE.4	**EU0**	Enable UART 0 interrupt.
IE.3	**ET1**	Enable Timer 1 overflow interrupt.
IE.2	**EX1**	Enable External Interrupt 1 (INT1, P3.3)
IE.1	**ET0**	Enable Timer 0 overflow interrupt.
IE.0	**EX0**	Enable External Interrupt 0 (INT0, P3.2)

INTERRUPT ENABLE 1 (IE1)

Address: E8h *Bit Addressable* *Reset value: 00000000b*

IE1.7	IE1.6	IE1.5	IE1.4	IE1.3	IE1.2	IE1.1	IE1.0
-	**-**	**ET4**	**EMCE**	**EU1**	**ET3**	**ESYNC**	**EINT2**
R	R	R/W	R/W	R/W	R/W	R/W	R/W

For each bit in this register, a 1 enables the corresponding interrupt, and a 0 disables it. The allocation of interrupts to bits is as follows:

IE1.7	**-**	*Not implemented. Returns zero when read.*
IE1.6	**-**	*Not implemented. Returns zero when read.*
IE1.5	**ET4**	Enable Timer 4 (Capture Timer) interrupt
IE1.4	**EMCE**	Enable MCE interrupt
IE1.3	**EU1**	Enable UART 1 interrupt.
IE1.2	**ET3**	Enable Timer 3 (Periodic Timer) interrupt.
IE1.1	**ESYNC**	Enable SYNC Interrupt
IE1.0	**EINT2**	Enable External Interrupt 2 (INT2, P3.0)

P4 INTERRUPT MODE 0 (P4IM0)

Address: C2h *Not Bit Addressable* *Reset value: 11111111b*

P4IM0.7	P4IM0.6	P4IM0.5	P4IM0.4	P4IM0.3	P4IM0.2	P4IM01	P4IM0.0
INT10(0)	**INT9(0)**	**INT8(0)**	**INT7(0)**	**INT6(0)**	**INT5(0)**	**INT4(0)**	**INT3(0)**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

P4 INTERRUPT MODE 1 (P4IM1)

Address: C3h *Not Bit Addressable* *Reset value: 11111111b*

P4IM1.7	P4IM1.6	P4IM1.5	P4IM1.4	P4IM1.3	P4IM1.2	P4IM11	P4IM1.0
INT10(1)	**INT9(1)**	**INT8(1)**	**INT7(1)**	**INT6(1)**	**INT5(1)**	**INT4(1)**	**INT3(1)**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Each pair of bits from registers P4IM0 and P4IM1 define the interrupt mode for the corresponding P4 general-purpose I/O pin, as follows:

INTx(1)	INTx(0)	Interrupt Mode for INTx
0	0	Interrupt is disabled
0	1	Rising edge sensitive
1	0	Falling edge sensitive
1	1	Level sensitive (low signal generates an interrupt)

For example, if bit INT6(1) in P4IM1 is set to 1 and INT6(0) in P4IM0 is set to 0, then an interrupt is generated on the falling edge of input P4.3.

INTERRUPT STATUS AND ACKNOWLEDGE (IS)

Address: C0h *Bit Addressable* *Reset value: 00000000b*

IS.7	IS.6	IS.5	IS.4	IS.3	IS.2	IS.1	IS.0
-	**-**	**T4S**	**MCES**	**U1S**	**T3S**	**SYNCS**	**INT2S**
R/W	R/W	R/W	R/W	R	R/W	R/W	R/W

For each bit in this register, a 1 indicates that the associated interrupt has occurred and is pending. When written 0, the pending interrupt is cleared. (Note that in general it is not necessary to write to this register since most interrupts are cleared automatically when the interrupt is serviced.)

The allocation of status and acknowledge bits is as follows:

IS.7	**-**	*Not implemented. Returns zero when read.*
IS.6	**-**	*Not implemented. Returns zero when read.*
IS.5	**T4S**	Timer 4 (Capture Timer) interrupt is pending. Write 0 to reset.
IS.4	**MCES**	MCE interrupt is pending. Write 0 to reset.
IS.3	**U1S**	UART1 interrupt is pending. Write 0 to reset.
IS.2	**T3S**	Timer 3 (Periodic Timer) Interrupt is pending. Write 0 to reset.
IS.1	**SYNCS**	SYNC interrupt is pending. Write 0 to reset.
IS.0	**INT2S**	External Interrupt 2 (INT2) is pending. Write 0 to reset.

P4 INTERRUPT STATUS AND ACKNOWLEDGE (P4IS)

Address: C1h *Bit Addressable* *Reset value: 00000000b*

P4IS.7	P4IS.6	P4IS.5	P4IS.4	P4IS.3	P4IS.2	P4IS.1	P4IS.0
IS10	**IS9**	**IS8**	**IS7**	**IS6**	**IS5**	**IS4**	**IS3**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

For each bit in the P4IS register, a 1 indicates that the associated interrupt has occurred and is pending. When a bit written to 0, the pending interrupt is cleared. The P4 interrupts are not automatically cleared when they are serviced. The appropriate P4IS bit *must* be written to clear the pending interrupt.

P4IS.7	**IS10**	External interrupt INT10 is pending. Write 0 to reset.
P4IS.6	**IS9**	External interrupt INT9 is pending. Write 0 to reset.
P4IS.5	**IS8**	External interrupt INT8 is pending. Write 0 to reset.
P4IS.4	**IS7**	External interrupt INT7 is pending. Write 0 to reset.
P4IS.3	**IS6**	External interrupt INT6 is pending. Write 0 to reset.
P4IS.2	**IS5**	External interrupt INT5 is pending. Write 0 to reset.
P4IS.1	**IS4**	External interrupt INT4 is pending. Write 0 to reset.
P4IS.0	**IS3**	External interrupt INT3 is pending. Write 0 to reset.

3.3.5 Interrupt Priority

The standard 8051 architecture supports a two-level interrupt priority scheme. Under the two-level priority scheme, the priority level is decided solely on the IP and IP1 value.

Details of the registers are given below. IP and IP1 are bit-addressable.

Note: No priority level is assigned to the NMI. It simply takes precedence over all other interrupts.

INTERRUPT PRIORITY (IP)

Address: B8h *Bit Addressable* *Reset value: 00000000b*

IP.7	IP.6	IP.5	IP.4	IP.3	IP.2	IP.1	IP.0
-	**-**	**PT2**	**PU0**	**PT1**	**PX1**	**PT0**	**PX0**
R	R	R/W	R/W	R/W	R/W	R/W	R/W

For each bit of the IP register, "1" selects high priority for the interrupt enabled by the corresponding bit of the IE register, while "0" selects low priority for this interrupt.

The allocation of interrupts to bits is as follows:

IP.7	**-**	*Reserved.*
IP.6	**-**	*Reserved.*
IP.5	**PT2**	Select priority for Timer 2 overflow Interrupt.
IP.4	**PU0**	Select priority for UART 0 Interrupt.
IP.3	**PT1**	Select priority for Timer 1 overflow Interrupt.
IP.2	**PX1**	Select priority for External Interrupt 1. (P3.3)
IP.1	**PT0**	Select priority for Timer 0 overflow Interrupt.
IP.0	**PX0**	Select priority for External Interrupt 0. (P3.2)

INTERRUPT PRIORITY 1 (IP1)

Address: F8h *Bit Addressable* *Reset value: 00000000b*

IP1.7	IP1.6	IP1.5	IP1.4	IP1.3	IP1.2	IP1.1	IP1.0
-	**-**	**PT4**	**PMCE**	**PU1**	**PT3**	**PSYNC**	**PINT2**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

For each bit of the IP1 register, "1" selects high priority for the interrupt enabled by the corresponding bit of the IE1 register, while "0" selects low priority for this interrupt.

The allocation of interrupts to bits is as follows:

IP1.7	**-**	*Not implemented. Returns zero when read.*
IP1.6	**-**	*Not implemented. Returns zero when read.*
IP1.5	**PT4**	Select priority for Timer 4 (Capture Timer) Interrupt.
IP1.4	**PMCE**	Select priority for MCE Interrupt.
IP1.3	**PU1**	Select priority for UART1 Interrupt.
IP1.2	**PT3**	Select priority for Timer 3 (Periodic Timer) Interrupt.
IP1.1	**PSYNC**	Select priority for SYNC Interrupt.
IP1.0	**PX2**	Select priority for External Interrupt 2. (P3.0)

3.3.6 Service Order

An interrupt service routine may only be interrupted by an interrupt of higher priority and, if two interrupts of different priority occur at the same time, the higher level interrupt will be serviced first. An interrupt cannot be interrupted by another interrupt of the same or a lower priority level.

If two interrupts of the same priority level occur simultaneously, a polling sequence is observed as follows:

Source	Level	Description
NMI	0 (Highest)	Non-Maskable Interrupt
IE0	1	External Interrupt 0 from P3.2/INT0
TF0	2	Timer/Counter 0 Interrupt
IE1	3	External Interrupt 1 from P3.3/INT1
TF1	4	Timer/Counter 1 Interrupt
U0	5	UART 0 Interrupt
TF2	6	Timer/Counter 2 Interrupt
INT2	7	External Interrupt 2 from P3.0/INT2
SYNC	8	SYNC Interrupt
T3	9	Timer 3 (Periodic Timer) Interrupt
U1	10	UART 1 Interrupt
MCE	11	MCE Interrupt
T4	12 (Lowest)	Timer 4 (Capture Timer) Interrupt

Table 8. Interrupt Service Order

3.3.7 Interrupt Latency

The response time in a single interrupt system is between 3 and 9 machine cycles.

3.3.8 Interrupt Vectors

When an interrupt is serviced, a long call instruction is executed to an interrupt vector address determined by the source of the interrupt. The vector associated with each interrupt source is shown in Table 7.

3.4 Timers

This section describes the three general-purpose timer/counter devices and the three additional special-purpose timers: a periodic timer, a watchdog timer and a capture timer.

3.4.1 Timer Prescaler

A prescaler register PSCL can be configured to divide the clock on input to the three general-purpose timers and the periodic timer. The clock rate on input to each of the four timers can be individually configured as described below.

The clock input to the watchdog timer and capture timer cannot be prescaled.

TIMER PRESCALER (PSCL)

Address: A9h *Not Bit Addressable* *Reset value: 00000000b*

PSCL.7	PSCL.6	PSCL.5	PSCL.4	PSCL.3	PSCL.2	PSCL.1	PSCL.0
P1(3)	**P0(3)**	**P1(2)**	**P0(2)**	**P1(1)**	**P0(1)**	**P1(0)**	**P0(0)**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The bit definitions for the PSCL register are as follows:

PSCL.7	**P1(3)**	Periodic timer prescale control bit P1.
PSCL.6	**P0(3)**	Periodic timer prescale control bit P0.
PSCL.5	**P1(2)**	Timer 2 prescale control bit P1.
PSCL.4	**P0(2)**	Timer 2 prescale control bit P0.
PSCL.3	**P1(1)**	Timer 1 prescale control bit P1.
PSCL.2	**P0(1)**	Timer 1 prescale control bit P0.
PSCL.1	**P1(0)**	Timer 0 prescale control bit P1.
PSCL.0	**P0(0)**	Timer 0 prescale control bit P0.

For all four timers, the prescale bits P0 and P1 apply as follows:

P1	P0	Function
0	0	No prescaler. The counting clock is PCLK, half the SYSCLK.
0	1	PCLK is divided by 16 (or SYSCLK divided by 32).
1	0	PCLK is divided by 256 (or SYSCLK divided by 512).
1	1	PCLK is divided by 4096 (or SYSCLK divided by 8192).

3.4.2 General-Purpose Timer/Counters

Two 16-bit timer/counters are provided, Timer 0 and Timer 1. The TCON and TMOD registers are used to set the mode of operation and to control the running and interrupt generation of these two devices, with the timer/counter values stored in two pairs of 8-bit registers (TL0, TH0 and TL1, TH1).

TIMER /COUNTER CONTROL (TCON)

Address: 88h *Bit Addressable* *Reset value: 00000000b*

TCON.7	TCON.6	TCON.5	TCON.4	TCON.3	TCON.2	TCON.1	TCON.0
TF1	**TR1**	**TF0**	**TR0**	**IEDG1**	**IT1**	**IEDG0**	**IT0**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The bit definitions for this register are as follows:

TCON.7	**TF1**	Timer 1 overflow flag. Set by hardware when Timer/Counter 1 overflows. Cleared by hardware when the processor calls the interrupt service routine.
TCON.6	**TR1**	Timer 1 run control. If 1, timer runs; if 0, timer is halted.
TCON.5	**TF0**	Timer 0 overflow flag. Set by hardware when Timer/Counter 0 overflows. Cleared by hardware when the processor calls the interrupt service routine.
TCON.4	**TR0**	Timer 0 run control. If 1, timer runs; if 0, timer is halted.
TCON.3	**IEDG1**	External Interrupt 1 edge flag. Set by hardware when an External Interrupt 1 edge is detected.
TCON.2	**IT1**	External Interrupt 1 control bit. If 1, External Interrupt 1 is "edge-triggered"; if 0, External Interrupt 1 is "level triggered" (see Section 3.3).
TCON.1	**IEDG0**	External Interrupt 0 edge flag. Set by hardware when an External Interrupt 0 edge is detected.
TCON.0	**IT0**	External Interrupt 0 control bit. If 1, External Interrupt 0 is "edge-triggered"; if 0, External Interrupt 0 is "level triggered" (see Section 3.3).

TIMER /COUNTER MODE (TMOD)

Address: 89h *Not Bit Addressable* *Reset value: 00000000b*

TMOD.7	TMOD.6	TMOD.5	TMOD.4	TMOD.3	TMOD.2	TMOD.1	TMOD.0
GATE1	**C/NT1**	**M1(1)**	**M0(1)**	**GATE0**	**C/NT0**	**M1(0)**	**M0(0)**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The bit definitions for this register are as follows:

TMOD.7	**GATE1**	Timer 1 gate flag. When TCON.6 is set and GATE1 = 1, Timer/Counter 1 will only run if INT1 pin is 1 (hardware control). When GATE1 = 0, Timer/Counter 1 will only run if TCON.6 = 1(software control).
TMOD.6	**C/NT1**	Timer/Counter 1 selector. If 0, input is from internal system clock; if 1, input is from T1 pin.
TMOD.5	**M1(1)**	Timer 1 Mode control bit M1.
TMOD.4	**M0(1)**	Timer 1 Mode control bit M0.
TMOD.3	**GATE0**	Timer 0 gate flag. When TCON.4 is set and GATE0 = 1, Timer/Counter 0 will only run if INT0 pin is 1 (hardware control). When GATE0 = 0, Timer/Counter 0 will only run if TCON.4 = 1(software control).
TMOD.2	**C/NT0**	Timer/Counter 0 selector. If 0, input is from internal system clock; if 1, input is from T0 pin.
TMOD.1	**M1(0)**	Timer 0 Mode control bit M1.
TMOD.0	**M0(0)**	Timer 0 Mode control bit M0.

For both timer/counters, the mode bits M0 and M1 apply as follows:

M1	M0	Operating Mode
0	0	Mode 0: 13-bit timer/counter (M8048 compatible mode).
0	1	Mode 1: 16-bit timer/counter.
1	0	Mode 2: 8-bit auto-reload timer/counter.
1	1	Mode 3: Timer 0 is split into two halves. TL0 is an 8-bit timer/counter controlled by the standard Timer 0 control bits. TH0 is an 8-bit timer/counter controlled by the standard Timer 1 control bits. In this mode, the prescaler will have no effect on TH0. TH1 and TL1 are held (Timer 1 is stopped).

TIMER / COUNTER DATA (TL0, TL1, TH0, TH1)

Address: 8Ah (TL0), 8Bh (TL1), 8Ch (TH0), 8Dh (TH1) *Not Bit Addressable* *Reset value: 00h*

TL0 and TH0 are the low and high bytes of Timer/Counter 0 respectively. TL1 and TH1 are the low and high bytes of Timer/Counter 1 respectively. In Mode 2, the TL register is an 8-bit counter while TH stores the reload value. In Mode 2, if the prescaler is used, then the timer will not generate an interrupt.

On reset, all timer/counter registers are 00h.

TIMER / COUNTER 2 CONTROL (T2CON)

Address: C8h *Bit Addressable* *Reset value: 00000000b*

T2CON.7	T2CON.6	T2CON.5	T2CON.4	T2CON.3	T2CON.2	T2CON.1	T2CON.0
TF2	**EXF2**	**RCLK2**	**TCLK2**	**EXEN2**	**TR2**	**C/NT2**	**CP/NRL2**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit definitions for this register are as follows:

T2CON.7	**TF2**	Timer 2 overflow flag. Set by hardware when Timer/Counter 2 overflows unless either RCLK or TCLK is set to 1. This bit is not cleared by hardware when the processor calls the interrupt service routine.
T2CON.6	**EXF2**	Timer 2 external flag. This bit is set when a capture or reload is triggered by a negative transition on T2EX and EXEN2 is set to 1. If the Timer 2 interrupt is enabled, setting this bit will cause an interrupt to the Timer 2 vector.
T2CON.5	**RCLK2**	If this bit is set, the Serial Port receive clock is driven from the overflow pulses of Timer 2.
T2CON.4	**TCLK2**	If this bit is set, the Serial Port transmit clock is driven from the overflow pulses of Timer 2.
T2CON.3	**EXEN2**	Timer 2 External interrupt enable flag. When set, a negative edge on T2EX will trigger a capture or auto-reload.
T2CON.2	**TR2**	Run control bit for Timer 2. If set to 1, the timer is enabled.
T2CON.1	**CT2**	Timer/Counter select. A 0 selects internal timer mode; a 1 selects external counter mode.
T2CON.0	**CPRL2**	Capture/Reload control. When set, captures occur on negative transitions of T2EX (if EXEN2 is set). When 0, auto-reloads are performed on timer overflows or on negative transitions of T2EX(if EXEN2 is set). If either RCLK or TCLK is 1, this bit is ignored and auto-reloads are performed on timer overflows.

TIMER / COUNTER 2 DATA (RCP2L, RCP2H, TL2, TH2)

Address: CAh (RCP2L), CBh (RCP2H), CCh (TL2), CDh (TH2) *Not Bit Addressable* *Reset value: 00h*

TL2 and TH2 are the low and high bytes of Timer/Counter 2 data. RCP2L and RCP2H are the low and high bytes of the Timer 2 capture registers. These registers are also used for auto-reload.

On reset all these registers are 00h.

3.4.2.1 Modes of Operation

IRMCF300 includes three general-purpose counter/timers (Timers 0, 1 and 2). For each timer's counter value, there are two 8-bit special function registers (SFRs), one providing the low byte of the timer/counter value and the other the high byte. Further SFRs are used to configure and control the timers.

In Timer mode, the counter/timers count machine cycles; in Counter mode, the counter/timers count high-to-low transitions on the corresponding input pin (T0, T1, T2). Applications for the timers include measuring the time interval between events, counting events and generating a signal at regular intervals.

Timers 0 and 1 can each be configured either as a 16-bit counter/timer, a 13-bit counter/timer, or an 8-bit auto-reload counter/timer. (The auto-reload option allows automatic resetting in a counter counting up to 256.) Alternatively, Timer 0 can be split into two 8-bit timers. (The 13-bit mode provides an 8-bit counter with a divide-by-32 prescaler and is included solely for compatibility with Intel 8048 devices.)

Timer 2 has two modes of operation: a capture mode in which the current value of the timer is captured into the RCP2L and RCP2H registers; and an auto-reload mode in which Timer 2 is automatically reloaded with the contents of RCP2L and RCP2H.

3.4.2.2 Configuring the Timers

Timers 0 and 1 are configured by writing to the TMOD register. The high-order nibble of the register controls Timer 1 while the lower nibble controls Timer 0.

The controls included comprise a GATE flag which sets whether the timer is under hardware or software control (Bits 7 and 3 respectively); a Counter/Timer selector which controls whether the timer is to be triggered by the internal system clock or from the corresponding input pin (Bits 6 and 2 respectively) and a pair of bits which select the mode in which the timer is used (Bits 5 & 4 and 1 & 0, respectively).

Once the mode is configured, the operation of Timers 0 and 1 is controlled using the TCON register.

3.4.2.3 Using the Timers to Measure a Time Interval

When triggered by the internal system clock, the timers are incremented once every machine cycle (i.e. once every two CLKs). That is, the clock input to the timers is PCLK, which is equal to SYSCLK / 2. (See Section 3.2.3 for system clock configuration.)

To use any of the timers to measure a time interval, you therefore need to:

1. Set the timer you propose to use into the required timer mode by setting the appropriate mode bits.
2. Set the corresponding C/NT bit to 0 so that the timer is triggered by the internal system clock.
3. Use the GATE bit to put the running of the timer under hardware or software control as appropriate.
4. Set the required initial value for the timer by writing to the associated SFRs.
5. Start timing by setting the appropriate Run bit with an instruction such as SETB TR1.
6. Stop timing by clearing the Run bit (CLR TR1)
7. Calculate the elapsed time by dividing the difference between the initial and the final setting by PCLK (half of the system clock frequency).

Note: In 16-bit mode, a timer can count up to 65,536, which is equivalent to 65,536x2/f_{osc} seconds (where f_{osc} is the input clock frequency). In 13-bit mode, it can count up to 8,192 or 8,192x2/f_{osc} seconds, while in an 8-bit mode, it can count up to 256 or 256x2/f_{osc} seconds. To measure longer periods than these, a count must be kept of the number of times the timer overflows.

Note also that the total of a number of separate time intervals, for example the total time that some device is switched on, can be determined simply by stopping and starting the timer at the appropriate points during the period over which the time is measured.

Each time the timer is restarted, it will continue counting from the value at which it was previously halted, giving the required cumulative total.

3.4.2.4 Using the Timers to Signal When a Defined Period Has Elapsed

A timer can be used to signal when a defined period has elapsed by letting the timers count machine cycles. There are two approaches that can be taken.

The first option is to calculate the value that you will require the timer to count up to (either once or many times) and use a CJNE instruction, for example, to watch for when the timer reaches the required value.

The alternative approach is to set the initial value of the timer equal to its maximum setting minus the value you calculate and use the corresponding overflow flag to signal when the required time has elapsed. In particular, you may be able to use a timer in its Auto-Reload mode to count from this initial value up to overflow as many times as are required. The application just needs to keep track of the number of times the timer has overflowed.

Note: The overflow flags for Timer 0 and Timer 1 (TF0 and TF1) are in the TCON register. The overflow flag for Timer 2 (TF2) is included in the T2CON register.

3.4.2.5 Using the Timers as Event Counters

To use any of the timers as an event counter, you need to put it into counter mode by setting the corresponding C/NT bit to 1. Then, rather than counting machine cycles, the timer will count high-to-low transitions on the corresponding timer input line (T0, T1 or T2).

Note: T0, T1 and T2 are alternate uses of PORT3.4, PORT3.5, and PORT1.0 respectively. You should also note that the timer input lines are sampled once every machine cycle (at the end of the second phase) and the count is incremented when the samples record a high in one cycle and a low in the following cycle. Recognizing a transition therefore takes two machine cycles or 4 SYSCLK periods. Events that have a shorter time period will be undersampled and will therefore be not recognized reliably.

3.4.2.6 Reading the Timers

Timers 0 and 1 are straightforward to read when they are operating in one of their 8-bit modes as all that is required is a simple read of the appropriate SFR. Reading a timer that is operating in either a 13-bit or a 16-bit mode, however, takes two cycles – leaving you open to the risk that the timer may change its value between the two reads.

Two strategies may help here. One approach is to read the high byte, then read the low byte, then read the high byte again. If the high byte hasn't changed, the readings made correctly record the value of the timer at the time the low byte was read. If the high byte has changed, however, the readings should be made again as the values read give an uncertain result.

The other option is to stop the timer by clearing the appropriate Run bit while the reading is made. However, this approach should only be taken if the application for which the IRMCF300 is being used can tolerate the timer being stopped for a short while.

3.4.3 Periodic Timer

The periodic timer is used to produce interrupts at preprogrammed intervals. It consists of a 16-bit counter and a 16-bit period (limit) register. The period register can be read or written at any time, but the counter cannot be read or modified. The period is accessed as two 8-bit registers (high byte at A7h and low byte at A6h).

The period register determines the timer period. The counter counts down from the period register value to "1". When the counter reaches "1", an interrupt is generated and the timer automatically begins counting down again from the period register value. In order to activate the counter, a non-zero value should be written into the period register. If the period register is zero, the counter stops and no interrupts are generated. The periodic timer is inactive (period set to zero) on hardware reset.

The timer runs at PCLK, which is a half of the SYSCLK. (See Section 3.2.3 for system clock configuration.)

TIMER 3 PERIOD REGISTER (TL3, TH3)

Address:	*A6h (TL3), A7h (TH3)*	*Not Bit Addressable*	*Reset value:*	*0000h*

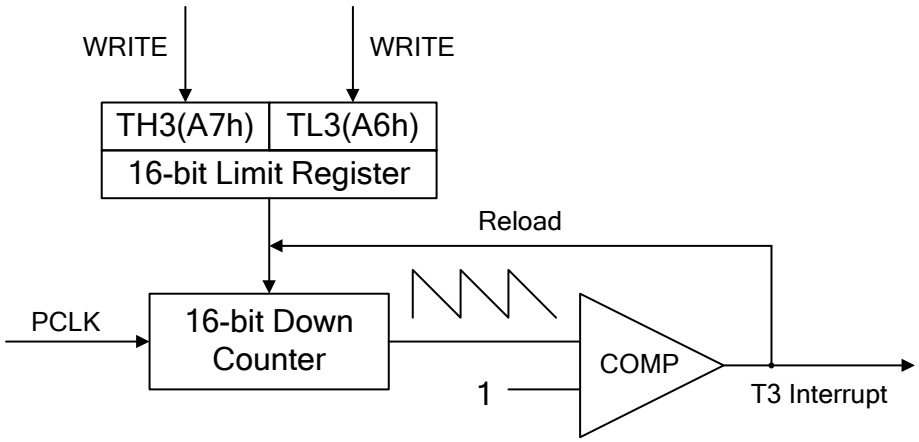


Figure 11. Periodic Timer

3.4.4 Watchdog Timer

The watchdog timer is a fail-safe mechanism that generates an automatic chip reset if the 8051 software fails to execute its intended program sequence for any reason. As with typical watchdog timers, the program must reset a counter on a regular basis to maintain normal operation; if the software fails to reset the counter (the program is not operating properly) the watchdog "times out" and resets the device. Unlike a typical watchdog timer, however, the IRMCF300 watchdog is driven by an internal analog oscillator so the timer will continue to run even if the external oscillator fails. In this case software execution halts, but the watchdog times out as it should and generates a chip reset so that all signals are reset to their initial power-on values and the motor drive reverts to an idle state.

The watchdog timer is a 16-bit timer with an associated 16-bit watchdog limit register. The clock is taken directly from a special on-chip independent 1 MHz (approximate) analog oscillator, divided by 16 and fed to the watchdog timer. When the 16-bit watchdog counter reaches (counts up to) the value in the 16-bit watchdog limit, an internal reset is generated. Whenever a user program writes or reads the high byte of the 16-bit watchdog limit register, the 16-bit watchdog counter is reset to zero and starts counting again toward the specified value in the limit register. The block diagram is shown in Figure 12.

After power-on reset, the watchdog limit register is set to the maximum value (FFFFh) and the watchdog timer is enabled. Every time the high byte of the limit register is read the counter starts again from zero. Writing a non-zero value to the high byte of the limit register also starts the counter from zero. Writing "0" to both bytes of the limit register disables the watchdog timer.

With the watchdog enabled and the limit register set to the maximum value, the counter reaches the limit value in approximately one second (2^16 * 1/62.5kHz). Therefore, the user program must reset the counter (by reading or writing the high byte of the limit register) at regular intervals of less than one second to avoid a system reset. If it is desirable to detect a software "hang" in less than one second, the value in the limit register can be adjusted accordingly.

The watchdog timer is always disabled when the system is in the debug mode (the JTAG debug port is active).

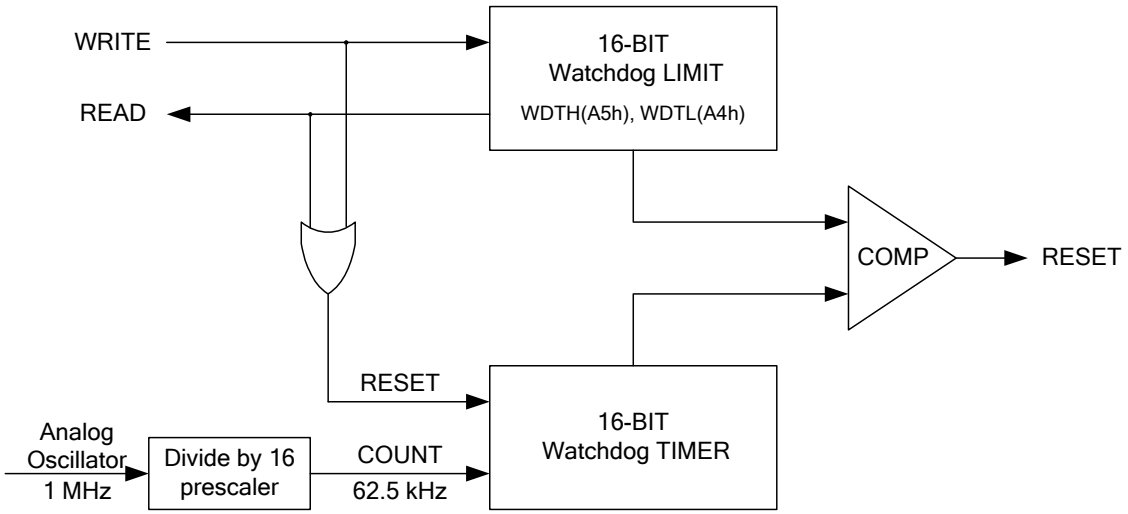


Figure 12. Watchdog Timer

WATCHDOG LIMIT REGISTER (WDTL, WDTH)

Address: A4h (WDTL), A5h (WDTH) Not Bit Addressable Reset value: 0000h

The limit register is read/write. When the limit register is read, the 16-bit watchdog counter is cleared to zero and begins counting up again. When the limit register is written, a new value is stored into the limit register and at the same time the 16-bit watchdog timer is cleared and begins counting up again.

3.4.5 Capture Timer

The capture timer (Timer 4) is used to measure the duty cycle and frequency period of an external signal. P1.4 can be assigned to capture timer input through bit 5 of I/O Control register 1 (CAPE, IOCON1.5). The counting frequency resolution is selected using bit 7 of I/O Control register 1 (CAPR, IOCON1.7). It can be set to either PCLK (SYSCLK / 2) or PCLK scaled by a fixed 10-bit prescaler (SYSCLK / 2048).

Two 16-bit counter values are latched into the CPREVH/CPREVL (previous pulse time) and CLASTH/CLASTL (last pulse time) registers as the pulse times are measured. The interrupt polarity setup through bit 6 of I/O Control register 1 (CAPM, IOCON1.6) determines at which edge an interrupt is generated, and also which waveform segment is considered the “last pulse time” and which is considered the “previous pulse time”.

Figure 13 shows the behavior of the capture timer for both rising-edge and falling-edge configuration.

The pulse time registers must be read in the following order: CLASTL, CLASTH, CPREVL, CPREVH. When CLASTL is read, the values in all four registers are held (no new values are latched into the registers). This ensures that coherent data from a single duty cycle can be retrieved. When CPREVH is read, the registers are released and the timer begins to sample new values. CPREVH must be read in order to re-enable the timer.

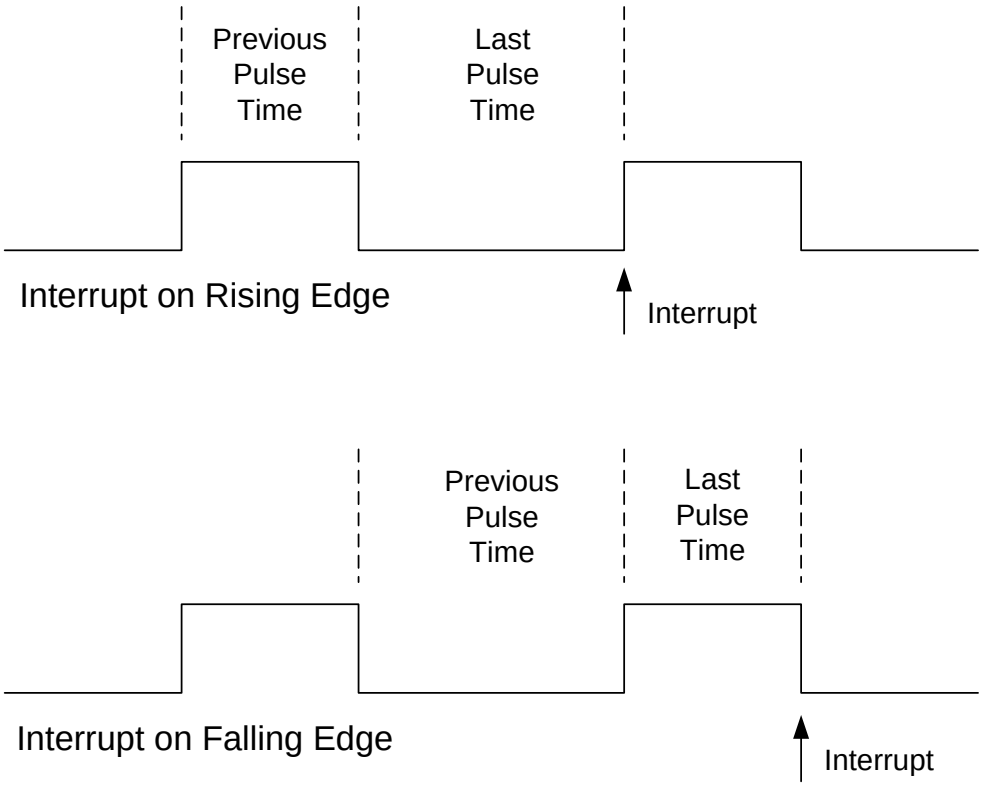


Figure 13. Capture Timer

CAPTURE LAST TIME DATA (CLASTL, CLASTH)

Address:	*B4h(CLASTL), B5h(CLASTH)*	*Not Bit Addressable*	*Reset value:*	*Undefined*

The read-only Capture Last Time data register is used to read the 16-bit counter value for the most recent pulse time period measured by the capture timer. CLASTL should be read first to hold the current values in the capture timer data registers.

CAPTURE PREVIOUS TIME DATA (CPREVL, CPREVH)

Address:	*B6h(CPREVL), B7h(CPREVH)*	*Not Bit Addressable*	*Reset value:*	*Undefined*

The read-only Capture Previous Time data register is used to read the 16-bit counter value for the pulse time period immediately preceding the "last time" period. CPREVH should be read last to release the capture timer data registers, and must be read to re-enable the capture timer.

3.5 UARTs

The IRMCF300 has up to two UARTs which are different from standard 8051 UART. There are four SFRs to operate each UART:

For UART0: U0CTL, U0STAT, U0BUF, and U0BR
For UART1: U1CTL, U1STAT, U1BUF, and U1BR

Both UARTs offer the same features, which can be summarized as follows:

- Programmable to 8-bit or 9-bit parity (odd/even) or 9-bit data operation.
- Programmable 8-bit baud rate. SYSCLK divided by the baud rate value (U0BR/U1BR) gives the sampling clock frequency, which is 16 times the UART bit clock.
- The 16 times clock samples three times in the middle of each bit. If the data is different, a noisy indication is set. This noisy bit is sticky across the whole character.
- The receive and transmit data buffers each hold two characters.
- An overrun error is detected if new data cannot be stored in the receive buffer because the buffer is full.
- A framing error is detected if no stop bit present.
- In discard mode bytes with errors are simply discarded so the CPU doesn't have to process them.
- An interrupt is issued when the receive buffer contains data or when the transmit buffer has available space.
- When the hunt mode bit is set, the UART generates an interrupt only when the ninth bit is equal to the U0CTL.2/U1CTL.2 bit. This feature may be used to hunt for the start of a message.
- In case the UART receives a constant input of "0", it will generate an interrupt at a byte interval. (The number of bits per byte depends on the parity setting.) This interval includes a start bit, data (8/9) bits and a stop bit. Then, because the stop bit is '0', a framing error will occur.

UART0 CONTROL (U0CTL)

Address: 94h *Not Bit Addressable* *Reset value: 00000000b*

U0CTL.7	U0CTL.6	U0CTL.5	U0CTL.4	U0CTL.3	U0CTL.2	U0CTL.1	U0CTL.0
-	**U0XINTE**	**U0HUNT**	**U0EN**	**U0DISC**	**U0DATA**	**U0M1**	**U0M0**
	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The bit definitions for this register are as follows:

U0CTL.7	**-**	Reserved
U0CTL.6	**U0XINTE**	UART0 transmit interrupt enable.
U0CTL.5	**U0HUNT**	When this bit is set, the UART hunts for a byte with bit 9 set as specified by U0CTL.2 (U0DATA) and only then generates a receive interrupt. All bytes before that are ignored. The "hunt" feature is only available in UART Mode 3 (U0M1=1, U0M0=1).
U0CTL.4	**U0EN**	Enables the activity of UART0.
U0CTL.3	**U0DISC**	Setting the "discard" bit causes UART0 to discard data received with errors such as framing error, parity error or noise error.
U0CTL.2	**U0DATA**	In UART Mode 3 (U0M1=1, U0M0=1), this bit is the data for bit 9.
U0CTL.1	**U0M1**	UART0 Mode 1 (see below)
U0CTL.0	**U0M0**	UART0 Mode 0 (see below)

UART1 CONTROL (U1CTL)

Address: 9Ch *Not Bit Addressable* *Reset value: 00000000b*

U1CTL.7	U1CTL.6	U1CTL.5	U1CTL.4	U1CTL.3	U1CTL.2	U1CTL.1	U1CTL.0
-	**U1XINTE**	**U1HUNT**	**U1EN**	**U1DISC**	**U1DATA**	**U1M1**	**U1M0**
	R/W	R/W	R/W	R/W	R/W	R/W	R/W

The bit definitions for this register are as follows:

U1CTL.7	**-**	Reserved
U1CTL.6	**U1XINTE**	UART1 transmit interrupt enable.
U1CTL.5	**U1HUNT**	When this bit is set, the UART hunts for a byte with bit 9 set as specified by U1CTL.2 (U1DATA) and only then generates a receive interrupt. All bytes before that are ignored. The "hunt" feature is only available in UART Mode 3 (U1M1=1, U1M0=1).
U1CTL.4	**U1EN**	Enables the activity of UART1.
U1CTL.3	**U1DISC**	Setting the "discard" bit causes UART1 to discard data received with errors such as framing error, parity error or noise error.
U1CTL.2	**U1DATA**	In UART Mode 3 (U1M1=1, U1M0=1), this bit is the data for bit 9.
U1CTL.1	**U1M1**	UART1 Mode 1 (see below)
U1CTL.0	**U1M0**	UART1 Mode 0 (see below)

For both U0CTL and U1CTL the mode control bits operate as follows:

U0M1, U1M1	U0M0, U1M0	Operating Mode	Parity
0	0	Mode 0: 8-bit UART	No Parity
0	1	Mode 1: 9-bit UART	Odd Parity
1	0	Mode 2: 9-bit UART	Even Parity
1	1	Mode 3: 9-bit UART	Data Parity

The UART receive interrupt is asserted whenever a character has been received and is ready to be read out of the UART Buffer (U0BUF/U1BUF). The interrupt is no longer asserted after the 8051 reads the character out of the UART Buffer (U0BUF/U1BUF).

The UART transmit interrupt is asserted whenever there is room in the UART Buffer (U0BUF/U1BUF) for another character to be transmitted. The 8051 software must disable this interrupt by writing "0" to the U0XINTE/U1XINTE bit in the control register (U0CTL.6/U1CTL.6) when there are no characters to be sent.

UART0 STATUS (U0STAT)

Address: 95h *Not Bit Addressable* *Reset value: 00000000b*

U0STAT.7	U0STAT.6	U0STAT.5	U0STAT.4	U0STAT.3	U0STAT.2	U0STAT.1	U0STAT.0
-	**U0BIT9**	**U0OE**	**U0PE**	**U0FE**	**U0NE**	**U0TXEM**	**U0RXV**
	R	R	R	R	R	R	R

The bit definitions for this register are as follows:

U0STAT.7	-	Reserved
U0STAT.6	**U0BIT9**	Ninth bit of received data
U0STAT.5	**U0OE**	Current reception buffer was over-run
U0STAT.4	**U0PE**	Current reception buffer had parity error
U0STAT.3	**U0FE**	Current reception buffer had framing error
U0STAT.2	**U0NE**	Current reception buffer had noise
U0STAT.1	**U0TXEM**	Transmission buffer is empty
U0STAT.0	**U0RXV**	Reception buffer has data

UART1 STATUS (U1STAT)

Address: 9Dh *Not Bit Addressable* *Reset value: 00000000b*

U1STAT.7	U1STAT.6	U1STAT.5	U1STAT.4	U1STAT.3	U1STAT.2	U1STAT.1	U1STAT.0
-	**U1BIT9**	**U1OE**	**U1PE**	**U1FE**	**U1NE**	**U1TXEM**	**U1RXV**
	R	R	R	R	R	R	R

The bit definitions for this register are as follows:

U1STAT.7	-	Reserved
U1STAT.6	**U1BIT9**	Ninth bit of received data
U1STAT.5	**U1OE**	Current reception buffer was over-run
U1STAT.4	**U1PE**	Current reception buffer had parity error
U1STAT.3	**U1FE**	Current reception buffer had framing error
U1STAT.2	**U1NE**	Current reception buffer had noise
U1STAT.1	**U1TXEM**	Transmission buffer is empty
U1STAT.0	**U1RXV**	Reception buffer has data

UART BAUD RATE (U0BR, U0BRH, U1BR, U1BRH)

Address: 97h (U0BR),93h (U0BRH) *Not Bit Addressable* *Reset value: 00h*
9Fh (U1BR)9Bh (U1BRH)

The value in the baud rate register divides the system clock (SYSCLK) to create the sampling clock. The sampling clock is 16 times faster than the bit rate. The formula for calculating the baud rate register value is:

*value = (SysClk / (baud rate * 16)) - 1*

U0BRH and U1BRH are the high bytes of the UART0 and UART1 baud rate, respectively. **These registers are only valid for the K-version of the IC and allow for very low baud rates, if desired.**

UART DATA BUFFER (U0BUF, U1BUF)

Address: 96h (U0BUF), *Not Bit Addressable* *Reset value: 00h*
9Eh (U1BUF)

To transmit an 8-bit character, write it to the U0BUF/U1BUF register. For UART0, a character should be written to U0BUF only when the U0TXEM (U0STAT.1) is "1" (transmit buffer is empty). For UART1, a character should be written to U1BUF only when the U1TXEM (U1STAT.1) is "1".

For UART0, when the U0RXV (U0STAT.0) bit is "1" (receive buffer has data), read the U0BUF register to read an 8-bit character from the UART0 receive buffer. For UART1, when the U1RXV (U1STAT.0) bit is "1", read the U1BUF register to read an 8-bit character from the UART1 receive buffer. Prior to reading the received data, the error status bits in the status register (U0STAT/U1STAT) should be examined.

3.6 D/A PWM

The IRMCF300 has up to three digital-to-analog PWM output ports. All of them are based on 8-bit data with a resolution of 256 counts. Each D/A PWM has an 8-bit data register and an 8-bit timer. The value written into the 8-bit data register sets the modulation index of the PWM output. On reset, the data registers are set to zero, which effectively disables the PWM. Writing any value other than zero enables the PWM and produces the desired waveform.

The I/O Control Register 0 bit 2 (AOPWMF , IOCON0.2) determines the carrier frequency. It can be PCLK/4096 (if IOCON0.2 = 1) or PCLK/1024 (if IOCON0.2 = 0). The period of the PWM is fixed to 256 clock cycles. The data register determines the duty cycle.

D/A PWM OUTPUT DATA (DAD0, DAD1, DAD2)

Address:	*ADh (DAD0), AEh (DAD1), AFh (DAD2)*	*Not Bit Addressable*	*Reset value: 00h*

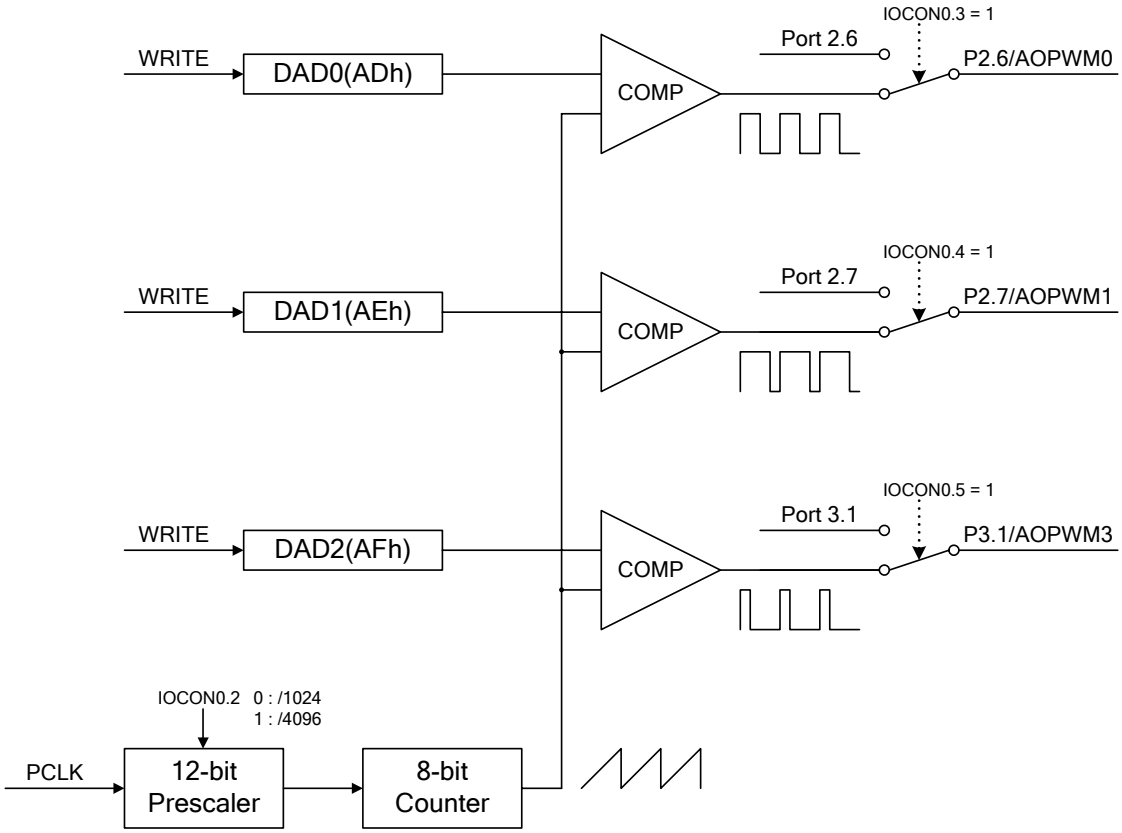


Figure 14. D/A PWM Output

3.7 I²C / SPI Serial Interface

The I²C / SPI interface is used for communication with the external EEPROMs and has a dual purpose. One is to interface to the application data EEPROM. The other is to download the 8051 application program from EEPROM to internal RAM in case of IRMCF300. The function of downloading the 8051 program is described in Section 2. The SFRs used for the I²C interface are summarized in Table 9 and the SFRs used for the SPI interface are summarized in Table 10.

Note that the user must choose between the two interfaces. The SFRs and I/O pins for the I²C and SPI interfaces are shared and only one type of device can be supported for a given application.

Register	Address	Description	Reset Value
I2CAL, I2CAH	B9h, BAh	I²C Transaction address low, high	00h
I2CTD	BBh	I²C Transaction data	00h
I2CCD	BCh	I²C Command data	00h
I2CDA	BDh	I²C Device address	A2h
I2CBC	BEh	I²C Baud rate control	C8h
I2CNF	BFh	I²C Schmitt trigger noise filter	33h

Table 9. SFRs for I²C

Register	Address	Description	Reset Value
I2CTD	BBh	SPI Transaction data	00h
I2CCD	BCh	SPI Command data	00h
I2CBC	BEh	SPI Baud rate control	C8h

Table 10. SFRs for SPI

I²C TRANSACTION ADDRESS REGISTER (I2CAL, I2CAH)

Address: B9h (I2CAL), BAh (I2CAH) Not Bit Addressable Reset value: 0000h

The transaction address register is read/write and is used only for the I²C interface. Two-byte consecutive data forms a 16-bit address (offset) within an EEPROM and specifies the location to be accessed for a read or write operation. For an EEPROM with 256 bytes or fewer, only the low byte address register (I2CAL) is used.

I²C / SPI TRANSACTION DATA REGISTER (I2CTD)

Address: BBh Not Bit Addressable Reset value: 00h

The transaction data register is read/write. This register is used for both I²C and SPI to transfer the data to or from the target EEPROM.

I²C / SPI COMMAND REGISTER (I2CCD)

Address: BCh Not Bit Addressable Reset value: 00000000b

I2CCD.7	I2CCD.6	I2CCD.5	I2CCD.4	I2CCD.3	I2CCD.2	I2CCD.1	I2CCD.0
BUSY	**-**	**-**	**SPISEL**	**SPIEN**	**CMD2**	**CMD1**	**CMD0**
R			R/W	R/W	R/W	R/W	R/W

The bit definitions for this register are as follows:

I2CCD.7	**BUSY**	This bit is set to "1" when the I^2C or SPI interface is busy with an operation.
I2CCD.6	**-**	*Reserved.*
I2CCD.5	**-**	*Reserved.*
I2CCD.4	**SPICS1**	SPI CS1 enable. 0: CS1 is not used, 1: CS1 is used. If an SPI device is configured at CS1, this bit should always be set to 1, even when issuing commands for the device at CS0. Always set this bit to 0 for I^2C.
I2CCD.3	**SPIEN**	SPI interface enable. 0: enable I^2C, 1: enable SPI.
I2CCD.2	**CMD2**	SPI / I^2C command bit 2.
I2CCD.1	**CMD1**	SPI / I^2C command bit 1.
I2CCD.0	**CMD0**	SPI / I^2C command bit 0.

SPI / I^2C commands are initiated by writing to the command register. The command type is encoded in bits CMD2 – CMD0, as follows:

CMD2	CMD1	CMD0	I^2C Command Description	SPI Command Description
0	0	0	I^2C reset	Byte write CS0, CS0 low on completion
0	0	1	8-bit address random write	Byte read CS0, CS0 low on completion
0	1	0	8-bit address random read	Byte write CS1, CS1 low on completion
0	1	1	16-bit address random write	Byte read CS1, CS1 low on completion
1	0	0	16-bit address random read	Byte write CS0, CS0 high on completion
1	0	1	Current address read	Byte read CS0, CS0 high on completion
1	1	0	Next address read	Byte write CS1, CS1 high on completion
1	1	1	Previous address read	Byte read CS1, CS1 high on completion

I^2C DEVICE ADDRESS REGISTER (I2CDA)

Address: BDh *Not Bit Addressable* *Reset value: 10100010b*

I2CDA.7	I2CDA.6	I2CDA.5	I2CDA.4	I2CDA.3	I2CDA.2	I2CDA.1	I2CDA.0
A6	**A5**	**A4**	**A3**	**A2**	**A1**	**A0**	**-**
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R

This register is used for the I^2C interface only. Bits A6 through A3 must always be set to "1010" in order to address any EEPROM device on the I^2C bus. Bits A2, A1, and A0 select an individual EEPROM device.

For proper operation of the 8051 boot sequence at power up, a 32k byte or 64k byte device (e.g., 24C256 or 24C512) must be provided with IRMCF300. These devices typically have the upper five bits of address hard-wired to "10100" and have two configurable address pins A1 and A0. The device must be configured so that the device address A1 pin is pulled down (A1 = 0) and A0 is pulled up (A0 = 1). This sets the device address to match the default value of the I2CDA register.

Up to three additional EEPROM devices can be connected to the same I^2C bus. These are typically used for storage of application specific data (e.g., motor tuning parameters). These EEPROM devices can be configured for any 3-bit address except 001. Before accessing an EEPROM device, the 8051 software must select the appropriate device address by writing to bits A2, A1 and A0 of the I2CDA register.

A2	A1	A0	Device
0	0	1	Default 64k byte device for 8051 program storage, (e.g., 24C512)
0 or 1	0	0	Any other device for user parameter storage.
0 or 1	1	0	
0 or 1	1	1	

I²C / SPI BAUD RATE CONTROL (I2CBC)

Address:	*BEh*	*Not Bit Addressable*	*Reset value:*	*00010000b*

The baud rate control register configures the bit rate for I²C or SPI serial communication. The clock frequency of the SCL pin is determined by this register value. The value in this register feeds the high-order 8 bits of a 12 bit frequency divider and can be calculated using the equation:

$$I2CBC = \frac{1}{16} \cdot \left(\frac{SysClk}{2 \cdot f_{SCL}} - 1 \right)$$

where *SysClk* is the system clock rate and f_{SCL} is the SCL frequency

The default value is 16 (0x10), which generates a 97 kHz bit rate at the default clock rate of 50 MHz. For the boot procedure performed at power up this SFR is not used (see Section 2 for details).

I²C NOISE FILTER (I2CNF)

Address:	*BFh*	*Not Bit Addressable*	*Reset value:*	*00010100b*

The noise filter register configures the I²C noise filter time constant and is not used for the SPI interface. The valid range of values is: 0 – 31 (5 bits). The filter is a one bit FIR filter clocked by the system clock. Maximum filter delay is calculated as follows:

*Delay time = I2CNF value * 15 nanoseconds*

The default value is 20 (0x14), which provides a 300 nanoseconds filter time constant and delay. Any noise with a pulse width smaller than 300 nanoseconds will be rejected.

3.7.1 Command Descriptions for the I²C Interface

This section describes how to use the I²C interface to read and write typical I²C EEPROM devices.

3.7.1.1 Reset Command

After power-on or reset, a reset command should be issued to the I2C interface before performing any read or write operations. The I²C reset can also be used to force a device on the bus to a known state after an error condition such as a power drop (brown-out), strong electrical noise, or a hardware reset during an I²C operation. An I²C reset is automatically generated if the boot procedure (Section 2) fails.

The I²C reset operation executes a Start, nine clock cycles with high data, and then a Stop. If a device has failed to relinquish the I²C bus, this sequence should bring the device to a stopped state and allow the bus to function normally.

3.7.1.2 Read and Write Commands

Use the 8-bit address commands for 256 byte EEPROM devices and the 16-bit address commands for larger devices. The current address read command is a no-address transaction, where the address is automatically incremented from the previous transaction.

Note: After issuing any read command, software must wait for the data in the I2CTD register to become valid before reading that register. Valid data in the I2CTD register is indicated by a “0” value in the BUSY bit of the I2CCD register.

The typical sequence used from the 8051 to **write** to EEPROM is as follows:

1. Reset the I^2C bus by writing "0" to the command register (I2CCD).
2. Select the device address by writing to the I2CDA register.
3. Wait for the BUSY bit to be cleared in the I2CCD register.
4. Write the desired EEPROM offset to the transaction address high and low registers (I2CAL, I2CAH).
5. Write the data byte into the transaction data register (I2CTD)
6. Select an 8-bit or 16-bit write operation by writing the appropriate value to the command register (I2CCD).
7. Wait for the BUSY bit to be cleared in the I2CCD register.
8. Repeat steps 4 – 7 to write additional data bytes.

The typical sequence used from the 8051 to **read** from EEPROM is as follows:

1. Reset the I^2C bus by writing "0" to the command register (I2CCD).
2. Select the device address by writing to the I2CDA register.
3. Wait for the BUSY bit to be cleared in the I2CCD register.
4. Write the desired EEPROM offset to the transaction address high and low registers (I2CAL, I2CAH).
5. Select an 8-bit or 16-bit read operation by writing the appropriate value to the command register (I2CCD).
6. Wait for the BUSY bit to be cleared in the I2CCD register.
7. Read the data byte from the transaction data register (I2CTD)
8. Repeat steps 4 – 7 to read additional data bytes.

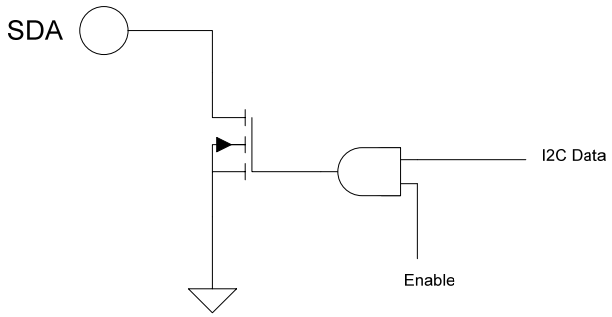


Figure 15. I^2C Pin Structure

3.7.2 Command Descriptions for the SPI Interface

This section describes how to use the SPI interface to read and write typical SPI EEPROM devices.

Each SPI command reads or writes a single byte of data. Typical SPI devices define instruction sequences composed of multiple bytes. A series of SPI commands is required to issue a single multi-byte instruction sequence to an SPI device. When issuing multi-byte instructions, use the set of command codes that specifies "CSn low on completion" for all but the last command of the sequence. For the last command, use a command code from the set that specifies "CSn high on completion". The SPI device recognizes the end of the instruction sequence when the CS signal returns to the high state. For single-byte instructions (such as write enable), always use a command code that sets CS high on completion.

3.7.2.1 Read Instructions

Reading a byte of data from an SPI EEPROM device typically requires a sequence of four commands to be issued to the SPI interface: three write commands are issued to send the read instruction and the 16-byte address, followed by a read command to read the data. Reading multiple bytes of data from sequential locations can usually be accomplished by writing the read instruction and 16-byte starting address, followed by a series of read commands, one for each data byte. Only the last read command sets the CS signal high on completion.

Note: After issuing any read command, software must wait for the data in the I2CTD register to become valid before reading that register. Valid data in the I2CTD register is indicated by a "0" value in the BUSY bit of the I2CCD register.

The typical sequence used from the 8051 to **read** from an SPI EEPROM is as follows:

1. Write a Read instruction code (specific to the SPI device) to the I2CTD register.
2. Write a byte write command code (CS low on completion) to the I2CCD register.
3. Wait for the BUSY bit to be cleared in the I2CCD register .
4. Write the high byte of the 16-bit EEPROM address to the I2CTD register.
5. Write a byte write command code (CS low on completion) to the I2CCD register.
6. Wait for the BUSY bit to be cleared in the I2CCD register .
7. Write the low byte of the 16-bit EEPROM address to the I2CTD register.
8. Write a byte write command code (CS low on completion) to the I2CCD register.
9. Wait for the BUSY bit to be cleared in the I2CCD register.
10. Write a byte read command code (CS high on completion) to the I2CCD register.
11. Wait for the BUSY bit to be cleared in the I2CCD register.
12. Read the data byte from the I2CTD register.

To read more than one byte, follow the steps above, but use a read command code with CS low on completion at step 10. Repeat steps 10 - 12 for each additional data byte to be read. For the last read only, use the read command code with CS high on completion.

3.7.2.2 Write Instructions

Writing to an SPI EEPROM device typically requires the device to be write enabled first as a separate instruction (setting the CS signal high on completion). The write instruction follows, which typically requires a sequence of four write command to send the write instruction, the 16-bit address and the data byte. Writing multiple bytes of data to sequential locations can usually be accomplished by sending the write instruction and 16-byte starting address, followed by a series of write commands, one for each data byte. Only the last write command sets the CS signal high on completion.

Note: After issuing any write command, software must wait for the command to complete before writing another data byte to the I2CTD register. Command completion is indicated by a "0" value in the BUSY bit of the I2CCD register.

The typical sequence used from the 8051 to **write** to an SPI EEPROM is as follows:

1. Write a Write Enable instruction code (specific to the SPI device) to the I2CTD register.
2. Write a byte write command code (CS high on completion) to the I2CCD register.
3. Wait for the BUSY bit to be cleared in the I2CCD register .
4. Write a Write instruction code (specific to the SPI device) to the I2CTD register.
5. Write a byte write command code (CS low on completion) to the I2CCD register.
6. Wait for the BUSY bit to be cleared in the I2CCD register .
7. Write the high byte of the 16-bit EEPROM address to the I2CTD register.
8. Write a byte write command code (CS low on completion) to the I2CCD register.
9. Wait for the BUSY bit to be cleared in the I2CCD register .
10. Write the low byte of the 16-bit EEPROM address to the I2CTD register.
11. Write a byte write command code (CS low on completion) to the I2CCD register.
12. Wait for the BUSY bit to be cleared in the I2CCD register.
13. Write the data byte to the I2CTD register.
14. Write a byte write command code (CS high on completion) to the I2CCD register.

To write more than one byte, follow the steps above, but use a write command code with CS low on completion at step 14. Repeat steps 12 - 14 for each additional data byte to be written. For the last write only, use the write command code with CS high on completion.

4 Motion Control Engine

The Motion Control Engine (MCE) is a collection of hardware modules—the building blocks necessary to implement high efficiency sinusoidal sensorless control for Permanent Magnet motors. The MCE Library provides a graphical representation of these modules.

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 user can select and connect the library blocks to form a control algorithm. Using the MCE library in the MATLAB/Simulink™ environment, the user can design custom control loops (closed loop sensorless current control, closed loop speed control, etc.) based on application requirements. A graphic compiler analyses the completed design and automatically translates it into a sequence of MCE-specific machine code for integration with the IRMCx300. Operating on the IRMCF300 , the MCE machine code essentially customizes the device for the user's specific application requirements.

The two basic types of hardware resources available on the IRMCF300 are Motion Peripherals and Control blocks.

Motion peripherals process analog and digital signals and interface to "the outside world"—hardware external to the IRMCF300 IC. Examples are the Low Loss Space Vector PWM module, A/D converter module, and single shunt current reconstruction module. These modules are colored yellow throughout this document and the design entry tool (Matlab/Simulink™) to distinguish them from other hardware 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. The motion peripheral blocks are described in Section 4.3. Registers used to configure and monitor the operation of the motion peripheral blocks from an 8051 application or a host system are described in Section 4.4.

Control 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. Control block signals can be connected to another Control Block or to a Motion Peripheral module. Control Blocks are colored green (for math) or blue (all others) throughout this document and the design entry tool (MATLAB/Simulink™) to distinguish them from the Motion Peripherals. The control blocks are described in Section 4.2. 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. The support blocks are described in Section 6.3.

Table 11 summarizes all the blocks in the MCE library. For each block, the table entry references the document section that describes the block in detail. For more information about using the MCE library in MATLAB/Simulink, refer to Section 6.3.

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

Module Category	Module Name	Description	Document Section
Control Blocks – Frequency Domain	PI	Proportional plus integral	4.2.1.1
	LOWPASS_FILT	First order low pass filter	4.2.1.2
	HIGHPASS_FILT	First order high pass filter (differentiator and lag)	4.2.1.3
Control Blocks – Coordinate Transformation	VECROT	Vector rotator	4.2.2.1
	CLARK	Inverse Clark transformation	4.2.2.2
Control Blocks – Utility	RAMP	Linear ramp function	4.2.3.2
	ATAN	Arc tangent lookup table	4.2.3.3
	LIMIT	Limit function	4.2.3.1
	FUNCTION_BLOCK	Five-input two-dimensional function	4.2.3.4
	COMPARATOR	Compare two inputs	4.2.3.5

Module Category	Module Name	Description	Document Section
Control Blocks – Utility	SWITCH	Switch between two inputs based on a third	4.2.3.6
	BIT_LATCH	Bit latch	4.2.3.7
	PEAK_DETECT	Peak detect	4.2.3.8
	TRANSITION	One shot pulse generator	4.2.3.9
	INTEGRAL2	Integral with limit	4.2.3.10
	PFC_FFD	PFC Feed Forward	4.2.3.11
Control Blocks – Math	MUL_DIV	Multiply with extraction (signed/unsigned)	4.2.4.6
	DIVIDE	Divide (signed/unsigned)	4.2.4.7
	SUM	Adder	4.2.4.2
	ACCUMULATOR	Accumulator	4.2.4.3
	COUNTER	Counter with limit	4.2.4.4
	DIFF	Difference	4.2.4.1
	SHIFT	Multiply by a power of two	4.2.3.6
	AND	Logical AND (16 bit)	4.2.4.10
	OR	Logical OR (16 bit)	4.2.4.11
	XOR	Logical XOR (16 bit)	4.2.4.12
	NOT	Logical NOT (16 bit)	4.2.4.8
	NEGATE	Logical NEGATE (16 bit)	4.2.4.9
Motion Peripheral Blocks	SENSORLESS_FOC	Sensorless field orientation control	4.3.1
	LOWLOSS_SVPWM	Low loss space vector modulator	4.3.5
	SINGLE_I_SHUNT	Single shunt current reconstruction	4.3.2
	DC_BUS_VOLTAGE	DC bus voltage monitor	4.3.3
	A_D	A/D converters	4.3.4
	FAULTS	Drive fault status	4.3.6
	MCE_FAULT	MCE fault generator	4.3.7
	PFC_PWM	PFC pulse width modulator	4.3.8
	PFC_SENSE	PFC current/voltage reconstruction	4.3.9
Custom Support Blocks	Configure PWM	Hierarchical connection to PWM	6.3.1.1
	Configure Control Loop	Hierarchical connection to control loops	6.3.1.1
	Read Register	Output to host/8051 interface	6.3.1.2
	Write Register	Inputs from host/8051 interface	6.3.1.2
	MCE Compiler	Shortcut to the MCE Compiler	6.3.1.5
	Host Register Summary	Read and Write register summary display	6.3.1.5
Standard Simulink Support Blocks	Enabled Subsystem	Hierarchical organization block	6.3.2.1
	Constant	Defines a constant input value	6.3.2.2
	Scope	Defines an output for MCEDEsigner trace function	6.3.2.3
	Input Port	Defines an input for a custom macro block	6.3.2.4
	Output Port	Defines an output for a custom macro block	6.3.2.5
	Goto	Off-page connector (output)	6.3.2.6
	From	Off-page connector (input)	6.3.2.6
	Unit Delay	PWM cycle signal delay	6.3.2.7

Table 11. MCE Library Elements

4.1 Rotating Frame Notation and Conventions

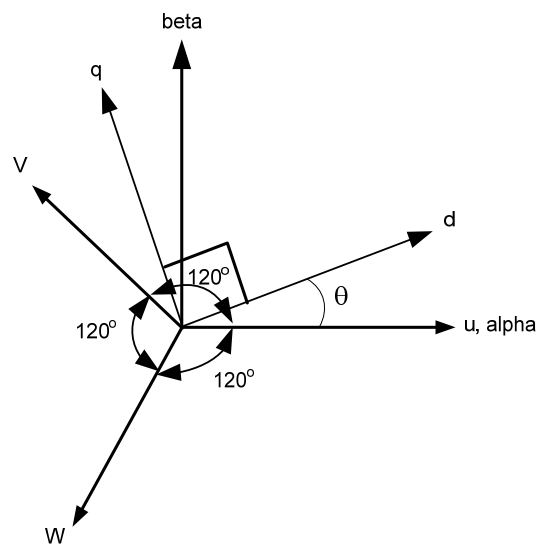
Throughout this document, the following convention is used to describe input and output notation with regard to the rotating frame for field orientation control.

The stationary frame variables "alpha" and "beta" are orthogonal. The three-phase stationary frame variables "u", "v", and "w" are 120 degree apart. "alpha" is aligned to "u". The synchronously rotating frame variables "d" and "q" are relative to the moving angle of θ.

Forward Vector Rotation:

$$\begin{bmatrix} alpha \\ beta \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} d \\ q \end{bmatrix}$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} \end{bmatrix} \cdot \begin{bmatrix} alpha \\ beta \end{bmatrix}$$



Inverse Vector Rotation:

$$\begin{bmatrix} alpha \\ beta \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

$$\begin{bmatrix} d \\ q \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \cdot \begin{bmatrix} alpha \\ beta \end{bmatrix}$$

4.2 Control Blocks

This section describes the MCE control blocks, which include frequency domain and coordinate transformation modules, general utility functions and math operations. In the Simulink MCE libraries, the math blocks are found in the Math library and the remaining control blocks are located in the Control library.

The connections between the various control blocks in the MCE design determine their order of execution, and they execute sequentially (not in parallel). A worst case timing estimate is shown for each control block. The designer should take care to ensure that the total execution time for all control blocks in the design does not exceed the configured PWM preiod.

4.2.1 Frequency Domain Blocks

4.2.1.1 PI – Proportional Plus Integral

The PI block performs the following function in the s-domain:

$$\frac{Output}{Input} = Kp + \frac{Ki}{s}$$

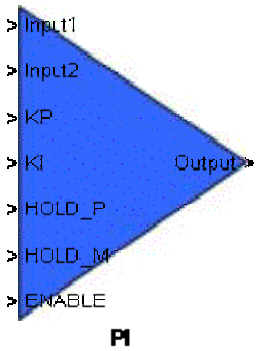
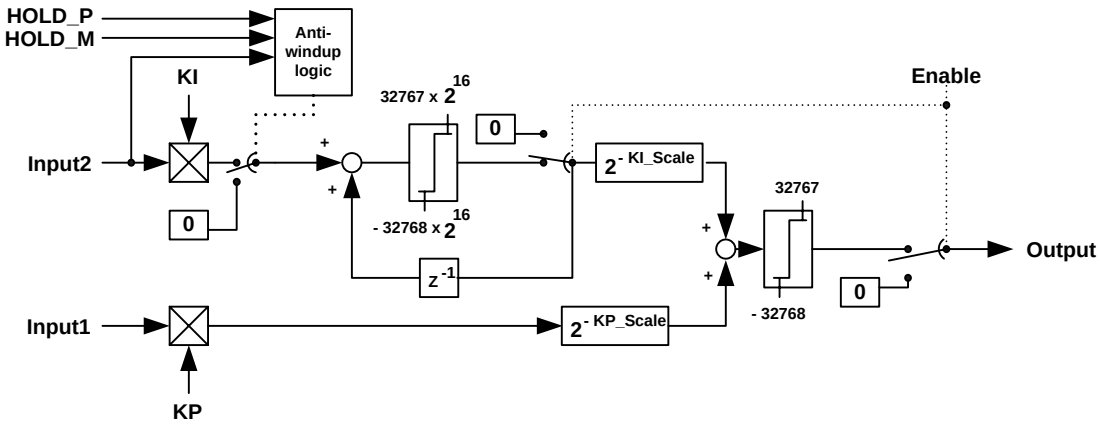


Figure 16. PI Block

And in the discrete domain:



In terms of pseudo code, the PI regulator can be represented by:

```
KP_Prod(n) = INPUT1(n)   x   KP
[s31:0]        [s15:0]          [s15:0]
KP_Prod(n) =  KP_Prod(n) right shift  KP_SCALE
[s31:0]         [s31:0]

If((INPUT2(n) > 0 and HOLD_P) or (INPUT2(n) < 0 and HOLD_M))  // antiwindup
     [s15:0]                [1]                [s15:0]                    [1]
        KI_Prod(n) = 0
        [s31:0]
Else
        KI_Prod(n)   = INPUT2(n)   x   KI
        [s31:0]        [s15:0]            [s15:0]

If(ENABLE)
          Int(n)    =     KI_Prod(n)  +    Int(n-1)
          [s31:0]          [s31:0]           [s31:0]
          Protect sign 32-bit overflow on Int(n)
Else
          Int(n) = 0
          [s31:0]

KI_Int(n)      =  Int(n) right shift KI_SCALE
[s31:0]            [s31:0]

Temp1(n)  =    KP_Prod(n)  +    KI_Int(n)
[s31:0]           [s31:0]             [s31:0]
 Protect 32 bit overflow on Temp1(n)
Temp2(n) = Limit Temp1(n) to [s15:0]
 [s15:0]

If(ENABLE)
      OUTPUT  =  Temp2(n)
       [s15:0]        [s15:0]
Else
       OUTPUT = 0
```

The scalers (KP_Scaler, KI_Scaler) are accessible by double clicking the PI block in the MODEL file. The scalers must be from 0 – 31. Note: The scalers can only be updated during compile time (MCE compiler).

Also note that when the output limits at the maximum or minimum 16-bit number, the internal 32-bit recursive register (I_output) continues to accumulate. To prevent this "wind-up" action, the PI block should be used in conjunction with the LIMIT block. The integration can be halted by feeding the SATP or SATM outputs of LIMIT back to HOLD_P and HOLD_M inputs of PI, respectively.

Signal name	Description	I/O	Type
Input1	Input for proportional path	Input	16 bit, signed integer
Input2	Input for integral path	Input	16 bit, signed integer
KP	Proportional gain	Input	16 bit, signed integer [1]
KI	Integral gain	Input	16 bit, signed integer [1]
HOLD_P	Hold Integrator Positive going	Input	Boolean, 0 = no hold 1 = hold integrator positive going
HOLD_M	Hold Integrator Negative going	Input	Boolean,

			0 = no hold 1 = hold integrator negative going
ENABLE	Enable/Reset	Input	Boolean, 0 = reset all recursive data and output 1 = normal operation
Output	Output	Output	16 bit, signed integer

Table 12. PI User Inputs and Outputs

Signal name	Description	I/O	Type
KP_Scale	Scale factor for Kp gain	Input	8 bit scaler [1]
KI_Scale	Scale factor for Ki gain	Input	8 bit scaler [1]

Table 13. PI System Inputs and Outputs

Note 1. The range of KP and KI is from 0 to 32767 and can be interpreted as simple floating format. For KP and KI, the system uses a simple floating point format as shown below. The mantissa is a 16-bit unsigned integer and scaler is an 8-bit value, however only **5** LSB is being used (0 – 31 range). The scaler represents a negative power of two applied to the mantissa.

Mantissa:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Scaler positions decimal point

Scaler:

7	6	5	4	3	2	1	0
0	0	0					

Data range:

$1 * 2^{-31} – 65,535$

Status	Clock cycles
Min. case	34
Max. case	63

Table 14. PI Execution Time

4.2.1.2 LOWPASS_FILT – First Order Low Pass Filter

The low pass filter block performs the following function:

$$\frac{Output}{Input} = \frac{1}{1 + S \bullet Tau}$$

where Tau is the filter time constant (1/Wc).

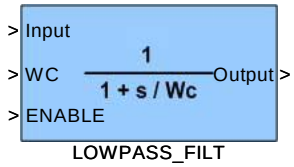
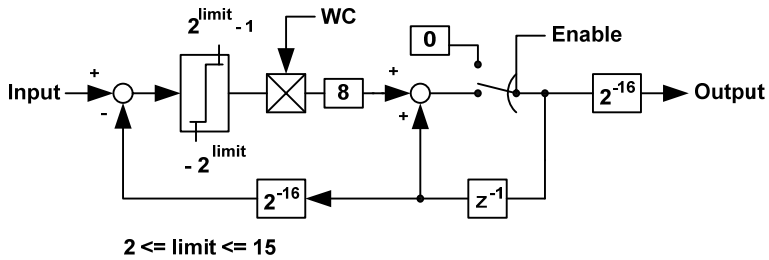


Figure 17. LOWPASS_FILT Block

Realization in the discrete domain:



In terms of Psuedo code, the lowpass filter can be realized as:

```
Temp1(n)   =   Input(n)    -   Output(n-1)
 [s15:0]        [s15:0]          [s15:0]
 Overflow protect Temp1(n) to 16-bit signed
 Temp1(n)  =  Temp1(n)  x   WC
  [s31:0]         [s15:0]          [s15:0]
 Temp1(n) = Shift Temp1(n) left 3 bit
  [s31:0]              [s31:0]

  If (ENABLE)
          Int(n)   =  Int(n)  +  Temp1(n)
          [s31:0]     [s31:0]     [s31:0]
           Overflow protected to 32 bit sign
  Else
          Int(n) = 0

  Output(n) =  Int(n) [s31:16]
  [s15:0]
```

Double clicking on the block in the Simulink model file will allow the designer to access a limit value parameter. The parameter restricts the difference between the filtered output and the new input to the range [-2^{LIMIT}, $2^{LIMIT} - 1$]. The parameter can be any value between 2 and 15, with default value of 15. Note: The LIMIT value parameter can only be updated during compile time (MCE compiler).

Relationship between the actual filter time constant (sec.) and the configurable parameter WC (input) is given by:

$$Tau = \frac{DeltaT}{WC} \times 2^{13} \quad \text{[sec]}$$

where: Tau is defined as the filter time constant in sec. which also correpsonds to 1/Wc as shown in Figure 17.
WC (in digital counts) is the configurable input of the filter block (Figure 17).
DeltaT is the sampling time of the filter in sec..

Note: Theorectically, the minimum filter time constant should be larger than 2 times the filter sampling time to prevent digital filter instability. The filter gain and phase characteristics is shown in Figure 18 where the normalized frequency 1 corresponds to 1/Tau (rad./sec).

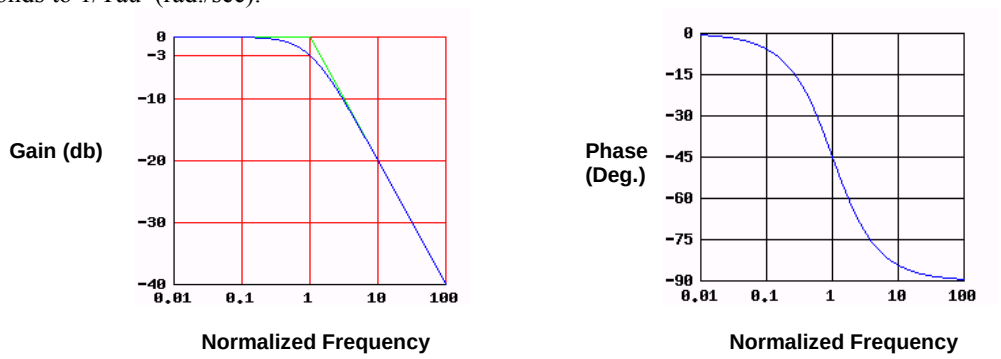


Figure 18. LOWPASS_FILT frequency response

Signal name	Description	I/O	Type
ENABLE	1 – enable filter 0 – disable filter, output = 0	Input	Boolean,
WC	Filter bandwidth in digital counts.	Input	16 bit, signed integer [1]
Input	Filter input	Input	16 bit, signed integer
Output	Filter Output (rounding)	Output	16 bit, signed integer

Table 15. LOWPASS_FILT User Inputs and Outputs

Note 1: The allowable data range of WC is 0 to 2^{13}. Numerical overflow will occur if WC is outside the specified range. To pass Input unchanged to Output (turn off filter action), set WC to 2^{13}.

Status	Clock cycles
Max. case	–33

Table 16. LOWPASS_FILT Execution Time

4.2.1.3 HIGHPASS_FILT – First Order High Pass Filter

The HIGHPASS_FILT block performs the following S-domian function:

$$\frac{Output}{Input} = \frac{s \bullet Tau}{1 + s \bullet Tau} \quad \text{where Tau = 1/Wc}$$

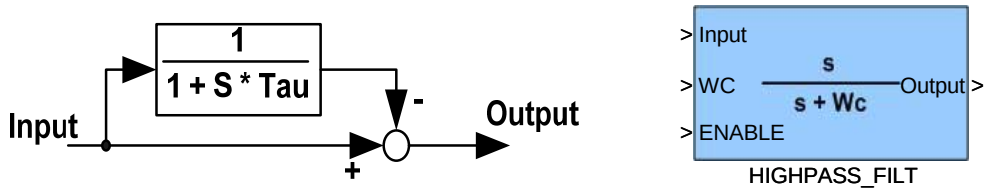


Figure 19. HIGHPASS_FILT Block

The highpass filter can be realized by the following psuedo code:

```
Temp1(n)   =   Input(n)    -   Output(n-1)
 [s15:0]        [s15:0]          [s15:0]
 Overflow protect Temp1(n) to 16-bit signed
 Temp1(n)  =  Temp1(n)  x   WC
  [s31:0]         [s15:0]          [s15:0]
 Temp1(n) = Shift Temp1(n) left 3 bit
  [s31:0]              [s31:0]

 If (ENABLE)
          Int(n)   =  Int(n)  +  Temp1(n)
          [s31:0]    [s31:0]    [s31:0]
           Overflow protected to 32 bit sign
 Else
          Int(n) = 0

  Temp2(n) = Input(n) – Int(n)[s31:16]
   [s15:0]       [s15:0]      [s15:0]
   Temp3(n) = Overflow protect Temp2 to 16-bit signed
   If(ENABLE)
          Output(n) = Temp3(n)
          [s15:0]         [s15:0]
   Else
           Output(n) = 0
```

When the ENABLE signal is low, the output of the block is zero. In order for the Output to pass the Input signal unchanged, set WC = 0 and pulse the ENABLE input to clear any recursive data. WC should be in the range [0, 2^{13}], otherwise Output could have over/under shoot spikes or even large oscillations.
The high pass filter is constructed by a low pass filter and a summing as shown in Figure 19. The low pass filter time contant (Tau) determines the high pass filter cutoff frequency (Wc).
Relationship between the actual filter time constant (sec.) and the configurable parameter WC (input) is given by:

$$Tau = \frac{DeltaT}{WC} \times 2^{13} \quad \text{[sec]}$$

where: Tau is defined as the low pass filter time constant in sec.
WC (in digital counts) is the configurable input of the filter block (Figure 19).

DeltaT is the sampling time of the filter in sec..

Note: Theorectically, the minimum filter time constant should be larger than 2 times the filter sampling time to prevent digital filter instability.

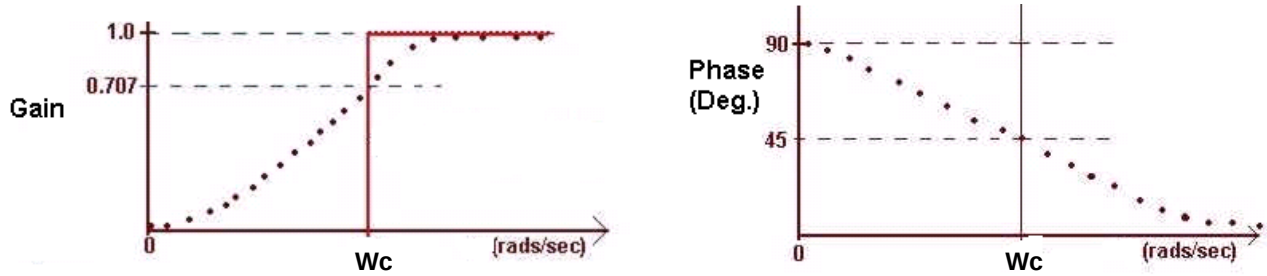


Figure 20. HIGHPASS_FILT frequency response

Figure 20 shows the frequency response of the high pass filter, the filter cutoff frequency (Wc in rad./sec) corresponds to 1/Tau.

Signal name	Description	I/O	Type
Input	Filter input	Input	16 bit, signed integer
WC	Filter bandwidth	Input	16 bit, unsigned integer [1]
ENABLE	1 – enable filter 0 – disable filter, output = 0	Input	Boolean,
Output	Filter output	Output	16 bit, signed integer

Table 17. HIGHPASS_FILT User Inputs and Outputs

Note 1: The allowable data range of WC is 0 to 2^{13}. Numerical overflow will occur if WC is outside the specified range. To pass Input unchanged to Output (turn off filter action), set WC to 0.

Status	Clock cycles
Max. case	87

Table 18. HIGHPASS_FILT Execution Time

4.2.2 Coordinate Transformation Blocks

4.2.2.1 VECROT – Vector Rotation

The VECROT block performs the following function in the discrete domain:

$$\begin{bmatrix} Output1(n) \\ Output2(n) \end{bmatrix} = 1.64676 \cdot \begin{bmatrix} \cos\theta(n) & -\sin\theta(n) \\ \sin\theta(n) & \cos\theta(n) \end{bmatrix} \cdot \begin{bmatrix} Input1(n) \\ Input2(n) \end{bmatrix}$$

Where $0 \le \theta(n) \le 2\pi$, $\theta(n) = \text{THETA(n)} \times 2\pi/4096$.

The VECROT block follows 3xx series vector rotation convention which is described in section 4.1.

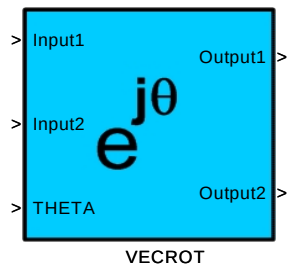


Figure 21. VECROT Block

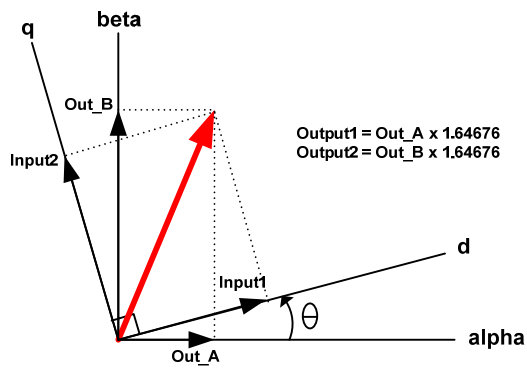


Figure 22. VECROT vector interpretation

Signal name	Description	I/O	Type
Input1	Input1 aligns with d axis of Figure 22	Input	16 bit, signed integer [1]
Input2	Input 2 aligns with q axis of Figure 22.	Input	16 bit, signed integer [1]
THETA	Rotator angle (4096 digital counts = 2π)	Input	16 bit, signed integer [2]
Output1	Output1 aligns with alpha axis of Figure 22.	Output	16 bit, signed integer
Output2	Output2 aligns with beta axis of Figure 22.	Output	16 bit, signed integer

Table 19. VECROT Inputs and Outputs

Note 1: range of input 1 and 2 is less than 16-bit: $-2^{12} \le Input1 \le 2^{12}$, $-2^{12} \le Input2 \le 2^{12}$

Note 2: THETA mapping: $0 \le THETA \le 4096$ digital counts maps to 2π

Status	Clock cycles
Nominal	45

Table 20. VECROT Execution Time

4.2.2.2 CLARK – Inverse Clark Transformation

The inverse Clark block is a 3-phase to 2-phase transformation.

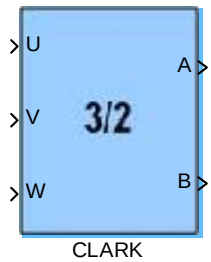


Figure 23. CLARK Block

The module implements the following equation in the discrete domain:

$$\begin{bmatrix} A(n) \\ B(n) \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{3}} & -\frac{1}{\sqrt{3}} \end{bmatrix} \cdot \begin{bmatrix} U(n) \\ V(n) \\ W(n) \end{bmatrix}$$

Pseudo code for Inverse clark is realized by:

```
 A(n)     =      U(n)
[s11:0]          [s11:0]
Temp1(n) =  V(n)     –     W(n)
[s12:0]       [s11:0]        [s11:0]
Temp2(n) =  shift right 11 (Temp1(n)   *    2365)
 [s14:0]                          [s12:0]          [s12:0]
 B(n)  =  Temp2(n) [12:1]
 [s11:0]
```

Figure 24 shows the CLARK vector operation, the relationship (2/3 scaler) of B and B”is used to preserve power invariant transformation.

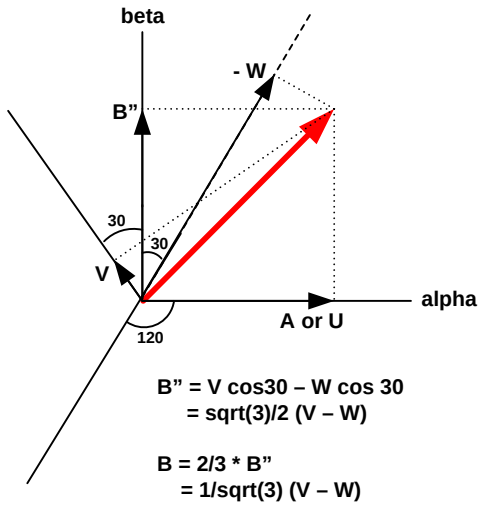


Figure 24. CLARK vector interpretation

Signal Name	Description	I/O	Type
U	Phase U (one of 3-phase)	Input	16 bit, signed integer [1]
V	Phase V (one of 3-phase)	Input	16 bit, signed integer [1]
W	Phase W (one of 3-phase)	Input	16 bit, signed integer [1]
A	A (orthogonal set with B)	Output	16 bit, signed integer
B	B (orthogonal set with A)	Output	16 bit, signed integer

Table 21. CLARK Inputs and Outputs

Note 1: range of U, V and W should be restricted between $\pm 2^{12}$.

Status	Clock cycles
Nominal	21

Table 22. CLARK Execution Time

4.2.3 Utility Blocks

4.2.3.1 LIMIT

Figure 25 shows the inputs and outputs of the LIMIT block, which are also described in Table 23. Note that if LIMIT_P < LIMIT_M then OUT = LIMIT_M and SATM = 1 for all values of Input. Also, if LIMIT_P = LIMIT M, then OUT = LIMIT_M and SATM = SATP = 1 for all values of Input.

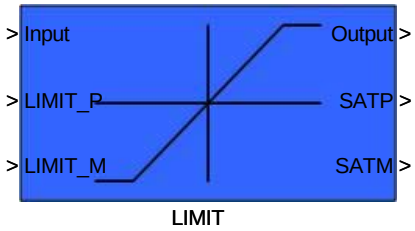


Figure 25. LIMIT Block

The LIMIT block performs the following function:

```
Output = Input
SATM = 0
SATP = 0
If (Input <= LIMIT_M ) then
        Output = LIMIT_M
        SATM = 1
        SATP = 0
If (Input >= LIMIT_P) then
        Output = LIMIT_P
        SATP = 1
        SATM = 0
```

Note: Input, Output, LIMIT_M and LIMIT_P are [s15:0].

Signal name	Description	I/O	Type
Input	Input	Input	16 bit, signed integer
LIMIT_P	Positive Limit Threshold	Input	16 bit, signed integer
LIMIT_M	Negative Limit Threshold	Input	16 bit, signed integer
SATP	Status flag for indicating upper limit (Limit_P) saturation	Output	Boolean, 0 = not saturated 1 = saturated
SATM	Status flag for indicating lower limit (Limit_M) saturation	Output	Boolean, 0 = not saturated 1 = saturated
Output	Output	Output	16bit, signed integer

Table 23. LIMIT Inputs and Outputs

Status	Clock cycles
Max. case	24

Table 24. LIMIT Execution Time

4.2.3.2 RAMP – Linear Ramp

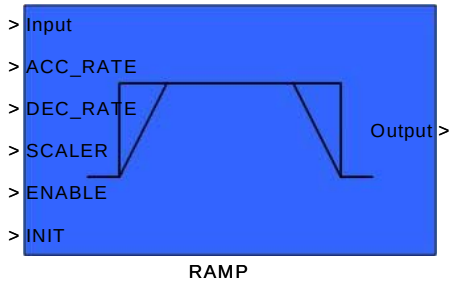
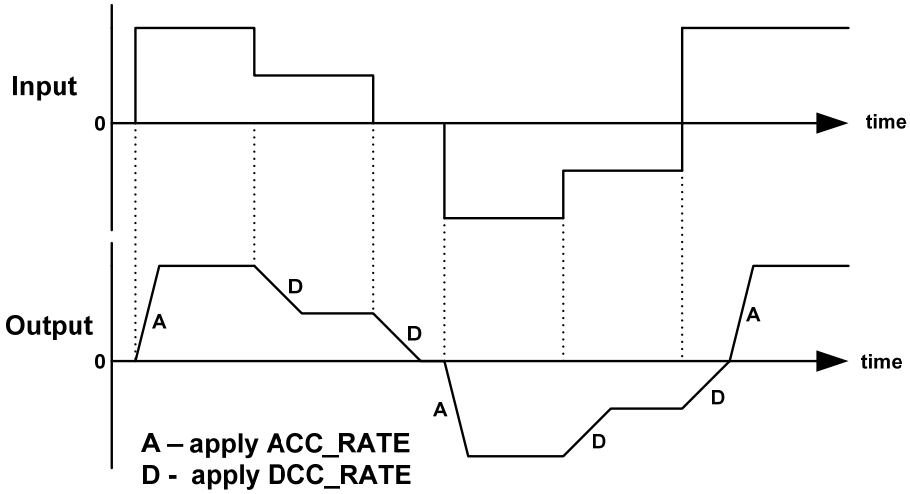


Figure 26. RAMP Block

The RAMP block provides rate limiting function. The rate of change of input is limited by configurable parameters ACC_RATE and DEC_RATE of the RAMP block inputs. Figure below illustrates (ACC_RATE = 4x DCC_RATE) the application of rate limiting under step changes of the input.



Example: ACC_RATE = 4 x DCC_RATE

The scaling of ACC_RATE and DEC_RATE is given by:

$$\text{Rate limit} = \frac{ACC_RATE}{2^{Scaler} \times \Delta T} \quad \text{[digital counts per sec.]}$$

$$\text{or} \quad = \frac{DEC_RATE}{2^{Scaler} \times \Delta T} \quad \text{[digital counts per sec.]}$$

Where ΔT = sampling time of the RAMP block

Note: It is not recommend to change SCALER when the RAMP block is enabled. Doing so could result in a discontinuous jump of the Output value.

The ramp function can be realized by the following Pseudo code:

```
if(ENABLE)
    Delta  = Input x 2^SCALER – Int32(n)
```

```
        [s31:0]  [s15:0]            [s31:0]
        If((Int32(n)>0 and Delta>0) or (Int32(n)<0 and Delta<0))
              RATE = ACC_RATE
        Elseif((Int32(n)<0 and Delta>0) or (Int32(n)>0 and Delta<0))
              RATE = DCC_RATE

        If(Delta > RATE) Delta = RATE
        Elseif (Delta < -RATE) Delta = -RATE

        Int32(n)= Int32(n-1) +  RATE
        [s31:0]    [s31:0]     [s31:0]
        Temp(n) =   Int32(n) x 2^(-SCALER)
        [s31:0]    [s31:0]
        Output  = temp[s15:0]
        [s15:0]
    else
        Int32(n) = 2^SCALER x INIT
        [s31:0]           [s15:0]
        Output  = INIT
        [s15:0]   [s15:0]
    endif
```

Signal name	Description	I/O	Type
Input	Input	Input	16-bit, signed integer
ACC_RATE	Acceleration rate limit	Input	16-bit, signed integer. (range: 0 – 32767)
DEC_RATE	Decceleration rate limit	Input	16-bit, signed integer. (range: 0 – 32767).
SCALER	scaler for ramp rate range accomodation	Input	8-bit unsigned integer [1]
ENABLE	Block enable control bit	Input	Boolean, 0 = ramp function disabled (output = INIT) 1 = ramp function enabled
INIT	Reset value of block output	Input	16-bit, signed integer If(ENABLE = 0) Output = INIT
Output	Output	Output	16-bit, signed integer

Table 25. RAMP User Inputs and Outputs

Note 1: Although SCALER is an 8-bit unsigned interger, in order to avoid integer arithmetic overflow, the maximum value of SCALER should be restricted to 16.

Status	Clock cycles
Min. case	58
Max. case	68

Table 26. RAMP Execution Time

4.2.3.3 ATAN – Arc Tangent block

The ATAN block performs an arc tangent function. In order to optimize memory utilization, instead of using look-up table, the arc tangent function is implemented by Cordic algorithm inside the 300 series control IC. The block symbol and its mathematical approximation are shown in Figure 27.

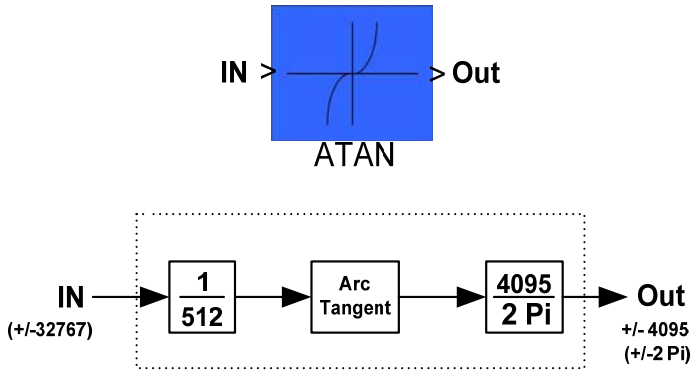


Figure 27. ATAN Block

Signal name	Description	I/O	Type
INPUT	Input	Input	16bit, signed integer
OUTPUT	Angle 4095 = 2π	Output	16bit, signed integer

Table 27. ATAN Inputs and Outputs

Status	Clock cycles
Nominal	24

Table 28. ATAN Execution Time

4.2.3.4 FUNCTION_BLOCK

The FUNCTION_BLOCK block maps a 16-bit signed input into a 16-bit signed integer output. The shape of the function is defined by the user providing six points (p0 – p5 of Figure 29) in two dimensional space. The six points are literal (constant) values provided at compile time. Figure 28 shows the FUNCTION_BLOCK symbol and Figure 29 shows a sample function in which the values between points are linearly interpolated. Table 29 describes all the inputs and the output.

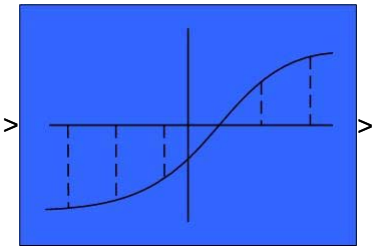


Figure 28. FUNCTION_BLOCK Block

The user provides six pairs of literal integer values, entered using a Simulink custom mask dialog, which is accessed by double-clicking the Function block: (X0,Y0) (X1,Y1) ….. (X5,Y5). It is the user's responsibility to ensure that no overflow can occur over the usable function range, or the results may be unpredictable. Outside the six points, the function is extrapolated. For input values less than point1, the slope p0-p1 is used. For inputs greater than point4, the slope p4-p5 is used.

Internally the function is represented by four coordinates (p1 – p4) and five slopes. (Points p0 and p5 are used only to determine the slope of the function outside the defined range.) These coordinates and slopes are computed by the MCE compiler and the thirteen values (four coordinate pairs and five slopes) are packed into ten input registers (each 16 bits wide). All values are signed.

The coordinates are represented internally by the high-order 12 bits of the input coordinate value appended with 4 binary zeroes, so the minimal X resolution is 16 while the minimal Y resolution is 16*slope. Slopes are represented by a 13-bit signed integer (except the p0 – p1 slope, which has only 12 bits) with a valid range of –4096 to 4095 (–2048 to 2047 for the p0 – p1slope). Slopes are normalized per 256 X distance, so the slope value is how much the Y changes when X increments by 256. In practice, this limits the slopes to –16 to 15.996 (–8 to 7.996 for the p0 – p1 slope). The accuracy of slopes is enough for practical cases.

In addition, the user provides a single (variable) input value, which the compiler passes directly to the MCE FUNCTION_BLOCK module. The module returns a signed 16-bit integer, which is the Y value corresponding to the input X value for the user defined function.

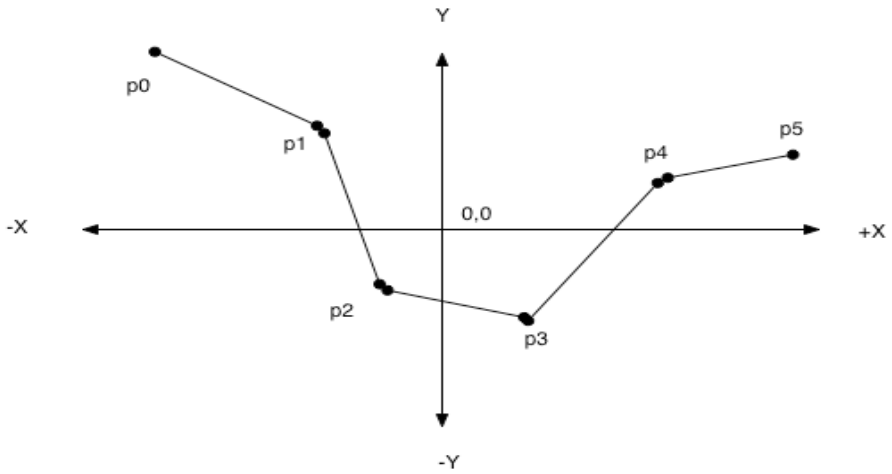


Figure 29. FUNCTION_BLOCK Example

Signal name	Description	I/O	Type
Input	x-coordinate input	Input	16 bit, signed integer
X0	1st point x-coordinate	Input	16 bit, signed integer
X1	2nd point x-coordinate	Input	16 bit, signed integer
X2	3rd point x-coordinate	Input	16 bit, signed integer
X3	4th point x-coordinate	Input	16 bit, signed integer
X4	5th point x-coordinate	Input	16 bit, signed integer
X5	6th point x-coordinate	Input	16 bit, signed integer
Y0	1st point y-coordinate	Input	16 bit, signed integer
Y1	2nd point y-coordinate	Input	16 bit, signed integer
Y2	3rd point y-coordinate	Input	16 bit, signed integer
Y3	4th point y-coordinate	Input	16 bit, signed integer
Y4	5th point y-coordinate	Input	16 bit, signed integer
Y5	6th point y-coordinate	Input	16 bit, signed integer
Output	y-coordinate interpolated output	Output	16 bit, signed integer

Table 29. FUNCTION_BLOCK Inputs and Outputs

Status	Clock cycles
Nominal	49

Table 30. FUNCTION_BLOCK Execution Time

4.2.3.5 COMPARATOR

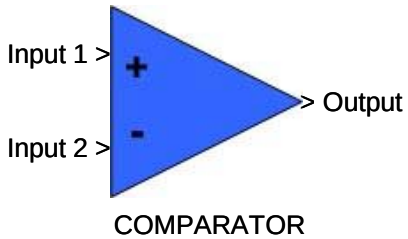


Figure 30. COMPARATOR Block

The following pseudo-code describes the function of the COMPARATOR block:

```
If (Input1 > = Input2) then
   [s15:0]      [s15:0]
        Output = 1
Else
        Output = 0
```

Signal name	Description	I/O	Type
Input1	Input 1	Input	16 bit, signed integer
Input2	Input 2	Input	16 bit, signed integer
Output	Output	Output	Boolean, 0 if Input1 < Input2 1 if Input1 >= Input2

Table 31. COMPARATOR Inputs and Outputs

Status	Clock cycles
Max. case	12

Table 32. COMPARATOR Execution Time

4.2.3.6 SWITCH

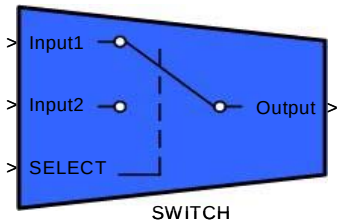


Figure 31. SWITCH Block

The operation of the SWITCH block can be described with the following psuedo-code:

```
If (SELECT = 0) then
        Output = Input1
Else
        Output = Input2
```

Note: if the SELECT input is non-Boolean, the LSB of the SELECT input will be used for logic decision.

Signal name	Description	I/O	Type
Input1	Input 1	Input	16 bit, signed integer
Input2	Input 2	Input	16 bit, signed integer
SELECT	Select between Input 1 or 2	Input	Boolean 0 = select input1 1 = select input2
Output	Output	Output	16 bit, signed integer

Table 33. SWITCH Inputs and Outputs

Status	Clock cycles
Max. case	–8

Table 34. SWITCH Execution Time

4.2.3.7 BIT_LATCH

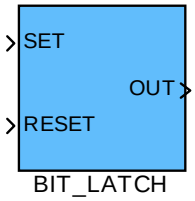
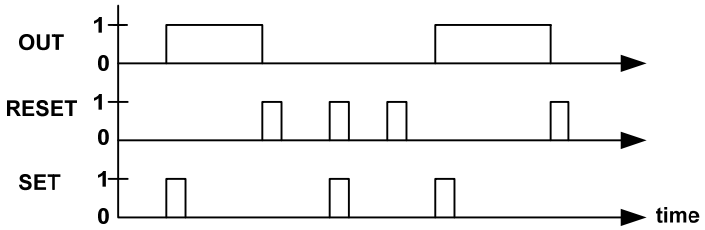


Figure 32. BIT_LATCH Block

The following pseudo-code describes the operation of the BIT_LATCH block:

If (positive edge (0 to 1) transition on RESET) then
OUT = 0
Elseif (positive edge (0 to 1) transition on SET) then
OUT = 1
Else no change on OUT

A Bit latch example is shown below to demonstrate the operation of the block:



The minimum duration of SET and RESET pulses should be larger than or equal to the execution rate of the BIT_LATCH block. In case inputs are connected by mistake to non-Boolean signals, the LSB of the non-Boolean signals will be used to determine logic operation.

Signal name	Description	I/O	Type
SET	Positive edge triggered input	Input	Boolean
RESET	Positive edge triggered input	Input	Boolean
OUT	Latched output	Output	Boolean

Table 35. BIT_LATCH User Inputs and Outputs

Status	Clock cycles
Max. case	26

Table 36. BIT_LATCH Execution Time

4.2.3.8 PEAK_DETECT

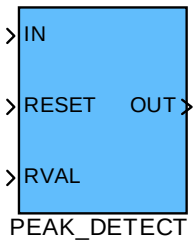


Figure 33. PEAK_DETECT Block

When RESET changes from a low to high value, the peak value of the input (IN) will appear at the output (OUT) for one block execution cycle. Thereafter, if RESET persists, the output will change to RVAL (OUT = RVAL). When RESET goes low, the input will be scanned for peak value again. It is recommended that the designer use the TRANSITION block to generate the RESET signal for continuous peak detection.

The following pseudo-code describes the operation of the PEAK_DETECT block:

```
If (RESET == 1)
        OUT     =   Store_Max
        [s15:0]     [s15:0]
        Store_Max = RVAL
        [s15:0]         [s15:0]
else
        if (IN        >=   Store_Max)
           [s15:0]
                Store_Max = IN
endif
```

Signal name	Description	I/O	Type
IN	Input	Input	16 bit, signed integer
RESET	Reset	Input	Boolean 0 = scan max. value of input 1 = output max and reset
RVAL	Reset Value	Input	16 bit, signed integer
OUT	Output	Output	16 bit, signed integer

Table 37. PEAK_DETECT User Inputs and Outputs

Status	Clock cycles
RESET = 0	13
RESET = 1	16

Table 38. PEAK_DETECT Execution Time

4.2.3.9 TRANSITION – One Shot Pulse Generator

The TRANSITION block provides transition detection of a Boolean input signal (IN). Transition detection format (positive and/or negative edge) can be configured by input POL (see Table 39).

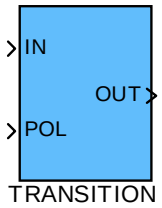


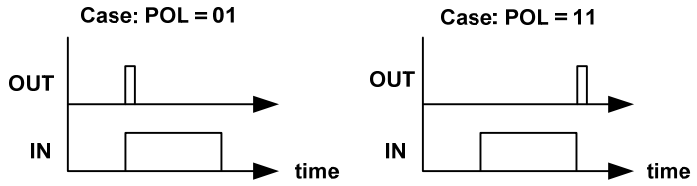
Figure 34. TRANSITION Block

The following pseudo-code describes the function of the TRANSITION block:

```
If  (POL > 0)  // case positive transition detection
{
        If ( IN = 1 and  IN_old = 0) OUT = 1
        Else OUT = 0
}
Elseif  (POL = 0) // case positive or negative transition detection
{
        If ( IN not equal to IN_OLD)  OUT = 1
        Else OUT = 0
}
Elseif  (POL < 0)  // case negative transition detection
{
        If ( IN= 0 and  IN_old = 1) OUT = 1
        Else OUT = 0
}

IN_old = IN
```

Note: IN_old is the previous sample of IN (input).



Examples of transition detection

Signal name	Description	I/O	Type
IN	Input signal	Input	Boolean
POL	Configure detection format 00 detects ↑↓ (+ & -) 01 detects ↑ (+) 10 detects ↓ (-) 11 detects ↓ (-)	Input	2-bit, signed integer
OUT	One pulse output for an edge being detected on the input.	Output	Boolean

Table 39. TRANSITION User Inputs and Outputs

Status	Clock cycles
Min. case	16
Max. case	27

Table 40. TRANSITION Execution Time

4.2.3.10 INTEGRAL2 – Integral with Limit

This blocks performs integration with configurable limits.

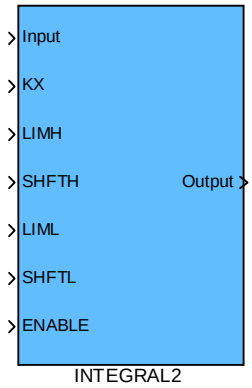


Figure 35. INTEGRAL2 Block

The following pseudo-code describes the function of the INTEGRAL2 block:

```
If ENABLE == 0
        Int(n) = 0
        [s31:0]
else
        Temp1(n)  =  KX       x     Input
        [s31:0]      [s15:0]        [s15:0]
        Int(n)    =  Temp1(n)  +   Int(n-1)
        [s31:0]      [s31:0]        [s31:0]
        Limit Int(n) to 32-bit signed value

        Temp1(n)  =  Shift left LIMH  by SHIFTH bits
        [s31:0]                [s31:0]
        Temp2(n)  =  Shift left LIML by SHIFTL bits
        [s31:0]                [s31:0]
```

If Int(n) > Temp1(n)
[s31:0] [s31:0]
Int(n) = Temp1(n)
If Int(n) < Temp2(n)
[s31:0] [s31:0]
Int(n) = Temp2(n)

Output(n) = Int(n) [s31:16]
[s15:0]

SHIFTH and SHIFTL are 5-bit unsigned integers with valid range of 0 – 31. Choosing SHIFTH and SHIFTL outside this range could overflow internal registers. Please ensure that LIMH * 2^{SHIFTH} > LIML * 2^{SHIFTL} for proper block utilization.

Signal name	Description	I/O	Type
Input	Input	Input	16 bit, signed integer
KX	Input Gain	Input	16 bit, signed integer
LIMH	Upper limit	Input	16 bit, signed integer
SHIFTH	Scaler for LIMH	Input	5 bit, unsigned integer
LIML	Lower limit	Input	16 bit, signed integer
SHIFTL	Scaler for LIML	Input	5 bit, unsigned integer
ENABLE	Enable/reset	Input	Boolean 0 = Reset recursive data 1 = Normal operation
Output	Output	Output	16 bit, signed integer

Table 41. INTEGRAL2 User Inputs and Outputs

Status	Clock cycles
Min. case	44
Max. case	74

Table 42. INTEGRAL2 Execution Time

4.2.3.11 PFC_FFD

The PFC Feed Forward (PFC_FFD) module provides a unique feed-forward function for digital PFC control. It can substantially improve the PFC current control performance, especially when operating with a low A/D sampling rate. Figure 37 shows a block diagram of the internal calculation of the PFC_FFD block. The MCEWizard provides proper parameters for the inputs to this block based on the user's application parameters.

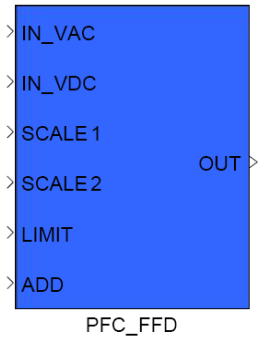


Figure 36. PFC_FFD Block

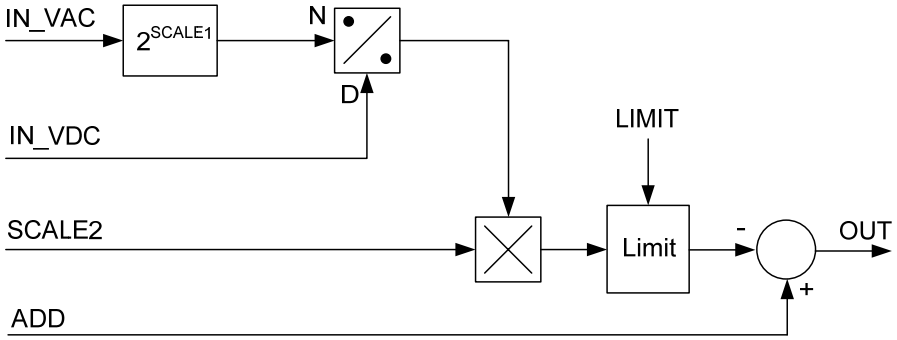


Figure 37. Block Diagram of PFC_FFD

Signal name	Description	I/O	Type
IN_VAC	VAC Input	Input	16 bit, signed integer
IN_VDC	VDC Input	Input	16 bit, signed integer
SCALE1	First scaler	Input	16 bit, signed integer
SCALE2	Secondary scaler	Input	16 bit, signed integer
LIMIT	Positive limit	Input	16 bit, signed integer
ADD	DC constant	Input	16 bit, signed integer
OUT	Output	Output	16 bit, signed integer

Table 43. PFC_FFD Inputs and Outputs

Status	Clock cycles
Normal Operation	95

Table 44. PFC_FFD Execution Time

4.2.4 Math

4.2.4.1 DIFF – Subtraction

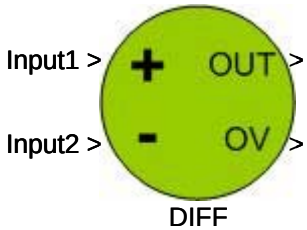


Figure 38. DIFF Block

The following pseudo-code describes the function of the DIFF block:

```
Temp  =   Input1 – Input2
[s15:0]    [s15:0]    [s15:0]
If Temp overflow then
          Temp = 32,767
          OV = 1
Elseif Temp underflow then
          Temp = -32,768
          OV = 1
Else
          OV =  0
 OUT = Temp
 [s15:0]
```

Signal name	Description	I/O	Type
Input1	DIFF block 1st input	Input	16 bit, signed integer
Input2	DIFF block 2nd input	Input	16 bit, signed integer
OUT	DIFF block output	Output	16 bit, signed integer
OV	overflow/underflow status bit	Output	Boolean, 1 16-bit signed integer overflow 0 no overflow

Table 45. DIFF Inputs and Outputs

Status	Clock cycles
Max. case	18

Table 46. DIFF Execution Time

4.2.4.2 SUM – Addition

The function of the SUM block can be described using the following pseudo-code:

```
Temp    =   IN1    +    IN2
[s15:0]     [s15:0]     [s15:0]
If Temp positive overflow (greater than 32,767) then
        Temp = 32,767
        OV = 1
Elseif OUT negative underflow (less than -32,768) then
        Temp = -32,768
        OV = 1
Else
        OV = 0
OUT  =  Temp
[s15:0]
```

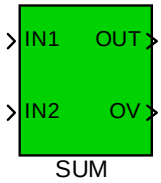


Figure 39. SUM Block

Signal name	Description	I/O	Type
IN1	SUM block 1st input	Input	16 bit, signed integer
IN2	SUM block 2nd input	Input	16 bit, signed integer
OUT	SUM block output	Output	16 bit, signed integer
OV	overflow/underflow status flag	Output	Boolean 1 16-bit signed integer overflow 0 no overflow

Table 47. SUM Inputs and Outputs

Status	Clock cycles
Max. case	22

Table 48. SUM Execution Time

4.2.4.3 ACCUMULATOR

This accumulator block is an integrator without overflow protect. This block can be used to build counters.

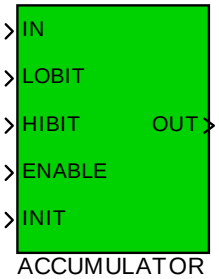


Figure 40. ACCUMULATOR Block

The following pseudo-code describes the operation of the ACCUMULATOR block:

```
If (ENABLE == 0)
        Sign extend INIT to 32 bits
        Output32 = INIT        *      2 LOBIT
         [s31:0]       [s15:0]

Output32 = Output32   +     IN
[31:0]        [31:0]           [s15:0]

If (HIBIT – LOBIT) > 15
        OUT = Output32 [LOBIT +15 : LOBIT]
        [15:0]
Else
        OUT = Output32 [HIBIT : LOBIT]
         [15:0]
```

If HIBIT or LOBIT are outside of the allowed range (0 <= LO/HIBIT <= 31) or LOBIT > HIBIT, then the OUT signal will be invalid. When HIBIT – LOBIT is less than 15, unfilled MSBs of the output will be zero.
Note: When enable = 0, the output of accumulator block is equal to (INIT + IN/2 ^{LOBIT}).

Signal Name	Description	I/O	Type
IN	Accumulator input	Input	16 bit, signed integer
LOBIT	Output extraction low bit.	Input	5 bit, unsigned integer
HIBIT	Output extraction high bit.	Input	5 bit, unsigned integer
ENABLE	Enable control bit	Input	Boolean, 0 freeze output Out = INIT + IN 1 normal accumulating
INIT	Reset value	Input	16 bit, unsigned integer
OUT	Output	Output	16 bit, unsigned integer

Table 49. ACCUMULATOR User Inputs and Outputs

Status	Clock cycles
Normal Operation	50

Table 50. ACCUMULATOR Execution Time

4.2.4.4 COUNTER

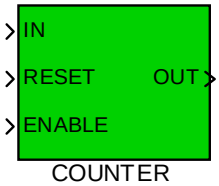


Figure 41. COUNTER Block

The pseudo-code shown below describes the operation of the COUNTER block.

```
If (RESET == 1 )
        OUT(n) = 0
         [s15:0]
Else
        If (ENABLE == 1 )
                OUT(n) = OUT(n-1) + IN(n)
                [s15:0]    [s15:0]       [s15:0]
                 Overflow protect OUT(n) to 16-bit signed interger
        Endif
 Endif
```

For example, if ENABLE = 1 and IN = 1, OUT will count up by 1 for each block execution until it reaches 32,767. The output (OUT) will remain at 32,767 until a negative integer is supplied at IN, or RESET = 1.

Signal name	Description	I/O	Type
IN	Counter Input	Input	16 bit, signed integer
RESET	Reset counter control bit	Input	Boolean 0 = normal operation, 1 = initialize output to 0
ENABLE	Enable Count control bit	Input	Boolean 0 = freeze output value 1 = unfreeze output
OUT	Output	Output	16-bit signed integer

Table 51. COUNTER User Inputs and Outputs

Status	Clock cycles
Nominal	17

Table 52. COUNTER Execution Time

4.2.4.5 SHIFT – Multiply by a Power of Two

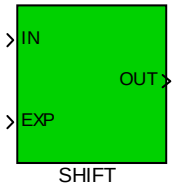


Figure 42. SHIFT Block

The SHIFT block multiplies an input by a power of two, as shown below.

$$\underset{[s15:0]}{OUT} = \underset{[s15:0]}{IN} \times 2^{EXP}$$

This operation can also be viewed as a binary shift left or right, where EXP specifies the number of bits to shift the IN value. A positive value for EXP shifts left and a negative value shifts right. The SHIFT block retains the sign bit of the input when shifting right (negative EXP).

Note:Please ensure that $-32768 <= IN * 2^{EXP} <= 32767$ *to avoid output overflow*

Signal name	Description	I/O	Type
IN	Input	Input	16-bit signed integer
EXP	Exponent	Input	-15 < EXP < 15
OUT	Output	Output	16-bit signed integer

Table 53. SHIFT Inputs and Outputs

Status	Clock cycles
Nominal	5

Table 54. SHIFT Execution Time

4.2.4.6 MUL_DIV – Signed / Unsigned Multiplier with Extraction

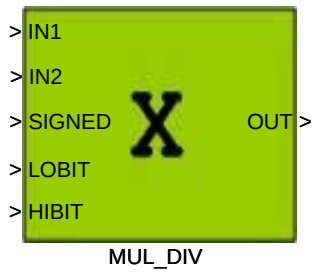


Figure 43. MUL_DIV Block

Pseudo code for MUL_DIV is given by:

```
If (SIGNED == 1)
        Temp =     IN1     x     IN2
        [s31:0]     [s15:0]      [s15:0]
        OUT  =  Temp[HIBIT : LOBIT]
        [s15:0]
        Overflow protect OUT to 16-bit signed interger
else
        Temp =     IN1     x     IN2
        [31:0]       [15:0]      [15:0]
        OUT  =  Temp[HIBIT : LOBIT]
        [15:0]
        Overflow protect OUT to 16-bit unsigned interger
End
```

Note:
If(HIBIT – LOBIT) < 15, say if LOBIT = 2, HIBIT = 13, only 12-bit will be extracted from Temp (OUT[11 :0] = Temp[13: 2]). The upper significant bits of OUT (OUT[15:12] will be filled with zeros.

If (HIBIT – LOBIT) > 15, MCE compiler will issue an error.

In the case that the 32-bit result overflows HIBIT, the output will be the largest integer for an N-bit signed or unsigned integer, depending on the mode, where N = (HIBIT – LOBIT + 1). For signed underflow, the output will be the most negative N-bit signed integer.

SIGNED, LOBIT and HIBIT are compile time parameters and should be connected to Simulink Constant blocks rather than variables.

Signal name	Description	I/O	Type
IN1	Input1	Input	16 bit, signed integer
IN2	Input2	Input	16 bit, signed integer
SIGNED	Signed / Unsigned	Input	Boolean constant, 0 = unsigned, 1 = signed
LOBIT	Starting bit	Input	5 bit, unsigned (0 – 31)
HIBIT	Ending bit	Input	5 bit, unsigned (0 – 31)
OUT	Output	Output	16bit, signed integer

Table 55. MUL_DIV Inputs and Outputs

Status	Clock cycles
Signed	10
Unsigned	17

Table 56. MUL_DIV Execution Time

4.2.4.7 DIVIDE

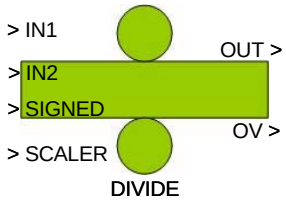


Figure 44. DIVIDE Block

The following pseudo-code describes the operation of the DIVIDE block:

```
If (IN2 == 0)
        Output = 0
        OV = 1
Else
        Temp =    IN1      x      2^SCALER
        [s31:0]   [s15:0]         [4:0]
        Output =  Temp    /    IN2
        [s31:0]   [s31:0]    [s15:0]
        If SIGNED = 0
            If (Output > 65535)
                Output = 65535
                OV = 1
        Else
            If (Output > 32767)
                Output = 32767
                OV = 1
            Elseif (Output < -32768)
                Output = -32768
                OV = 1
            Else OV = 0
        Endif
Endif
OUT = Output[15:0]
```

In the case of signed division, if $(IN1*2^{SCALER})/IN2$ overflows the 32-bit signed internal register (Output), the output may have incorrect sign and the overflow (OV) bit will not be set.

Signal name	Description	I/O	Type
IN1	Dividend	Input	16 bit, signed integer
IN2	Divisor	Input	16 bit, signed integer
SCALER	Dividend scale factor	Input	5 bit, unsigned integer (0 – 16)
SIGNED	Signed / Unsigned selector	Input	Boolean 0 = unsigned 1 = signed
OUT	Output	Output	16 bit, signed integer
OV	Overflow flag	Output	Boolean 0 = no overflow 1 = overflow

Table 57. DIVIDE Inputs and Outputs

Status	Clock cycles
Signed	41
Unsigned	43

Table 58. DIVIDE Execution Time

4.2.4.8 NOT – Bitwise Inversion

The NOT block performs a bitwise (logical) inversion of the unsigned 16-bit input value. That is, all 0 bits are set to 1 and all 1 bits are set to 0. For example, for Input = 30764, Output = 2003

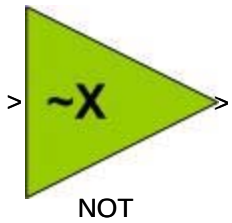


Figure 45. NOT Block

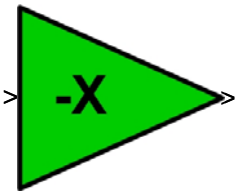
Signal name	Description	I/O	Type
Input	Input	Input	16 bit, unsigned integer
Output	Output	Output	16 bit, unsigned integer

Table 59. NOT Inputs and Outputs

Status	Clock cycles
Nominal	5

Table 60. NOT Execution Time

4.2.4.9 NEGATE – Two's Complement



NEGATE

Figure 46. NEGATE Block

The NEGATE block performs a two's complement of the signed 16-bit input value.

Output = - Input
[s15:0] [s15:0]

Note: 16-bit signed input range excludes -32768 (-32767 <= Input <= 32767). If input = –32768, output = –32768.

Signal name	Description	I/O	Type
Input	Input	Input	16 bit signed integer
Output	Output	Output	16 bit,signed integer

Table 61. NEGATE Inputs and Outputs

Status	Clock cycles
Nominal	5

Table 62. NEGATE Execution Time

4.2.4.10 AND – Bitwise Logical AND

The AND block performs a bitwise logical AND operation on two 16-bit unsigned input values. For example, INPUT1 = 5319 and INPUT2 = 30764 yeilds OUTPUT = 4100.

AND

Figure 47. AND Block

Signal name	Description	I/O	Type
INPUT1	Input1	Input	16 bit, unsigned integer
INPUT2	Input2	Input	16 bit, unsigned integer
OUTPUT	Output	Output	16 bit, unsigned integer

Table 63. AND Inputs and Outputs

Status	Clock cycles
Nominal	6

Table 64. AND Execution Time

4.2.4.11 OR – Bitwise Logical OR

The OR block performs a bitwise logical OR operation on two 16-bit unsigned input values. For example, INPUT1 = 5319 and INPUT2 = 30764 yeilds OUTPUT = 31983.

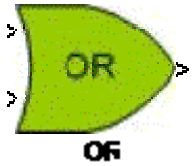


Figure 48. OR Block

Signal name	Description	I/O	Type
INPUT1	Input 1	Input	16 bit, unsigned integer
INPUT2	Input 2	Input	16 bit, unsigned integer
OUTPUT	Output	Output	16 bit, unsigned integer

Table 65. OR Inputs and Outputs

Status	Clock cycles
Nominal	6

Table 66. OR Execution Time

4.2.4.12 XOR – Bitwise Logical Exclusive OR

The XOR block performs a bitwise logical exclusive OR operation on two 16-bit unsigned input values. For example, INPUT1 = 5319 and INPUT2 = 30764 yeilds OUTPUT = 27883.

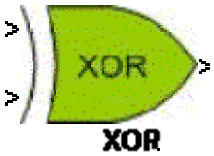


Figure 49. XOR Block

Signal name	Description	I/O	Type
INPUT1	Input 1	Input	16 bit, unsigned integer
INPUT2	Input 2	Input	16 bit, unsigned integer
OUTPUT	Output	Output	16 bit, unsigned integer

Table 67. XOR Inputs and Outputs

Status	Clock cycles
Nominal	6

Table 68. XOR Execution Time

4.3 Motion Peripherals

This section describes the motion peripheral blocks of the MCE library. Each motion peripheral block provides an interface to fixed elements of the IRMCx300 hardware. Therefore, unlike other blocks in the library, the motion peripherals may be used only once in a design, or once for each motor, where noted.

Motion peripherals differ from control blocks in their manner of execution. The connections to motion peripherals within the design do not determine the order of module execution. Motion peripheral inputs and outputs simply provide an interface to the registers that configure, control and monitor the associated hardware elements. Since motion peripherals are tied directly to IRMCx300 hardware elements, they run in parallel with the MCE and execute whether or not they are included in the MCE design. Timing estimates for the motion peripherals are included for informational purposes only. It is not necessary to include motion peripherals when determining the total execution time for the design.

4.3.1 SENSORLESS_FOC

The SENSORLESS_FOC module consists of current regulators, rotor angle estimator and startup control logic. These components forms the Sensorless Field-Oriented current regulated drive of the 300 series control IC. These components will be described in the subsequent sections.

For maximum configuration of the 300 series control IC, 2 motor drive controls (Motor1 and Motor2) and a PFC control can be realized simultaneously. The SENSORLESS_FOC module can be used once in the Motor1 portion of the MCE design and once in the Motor2 portion. The inputs and outputs of this block can be customized using the CustomMotPer utility described in Section 6.6. Figure 50 shows the block's inputs and outputs in the default configuration. The entire list of available inputs and outputs is presented in alphabetical order in Table 69.

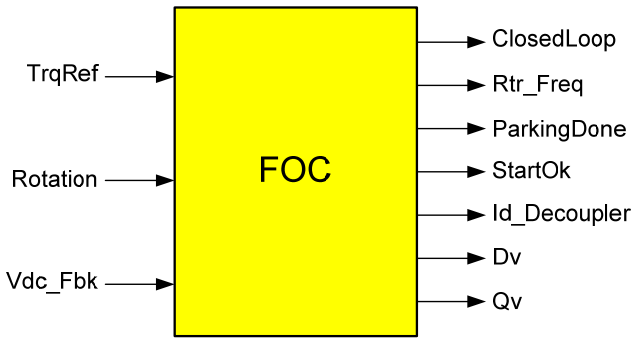


Figure 50. SENSORLESS_FOC Block

The SENSORLESS_FOC library block provides an interface to a subset of the IRMCx300 motion peripheral registers. All registers that control and monitor the operation of the sensorless field orientation control are accessible through the SENSORLESS_FOC block (with customization). Note that many of these registers values are calculated using the IRMCx31x MCEWizard tool and are typically initialized only once at system startup from an 8051 or host application. It is normally not necessary to update these registers from within the MCE design and they are, therefore, rarely customized as inputs to the SENSORLESS_FOC block. These inputs are identified by the term "Config Input" in the "I/O" column of Table 69.

Each signal listed in the table corresponds directly to one of the motion peripheral registers described in Section 4.4 or, where noted, to a bit field within a motion peripheral register. The signal name is the same as the register or bit field name. The rightmost column of Table 69 provides a reference to the document section that describes the associated register.

Execution time for the SENSORLESS_FOC block is 318 system clock cycles for each of the two motors.

Signal name	Description	I/O	Reference for Detailed Description and Scaling
AngDel	Gain adjustment for current angle	Config Input	4.4.7
AngLim	Maximum limit on current angle phase	Config Input	4.4.7
AtanTau	Angle compensation for phase shift	Config Input	4.4.7
CatchEnb	Enable catch spin (MtrCtrlBits_S, bit 0)	Config Input	4.4.9
ClosedLoop	Closed loop mode (StatusFlags, bit 3)	Output	4.4.21
Di	Flux current (d-axis) feedback	Output	4.4.23
DiagSelect	Select diagnostic (MtrCtrlBits, bits 0 – 3)	Config Input	4.4.9
Dv	D-axis command modulation index	Output	4.4.23
ExtFwdAngle	Angle sum into forward rotating angle	Input	4.4.18
ExtRevAngle	Angle sum into reverse rotating angle	Input	4.4.18
Ext_Flx_Alpha	External Alpha flux input	Input	4.4.18
Ext_Flx_Beta	External Beta flux input	Input	4.4.18
FlxThrH	Upper flux threshold level for drive startup	Config Input	4.4.10
FlxThrL	Lower flux threshold level for drive startup	Config Input	4.4.10
Flx_Alpha	Estimated motor flux of Alpha axis	Output	4.4.23
FlxAInit	Initial Alpha flux level	Config Input	4.4.7
FlxBInit	Initial Beta flux level	Config Input	4.4.7
Flx_M	Fundamental amplitude of Flx_Alpha	Output	4.4.23
FlxTau	Adjustment for flux estimator bandwidth	Config Input	4.4.7
FocEnable	FOC regulators enabled (StatusFlags, bit 1)	Output	4.4.21
FreqBW	Filter bandwidth for motor frequency	Config Input	4.4.7
I_Alpha	Alpha phase current	Output	4.4.23
I_Beta	Beta phase current	Output	4.4.23
IdRefExt	Command d-axis motor current, normal operation	Input	4.4.16
IdRef_C	d-axis command current	Output	4.4.23
Id_Decoupler	d-axis Current Decoupler output	Output	4.4.25
IfbkScl	Current gain	Config Input	4.4.3
IqRef_C	q-axis command current	Output	4.4.23
IregCompEnb	dc bus compensation (MtrCtrlBits_S, bit 4)	Config Input	4.4.9
IScl	Current gain scaler for flux estimator	Config Input	4.4.7
KpIreg	Proportional gain of q-axis current regulator	Config Input	4.4.5
KpIregD	Proportional gain of d-axis current regulator	Config Input	4.4.5
KxIreg	Integral gain of d- and q-axis current regulator	Config Input	4.4.5
KTorque	Gain used in open-loop startup mode	Config Input	4.4.8
L0	Apparent inductance of the motor	Config Input	4.4.7
LSlncy	Apparent saliency inductance of the motor	Config Input	4.4.7
Min_Spd	Minimum permissible drive operating speed	Config Input	4.4.6
NumRetries	Number of startup retries	Config Input	4.4.10
ParkAng	Second angle for parking stage	Config Input	4.4.9
ParkAng1	First angle for parking stage	Config Input	4.4.9
ParkI	dc current injection for parking stage	Config Input	4.4.9
ParkingDone	Parking complete (StatusFlags, bit 4)	Output	4.4.21
ParkingOne	Parking first stage complete (StatusFlags, bit 5)	Output	4.4.21
ParkIRet	dc current injection during retry parking	Config Input	4.4.10
ParkTm	Total parking duration	Config Input	4.4.9
ParkTmRet	Total retry parking duration	Config Input	4.4.10
PhsLosDisable	Disable phase loss detection (MtrCtrlBits, bit 4)	Config Input	4.4.9

Signal name	Description	I/O	Reference for Detailed Description and Scaling
PllFreqLim	Frequency limit of the PLL integral gain output	Config Input	4.4.14
PllIntLim	PLL frequency limit during catch spin	Config Input	4.4.14
PllKi	PLL tracking integral gain	Config Input	4.4.7
PllKp	PLL tracking proportional gain	Config Input	4.4.7
PwmEnable	PWM gatings enabled (StatusFlags, bit 2)	Output	4.4.21
Qi	q-axis current feedback	Output	4.4.23
Qv	q-axis command modulation	Output	4.4.23
RetryOccurred	Startup retry has occurred	Input	4.4.13
RetryTm	Startup flux sampling instant	Config Input	4.4.10
Rotation	Direction of rotation	Input	4.4.6
RotatorAngle	Estimated rotor angle	Output	4.4.23
Rs	Motor per phase resistance	Config Input	4.4.7
Rtr_Freq	Estimated unfiltered rotor electrical frequency	Output	4.4.24
Search_Ang	Rotor angle search command	Input	4.4.14
StartFail	Startup failed (StatusFlags, bit 6)	Output	4.4.21
Start_Lim	Desired startup current	Config Input	4.4.6
StartOk	Startup succeeded (StatusFlags, bit 7)	Output	4.4.21
TrqRef	Command current input	Input	4.4.6
TwoPhsEnable	Two-phase modulation enabled (StatusFlags, bit 0)	Output	4.4.21
TwoRetries	Two startup retries have occurred	Input	4.4.13
UdFeedFwd	d-axis modulation feedforward	Input	4.4.18
UqFeedFwd	q-axis modulation feedforward	Input	4.4.18
Use2xFrqScl	Use 2x frequency scale (MtrCtrlBits, bit 5)	Config Input	4.4.9
Use4xFrqScl	Use 4x frequency scale (MtrCtrlBits_S, bit 1)	Config Input	4.4.9
Use2xMagScl	Use 8x or 16x (applies only when Use4xMagScl=0) flux attenuation for PLL (MtrCtrlBits_S, bit 2).	Config Input	4.4.9
Use4xMagScl	Use 4x flux attenuation for PLL (MtrCtrlBits_S, bit 3)	Config Input	4.4.9
UseExtFlux	Use external fluxes for PLL (MtrCtrlBits_S, bit 5)	Config Input	4.4.9
Vdc_Fbk	Filtered dc bus voltage feedback	Input	4.4.17
VdcRcp	Reciprocal of DC bus voltage feedback ** Note 1	Input	4.4.17
VdLim	d-axis current regulator output limit	Config Input	4.4.5
VFFreq	Volts/Hertz frequency for open-loop diagnostic	Input	4.4.6
VFGain	Volts per Hertz gain scaler for open-loop diagnostic	Input	4.4.8
VoltScl	Internal voltage scaling gain	Config Input	4.4.7
VqLim	q-axis current regulator output limit	Config Input	4.4.5
V_Alpha	Alpha motor phase voltage	Output	4.4.23
WeThr	Transition level for closed-loop operation	Config Input	4.4.9
ZeroSpdDisable	Zero speed fault disable (MtrCtrlBits, bit 6)	Config Input	4.4.9
Zero_Vec_Req	Zero output voltage command	Input	4.4.14

Note 1. The VdcRcp input is a special case. This input can be added to at most one SENSORLESS_FOC block in the design (either motor 1 or motor 2), since it corresponds to a register that is not duplicated for the two motors.

Table 69. SENSORLESS_FOC Available Inputs and Outputs

The outputs of the SENSORLESS_FOC block are valid immediately at the start of each PWM cycle and represent the values calculated during the previous cycle.

Figure 51 shows a block diagram of the sensorless field orientation control functions. There are three main components to the Sensorless FOC block. These components are Current Control, Rotor Position estimation and Startup Control.

4.3.1.1 Current Control

The current control system consists of two PI regulators, two Vector Rotators (forward and backward), a Current Decoupler, a Clark transformation (3 → 2 phase) and a dc bus compensation.

Two Proportional plus Integral (PI) type current regulators with output limits and Anti-windup control are provided for torque and flux current control of motors. These two PI regulators operate in conjunction with a forward and a backward vector rotator to form a synchronously rotating frame current control system. The rotor magnet position is chosen to be the frame of reference for the current control. This is done to achieve "Field-Orientation Control". The rotor position is supplied by an angle estimator.

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

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

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 permanent magnet

There are two torque components associated with the motor torque equation. The first component (Cylindrical torque) is due to interaction between rotor magnet flux and stator q-axis current. The second component (reluctance torque) is due to 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. A larger negative Id is required for higher motor loading in order to maintain maximum Torque per Ampere performance.

The current controller receives current commands (d-q) from a Current decoupler. This current decoupler is implemented for the purpose of providing optimum q-axis and d-axis (rotor magnet) command current to achieve maximum motor Torque per Ampere generation. The current command output (Id_Decoupler) of the decoupler will be a negative value in the case of motor with saliency (Ld not equal Lq). More detail explaination on the current trajectory profiling of the current decoupler can be found in the Application Developer's Guide, under the Interior Permanent Magnet Motor Control section.

The d-axis command current (IdRefExt) is an input to the Sensorless FOC (Figure 51). This flux current input should be fed from the d-axis output (Id_Decoupler) of the Current decoupler. If a more elaborate d-axis current control regime (for instance: Field-Weakening Control) is desired, the d-axis command current can be further processed prior to feeding the d-axis command current input (IdRefExt) of the Sensorless FOC block. The Field-Weakening Control regime is done in the MCE application layer of the reference design platform.

The d-q current regulator outputs can be compensated by dc bus voltage if a large variation of dc bus voltage is expected. (This option is disabled by default.) The inputs to the forward Vector Rotator (converts dc to ac

waveforms) are PWM modulation depths. These signals are fed to a Space Vector PWM (SVPWM) modulator for inverter firing control.

4.3.1.2 Rotor Position Estimation

The rotor position estimation consists of a flux estimator and an Angle-Frequency generator. The flux component generated by the motor rotor magnet is computed by a flux estimator. The fundamental principle of the flux estimator utilizes integration of motor BEMF voltages. Due to inherent dc offset problems, pure integration cannot be used. Therefore a non-ideal integrator (high-pass filter) is used instead. The motor BEMF voltage is computed by motor model which uses motor resistance and inductance parameters, current feedbacks (I_Alpha and I_Beta) and estimated motor terminal voltages (constructed by Av, Bv and VdcFbk) as the model inputs. The output of the flux estimator represents rotor magnet fluxes in Alpha-Beta (stationary orthogonal frame, u-phase aligned with Alpha) two-phase quantities. The outputs of the flux estimator are a pair of quadrature flux signals. In order to extract the rotor position and frequency out of this pair of quadrature signals, an Angle-Frequency generator (Figure 51) is employed. The Angle-Frequency generator computes position and frequency from the Alpha-Beta flux inputs. However, users can also have the flexibility of using external fluxes (Ext_Flx_Alpha and Ext_Flx_Beta) instead of internally generated fluxes. This allows implementation of a Field-oriented control scheme using enhanced position sensing or flux sensing methods to achieve very low speed operations.

Another level of flexibility is provided with the option to completely replace the estimated rotor angle by external signals (ExtFwdAngle, ExtRevAngle).

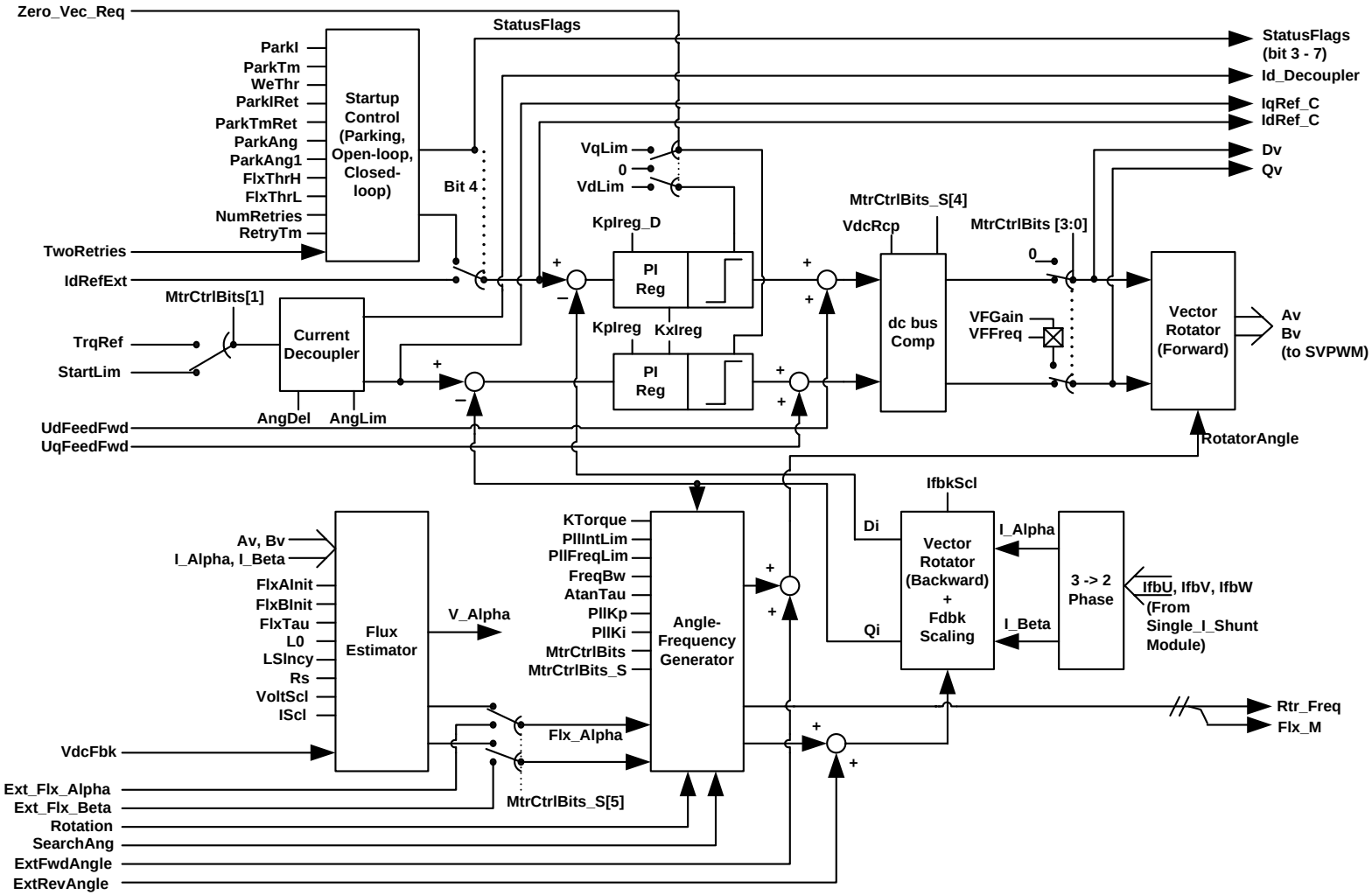


Figure 51. SENSORLESS_FOC Block Diagram

4.3.1.3 Startup Control

The motor control is performed without a shaft encoder (Sensorless). As mentioned in the previous section (4.3.1.2), the flux estimator utilizes integration of motor BEMF, which imposes a lower speed limit to motor torque control at low speeds (typically less than 5% rated rpm). Due to fact that motor BEMF signal is small at low speeds and eventually disappears at zero speed, it is impractical to track motor flux at startup. Therefore, a special startup sequence is implemented to provide robust startup. Startup control components inside the Sensorless FOC block are provided to assist drive startup with the ability to configure the controller dynamically to three unique operating states (Parking, Open-loop or Closed-loop). These three states are illustrated in Figure 52 and described below.

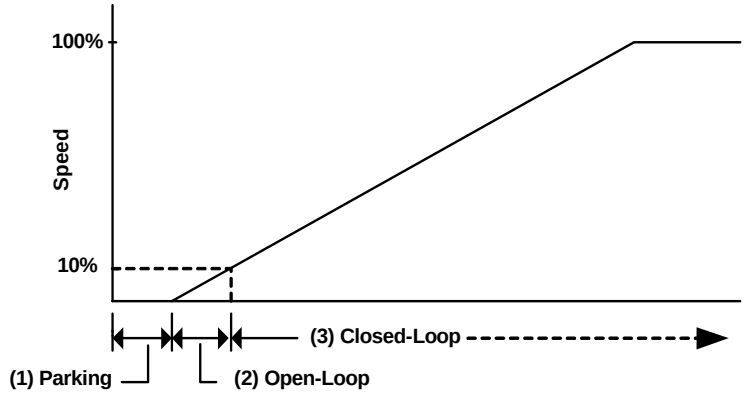


Figure 52. Drive Control Modes

State 1: Parking

The initial rotor angle is identified by forcing DC current into the motor winding and hence forcing the motor shaft to park at a certain prescribed angle. During initial inverter startup, a dc current is impressed in the stator winding. Since the initial rotor position is unknown, the relative position between rotor and the current vector is arbitrary. In most cases, the rotor will pull towards the current vector and form alignment as shown in Figure 53a and b. However, there are situations where the rotor flux is 180 degrees out of phase with the applied current vector (Figure 53c). This condition will cause a failure in flux current alignment.

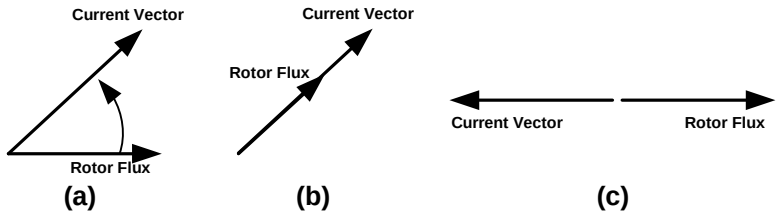


Figure 53. Flux and Current vector displacement

In order to avoid the condition of Figure 53c, in the 300 series controller, dc injection is applied to the motor in 2 stages as shown in Figure 54. In the first stage, a current vector (Figure 54) is applied with configurable amplitude (ParkI) and angle (ParkAng_1). In the second stage (happens at time equal to ParkTm/4), a second current vector of the same amplitude but different angle (at ParkAng) is applied. The 2 stages are used to avoid misalignment of magnetic polarity. The effectiveness of pulling rotor to the prescribed current vector position is improved as a consequence of 2-stage parking. For instance, in Figure 54, stage 1 dc current injection uses 120 degree (ParkAng_1) and stage 2 uses 90 degree (ParkAng), these are default values for the 300 series MCEWizard setup.

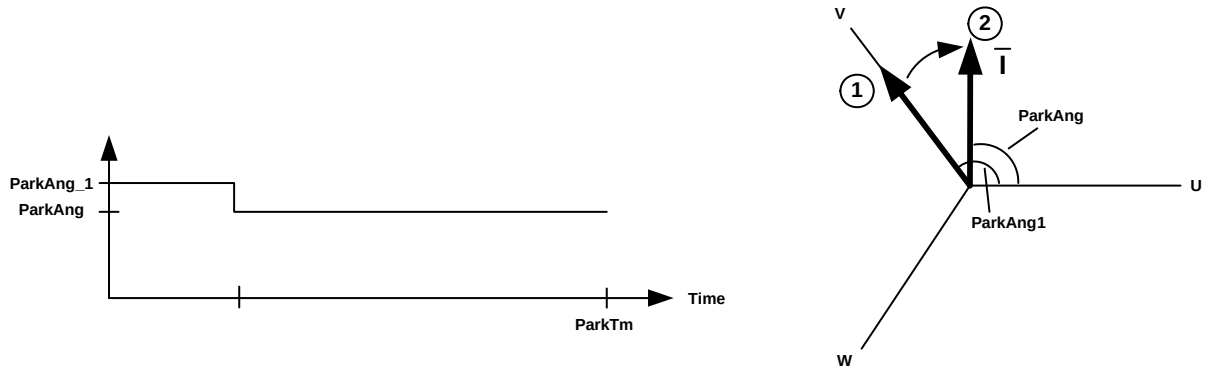


Figure 54. 2-Stage Parking

State 2: Open-loop Angle Estimation
At zero speed or low speed (<10%) conditions, it is difficult to accurately measure or estimate motor voltages due to the low amplitude of motor back EMF (BEMF). In most Sensorless (no shaft encoder) control drives, the tracking of rotor angle based on BEMF normally fails at low speeds (< 5%). Therefore, Sensorless control of a permanent magnet motor drive requires some means of starting the motor. In most cases, the motor is started in an open-loop fashion. As soon as the motor speed picks up (typically >10%), the drive switches to closed-loop (uses current and/or voltage feedback) control mode. However, during the switchover from open-loop to closed-loop mode, torque and current pulsation may occur due to mode transitioning. In 300 series controller, a unique switch-over algorithm (patented) has been implemented to suppress torque pulsation during mode transitioning.

Immediately after parking, the drive controller enters a quasi open-loop mode. The rotor angle is estimated in an open-loop fashion, which utilizes a simple (one model parameter KTorque) motor-load mechanical model to estimate the rotor angle. However, unlike the traditional open-loop control which does not uses feedback signals, in the quasi open-loop control mode, current regulation is preserved. This ensures limitation on the maximum current capability imposed by the power inverter. If mismatch between external load characteristics and the internal motor-load model is exceedingly large, start-up performance will suffer. Generally, minor tuning may be required to achieve optimal (max torque per ampere) startup performance, this tuning is described in the IRMCx300 application developer's guide.

State 3: Closed-loop Angle Estimation
Motor speed increases during start-up; the motor voltage also builds up due to the increase in speed. Useful information for rotor angle estimation can be then be extracted from the motor voltage (estimated by using PWM modulation depth and DC bus voltage) as motor speed increases. The drive will enter Closed-loop control mode as shown in Figure 52.

During a Sensorless motor drive start-up, the motor torque has to overcome drive stiction and friction in order to successfully increase speed. However, the motor shaft stiction and friction may vary (increase dramatically) due to applied load characteristics. For instance, the stiction of an outdoor pump under colder temperature is higher. In some cases, motor shaft may even be partially jammed. Under such circumstance, careful tuning (refer to IRMCx300 application developer's guide) of open-loop startup parameter (KTorque) and parking current may avoid a startup problem. However, startup failure may persist. This startup failure normally occurs and can be detected during mode transition (open-loop to closed-loop). In the Sensorless FOC block, a start fail detection signal (Statusflags bit 6) is provided for startup failure detection. This signal can be used by a master Motor control sequencer to carry appropriate actions (for instance: startup retry) upon drive startup failure.

4.3.2 SINGLE_I_SHUNT

The SINGLE_I_SHUNT module can be used once in the Motor1 portion of the MCE design and once in the Motor2 portion. The inputs and outputs of this block can be customized using the CustomMotPer utility described in Section 6.6. Figure 55 shows the block's outputs in the default configuration. (There are no inputs in the default configuration.) The entire list of available inputs and outputs is presented in alphabetical order in Table 70.

The SINGLE_I_SHUNT library block provides an interface to a subset of the IRMCx300 motion peripheral registers. All registers that control and monitor the operation of the single current shunt are accessible through the SINGLE_I_SHUNT block (with customization). Note that some of these registers values are calculated using the IRMCx300 MCEWizard tool and are typically initialized only once at system startup from an 8051 or host application. It is normally not necessary to update these registers from within the MCE design and they are, therefore, rarely customized as inputs to the SINGLE_I_SHUNT block. These inputs are identified by the term "Config Input" in the "I/O" column of Table 70. The current feedback outputs (IfbV and IfbW) of this module are internally connected to the Sensorless FOC module (Figure 51) to achieve current regulation and rotor angle estimation.

Each signal listed in Table 70 corresponds directly to one of the motion peripheral registers described in Section 4.4 or, where noted, to a bit field within a motion peripheral register. The signal name is the same as the register or bit field name. The rightmost column of Table 70 provides a reference to the document section that describes the associated register.

Execution time for the SINGLE_I_SHUNT block is approximately 7.33 microseconds for each of the two motors, independent of the system clock rate.

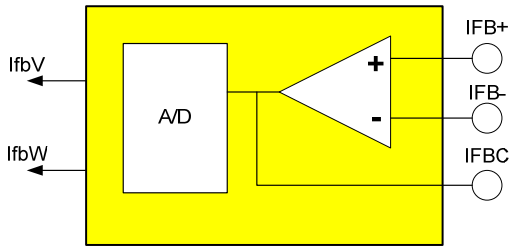


Figure 55. SINGLE_I_SHUNT Block

Signal name	Description	I/O	Reference for detailed description and scaling
IfbOffset	Current feedback dc offset compensation	Output	4.4.25
IfbOffsetCalc	Current feedback dc offset control line	Input	4.4.13
IfbV	Reconstructed motor phase V current	Output	4.4.23
IfbW	Reconstructed motor phase V current	Output	4.4.23
ScsSamples	Setup for adaptive current feedback sampling	Config Input	4.4.12
SHDelay	Hardware PWM gate propagation delay	Config Input	4.4.12
TCntMin2Phs	Minimum PWM pulse width for 2-phase modulation mode	Config Input	4.4.12
TCntMin3Phs	minimum PWM pulse width for 3-phase modulation mode	Config Input	4.4.12

Table 70. SINGLE_I_SHUNT Available Inputs and Outputs

The outputs of the SINGLE_I_SHUNT block are valid immediately at the start of each PWM cycle and represent the values calculated during the previous cycle.

A two-level inverter can produce eight possible basis voltage vectors; any desired (command) voltage vector can be formed by these eight vectors, up to the inverter maximum output voltage limit (determined by the dc bus voltage

level). In a PWM inverter drive system, the information of motor phase can be observed from the dc bus current when non-zero basis vectors are used. Each basis vector is assigned a specific time in a PWM cycle in order to generate the command voltage vector. However, if the time spent on a basis vector is not long enough, the motor current cannot be observed.

The dc link current consists of high frequency (PWM switching) current pulses. These current pulses coupled with circuit layout parasitic and reverse recovery diode current cause ringing in the dc link feedback current as illustrated in Figure 56. This figure displays a snap shot of the dc link current and the current sampling instances. In this case, the sampling instances are placed at the center of the corresponding active pulse. As can be seen in this figure, if the current pulse width reduces (occurs at low modulation index or sector crossing), the sampling instances will migrate into the current ringing area (beginning of the current step) and erroneous current feedback sampling will result. Therefore, minimum pulse constraints have to be inserted to allow reliable dc link current feedback sampling.

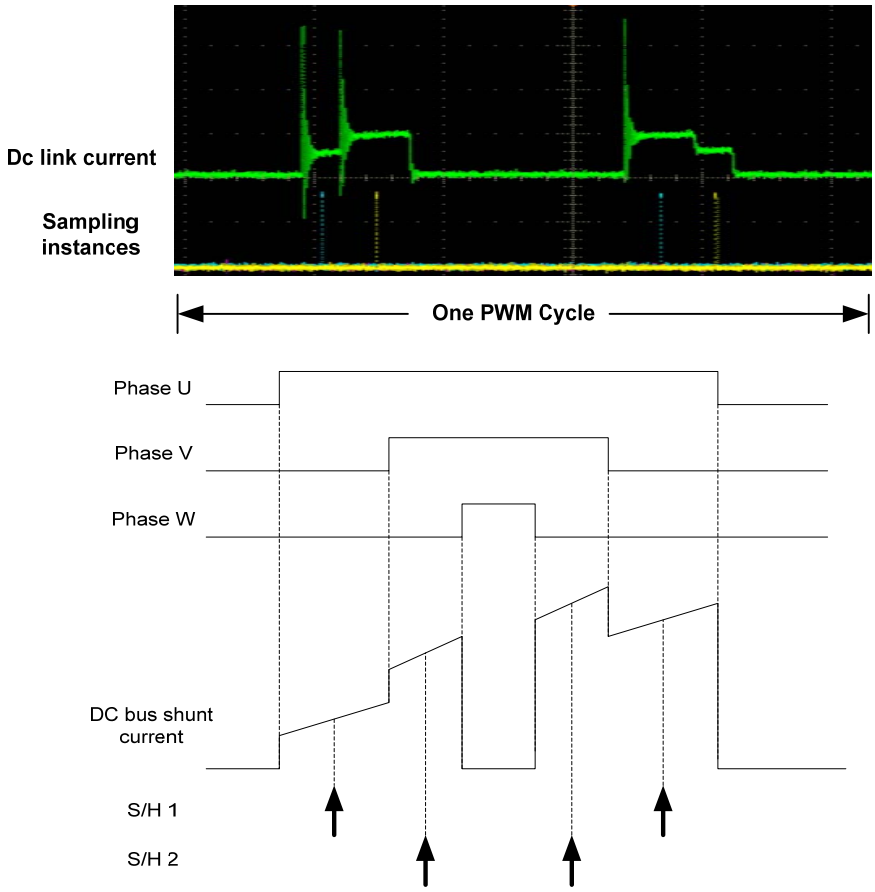


Figure 56. Single Shunt Current Sense Timing

Figure 57 shows the insertion of minimum time constraints (TCntMin3Phs, TCntMin2Phs) with the sampling instances (S1 to S4) corresponding to idealized PWM command gatings. Two active vectors (100, 110) are shown in Figure 57. As the voltage vector rotates from vector 100 to 110, the time duration of active vector 100 will decrease while vector 110 will increase as illustrated in Figure 57a and Figure 57b. The minimum pulse constraint has been reached for active vector 100 and 110 (Figure 57a and b). A Similar situation occurs in the case of 2-phase modulation.

Due to gate propagation delay between command gatings and actual power devices firing, a time shift parameter (SHDelay, section 4.4.12) is provided for fine tuning of the actual sampling instances (S1 to S4). This tuning procedure is described in the IRMCx300 Application Developer's Guide.

The current sensing channel contains the operational amplifier and two parallel sample/hold circuits. The operational amplifier can be used for adjusting analog input voltage developed across the shunt resister. The operational amplifier is powered by 1.8V (AVDD) and the output common mode voltage of the operational amplifier should be mapped to 0 – 1.2V. Typical application connection is a differential mode amplifier, which can be realized using the external resistors and capacitor as shown in Figure 60.

Two parallel sample/hold circuits are designed to capture two motor phase chopped current signals by synchronizing to the PWM switching pattern. It is designed to hold for a maximum 10 microseconds and is able to charge to half of the common mode voltage (0.6V) within 250 nanoseconds. The sample/hold switch is normally closed and opened at the center point of a new active voltage vector, as shown in Figure 56.

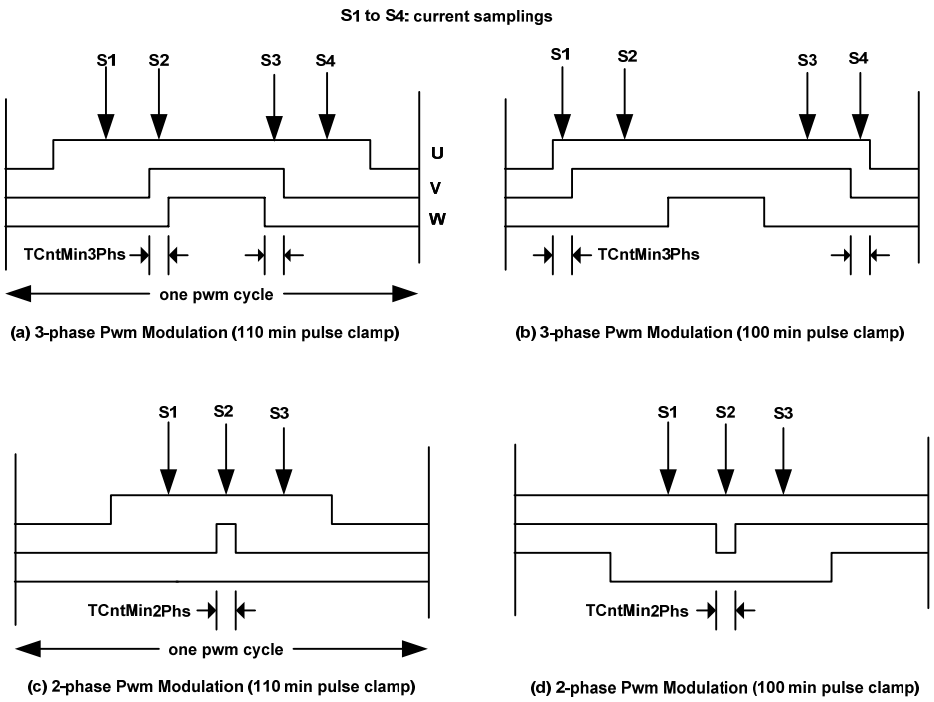


Figure 57. Single Current Shunt Registers (TCntMin2Phs, TCntMin3Phs)

The minimum pulse width constraints have the disadvantage of creating harmonics in the drive current waveform. These harmonics can lead to acoustic noise at low modulation (generally low speed) where the motor spends a high proportion of time under the minimum pulse constraint. The register ScsSamples allows one to enter an undersampling mode when the drive enters the regions of minimum pulse constraint. The register specifies the umdersampling rate (1/2, 1/4, 1/8, etc), so that the minimum pulse width (and current sampling) is applied only in the specified fraction of the PWM cycles which are in a region of minimum pulse constraint. By reducing the number of PWM cycles which are under minimum pulse constraint, harmonics and acoustic noise can be reduced. The disadvantage of the undersampling is slower dynamic response.

4.3.3 DC_BUS_VOLTAGE

The DC_BUS_VOLTAGE module is shown in Figure 58. Its outputs are listed in Table 71. The block has no inputs.

Execution time for the DC_BUS_VOLTAGE block is approximately 1.83 microseconds, independent of the system clock rate.

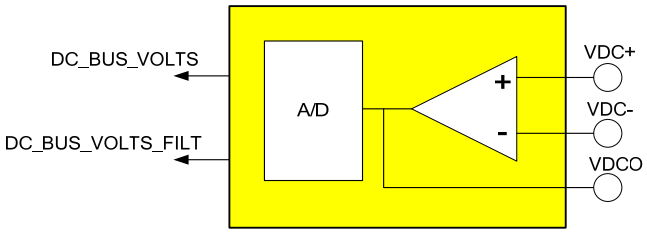


Figure 58. DC_BUS_VOLTAGE Block

Signal name	Description	I/O	Type
DC_BUS_VOLTS	DC bus voltage	Output	16 bit, signed integer
DC_BUS_VOLTS_FILT	Filtered DC bus voltage	Output	16 bit, signed integer

Table 71. DC_BUS_VOLTAGE Outputs

The outputs of the DC_BUS_VOLTAGE block are valid immediately at the start of each PWM cycle and represent the values calculated during the previous cycle. Output DC_BUS_VOLTS corresponds to the DcBusVolts register (Section 4.4.22) and output DC_BUS_VOLTS_FILT corresponds to register DcBusVoltsFilt (Section 4.4.20). Note; filtered dc bus signal (DC_BUS_VOLTS_FILT) is generated by first order low pass filtering (0.49 msec filter time constant) of raw dc bus signal (DC_BUS_VOLTS).

4.3.4 A_D – A/D Converter

Each of the IRMCx300 Series products has at least one general purpose analog input and one analog input dedicated to dc bus sensing. (In the IRMCx312, the dc bus channel contains an op-amp.) Access to the general-purpose converted outputs is provided through the A_D library blocks. Because each product has a different number of analog inputs, separate blocks are provided for interface to each, as shown in Figure 59. The outputs of the the A_D blocks are listed in Table 72. The A_D blocks have no inputs.

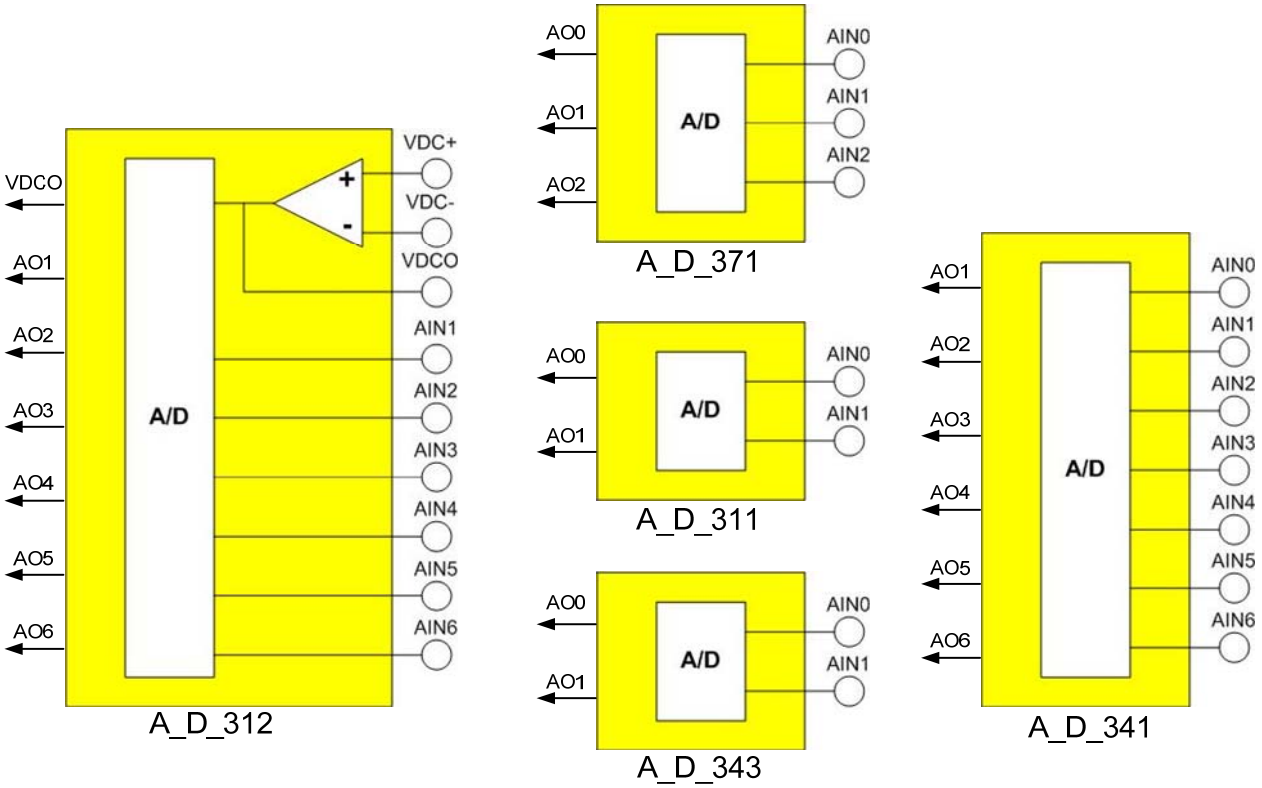


Figure 59. A/D Interface Blocks

Signal name	Description	I/O	Type
VDC/AO0	A/D converter raw output for operational amplifier VDC	Output	12 bit, signed integer
AO1	A/D converter raw output for input AIN1	Output	12 bit, signed integer
AO2 *(312, 341, 371 only)*	A/D converter raw output for input AIN2	Output	12 bit, signed integer
AO3 *(312, 341 only)*	A/D converter raw output for input AIN3	Output	12 bit, signed integer
AO4 *(312, 341 only)*	A/D converter raw output for input AIN4	Output	12 bit, signed integer
AO5 *(312, 341 only)*	A/D converter raw output for input AIN5	Output	12 bit, signed integer
AO6 *(312, 341 only)*	A/D converter raw output for input AIN6	Output	12 bit, signed integer

Table 72. A_D Outputs

Conversion time for each channel is a maximum of 2.0 microseconds. Unlike a traditional A/D converter in a microcontroller, the conversion process and associated timing of the sample/hold and multiplexer are automated by

internal hardware logic. This is due to the fact that there is a dedicated analog input channel for single shunt current feedback and it requires specific timing combined with sample/hold (See 4.3.2).

The input circuit and application connections are shown in Figure 60 for the maximum input case of the 300 series A/D input structure. The number of inputs is reduced for some of the 300 series ICs with reduced pin counts. For instance, AIN2 – AIN6 are not available on the IRMCx311. Only in the IRMCx312 are the VDC op-amp inputs accessible; in other versions, AIN0 is equivalent to VDCO and VDC+ and VDC– do not exist.

All analog circuitry is referenced to AVDD (1.8V) and AGND. Besides the current sensing channel, there are up to six unbuffered analog inputs with a 0 – 1.2V input range. The A/D data update rate is synchronous to the PWM carrier frequency. One of the unbuffered A/D channels (AIN1 – AIN6) is updated on each PWM cycle, so that each channel is updated once every six cycles, regardless of how many channels are available in the particular IRMCx300 product.

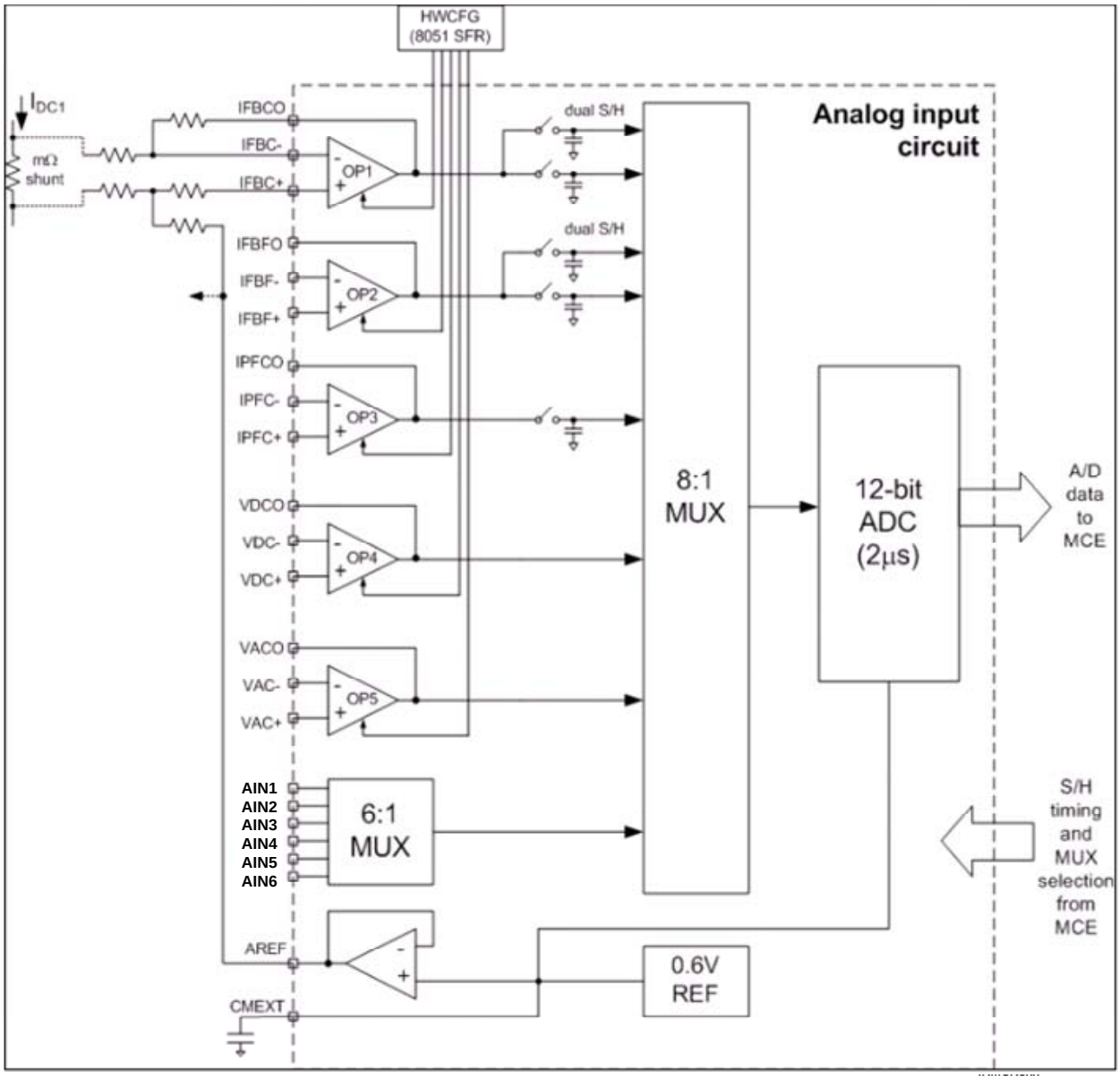


Figure 60. IRMCx312 A/D Converter Structure

4.3.5 Low Loss Space Vector PWM

The Low Loss Space Vector PWM (LOWLOSS_SVPWM) module accepts modulation index commands and generates the appropriate gating waveforms for each PWM cycle. The inputs (Alpha-Beta modulation depth) of the Space Vector modulator are normally connected internally to the outputs (Av, Bv of Figure 51) of the SENSORLESS_FOC module. However, users can utilize the LOWLOSS_SVPWM without SENSORLESS_FOC. When register UserVabEn (Figure 62) is set to 1, the LOWLOSS_SVPWM module accepts user generated modulation index commands (User_Alpha and User_Beta).

Execution time for the LOWLOSS_SVPWM block is 79 system clock cycles for each motor without over modulation or 106 system clock cycles for each motor with over modulation. (See Section 4.3.5.1 for a description of over modulation.)

The LOWLOSS_SVPWM module can be used once in the Motor1 portion of the MCE design and once in the Motor2 portion. The inputs and outputs of this block can be customized using the CustomMotPer utility described in Section 6.6. Figure 61 shows the block's inputs in the default configuration (the default configuration has no outputs). The entire list of available inputs and outputs is presented in alphabetical order in Table 73.

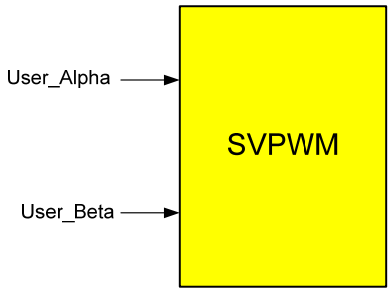


Figure 61. Low Loss SVPWM Block

The LOWLOSS_SVPWM library block provides an interface to a subset of the IRMCx300 motion peripheral registers. All registers that control and monitor the operation of the LOWLOSS_SVPWM module are accessible through the LOWLOSS_SVPWM block (with customization). Note that many of these register values are calculated using the IRMCx300 MCEWizard tool and are typically initialized only once at system startup from an 8051 or host application. It is normally not necessary to update these registers from within the MCE design and they are, therefore, rarely customized as inputs to the LOWLOSS_SVPWM block. These inputs are identified by the term "Config Input" in the "I/O" column of Table 73.

Each signal listed in the table corresponds directly to one of the motion peripheral registers described in Section 4.4 or, where noted, to a bit field within a motion peripheral register. The signal name is the same as the register or bit field name. The rightmost column of Table 73 provides a reference to the document section where the associated register is described. The utilization of these registers within the low loss SVPWM block is shown in Figure 62.

Signal name	Description	I/O	Reference for Detailed Description and Scaling
CfgPWMUH	U phase high side pwm configuration (port_ctrl0 or port_ctrl1, bits 0 – 1)	Input	4.4.1
CfgPWMUL	U phase low side pwm configuration (port_ctrl0 or port_ctrl1, bits 2 –3)	Input	4.4.1
CfgPWMVH	V phase high side pwm configuration (port_ctrl0 or port_ctrl1, bits 4 – 5)	Input	4.4.1
CfgPWMVL	V phase low side pwm configuration (port_ctrl0 or port_ctrl1, bits 6 – 7)	Input	4.4.1
CfgPWMWH	W phase high side pwm configuration (port_ctrl0 or port_ctrl1, bits 8 – 9)	Input	4.4.1
CfgPWMWL	W phase low side pwm configuration (port_ctrl0 or port_ctrl1, bits 10 – 11)	Input	4.4.1
CriticalOv	Activate zero vector PWM state	Input	4.4.16
FocEnable	FOC enable command (pwmctrl, bit 1)	Input	4.4.4
GateSenGateKill	Configure Gate kill sense (pwmcfg, bit 4)	Config Input	4.4.2
GateSenHigh	Configure Gate Sense high side (pwmcfg, bit 3)	Config Input	4.4.2
GateSenLow	Configure Gate Sense low side (pwmcfg, bit 2)	Config Input	4.4.2
GCChargePD	Number of charging pulses between motor phases	Config Input	4.4.2
GCChargePW	Gate Pre-charge duration	Config Input	4.4.2
ModScl	Space Vector PWM scaling	Config Input	4.4.2
MotorSpeed	Filtered motor speed	Input	4.4.6
Precharge	Gating pre-charge command (pwmctrl, bit 2)	Input	4.4.4
Pwm2HiThr	3-phase to 2-phase PWM high threshold	Config Input	4.4.2
Pwm2LowThr	3-phase to 2-phase PWM low threshold	Config Input	4.4.2
PwmDeadTm	Inverter blanking time	Config Input	4.4.2
PwmEnable	Pwm enable command (pwmctrl, bit 0)	Input	4.4.4
PwmGateEnb	Enable pwm gating (pwmctrl, bit 3)	Input	4.4.4
PwmGuardBand	Pwm guard band	Config Input	4.4.2
PwmPeriodConfig	PWM carrier period configuration	Config Input	4.4.2
PWMUH	U phase high side output (pwm_lines, bit 6 or 0)	Output	4.4.20
PWMUL	U phase low side output (pwm_lines, bit 7 or 1)	Output	4.4.20
PWMVH	V phase high side output (pwm_lines, bit 8 or 2)	Output	4.4.20
PWMVL	V phase low side output (pwm_lines, bit 9 or 3)	Output	4.4.20
PWMWH	W phase high side output (pwm_lines, bit 10 or 4)	Output	4.4.20
PWMWL	W phase low side output (pwm_lines, bit 11 or 5)	Output	4.4.20
TwoPhsEnb	Select 2-phase modulation (TwoPhsCtrl, bit 0)	Config Input	4.4.2
TwoPhsType	2-phase modulation type (TwoPhsCtrl, bit 1)	Config Input	4.4.2
User_Alpha	User Alpha modulation index	Input	4.4.15
User_Beta	User Beta modulation index	Input	4.4.15
User_U	User U-phase duty ratio control	Input	4.4.15
UserVabEn	SVPWM modulation input selector	Input	4.4.15
UserVuvwEn	PWM pattern selector	Input	4.4.15
User_V	User V-phase duty ratio control	Input	4.4.15
User_W	User W-phase duty ratio control	Input	4.4.15

Table 73. SVPWM Available Inputs and Outputs

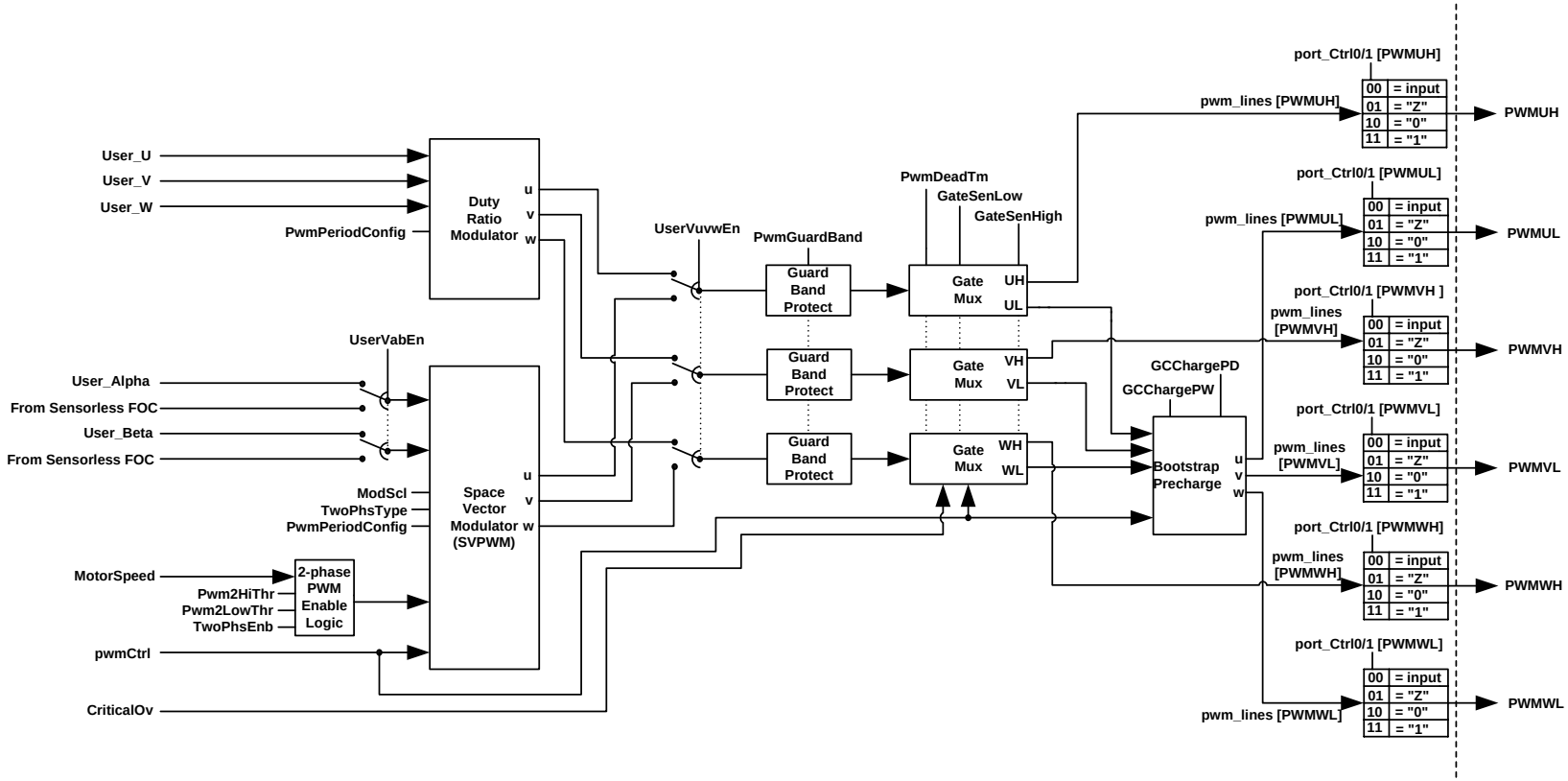


Figure 62. SVPWM Internal Block Diagram

4.3.5.1 SVPWM Transfer Characteristics

A three-phase two-level inverter with dc link configuration can have eight possible switching states, which generates the output voltage of the inverter. Each inverter switching state generates a voltage Space Vector (V1 to V6 active vectors, V0 and V7 zero voltage vectors) in the Space Vector plane, as shown in Figure 63. The magnitude of each active vector (V1 to V6) is 2/3 Vdc (dc bus voltage).

The user modulation inputs U_Alpha and U_Beta are related to the modulation depth by:

$$Umag = \sqrt{(U_Alpha^2 + U_Beta^2)}$$

The maximum achievable modulation in the linear operating range occurs when modulation (Umag) reaches Mod_Pk (default: 2355). Under such circumstance, the voltage vector touches the unit circle (Figure 63). The corresponding inverter line rms voltage is Vdc / sqrt (2) (Vdc – dc bus voltage). It is best to operate the PWM inverter in the linear range to minimize current harmonics.

Within the linear modulation range where Umag < 2355, the theorectical relationship between inverter output voltage (Vllrms) and modulation depth (Umag) is given by:

$$Vllrms = \frac{Umag \times Vdc}{Mod_Pk \times \sqrt{2}} \quad \text{[line-to-line volts rms]}$$

where $Mod_Pk = 2355$

Note: In practice, when Umag = 2355, the inverter output voltage will be slightly lower than the theoretical value ($Vdc/\sqrt{2}$) due to inverter losses, switching devices voltage drop and inverter blanking time insertion.

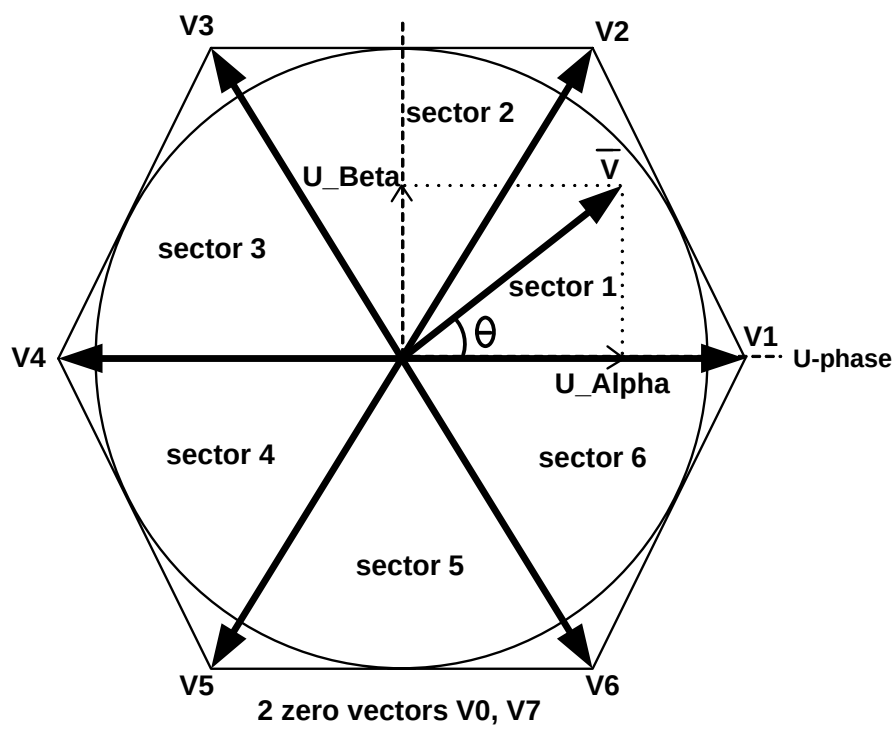


Figure 63. Space Vector Diagram

Over modulation occurs when modulation Umag > Mod_Pk. This corresponds to the condition where the voltage vector in 2 increases beyond the hexagon boundary. Under such circumstance, the Space Vector PWM algorithm will rescale the magnitude of the voltage vector to fit within the Hexagon limit. The magnitude of the voltage vector is restricted within the Hexagon; however, the phase angle (θ) is always preserved. The transfer gain of the PWM modulator reduces and becomes non-linear in the over modulation region. Voltage vector rescaling is illustrated in Figure 64

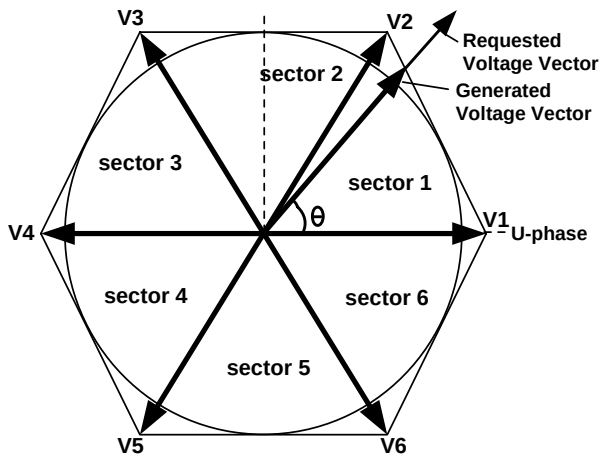


Figure 64. Voltage Vector Rescaling

4.3.5.2 Deadtime Insertion Logic

Blanking time is inserted to avoid shoot through between top and bottom devices of the same inverter leg. Register PwmDeadTm specifies the amount of inverter blanking time (dead time).

The deadtime insertion logic chops off the high side commanded volt*seconds by the amount of deadtime and adds the same amount of volt*seconds to the low side signal. Thus, it eliminates the complete high side turn on pulse if the commanded volt*seconds is less than the programmed deadtime.

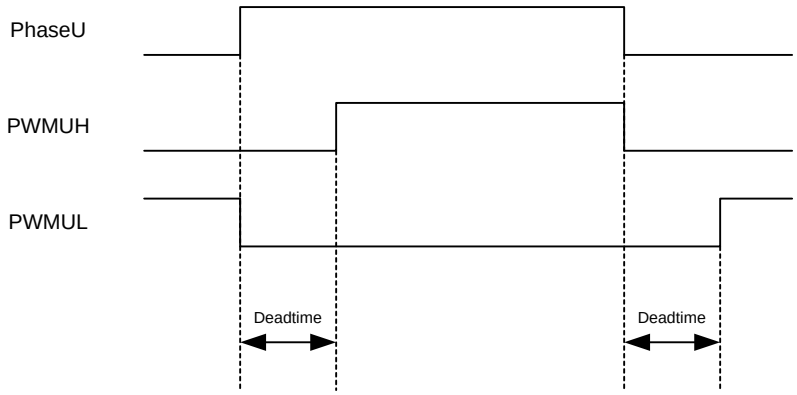


Figure 65. Deadtime Insertion

The deadtime insertion logic inserts the programmed deadtime between the high and low side gate signals within a phase, as shown in Figure 65. The deadtime register is also double buffered to allow "on the fly" deadtime change and control while PWM logic is active.

4.3.5.3 Three-Phase and Two-Phase Modulation

Three-phase and Two-phase Space Vector PWM modulation options are provided.

The Volt-sec generated by the two PWM schemes is identical under the same modulation depth input. However with two-phase modulation the switching instances per PWM cycle is reduced as shown in Figure 66. Therefore, total inverter loss is reduced and the loss reduction is significant especially when higher switching frequencies (>10KHz) are employed. Figure 66 shows the switching pattern for one PWM cycle when the voltage vector is inside sector 1.

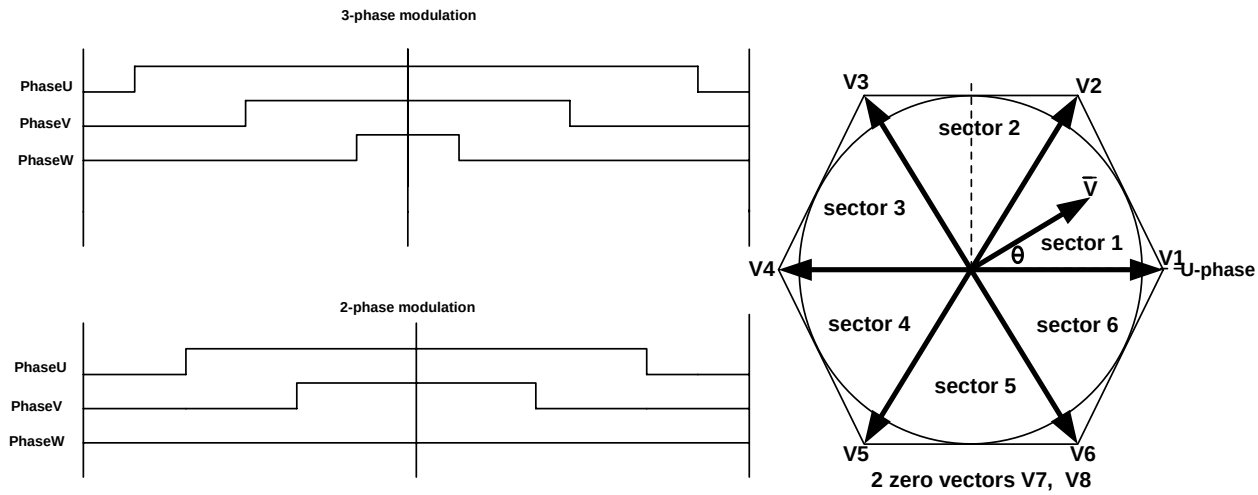
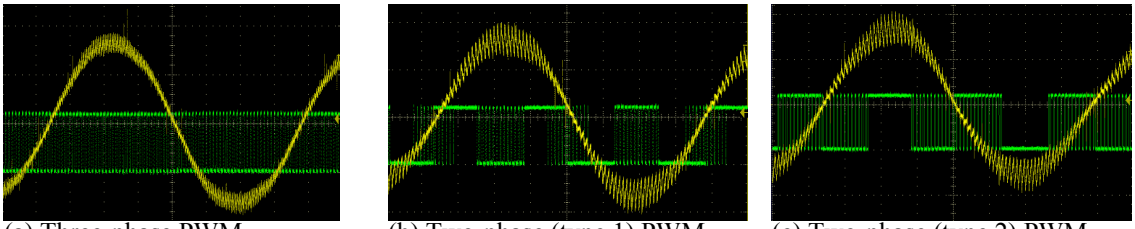


Figure 66. Three-Phase and Two-Phase Modulation

Two types of two-phase modulation schemes are provided. TwoPhsType (register TwoPhsCtrl, bit 1) specifies the selection. Figure 67 illustrates the inverter Pole voltage and motor current of various types of PWM schemes.



(a) Three-phase PWM (b) Two-phase (type 1) PWM (c) Two-phase (type 2) PWM

Figure 67. Types of Space Vector PWM

When control bit TwoPhsEnb (register TwoPhsCtrl, bit 0) is set to 1, the SVPWM algorithm permits the transitioning of Three-phase to Two-Phase SVPWM when motor speed exceeds speed threshold Pwm2HiThr. Three-phase SVPWM will resume when motor speed drops below speed threshold Pwm2LowThr.

4.3.5.4 Guard Band

PWM Guard band (PwmGuardBand) protection can be applied such that PWM switching at high modulation cannot migrate into the beginning and end of a PWM cycle (Figure 68). In some cases (depends on hardware design such as PCB layout), guard band insertion can improve feedback noise immunity for signals sampled near the beginning and end of a PWM cycle. However, Guard band insertion will reduce maximum achievable inverter output voltage. The default value of PwmGuardBand is zero.

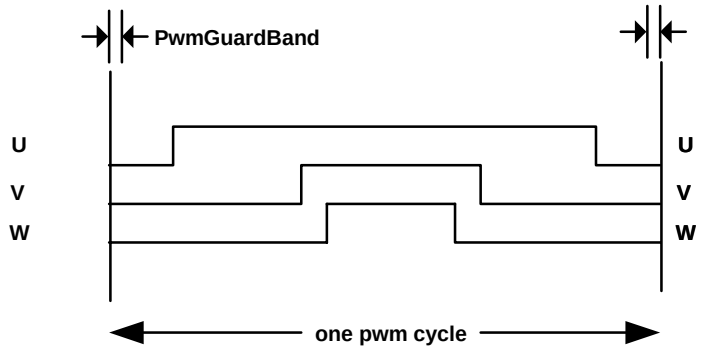


Figure 68. Guard Band Insertion

4.3.5.5 PWM Pre-Charge Control

Zero vector (low side devices on) is normally applied for the initial turn-on of the PWM inverter output. If a Bootstrap gate driver is used, the Bootstrap capacitors (u, v, w phase) will all be charged simultaneously. In some applications, the bootstrap capacitor charging current can be significantly higher than the motor rated current. This will cause a nuisance Itrip as soon as the inverter firing begins. The Bootstrap capacitor charging current can be significantly reduced by the built-in Pre-charge control function of the SVPWM module (Figure 62).

Instead of turning on all low side devices simultaneously for a prolonged duration, the gate Pre-charge control (register pwmctrl, bit 2) will schedule an alternating (u, v, w phase) charging sequence with programmable (GCChargePW) charging pulse duration. Figure 69 illustrates the pre-charge sequence and the corresponding dc link current. This current represents the charging current of the bootstrap capacitors (U, V, W phases). As can be seen from this figure, the charging current reduces significantly after two charging cycles (U→V→W).

PWMUL

PWMVL

PWMWL

Dc link current

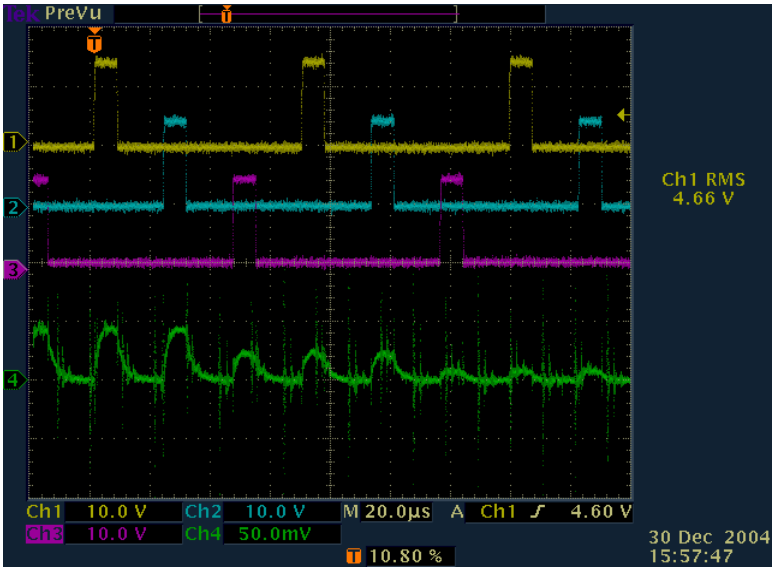


Figure 69. Bootstrap Pre-Charge Sequence

As shown in Figure 70, register GCChargePW controls the charging pulse duration while register GCChargePD controls the spacing between u, v and w phase charging. The spacing (GCChargePD) between consecutive charging is specified as number of charge pulses.

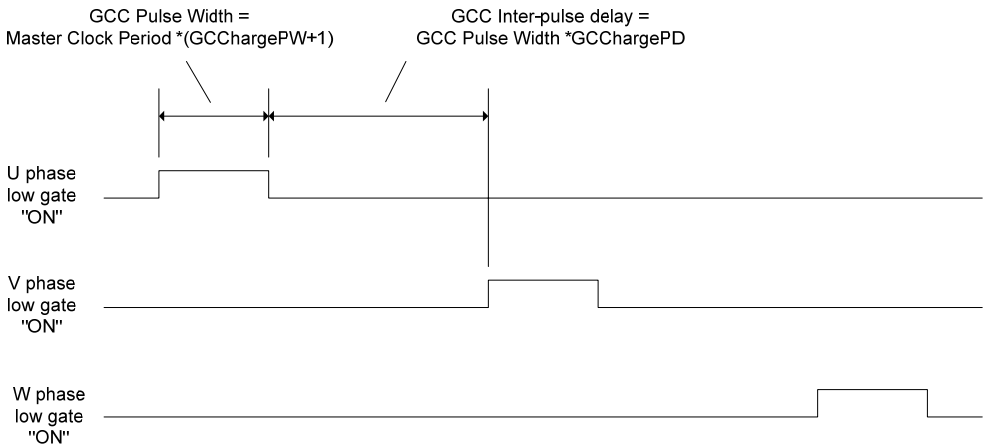


Figure 70. Timing of Bootstrap Capacitor Charging

4.3.5.6 Duty Ratio Control Mode

In order to provide increased pwm flexibility, a different type of PWM modulation (duty ratio) rather than Space Vector modulation can be realized in the SVPWM module. Users can bypass (Figure 62) the Space Vector modulator and apply duty ratio control for each motor phase. If register UserVuvwEn is set to one, SVPWM is bypassed and the duty ratio modulation mode is selected. In this mode, the duty ratio of each inverter phase can be independently controlled via User_U, User_V and User_W. Figure 71 illustrates the duty ratio control for U phase. Register PwmPeriodConfig is the number of digital counts corresponding to half of a PWM cycle. When User_U equals PwmPeriodConfig, 100% duty is achieved.

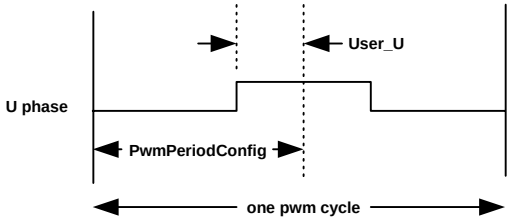


Figure 71. Duty Ratio Control

4.3.6 FAULTS Block

The FAULTS block provides outputs associated with the FaultFlags motion peripheral register defined in Section 4.4.20. Figure 72 shows the FAULTS block and its outputs are listed in Table 74. All outputs are Boolean values, with 0 indicating no fault condition and 1 indicating that the fault condition is present. The block has no inputs.

There is no execution time associated with the FAULTS block; it simply provides an interface to fault status information produced by other modules within the system.

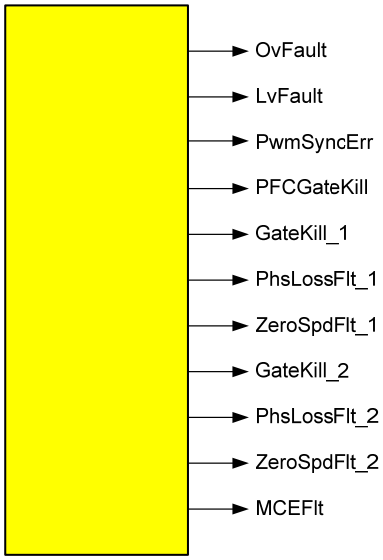


Figure 72. FAULTS Block

Signal name	Description	I/O	Type
OvFault	dc bus over voltage fault	Output	Boolean
LvFault	dc bus low voltage fault	Output	Boolean
PwmSyncErr	PWM synchronization error	Output	Boolean
PFCGateKill	PFC Gate kill fault	Output	Boolean
GateKill_1	Motor 1 Gate kill fault	Output	Boolean
PhsLossFlt_1	Motor 1 Phase loss fault	Output	Boolean
ZeroSpdFlt_1	Motor 1 Zero speed fault	Output	Boolean
GateKill_2	Motor 2 Gate kill fault	Output	Boolean
PhsLossFlt_2	Motor 2 Phase loss fault	Output	Boolean
ZeroSpdFlt_2	Motor 2 Zero speed fault	Output	Boolean
MCEFlt	MCE-generated fault	Output	Boolean

Table 74. FAULTS Block Outputs

4.3.7 MCE_FAULT Generator

The MCE_FAULT block is used to generate a fault condition from within the MCE design. Figure 73 shows the block. It has a single input, as described in Table 75. The Boolean input corresponds directly to the MceFault bit of the FaultClear register.

There is no execution time associated with the FAULTS block; it simply provides an interface to the MceFault bit.

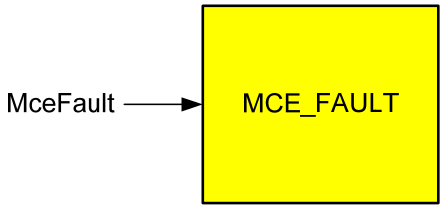


Figure 73. MCE_FAULT Block

Signal name	Description	I/O	Type
MceFault	MCE Fault condition	Input	Boolean 0 = No fault condition; 1 = Generate fault

Table 75. MCE_FAULT Block Inputs

4.3.8 PFC_PWM

This block generates the PFC PWM control pulses to the gate driver based on the duty cycle command produced in the PFC control loop, which is implemented in the MCE or 8051. Meanwhile, with a built-in PWM Blanking function, it provides a protection against a nuisance over-current fault situation when the AC input voltage becomes higher than the DC bus voltage under normal PFC operation. The PFC_PWM block also provides the flexibility of designing the circuit to operate as either full-mode boost PFC or partial-mode high-frequency boost PFC (IR patented).

The PFC_PWM library block provides an interface to a subset of the IRMCx31x motion peripheral registers that control and monitor the operation of the PFC_PWM module. Only some of these registers are accessible through the PFC_PWM block. Those that are typically initialized only once at system startup from an 8051 or host application are not provided since it is not necessary to update these registers from within the MCE design.

Figure 74 shows the PFC_PWM block and Table 76 lists its inputs and outputs.

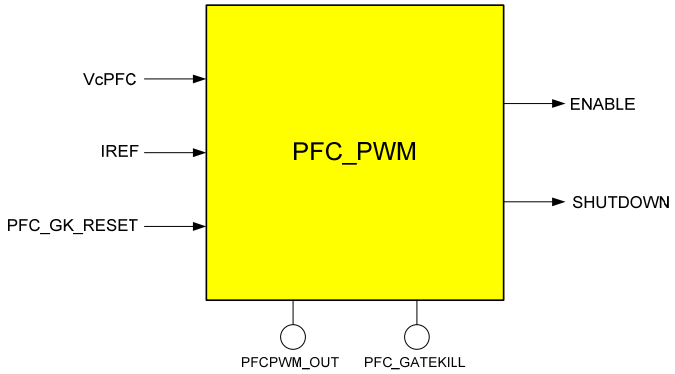


Figure 74. PFC_PWM Block

Signal name	Description	I/O	Type
VcPFC	PFC PWM duty cycle command	Input	16 bit, unsigned integer
IREF	Reference command of PFC current loop	Input	12 bit, unsigned integer
PFC_GK_RESET	PFC Gate kill reset	Input	Boolean, 0 = latch PFCGateKill fault 1 = clear latched PFCGateKill fault
ENABLE	PFC PWM enable status	Output	Boolean, 0 = PFC PWM output disabled 1 = PFC PWM output enabled
SHUTDOWN	ShutDown signal status	Output	Boolean, 0 = Shutdown inactive 1 = Shutdown active

Table 76. PFC_PWM Inputs and Outputs

Each of the PFC_PWM inputs and outputs corresponds to a motion control register or bit field within a register, as follows:

- Input VcPFC corresponds to write register VcPfcScaled (Section 4.4.19).
- Input IREF corresponds to write register True_IRef (Section 4.4.19).
- Input PFC_GK_RESET corresponds to bit field PFCGKReset in write register PFC_Ctrl (Section 4.4.19).
- Output ENABLE is a "mirror" of the value written to register PFCEnable (Section 4.4.19).
- Output SHUTDOWN corresponds to read register ShutDown (Section 4.4.26).

Note that outputs ENABLE and SHUTDOWN must be used in combination to determine whether PFC PWM output is enabled or disabled.

The internal block diagram in Figure 75 shows the two main components of the PFC PWM block: PWM Generation and PWM Blanking. This diagram shows all the motion peripheral registsers that are associated with the operation of the PFC PWM block. Those that are accessible through the PFC_PWM library block are shown in **bold** type. Those that are written from a host or 8051 application are shown in *italic* type.

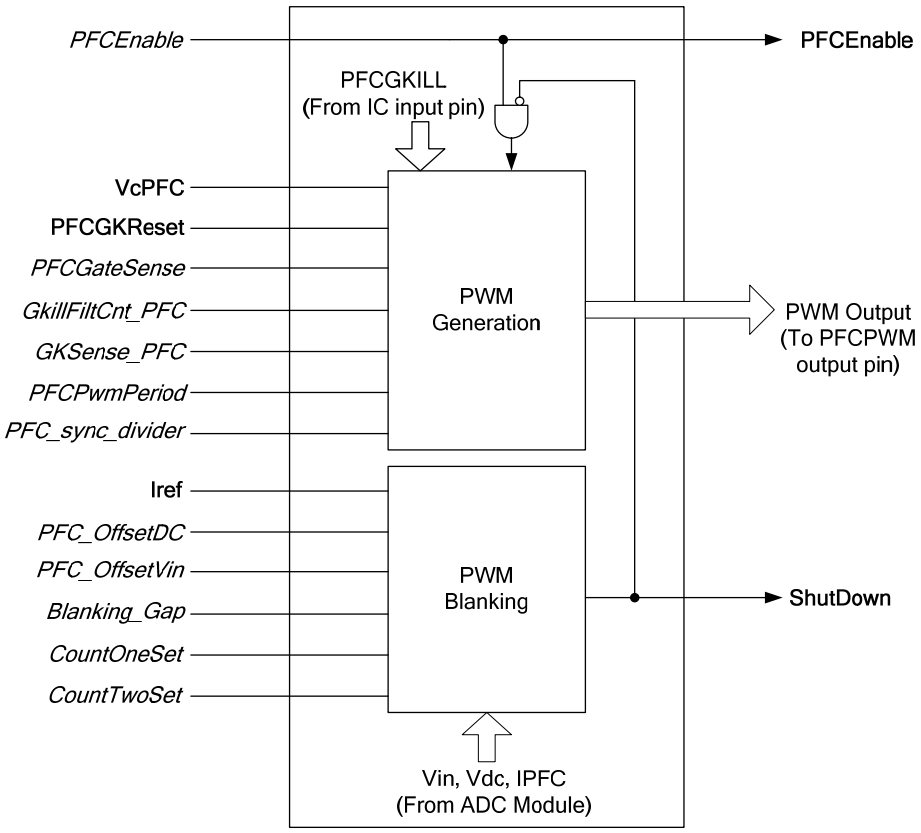


Figure 75. PFC_PWM Internal Block Diagram

4.3.8.1 PFC PWM Generation

The generation of the PWM output is based on a comparison between the duty cycle command VcPFC and a triangle PWM carrier. The PWM carrier frequency is set up by the register PFCPwmPeriod. The data update coherence is guaranteed by a SyncPulse-buffered design. A PFCPWM_Synpulse is generated based on the set up of the PWM carrier frequency. At every PFCPWM_Syncpulse, the latest VcPFC that is produced in the control loop is latched into the PWM Generation block. Prior to the arrival of the next PFCPWM_Syncpulse, this latched VcPFC signal is compared with the triangle PWM carrier. When the latched VcPFC is higher than the carrier, the PWM Output is High; when the latched VcPFC is lower than the carrier, the PWM Output is Low.

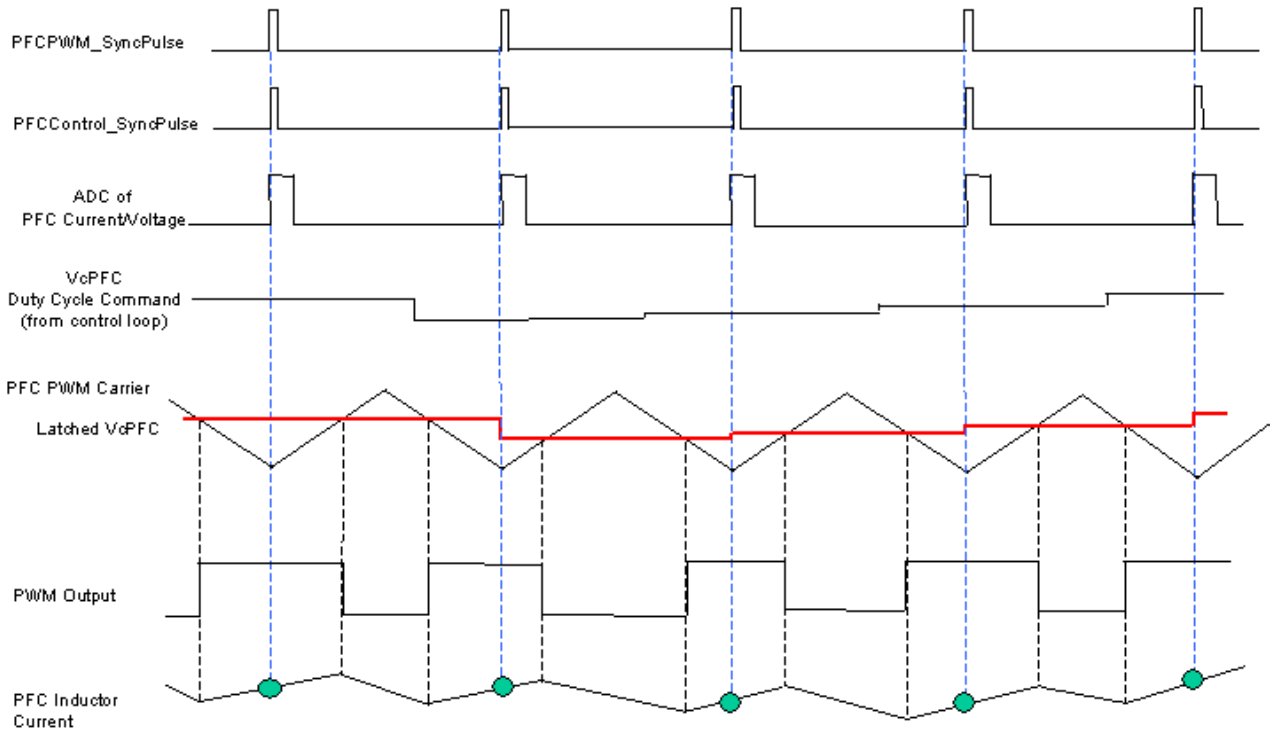


Figure 76. Generation of PFC PWM output and ADC timing (PFC_sync_divider = 0)

In addition to the PFCPWM_Syncpulse, there is a PFCControl_Syncpulse, which is synchronized with the PFCPWM_Syncpulse. The PFCControl_Syncpulse starts the A-to-D conversion of the PFC current and voltage signals, and then the execution of the PFC control loop. As can be seen from Figure 76 and Figure 77, the A/D sampling point occurs at the center of the period during which the PWM Output is High, which is the ON time of the PFC switching devices (IGBTs, MOSFETs), and is the center of the inductor current upslope (In Figure 76 and Figure 77, the step change in VcPFC is just for illustration. In reality, the duty cycle changes in two adjacent PWM cycles are very small). As a result, basically the average value of the PFC inductor current is sampled and converted to a digital signal for the control loop execution, and the switching noise has minimum influence on the ADC process. Normally for PFC designs, the choice of switching frequency, which is the PWM carrier frequency, involves many factors and trade-offs: control performance, current ripples, inductor size, switching losses, EMI noises, etc. This block provides users the flexibility to configure the ADC sampling rate to have different ratios with the PWM carrier frequency, using register PFC_sync_divider, as follows:

PFC_ADC Sampling Rate = PFC_PWM Carrier Freq. / (PFC_sync_divider[0:3] + 1)

Figure 77 shows the relationship between the ADC sampling rate and the PWM carrier frequency when PFC_sync_divider[0:3] = 1. By using the SyncDivider a high PFC carrier frequency can be achieved without increasing the computational load upon the Motion Control Engine.

There is one important restriction on the selection of the SyncDivider usage: The PFC_ADC Sampling Rate 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 the PFC_ADC Sampling Rate and the PWM carrier frequency over the course of one master PWM cycle.

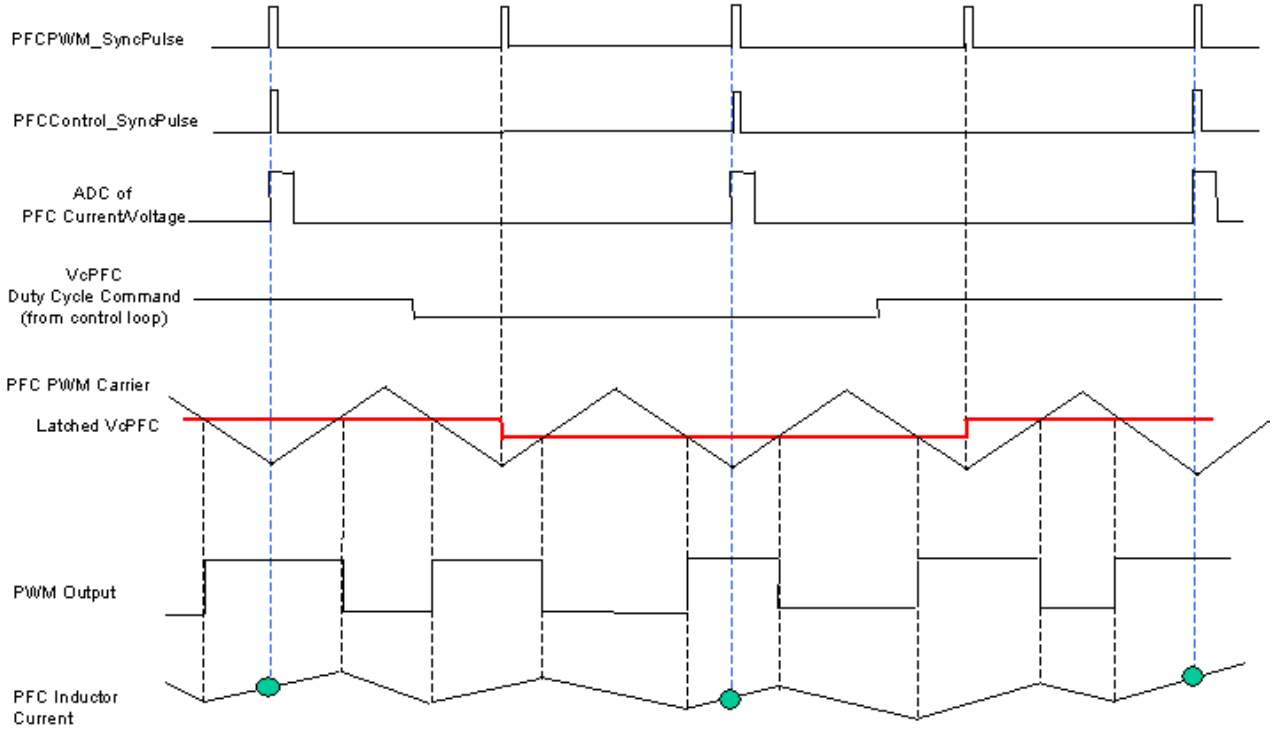


Figure 77. Generation of PFC PWM output and ADC timing (PFC_sync_divider = 1)

A further extension of the SyncDivider is called PFC Phasing. The PFC phasing offsets the PFC ADC sampling from the Master PWM Sync pulse (usually Motor 1). 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 value of PFCPhasing must be less than (PFCSyncRatio + 1), or else the PFC will not run at all.

Whenever a PFC GateKill occurs, the PWM Output is disabled. Meanwhile, the PWM Output is enabled/disabled by the combination of PFCEnable input and the ShutDown signal, which is the output the PWM_Blanking block: when PFCEnable = 1 and ShutDown = 0, enable PWM; when PFCEnable = 0 or ShutDown = 1, disable PWM.

4.3.8.2 PFC PWM Blanking

The full-mode boost PFC operation, including the conventional single-switch boost PFC circuit and the bridgeless PFC circuit, requires that the DC bus voltage must 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 could go to saturation because its volt-seconds can not be balanced. Consequently, high current can be generated and cause a nuisance over-current fault. The PWM_Blanking block provides protection by generating an output ShutDown signal.

This block instantly compares the DC bus voltage Vdc and AC input voltage Vin. 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 in Section 4.4.19).

Turn on/off conditions:

1) If Vin_Compare > Vdc_Compare + Blanking_Gap then Shutdown = 1

2) If Vin_comp > Vdc_comp & Blanking off for > 80 PWM cycles then Shutdown = 1

3) If CountTwoSet has expired & IPFC < IREF & Vin_Compare < Vdc_Compare + ¾*Blanking_Gap then Shutdown = 1

CountOneSet provides a minimum on time, in terms of PWM cycles, for the Shutdown signal. Once the minimum time has expired then CountTwo is incremented every PWM cycle only if IPFC is decreasing or near zero, otherwise CountTwo is reset to zero. CountTwo is the timer which determines if CountTwoSet has expired.

The PFC Blanking feature provides fast and smooth transitions between the PWM enable and disable modes. Note also that the integral component of the PFC current controller is reset on Shutdown. Users may also use this ShutDown output signal to enable and disable the PFC control loop (voltage, current, etc.), which can be implemented in either MCE or 8051.

4.3.9 PFC_SENSE

The PFC_SENSE module provides the ADC feedback signals that are used for the PFC control loop. Figure 78 shows the PFC_SENSE block and Table 77 lists its outputs. The PFC_SENSE block has no inputs.

The PFC_SENSE library block provides an interface to a subset of the IRMCx31x motion peripheral registers. All registers associated with the operation of the PFC_SENSE module are accessible through the PFC_SENSE block.

Each signal listed in the table corresponds directly to one of the motion peripheral registers described in Section 4.4. The register name associated with the signal is shown in the "Associated Register" column of the table. The rightmost column of the table provides a reference to the document section where the register is described. Each of the output signals can be traced through MCEDesigner. Each of these signals can be traced in MCEDesigner under the Signal name found in the leftmost colum.

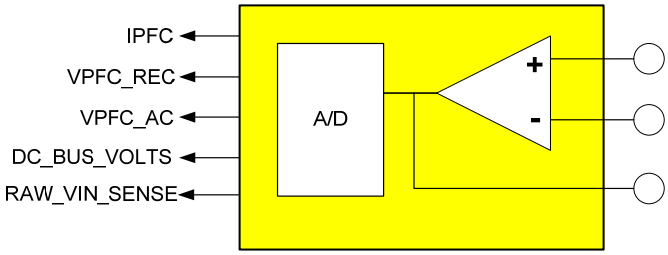


Figure 78. PFC_SENSE Block

Signal name	Description	I/O	Associated register	Reference for detailed description and scaling
IPFC	A/D feedback signal of AC input current, processed data	Output	I_IN	4.4.26
VPFC_REC	A/D feedback signal of AC input voltage, absolute value	Output	V_IN	4.4.26
VPFC_AC	A/D feedback signal of AC input voltage, bi-polar	Output	VinSense	4.4.26
DC_BUS_VOLTS	dc bus voltage feedback	Output	DcBusVolts	4.4.22
RAW_VIN_SENSE	Raw A/D feedback singal of AC input voltage, unprocessed	Output	RawVinSense	4.4.26

Table 77. PFC_SENSE Outputs

The PFC_SENSE block provides the ADC feedback signals that are used for the PFC control loop, in 12-bit digital counts: DC bus voltage, AC input current (IPFC), and AC input voltage. The AC input voltage is provided three times:

- VPFC_REC the absolute value, or rectified half-wave AC input voltage (register V_IN).
- VPFC_AC the bi-polar value, or bi-polar full-wave AC input voltage, offset corrected so that the average value corresponds to 2048 (register VinSense)
- RAW_VIN_SENSE Raw, unprocessed AC input signal from A/D converter (register RawVinSense)

For PFC control designs, VPFC_REC generates the half-wave sinusoidal reference for the current control loop, and VPFC_AC provides additional information such as line-voltage zero-crossing and timing. The RAW_VIN_SENSE signal is provided for use when the rectified AC voltage is sensed instead of the full bi-polar signal.

The AC input current (IPFC) is provided as corrected current feedback (register I_IN).

PFC sensing is synchronized with PFC PWM pulse generation, and the ADC sampling points of IPFC and VPFC occur at the center of the period during which the PWM Output is High. With this built-in feature, basically the average value of the PFC inductor current in every switching cycle is sampled and converted to a digital signal for the control loop execution, and the switching noise has minimum influence on the ADC process. For detailed ADC timing, refer to Figure 76 and Figure 77.

4.4 Motion Peripheral Registers

Certain aspects of IRMCx31x operation can be configured and monitored from an 8051 application or the MCEDesigner tool using a pre-defined register interface. These registers reside in MCE internal memory (not in the shared RAM). A special mechanism, defined in Section 5.2, is provided for reading and writing the registers. The sections below categorize the registers into functional groups and describe the purpose and format of each register.

4.4.1 System Write Register Group

port_ctrl0, port_ctrl1

Address:	*port_ctrl0: 04h* *port_ctrl1: 05h*	*Range:*	*Unsigned input* *with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter provides configuration of PWM output pins for the IC. Each PWM IC output pin can be configured independently to one of four different states, as follows:

00 – normal pwm state (connect to normal PWM pattern)
01 – "Z" high impedance state
10 – "0" zero state
11 – "1" one state

Each PWM IC output pin is controlled by writing one of the values defined above to a two-bit field of the port_ctrl0 or port_ctrl1 register. The port_ctrl0 register controls motor 1 and the port_ctrl1 register controls motor 2, with the bit fields defined as follows:

Bits 0 – 1	**CfgPWMUH**	Configure U phase high side
Bits 2 – 3	**CfgPWMUL**	Configure U phase low side
Bits 4 – 5	**CfgPWMVH**	Configure V phase high side
Bits 6 – 7	**CfgPWMVL**	Configure V phase low side
Bits 8 – 9	**CfgPWMWH**	Configure W phase high side
Bits 10 – 11	**CfgPWMWL**	Configure W phase low side

The PFC PWM output is controlled by writing one of the values defined above to a two-bit field of register port_ctrl0, as follows:

Bits 12 – 13	**CfgPFCPWM**	Configure PFC PWM

Bits 14 and 15 of port_ctrl0 and bits 12 – 15 of port_ctrl1 are unused and should be set to zero.

PwmMasterSel

Address:	*00h*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 = Select Motor 1 as master
1 = Select Motor 2 as master

Description: This parameter selects the master for PWM synchronization in multi-motor (Motor 1, motor 2 and PFC) systems. It is applied when PWM synchronization is enabled (register PwmSyncEnb). The slowest switching frequency pwm should be selected as master. When PwmSyncEnb = 1, the controller will check (on-line) whether all pwms are synchronized to the master. iMotion MCEWizard enforced synchronization by calculating appropriate pwm counter values (PwmPeriodConfig for both motor 1, 2 and PFCPwmPeriod). In practice, there should not be synchronization problem due to pwm counters being set by MCEWizard.

However, if users manually override pwm counter values that causes loss of synchronization, a synchronization fault (FaultFlags: PwmSyncErr) will be generated.

PwmSyncEnb

Address:	*01h*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 – Disable PWM synchronization
1 – Enable PWM synchronization

Description: This parameter enables PWM synchronization between motor 1, motor 2 and PFC. It is recommended to enable PWM synchronization for multi-motor systems. iMotion MCEWizard determines this parameter based on user's inputs.

AdcBaudDiv

Address:	*03h*	*Range:*	*Unsigned input* *0 – 63*	*Reset value:*	*11*

Scaling or Notation: SysClk / 6 MHz
where SysClk is the system clock frequency in MHz.

Description: This parameter specifies the counter reset value in maintaining fixed time strobes (for the A/D converter) when a system clock value (SysClk) other than the default is used. iMotion MCEWizard calculates this parameter.

FaultClear

Address:	*07h*	*Range:*	*Unsigned input* *with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This register can be used to clear latched drive faults generated inside the Faults module. It is also used for setting a fault condition generated from the MCE processor. The bit fields are defined as follows:

Bit 0	**FaultClear**	
	0	No action
	1	Clear all faults
Bit 1	**MceFault**	
	0	No action
	1	Set an MCE fault condition
Bits 2 – 15	Unused; set to zero	

SyncStDiv

Address:	*02h*	*Range:*	*Unsigned input* *0 – 4095*	*Reset value:*	*1842*

Scaling or Notation: Time strobe duration = SyncStDiv / SysClk [usec]
where SysClk is the system clock frequency in MHz.

Description: This parameter specifies a counter reset value for deriving fixed time strobes (for execution of modules inside Sensorless FOC block) when a system clock rate (SysClk) other than the default is selected.

syscfg

Address:	*06h*	*Range:*	*Unsigned input with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This register is used to configure system-wide options. The bit fields are defined as follows:

Bit 0 **DcMonitorEn**
- 0 Disable dc bus overvoltage and undervoltage protection
- 1 Enable dc bus overvoltage and undervoltage protection

Bit 1 **PfcSampleDisable**
- 0 Enable PFC feedback sample
- 1 Disable PFC feedback sample

Bit 2 **Mtr2SampleDisable**
- 0 Enable motor 2 feedback sample
- 1 Disable motor 2 feedback sample

Bits 3 – 15 Unused; set to zero

iMotion MCEWizard sets up system configuration bits (syscfg) based on user's inputs. For instance: if single motor reference design platoform is selected, bit 1 and 2 will be set to 1.

4.4.2 PWM Configuration Write Register Group

GCChargePD

Address:	*13h (Motor1)* *5Ch (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*255*

Scaling or Notation: Number of charge pulse spacing = GCChargePD.

Description: This parameter specifies the number of charging pulses spacing between motor phases (u, v, w) for the Bootstrap Pre-charge function.

Zero vector (low side devices on) is normally applied for the initial turn-on of the PWM inverter output. If the Bootstrap gate driver is used, the Bootstrap capacitors (u, v, w phase) will all be charged simultaneously. This charging current can be significantly reduced by the built-in Pre-charge control function.

Instead of charging all low side devices simultaneously, the gate Pre-charge control (activated using the Precharge bit of register pwmctrl, see Section 4.4.4) will schedule an alternating (u, v, w phase) charging sequence with programmable charging pulse duration (register GCChargePW).

Parameter GCChargePW controls the charging pulse duration while parameter GCChargePD controls the spacing between u, v and w phase charging. The spacing (GCChargePD) between consecutive charging is specified as number of charge pulses. Figure 70 illustrates this relationship.

GCChargePW

Address:	*12h (Motor1)* *5Bh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*255*

Scaling or Notation: Gate Precharge duration = $\frac{GCChargePW + 1}{SysClk}$ [usec]

where $SysClk$ is the system clock frequency in MHz.

Description: This parameter specifies the Gate Pre-charge duration. Zero vector (low side devices on) is normally applied for the initial turn-on of the PWM inverter output. If the Bootstrap gate driver is used, the Bootstrap capacitors (u, v, w phase) will all be charged simultaneously. This charging current can be reduced by the built-in Pre-charge control function.

Instead of charging all low side devices simultaneously, the gate Pre-charge control (activated using the Precharge bit of register pwmctrl, see Section 4.4.4) will schedule an alternating (u, v, w phase) charging sequence with programmable charging pulse duration.

Parameter GCChargePW controls the charging pulse duration while parameter GCChargePD controls the spacing between u, v and w phase charging. Figure 70 illustrates this relationship.

ModScl

Address:	*10h (Motor1)* *59h (Motor2)*	*Range:*	*Unsigned input* *0 – 65535*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter provides scaling (between modulation depth in digital count to inverter output voltage in voltages rms) for the Space Vector PWM. In 300-series iMOTION products, the setting of ModScl is calculated by iMotion MCEWizard to target an inverter line-to-line output voltage of $Vdc/\sqrt{2}$ rms volts at 2355 digital count of modulation depth (input to the Space Vector modulator, $\sqrt{U_Alpha^2 + U_Beta^2}$). For instance, if U_Alpha = 2355 and U_Beta = 0, then the inverter output line-to-line voltage is equal to $Vdc/\sqrt{2}$ volts rms. If users reduce the value of ModScl by 50%, then the inverter output voltage will be half ($0.5 \times Vdc/\sqrt{2}$) at the same modulation (U_Alpha = 2355 and U_Beta = 0).

pwmcfg

Address:	*15h (Motor1)* *5Eh (Motor2)*	*Range:*	*Unsigned input* *with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter configures the logic sense of the gate signal. The setting of each Gate Sense bit (bits 2 – 4) is defined as follows:

0 = low true gating convention
1 = high true gating convention

The bit fields are defined as follows:

Bits 0 – 1	Unused; set to zero	
Bit 2	**GateSenLow**	Gate sense low side devices
Bit 3	**GateSenHigh**	Gate sense high side devices
Bit 4	**GateSenGateKill**	Sense for Gate kill
Bits 5 – 15	Unused; set to zero	

PwmDeadTm

Address:	*0Fh (Motor1)* *58h (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*60*

Scaling or Notation: Inverter blanking time = PwmDeadTm / SysClk [usec]
where SysClk is the system clock frequency in MHz.

Description: Inverter blanking time for avoiding shoot through between high side and low side devices. The blanking time setting is power device and application dependent. In some applications, power device switching is intentionally slowed down to reduce EMI noise. Hence inverter blanking time is extended to accommodate slower switching profile.

PwmGuardBand

Address:	*0Eh (Motor1)* *57h (Motor2)*	*Range:*	*Unsigned input* *0 – 1023*	*Reset value:*	*0*

Scaling or Notation: Guard band duration = PwmGuardBand / SysClk [usec]
where SysClk is the system clock frequency in MHz.

Description: This parameter provides a guard band such that PWM switching at high modulation cannot migrate into the beginning and end of a PWM cycle. The guard band insertion can improve feedback noise immunity for signals sampled near the beginning and end of a PWM cycle. The parameter only applies to the 3-phase Space Vector modulation scheme. The guard band duration is independent of PWM carrier frequency.

Note: Guard band insertion will reduce the maximum achievable inverter output voltage.

PwmPeriodConfig

Address:	*11h (Motor1)* *5Ah (Motor2)*	*Range:*	*Unsigned input* *0 – 16383*	*Reset value:*	*5999*

Scaling or Notation: See description.

Description: This parameter specifies the number of digital counts of the SVPWM counter for half of a pwm cycle. The resolution of pwm is affected by the system clock (SysClk) being used. And this parameter is directly related to the system clock and the inverter switching frequency being chosen. The relationship is given by:

$$\text{PwmPeriodConfig} = \frac{SysClk \times 1000}{2 \times FreqPwm} - 1$$

where:

$SysClk$ is the system clock in MHz and

$FreqPwm$ is the inverter switching frequency in KHz.

There is a restriction regarding the relationship between system clock rate (SysClk) and PWM frequency (FreqPwm). Refer to Section 3.2.3 and Figure 10 for more information.

Pwm2HiThr

Address:	*0Ch (Motor1)* *55h (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Speed threshold = $0.007813 \times MaxRpm \times Pwm2HiThr$ [rpm]

where $MaxRpm$ is the maximum application speed in rpm. It is a required entry ("Max RPM") in the MCEWizard.

Description: This parameter defines the upper speed threshold for switching from 3-phase to 2-phase PWM. The transition between 3-phase and 2-phase PWM is done using a hysteresis band on speed.

Pwm2LowThr

Address:	*0Dh (Motor1)* *56h (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Speed threshold = $0.007813 \times MaxRpm \times Pwm2LowThr$ [rpm]

where $MaxRpm$ is the maximum application speed in rpm. It is a required entry ("Max RPM") in the MCEWizard.

Description: This parameter defines the lower speed threshold for switching from 2-phase to 3-phase PWM. The transition between 3-phase and 2-phase PWM is done using a hysteresis band on speed.

TwoPhsCtrl

Address:	*FCh (Motor1)* *FDh (Motor2)*	*Range:*	*Unsigned input* *with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the setup for 2-phase modulation. If bit TwoPhsEnb = 0, 3-phase modulation will always be used. If bit TwoPhsEnb = 1, inverter PWM will transition from 3-phase to 2-phase PWM modulation whenever the absolute motor speed exceeds the threshold specified by register Pwm2HiThr and transitions back to 3-phase mode whenever the motor speed falls below the threshold specified by register Pwm2LowThr. Refer to Section 4.3.5.3 for a discussion of PWM modulation types.

Bit 0 **TwoPhsEnb**
- 0 Disable 2-phase modulation
- 1 Enable 2-phase modulation

Bit 1 **TwoPhsType**
- 0 Select 2-phase modulation type 1
- 1 Select 2-phase modulation type 2

GkillFiltCnt

Address:	*14h (Motor1)* *5Dh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*59*

Scaling or Notation: Number of system clock cycles for gatekill fault = GkillFiltCnt (see description)

Description: Parameter GkillFiltCnt defines the number of system clock cycles that the Gatekill signal (IC input pin GATEKILL) must persist before a latched fault (pwm shut down) is generated. This is done to avoid nuisance drive shut down due to noise. .

4.4.3 Current Feedback Configuration Write Register Group

IfbkScl

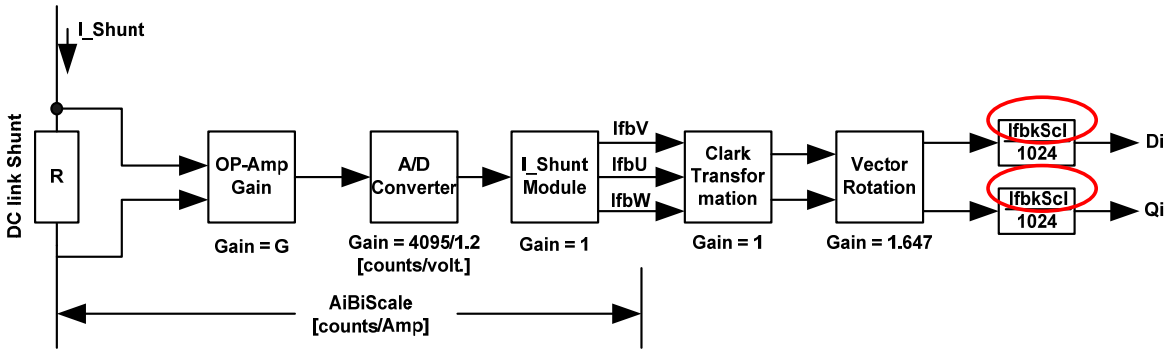
Address:	*16h (Motor1)* *5Fh (Motor2)*	*Range:*	*Signed input* *-16384 – 16383*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter provides current gain such that 4095 digital counts of d-axis or q-axis current represents rated motor current. IfbkScl is calculated in the MCEWizard and is a function of motor rated Amps and analog current scaling (how many amps per volt of A/D input). The following block diagram shows the motor current feedback path of the iMotion controller IC and the involvement of this gain parameter (IfbkScl). The current feedback path involves Shunt resistor, analog signal conditioning (OP-Amp), A/D converter, Single shunt current reconstruction, Clark transformation and Vector rotator. Each component of the feedback path has its associate gain scaling which is shown in the figure. The scaling of IfbkScl is calculated such that at I_Shunt = Rated motor Amps, the value of current magnitude ($\sqrt{Qi^2 + Di^2}$)is equal to 4095 digital counts. For instance: if Di (d-axis current) is regulated to zero (SPM motors) value, Qi will be equal to 4095 when I_Shunt = Rated motor Amps. As shown in the diagram below, based on the target objective of obtaining 4095 digital counts for rated motor current, the current feedback gain (IfbkScl) can be calculated by:

$$\text{IfbkScl} = \frac{4095 \times 1024}{1.647 \times AiBiScale \times RatedMotorAmps \times \sqrt{2}}$$

Where: RatedMotorAmps is in rms Amps. This calculation for the current feedback gain is embedded in the iMotion MCEWizard.



AiBiScale = R x G x 4095 / 1.2 [counts/Amp]

Qi = (IfbkScl / 1024) x 1.647 x AiBiScale x I_Shunt [counts]

Where:
Di and Qi are the d and q current in digital count.
I_Shunt is the dc link current in Amps.
G is the analog amplifier gain. This amplifier is inside the Control IC.
R is the shunt resistance value in ohms.

4.4.4 System Control Write Register Group

pwmctrl

Address:	*17h (Motor1)* *60h (Motor2)*	*Range:*	*Unsigned input* *with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This register is used to configure drive control parameters. These control bits are normally manipulated by a master drive sequencer to perform startup control and shut down control.

Bit 0	**PwmEnable**	
	0	Disable PWM
	1	Enable PWM
Bit 1	**FocEnable**	
	0	Disable Field-Oriented Control regulators
	1	Enable Field-Oriented Control regulators
Bit 2	**Precharge**	
	0	Disable gating pre-charge for Bootstrap capacitors
	1	Enable gating pre-charge for Bootstrap capacitors
Bit 3	**PwmGateEnb**	
	0	Disable PWM gatings
	1	Enable PWM gatings
Bits 4 – 15	Unused; set to zero	

4.4.5 Torque Loop Configuration Write Register Group

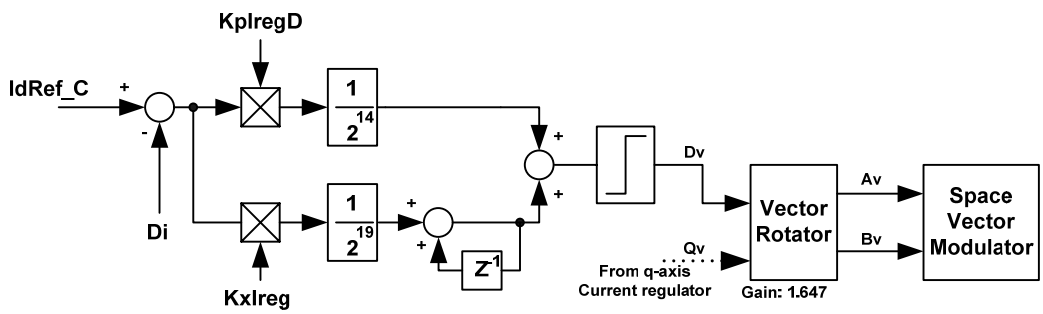
KpIreg

Address:	*18h (Motor1)* *61h (Motor2)*	*Range:*	*Unsigned input* *0 – 32767*	*Reset value:*	*0*

Scaling or Notation: % modulation index = KpIreg × 0.01748 × IqErr / RatedMotorAmps [%]
Where: *IqErr* is absolute current error (abs(command – feedback)) of q-axis in Amps

Description: This parameter specifies the proportional gain of the q-axis current regulator. The parameter relates current error to modulation index. 100% modulation corresponds to the maximum achievable value of the SVPWM linear range. The corresponding theoretical rms motor line voltage at 100% modulation is $Vdc/\sqrt{2}$.

The d-axis channel current regulator gain scaling is shown in the figure below. Q-axis current regulator structure is identical (replace gain KpIregD with KpIreg) to the d-axis. This figure illustrates the gain scaling starting from current error (IdRef_C – Di) up to the input of the Space Vector Modulator. Note: Only gain scalings are illustrated in the figure, antiwindup, dc bus compensation and preservation of numerical truncation are not shown.



KpIregD

Address:	*19h (Motor1)* *62h (Motor2)*	*Range:*	*Unsigned input* *0 – 32767*	*Reset value:*	*0*

Scaling or Notation: % modulation index = KpIregD × 0.01748 × IdErr / RatedMotorAmps [%]
Where: *IdErr* is absolute current error (Abs(command – feedback)) of q-axis in Amps

Description: This parameter specifies the proportional gain of the d-axis current regulator. The parameter relates current error to modulation index. 100% modulation corresponds to the maximum achievable value of the SVPWM linear range. The corresponding theoretical rms motor line voltage at 100% modulation is $Vdc/\sqrt{2}$. (see also KpIreg for internal scaling).

KxIreg

Address:	*1Ah (Motor1)* *63h (Motor2)*	*Range:*	*Unsigned input* *0 – 32767*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the integral gain of the d-axis and q-axis current regulator. The parameter relates current error second (current error integration) to modulation index. The scaling depends on the current regulator execution rate which is directly related to the pwm frequency.

Scaling: Time duration = $\frac{1430 \times 2^{19} \times PwmPeriod}{Ierr \times KxIreg}$ [sec.]

where Ierr is the current error in digital count (4095 = rated motor current) and PwmPeriod is the pwm period in sec. The above scaling defines the time required to reach full (100%) modulation (Dv or Qv) when a fixed current error (Ierr) is present. 100% modulation corresponds to the maximum achievable value of the SVPWM linear range. The corresponding theoretical rms motor line voltage at 100% modulation is $Vdc/\sqrt{2}$. As can be seen from this scaling, when PwmPeriod increases (lower pwm switching frequency), KxIreg has to be increased to maintain integral gain strength (time duration to reach 100% modulation). (see KpIreg for internal scaling diagram).

VdLim

Address:	*1Ch (Motor1)* *65h (Motor2)*	*Range:*	*Unsigned input* *0 – 2047*	*Reset value:*	*0*

Scaling or Notation: 1430 = 100 [% modulation]

Description: This parameter specifies the d-axis current regulator output limit. 100% modulation corresponds to the maximum achievable value of the SVPWM linear range. The corresponding theoretical rms motor line voltage at 100% modulation is $Vdc/\sqrt{2}$.

VqLim

Address:	*1Bh (Motor1)* *64h (Motor2)*	*Range:*	*Unsigned input* *0 – 2047*	*Reset value:*	*0*

Scaling or Notation: 1430 = 100 [% modulation]

Description: This parameter specifies the q-axis current regulator output limit. 100% modulation corresponds to the maximum achievable value of the SVPWM linear range. The corresponding theoretical rms motor line voltage at 100% modulation is $Vdc/\sqrt{2}$.

4.4.6 Velocity Control Write Register Group

Rotation

Address:	*42h (Motor1)* *8Bh (Motor2)*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 = positive rotation (u-v-w) ; 1 = negative rotation (u-w-v)
Description: This parameter defines the direction of rotation.

Start_Lim

Address:	*3Dh (Motor1)* *86h (Motor2)*	*Range:*	*Unsigned input* *0 – 16383*	*Reset value:*	*0*

Scaling or Notation: 4095 = 100 [% motor Amps]
Description: This parameter specifies the desired Startup current when the Startup diagnostic mode is selected (MtrCtrlBits[1]). The motor will be accelerated using this startup current until a successful startup has been detected (StartOk bit in register StatusFlags; see Section 4.4.21). Typical settings for this parameter range from 50 to 120% (application dependent) of rated motor Amps. If the Startup diagnostic mode is not selected, then the current will be set by the TrqRef parameter.

TrqRef

Address:	*3Eh (Motor1)* *87h (Motor2)*	*Range:*	*Signed input* *-16384 – 16383*	*Reset value:*	*0*

Scaling or Notation: 4095 = 100 [% rated motor Amps]
Description: Command current input to inner-loop regulator (inside Sensorless FOC block, Figure 51). In practice, this signal is fed by the output of a speed regulator.

MotorSpeed

Address:	*3Fh (Motor1)* *88h (Motor2)*	*Range:*	*Signed input* *-16384 – 16383*	*Reset value:*	*0*

Scaling or Notation: 16383 = MaxRpm [rpm]
Description: This speed input is used by the SVPWM block for determining switchover between 3-phase and 2-phase PWM modulation scheme. In the 300 series reference design, the raw motor frequency (Rtr_Freq) output of the Sensorless FOC block is scaled and filtered (typically 3 – 7 msec.) to form a filtered speed signal which is used for speed regulation and to drive the MotorSpeed input of the SVPWM block.

Min_Spd

Address:	*40h (Motor1)* *89h (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Minimum speed $= \dfrac{Min_Spd \times MaxRPM}{2048}$ [rpm]

where $MaxRPM$ is the maximum application speed, entered in the MCEWizard ("Max RPM").

Description: This parameter defines the minimum permissible drive operating speed level. It is used to detect a zero speed trip fault and also for minimum command speed clamping (in MCE application software). A zero speed fault will be generated if motor speed falls below half of the value of minimum drive speed for a time duration of more than two seconds. The check for zero speed fault can be disabled through bit field ZeroSpdDisable of register MtrCtrlBits (Section 4.4.9).

VFFreq

Address:	*41h (Motor1)* *8Ah (Motor2)*	*Range:*	*Unsigned input* *-16384 – 16383*	*Reset value:*	*0*

Scaling or Notation: Frequency = VFFreq × 0.01552583 [Hz]

Description: This parameter specifies the Volts per Hertz frequency command for the purpose of the open-loop diagnostic. In this diagnostic mode, the command target frequency (VFFreq) is multiplied by VFGain (see Section 4.4.8) to generate a modulation index, thereby maintaining a constant Volts/Hz ratio under constant dc bus operation. This mode of operation generates prescribed inverter voltages without requiring current feedbacks. Therefore, it can be used to drive passive load or Induction motor load for troubleshooting hardware (feedback and PWM) related issues.

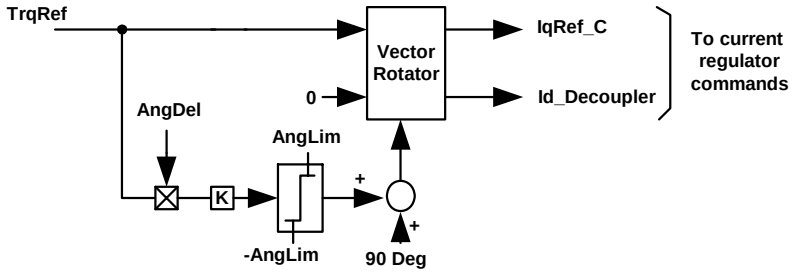
4.4.7 Closed Loop Angle Estimator Write Register Group

AngDel

Address:	*34h (Motor1)* *7Dh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Angle advancement = AngDel × 0.35156 × I_Motor / Rated Motor Amps [Deg]
Where I_Motor is the operating motor current in Amps

Description: This parameter provides gain adjustment for current angle advancement. The current angle advancement is added to a fixed defaulted phase (90 Deg) and the rotor angle to form the relative phasing of the current vector. Current angle advancement is required for Permanent Magnet motor with rotor saliency (Interior Permanent Magnet Motors). A value of zero represents zero angle advancement and therefore the current vector is placed at 90 degrees with respect to the rotor angle. Details on angle advancement function are given in the Application Developer's Guide under the Interior Permanent Magnet Motor Control section. Diagram below shows the implementation of the angle advancement function and the related controller parameters.



AngLim

Address:	*35h (Motor1)* *7Eh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Angle limit = AngLim × 0.17578 [Deg.]

Description: This parameter provides the maximum limit on the current angle phase advancement specified by register AngDel. Details on angle advancement function are given in the Application Developer's Guide (Interior Permanent magnet motors). (see also AngDel).

AtanTau

Address:	*32h (Motor1)* *7Bh (Motor2)*	*Range:*	*Unsigned input* *0 – 32767*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter provides angle compensation (frequency dependent) for the phase shift introduced by flux integration time constant (register FlxTau; see Section 4.4.7). Inside the Sensorless FOC module (Figure 51 Angle Frequency generator) , frequency (Rtr_Freq) is multiplied by a time contant (AtanTau) to form a compensating angle. This angle represents the phase shift introduced by the non-ideal flux integrators (low pass filter). Pure (ideal) integrator cannot be used due to dc offset problem. The flux integration time constant is an entry of the iMotion MCEWizard. Typical range of integrator time constant is in the range of 0.01 to 0.025 sec.

Scaling: Time constant = $\dfrac{A\tan Tau \times PwmPeriod}{2 \times PI \times FreqScl}$ [sec.]

where $FreqScl$ is determined by:

if (Use4xFreqScl = 1) FreqScl = 4
else if (Use2xFreqScl = 1) FreqScl = 2

else FreqScl = 1

(Use4xFreqScl and Use2xFreqScl are bit fields of the MtrCtrlBits_S and MtrCtrlBits registers, respectively. See Section 4.4.9.). Both MtrCtrlBits_S and MtrCtrlBits are populated by iMotion MCEWizard. (see Figure 79 for details)

FlxAInit

Address:	*28h (Motor1)* *71h (Motor2)*	*Range:*	*Signed input* *-128 – 127*	*Reset value:*	*0*

Scaling or Notation: Initial Alpha flux = FlxAInit × 100 / 78 [% rated flux]

Description: This parameter specifies the initial Alpha flux level right after parking stage of motor startup. The initial Alpha flux level dependents on the parking angle (ParkAng). MCEWizard calculates this parameter based on the parking angle setup.

FlxBInit

Address:	*29h (Motor1)* *72h (Motor2)*	*Range:*	*Signed input* *-128 – 127*	*Reset value:*	*0*

Scaling or Notation: Initial Beta flux = FlxBInit × 100 / 78 [% rated flux]

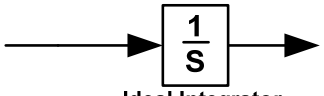
Description: This parameter specifies the initial Beta flux level right after parking stage of motor startup. The initial Beta flux level dependents on the parking angle (ParkAng). MCEWizard calculates this parameter based on the parking angle setup.

FlxTau

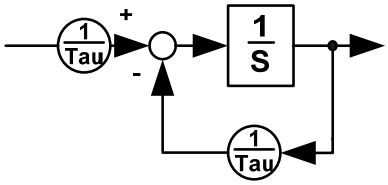
Address:	*2Ch (Motor1)* *75h (Motor2)*	*Range:*	*Unsigned input* *0 – 8191*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: Motor flux is calculated by integration of estimated voltages. Pure (ideal) integrator cannot be used due to dc offset problem. The integration is done using non-ideal integrator (low pass filter) as shown in the diagram below. The flux integration time constant (Tau) is an entry of the iMotion MCEWizard. Typical range of non-ideal integrator time constant is in the range of 0.01 to 0.025 sec.



Ideal Integrator



Non-ideal integrator

This parameter provides the adjustment for the integrator time constant. FlxTau is inversely proportional to the "Flux estimator time constant" entered in the MCEWizard. The relationship of the Flux estimator time constant and FlxTau is given by:

$$\text{Flux estimator time constant} = \frac{2^{18} \times PwmPeriod}{FlxTau} - PwmPeriod \quad \text{[sec.]}$$

where $PwmPeriod$ = 1/(PWM switching frequency) [sec].

FreqBW

Address:	*33h (Motor1)* *7Ch (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Filter bandwidth = FreqBW / (8192 × pwm cycle time) [rad/sec]
Where: pwm cycle time is the pwm period in sec.

Description: This parameter specifies the filter (first order) bandwidth for estimated motor frequency. This filtered motor frequency is applied to correct the phase shift introduced by the flux integration time constant (register FlxTau). The phase correction is done by taking arctan of filtered frequency x integrator time constant Tau. Since the flux integrator is a first order low pass filter, its phase shift can be easily compensated by arctan of frequency x Tau (see also Figure 79 for details).

IScl

Address:	*2Eh (Motor1)* *77h (Motor2)*	*Range:*	*Unsigned input* *0 – 32767*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the current gain scaler for the flux estimator. The MCEWizard calculates this parameter to provide appropriate drive current scaling. This current scaling is inside the flux estimator.

L0

Address:	*2Bh (Motor1)* *74h (Motor2)*	*Range:*	*Unsigned input* *0 – 32767*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the apparent inductance of the motor and it is used by the flux estimator (Figure 51) for calculating motor flux. It is proportional to:

(Ld + Lq) / 2

where Ld and Lq are the d and q axis motor inductance.

The scaling constant between the actual inductance in Henry and this parameter is provided in the MCEWizard tool .

LSlncy

Address:	*2Fh (Motor1)* *78h (Motor2)*	*Range:*	*Unsigned input* *0 – 32767*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the apparent saliency inductance of the motor and it is used by the flux estimator (Figure 51) for calculating motor flux. It is proportional to:

(Lq – Ld) / 2

where Ld and Lq are the d and q axis motor inductance. Typically, $0.95 < Lq/Ld < 1.05$ for Surface PM motors and $1.2 < Lq/Ld < 2.5$ for Interior Permanent magnet motors.

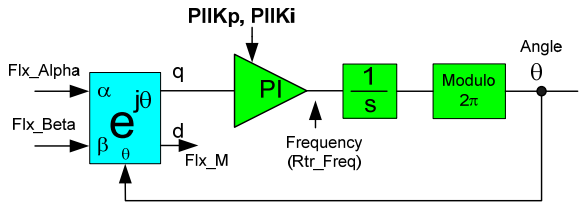
The scaling between the actual inductance in Henry and this parameter is provided in the MCEWizard tool.

PllKi

Address:	*31h (Motor1)* *7Ah (Motor2)*	*Range:*	*Unsigned input* *0 – 8191*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the Angle Frequency Generator (Figure 51) tracking integral gain. The Angle Frequency Generator is mainly a phase lock loop (PLL) device. Diagrams below shows a simplified and a detailed PLL architecture (Figure 79). As can be seen in this diagram, PLLKi relates internal PLL tracking error (q) to frequency (Rtr_Freq). A larger value of PllKi will increase tracking bandwidth at the expense of increasing speed or frequency ripple.



Simplified block diagram of a Flux PLL

MCEWizard calculates this gain based on the selected PLL bandwidth.

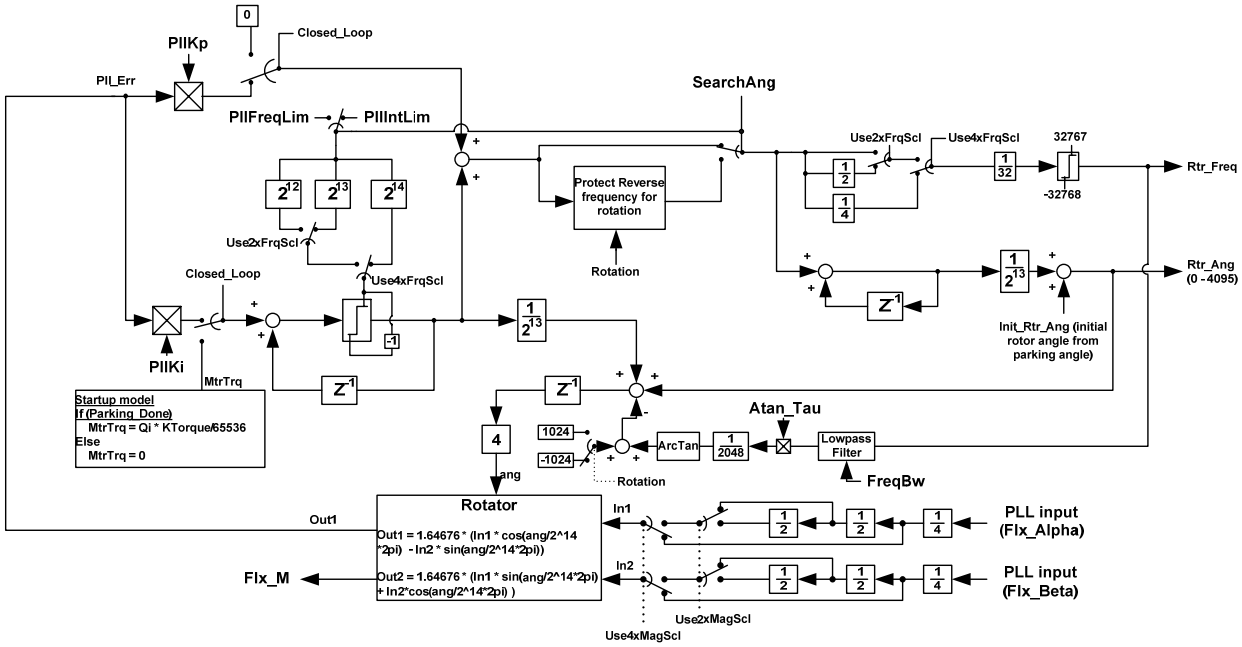


Figure 79. Detail scaling of PLL

PllKp

Address:	*30h (Motor1)* *79h (Motor2)*	*Range:*	*Unsigned input* *0 – 8191*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the Angle Frequency Generator (Figure 51) tracking proportional gain. The Angle Frequency Generator is mainly a phase lock loop (PLL) device. A larger value of PllKp will increase tracking bandwidth at the expense of increasing speed or frequency ripple. (See Figure 79)

The MCEWizard calculates this gain based on the selected PLL bandwidth.

Rs

Address:	*2Ah (Motor1)* *73h (Motor2)*	*Range:*	*Unsigned input* *0 – 32767*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the motor per phase equivalent (motor + cable) resistance at 25 Deg C. The scaling between the actual motor resistance and this parameter depends on drive

voltage and current scaling. The relationship between the actual resistance in ohms and Rs is formulated in the MCEWizard.

VoltScl

Address:	*2Dh (Motor1)* *76h (Motor2)*	*Range:*	*Unsigned input* *0 – 32767*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the gain scaler for translating voltage to internal voltage scaling of the flux estimator. The MCEWizard calculates this parameter for appropriate internal drive voltage scaling.

4.4.8 Open Loop Angle Estimator Write Register Group

KTorque

Address:	*38h (Motor1)* *81h (Motor2)*	*Range:*	*Unsigned input* *0 – 65535*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the motor mechanical model gain used in Open-loop startup mode. KTorque relates motor developed torque to drive acceleration. This gain plays an important role in robust startup. The acceleration scaling is given by:

$$\text{Drive acceleration} = \frac{KTorque \times IMotor \times FreqPwm^2}{RatedMotorAmps \times 2^{29}} \quad \text{[Hz/sec]}$$

where $FreqPwm$ is the inverter switching frequency in Hz.
$IMotor$ is the motor current in Amps

For instance: At rated motor Amps (IMotor = RatedMotorAmps) and 10KHz inverter pwm frequency, setting KTorque = 100 will yield a 18.63 Hz/sec acceleration rate. At 50% rated motor Amps, the acceleration will reduce by 50%.

VFGain

Address:	*39h (Motor1)* *82h (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the Volts per Hertz gain scaler for the purpose of the open-loop diagnostic. In this diagnostic mode, the command target frequency (VFFreq; see Section 4.4.6) is multiplied by VFGain to generate a modulation index, thereby maintaining a constant Volts/Hz ratio under constant dc bus operation. This mode of operation generates prescribed inverter voltages without requiring current feedbacks. Therefore, it can be used to drive passive load or Induction motor load for troubleshooting hardware (feedback and PWM) related issues.

The scaling relationship is given by:

$$\text{Modulation Index} = \text{VFGain} \times \text{VFFreq} / 2^8 \quad \text{[digital count of modulation]}$$

where 1430 represents 100% Modulation Index (VLine = $Vdc/\sqrt{2}$) and VFFreq is the frequency command (the value of register VFFreq, as described in Section 4.4.6).

4.4.9 Startup Angle Estimator Write Register Group

MtrCtrlBits

Address:	*0Ah (Motor1)* *53h (Motor2)*	*Range:*	*Unsigned input* *with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the configuration of various motor control functions.

Bits 0 – 3 **DiagSelect**
- 0000 No diagnostic enabled
- 0001 Enable parking diagnostic
- 0010 Enable start-up diagnostic
- 0101 Enable current regulator diagnostic
- 1001 Enable Volts/Hz diagnostic

Bit 4 **PhsLosDisable**
- 0 Enable phase loss detection
- 1 Disable phase loss detection

Bit 5 **Use2xFrqScl (see** Figure 79**)**
- 0 Do not use 2x frequency scale
- 1 Use 2x frequency scale

Bit 6 **ZeroSpdDisable**
- 0 Enabled zero speed fault detection
- 1 Disable zero speed fault detection

Bits 7 – 15 Unused; set to zero

MtrCtrlBits_S

Address:	*0Bh (Motor1)* *54h (Motor2)*	*Range:*	*Unsigned input* *with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the configuration of various motor control functions. This parameter is a bit-packed parameter which serves as software jumpers for turning on and off certain motor control functions. Certain bits (for instance: Bit 2 and 3) in this parameter provide internal scaling adjustment such that data scaling can be managed within the specified range (16-bit signed). For instance: Bit 2 and 3 (flux attenuation jumpers for PLL) are used to scale the inputs (Flx_Alpha and Flx_Beta of Figure 79) of the PLL such that the PLL gains (PllKp and PllKi of Figure 79) can always stay within range for different operating conditions (frequency range). These 2 bits are configured by MCE wizard. With these 2 bits, the inputs (Flx_Alpha and Flx_Beta) can be attenuated by 4, 8 and 16 times as shown in Figure 79.

Bit 0 **CatchEnb**
- 0 Disable catch spin function
- 1 Enable catch spin function

Bit 1 **Use4xFrqScl (see** Figure 79**)**
- 0 Do not use 4x frequency scale
- 1 Use 4x frequency scale

Bit 2 **Use2xMagScl (see** Figure 79**) (Note:** this bit only applies when bit 3 = 0)
- 0 Attenuate 16x flux amplitude.
- 1 Attenuate 8x flux amplitude.

Bit 3 **Use4xMagScl (see** Figure 79**)**
- 0 Attenuate 8x or 16x (depends on Bit 2) flux amplitude for PLL inputs
- 1 Attenuate 4x flux amplitude for PLL inputs

Bit 4 **IregCompEnb**
- 0 Disable dc bus compensation for current regulators
- 1 Enable dc bus compensation for current regulators

Bit 5	**UseExtFlux**	
	0	Use internal fluxes for PLL
	1	Use external fluxes fluxes (Ext_Flx_Alpha, Ext_Flx_Beta, Section 4.4.18) for PLL
Bits 6 - 15	Unused; set to zero	

ParkAng1, ParkAng

Address:	*ParkAng1:* *21h (Motor1)* *6Ah (Motor2)* *ParkAng:* *20h (Motor1)* *69h (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Angle = ParkAng / 64 × 90 [Deg]

Description: These parameters specify the angle to be used in the parking stage of startup. During parking, two parking angles are used. This angle is between motor U-phase and the applied current vector. The drive will first use ParkAng1 for a short duration (25% of total parking duration); thereafter, the parking angle will switch to ParkAng to complete the parking duration.

ParkI

Address:	*1Dh (Motor1)* *66h (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: see description

Description: This parameter specifies the amount of dc current injection during startup parking stage.
The relationship of ParkI and actual W-phase motor current (Iw) during parking is given by:

$$Iw = ParkI \times 0.003399 \times \sqrt{2} \times \text{rated motor Amp} \times \cos(Angle - 60°) \quad \text{[peak Amps]}$$

ParkTm

Address:	*1Eh (Motor1)* *67h (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Parking Time duration = ParkTm × 0.01568627 [sec.]

Description: This parameter specifies the total parking duration.

The maximum parking duration that can be set directly using this register is four seconds. 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 bit field DiagSelect of register MtrCtrlBits (see Section 4.4.9). The Parking Diagnostic overrides control of parking duration using the ParkTm register.

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.

WeThr

Address:	*1Fh (Motor1)* *68h (Motor2)*	*Range:*	*Unsigned input* *0 – 32767*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the transition level (frequency) from Open-loop to Closed-loop mode operation. The scaling between We_Thr and actual switch over frequency in Hz is given by:

Switch over frequency = We_Thr × FreqScl × FreqPwm / 2^{20} [Hz]
where:
FreqPwm is the inverter pwm frequency in Hz; and
FreqScl is the frequency scaler, determined as:

```
if ( Use4xFreqScl = 1 ) FreqScl = 4
else if ( Use2xFreqScl = 1 ) FreqScl = 2
else FreqScl = 1
```

(Use4xFreqScl and Use2xFreqScl are bit fields of the MtrCtrlBits_S and MtrCtrlBits registers, respectively. See Section 4.4.9.). These registers are calculated by MCEWizard.

4.4.10 Startup Retrial Write Register Group

FlxThrH

Address:	*26h (Motor1)* *6Fh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Upper Flux threshold = $\frac{FlxThrH \times 128}{RatedFlxCounts \times 1.647 \times M}$ x 100 [% rated flux]

where: RatedFlxCounts = 5000 (default by MCEWizard)
if (Use4xMagScl = 1) M = 4
Elseif (Use2xMagScl = 1) M = 2
Else M = 1

Description: In the Sensorless FOC block, a start fail detection signal (Statusflags bit 6) is provided for startup failure detection. This signal can be used by a master motor control sequencer to carry appropriate actions (for instance: startup retry) upon drive startup failure. A successful startup detection is determined by comparing two flux threshold levels (FlxThrH and FlxThrL) and the calculated motor flux (Flx_M). This parameter specifies the upper flux threshold level for determining a successful drive startup.

FlxThrL

Address:	*25h (Motor1)* *6Eh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Lower Flux thershold = $\frac{FlxThrL \times 64}{RatedFlxCounts \times 1.647 \times M}$ x 100 [% rated flux]

where: RatedFlxCounts = 5000 (default by MCEWizard)
if (Use4xMagScl = 1) M = 4
Elseif (Use2xMagScl = 1) M = 2
Else M = 1

Description: In the Sensorless FOC block, a start fail detection signal (Statusflags bit 6) is provided for startup failure detection. This signal can be used by a master motor control sequencer to carry appropriate actions (for instance: startup retry) upon drive startup failure. A successful startup detection is determined by comparing two flux threshold levels (FlxThrH and FlxThrL) and the calculated motor flux (Flx_M). This parameter specifies the lower flux threshold level for determining a successful drive startup.

NumRetries

Address:	*27h (Motor1)* *70h (Motor2)*	*Range:*	*Unsigned input* *0 – 15*	*Reset value:*	*0*

Scaling or Notation: Number of retries = NumRetries

Description: This parameter is intented for specifying the allowable number of startup retries. In the Sensorless FOC block, parameter NumRetries only affacets the enabling of disabling of the start failure detection function. If NumRetries = 0, startup fail detection is disabled, otherwise it is enabled, allowing the use of the StartFail bit of the StatusFlags register. Inside the Sensorless FOC block, only start fail detection logic is implemented. A master control sequencer should be in place to implement the actual startup retry sequence (start retry and generate retry fault) according to the user's requirement.

ParkIRet

Address:	*22h (Motor1)* *6Bh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: see description

Description: This parameter specifies the amount of dc current injection during Retry startup parking stage. The Retry parking stage is identified by TwoRetries bit (Sensorless FOC block input). When TwoRetries = 1, Parking current employes ParkIRet, otherwise ParkI will be used. A master control sequencer should be in place to implement the actual startup retry sequence (start retry and generate retry fault) according to the user's requirement. This master control sequencer should manipulate TwoRetries bit to signal the Sensorless FOC block to use different parking configuration (ParkIRet, ParkTmRet) during a retry startup.

The relationship of ParkIRet and W-phase motor current (Iw) during parking is given by:

$$Iw = ParkIRet \times 0.003399 \times \sqrt{2} \times \text{rated motor Amp} \quad \text{[peak Amps]}$$

ParkTmRet

Address:	*23h (Motor1)* *6Ch (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Parking time duration = ParkTmRet × 0.01568627 [sec.]

Description: This parameter specifies the total parking duration in the startup Retry mode. Startup Retry mode is identified by TwoRetries bit (Sensorless FOC block input). When TwoRetries = 1, Parking time employes ParkTmRet, otherwise ParkTm will be used. A master control sequencer should be in place to implement the actual startup retry sequence (start retry and generate retry fault) according to the user's requirement. This master control sequencer should manipulate TwoRetries bit to signal the Sensorless FOC block to use different parking configuration (ParkIRet, ParkTmRet) during a retry startup. Users may want to increase parking current and parking duration during a retry startup.

The maximum parking duration is four seconds but can be extended using the Parking Diagnostic. Please refer to register ParkTm in Section 4.4.9 for a description of the procedure.

RetryTm

Address:	*24h (Motor1)* *6Dh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Startup flux sampling instance = RetryTm × 1.966 [msec.]

Description: In the Sensorless FOC block, a successful start detection is determined by comparing two flux band levels (FlxThrH and FlxThrL) and the calculated motor flux (Flx_M). The comparison is done at a certain time duration (RetryTm) after open-loop startup is accomplished. This parameter specifies the sampling instant of motor flux (Flx_M) for determination of a successful startup. RetryTm is measured from the time Closed-loop is activated. This sampling delay is required to ensure a valid flux establishment in the drive.

4.4.11 Phase Loss Detect Write Register Group

AdjPark1, AdjPark2

Address:	*AdjPark1:* *43h (Motor1)* *8Ch (Motor2)* *AdjPark2:* *44h (Motor1)* *8Dh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: These two parameters specify the Phase loss detection current gain scalers.
During parking stage of drive startup, W-phase motor current is compared against anticipated parking current levels (2-stage parking) to determine whether a phase loss (connection between inverter and motor) is presented. Since, current regulation is enforced during parking, the motor current will track command current under normal circumstances. If the current error is larger than a certain threshold (PhsLosThr), a phase loss condition is generated. The command current levels are computed using the parking current (ParkI, see Section 4.4.9) . The value of ParkI is multiplied by a scaler of either AdjPark1 (first stage of parking) or AdjPark2 (second stage of parking) for proper current comparison between command and actual w-phase current (see section 4.4.24 register IDiff_Fil for phase loss detection diagram)

In MCEWizard, these current gain scalers are calculated such that proper scaling is applied to parking current (ParkI) for comparison to the scaled feedback current Iw (see section 4.4.24 register IDiff_Fil for more details) .

Phase loss detection can be disabled through bit PhsLosDisable in register MtrCtrlBits (see Section 4.4.9).

PhsLosThr

Address:	*45h (Motor1)* *8Eh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: Phase loss Current thershold = PhsLosThr / Iv_Iw_Scl [Amps peak]
where Iv_Iw_Scl is the Current Feedback Scaling (counts/Amp peak as given in the "Verify & Save Page" of the MCEWizard).

Description: This parameter configures the phase loss detection current error threshold level. During parking stage of drive startup, w-phase motor current is compared against anticipated current levels to determine whether a phase loss (connection between inverter and motor) is presented. If the absolute current error is larger than the threshold determined by PhsLosThr, a phase loss fault is generated. (see section 4.4.24 IDiff_Fil for diagram)

4.4.12 Single Shunt Write Register Group

ScsSamples

Address:	*F4h (Motor1)* *FBh (Motor2)*	*Range:*	*Unsigned input* *with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the setup for adaptive current feedback sampling. In the region of minimum pulse constraint (registers TCntMin3Phs or TCntMin2Phs), the frequency of feedback sampling can be reduced (undersampling) in order to minimize the occurrence of minimum pulse clamping. This is done to suppress audible noise for noise-sensitive applications.

Bits 0 – 3 **FbkSampleRate**
This value provides setup for feedback sampling rate in the region of minimum pulse constraint, as follows:

Feedback sample rate = FbkSampleRate + 1 [Pwm cycles/feedback]

Bits 4 – 7 **GainAtten**
This value provides setup for current regulator gain attenuation, as follows:

Gain attenuation = 1/ (2 ^ GainAtten)

Bits 8 – 15 Unused; set to 0.

For instance, a value of 35 (00100011b) in ScsSamples implies 4 Pwm cycles/feedback and 4 times current regulator gain attenuation.

In practice, the gain attenuation should be set equal to the number of Pwm cycles/feedback. It is preferred to use a minimum possible value of Pwm cycles/feedback to satisfy an application in terms of audible noise. The default setting of ScsSamples is 0 (1 PWM cycle/feedback).

SHDelay

Address:	*4Bh (Motor1)* *94h (Motor2)*	*Range:*	*Unsigned input* *1 – 1023*	*Reset value:*	*120*

Scaling or Notation: Delay time = SHDelay / SysClk [usec.]
where SysClk is the system clock frequency in MHz.

Description: This parameter specifies the hardware PWM gate propagation delay. It is measured from IC gating output to the actual turn-on of the power-switching device. It is used by the SINGLE_I_SHUNT module to schedule current sampling instants. In practice, the total PWM gate propagation delay is dominated by gate driver IC for the 300 series reference design platforms.

TCntMin2Phs

Address:	*49h (Motor1)* *92h (Motor2)*	*Range:*	*Unsigned input* *3 – 1023*	*Reset value:*	*180*

Scaling or Notation: Minimum pulse width = TCntMin2Phs / SysClk [usec.]
where SysClk is the system clock frequency in MHz.

Description: This parameter specifies the minimum PWM pulse width to be used when 2-phase modulation mode is allowed (bit field TwoPhsEnb in register TwoPhsCtrl; see Section 4.4.2).

Figure 57 (c) and (d) illustrate minimum pulse clamping of a 2-phase modulation scheme.

TCntMin3Phs

Address:	*4Ah (Motor1)* *93h (Motor2)*	*Range:*	*Unsigned input* *3 – 1023*	*Reset value:*	*359*

Scaling or Notation: Minimum pulse width = TCntMin3Phs / SysClk [usec.]
where SysClk is the system clock frequency in MHz.

Description: This parameter specifies the minimum PWM pulse width to be used in 3-phase modulation mode. Figure 57 (a) and (b) illustrate minimum pulse clamping of a 3-phase modulation scheme.

4.4.13 Start/Stop Sequencing Write Register Group

RetryOccurred

Address:	*47h (Motor1)* *90h (Motor2)*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 = no retry; 1 = startup retry occurred

Description: The motor control startup sequencing is implemented outside the RTL in order to provide sequencing flexibility. This status line (RetryOccured) signals the RTL that a startup retry has occurred. Upon receiving this signal, the Sensorless FOC module will prohibit phase loss detection. This is done to avoid nusance phase loss trip during retry where large parking current (ParkIRet) may cause excessive motor shaft movement during parking.

TwoRetries

Address:	*46h (Motor1)* *8Fh (Motor2)*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 = two retries have not yet occurred; 1 = two retries have occurred

Description: This status bit signals that two startup retries have occurred, primarily for use in the startup retry function. Retry parking stage is identified by TwoRetries bit (Sensorless FOC block input). When TwoRetries = 1, this signals the Sensorless FOC module to employe a different set of parking configuration (ParkIRet, ParkTmRet), otherwise the normal configuration (ParkI, Parktm) will be used.

IfbOffsetCalc

Address:	*48h (Motor1)* *91h (Motor2)*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 = deactivate dc current adjustment; 1 = activate dc current adjustment

Description: Current feedback dc offset control line.

During dc current adjustment (IfbOffsetCalc = 1), the motor feedback current will be averaged. The average value is stored in register IfbOffset (see Section 4.4.25). This control line should be deactivated as soon as the inverter run command is issued. When IfbOffsetcalc = 1, 4096 samples of current feedback will be acquired and averaged. Each sample corresponds to one PWM cycle time duration, therefore for lower inverter switching frequency (longer PWM cycle time), the time require for current averaging will be longer. For instance, (4096 x 0.1) msec will be required to accomplish current averaging for 10 KHz inverter switching frequency. It is crucial to allow sufficient timing to finish current averaging before running motor. If IfbOffsetcalc goes low before 4096 PWM cycles, the averaging will be aborted. Motor Control sequencer should handle the sequencing of this control line.

4.4.14 Catch Spin Write Register Group

PllFreqLim

Address:	*36h (Motor1)* *7Fh (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the frequency limit of the PLL integral gain output. The relationship between the actual frequency in Hz and this parameter is given by:

Frequency limit = PllFreqLim x FreqPwm x FreqScl / 2^13

where:
A is the actual frequency in Hz;
FreqPwm is the inverter pwm frequency in Hz; and
FreqScl is the frequency scaler, determined as:
if (Use4xFreqScl = 1) FreqScl = 4
else if (Use2xFreqScl = 1) FreqScl = 2
else FreqScl = 1
(Use4xFreqScl and Use2xFreqScl are bit fields of the MtrCtrlBits_S and MtrCtrlBits registers, respectively. See Section 4.4.9. and Figure 79)

PllIntLim

Address:	*3Ah (Motor1)* *83h (Motor2)*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the frequency limit of the PLL integral gain output during Catch-Spin mode. The relationship between the actual frequency in Hz and this parameter is given by:

Frequency limit = PllIntLim x FreqPwm x FreqScl /2^13 [Hz]

where:
FreqPwm is the inverter pwm frequency in Hz; and
FreqScl is the frequency scaler, determined as:
if (Use4xFreqScl = 1) FreqScl = 4
else if (Use2xFreqScl = 1) FreqScl = 2
else FreqScl = 1
(Use4xFreqScl and Use2xFreqScl are bit fields of the MtrCtrlBits_S and MtrCtrlBits registers, respectively. See Section 4.4.9 and Figure 79.)

Search_Ang

Address:	*3Bh (Motor1)* *84h (Motor2)*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

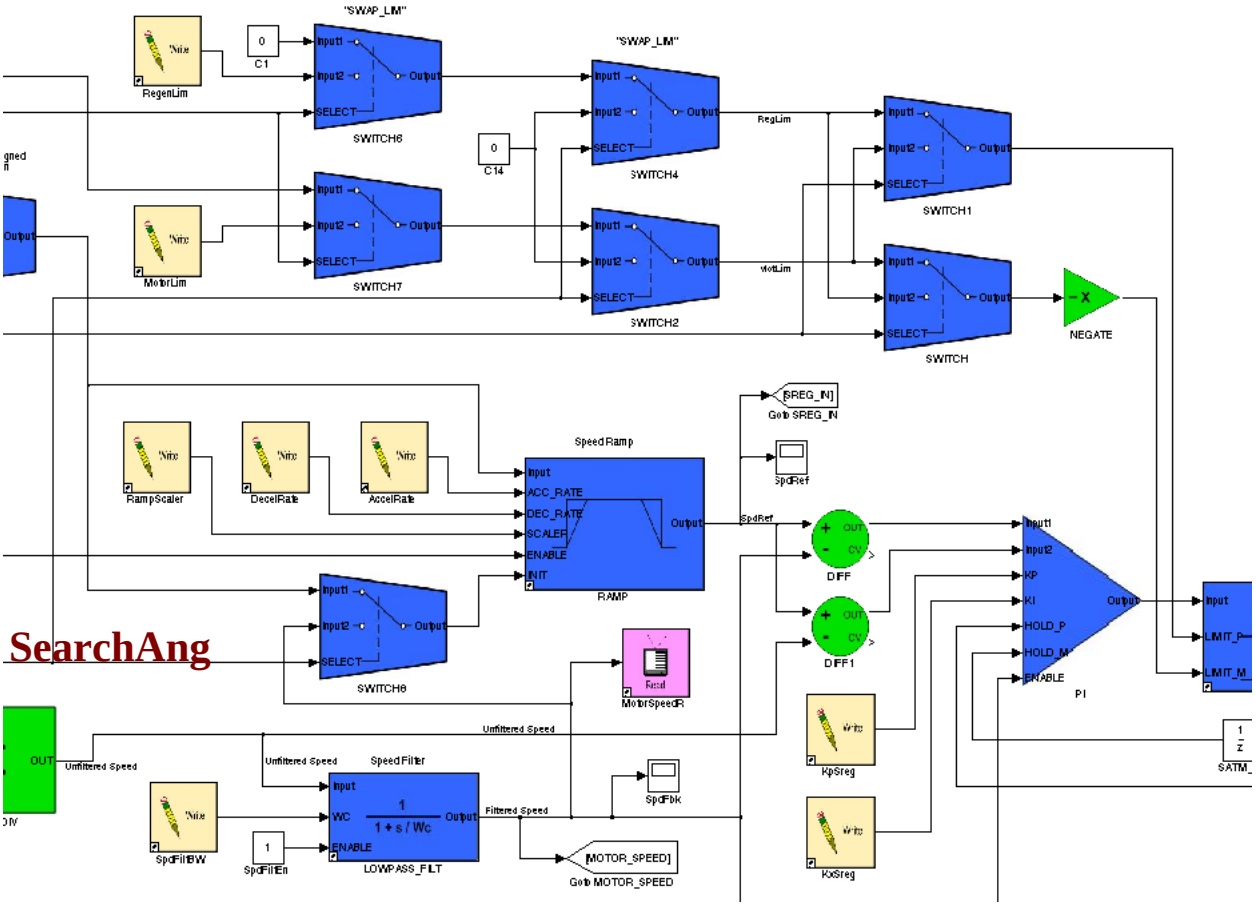
Scaling or Notation: 0 = Skip searching for rotor angle;
1 = Search for rotor angle before applying torque

Description: This control bit commands the Sensorless FOC to search for the rotor angle before enabling torque. This is done to ensure smooth start into a spinning motor. In practice, the motor control sequencer will issue this control bit at the very beginning of the run command if the Catch-Spin function is enabled (CatchEnb bit in register MtrCtrlBits_S; see Section 4.4.9).
SearchAng bit is set and reset by a motor control sequencer (typically locate in 8051 code) to allow implementation of Catch-Spin function. It affects MCE application layer of the reference design and RTL (Sensorless FOC block) of the 300 series. The following describes how SearchAng bit affects MCE application layer and the Sensorless FOC block.

MCE Application layer

- The motor and regen limit are set to zero. These force a zero current command on Iq and Id. This is the first stage of Catch-Spin, attempting to force zero motor current. When zero motor current is forced, the motor inverter output voltage will match up with motor BEMF to achieve zero motor current and synchronization between inverter voltage and motor BEMF.

- The Speed Ramp is preconditioned to SpdFbk (see diagram below). This is done to allow smooth transfer when speed regulator is released for the forward catch case.



SearchAng bit in MCE application layer (reference design only)

SearhAng bit manipulation inside Sensorless FOC module (RTL) (see also Figure 79**)**

1) Parking_Done and Closed_Loop bits are affected by the SearchAng bit.

Normally when SearchAng = 0 and drive start command issued (FOCEnable = 1), Parking_Done is set when parking time (ParkTm) expires. And Closed-loop bit are set when estimated frequency from PLL > WeThr (switchover frequency Threshold). However, when SearchAng = 1, both Parking_Done and Closed-loop bit are immediately set to 1 and latched on. This is done to allow Flux PLL to track flux immediately without going through parking and open-loop start.

2) PLL frequency integral limit is affected by SearchAng

The output frequency of the PLL is normally (SearchAng = 0) limited to PllFreqLi (MCEWizard default to 105% of max speed/frequency). However, when SearchAng =1, the maximum output frequency will be limited by PllIntLim (MCEWizard defaults to 75% of rated motor frequency). This is done to allow a faster angle search if user already know the possible catch speed range under consideration. For instance, for an outdoor fan, the maximum catch speed is specify at +/- 400rpm and the fan rated speed is 1000 to 1200rpm, one can use a lower PllIntLim (say 60%) value to allow a faster angle convergence. This implementation is shown in Figure 79.

3) PLL output frequency protection is affected by SearchAng.

Normally (SearchAng = 0) if speed rotation command (rotation) is set to positive, the PLL output frequency is protected to go opposite direction. Similarly, if speed rotation command is set to negative, the PLL output frequency is protected to go opposite direction. When SearchAng = 1, this PLL frequency protection is removed. It is done to allow forward and reverse catch-spin. For instance in an outdoor fan, initially, one cannot determine whether speed is forward or reverse. Therefore, one has to allow the PLL frequency to go both negative and positive to find the correct frequency polarity. This implementation is also shown in Figure 79.

Zero_Vec_Req

Address:	*3Ch (Motor1)* *85h (Motor2)*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 = Current regulator output limits (d and q) resumes normal value;
1 = Current regulator output limits (d and q) are set to zero

Description: This control bit commands the Sensorless controller to issue zero output voltage (zero modulation). When this register value is set high, the outputs of the current regulators (d-q) are clamped to zero. When the register value is set low, the current regulator output limits resume their normal values, determined by the settings of registers VdLim and VqLim (see Section 4.4.5 and Figure 51).

4.4.15 User Control Write Register Group

UserVabEn

Address:	*4Ch (Motor1)* *95h (Motor2)*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 = Select Sensorless FOC to drive SVPWM input modulation
1 = Select User_Alpha to drive SVPWM input modulation

Description: This parameter selects the SVPWM modulation input (see Figure 51).

UserVuvwEn

Address:	*4Dh (Motor1)* *96h (Motor2)*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 = Select gating control from SVPWM;
1 = Select gating control from duty ratio modulator.

Description: PWM pattern selector. Setting UserVuvwEn to 1 selects gating pattern generation from duty ratio control. The built-in SVPWM will be bypassed and registers User_U, User_V and User_W are used instead (see Figure 51).

User_Alpha

Address:	*4Eh (Motor1)* *97h (Motor2)*	*Range:*	*Signed input* *-32768 – 32767*	*Reset value:*	*0*

Scaling or Notation: 2355 = 100% modulation

Description: This parameter provides the User Alpha (align with U phase) modulation index.

The Space Vector modulator is normally driven by a modulation index generated by the Sensorless FOC. When register UserVabEn = 1, users can bypass the Sensorless FOC and drive Alpha and Beta modulation directly to the inputs of the SVPWM (see Figure 51). 100% modulation provides inverter line-to-line theoretical rms output voltage of $Vdc/\sqrt{2}$.

User_Beta

Address:	*4Fh (Motor1)* *98h (Motor2)*	*Range:*	*Signed input* *-32768 – 32767*	*Reset value:*	*0*

Scaling or Notation: 2355 = 100% modulation

Description: This parameter provides the User Beta (Orthogonal to U phase) modulation index.

The Space Vector modulator is normally driven by a modulation index generated by the Sensorless FOC. When register UserVabEn = 1, users can bypass the Sensorless FOC and drive Alpha and Beta modulation directly to the inputs of the SVPWM (see Figure 51). 100% modulation provides inverter line-to-line theoretical rms output voltage of $Vdc/\sqrt{2}$.

User_U

Address:	*50h (Motor1)* *99h (Motor2)*	*Range:*	*Unsigned input* *0 – 65535*	*Reset value:*	*0*

Scaling or Notation: U phase duty ratio = User_U / PwmPeriodConfig [duty ratio]
where PwmPeriodConfig is the value of register PwmPeriodConfig (see Section 4.4.2).

Description: This parameter provides the User U-phase duty ratio control.

The inverter-gating pattern is normally generated by the Space Vector Modulator (SVPWM). By setting register UserVuvwEn = 1, users can bypass SVPWM and directly control u-phase gating duty ratio (symmetrically centered) via User_U (see Figure 51).

User_V

Address:	*51h (Motor1)* *9Ah (Motor2)*	*Range:*	*Unsigned input* *0 – 65535*	*Reset value:*	*0*

Scaling or Notation: V phase duty ratio = User_V / PwmPeriodConfig [duty ratio]
where PwmPeriodConfig is the value of register PwmPeriodConfig (see Section 4.4.2).

Description: This parameter provides the User V-phase duty ratio control.

The inverter-gating pattern is normally generated by the Space Vector Modulator (SVPWM). By setting register UserVuvwEn = 1, users can bypass SVPWM and directly control v-phase gating duty ratio (symmetrically centered) via User_V (see Figure 51).

User_W

Address:	*52h (Motor1)* *9Bh (Motor2)*	*Range:*	*Unsigned input* *0 – 65535*	*Reset value:*	*0*

Scaling or Notation: W phase duty ratio = User_W / PwmPeriodConfig [duty ratio]
where PwmPeriodConfig is the value of register PwmPeriodConfig (see Section 4.4.2).

Description: This parameter provides the User W-phase duty ratio control.

The inverter-gating pattern is normally generated by the Space Vector Modulator (SVPWM). By setting register UserVuvwEn = 1, users can bypass SVPWM and directly control w-phase gating duty ratio (symmetrically centered) via User_W (see Figure 51).

4.4.16 Field Weakening Control Write Register Group

IdRefExt

Address:	*37h (Motor1)* *80h (Motor2)*	*Range:*	*Signed input* *-16384 – 16383*	*Reset value:*	*0*

Scaling or Notation: 4095 = 100 [% rated motor current]

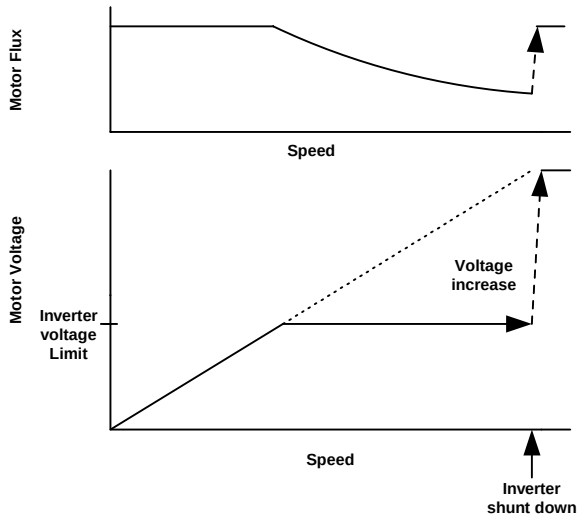
Description: This parameter specifies the command d-axis motor current in normal operation (excluding parking). The purpose of d-axis current control is to obtain optimal torque per ampere and field-weakening control.

CriticalOv

Address:	*FEh (Motor1)* *FFh (Motor2)*	*Range:*	*Boolean input* *0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 = deactivate; 1 = activate

Description: This signal activates zero vector (low side devices turn-on) PWM state independent of any condition (including faults). This is particularly useful for implementing dc bus over voltage protection in a non-regenerative drive application. The figure below illustrates a critical overvoltage condition in the Field-Weakening range.



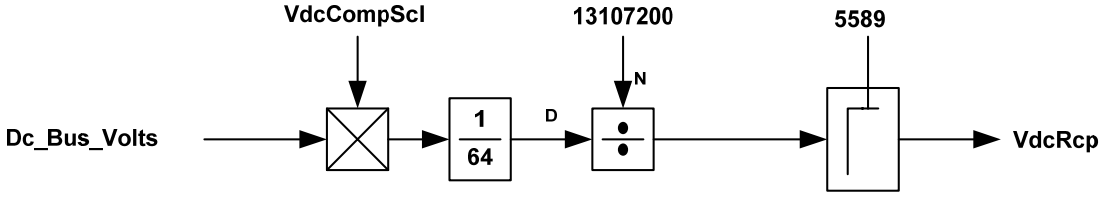
4.4.17 Protection Write Register Group

VdcRcp

Address:	*ECh*	*Range:*	*Unsigned input* 0 – 5589	*Reset value:*	*4096*

Scaling or Notation: 4096 = Nominal dc bus

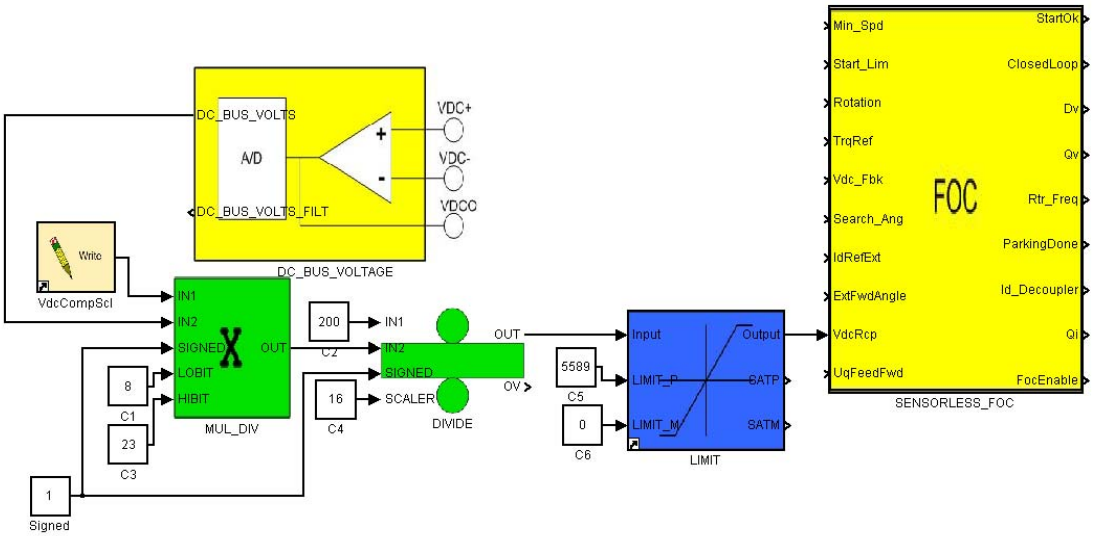
Description: This signal represents the reciprocal of dc bus voltage feedback. It can be used for current regulator dc bus compensation. If dc bus compensation is enabled (bit IregCompEnb of register MtrCtrlBits_S; see Section 4.4.9), the current regulator will use this signal to perform dc bus compensation. Figure 80 shows how VdcRcp is calculated.



VdcCompScl is a scaler
VdcCompScl = 3200 * 64 / (VdcNom * VdcScl)
where:
VdcNom - the nonimal dc bus voltage in volts.
(MCEWizard: "Nominal DC Bus Voltage")
VdcScl - dc bus scaling in digital counts per volt.
(MCEWizard: "DC Bus Feedback Scaling")

Figure 80. Calculation of VdcRcp

Note that the VdcRcp calculation is not implemented in the factory installed MCE program, so dc bus compensation will not work. The diagram below shows an implementation of dc bus compensation in Simulink (with all the other connections and blocks stripped away).



One final consideration for robust and effective DC Bus compensation is that the Vdc_Fbk input to the FOC must be the same signal as the one used to create the input to VdcRcp. In this example, DC_BUS_VOLTS should be connected to Vdc_Fbk.

Vdc_Fbk

Address:	*E9h (Motor1)* *EDh (Motor2)*	*Range:*	*Unsigned input* *0 – 4095*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: Filtered (0.25 to 0.5 msec time constant) Dc bus voltage feedback. This is the filtered version of the raw dc bus voltage feedback. It is used by the Sensorless motor controller for motor phase voltage reconstruction (modulation index and dc bus voltage). Normally, this should be driven by DcBusVoltsFilt of the dc bus feedback module output of the MCE.

Scaling: Dc bus feedback = Vdc_Fbk / VdcScl [Volts]

where VdcScl is the dc bus scaling in digital counts per volt. VdcScl relates the actual voltage to the raw A/D counts. VdcScl ("DC Bus feedback Scaling") is entered in the MCEWizard.

DcBusOvLevel

Address:	*08h*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*210*

Scaling or Notation: Dc bus over voltage trip level = DcBusOvLevel × 16 / VdcScl [volts]

where VdcScl is the dc bus scale (entered as "dc bus Scale" in the MCEWizard) in counts per volt.

Description: This parameter defines the dc bus over voltage trip level. A dc bus over voltage fault will be generated if dc bus voltage exceeds this threshold. Refer to the description of the FaultFlags register in Section 4.4.20 for more information.

DcBusLvLevel

Address:	*09h*	*Range:*	*Unsigned input* *0 – 255*	*Reset value:*	*61*

Scaling or Notation: Dc bus under voltage trip level = DcBusLvLevel ×16 / VdcScl [volts]

where VdcScl is the dc bus scale (entered as "dc bus Scale" in the MCEWizard) in counts per volt.

Description: This parameter defines the dc bus under voltage trip level. A dc bus under trip voltage fault will be generated if dc bus voltage falls below this threshold. Refer to the description of the FaultFlags register in Section 4.4.20 for more information.

4.4.18 External Signals Write Register Group

ExtFwdAngle

Address:	*F1h (Motor1)* *F8h (Motor2)*	*Range:*	*Unsigned input* *0 – 4095*	*Reset value:*	*0*

Scaling or Notation: 1024 = 90 [Degs]

Description: This parameter provides an angle sum into the forward rotating angle (can be set to zero by gain adjustment) of the current regulator. Users have the flexibility of providing their own angle calculation for improved controller performance. The ability to employ an external angle also provides the flexibility of implementing Field-Oriented Control for various types of rotating field motors.

ExtRevAngle

Address:	*F0h (Motor1)* *F7h (Motor2)*	*Range:*	*Unsigned input* *0 – 4095*	*Reset value:*	*0*

Scaling or Notation: 1024 = 90 [Degs]

Description: This parameter provides an angle sum into the reverse rotating angle (can be set to zero by gain adjustment) of the current regulator. Users have the flexibility of providing their own angle calculation for improved controller performance. The ability to employ an external angle also provides the flexibility of implementing Field-Oriented Control for various types of rotating field motors.

Ext_Flx_Alpha, Ext_Flx_Beta

Address:	*Ext_Flx_Alpha:* *F2h (Motor1)* *F9h (Motor2)* *Ext_Flx_Beta:* *F3h (Motor1)* *FAh (Motor2)*	*Range:*	*Signed input* *-32768 – 32767*	*Reset value:*	*0*

Scaling or Notation: 5000 = rated motor flux [Volt-sec]

Description: The Sensorless FOC block calculates estimated fluxes (Alpha and Beta pair) internally. These internally generated fluxes can be replaced by external fluxes (Ext_Flx_Alpha and Ext_Flx_Beta). This allows users to substitute appropriate fluxes for implementing Field-Oriented Control of various types of rotating field motors. Setting UseExtFlux = 1 selects these external fluxes. (UseExtFlux is a bit field of register MtrCtrlBits_S; see Section 4.4.9.)

UdFeedFwd

Address:	*EFh (Motor1)* *F6h (Motor2)*	*Range:*	*Signed input* *-2048 – 2047*	*Reset value:*	*0*

Scaling or Notation: 1430 = 100 [% modulation]

Description: This parameter provides d-axis modulation feedforward. This signal sums into the output of the d-axis current regulator.

100% modulation corresponds to the maximum achievable value of the SVPWM linear range. The corresponding theoretical rms motor line voltage at 100% modulation is $Vdc/\sqrt{2}$.

UqFeedFwd

Address:	*EEh (Motor1)* *F5h (Motor2)*	*Range:*	*Signed input* *-2048 – 2047*	*Reset value:*	*0*

Scaling or Notation: 1430 = 100 [% modulation]

Description: This parameter provides q-axis modulation feedforward. This signal sums into the output of the q-axis current regulator.

100% modulation corresponds to the maximum achievable value of the SVPWM linear range. The corresponding theoretical rms motor line voltage at 100% modulation is Vdc / sqrt (2).

4.4.19 PFC Control Write Register Group

PFC_Ctrl

Address:	*9Ch*	*Range:*	*Unsigned input with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter controls the operation of the PFC.

Bit 0	Unused; set to 0
Bit 1	**PFCGateSense** 0 = low true gating convention 1 = high true gating convention
Bit 2	**PFCGKReset** 0 = allow the PFCGateKill fault to be latched 1 = clear the latched PFCGateKill fault
Bit 3	Reserved; **must be set to 1**
Bits 4 – 15	Unused; set to 0

When a PFC over current GateKill fault occurs, the PFC PWM output is disabled immediately, regardless of the setting of the PFCGKReset bit. When PFCGKReset is set to 0, it allows the PFCGateKill fault to be latched for further diagnosis and processing, even if the PFC current itself has reduced to below the threshold level. When PFCGKReset is set to 1, the latched PFCGateKill fault is cleared.

GkillFiltCnt_PFC

Address:	*9Dh*	*Range:*	*Unsigned input 0 – 255*	*Reset value:*	*59*

Scaling or Notation: See description.

Description: This parameter specifies the PFC GateKill Filter delay, given by:

GkillFiltCnt_PFC (in digital counts) = (SysClk * DelayTime) – 1

where SysClk is the system clock frequency in MHz and DelayTime is in μsec.

PFCPwmPeriod

Address:	*9Eh*	*Range:*	*Unsigned input 0 – 65535*	*Reset value:*	*416*

Scaling or Notation: See description.

Description: This parameter specifies the PFC PWM carrier frequency. The register value can be calculated by:

PFCPwmPeriod = 1000 * SysClk / (2 * PFC PWM Carrier Frequency) – 1

where SysClk is the system clock frequency in MHz and the PFC PWM carrier frequency is in kHz.

Iref

Address:	*9Fh*	*Range:*	*Unsigned input 0 – 2047*	*Reset value:*	*0*

Scaling or Notation: 2047 = Maximum IPFC [Amps]

Description: This parameter provides the reference command of PFC current loop. This signal is normally obtained from the PFC multiplier that produces the reference command to the current loop. It is the responsibility of the hardware designer to ensure that maximum IPFC corresponds to an Iref value of 2047.

PFC_OffsetDC

Address:	*A0h*	*Range:*	*Unsigned input 0 – 255*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter represents the manual offset on DC bus voltage Vdc. It is used to instantly compare Vdc and AC input voltage Vin in order to generate the ShutDown signal. The value is given by:

PFC_OffsetDC = Vdc_Compare – Vdc_ADC

where Vdc_ADC is the ADC feedback result of Vdc (in the range 0 – 4095), and Vdc_Compare is the actual signal used for the comparison.

PFC_OffsetVin

Address:	*A1h*	*Range:*	*Unsigned input 0 – 255*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter represents the Manual offset on AC input voltage Vin. It is used to instantly compare Vdc and Vin in order to generate the ShutDown signal. The value is given by:

PFC_OffsetVin = Vin_Compare – 2 * Vin_ADC

where Vin_ADC is the ADC feedback result of AC input voltage Vin (an absolute value, or rectified half-wave input voltage, in the range 0 – 2047), and Vin_Compare is the actual signal used for the comparison.

Blanking_Gap

Address:	*A2h*	*Range:*	*Unsigned input 0 – 255*	*Reset value:*	*0*

Scaling or Notation: 1 count = (1 / 4095) * Maximum Vdc [Volt]

Description: This parameter defines the hysteresis gap for instantly comparing Vdc and Vin. It is the responsibility of the hardware designer to ensure that Maximum Vdc corresponds to a count of 4095. Typically, Vdc = 492V matches a count of 4095, and Blanking_Gap is set to be around 50 counts. See Section 4.3.8.2 for detailed information on how this register is used.

CountOneSet

Address:	*A3h*	*Range:*	*Unsigned input 0 – 255*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: In the PFCPWM Blanking function of the PFC_PWM block, once the Shutdown signal is asserted to High, the PWM output is disabled immediately. This parameter provides a minimum limit of the time width during which the Shutdown signal will stay at High. The actual time width (in msec) is given by:

TimeWidth = CountOneSet / PFC_PWM Carrier Frequency

where the PFC_PWM Carrier Frequency is in kHz.

CountTwoSet

Address:	*A4h*	*Range:*	*Unsigned input 0 – 255*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter sets up a filter delay for the PFC current feedback signal, which is used in the PFCPWM Blanking function of the PFC_PWM block. Recommend values for CountTwoSet are between 3 and 5. See Section 4.3.8.2 for detailed information on how this register is used.The actual delay time (in msec) is given by:

DelayTime = CountTwoSet / PFC_PWM Carrier Frequency

where the PFC_PWM Carrier Frequency is in kHz.

VcPFC

Address:	*A6h*	*Range:*	*Unsigned input 0 – 65535*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This parameter specifies the PFC PWM duty cycle command. The value represents the PFC PWM duty cycle (ratio of switch ON time over switching period) with respect to the value of register PFCPwmPeriod, as follows:

% Duty Cycle = (VcPFC / PFCPwmPeriod) * 100.

This value is normally generated from the PFC control loop, which is implemented in the MCE microprocessor.

PFCEnable

Address:	*A7h*	*Range:*	*Boolean input 0 or 1*	*Reset value:*	*0*

Scaling or Notation: 0 = Disable PFC PWM output and enable offset correction of PFC current feedback;
1 = Enable PFC PWM output and disable offset correction of PFC current feedback.

Description: This parameter, in combination with the ShutDown signal, enables or disabled the PFC PWM output. (The ShutDown signal is generated internally to the PFC Blanking module of the PFC_PWM block. See register ShutDown, Section 4.4.26.) When PFCEnable is set to 1, PFC PWM output is enabled if ShutDown is 0. When PFCEnable is set to 0, PFC PWM output is disabled regardless of the value of ShutDown.

The PFCEnable register also enables and disables the offset correction of PFC current feedback, as follows: When PFCEnable is set to 1, offset correction is disabled; when PFCEnable is set to 0, offset correction is enabled. Note that offset correction of PFC current feedback should only be conducted under no load conditions.

GKSense_PFC

Address:	*A8h*	*Range:*	*Boolean input 0 or 1*	*Reset value:*	*0*

Scaling or Notation: See description.
Description: This parameter sets the logic sense of the PFC gate kill fault, as follows:

0 = If the IC's PGateKill pin becomes 0, the GateKill fault occurs and the PFC PWM is disabled instantly. (This setting is used for most designs.)
1 = If the IC's PGateKill pin becomes 1, the GateKill fault occurs, and the PFC PWM is disabled instantly.

PFC_sync_divider

Address:	*A9h*	*Range:*	*Unsigned input with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.
Description: This parameter configures PFC synchronization and phasing.

Bits 0 – 3 **PFCSyncRatio**
This parameter provides a ratio between the PFC PWM carrier frequency and the PFC A/D sampling frequency (which is also the PFC control loop execution rate), as follows:

PFC_A/D Sampling Freq. = PFC_PWM Carrier Freq. / (PFCSyncRatio + 1)

PFCSyncRatio can have values in the range 0 – 15.

Bits 4 – 7 **PFCPhasing**
This parameter determines the phase of the PFC A/D sampling period relative to the master PWM sync pulse. (The master is configured using register PwmMasterSel, Section 4.4.1.) The value of PFCPhasing specifies the number of cycles PFC A/D sampling is delayed following the master motor control sync pulse. As an example, assume the following settings:

- PwmMasterSel it 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 eigth PFC PWM cycle.
- PFCSyncRatio is set to 3, configuring PFC A/D sampling on every fourth PFC PWM cycle (10 KHz).

Then, if PFCPhasing is set to 0, PFC A/D sampling is synchronized to the Motor 1 sync pulse. If PFCPhasing is set to 1, PFC A/D sampling is delayed one cycle following the Motor 1 sync pulse, and so on. Figure 81 illustrates the timing of PFC A/D sampling when PFCPhasing is set to 2.

The PFC A/D sampling frequency should be a multiple of the master 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 must be less than (PFCSyncRatio + 1), or else the PFC will not run at all.

Bits 8 – 15 Unused; set to 0

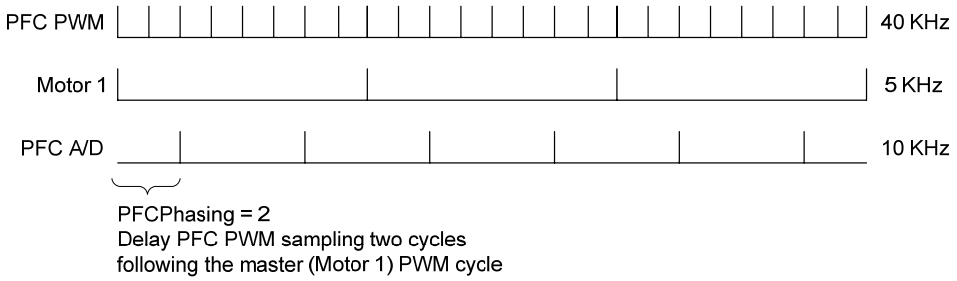


Figure 81. PFCPhasing Example

4.4.20 System Read Register Group

pwm_lines

Address:	*B4h*	*Range:*	*Unsigned output with bit field definitions*	*Reset value:*	*undefined*

Scaling or Notation: See description.

Description: This register provides internal PWM Gating bits. The IC PWM output pins are related to these bits as configured using registers port_ctrl0 and port_ctrl1 (see Section 4.4.1). (Note that Motor1 bit order is correct as listed below.)

Motor1	Motor2		
Bit 7	Bit 0	**PWMUH**	U phase high side output
Bit 6	Bit 1	**PWMUL**	U phase low side output
Bit 8	Bit 2	**PWMVH**	V phase high side output
Bit 9	Bit 3	**PWMVL**	V phase low side output
Bit 10	Bit 4	**PWMWH**	W phase high side output
Bit 11	Bit 5	**PWMWL**	W phase low side output
Bit 12		**PWMPFC**	PFC gating
Bits 13 – 15		Unused	

AIN1, AIN2, AIN3, AIN4, AIN5, AIN6

Address:	*ABh (AIN1) – B0h (AIN6)*	*Range:*	*Unsigned output 0 – 4095*	*Reset value:*	*undefined*

Scaling or Notation: 0 – 1.2V maps to 0 – 4095 digital counts

Description: A/D converter raw output for IC pin inputs AIN1 – AIN6. These signals are sampled once every six PWM cycles of Motor 1. AIN2 – AIN6 are not available on the IRMCx311.

AIN_TEST

Address:	*B1h*	*Range:*	*Unsigned output 0 – 4095*	*Reset value:*	*undefined*

Description: This register is reserved for future use.

DcBusVolts_DG

Address:	*B5h*	*Range:*	*Unsigned output 0 – 4095*	*Reset value:*	*undefined*

Scaling or Notation: DC bus voltage = DcBusVolts_DG / VdcScl [Volts]
where VdcScl is the dc bus scaling in digital counts per volt. VdcScl relates the actual voltage to the raw A/D counts. VdcScl is an entry ("dc bus Scale") in the MCEWizard.

Description: This register provides synchronized dc bus voltage feedback. This signal is synchronized to PWM cycle data transfer. It is generated from DcBusVolts (see Section 4.4.22).

DcBusVoltsFilt

Address:	*B2h*	*Range:*	*Unsigned output 0 – 4095*	*Reset value:*	*undefined*

Scaling or Notation: Dc bus voltage = DcBusVoltsFilt / VdcScl [Volts]
where VdcScl is the dc bus scaling in digital counts per volt. VdcScl relates the actual voltage to the raw A/D counts. VdcScl is an entry ("dc bus Scale") in the MCEWizard.

Description: This register provides filtered (0.492 msec time constant) dc bus voltage feedback. This is the filtered version of the raw dc bus voltage feedback (see register DcBusVolts in Section 4.4.22).

FaultFlags

Address:	*B3h*	*Range:*	*Unsigned output with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This register provides drive fault status. Most faults are handled outside the Faults module by a fault handling routine (2 msec execution rate on the 8051 processor) with the exception of Gate Kill faults. Gate Kill faults are handled within the Faults module and will instantly initiate inverter and regulator shutdown.. The FaultFlags register indicates currently pending fault conditions. The FaultClear register (Section 4.4.1) is used to reset fault conditions.

For all bit fields defined below, a value of 1 indicates that the corresponding fault condition is pending.

Bit 0	**OvFault**	dc bus over voltage trip fault. For more information, refer to Section 4.4.17 (DcBusOvLevel) and 4.4.1 (bit DcMonitorEn in registesr syscfg).
Bit 1	**LvFault**	dc bus under voltage trip fault. For more information, refer to Section 4.4.17 (DcBusLvLevel) and 4.4.1 (bit DcMonitorEn in registesr syscfg).
Bit 2	**PwmSyncErr**	Pwm synchronization error fault. This fault indicates that Motor 1, 2 and PFC are out of synchronization. Values for registers PwmPeriodConfig (Section 4.4.2) and PFCPwmPeriod (Section 4.4.19) must be calculated correctly to achieve PWM synchronization. Refer to Section 4.4.1 (PwmSyncEnb) for more information.
Bit 3	**PFCGateKill**	PFC Gate Kill fault. Refer to Section 4.4.19 (GkillFiltCnt_PFC) for more information.
Bit 4	**GateKill_2**	Motor 2 Gate Kill fault. Refer to Section 4.4.2 (GkillFiltCnt) for more information.
Bit 5	Unused (reserved)	
Bit 6	**PhsLossFlt_2**	Motor 2 phase loss fault. Refer to Sections 4.4.11 (PhsLosThr) and 4.4.9 (bit PhsLosDisable in register MtrCtrlBits) for more information.
Bit 7	**ZeroSpdFlt_2**	Motor 2 zero speed fault. Refer to Sections 4.4.6 (Min_Spd) and 4.4.9 (bit ZeroSpdDisable in register MtrCtrlBits) for more information.
Bit 8	**GateKill_1**	Motor 1 Gate Kill fault. Refer to Section 4.4.2 (GkillFiltCnt) for more information.
Bit 9	Unused (reserved)	
Bit 10	**PhsLossFlt_1**	Motor 1 phase loss fault. Refer to Sections 4.4.11 (PhsLosThr) and 4.4.9 (bit PhsLosDisable in register MtrCtrlBits) for more information.
Bit 11	**ZeroSpdFlt_1**	Motor 1 zero speed fault. Refer to Sections 4.4.6 (Min_Spd) and 4.4.9 (bit ZeroSpdDisable in register MtrCtrlBits) for more information.
Bit 12	**MCEFlt**	The MCE has generated a fault condition.
Bits 13 – 15	Unused (reserved)	

4.4.21 System Status Read Register Group

StatusFlags

Address:	*C8h (Motor1)* *DBh (Motor2)*	*Range:*	*Unsigned output* *with bit field definitions*	*Reset value:*	*0*

Scaling or Notation: See description.

Description: This register provides status of the motor controller.

Bit 0 **TwoPhsEnable**
- 0 Two phase modulation is not enabled
- 1 Two phase modulation is enabled

Bit 1 **FocEnable**
- 0 Field-Oriented Control regulators are not enabled
- 1 Field-Oriented Control regulators are enabled

Bit 2 **PwmEnable**
- 0 PWM gatings are not enabled
- 1 PWM gatings are enabled

Bit 3 **ClosedLoop**
- 0 Closed-loop mode is not enabled
- 1 Closed-loop mode is enabled

Bit 4 **ParkingDone**
- 0 Parking stage is not complete.
- 1 Parking done; Parking stage has been accomplished.

Bit 5 **ParkingOne**
- 0 First stage of Parking is not complete
- 1 First stage (25% of the total parking duration) of Parking has been accomplished

Bit 6 **StartFail**
- 0 No startup failure
- 1 Startup has failed (latched until drive restart occurs).

Bit 7 **StartOk**
- 0 Startup in progress or drive stopped
- 1 Startup has succeeded (cleared whenever drive stops)

Bits 8 – 15 Unused

4.4.22 DC Bus Voltage Read Register Group

DcBusVolts

Address:	*E4h*	*Range:*	*Unsigned output* *0 - 4095*	*Reset value:*	*undefined*

Scaling or Notation: See description.

Description: This register provides dc bus voltage feedback.

Scaling: Dc bus voltage = DcBusVolts / VdcScl [Volts]

where VdcScl is the dc bus scaling in digital counts per volt. VdcScl relates the actual voltage to the raw A/D counts. VdcScl is an entry ("dc bus Scale") in the MCEWizard.

The value of this register can be traced in MCEDesigner under the name DC_BUS_VOLTS.

4.4.23 FOC Diagnostic Data Read Register Group

Di

Address:	*C0h (Motor1)* *D3h (Motor2)*	*Range:*	*Signed output* *-16384 – 16383*	*Reset value:*	*undefined*

Scaling or Notation: 4095 = rated motor current [Amps]

Description: This register provides flux current (d-axis) feedback. This signal is reconstructed from single current shunt current feedback.

Dv

Address:	*C2h (Motor1)* *D5h (Motor2)*	*Range:*	*Signed output* *-2048 – 2047*	*Reset value:*	*undefined*

Scaling or Notation: 1430 = 100 [% modulation]

Description: This register provides d-axis command modulation index (output of d-axis current regulator).

Flx_Alpha

Address:	*BCh (Motor1)* *CFh (Motor2)*	*Range:*	*Signed output* *-32768 – 32767*	*Reset value:*	*undefined*

Scaling or Notation: 5000 = 100 [% rated flux]

Description: This register provides estimated motor flux of Alpha axis. This signal is calculated by the flux estimator. At speeds less than 5% rated, the estimated flux amplitude will start to decrease gradually. This is a consequence of the transfer characteristics of the flux estimator to reject dc offset.

Flx_M

Address:	*BBh (Motor1)* *CEh (Motor2)*	*Range:*	*Unsigned output* *0 – 8191*	*Reset value:*	*undefined*

Scaling or Notation: % rated flux = $\dfrac{Flx_M \times 16}{RatedFlxCounts \times 1.647 \times M}$ x 100 [%]

where: RatedFlxCounts = 5000 (default of MCEWizard)
if (Use4xMagScl = 1) M = 4
Elseif (Use2xMagScl = 1) M = 2
Else M = 1

Description: This signal represents the fundamental flux amplitude. (see Figure 79 for the generation of Flx_M).

I_Alpha

Address:	*BEh (Motor1)* *D1h (Motor2)*	*Range:*	*Signed output* *-32768 – 32767*	*Reset value:*	*undefined*

Scaling or Notation: See description.

Description: This register provides Alpha phase current of the Alpha-Beta orthogonal frame (Alpha aligned with u-phase). This current is calculated from the dc bus link current feedback.

The scaling of this variable depends on the analog gain setup of the current feedback path ("Current Amp. gain" setting in the MCEWizard). Ai_Bi_scale ("Ai Bi scale" setting in the MCEWizard) specifies 8-times scaling in digital counts/Amps peak for this variable.

Scaling = 8 * Ai_Bi_scale [digital counts/Amps peak]

I_Beta

Address:	*BFh (Motor1)* *D2h (Motor2)*	*Range:*	*Signed output* *-32768 – 32767*	*Reset value:*	*undefined*

Scaling or Notation: See description.

Description: This register provides Beta phase current of the Alpha-Beta orthogonal frame (Alpha aligned with u-phase). This current is calculated from the dc bus link current feedback.

The scaling of this variable depends on the analog gain setup of the current feedback path ("Current Amp. gain" setting in the MCEWizard). Ai_Bi_scale ("Ai Bi scale" setting in the MCEWizard) specifies 8-times scaling in digital counts/Amps peak for this variable.

Scaling = 8 * Ai_Bi_scale [digital counts/Amps peak]

IdRef_C

Address:	*C5h (Motor1)* *D8h (Motor2)*	*Range:*	*Signed output* *-16384 – 16383*	*Reset value:*	*undefined*

Scaling or Notation: 4095 = 100 [% rated motor current]

Description: This register provides d-axis command current. This is the current command being used by the d-axis current regulator.

IqRef_C

Address:	*C4h (Motor1)* *D7h (Motor2)*	*Range:*	*Signed output* *-16384 – 16383*	*Reset value:*	*undefined*

Scaling or Notation: 4095 = 100 [% rated motor current]

Description: This register provides q-axis command current. This is the current command being used by the q-axis current regulator.

Qi

Address:	*C1h (Motor1)* *D4h (Motor2)*	*Range:*	*Signed output* *-16384 – 16383*	*Reset value:*	*undefined*

Scaling or Notation: 4095 = 100 [% rated motor Amps]

Description: This register provides torque current (q-axis) current feedback.

Qv

Address:	*C3h (Motor1)* *D6h (Motor2)*	*Range:*	*Signed output* *-2048 – 2047*	*Reset value:*	*undefined*

Scaling or Notation: 1430 = 100 [% modulation]

Description: This register provides q-axis command modulation (output of q-axis current regulator).

RotatorAngle

Address:	*C6h (Motor1)* *D9h (Motor2)*	*Range:*	*Unsigned output* *0 – 4095*	*Reset value:*	*undefined*

Scaling or Notation: 1024 = 90 [Deg.]

Description: This is the estimated rotor angle. It is used for the Field-Oriented control reference frame.

V_Alpha

Address:	*BDh (Motor1)* *D0h (Motor2)*	*Range:*	*Signed output* *-32768 – 32767*	*Reset value:*	*undefined*

Scaling or Notation: See description.

Description: This register provides Alpha motor phase voltage. This signal is constructed by dc bus voltage feedback and modulation index (input of SVPWM).

The scaling of this signal depends on motor parameters and application settings and is entered in the MCEWizard ("V_Alpha Scale").

IfbV

Address:	*B9h (Motor1)* *CCh (Motor2)*	*Range:*	*Signed output* *-32768 – 32767*	*Reset value:*	*undefined*

Scaling or Notation: See description.

Description: This register provides reconstructed motor phase V current (offset eliminated). This current is calculated from the dc bus link current feedback, sampled on every PWM cycle of the corresponding motor (motor 1 or motor 2).

The scaling of this variable depends on the analog gain setup of the current feedback path ("Current Amp. gain" entry in the MCEWizard). The entry "Ai Bi scale" in the MCEWizard specifies the scaling in digital counts/Amps peak for this variable.

IfbW

Address:	*BAh (Motor1)* *CDh (Motor2)*	*Range:*	*Signed output* *-32768 – 32767*	*Reset value:*	*undefined*

Scaling or Notation: See description.

Description: This register provides reconstructed motor phase W current (offset eliminated). This current is calculated from the dc bus link current feedback, sampled on every PWM cycle of the corresponding motor (motor 1 or motor 2).

The scaling of this variable depends on the analog gain setup of the current feedback path ("Current Amp. gain" entry in the MCEWizard). The entry "Ai Bi scale" in the MCEWizard specifies the scaling in digital counts/Amps peak for this variable.

SpdFbk

Address:	*C7h (Motor1)* *DAh (Motor2)*	*Range:*	*Signed output* *-16384 – 16383*	*Reset value:*	*undefined*

Scaling or Notation: 16383 = MaxRpm [rpm]
where MaxRpm is the maximum application speed (entry "Max RPM" in the MCEWizard).

Description: This register provides filtered (typically 3 to 7 msec) motor speed. This signal is generated by write register MotorSpeed (Section 4.4.6). It is synchronously sampled to coordinate with other output signals (Alpha-Beta currents).

4.4.24 Velocity Status Read Register Group

Rtr_Freq

Address:	*B7h (Motor1)* *CAh (Motor2)*	*Range:*	*Signed output* *-32768 – 32767*	*Reset value:*	*undefined*

Scaling or Notation: See description.

Description: This register provides estimated unfiltered rotor electrical frequency. The rotor electrical frequency is the same as the stator fundamental frequency for synchronous motors.

Rotor electrical frequency = Rtr_Freq * FreqPwm * FreqScl / 2^20 [Hz]

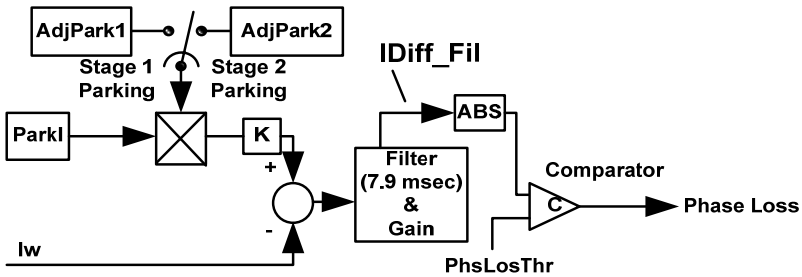
where:
FreqPwm is the inverter pwm switching frequency in Hz; and
FreqScl is the frequency scaler, determined as:
if (Use4xFreqScl = 1) FreqScl = 4
else if (Use2xFreqScl = 1) FreqScl = 2
else FreqScl = 1
(Use4xFreqScl and Use2xFreqScl are bit fields of the MtrCtrlBits_S and MtrCtrlBits registers, respectively. See Section 4.4.9.)

IDiff_Fil

Address:	*B8h (Motor1)* *CBh (Motor2)*	*Range:*	*Unsigned output* *0 - 4095*	*Reset value:*	*undefined*

Scaling or Notation: Parking current error = IDiff_Fil × / Ai_Bi_Scl [Amps peak]
where Ai_Bi_Scl is the current scaling (counts/Amp peak calculated (AiBiScale) and displayed in the MCEWizard).

Description: This register provides absolute parking current error for Phase Loss detection. This signal represents the absolute current error between anticipated w-phase current (calculated from parking current and the actual w-phase feedback current. During parking, if the anticipated w-phase current should match up with the actual w-phase current at the end of the parking duration. If the absolute difference is larger than a certain threshold (PhsLosThr), phase loss is assumed. This register is output for diagnostic purposes. Diagram below shows how IDiff_Fil is calculated.



4.4.25 Current Feedback Offset Read Register Group

Id_Decoupler

Address:	*EAh (Motor1)* *EBh (Motor2)*	*Range:*	*Signed output* *-16384 – 16383*	*Reset value:*	*undefined*

Scaling or Notation: 4095 = 100 [% rated motor current]

Description: This register provides d-axis Current Decoupler output. The Current Decoupler provides the optimal current angle for maximum torque per ampere control. In practice, IdRefExt input (see Section 4.4.16) should include Id_Decoupler.

IfbOffset

Address:	*B6h (Motor1)* *C9h (Motor2)*	*Range:*	*Signed output* *-32768 – 32767*	*Reset value:*	*undefined*

Scaling or Notation: See description

Description: This register provides current feedback dc offset compensation. The scaling of this variable depends on the analog gain setup of the current feedback path ("Current Amp. gain" in the MCEWizard). The entry "Ai Bi scale" in the MCEWizard specifies the scaling in digital counts/Amps peak for this variable.

4.4.26 PFC Status Read Register Group

V_IN

Address: *DCh* *Range:* *Unsigned output 0 – 2047* *Reset value:* *undefined*

Scaling or Notation: See description.

Description: This parameter provides the A/D feedback signal of AC input voltage, as an absolute value, or rectified half-wave AC input voltage. The following table shows how the input voltage correlates to values of the V_IN register.

ADC Input Voltage	V_IN Digital Count
0 V	2047
0.3 V	1023
0.6 V	0
0.9 V	1023
1.2 V	2047

This register can be traced in MCEDesigner under the name VPFC_REC.

VinSense

Address: *DDh* *Range:* *Unsigned output 0 – 4095* *Reset value:* *undefined*

Scaling or Notation: See description.

Description: This parameter provides the A/D feedback signal of the bi-polar AC input voltage, shifted by the average value to remove offset. The RawVinSense signal is averaged and shifted so that the average corresponds to 2048 (the midpoint of the A/D range).
This register can be traced in MCEDesigner under the name VPFC_AC.

RawVinSense

Address: *DEh* *Range:* *Unsigned output 0 – 4095* *Reset value:* *undefined*

Scaling or Notation: See description.

Description: This parameter provides the raw A/D feedback signal of the AC input voltage, as a bi-polar value, or bi-polar full-wave AC input voltage. The following table shows how the input voltage correlates to values of the RawVinSense register.

ADC Input Voltage	RawVinSense Digital Count
0 V	0
0.3 V	1023
0.6 V	2048
0.9 V	3071
1.2 V	4095

I_IN

Address: *E0h* *Range:* *Unsigned output 0 – 2047* *Reset value:* *undefined*

Scaling or Notation: See description.

Description: This parameter provides the A/D feedback signal of AC input current. In order to achieve the correct current feedback, the IPFC operational amplifier must be set up for inverting input. The following table shows how the input voltage correlates to values of the I_IN register.

ADC Input Voltage	V_IN Digital Count
0 V	2047
0.3 V	1023

0.6 V	0
> 0.6 V	0 (the sensed PFC current is uni-directional)

This register can be traced in MCEDesigner under the name IPFC.

ShutDown

Address:	*E5h*	*Range:*	*Boolean output* *0 or 1*	*Reset value:*	*undefined*

Scaling or Notation: 0 = PFC PWM output is enabled; 1 = PFC PWM output is disabled.

Description: This parameter is an output signal of the PFC_PWM block's Blanking module. ShutDown, together with the PFCEnable input (Section 4.4.19), enables or disables the PFC PWM output. When Shutdown is 0, PFC PWM output is enabled if PFCEnable is set to 1. When ShutDown is 1, PFC PWM output is disabled regardless of the value of PFCEnable. For detailed information on the setting and clearing of Shutdown, see Section 4.3.8.2.

This output can be used in an MCE design or 8051 application code to enable, disable or reset the PFC control loop (voltage, current, etc.).

PfcOffsetV

Address:	*E6h*	*Range:*	*Unsigned output* *0 – 4095*	*Reset value:*	*undefined*

Scaling or Notation: 1 count = (1 / 4095) * Maximum Vdc [Volt]

Description: This parameter provides the offset digital count of AC input voltage feedback, corresponding to the range of Vdc ADC feedback digital count (0 – 4095). It is the responsibility of the hardware designer to ensure that Maximum Vdc corresponds to a count of 4095. Typically, Vdc = 492V matches a count of 4095.

PfcOffsetI

Address:	*E7h*	*Range:*	*Unsigned output* *0 – 4095*	*Reset value:*	*undefined*

Scaling or Notation: 2047 = Maximum IPFC [Amps]

Description: This parameter provides the offset digital count of AC input current feedback. It is the responsibility of the hardware designer to ensure that maximum IPFC matches the value of 2047 in PfcOffsetI.

5 8051 / MCE Interface

This section describes the methods by which the 8051 processor and MCE communicate. This communication can be divided into three categories:

- Shared RAM
- MCE motion peripheral configuration register interface
- Interrupts from the MCE to the 8051

5.1 The Shared RAM

The IRMCF300 contains 8192 bytes of RAM, all of which is accessible to both the 8051 processor and the MCE. (The IRMCF300 also contains 48K bytes of 8051 private program RAM for software development, described in Section 1.2.) From the 8051 processor, the shared RAM is accessed as external data RAM at address range 0xE000 – 0xFFFF. The shared RAM is logically divided into three regions:

- MCE data RAM
- MCE program RAM
- 8051 data RAM

Typically 512 bytes are allocated for MCE data. These locations are used for MCE private storage and for information passed between the MCE and the 8051. Both the 8051 and the MCE access this area of RAM for reading and writing.

Up to an additional 5632 bytes are allocated for MCE instruction (program) space. The hardware loads the MCE program into this area of RAM at power up (as described in Section 2). The 8051 does not read or write this area of RAM.

The upper 2k bytes of RAM are available for 8051 data storage. The MCE does not access this area.

The boundaries between the three sections of RAM are dynamic and determined by the MCE compiler at compilation time. The compiler always reserves 512 bytes for MCE data at address 0xE000, beginning the MCE program at address 0xE200. Depending on the size of the MCE application program, the compiler may allocate less than the allotted 5632 bytes for MCE program, in which case more memory could be used for 8051 data RAM. For example, if the MCE program is smaller than 3584 bytes, 8051 data could begin at address 0xF000 instead of 0xF800. The 8051 data space is defined in the 8051 compiler used to generate the code (in Keil uVision, it is found in “Options for Target”). Note that the compiler displays the total MCE program size in words, so the displayed size must be multiplied by two to determine the program size in bytes.

5.1.1 Reading and Writing Shared RAM

The MCE is based on 16-bit processing. All of its memory accesses are on 16-bit boundaries and it always reads and writes 16-bit words. However, the 8051 is an 8-bit processor and reads and writes only 8-bit words. This could potentially lead to a race hazard when the two processors access shared RAM, since the 8051 requires two memory accesses to read or write a 16-bit word. The 8051 could potentially read the first byte of a word and the MCE could modify the word before the 8051 reads the second byte, resulting in corrupted data.

To prevent data corruption in shared RAM when reading and writing 16-bit values, special hardware assistance is implemented using an extension register defined as a group of 8051 special function registers (SFRs). Whenever an 8051 application reads or writes a 16-bit or 32-bit value that is shared with the MCE processor, it should use this special “coherent data” mechanism to insure data integrity. This mechanism need not be used for 8-bit accesses or when the 8051 is accessing data in shared RAM that is not shared with the MCE.

The coherent data mechanism also allows 32-bit reads and writes to be performed from the 8051 processor without data corruption. However, all MCE shared data is defined as 16-bit values so this operation is not described here.

MCE COHERENT DATA (MCECD0, MCECD1, MCECD2, MCECD3)

Address:	*E9h (MCECD0),* *EAh (MCECD1),* *EBh (MCECD2,* *ECh (MCECD3)*	*Not Bit Addressable*	*Reset value:*	*00000000b*

Since the MCE uses 16-bit addressing and the 8051 uses 8-bit addressing, all MCE data are aligned at even addresses in 8051 memory space. The MCE stores data in little-endian (Intel) byte ordering, so within each 16-bit value, the low-order byte is at the lower (even) 8051 address and the high-order byte is located at the upper (odd) 8051 address.

To write a 16-bit value to shared RAM at address *N*, use the following procedure:

1. Disable interrupts if necessary to prevent other 8051 accesses to shared memory while the operation is in progress.
2. Write the low-order byte of data to register MCECD2.
3. Write the high-order byte of data to the odd address (*N* + 1).
4. Re-enable interrupts if they were disabled at step 1.

The procedure for reading a 16-bit value from shared RAM depends on whether the address to be read is at a longword (32-bit) boundary. An address is at a longword boundary if it is evenly divisible by four. When all values are aligned at even addresses (as they are in MCE shared memory), an easy way to check for a longword boundary is to test bit 1 of the address. If bit 1 is zero, the address is at a longword boundary.

Use the following procedure to read a 16-bit value from shared RAM at address *N*:

1. Disable interrupts if necessary to prevent other 8051 accesses to shared memory while the operation is in progress.
2. Read the low-order byte from the even address (*N*).
3. If the address *N* is at a longword boundary, read the high-order byte from register MCECD1. Otherwise, read the high-order byte from register MCECD3.
4. Re-enable interrupts if they were disabled at step 1.

The sample 8051 code included with the Reference Design Kits, IRSamples, contains predefined functions to correctly handle the shared RAM access from the 8051.

5.1.2 Arbitration

RAM arbitration is required because both processors (the 8051 and the MCE) can attempt simultaneous access to the shared RAM. In general, the arbiter gives the MCE sequencer precedence over the 8051. Since the 8051 program and internal data RAM are not shared the 8051 can generally execute instructions without restriction. 8051 instruction execution is held off only when the instruction accesses shared RAM and an MCE RAM access is pending or in progress. The MCE, on the other hand, fetches its instructions from shared RAM and stores all of its data there. In addition, the MCE performs many time critical operations that cannot be delayed. MCE instruction execution is held off only if an 8051 RAM access is already in progress or if the 8051 has been stalled for more than five clock cycles due to continued MCE RAM access.

5.2 Motion Peripheral Register Interface

To read and write the motion peripheral registers described in Section 4.4, an 8051 application must follow a specific procedure using the MCE Access SFRs as described below.

MCE ACCESS ADDRESS (MCEAAH, MCEAAL)

Address:	*EEh (MCEAAH), EFh (MCEAAL)*	*Not Bit Addressable*	*Reset value:*	*00000000b*

MCE ACCESS DATA (MCEAD0, MCEAD1, MCEAD2, MCEAD3)

Address:	*D1h (MCEAD0), D2h (MCEAD1), D3h (MCEAD2, D4h (MCEAD3)*	*Not Bit Addressable*	*Reset value:*	*00000000b*

MCE STATUS REGISTER (MCESS)

Address:	*DDh*	*Not Bit Addressable*	*Reset value:*	*00000000b*

MCESS.7	MCESS.6	MCESS.5	MCESS.4	MCESS.3	MCESS.2	MCESS.1	MCESS.0
BUSY	**STALL**	-	-	-	-	-	-
R	R						

SYNC.7	**BUSY**	SFR access indication. This bit will be set during any SFR access.
SYNC.6	**STALL**	MCE stall indication. This bit is set while a green block is executing or while the MCE is busy reading data from the data memory.
MCESS.5 - MCESS.0	-	*Not implemented. Returns zero when read.*

All MCE motion peripheral configuration registers are 16-bit values. An address is assigned to each register. (These correspond to locations in MCE private address space. They are not shared RAM addresses.) The MCE access mechanism also allows 32-bit registers to be read and written but that operation is not described here since the MCE defines no 32-bit registers.

To write to a 16-bit motion peripheral configuration register, use the following procedure:

1. Disable interrupts if necessary to prevent other 8051 accesses to the configuration registers while the operation is in progress.
2. Write the low-order byte of the register address to MCEAAL.
3. Write the high-order byte of the register address to MCEAAH.
4. Write the low-order byte of the data value to MCEAD2.
5. Write the high-order byte of the data value to MCDAD3.
6. Re-enable interrupts if they were disabled at step 1.

To read a 16-bit motion peripheral configuration register, use the following procedure:

1. Disable interrupts if necessary to prevent other 8051 accesses to the configuration registers while the operation is in progress.
2. Write the low-order byte of the register address to MCEAAL.
3. Write the high-order byte of the register address to MCEAAH.
4. Read the MCESS register and wait if necessary for the BUSY bit (MCESS.7) to be cleared (zero).
5. Read the low-order byte of the data value from MCEAD0.
6. Read the high-order byte of the data value from MCDAD1.
7. Re-enable interrupts if they were disabled at step 1.

The sample 8051 code included with the Reference Design Kits, IRSamples, contains predefined functions to correctly handle the reading and writing of motion peripheral registers from the 8051.

5.3 Interrupts from the MCE to the 8051

Two interrupt sources are defined for communication between the MCE and the 8051. The first is the general-purpose MCE interrupt, which can be generated from the MCE, but is currently unused. The second is the SYNC interrupt, which is generated from the MCE to signal the 8051 that a SYNC pulse has occurred.

The SYNC interrupt is a periodic event signal generated by the MCE. Its timing is illustrated in Figure 82. This is the most important signal used for synchronization between the 8051 (CPU side) and the MCE (motion control side). An 8051 application software task that needs to pass commands to the MCE and/or receive updated data from the MCE may require specific synchronization with the MCE. This is due to the fact that MCE computation is initiated and triggered by the SYNC pulse at every PWM carrier frequency period. It is also true that six PWM outputs to the power device gate drive will occur at exactly one clock moment of the system clock at the beginning of the SYNC event. If synchronization is not implemented and the 8051 application software writes multiple data items to the MCE via the shared RAM, it is possible that some of the data are written in the previous MCE scan period while the rest of data are written in the current MCE scan period.

Therefore, an 8051 application software should use the SYNC signal for synchronization to insure that multiple data items are updated or read coherently within a particular scan period.

The SYNC signal is also generated in an execution overrun fault condition, which occurs if the MCE does not complete its processing (indicated by the bar labeled "MCE computation" in Figure 82) before the end of the PWM period (*i.e.*, before the next SYNC pulse).

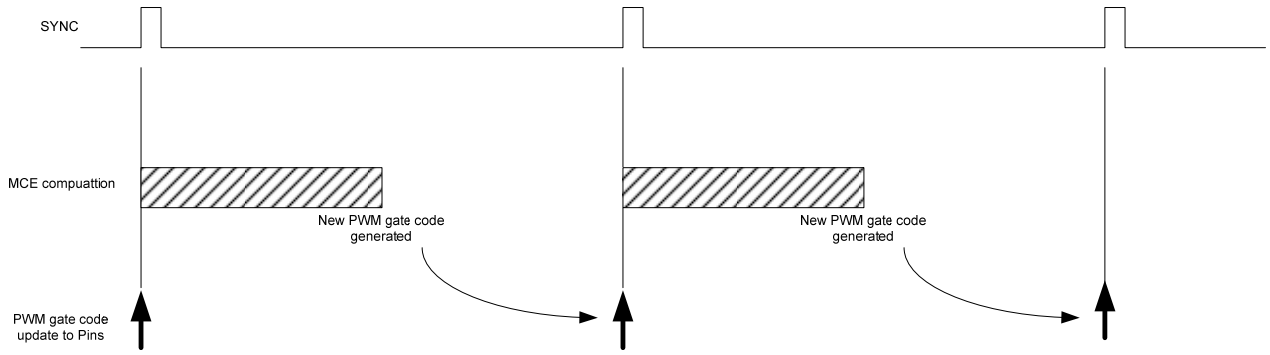


Figure 82. Timing of Sync and MCE Computation

When a SYNC interrupt occurs, the SYNC Status register can be read to determine the cause of the interrupt as shown below. To clear the SYNCS register after servicing the interrupt, write zero to the appropriate bit(s).

SYNC STATUS REGISTER (SYNCS)

Address: DEh *Not Bit Addressable* *Reset value: 00000000b*

SYNC.7	SYNC.6	SYNC.5	SYNC.4	SYNC.3	SYNC.2	SYNC.1	SYNC.0
-	**-**	**EXEF2**	**EXEF1**	**EXEFP**	**SYNC2**	**SYNC1**	**SYNCP**

Bit definitions for this register are as follows:

SYNC.7	**-**	*Not implemented. Returns zero when read.*
SYNC.6	**-**	*Not implemented. Returns zero when read.*
SYNC.5	**EXEF2**	If this bit is set, the interrupt was caused by a Motor 2 execution overrun fault.
SYNC.4	**EXEF1**	If this bit is set, the interrupt was caused by a Motor 1 execution overrun fault.
SYNC.3	**EXEFP**	If this bit is set, the interrupt was caused by a PFC execution overrun fault.
SYNC.2	**SYNC2**	If this bit is set, the interrupt was caused by a Motor 2 SYNC pulse.
SYNC.1	**SYNC1**	If this bit is set, the interrupt was caused by a Motor 1 SYNC pulse.
SYNC.0	**SYNCP**	If this bit is set, the interrupt was caused by a PFC SYNC pulse.

6 The MCE Development Process

This section describes how to use the MCE compiler, which allows engineers to easily realize a design using MATLAB's Simulink graphical user interface. Motion control blocks provided by International Rectifier in the form of a Simulink library represent the available IRMCF300 functions.

The MCE 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, which analyzes the Simulink design and generates a corresponding file that is executed by the MCE processor on the IRMCx3xx.
- MCEDesigner, which provides a graphical user interface to the IRCMx3xx to allow download of the MCE executable file, control of MCE operation, and analysis of system function and performance. MCEDesigner is described in a separate document.

The MCE development tools software distribution is organized beneath a main directory named `MCE Compiler`. The main directory contains three subdirectories: `Simulink Library`, which contains the Simulink library blocks; `Matlab`, which contains the MATLAB scripts that implement the graphical interfaces described in Sections 6.4 and 6.5; and `bin`, which contains the executable files and linkable object files for the MCE compiler.

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

6.1 MCE Design Generation

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, or layered, design.

The MCE compiler analyzes the graphical elements defined in a model file to generate MCE instructions to implement the represented design. The compiler has two modes of operation: it can process a model file that represents a complete IRMCK3xx system; or it can process a model that represents a subsystem or "macro block" to be used within a system design.

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 or a linkable (intermediate) object file for a macro block (subsystem) definition. For a complete system build, the compiler can also create 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 IRMCx31x.

6.2 Creating an MCE Design Using Simulink

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

Section 6.2.1 describes how to create a complete system design for execution on the IRMCK3xx. Section 6.2.2 describes how to create a macro block (subsystem) design that you can use as a building block in your system designs.

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.)

6.2.1 Creating a Complete System Design

This section describes how to create a complete system design for execution on the IRMCx31x. If you want to create a macro block that you can use in your system designs, refer to Section 6.2.2.

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 Simulink model file with the appropriate MCE subsystem hierarchy. The easiest way to do this is to make a copy of the model file template.mdl 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 6.7.

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. Refer to Section 6.3 for more information about the MCE library blocks and other design components.

Step 4.
Customize your read and write register blocks. Write register blocks define configurable 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.

Step 5.
When you are satisfied with your Simulink design, it's time to run the compiler. This procedure is detailed in Section 6.4.

6.2.2 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 IRMCx31x, refer to Section 6.2.1.

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. Refer to Section 6.3 for more information about the MCE library blocks and other design components.

To define inputs and outputs for your macro block, use Simulink input and output port elements. (Refer to Section 6.3.2 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 6.4.

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 6.3.1) 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.

6.3 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.

6.3.1 The MCE Library

The main window of MCE Simulink library is shown in Figure 83.

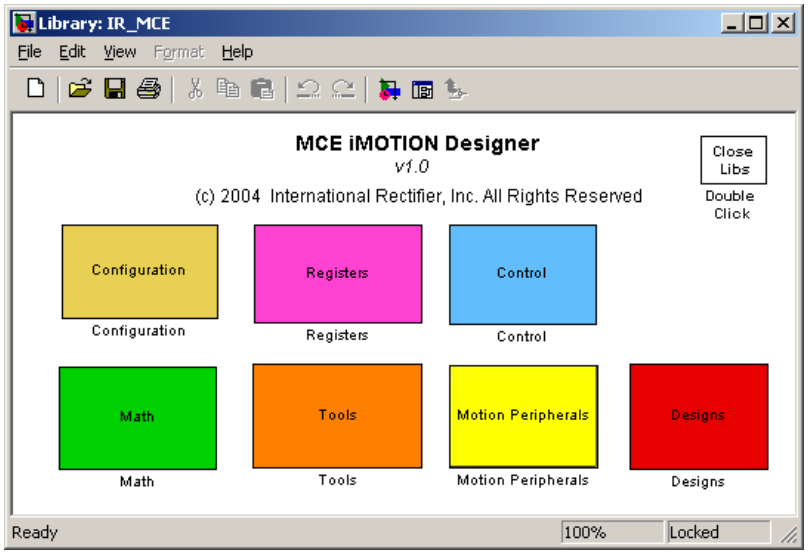


Figure 83. MCE Simulink Library

There are seven library groups, described below.

6.3.1.1 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`, these blocks are already included at the appropriate locations in the subsystem hierarchy. (See Section 6.7 for more information.) These blocks cannot be used in macro block definitions.

6.3.1.2 Registers

The Registers group contains host 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 other host interface) 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 configurable 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. These blocks cannot be used in macro block definitions.

6.3.1.3 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. Control blocks can also be used in macro block definitions.

6.3.1.4 Math

The Math group contains general-purpose math blocks that you can use in any of your control loop subsystems or in a macro block definition.

6.3.1.5 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 6.4 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 6.5) and a tool that allows you to customize the inputs and outputs of certain motion peripheral blocks (described in Section 6.6).

6.3.1.6 Motion Peripherals

The Motion Peripherals group contains the special-function motion peripheral blocks that can be included in your control loop subsystems and macro block definitions.

6.3.1.7 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 and macro blocks to this library group if you wish.

6.3.2 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.

6.3.2.1 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 6.7 for more information about the use of the Enabled Subsystem block in the MCE design hierarchy. Enabled subsystem blocks cannot be used in macro block definitions.

6.3.2.2 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.

6.3.2.3 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. Scope blocks cannot be used in macro block definitions.

6.3.2.4 Input Port

To create an input to a macro block, drag an In1 block into the macro block definition and give it a unique name to identify the input. When you encapsulate the design into a subsystem, Simulink creates an input port for the subsystem to represent each In1 block in the design. In1 blocks cannot be used in a complete system design. (A complete system is self-contained and has no external connections.)

6.3.2.5 Output Port

To create an output from a macro block, drag an Out1 block into the macro block definition and give it a unique name to identify the output. When you encapsulate the design into a subsystem, Simulink creates an output port for the subsystem to represent each Out1 block in the design. Out1 blocks cannot be used in a complete system design. (A complete system is self-contained and has no external connections.)

6.3.2.6 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 circuitous lines, you can also use Goto and From blocks to connect elements at distant points within the same subsystem or in a macro block definition.

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.

6.3.2.7 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.)

6.4 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 84.

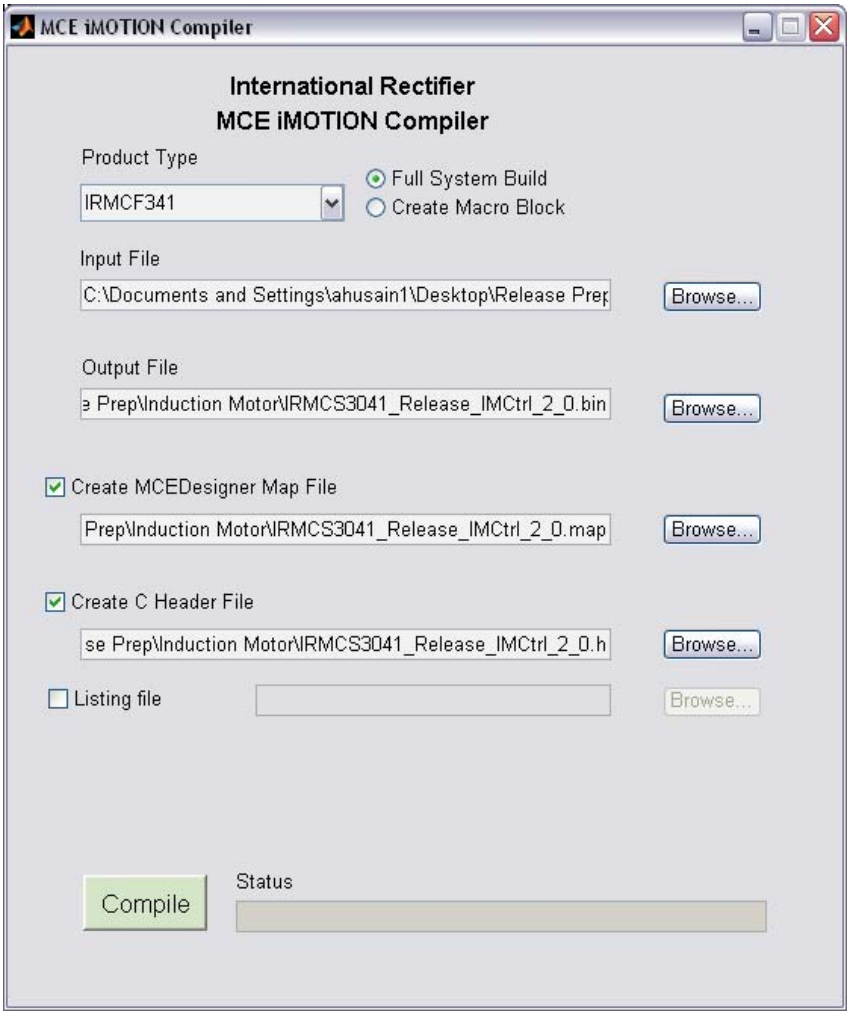


Figure 84. MCE Compiler Input Screen

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 85.

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 is too large to fit in the available memory. (Refer to Section 1.2 for information on memory size.) 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.

Note that 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 Section 4.2 are more accurate and provide a range of cycle counts when execution time varies depending on conditions. For this reason, the compiler's execution time estimate will generally exceed the estimate you would obtain by summing the documented execution times for each block in the design.

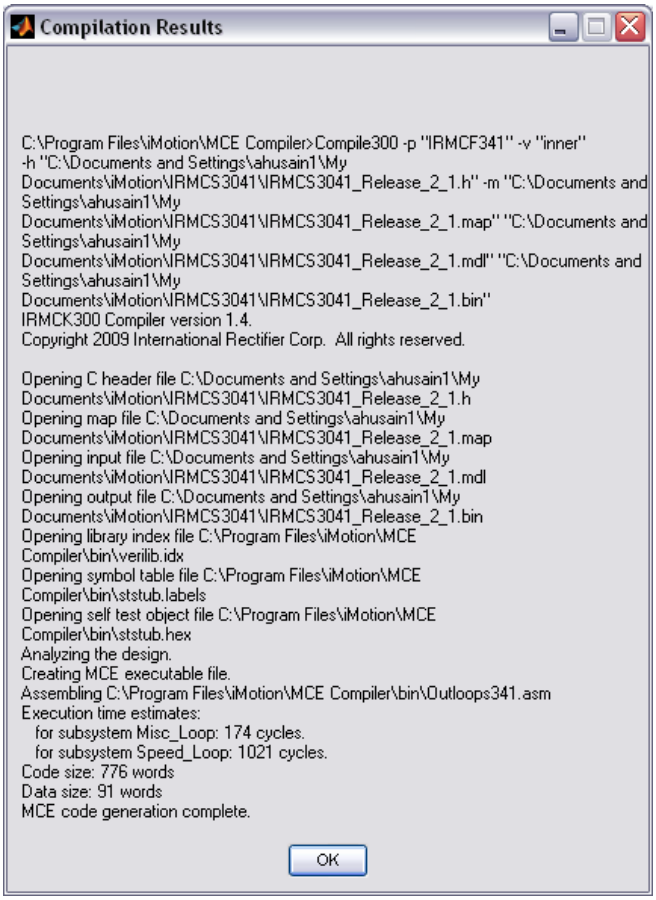


Figure 85. MCE Compiler Results Example

6.5 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 86.

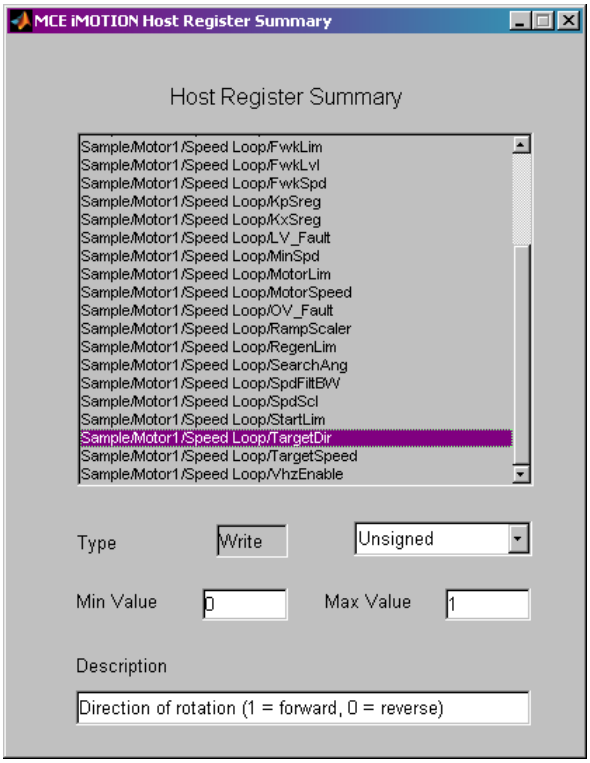


Figure 86. The Host Register Summary Utility

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.

Note that macro block definitions don't include host read and write registers, so the Host Register Summary utility is used only with full system designs.

6.6 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 selecting from lists of available signals.

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 87. 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 current-defined inputs for the block are shown in the list on the left-hand side of the window and the currently-defined outputs are shown on the right.

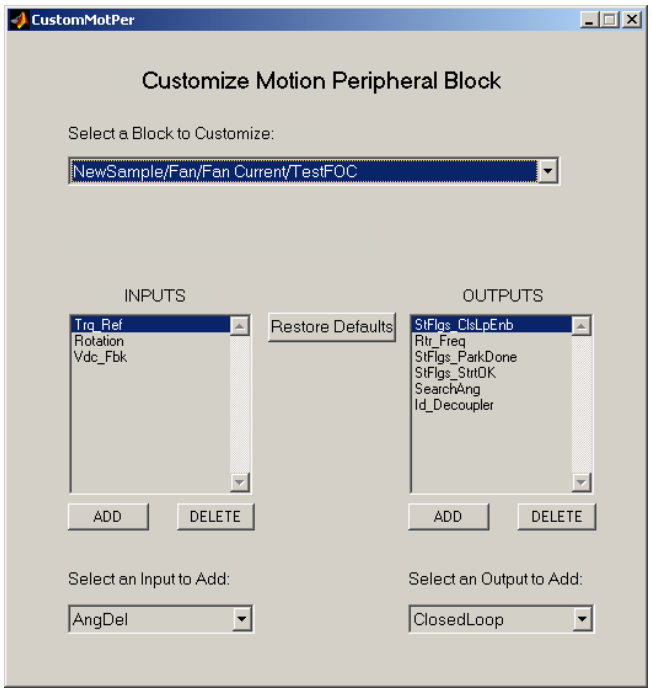


Figure 87. The CustomMotPer Utility

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.

6.7 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 88.

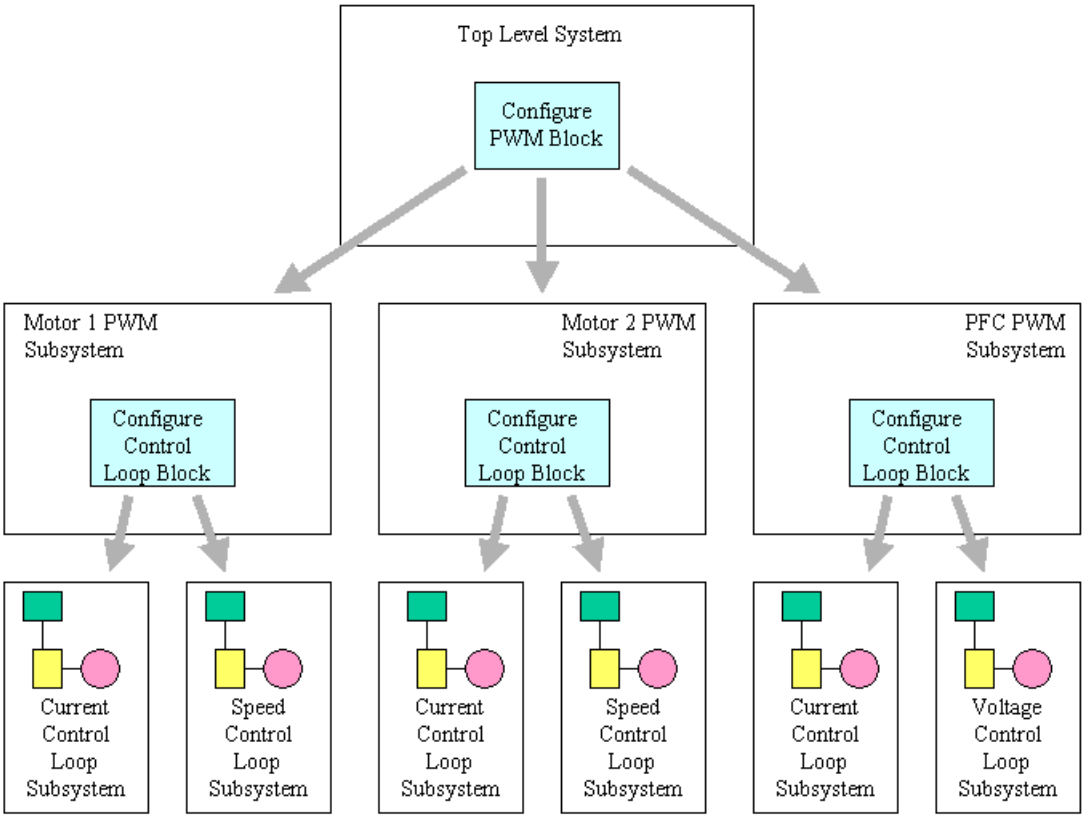


Figure 88. MCE Design Hierarchy

The top level of the system design contains a ***Configure PWM*** block and three PWM subsystem blocks, which are implemented using standard Simulink ***Enabled Subsystem*** blocks. The Configure PWM block has 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 block has three outputs, labeled ***Current***, ***Speed*** and ***Voltage***. 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. In the Motor 1 and Motor 2 PWM subsystems, the Current and Speed outputs are connected and the Voltage output is left unconnected. In the PFC subsystem, the Current and Voltage outputs are connected and the Speed output is left unconnected. 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.

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 three ***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 two ***Enabled Subsystem*** blocks into the model. Connect the Current and Speed 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. 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.

7 The 8051 Development Process

The IRMCx31x includes an 8051 microprocessor that can be used for motor control applications as well as more general applications such as an interface to a user control panel.

To support MCE development, International Rectifier provides the MCEDesigner tool, which allows a user to control and monitor the operation of the MCE by reading and writing host registers. MCEDesigner has two components: a user interface running on a PC and a helper application or "agent" running on the 8051 microprocessor. The agent software is supplied as an executable image that is stored in external EEPROM and loaded to IRMCx31x program RAM at power-up.

To assist the development of custom application software for the 8051, International Rectifier provides a number of source-code programming examples, such as interrupt handlers, UART driver and interface to the MCE shared RAM and RTL registers.

7.1 Source Code Samples

IR provides a sample Keil uVision2 project, the contents of which are described in the following sections. The project can be used to build a standalone 8051 executable image that performs simple operations and can communicate with a terminal emulation program such as Microsoft's Hyperterminal. The name of the uVision2 project file is IRsamples.Uv2.

Remember that you cannot use MCEDesigner to communicate with the IRMCx31x while you're running the sample code or custom software on the 8051.

7.1.1 EEPROM Programming

The files EepromI2C.c and EepromI2C.h show how to read and write EEPROM using the I^2C interface. Functions are provided to initialize the I^2C interface, read a byte from EEPROM and write a byte to EEPROM.

7.1.2 Register Interface

The following source files are provided to show how to interface to the MCE's shared RAM and RTL configuration registers:

regIf.c	Sample code to read and write RAM and RTL configuration registers.
regif.h	Sample register definitions used by regIf.c.
Coherent.SRC	Assembly language functions to read and write RAM registers using the coherent access SFRs (see Section 5.1.1).
SfrRegs.SRC	Assembly language functions to read and write RTL configuration registers using the MCE Access SFRs (see Section 5.2).

7.1.3 UART Driver

The files asyncDriver.c and asyncDriver.h provide a driver for the UART and a small sample application showing how to interface to the driver. The driver sets up the UART for standard 8-bit operation, handles transmit and receive interrupts, and buffers data using send and receive FIFOs (first-in-first-out buffers). The following functions are included:

asyncDriver	This is a test function that uses the UART read/write functions to read characters from the serial line, convert them from upper to lower case (or from lower to upper case) and write them back to the serial line. You can use this test with a Hyperterminal (or equivalent) connection. Each character you type in the Hyperterminal window should be echoed back to the window. Alphabetic characters a - z will echo as A - Z and characters A - Z will echo as a - z. Other characters are echoed exactly as entered.
sioIsr	This is the UART interrupt service routine, which handles transmit and receive interrupts. Received characters are placed in the receive FIFO. Characters to be transmitted are taken from the transmit FIFO.
sioInit	This function initializes the transmit and receive data structures and the SFRs that control the UART.
flushTx	This function initializes the transmit FIFO.
flushRx	This function initializes the receive FIFO.
setBaudRate	This function initializes the baud rate SFR for 57,600 bps, based on the default clock rate of 64 MHz.
putChar_	This function is called from a higher level (such as the asyncDriver test function) to transmit a character. If the transmitter is currently busy, it adds the character to the transmit FIFO. If no transmission is already in progress, it writes the character directly to the UART transmit buffer. The function returns 0 if the transmit FIFO is full (character cannot be accepted for transmission); or 1 if successful.
getChar_	This function is called from a higher level to read a received character from the receive FIFO. It returns 0 if the receive FIFO is empty (no character available) or 1 if successful.
xFifoRoom	This function is called from putChar_ to check the status of the transmit FIFO. It returns 0 if the transmit FIFO is full; 1 otherwise.
xFifoPutChar	This function is called from putChar_ to add a character to the transmit FIFO. It returns 0 if the transmit FIFO is full; 1 if the character was successfully added to the FIFO.
xFifoGetChar	This function is called from sioIsr to get the oldest character from the transmit FIFO. It returns 0 if the transmit FIFO is empty; 1 if a character is removed from the FIFO.
rFifoRoom	This function is called from sioIsr to check the status of the receive FIFO. It returns 0 if the FIFO is full; 1 if the received character was successfully added to the FIFO.
rFifoPutChar	This function is called from sioIsr to add a character to the receive FIFO. It returns 0 if the receive FIFO is full; 1 if the character was successfully added to the FIFO.
rFifoGetChar	This function is called from getChar_ to get the oldest character from the receive FIFO. It returns 0 if the receive FIFO is empty; 1 if a character is removed from the FIFO.

IMPORTANT NOTE
The transmit and receive FIFOs are manipulated from both the interrupt level and the "task" (non-interrupt) level. For this reason, it is very important to ensure that UART interrupts are disabled while characters are added to and removed from the FIFOs at the task level.

7.1.4 MCE Initialization

The files MceBoot.c MceBoot.h contain functions and definitions to initialize the MCE using data that has been programmed to EEPROM by the MCEDesigner tool. It assumes that the automatic boot process has copied the MCE code from EEPROM to shared RAM and an "MCE Info" structure from EEPROM to a fixed location in 8051 program RAM.

The function StartMce first copies the "MCE Info" structure from 8051 program RAM to a location in data RAM and verifies the validation field in the structure. If the validation field is incorrect, the entire structure is assumed to be invalid and the MCE is not initialized. Otherwise, the MCE Info structure provides the starting load address in RAM and the MCE execution address. The StartMce function uses this information to zero the MCE data area preceding the start of the MCE program. The function doMceBoot is called to initialize the MCE special registers and begin MCE execution.

7.1.5 Motor Control

The files MotorCtrl.c and MotorCtrl.h contain a simple example of motor drive configuration andcontrol. The main function MotorCtrl reads character commands from the serial port using the functions provided by the UART driver. You can use a Hyperterminal (or equivalent) connection to send commands and read responses. All commands control motor 1 only.

The sample code treats the motor as a state machine, with three states: DRIVE_IDLE, DRIVE_RUN and DRIVE_FAULT. The function *MotorCtrl* takes input commands from the serial port and passes valid ones to *MotorSeq*. Based on the current motor state, *MotorSeq* calls appropriate functions to implement the command or returns an error indicating that the command was invalid. If an invalid command is entered, 'Invalid Command' is returned to the HyperTerminal display. Listed below are the commands supported from the function *MotorCtrl*, with explanations of their operation.

C or c
Configure motor drive and clear faults. 'Configured' will be echoed back on the UART if successful. If the motor is running, the command is ignored and 'Invalid Command' is returned instead. See section Error! Reference source not found. above.

+
Set forward direction. 'Forward' is echoed when the operation is complete. If the motor is running or in a fault condition, the command is ignored and 'Invalid Command' is sent instead.

-
Set reverse direction. 'Reverse' is echoed when the operation is complete. If the motor is running or in a fault condition, the command is ignored and 'Invalid Command' is sent instead.

F or f
Clear fault condition. 'Fault Clear' is echoed when the operation is complete. If the drive is not in a fault condition, the command is ignored and 'Invalid Command' is sent instead.

G or g
Run motor. The motor is placed in run state and turns in the configured direction at a low speed. 'Started' is echoed when the operation is complete. If the motor is already running or in a fault condition, the command is ignored and 'Invalid Command' is sent instead.

S or s
Stop motor. The motor is stopped and 'Stopped' is echoed when the operation is complete. If the motor is already stopped or in a fault condition, the command is ignored and 'Invalid Command' is sent instead.

R or r
Set motor speed. This is a multi-character command. The command character must be followed by exactly four decimal digits (0 - 9) defining the target speed in rotor RPM. If the motor is not running the command is ignored and 'Invalid Command' is echoed. If the requested speed is out of range for the motor (according to the value of "#define Mtr_Max_Speed") then the Mtr_Max_Speed value will be used. Otherwise, the operation is performed after all four digits have been received, at which point 'Speed Set' is echoed. If a character other than a digit is received, an 'X' is echoed and the command is aborted.

?
(1) Get motor speed. This command returns the motor speed when the drive is running. The current speed is output in motor RPM. This RPM calculation relies on the parameters generated by the parameter configurator.
(2) Read FaultFlags. When the drive is in a fault state, this returns the value of the FaultFlags register. The register value is displayed in hexadecimal format.

H or h

Catch-Spin Start. This begins the catch-spin startup sequence for the motor. The system will monitor the speed and direction of the motor to determine if the motor should be stopped and reversed, or if the motor is already going in the correct direction and catch it. This startup mode is suitable for an instance where the motor may already be in motion due to outside forces (such as wind blowing a fan). This command is allowable only in the idle state, otherwise 'Invalid Command' is echoed. At the end of the sequence, 'CatchSpin Complete' is echoed.

T or t
Ramp Stop. This function will slowly ramp the motor down to zero speed. This is opposed to simply stopping the motor by halting the PWM. Upon successful stopping of the motor 'Ramp Stop Complete' will be echoed. The rate is determined by the *rampTime* variable, which is the time in seconds to ramp to zero.

Z or z
Zero Vector Brake. This function will turn on the zero vector brake command for 20 seconds, then halt the PWM and turn off zero vector brake. In the case of a fault, the function will break out the 20 second wait time and halt the PWM.

7.1.6 Other Operations

The file Timer.c contains a function that initializes timer 1 to generate interrupts at 20 millisecond intervals. A global variable "systicks" is incremented on each interrupt. Timer setup assumes the clock is running at the default rate of 64 MHz. Related definitions are provided in the file Timer.h.

The file utils.c contains utility functions to enable and disable a specified interrupt.

The file irmcx3xx.h contains definitions for all of the byte- and bit-addressable SFRs.

Execution begins at the function main in the file main.c. This function initiates each of the sample operations. The last is the UART sample, which does not return.

7.2 The Keil and FS2 Tools

It is strongly recommended that you use the following tools for 8051 software development:

- Keil Software PK51 Professional Developer's Kit (includes 8051 compiler, assembler, linker, debugger and uVision2 integrated development environment)
- First Silicon Solutions (FS2) ISA-M8051EW In-Target System Analyzer for the Mentor Graphics M8051EW Microprocessor Core. This product includes a debug pod and device driver that provides an interface between the Keil debugger and the IRMCF3xx for debugging.

This document and all 8051 software provided by IR for the IRMCx31x assumes that you are using these tools.

7.2.1 Software Installation

Install the Keil tools first, then FS2 according to the instructions provided with the installation media. When requested to select options for FS2 setup, leave all settings at the default values.

7.2.2 Software Setup

The software release for the IRMCF3xx provides several sample uVision2 projects that are set up appropriately for the IRMCx31x . Project files have the extension .uv2 (for example, SampleRegIf.uv2). When creating your own uVision2 project, it's recommended that you start with the settings in one of the sample projects.

Once you have set up a uVision project, click on the Debug tab in the project settings window (Options for Target…). Click the Use radio button in the top right-hand section of the display and select Fs2/Keil ISA-M8051EW Driver from the pull-down menu. If you don't see that option in the menu, the Fs2 driver is not installed properly.

7.2.3 The Keil Compiler

The Keil optimizing Cx51 compiler includes a number of enhancements specifically for the 8051 processor and embedded programming environments. It is recommended that you read the Cx51 Compiler User's Guide carefully.

The following tips may ease your software development effort:

- While debugging, use a fairly low level of compiler optimization, such as level 3. Using high optimization levels can give unexpected results in the debugger (breakpoints not hit when expected, for example). When your code is working properly, increase the optimization level and verify operation again.
- The compiler supports several types of pointers. Avoid using generic (three-byte) pointers, which produce larger and slower code and can be confusing when viewing memory locations with the debugger.
- The compiler supports several memory models. Due to the extensive external RAM included in the IRMCx31x , the large model should be used.
- The compiler supports special "sfr" and "sbit" keywords for accessing byte- and bit-addressable SFRs. See the IR sample file IRMCX3XX .h for SFR definitions using these keywords. There are many examples of their use in the source code.
- The special "interrupt" keyword must be used when defining an interrupt service routine. The functions Timer1 in SysCtrlSeq.c, sioIsr in asyncDriver.c are examples.
- The 8051 has very little stack space and the compiler does not normally store function parameters and local variables on the stack. For this reason, the special "reentrant" keyword must be used to define functions that may be reentrant or called recursively.
- Refer to the functions in the files Coherent.SRC and SfrRegs.SRC for examples of interfacing C and assembly language.

7.2.4 Debugging

Debugging 8051 code on the IRMCF3xx requires connection of the Fs2 pod to the JTAG connector on the development board. Once the pod is connected, use the following procedure to download the 8051 code to IRMCF3xx RAM. The steps must be executed in the order shown below.

1. On the host PC, run the Keil uVision2 application.
2. Apply power to the Fs2 pod.
3. Apply power to the development board.
4. From the uVision II Debug menu, select Start/Stop Debug Session.
5. Wait for the code to be downloaded to the target processor. A progress bar is shown in the lower left corner of the uVision2 window and disappears when download is complete.
6. Set breakpoints as desired and then select Go from the Debug menu (or click the Go toolbar button).

Restrictions while debugging:

- The single step operation cannot be used unless all interrupts are disabled.
- After stopping at a breakpoint, you must disable or delete the breakpoint before continuing unless all interrupts are disabled.

When you have finished debugging, select Stop Running from the Debug menu (or click the Stop toolbar button) and then select Start/Stop Debug Session from the Debug menu. Do not power off the development board or the Fs2 pod until you've terminated the uVision2 debug session.

7.3 Storing 8051 and MCE Code in EEPROM

When you debug your 8051 code using uVision2, the program loads the code into RAM each time you start a debug session. When you power off the development board, the code stored in RAM is lost.

You can use MCEDesigner to store your 8051 and MCE code in EEPROM. Refer to the MCEDesigner User's Guide for complete instructions.

Once you've programmed the 8051 and MCE code to EEPROM, the IRMCF3xx automatically loads both the 8051 and MCE code into RAM at power-up and reset. The IRMCF3xx boot process executes the 8051 code automatically after it loads the 8051 and MCE code into RAM. It does not execute the MCE code; the MCE remains in a stopped state while the 8051 begins executing. If you're using MCEDesigner (running the MCEDesigner agent code on the 8051), the agent starts MCE execution during its initialization, before it begins to communicate with MCEDesigner. If you're not using the MCEDesigner agent, your custom 8051 code is responsible for starting MCE execution.

International
IOR Rectifier

IR WORLD HEADQUARTERS: 233 Kansas St., El Segundo, California 90245, Tel: (310) 252-7105
Data and specifications subject to change without notice. 11/17/2006

www.irf.com