Freescale Semiconductor
Application Note

AN2988
Rev. 1.2, 11/2005

Using the MC3PHAC Motor Controller

by David Wilson
Motion Products Specialist
Freescale Semiconductor

The MC3PHAC is a single-chip intelligent controller designed specifically to meet the requirements for low-cost, variable-speed, 3-phase ac motor control systems. Target applications include:

- Low horsepower HVAC
- Home appliances
- Commercial laundry and dishwashers
- Process control
- Pumps and fans

Table of Contents

1 Introduction

One of the unique aspects of this device is that although it is adaptive and configurable based on its environment, it does not require any software development, which often translates into extra tools, expertise, and longer design cycles. This makes the MC3PHAC a perfect fit for applications requiring ac motor control but with limited software resources available, a tight schedule, or both.

An example configuration of an ac motor drive using the MC3PHAC is shown in Figure 1.

freescale™ semiconductor

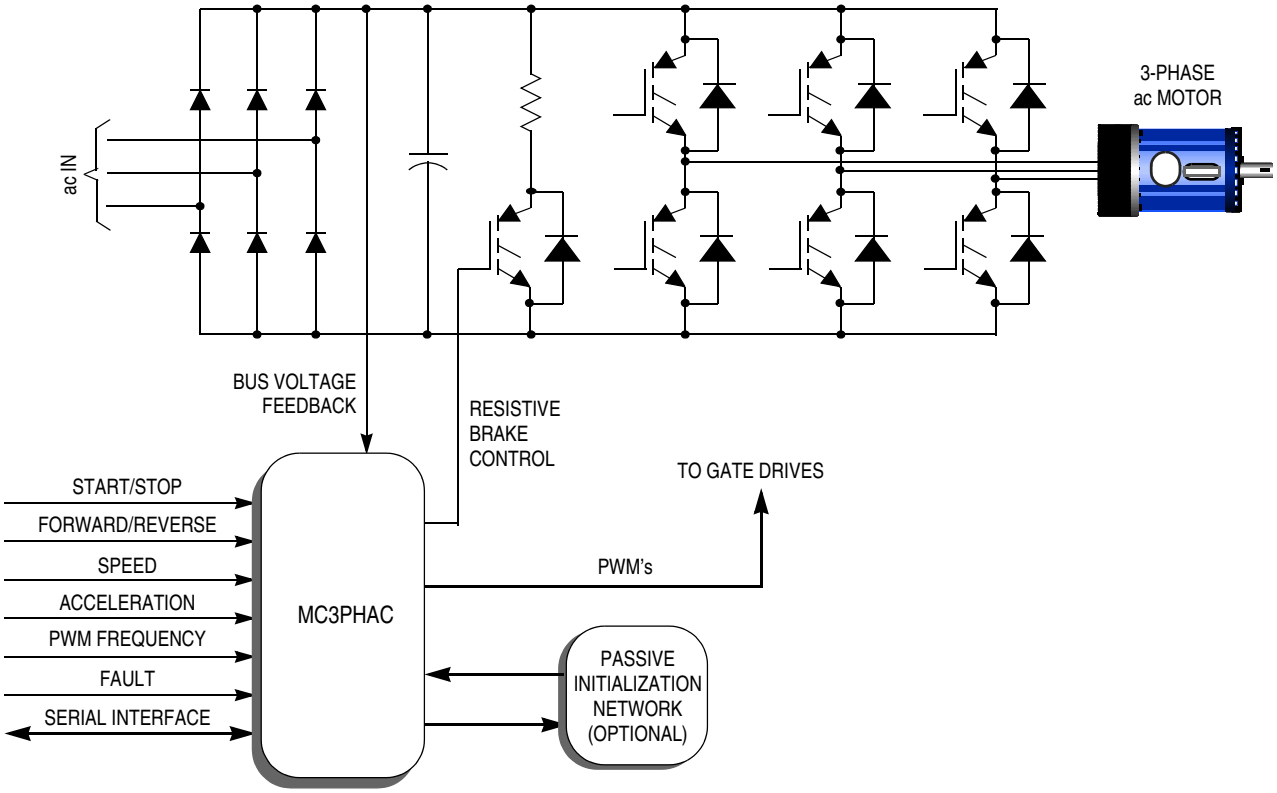


Figure 1. MC3PHAC Based Motor Control System

2 MC3PHAC Description

2.1 MC3PHAC Pinout

The MC3PHAC is available in a 28-pin PDIP, 28-pin SOIC, and 32-pin QFP, as shown in Figure 2. A description of each pin is detailed in Table 1.

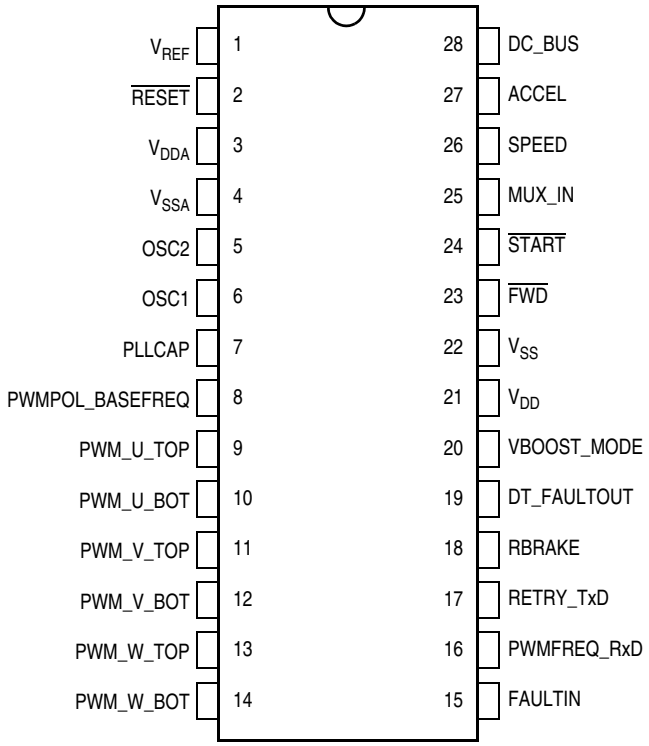


Figure 2. MC3PHAC Pin Connections for PDIP and SOIC

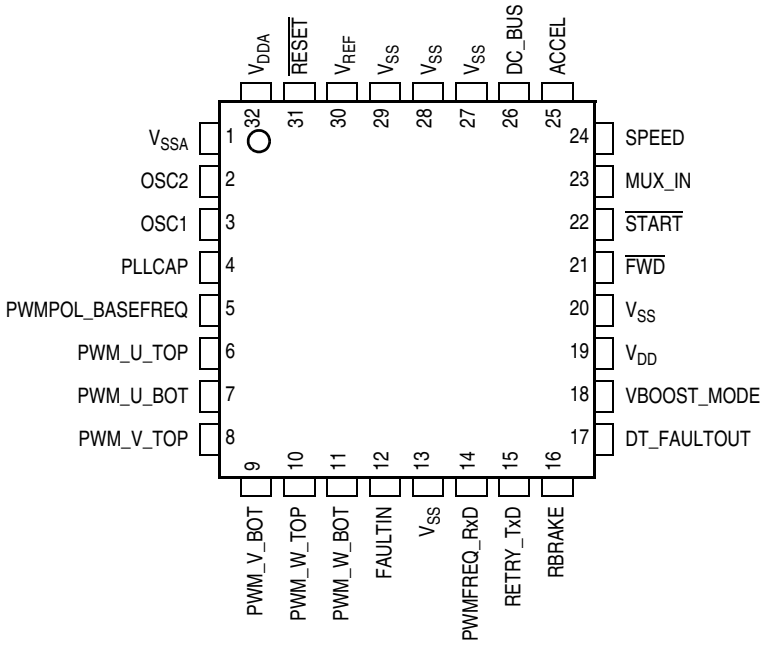


Figure 3. MC3PHAC Pin Connections for QFP

Table 1. MC3PHAC Pin Descriptions (Sheet 1 of 2)

Pin Number	Pin Name	Pin Function
1	V_{REF}	ADC V_{REF} High — Reference voltage input for the on board ADC. For best signal to noise performance, this pin should be tied to V_{DDA}.
2	$\overline{\text{RESET}}$	A logic low on this pin forces the MC3PHAC to its initial startup state. All PWM outputs are placed in a high impedance mode. $\overline{\text{RESET}}$ is a bidirectional pin, allowing a reset of the entire system. It is driven low when an internal reset source is asserted (e.g., loss of clock).
3	V_{DDA}	V_{DD} (Analog) — Provides power for the analog portions of the MC3PHAC, which include the clock generation circuit (PLL), and the ADC.
4	V_{SSA}	V_{SS} (Analog) — Return power for the analog portions of the MC3PHAC, which include the clock generation circuit (PLL), and the ADC.
5	OSC2	Osc Out — Oscillator output used as part of a crystal or ceramic resonator clock circuit. **NOTE**: Correct timing of the MC3PHAC is based on a 4.00 MHz oscillator.
6	OSC1	Osc In — Oscillator input used as part of a crystal or ceramic resonator clock circuit. Can also accept a signal from an external canned oscillator. **NOTE**: Correct timing of the MC3PHAC is based on a 4.00 MHz oscillator.
7	PLLCAP	PLL Damp — A capacitor from this pin to ground affects the stability and reaction time of the PLL clock circuit. Smaller values result in faster tracking of the reference frequency. Larger values result in better stability. A value of 0.1 μF is typical.
8	PWMPOL_ BASEFREQ	PWM Pol/Base Speed — Input which is sampled at specific moments during initialization to determine the PWM polarity and the base speed (50 or 60 Hz).
9	PWM_U_TOP	PWM U Top — PWM output signal for the top transistor driving motor phase U.
10	PWM_U_BOT	PWM U Bottom — PWM output signal for the bottom transistor driving motor phase U.
11	PWM_V_TOP	PWM V Top — PWM output signal for the top transistor driving motor phase V.
12	PWM_U_BOT	PWM V Bottom — PWM output signal for the bottom transistor driving motor phase V.
13	PWM_W_TOP	PWM W Top — PWM output signal for the top transistor driving motor phase W.
14	PWM_W_BOT	PWM W Bottom — PWM output signal for the bottom transistor driving motor phase W.
15	FAULTIN	Fault In — A logic high on this input will immediately disable the PWM outputs. A retry timeout interval will be initiated after this pin returns to a logic low state.
16	PWMFREQ_RxD	PWM Freq/Rx Data — In standalone mode, this pin is an output that drives low to indicate the MUX_IN pin is reading an analog voltage to specify the desired PWM frequency. In PC master mode, this pin is an input that receives UART serial data. The UART module on HC08 products is called the serial communication interface (SCI module).
17	RETRY_Tx	Retry Time/Tx Data — In standalone mode, this pin is an output that drives low to indicate the MUX_IN pin is reading an analog voltage to specify the time to wait after a fault before re-enabling the PWM outputs. In PC master mode, this pin is an output that transmits UART serial data.
18	RBRAKE	R Brake — This pin is an output that is driven to a logic high whenever the voltage on the DC_BUS pin exceeds a preset level, indicating a high bus voltage. This signal is intended to connect a resistor across the dc bus capacitor to prevent excess capacitor voltage.
19	DT_FAULTOUT	Dead-Time/Fault Out — In standalone mode, this pin is an output that drives low to indicate the MUX_IN pin is reading an analog voltage to specify the dead-time between the on states of the top and bottom PWM signals for a given motor phase. In PC master mode, this pin is an output that goes low whenever a fault condition occurs.

Table 1. MC3PHAC Pin Descriptions (Sheet 2 of 2)

Pin Number	Pin Name	Pin Function
20	VBOOST_MODE	Vboost/Mode Select — At startup, this pin is an input is sampled to determine whether to enter standalone mode (logic high), or PC master mode (logic low). In standalone mode, this pin is also used as an output that drives low to indicate the MUX_IN pin is reading an analog voltage to specify the amount of voltage boost to apply to the motor.
21	V_{DD}	+5 V power to the MC3PHAC
22	V_{SS}	+5 V return for the MC3PHAC
23	$\overline{FWD}$	Forward/Reverse — This pin is an input that is sampled to determine whether the motor should rotate in the forward or reverse direction.
24	$\overline{START}$	Start/Stop — This pin is an input that is sampled to determine whether the motor should be running or not.
25	MUX_IN	Parameter Mux In — In standalone mode, during initialization, this pin is an output that is used to determine PWM polarity and base speed. Otherwise, it is an analog input used to read several voltage levels that specify MC3PHAC operating parameters.
26	SPEED	Speed In — In standalone mode, during initialization, this pin is an output that is used to determine PWM polarity and base speed. Otherwise, it is an analog input used to read a voltage level corresponding to the desired steady-state speed of the motor.
27	ACCEL	Acceleration In — In standalone mode, during initialization, this pin is an output that is used to determine PWM polarity and base speed. Otherwise, it is an analog input used to read a voltage level corresponding to the desired acceleration of the motor.
28	DC_BUS	dc Bus In — In standalone mode, during initialization, this pin is an output that is used to determine PWM polarity and base speed. Otherwise, it is an analog input used to read a voltage level proportional to the dc bus voltage.

2.2 MC3PHAC Features

Three Phase Waveform Generation — The MC3PHAC generates six PWM signals that have been modulated with variable voltage and variable frequency information to control a three-phase ac motor. A third harmonic signal has been superimposed on top of the fundamental motor frequency to achieve full bus use. This results in a 15% increase in maximum output amplitude, compared to pure sinewave modulation.

The waveform is updated at a 5.3 kHz rate (except when the PWM frequency is 15.9 kHz), resulting in “near continuous” waveform quality. At 15.9 kHz, the waveform is updated at 4.0 kHz.

DSP Filtering — A 24 bit IIR digital filter is used on the SPEED signal in standalone mode, resulting in enhanced speed stability in noisy environments. The sampling period of the filter is 3 ms (except when the PWM frequency is 15.9 kHz), and it mimics the response of a single pole analog filter having a pole at 0.4 Hz. At a PWM frequency of 15.9 kHz, the sampling period is 4 ms and the pole is located at 0.3 Hz.

A complete description of the filter can be found in the Software Functionality section later in this document.

High Precision Calculations — Up to 32-bit variable resolution is employed for precision control and smooth performance. For example, the motor speed can be controlled with a resolution of 0.004 Hz.

Smooth Voltage Transitions — When the commanded speed of the motor passes through ± 1 Hz, the voltage is gently applied or removed depending on the direction of the speed change. This eliminates any pops or surges that may occur, especially under conditions of high voltage boost at low frequencies.

High Side Bootstrapping — Many motor drive topologies (especially high voltage drives) use optocouplers to supply the PWM signal to the high side transistors. Often, the high side transistor drive circuitry contains a charge pump circuit to create a floating power supply for each high side transistor that is dependent on low side PWMs to develop power. When the motor has been off for a period of time, the charge on the high side power supply capacitor is depleted, and must be replenished before proper PWM operation can resume.

To accommodate such topologies, the MC3PHAC will always provide 100 ms of 50% PWMs to only the low side transistors each time the motor is turned on. Because the top transistors remain off during this time, it has the effect of applying zero volts to the motor, and no motion occurs. After this period, motor waveform modulation begins, with PWMs also being applied to the high side transistors.

Fast Velocity Updating — During periods when the motor speed is changing, the rate at which the velocity is updated is critical to smooth operation. If these updates occur too infrequently, a ratcheting effect will be exhibited on the motor, which inhibits smooth torque performance. However, velocity profiling is a very calculation intensive operation to perform, which runs contrary to the above requirement.

The MC3PHAC uses a velocity pipelining technique that allows linear interpolation of the velocity values, resulting in a new velocity value every 189 μs (252 μs for 15.9 kHz PWMs). The net result is ultra smooth velocity transition, where each velocity step is not perceived by the motor.

Dynamic Bus Ripple Cancellation — The dc bus voltage is sensed by the MC3PHAC, and any deviations from a predetermined norm (3.5 V on the DC_BUS pin) result in corrections to the PWM values to counteract the effect of the bus voltage changes on the motor current. The frequency of this calculation is sufficiently high to permit compensation for line frequency ripple, as well as slower bus voltage changes resulting from regeneration or brown out conditions.

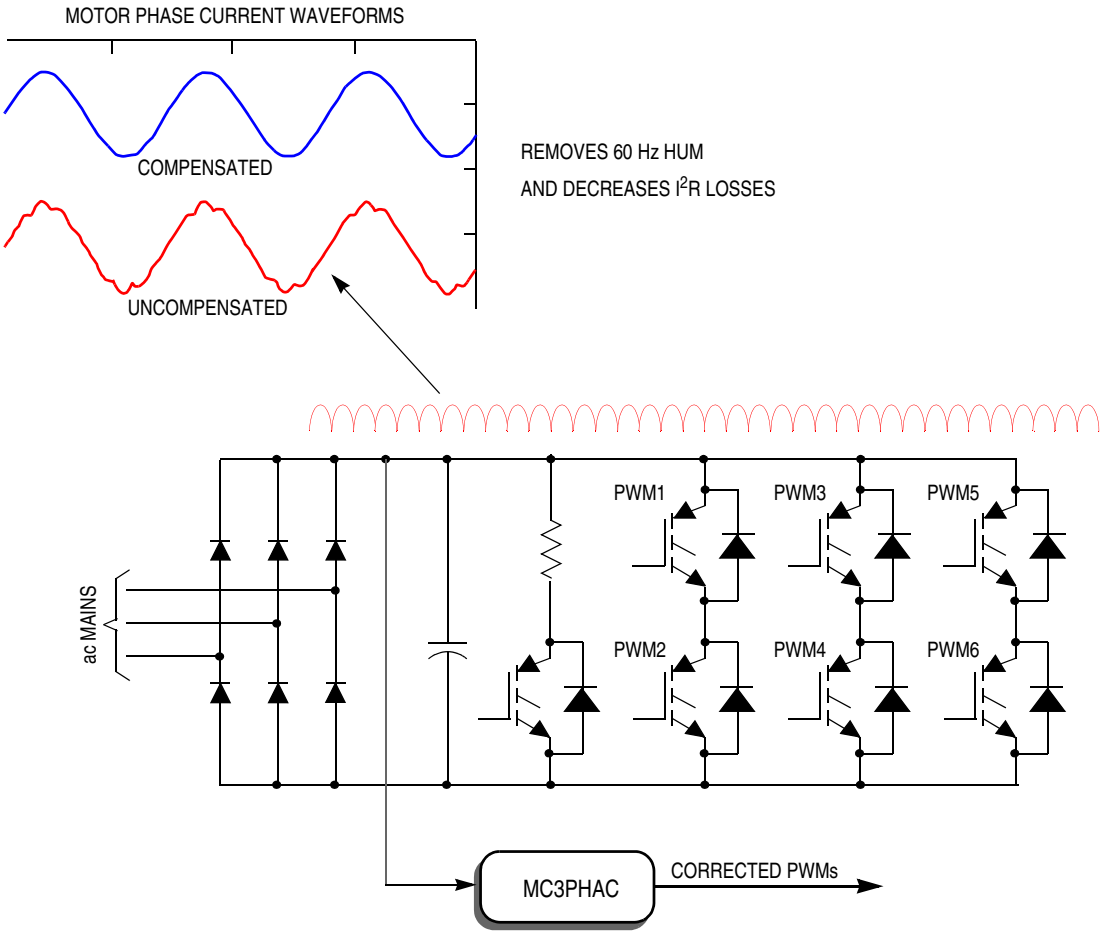


Figure 4. Dynamic Bus Ripple Cancellation

Selectable Base Speed — ac motors are designed to accept rated voltage at either 50 or 60 Hz. The voltage rate depends on the region where the motors are designed to be used. The MC3PHAC can accommodate both types of motors by allowing the voltage profile to reach maximum value at either 50 or 60 Hz. This parameter can be specified at initialization in standalone mode, or can be changed at any time in PC master mode.

Selectable PWM Polarity — The polarity of the PWM outputs may be specified such that a logic high on a PWM output can be either the asserted or negated state of the signal. In standalone mode, this parameter is specified at initialization, and applies to all six PWM outputs. In PC master mode, the polarity of the top PWM signals can be specified separately from the polarity of the bottom PWM signals. This specification can be made at any time, but after it is done, the polarities are locked and cannot be changed until a reset occurs. Also, any commands from FreeMASTER software that would have the effect of enabling PWMs are prevented by the MC3PHAC until the polarity has been specified.

In standalone mode, the base speed and PWM polarity are specified at the same time during initialization by connecting pin 25, 26, 27, or 28 exclusively to the PWMPOL_BASEFREQ input. During initialization, pins 25, 26, 27, and 28 are pulsed one at a time to determine which one has been connected to the PWMPOL_BASEFREQ input. Table 2 shows the selected PWM polarity and base speed as a function of which pin connection is made. It is not necessary to break this connection after the initialization phase has been completed. The MC3PHAC will function properly while this connection is in place.

Table 2. PWM Polarity and Base Speed Specification in Standalone Mode

Pin Connected to PWMPOL_BASEFREQ Pin	PWM Polarity	Base Speed
MUX_IN	Logic low = on	50 Hz
SPEED	Logic high = on	50 Hz
ACCEL	Logic low = on	60 Hz
DC_BUS	Logic high = on	60 Hz

Selectable PWM Frequency — The MC3PHAC accommodates four discrete PWM frequencies that can be changed on the fly while the motor is running. In standalone mode, the PWM frequency is specified by applying a voltage to the MUX_IN pin while the PWMFREQ_RxD pin is being driven low. Table 3 shows the required voltage levels on the MUX_IN pin, and the associated PWM frequency for each voltage range. The listed voltage ranges are based on 5 V being applied to the V_{REF} pin. The PWM frequencies are based on a 4.00 MHz frequency on the oscillator input.

Table 3. MUX_IN Voltages and Corresponding PWM Frequencies

Voltage Input	PWM Frequency
0 to 1 V	5.291 kHz
1.5 to 2.25 V	10.582 kHz
2.75 to 3.5 V	15.873 kHz
4 to 5 V	21.164 kHz

Selectable PWM Dead-time – Besides being able to specify the PWM frequency, the blanking time interval between the “on” states of complementary PWM pairs can also be specified. In standalone mode, this is accomplished by supplying a voltage to the MUX_IN pin while the DT_FAULTOUT pin is being driven low. In this way, dead-time can be specified with a scaling factor of 2.075 µs per volt (assuming V_{REF} is 5 V), with a minimum value of 0.5 µs. In PC master mode, this value can be selected to be anywhere between 0 and 32 µs.

In both standalone and PC master modes, the dead-time value can be written only once. Further updates of this parameter are locked out until a reset condition occurs.

Speed Control – The synchronous motor frequency can be specified in real time to be anywhere from 1 Hz to 128 Hz, by the voltage applied to the SPEED pin. The scaling factor is 25.6 Hz per volt (assuming ADC V_{REF} high is 5 V). This parameter can also be controlled directly from FreeMASTER software in real time.

As mentioned earlier, the SPEED pin is processed by a 24-bit digital filter to enhance the speed stability in noisy environments. This filter is activated only in standalone mode.

Acceleration Control – The motor acceleration can be specified in real time to be anywhere from 0.5 Hz/sec, all the way up to 128 Hz/sec, by the voltage applied to the ACCEL pin. The scaling factor is 25.6 Hz/sec per volt (assuming ADC V_{REF} high is 5 V). This parameter can also be controlled directly from FreeMASTER software in real time.

Voltage Profile Generation – The MC3PHAC controls the motor voltage in proportion to the specified frequency, as indicated in Figure 5.

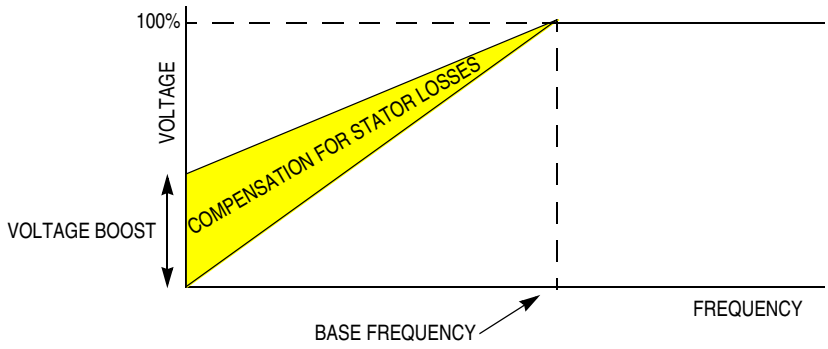


Figure 5. Voltage Profiling, Including Voltage Boost

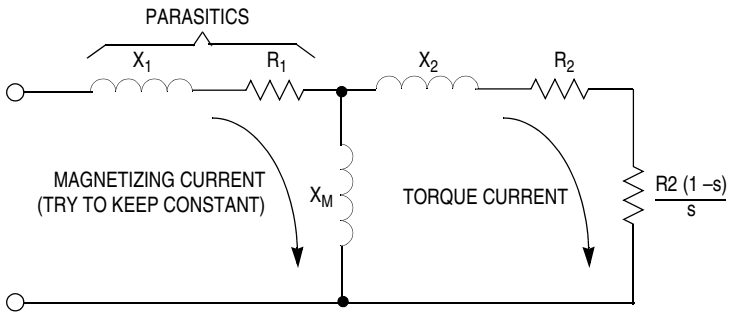


Figure 6. ac Motor Single Phase Model Showing Parasitic Stator Impedances

An ac motor is designed to draw a specified amount of magnetizing current when supplied with rated voltage at the base speed frequency. As the frequency decreases, assuming no stator losses, the voltage must decrease in exact proportion to maintain the required magnetizing current. In reality, as the frequency decreases, the voltage drop in the series stator resistance increases in proportion to the voltage across the magnetizing inductance. This has the effect of further reducing the voltage across the magnetizing inductor, and consequently, the magnetizing current. A schematic representation of this effect is illustrated in Figure 6. To compensate for this voltage loss, the voltage profile is boosted over the normal voltage curve shown in Figure 5, so that the magnetizing current remains constant over the speed range.

The MC3PHAC allows the voltage boost to be specified as a percentage of full voltage at 0 Hz, as shown in Figure 5. In standalone mode, voltage boost is specified during the initialization phase by supplying a voltage to the MUX_IN pin while the VBOOST_MODE pin is being driven low. In this way, voltage boost can be specified from 0% to 40%, with a scaling factor of 8% per volt (assuming ADC V_{REF} high is 5 V). In PC master mode, the voltage boost can be specified from 0% to 100%, and can be changed at anytime.

By using the voltage boost value, and the specified base speed frequency, the MC3PHAC has all the information required to generate a voltage profile automatically based on the generated waveform frequency. An additional feature exists in PC master mode where this voltage value can be overridden and controlled in real time. Specifying a voltage lower than the normal volts-per-Hz profile permits a softer torque response in certain ergonomic situations. It also allows for load power factor control and higher operating efficiencies with high inertia loads, or other loads where instantaneous changes in torque demand are not permitted. Details of this feature are discussed in the FreeMASTER Software Operation section of this document.

PLL Clock Generation — The OSC1 pin signal is used as a reference clock for an internal phase locked loop clocking circuit, which is used to drive the internal clocks of the MC3PHAC. This provides excellent protection against noise spikes that may occur on the OSC1 pin. In a clocking circuit that does not incorporate a PLL, a noise spike on the clock input can create a clock edge that violates the setup times of the clocking logic, and can cause the device to malfunction. The same noise spike applied to the input of a PLL clock circuit is perceived by the PLL as a change in its reference frequency, and the PLL output frequency begins to change in an attempt to lock on to the new frequency. However, before any appreciable change can occur, the spike is gone, and the PLL settles back in to the true reference frequency.

Fault Protection — The MC3PHAC supports an elaborate range of fault protection and prevention features. If a fault does occur, the MC3PHAC immediately disables PWMs and waits until the fault condition is cleared before starting a timer to re-enable the PWMs. In standalone mode, this timeout interval is specified during the initialization phase by supplying a voltage to the MUX_IN pin while the RETRY_Tx pin is being driven low. In this way, the retry time can be specified from 1 second to 60 seconds, with a scaling factor of 12 seconds per volt (assuming ADC V_{REF} high is 5 V). In PC master mode, the retry time can be specified from 0.25 second to over 4.5 hours, and can be changed at anytime.

The fault protection and prevention features are listed below:

- **External Fault Monitoring —** The FAULTIN pin accepts a digital signal that indicates a fault has been detected via external monitoring circuitry. A high level on this input results in the PWMs being immediately disabled. Typical fault conditions might be a dc bus overvoltage, bus overcurrent, or over-temperature. After this input returns to a logic low level, the fault retry timer begins running, and PWMs are re-enabled after the programmed timeout value is reached.
- **Lost Clock Protection —** If the signal on the OSC1 pin is lost altogether, the MC3PHAC will immediately disable the PWM outputs to protect the motor and power electronics. This is a special fault condition in that it will also cause the MC3PHAC to be reset. Lost clock detection is an important safety consideration, as many safety regulatory agencies are now requiring a dead crystal test be performed as part of the certification process.
- **Low V_{DD} Protection —** Whenever V_{DD} falls below 4 V, an on-board power supply monitor will reset the MC3PHAC. This allows the MC3PHAC to work properly with 5 V supplies of either 5% or 10% tolerance.

- **Bus Voltage Integrity Monitoring —** The DC_BUS pin is monitored at a 5.3 kHz frequency (4.0 kHz when the PWM frequency is set to 15.9 kHz), and any voltage reading outside of an acceptable window constitutes a fault condition. In standalone mode, the window thresholds are fixed at 4.47 V (128% of nominal), and 1.75 V (50% of nominal), where nominal is defined to be 3.5 V (assuming ADC V_{REF} high is 5 V). In PC master mode, both top and bottom window thresholds can be set independently to any value between 0 volts (0% of nominal), and greater than 5 volts (143% of nominal), and can be changed at any time. After the DC_BUS signal level returns to a value within the acceptable window, the fault retry timer begins running and PWMs are re-enabled after the programmed timeout value is reached.

  During power-up, it is possible that V_{DD} could reach operating voltage before the dc bus capacitor charges up to its nominal value. When the dc bus integrity is checked, an undervoltage would be detected and treated as a fault, with its associated timeout period. To prevent this, the MC3PHAC monitors the dc bus voltage during power-up in standalone mode, and waits until it is higher than the undervoltage threshold before continuing. During this time, all MC3PHAC functions are suspended. After this threshold is reached, the MC3PHAC will continue normally, with any further undervoltage conditions treated as a fault.

  If dc bus voltage monitoring is not desired, a voltage of 3.5 volts ± 5% should be supplied to the DC_BUS pin through an impedance of between 4.7 kΩ and 15 kΩ.
- **Regeneration control** — Regeneration is a process by which stored mechanical energy in the motor and load are transferred back into the drive electronics, usually as a result of an aggressive deceleration operation. In special cases where this process occurs frequently (e.g., elevator motor control systems), it is economical to incorporate special features in the motor drive to allow this energy to be supplied back to the ac mains. However, for most low-cost ac drives, this energy is stored in the dc bus capacitor by increasing its voltage. If this process is not checked, the dc bus voltage can rise to dangerous levels, which can destroy the bus capacitor or the transistors in the power inverter.

  The MC3PHAC incorporates two techniques to deal with regeneration before it becomes a problem:

  — **Resistive braking** — The DC_BUS pin is monitored at a 5.3 kHz frequency (4.0 kHz when the PWM frequency is set to 15.9 kHz), and when the voltage reaches a certain threshold, the RBRAKE pin is driven high. This signal can be used to connect a resistor across the dc bus capacitor, so that mechanical energy from the motor is dissipated as heat in the resistor, versus being stored as voltage on the capacitor. In standalone mode, the DC_BUS threshold required to assert the RBRAKE signal is fixed at 3.85 V (110% of nominal), where nominal is defined to be 3.5 V (assuming ADC V_{REF} high is 5 V). In PC master mode, this threshold can be set to any value between 0 V (0% of nominal), and greater than 5 V (143% of nominal), and can be changed at any time.

  — **Automatic Deceleration Control** — When decelerating the motor, the MC3PHAC attempts to use the specified acceleration value for deceleration as well. If the voltage on the DC_BUS pin reaches a certain threshold, the MC3PHAC begins to moderate the deceleration as a function of this voltage, as shown in Figure 7. The voltage range on the DC_BUS pin from when the deceleration begins to decrease, to when it reaches 0, is 0.62 V (assuming ADC V_{REF} high is 5 V). In standalone mode, the DC_BUS voltage where deceleration begins to decrease is fixed at 3.85 V (110% of nominal), where nominal is defined to be 3.5 V

(assuming ADC V_{REF} high is 5 V). In PC master mode, this threshold can be set to any value between 0 V (0% of nominal), and greater than 5 V (143% of nominal), and can be changed at any time.

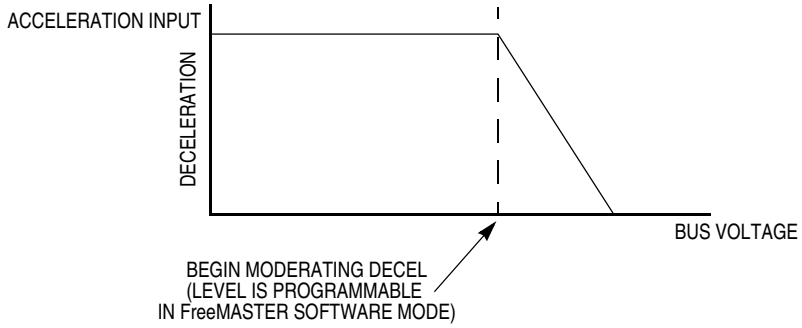


Figure 7. Deceleration as a Function of Bus Voltage

2.3 FreeMASTER Software Operation

2.3.1 Introduction to FreeMASTER Software

The MC3PHAC is compatible with Freescale's PC master serial interface protocol (FreeMASTER). Communication occurs over an on-board UART at 9600 baud to an external master device, which may be a microcontroller that also has an integrated UART, or a personal computer via a COM port. With FreeMASTER software, an external controller can monitor and control all aspects of the MC3PHAC operation.

The most popular master implementation is a PC, where a GUI interface has been layered on top of the FreeMASTER command protocol, complete with a graphical data display, and ActiveX control functions, also available from Freescale. It is beyond the scope of this document to describe the FreeMASTER protocol, or its implementation on a personal computer.

2.3.2 FreeMASTER Software Operation with the MC3PHAC

When power is first applied to the MC3PHAC, or if a logic low level is applied to the $\overline{\text{RESET}}$ pin, the MC3PHAC enters PC master mode if the VBOOST_MODE pin is low during the initialization phase. The MC3PHAC recognizes a subset of the FreeMASTER command set, which is listed in Table 4.

Table 4. Recognized FreeMASTER Commands

Command	Description
GETINFOBRIEF	MC3PHAC responds with brief summary of hardware setup and link configuration info.
READVAR8	MC3PHAC reads an 8-bit variable at a specified address, and responds with its value
READVAR16	MC3PHAC reads a 16-bit variable at a specified address, and responds with its value
READVAR32	MC3PHAC reads a 32-bit variable at a specified address, and responds with its value
WRITEVAR8	MC3PHAC writes an 8-bit variable at a specified address
WRITEVAR16	MC3PHAC writes a 16-bit variable at a specified address

With the READVARx commands, the addresses are checked for validity, and the command is executed only if the address is within proper limits. In general, a read command with an address value below $0060 or above $FE03 will not execute properly, but instead will return an “invalid operation” response. The addresses for the WRITEVARx commands are also checked for validity, and the data field is also limited to a valid range for each variable.

The user interface variables and their associated FreeMASTER addresses within the MC3PHAC are listed here in Table 5.

Table 5. User Interface Variables for Use with FreeMASTER Software

Name	Address	Read/Write	Size Bytes	Description	Valid Data
Commanded direction	$1000	W	1	Determines whether the motor should go forward, reverse, or stop.	Forward – $10 Reverse – $11 Stop – $20
Command reset	$1000	W	1	Forces the MC3PHAC to do an immediate reset.	$30
Commanded PWM frequency[1]	$1000	W	1	Specifies the frequency of the MC3PHAC PWM frequency.	5.3 kHz – $41 10.6 kHz – $42 15.9 kHz – $44 21.1 kHz – $48
Measured PWM period	$00A8	R	2	The modulus value supplied to the PWM generator used by the MC3PHAC. Value is multiplied by 250 ns to obtain PWM period.	$00BD - $05E8
Commanded PWM polarity[2, 3, 4]	$1000	W	1	Specifies the polarity of the MC3PHAC PWM outputs.	B+ T+ $50 B+ T – $54 B– T+ $58 B– T – $5C
Dead-time[2, 3, 4]	$0036	R/W	1	Specifies the dead-time used by the PWM generator. Dead-time = value times 125 ns. This is a write-once parameter.	$00 – $FF
Base speed[3]	$1000	W	1	Specifies the motor frequency at which full voltage is applied.	60 Hz – $60 50 Hz – $61
Acceleration[3]	$0060	R/W	2	Acceleration in Hz/sec. (8.8 format)	$0000 – $FFFF

Table 5. User Interface Variables for Use with FreeMASTER Software (continued)

Name	Address	Read/Write	Size Bytes	Description	Valid Data
Commanded motor frequency[3]	$0062	R/W	2	Commanded frequency in Hz. (8.8 format)	$0000 – $FFFF
Actual frequency	$0085	R	2	Actual frequency in Hz. (8.8 format)	$0000 – $FFFF
Status	$00C8	R	1	Status byte.	$00 – $FF
Voltage boost	$006C	R/W	1	0 Hertz voltage.	$00 – $FF
Modulation index	$0091	R	1	Voltage level (assuming no bus ripple compensation).	$00 – $FF
Maximum voltage	$0075	R/W	1	Maximum allowable modulation index value.	$00 – $FF
Bus voltage[5]	$0079	R	2	dc bus voltage reading.	$000 – 3FF
Fault timeout	$006A	R/W	2	Specifies the delay time after a fault condition before re-enabling the motor.	$0000 – $FFFF
Fault timer	$006D	R	2	Real time display of the fault timer.	$0000 – $FFFF
V_{Bus} decel value	$00C9	R/W	2	V_{Bus} readings above this value result in reduced deceleration.	$0000 – $FFFF
V_{Bus} RBRAKE value	$0064	R/W	2	V_{Bus} readings above this value result in the RBRAKE pin being asserted.	$0000 – $FFFF
V_{Bus} brownout value	$0066	R/W	2	V_{Bus} readings below this value result in an undervoltage fault.	$0000 – $FFFF
V_{Bus} overvoltage value	$0068	R/W	2	V_{Bus} readings above this value result in an overvoltage fault.	$0000 – $FFFF
Speed in ADC value[5]	$0095	R	2	Left justified 10 bit ADC reading of the SPEED pin.	$0000 – $FFC0
Setup	$00AE	R	1	Bit field indicating which setup parameters have been initialized before motion is permitted.	$E0 – $FF
Switch in	$0001	R	1	Bit field indicating the current state of the Start/Stop and Forward/Reverse switches.	$00 – $FF
Reset status[6]	$FE01	R	1	Indicates Cause of the last reset	$00 – $FF
Version	$EE00	R	4	MC3PHAC version	ASCII Field

NOTES:
1 The commanded PWM frequency cannot be written until the PWM outputs exit the high impedance state. The default PWM frequency is 15.873 kHz.
2 The PWM output pins remain in a high impedance state until this parameter is specified.
3 This parameter must be specified before motor motion can be initiated by the MC3PHAC.
4 This is a write-once parameter. The first write to this address will execute normally. Further attempts at writing this parameter will result in an illegal operation response from the MC3PHAC.
5 The value of this parameter is not valid until the PWM outputs exit the high impedance state.
6 The data in this field is only valid for one read. Further reads will return a value of $00.

Each bit variable listed in Table 5 is defined below:

Address: $00C8

	7	6	5	4	3	2	1	0
R		SPEED CHANGING	FORWARD MOTION	MOTOR ENERGIZED	RESISTIVE BRAKE	EXTERNAL FAULT TRIP	OVER VOLTAGE TRIP	UNDER VOLTAGE TRIP
W								
Reset	U	0	1	0	0	U	0	0

= Unimplemented or Reserved U = Unaffected

Figure 1. Status Register

Address: $00AE

	7	6	5	4	3	2	1	0
R				BASE FREQUENCY SET	SPEED SET	ACCELERATION SET	POLARITY SET	DEAD TIME SET
W								
Reset	1	1	1	0	0	0	0	0

= Unimplemented or Reserved

Figure 2. Setup Register

Address: $0001

	7	6	5	4	3	2	1	0
R		$\overline{\text{START}}$/ STOP	$\overline{\text{FWD}}$/ REVERSE		FAULT OUT	RESISTOR BRAKE		
W								
Reset	U	U	U	U	U	0	U	U

= Unimplemented or Reserved U = Unaffected

Figure 3. Switch In Register

Address: $FE01

	7	6	5	4	3	2	1	0
R	POWER UP	RESET PIN	MC3PHAC FUNCTIONAL FAULT	MC3PHAC FUNCTIONAL FAULT	PC MASTER SOFTWARE RESET COMMAND		LOW V_{DD} VOLTAGE	
W								
Reset	1	0	0	0	0	0	0	0

= Unimplemented or Reserved

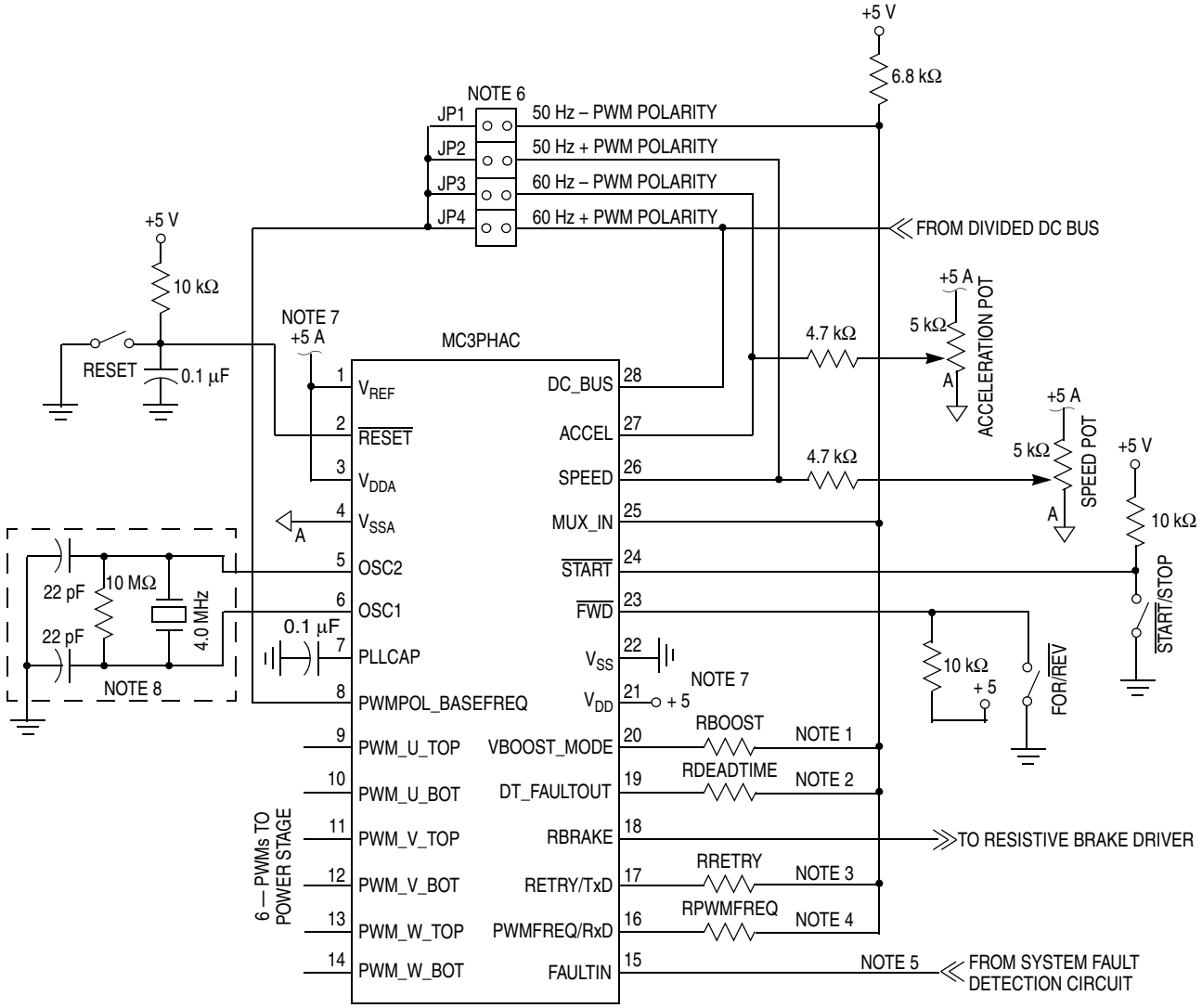
Figure 4. Reset Status Register

2.4 Stand-Alone Operation

If the VBOOST_MODE pin is high when the MC3PHAC is powered up, or after a reset, the MC3PHAC enters standalone mode. In this mode of operation, the functionality of many of the MC3PHAC pins change so that the device can control a motor without requiring setup information from an external master. By contrast, the MC3PHAC will drive certain pins corresponding to parameters that must be specified, while simultaneously monitoring the response on other pins.

In many cases, the parameter to be specified is represented as an analog voltage presented to the MUX_IN pin while certain other pins are driven low. In so doing, the MC3PHAC can accommodate an external analog mux that will switch various signals at the MUX_IN pinwhen the signal select line goes low. All signals must be in a range between 0 V and ADC V_{REF} high. As an economical alternative, an external passive network can be connected to each of the parameter select output pins and to the MUX_IN pin, as shown in Figure 8.

The equivalent impedance of this passive network as seen by the MUX_IN pin is very important, and should be in the range of 5 kΩ to 10 kΩ. If the resistance is too high, leakage current from the I/O pins will cause an offset voltage that will affect the accuracy of the reading. If the resistance is too low, the parameter select pins will not be able to sink the required current for an accurate reading. Assuming a pull-up resistor value of 6.8 kΩ (as indicated in Figure 8), the resulting value for each parameter as a function of the corresponding pull-down resistor value is shown in Figure 9, Figure 10, and Figure 11.



Notes:
1. See Figure 11.
2. See Figure 9.
3. See Figure 10.
4. See Table 3.
5. If no external fault circuit is provided, connect to V_{SS}.
6. Connect only one jumper.
7. Use bypass capacitors placed close to the MC3PHAC.
8. Consult crystal/resonator manufacturer for component values.

Figure 8. Standalone MC3PHAC Configuration

The $\overline{\text{START}}$ input pin is debounced internally so that a switch can be directly accommodated on this pin. The input is level sensitive, but a logic 1 level must exist on the pin before a logic 0 level will be processed as a start signal. This will prevent an accidental motor startup in the event of the MC3PHAC being powered up while the switch remains in the start position.

The $\overline{\text{FWD}}$ input pin is also debounced internally, and can also directly accommodate a switch connection. The input is also level sensitive.

Figure 8 also shows the jumper arrangement connected to the PWMPOL_BASEFREQ input pin. For proper operation, one and only one jumper connection can be made at any given time. Table 2 shows the polarity and base speed selections as a function of the jumper connection made.

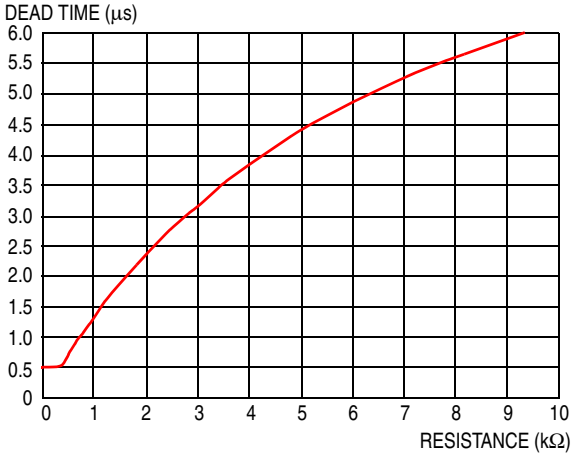


Figure 9. Dead-Time as a Function of the Pull-Down Resistor

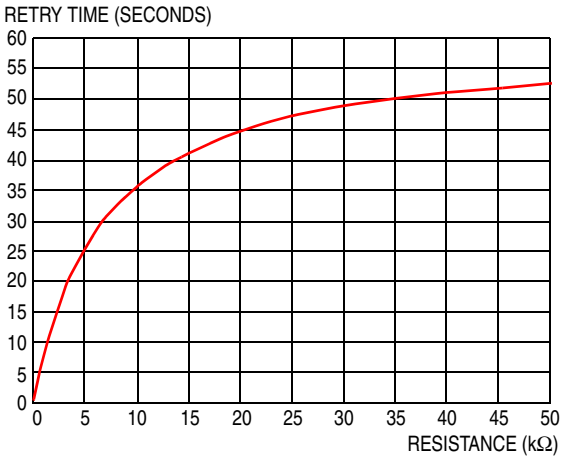


Figure 10. Retry Time as a Function of the Pull-Down Resistor

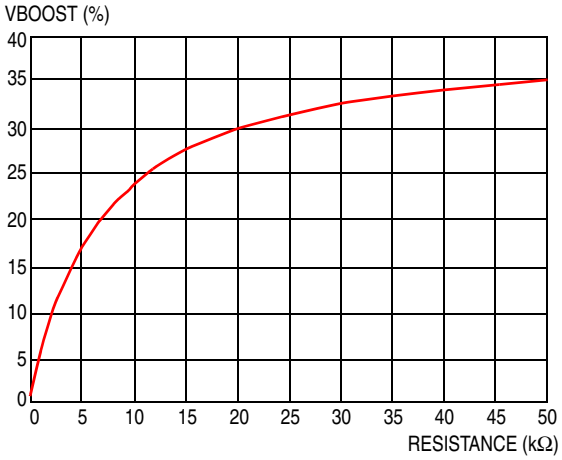


Figure 11. Voltage Boost as a Function of the Pull-Down Resistor

3 Software Functionality

3.1 Software Overview

The software for the MC3PHAC can be partitioned into routines comprising the background tasks and interrupt service routines. The functional interaction between the background tasks is shown in Figure 12. The interrupt service routines are listed in Figure 13.

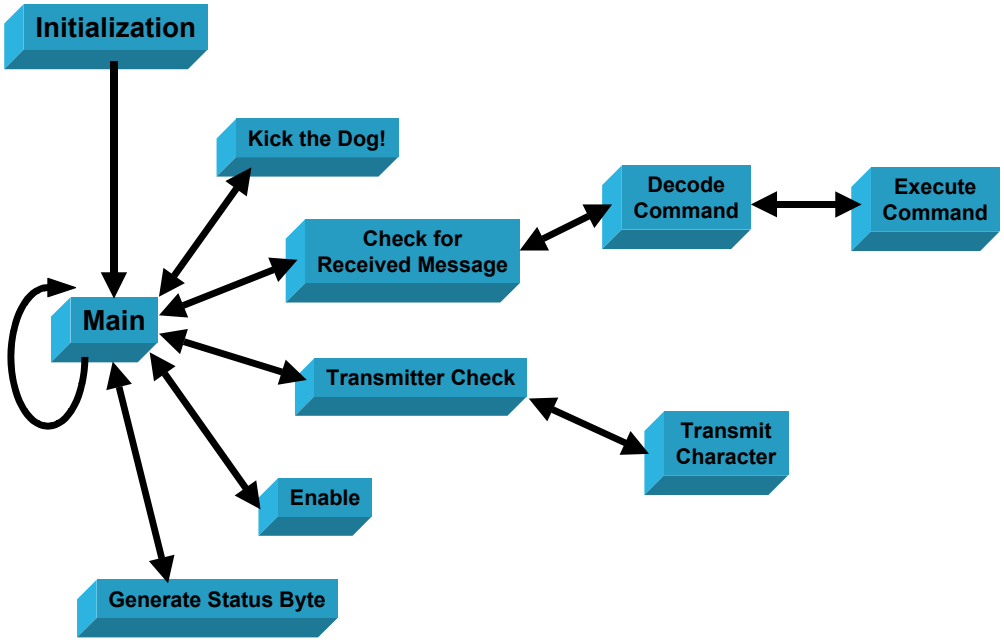


Figure 12. Background Tasks

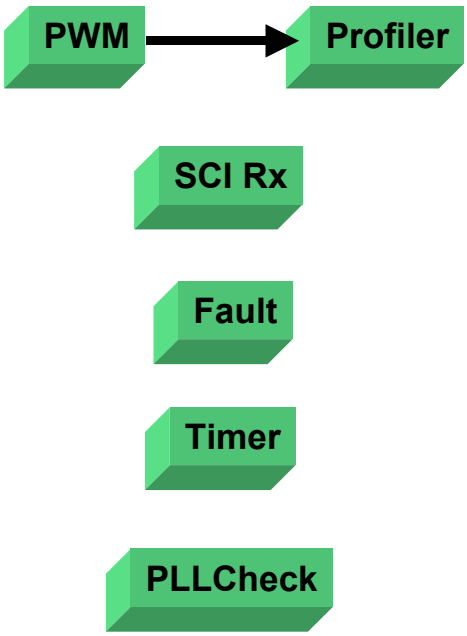


Figure 13. Interrupt Service Routines

3.2 Software Modules

The software is comprised of 18 separate files that must be assembled together to generate an executable file. Each of these files and its functions are listed below.

Motor.asm — This is the top-level file responsible for including all the other files in the application. When generating an executable version of the code, this is the file that must be assembled.

Mr4io.h — Includes the equates that define all register names and addresses in the MC3PHAC.

Zpage.h — Includes all variable declarations not associated with the FreeMASTER code. All variables reside in zero page of memory.

Pclink.h — Includes variable and label declarations for the FreeMASTER command processing code.

Pcsciisr.h — Includes variable and label declarations for the SCI transmitter and receiver code.

3rd_Harm.asm — Contains one cycle of the motor voltage waveform which consists of 512 entries, each entry an 8-bit value. The waveform is made up of a fundamental sinewave, with a 3rd harmonic sinewave superimposed on top of it.

Init.asm — This is the first executable code out of reset responsible for the following functions:

- Configuring the PLL to drive the bus at 8 MHz.
- Clearing all used RAM locations in zero page.
- Initializing the stack pointer to $011F (the last RAM location).
- Initializing all used peripherals and variables.
- Determine the operating mode (standalone or PC master). If PC master mode is selected, initialize the SCI and specific FreeMASTER variables. If standalone mode is selected, using a passive external network, determine PWM polarity, base speed, dead-time, voltage boost, fault retry time, and initialize the PWM module.
- The PWM module initialization code is also part of Init.asm.

Main.asm — This is the top level background loop responsible for invoking other background tasks. These tasks include:

- Servicing the watchdog
- Checking whether a valid FreeMASTER message has been received, and invoking the FreeMASTER decode routine.
- Checking whether the SCI transmitter is empty when sending a FreeMASTER response, so that the next character can be sent.
- Calling the enable routine, which is the next file to be discussed.
- Creating a status byte for use by FreeMASTER software.

Enable.asm – Responsible for determining whether the motor should be on or off, and which direction it should go. A block diagram of the enable routine is shown in Figure 14. As can be seen, the following tasks are performed:

- Debouncing of the $\overline{\text{START}}$ and $\overline{\text{FWD}}$ input pins.
- In the event of a fault, monitor the FAULTIN and DC_BUS pins to determine if normal motor operation is possible yet. If so, start the fault timer, and re-energize the motor at the end of the fault timeout period.
- Monitor the DC_BUS pin to turn off the RBRAKE pin if the dc bus voltage is below the RBRAKE threshold.
- Proper state sequencing to energize the motor.

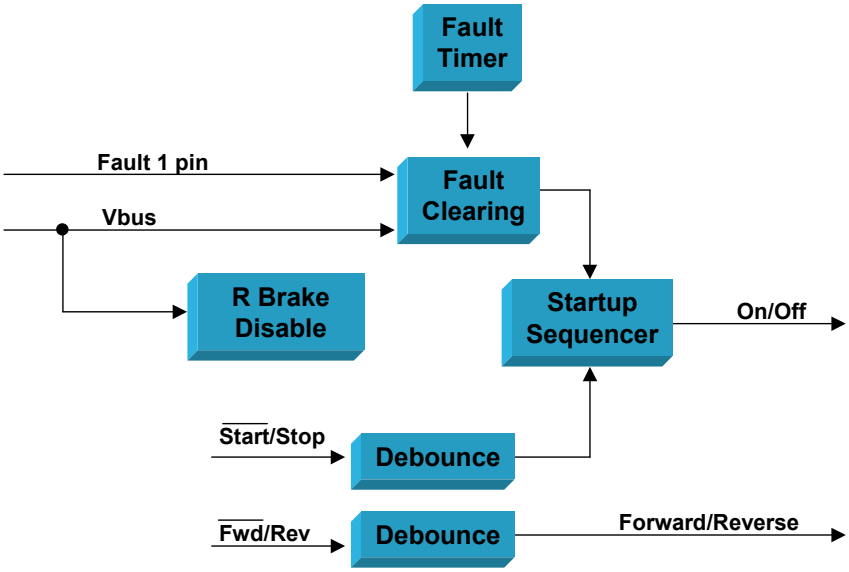


Figure 14. Enable.asm Functional Block Diagram

Profiler.asm – Processes the on/off and forward/reverse signals from Enable.asm and generates a velocity profile based upon the commanded speed and acceleration inputs. The dc bus voltage is also monitored, and used to reduce the deceleration rate if the voltage is too high. After the velocity is calculated, it is used to determine the proper voltage to apply to the motor based on the low-frequency voltage boost, the base speed, and the maximum allowable voltage as specified by FreeMASTER software. This voltage information is output to PWM.asm, along with the velocity information.

Even though the profiler runs with interrupts disabled, it is not considered a background task because it is invoked periodically from PWM.asm, which is an interrupt service routine. Also, the profiler uses a pipelined velocity calculation technique, which will be discussed later in this document. A block diagram of the profiler is shown in Figure 15.

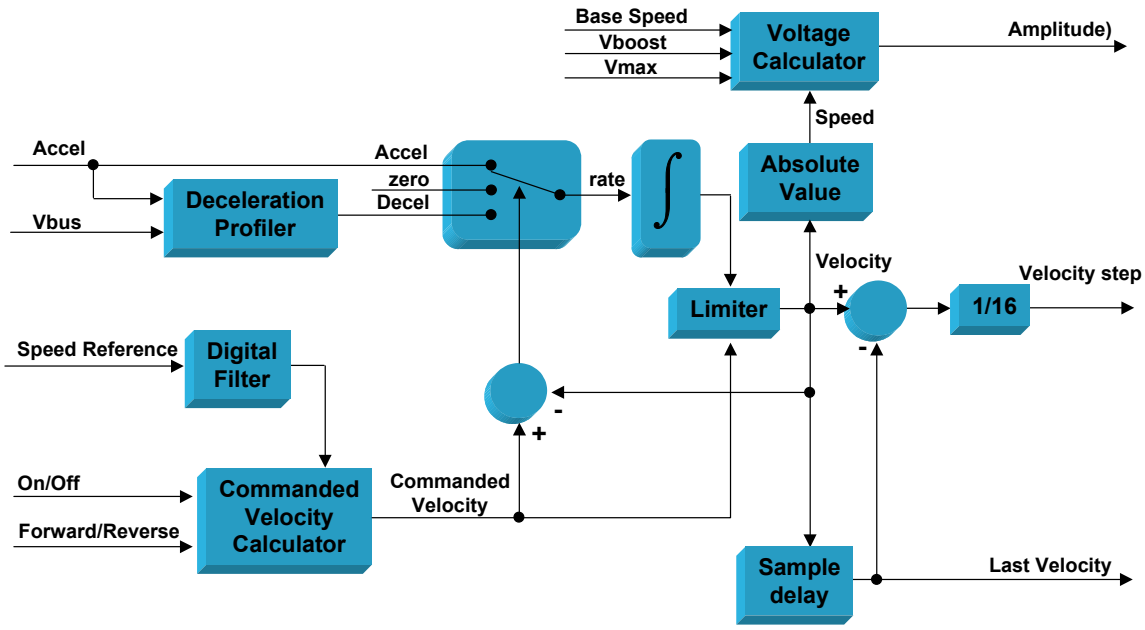


Figure 15. Profiler.asm Functional Block Diagram

PWM.asm – A block diagram of PWM.asm is shown in Figure 16. The main function of PWM.asm is to take the velocity and voltage information from the profiler, and turn this information into PWM values, which when supplied to the PWM module, result in waveforms of the proper frequency and amplitude on the motor. The bus voltage is also monitored, and if it is not within proper limits, a fault is generated. Also, the RBRAKE pin is asserted if the bus voltage exceeds the limit established for turning on the resistive brake. Finally, the bus voltage signal is used to modulate the PWM signal amplitude, so that any ripple or deviation from an established nominal voltage will be compensated.

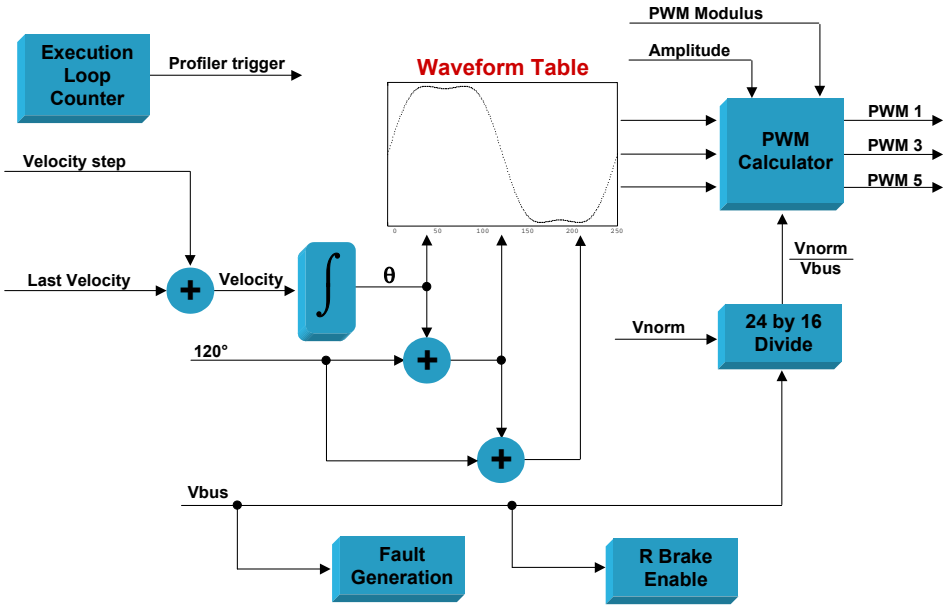


Figure 16. Functional Block Diagram of PWM.asm

Another task of PWM.asm is to launch the profiler at appropriate intervals. PWM.asm is a re-entrant interrupt service routine which is invoked by the PWM module reload interrupt.

Fault.asm — This file contains an interrupt service routine that is invoked by the assertion of the FAULTIN pin. It resets the fault timeout variable, and if PC master mode has been selected, it drives the DT_FAULTOUT pin low.

Pllcheck.asm — This is the interrupt service routine that is invoked whenever the PLL loses lock as a result of losing the crystal reference input, or other problems with the PLL. Unlike other faults in the motor control system that may occur as a result of unanticipated system stresses, a loss of lock on the PLL indicates a hardware problem. Therefore, PLLcheck.asm causes a reset of the MC3PHAC instead of simply waiting for a fault timeout period to resume normal operation. This is accomplished by forcing a jump to an illegal address in the memory map.

Timer.asm — This interrupt service routine is executed whenever timer B overflows. Assuming 8 MHz bus operation, this occurs every 0.262 seconds. The only function of Timer.asm is to increment a variable counter used for timeouts when a fault occurs.

Pclink.asm — This routine interprets and executes the commands sent from an external master via the FreeMASTER protocol. Table 4 lists all the FreeMASTER commands which are recognized and processed by this routine.

Pcsciisr.asm — This file contains two distinct routines related to the SCI. One is an interrupt service routine that is invoked whenever a character is received by the SCI. When a complete message is received, this routine inspects the length of the message and the checksum, and either validates the received message or builds an error response to be sent back to the master. This routine runs with interrupts enabled to allow the PWM.asm ISR to run when requested.

The other routine is a driver that manages character transmissions via the SCI back to the master. As each character is sent, its binary value is accumulated with the other characters in the message to generate a checksum at the end of the message.

No_int.asm — This routine is responsible for handling all errant interrupts that may occur, which do not have an interrupt handler associated with them. This file is primarily intended to be a debug tool to be assembled with the application during software development.

Vectors.asm — All interrupt service routine start addresses are defined in the vector table created by this file. There is no executable code in this module.

3.3 Standalone Initialization

If the MC3PHAC is connected as shown if Figure 8, then the VBOOST_MODE pin will be high out of reset because all I/O pins will be in a high impedance state. The initialization routine will interpret this as an indication to proceed in standalone mode, and all operational parameters will be determined via the following sequence of steps:

1. Port A bits 0 through 3 are configured as outputs, and driven high. The code waits for 200 µs to allow capacitors connected to these pins to charge up.
2. Port A bit 0 is then driven low, and the code waits for 200 µs to allow a capacitor connected to this pin to discharge.

3. The IRQ1F flag is checked to determine whether the IRQ1 pin has gone low. If so, the code interprets this as an indication that the PWM polarity should be negative, and the base speed is 50 Hz.
4. If the IRQ1F flag indicates that the IRQ1 pin hasn't gone low, port A bit 1 is driven low, and the code waits for 200 μs to allow a capacitor connected to this pin to discharge.
5. The IRQ1F flag is again checked to determine whether the IRQ1 pin has gone low. If so, the code interprets this as an indication to use positive PWM polarities, and the base speed should be 50 Hz.
6. If the IRQ1F flag indicates that the IRQ1 pin hasn't yet gone low, port A bit 2 is driven low, and the code waits for 200 μs to allow a capacitor connected to this pin to discharge.
7. The IRQ1F flag is again checked to determine whether the IRQ1 pin has gone low. If so, the code interprets this as an indication to use negative PWM polarities, and the base speed should be 60 Hz.
8. If the IRQ1F flag indicates that the IRQ1 pin hasn't yet gone low, the code interprets this as an indication to use positive PWM polarities, and the base speed should be 60 Hz.
9. Interrupts from IRQ1 are disabled henceforth, and all port A pins are reconfigured as inputs.
10. The ADC is configured for right justified, continuous conversion on the ATD0 pin.
11. Port B bits 2 and 3 are configured as outputs and are driven low. All other port B bits remain as inputs. Port B bit 2 is the RBRAKE output, and bit 3 is used for the dead-time determination.
12. The code waits for 2 ms to allow a capacitor on the ATD0 input to settle in to the correct voltage.
13. The ATD0 reading is scaled to a value that can be loaded into the dead-time register. Before it is loaded, the value is compared to $04 (0.5 μs for an 8 MHz bus). If it is less than this value, $04 is substituted for the value.
14. Port B bits 2 and 4 are configured as outputs, and driven low. All other port B bits are configured as inputs. Port B bit 4 is used for the voltage boost determination.
15. The code waits for 2 ms to allow a capacitor on the ATD0 input to settle in to the correct voltage. A reading is then taken on the ATD0 input, which is scaled to a value that can be used to specify the 0 Hz voltage boost value.
16. The ADC is reconfigured for left-justified, continuous conversions on the ATD0 pin.
17. Port B bits 1 and 2 are configured as outputs, and driven low. All other port B bits are configured as inputs. Port B bit 1 is used to determine the retry time after a fault has occurred.
18. The code waits for 2 ms to allow a capacitor on the ATD0 input to settle in to the correct voltage. A reading is then taken on the ATD0 input, which is scaled to a value that can be used to specify the retry time. If the value is less than $04 (retry time of 1.05 seconds for an 8 MHz bus), then $04 is substituted for the value.
19. Port B bits 0 and 2 are configured as outputs, and driven low. All other port B bits are configured as inputs. Port B bit 0 is used to determine the PWM frequency. Because this is a real time parameter, port B bit 0 remains low henceforth.
20. The ADC is reconfigured for right-justified, continuous conversions on the ATD0 pin.
21. The code waits for 2 ms to allow a capacitor on the ATD0 input to settle in to the correct voltage.
22. The PWM module is then initialized and enabled using the specified polarity information. Interrupts are enabled for the PWM module, as well as for the fault 1 input pin.
23. Before exiting the initialization routine, PLL interrupts are enabled, and global interrupts are also turned on.

3.4 FreeMASTER Message Processing

The process by which the MC3PHAC communicates over FreeMASTER software is illustrated in Figure 17. When a message is received over the SCI, it is placed in an internal RAM buffer and the checksum is inspected. Assuming the checksum is valid, the message is decoded to determine whether it represents a valid operation. If so, the operation is carried out, and a response is generated back to the master indicating this. If it is not a valid operation, or if the checksum of the received message is invalid, a response is also generated to indicate the problem, but no operation is performed. After the response is transmitted, which includes any requested data, the MC3PHAC returns to the idle state. An operation may be deemed invalid for any of the following reasons:

- Command is not recognized. All recognized commands are listed in Table 4.
- The command is recognized, but there is a problem with the operational parameters. For example, trying to read or write a location that is not permitted will result in an invalid operation.
- The command is recognized, and the operation parameters are correct, but command precedent overrides the operation. For example, sending a command to turn on the motor when certain parameters have not yet been specified (e.g., PWM polarity) will result in an invalid operation.

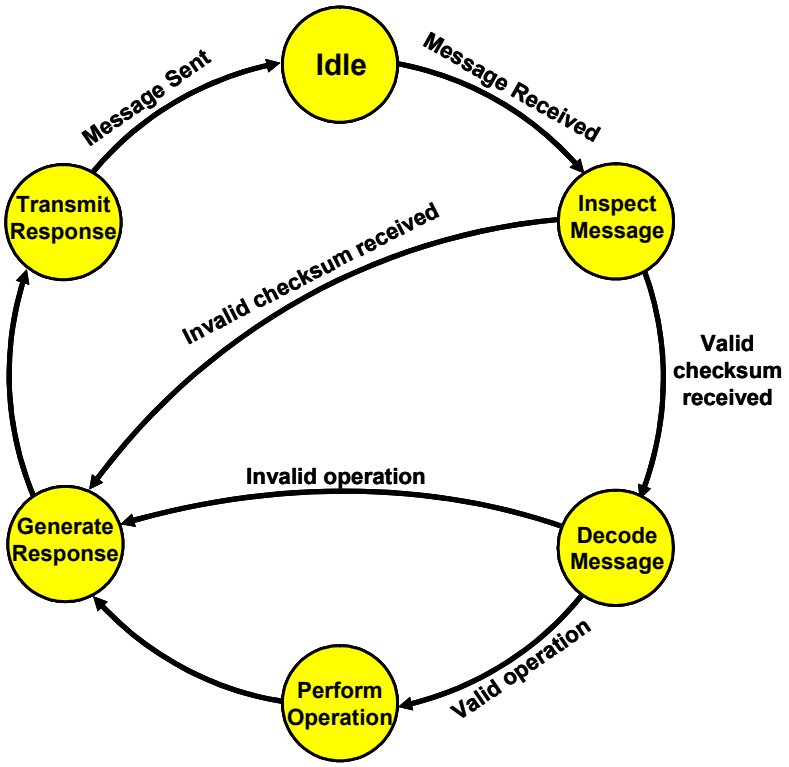


Figure 17. MC3PHAC FreeMASTER Functionality

3.4.1 SOM Characters

Every message in FreeMASTER software is predicated by a special character called the start of message (SOM) character, which is the ascii “+” character. In a character stream, anytime a + character is followed by a different character, it indicates that a new message follows and the receiver should resynchronize its state machine accordingly. Because the command field immediately follows the SOM field, a FreeMASTER command can never have a value of +. However, from time to time, another field in the message could happen to equate to the SOM character. To prevent a FreeMASTER receiver from erroneously syncing up its state machine to these fields, it is stipulated that any + character that occurs which is not in the SOM field must be duplicated by the transmitter. When two back-to-back + characters are received, the FreeMASTER receiver must know to discard one of the + characters and process the other one.

Figure 18 and Figure 19 represent flow diagrams of how the MC3PHAC treats SOM characters. Note that the receiver routine is interrupt driven, but the transmitter routine is called from a background task. The transmitter routine should not be interrupt driven because there are very lax timing requirements on a transmitted message, plus it frees up valuable ISR bandwidth that is needed by the waveform generator.

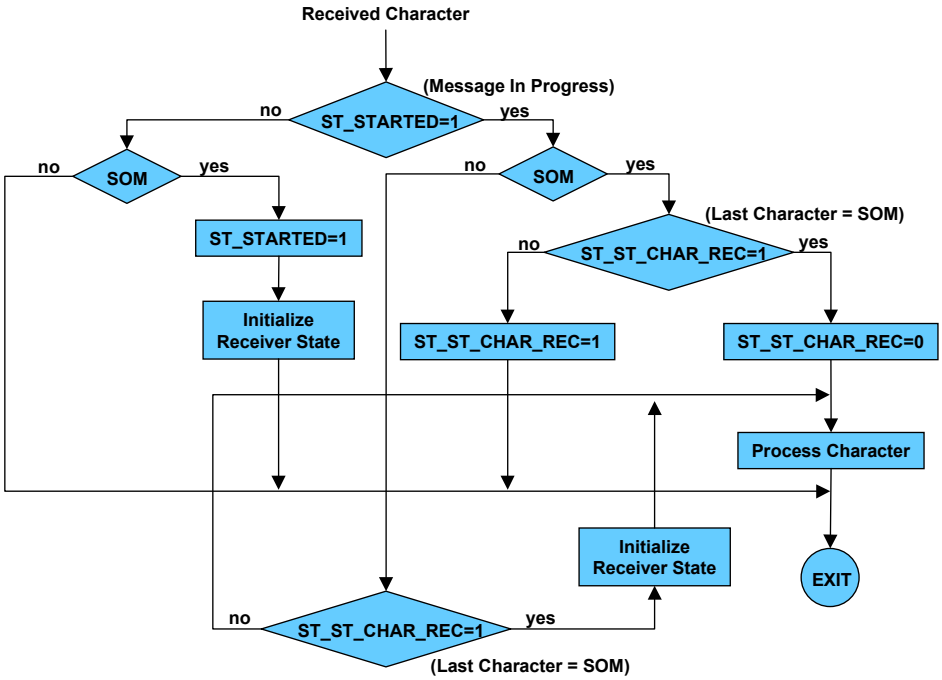


Figure 18. FreeMASTER Receiver ISR SOM Processing

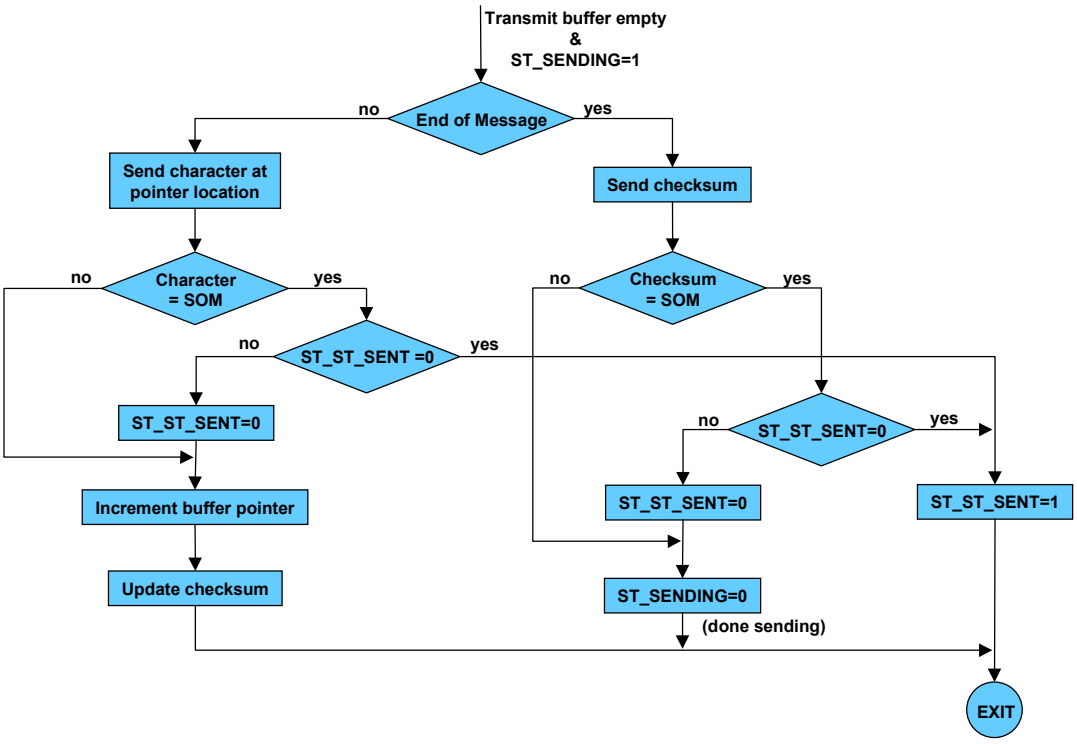


Figure 19. FreeMASTER Transmitter Routine

3.4.2 MC3PHAC Specific Commands

The MC3PHAC does not support the application command feature of the FreeMASTER protocol. Instead, all permissible MC3PHAC-specific commands are specified as the byte written to location $1000, which is an unused location in the MC3PHAC memory map. Each byte is broken down into an operation (upper nibble), and associated data (lower nibble). Table 6 shows all MC3PHAC specific commands, with their associated bit values.

Table 6. MC3PHAC Specific Commands

Command	Value	Bit Pattern
Forward	$10	**0001**xxx0
Reverse	$11	**0001**xxx1
Stop	$20	**0010**xxxx
Reset	$30	**0011**xxxx
PWM Freq. = 5.3 kHz	$41	**0100**0001
PWM Freq. = 10.6 kHz	$42	**0100**0010
PWM Freq. = 15.9 kHz	$44	**0100**0100
PWM Freq. = 21.2 kHz	$48	**0100**1000
PWM polarity T+, B+	$50	**0101**00xx
PWM polarity T–, B+	$54	**0101**01xx
PWM polarity T+, B–	$58	**0101**10xx
PWM polarity T–, B–	$5C	**0101**11xx
Base speed = 60 Hz	$60	**0110**xxx0
Base speed = 50 Hz	$61	**0110**xxx1

3.4.3 Command Precedent

The MC3PHAC incorporates a safety feature called command precedent, which prevents certain events from occurring until specific commands or parameters have been received. The PWM module will not be initialized until both PWM polarity and dead-time have been specified. These parameters must also be specified before the forward or reverse commands will be accepted, which turn on the motor. In addition, the acceleration, commanded speed, and base speed must be set before motor motion will be enabled. To accomplish this, a variable called blastoff is inspected before processing the forward or reverse commands. Whenever one of these parameters is specified, a corresponding bit in blastoff is set. When all the required bits are set, blastoff will equal $FF, which will allow the forward and reverse commands to be processed normally.

A functional equivalent of the command precedent logic is illustrated in Figure 20.

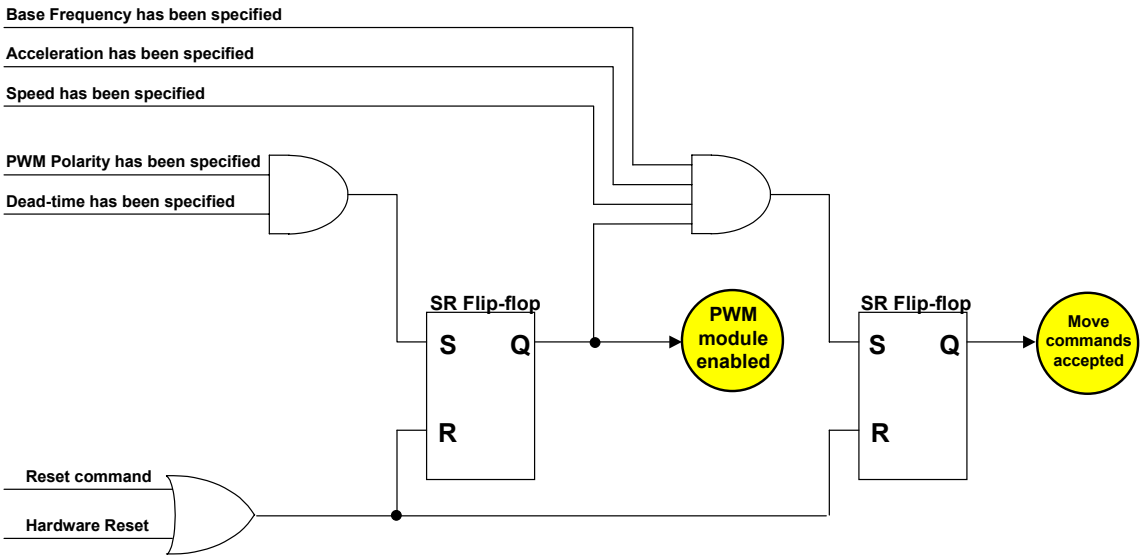


Figure 20. Command Precedent Functional Equivalent Logic

3.4.4 Data Limiting

Many parameters are inspected before being applied, and are limited to positive values if necessary. These parameters are:

- Commanded speed
- Resistive brake trip point
- Bus voltage brownout trip point
- Bus voltage over-voltage trip point
- Bus voltage deceleration trip point

Further information about these parameters may be found in Table 5.

3.5 Motor Enable and Disable Sequencing

With the exception of a fault condition, which causes the PWM outputs to be disabled immediately, the process of enabling and disabling the motor is largely the responsibility of the Enable.asm routine. Both start/stop, and forward/reverse switch inputs are processed by this routine to control the motor.

3.5.1 Switch Debouncing

Both switches are debounced using the same algorithm, as shown in Figure 21. This technique has the advantage of being level sensitive so that the debounced output can never get out of sync with the input, and it responds quickly to the changing input, even before the bouncing has stopped.

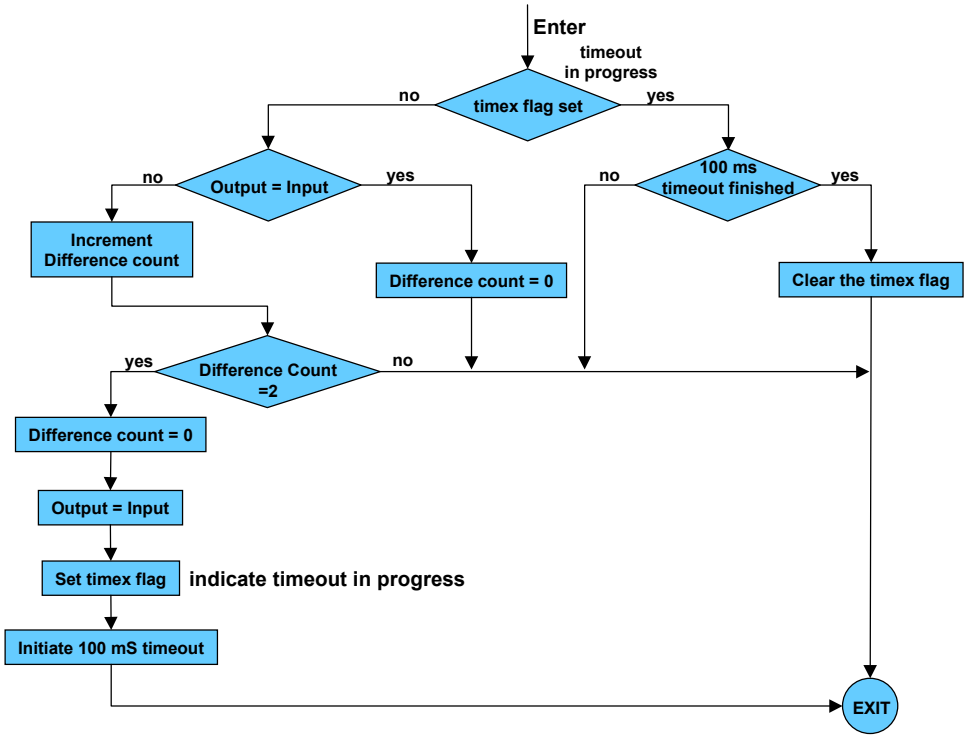


Figure 21. Switch Debouncing Algorithm

When the routine is entered, assuming that a debounce timeout period is not in progress, the switch input signal is compared with the debounced output. If they are different for two consecutive passes through the algorithm, then the debounced output is immediately set to the switch input value. Further comparisons between the output and input are locked out for a period of 100 ms, which is enough time to allow the switch to stop bouncing.

3.5.2 Motor Enable and Disable Sequencer

The debounced output of the start/stop switch is supplied to the motor enable/disable sequencer, as indicated in Figure 14. The block diagram for this routine is shown in Figure 22, and may appear more involved than expected. It incorporates several important features that are listed below:

- Start position lockout at power-up — It would be dangerous for the MC3PHAC to be able to accidentally energize the motor if the MC3PHAC were powered up while the start/stop switch was in the start position. To protect against this dangerous scenario, the MC3PHAC checks that the start/stop switch has been in the stop position at some point in time since power-up before the motor can be energized.
- High-side bootstrap — Many IPMs (integrated power modules) require that the high side gate drivers be bootstrapped off of PWMs applied to the low side transistors for a period of time. When the MC3PHAC transitions the motor from the off to the on state, it assumes that the high side gate drivers need to be charged first. It does this by applying 100 ms of 50% PWMs to only the bottom transistors before enabling the top transistors and beginning the velocity profile.
- When turning the motor off (commanded speed = 0), the routine waits until the modulation index is 0 before disabling the PWMs. This allows the profiler to gently remove the motor voltage for speeds under 1 Hz, so there is no voltage discontinuity seen at the motor. Such a discontinuity could result in a pop on the motor, especially in high voltage boost scenarios.

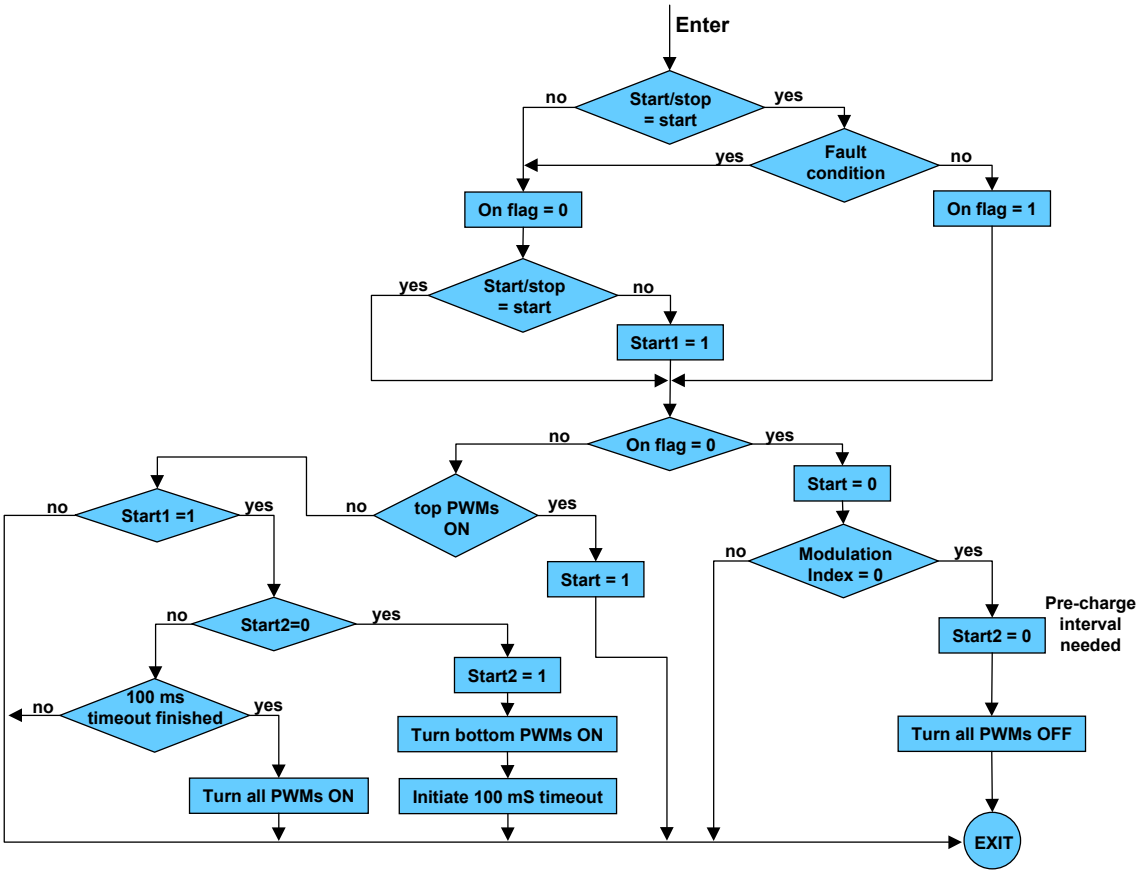


Figure 22. Motor Enable and Disable Sequencer

Referring to Figure 22, assuming no fault conditions, if the debounced $\overline{\text{START}}$ signal is in the start state, the on flag is set. If PC master mode is selected, the debounced $\overline{\text{START}}$ signal is overridden, and a separate input is supplied to this routine from the FreeMASTER software. If the Start1 flag is also set (indicating that the switch had been in the stop position at some time after power-up), then a cascade sequence is initiated, which will eventually lead to the motor being energized. Part of this sequence is controlled by the Start2 flag. If cleared, this flag indicates that the high side gate drivers must be charged before enabling the motor. The final output of this routine is the Start flag, which is used by the profiler to set the command speed to zero or a user specified value.

Before leaving our discussion of the Enable.asm routine, note that the tasks of turning off the RBRAKE pin and clearing fault conditions also belong to this routine. Turning off the RBRAKE pin quickly is not nearly as critical as turning it on quickly when it is needed. Therefore, the job of sampling the bus voltage and turning on the resistor brake when needed is relegated to the PWM ISR, which executes at a worst case frequency of 4.0 kHz.

3.6 Motor Frequency and Voltage Profiling

3.6.1 Velocity Calculation

The MC3PHAC generates a linear velocity profile. In other words, it ramps the motor speed by a constant value of Hz/sec/sec. Because the velocity profile is linear, and acceleration is the derivative of velocity, this implies that the acceleration is either 0 (when the speed isn't changing), or constant (when the speed is changing). If more complex profiling is desired, the acceleration variable must be profiled in real time external to the MC3PHAC, either through the real time acceleration input (pin 27), or via FreeMASTER software.

The velocity variables used in the profiler are signed, where negative values of velocity correspond to reverse rotation. To change speeds, the commanded velocity is set to a new value, and the profiler will move the motor frequency in the direction of the new speed by adding or subtracting the acceleration variable with each pass of the profiler routine. If this action results in the motor frequency moving farther away from 0 Hz, then the motor is accelerating. However, if such an action results in the motor frequency getting nearer to 0 Hz, then the motor is decelerating. This distinction is important because in the latter case, the motor is supplying mechanical energy to the drive, which can do significant damage if not held in check. The profiler must track deceleration events, and provide corrective action to reduce the deceleration if necessary.

With each pass of the profiler, the motor frequency is compared against the commanded velocity to determine whether velocity ramping is required. If the two are not equal, the profiler will nudge the motor frequency in the direction of the commanded velocity. Depending on the size of the nudge, this may result in the motor frequency overshooting the commanded velocity. Therefore, the magnitude of the motor frequency must immediately be compared to the magnitude of the commanded velocity, and if it is greater, the motor frequency be made to equal the commanded velocity. This action results in the motor frequency always serving to the commanded velocity with an adjustable slew rate.

If the profiler determines that the motor is accelerating, then the acceleration value is simply added to or subtracted from the velocity variable, depending on whether the velocity is positive or negative, respectively. However, if the profiler determines that the motor is decelerating, a more elaborate scheme must be implemented, as discussed below.

V_{Bus} is inspected to determine whether it is above a certain threshold that suggests the deceleration must be moderated. If it is below this threshold, the deceleration is initially equated to the acceleration value. However, if it is above this value, the deceleration is calculated as a function of acceleration and the V_{Bus} reading as defined by the following relationship:

$$deceleration = acceleration \cdot \left(1 - \frac{V_{bus} - dstart}{\$80}\right) \qquad \textit{Eqn. 1}$$

where:

dstart is the V_{Bus} reading above which we want to begin mitigating the deceleration value.

V_{Bus} is the ADC reading of the bus voltage, \$000 < V_{Bus} < \$3FF

This expression results in a linear taper of the deceleration value as a function of the bus voltage when V_{Bus} is greater than dstart, as illustrated in Figure 7.

The equation is only applicable for V_{Bus} readings in the range of dstart < V_{Bus} < dstart+\$80. For V_{Bus} readings above this range, the deceleration is set to 0.5 Hz/sec. In standalone mode, dstart is fixed at \$314, but in PC master mode, dstart is programmable.

The reason for the form of Equation 1 is because it can be implemented very efficiently on the CPU08. Because of the restrictions placed on the V_{Bus} reading, the numerator term V_{Bus} – dstart can never be less than \$01, or larger than \$7F, which means it can fit into an 8-bit value. We will assume this term is an integer, i.e., it is in 8.0 format. To divide this by \$80 (128), we realize that this is the same as multiplying by 0.0078125. As it turns out, this can be exactly represented in 0.8 format as \$02. So, the evaluation of the fraction in the brackets of Equation 1 can be accomplished by multiplying two 8-bit values, where the result is in 8.8 format. However, the integer portion of this expression is always guaranteed to be 0 because of the limitations on the V_{Bus} value. Therefore, only the least significant byte must be retained. To subtract this term from 1, we simply negate the 8-bit value, resulting in a value in 0.8 format. Therefore, the entire bracketed expression can be implemented in five assembly language instructions!

This 8-bit word is then multiplied by the 16-bit acceleration value to achieve the deceleration value.

To review, we have three possible values of deceleration depending on the V_{Bus} reading:

- The acceleration value, assuming $V_{Bus} \leq$ dstart
- Equation 1 if dstart < V_{Bus} < dstart + \$80
- 0.5 Hz/sec if $V_{Bus} \geq$ dstart + \$80

The resulting slope of Equation 1, as illustrated in Figure 7, is proportional to the gain of the closed-loop system that is controlling the deceleration in an effort to regulate V_{Bus}. Because this slope will change as a function of the acceleration value, as well as other parameters in the system, it is impossible to select a value of the slope that will guarantee stability of the loop for all systems. In an attempt to mitigate this potential problem, a time dependency is also introduced in the calculation of the deceleration value. If

Equation 1 dictates that the deceleration must be decreased quickly to prevent an overvoltage situation, it is permitted to do so. However, after V_{Bus} begins to decrease again, the deceleration is gradually increased at a maximum slew rate of approximately 167 Hz/sec^2. More precisely, it is allowed to increase at a maximum rate of 0.5 Hz/sec per each pass through the profiler routine. A perhaps inaccurate but intuitive analogy would be like quickly recoiling from the cold water of a swimming pool, followed by slowly easing your toe back into the water.

3.6.2 Voltage Calculation

The voltage profile as a function of motor speed is shown in Figure 5. Because this profile must be applied regardless of the direction of motor rotation, the absolute value of the velocity variable must be used.

The equation governing the voltage profile in Figure 5 is given as:

Eqn. 2

$$V(\omega) = \frac{\omega}{base\ speed} + v_{boost} \cdot \left(1 - \frac{\omega}{base\ speed}\right)$$

where:

$V(\omega)$ is the scaled motor voltage from 0 to 1, more commonly known as the *modulation index.*

base speed is the line frequency at which the motor was designed to work.

ω is the present motor frequency.

v_{boost} is the desired voltage boost specified at 0 Hz.

The above equation applies only for the condition of $0 \le \omega \le base\ speed$. If the motor frequency is greater than the base speed, then the modulation index is set to 1.

As was the case with the deceleration calculation, we have a fractional expression that would suggest the need for a divide operation. However, after the base speed has been specified (either 50 or 60 Hz), it may be treated as a constant. This implies that the inverse of both base speed options can be calculated *a priori*, and the selected value multiplied by ω in Equation 2. The radix format selected for 1/(base speed) is 0.16, which means that the two values are $0444 and $051E for 60 Hz and 50 Hz, respectively. The motor voltage is calculated as an 8-bit, unsigned variable.

A time dependency is also associated with the calculation of the motor voltage, in an effort to mitigate the voltage step associated with turn-on and turn-off. This is especially noticeable when the voltage boost is large. Whenever the motor frequency drops below 1 Hz, a value of 1 is subtracted from the voltage value for each pass of the profiler routine, which causes the motor voltage to be removed gently. The motor enable and disable sequencer located in the Enable.asm routine watches when the voltage value reaches 0, and disables the PWMs when this happens.

It is desirable to also apply the voltage gently to the motor when it is turned on. However, simply incrementing the voltage value by 1 for each pass of the profiler routine is not adequate if rapid acceleration is required. If the voltage value fails to increase at a rate commensurate with the motor frequency, the required V/Hz relationship will not be met, and the motor could stall. Therefore, the rate at which voltage is applied is related to the specified motor acceleration. This allows the voltage waveform

amplitude to rapidly catch up to where it must be on the V/Hz curve instead of applying a voltage step at turn-on. After it catches up, the normal voltage profile as dictated by Equation 2 resumes.

Finally, certain motor control applications require independent control of the voltage waveform with respect to the motor frequency. An example would be the case of a lightly loaded motor at full frequency, where the voltage can be reduced for better efficiency. Another example would be exercise equipment, where a softer torque response is preferred. While this requirement cannot be accommodated in standalone mode, it is possible with PC master mode by setting the voltage boost to maximum, and then controlling another variable called vmax. As seen in Equation 2, when v_{boost} is maximum (1), the expression for V(ω) is always 1, or full voltage, regardless of frequency. After V(ω) is calculated, its value is limited to not exceed vmax. If the user chooses not to set v_{boost} to maximum, then the voltage profile will follow Equation 2 up to the vmax limit, at which point the voltage will be clamped. The time dependencies of gently applying and removing the voltage still apply for the cases where vmax is used to control the voltage.

3.6.3 Velocity Scaling

Each time the PWM ISR is executed, new data is fetched from the waveform table and supplied to the PWM module, as indicated in Figure 16. The frequency of the resulting motor waveform will be directly related to how much the pointer value changes each time the PWM ISR is executed, with larger increments corresponding to higher frequencies. So the question becomes, “How can we translate the 16-bit motor frequency variable in the profiler routine to the correct wavetable pointer increment?” To answer this question correctly, we need knowledge of how frequently the pointer updates occur. For now, we will assume that PWM ISR interrupts (and consequently, pointer updates) occur every 189 μs, or at a 5.3 kHz rate.

To perform this conversion, we will use the following translation, being mindful of the units involved at each step:

Eqn. 3

$$\omega\left(\frac{cycles}{sec}\right)\cdot 189\cdot 10^{-6}\left(\frac{sec}{pointer\,update}\right)\cdot 256\left(\frac{wavetable\ points}{cycle}\right)$$

where:

ω is the 16-bit motor frequency variable in 8.8 format

When the units cancel out, the result is a frequency variable that has units of wavetable points per pointer update. This is exactly what is required for a value to be directly added to the wavetable pointer each time it is updated. Performing the multiplication in Equation 3 results in a scale factor of 0.048384, which when translated into 0.16 format, equals $0C63. However, for best resolution, any scale factor should use as much of its dynamic range as possible. To accomplish this, we multiply the scale factor by 16, yielding a new value of 0.774144, or $C62E in 0.16 format.

Multiplying this scale factor by the 16-bit motor frequency variable (in 8.8 format) results in a 32-bit result in 8.24 format. However, we must account for the 16x adjustment to the scale factor by *mentally* shifting the radix point four places to the left, resulting in a 4.28 format. Next, we will retain only the three most significant bytes, resulting in a 4.20 format. To make the value line up with the format used by the pointer,

we must shift the whole answer four places to the right. This moves the radix point between the two most significant bytes, resulting in an 8.16 format word. It can now be added directly to the pointer variable used to fetch wavetable data each time the PWM ISR is executed.

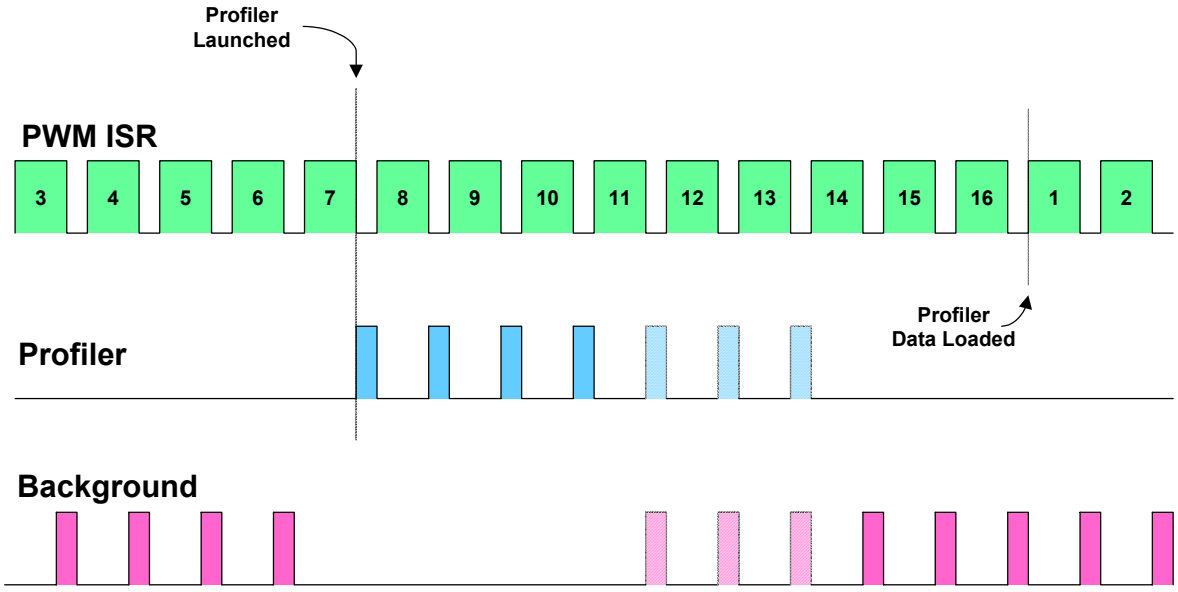
3.6.4 Acceleration Scaling

Motor acceleration is directly proportional to the amount the motor frequency variable changes each time it is updated. So the question becomes, “How can we translate the 16-bit acceleration variable to the correct increment to add to the motor frequency variable each time it is updated in the profiler routine?” To answer this question correctly, we need knowledge of how often the profiler routine is executed.

Figure 23 is a timing diagram that shows that the profiler is invoked every 16th PWM interrupt. From the previous discussion on velocity scaling, we will assume that the PWM interrupt occurs every 189 µs. To scale the acceleration properly, we use this information as illustrated in the following translation, being mindful of the units involved at each step:

Eqn. 4

$$acceleration\left(\frac{Hz}{sec}\right)\cdot 189\cdot 10^{-6}\left(\frac{sec}{PWM\ interrupt}\right)\cdot 16\left(\frac{PWM\ interrupt}{motor\ \ freq.update}\right)$$



High level indicates CPU is servicing this activity.

Figure 23. Timing Relationship Between PWM ISR, Profiler, and Background Tasks

When the units cancel out, our result is an acceleration variable that has units of Hz per motor frequency update. This is exactly what is required for a value to be added directly to the motor frequency variable each time it is updated. Performing the multiplication in Equation 4 results in a scale factor of 0.003024, which when translated into 0.16 format, equals $00C6. However, for best resolution, any scale factor should use as much of its dynamic range as possible. To accomplish this, we multiply the scale factor by 256, yielding a new value of 0.774144, or $C62E in 0.16 format.

Multiplying this scale factor by the 16-bit acceleration variable (in 7.9 format) results in a 32-bit number (in 7.25 format). However, we must account for the 256x adjustment to the scale factor by *mentally* shifting the radix point eight places to the left, resulting in a value with a 33-bit fractional part, which exceeds the 32-bit word size of the number. However, this is not a problem after we realize that the largest number we can obtain from a 16x16 multiply is 32 bits, i.e., bit 33 is always guaranteed to be 0. Next, we discard the two least significant bytes, leaving a value with a 17-bit fractional part, where the most significant bit is guaranteed to be 0. To make the value line up with the format used by the motor frequency variable, we must shift the whole answer one place to the right, with the implied 0 of bit 17 moving into the bit 16 spot. This moves the radix point to a spot immediately above the most significant byte, resulting in a 0.16 format word. It can now be added directly to the motor frequency variable each time the profiler routine is executed.

3.6.5 Special Case Scaling

Before leaving the discussion of frequency and acceleration scaling, we need to address the special case scenario when the PWM interrupt rate is not every 189 μs. This will happen whenever the 15.873 kHz PWM frequency option is selected; the reasons for which are presented later. Under this special case, the PWM interrupts will occur every 252 μs instead. As a consequence, different scale factors must be used in this case, and the profiler must monitor the PWM frequencies to know which scale factors to use. By using the same technique presented in the previous two sections, we come up with a scale factor for this PWM frequency of $841F, to be used for both acceleration and frequency scaling.

3.6.6 Velocity Pipelining

Referring back to Figure 23, we see that new profiler data is required by the PWM ISR every 16 interrupt cycles. Although the choice of numeric nomenclature is arbitrary, we will refer to the PWM ISR execution immediately following the reload of profiler data as cycle 1. At cycle 7, the PWM ISR will trigger the profiler routine, which runs in-between the PWM ISR executions as a background task. During this time, interrupts are enabled, which allows other ISR tasks to execute if required. The gap between the PWM ISR executions is typically 20 to 40 μs, and it usually takes about four such cycles for the profiler to finish calculating new data.

As illustrated in Figure 15, the profiler passes three pieces of information to the PWM ISR each time it is executed:

- The velocity calculated from the last pass of the profiler
- The difference in velocity from the last pass to this pass of the profiler (acceleration)
- The voltage (modulation index) to be applied to the waveforms

This results in a pipelined effect, where previous profiler data is used by the PWM ISR, while new data is being calculated in the background at the same time. This process is illustrated in Figure 24.

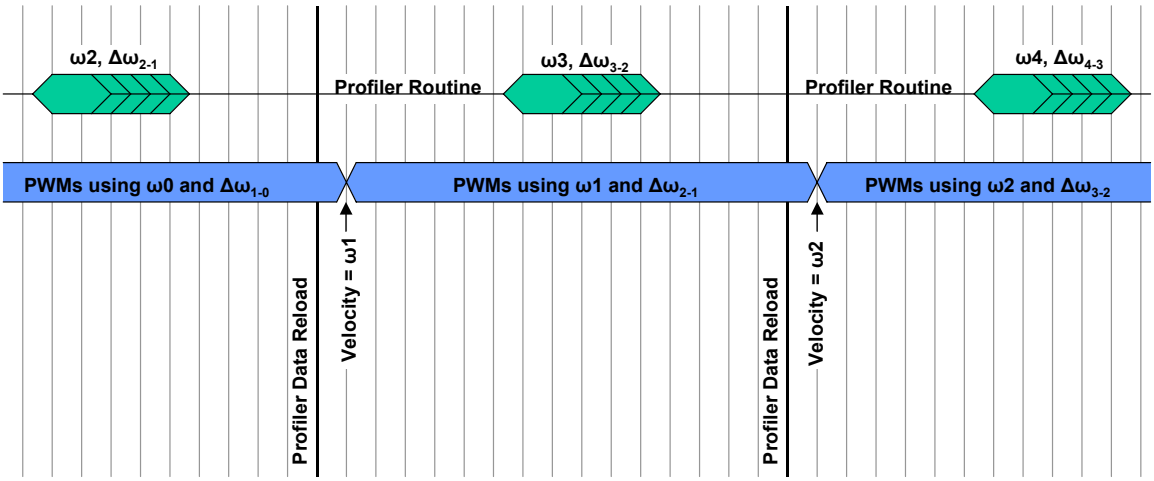


Figure 24. Velocity Pipelining.

Each vertical partition represents a PWM interrupt that instigates one pass through the PWM ISR. In this particular illustration, the first invocation of the profiler routine calculates a velocity we will call ω_2. However, as can be seen, the PWMs applied to the motor will not reach this velocity until much later; the reason for which will be explained momentarily. Also, note that the PWMs require one interrupt cycle to reflect the new data whenever a profiler data reload occurs. This is because the PWM module contains double-buffered registers for the PWM values. So while the new PWM values are being calculated and loaded into the buffers, they do not actually get used until the next PWM interrupt, at which point they are loaded into the PWM compare registers.

Because the velocity calculations are performed in an open-loop fashion (with the exception of bus voltage monitoring during deceleration), the phase lag represented by this pipeline will not adversely affect the performance of the system. However, phase delay will adversely affect performance in the case of deceleration modulation based on bus voltage feedback. To mitigate this effect, we see that the acceleration information ($\Delta\omega$) is applied immediately after the reload of profiler data.

3.6.7 Velocity Interpolation

Assuming that the PWM interrupts occur at 189 μs intervals, and new velocity is calculated every time the profiler executes (every 16 PWM interrupts), then the velocity will be incremented every 3 ms, or 331 times a second. Depending on the motor and the acceleration specified, this can result in discrete steps that may be felt by the motor. Smoother motor performance results if the velocity information is updated at a quicker rate. Unfortunately, to execute the profiler at a faster rate would require more CPU bandwidth than what is available on the CPU08. However, there is a way to get around this, as illustrated in Figure 25.

As discussed in the previous section on velocity pipelining, each time the profiler runs, as shown in the shaded portions of Figure 25, three pieces of information are supplied to the PWM ISR; the old velocity, the delta velocity, and the modulation index (not shown in Figure 25). Each velocity value calculated by the profiler is shown as a yellow circle, where the stair-stepped nature of these values is clearly evident. In the implementation of the delta velocity calculation, the value is actually right shifted four times before

being supplied to the PWM ISR, which has the effect of dividing it by 16. By now, the reason for updating the profiler information every 16 PWM ISR cycles may be evident. By adding this increment to the old velocity value each time the PWM ISR is executed, we will essentially generate a new velocity curve with 16x more resolution, as shown in Figure 25. This process is called linear interpolation, which will exactly track the desired velocity profile because the profiler in the MC3PHAC generates a linear profile. As a result, the motor will transition from one speed to the next with extremely smooth velocity performance.

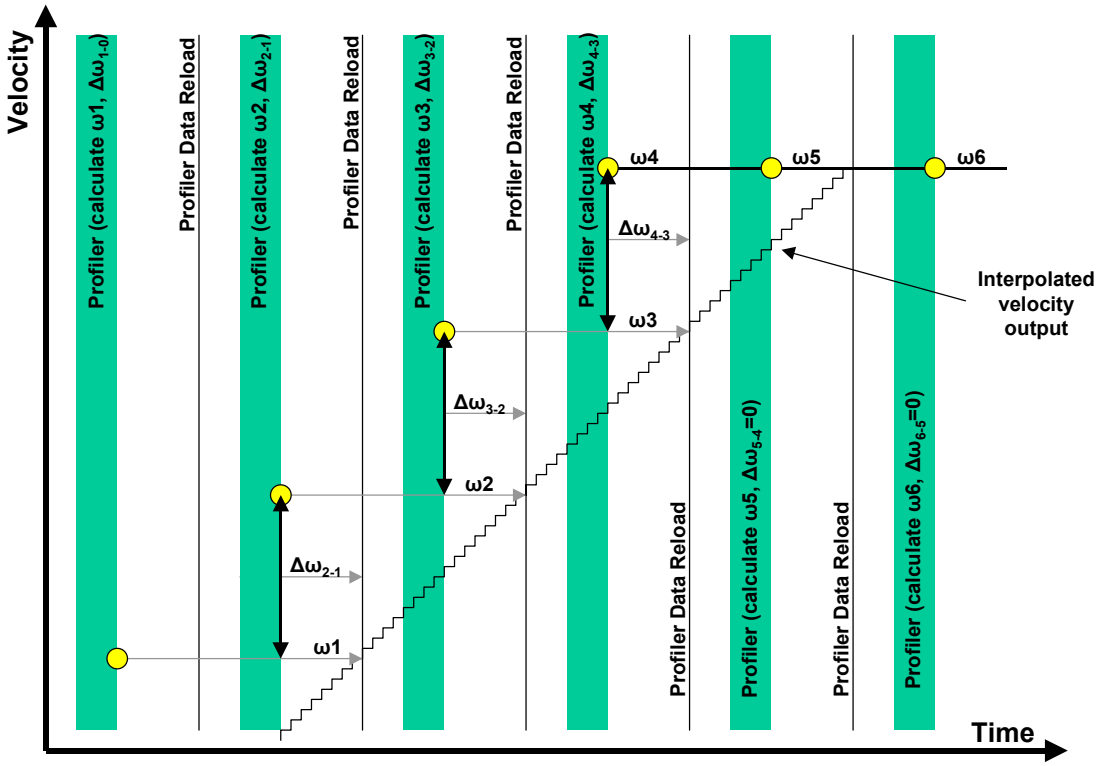


Figure 25. Velocity Interpolation

3.6.8 PWM Frequency Gear-Shifting

The MC3PHAC has the ability to change the PWM frequency on the fly, while the motor is running. However, when the PWM frequency is 15.873 kHz, it results in a PWM interrupt rate of 252 µs, which requires a different scaling factor for the velocity and acceleration data.

Because the profiler calculations are pipelined, this different scaling factor presents a problem. The profiler must monitor whenever a PWM frequency gearshift to or from 15.873 kHz occurs and respond appropriately. If the procedure in Figure 24 is followed when such a gearshift occurs, the profiler will calculate the new velocity (ω_x) and acceleration data using the new scale factor. However, the old velocity (ω_{x-1}, required as part of the reload), was calculated with the old scale factor. Because the velocity information is calculated assuming a specific wavetable pointer update rate, and because the update rate is now different, application of the old velocity value would result in a step function error in the velocity waveform. In addition, the delta velocity information (or acceleration) would be wrong as well, because the old velocity is required for this calculation. This perturbation in the velocity and acceleration waveforms would last for one complete profiler reload cycle.

So when the profiler detects a gearshift to or from 15.873 kHz, the following special case actions are performed during the profiler pass associated with the gearshift:

- All calculations associated with velocity ramping are suspended.
- The old velocity variable is re-scaled so that it will work correctly with the new update rate.
- The delta velocity variable is set to 0.

If the profiler isn't ramping the speed at the time, then these actions will have no effect on the motor. However, if the profiler is ramping the speed when the gearshift occurs, it will result in the velocity profile stalling (flattening out) for one profiler reload cycle, to allow the pipe to be refilled with good data. After this occurs, the profiler will resume normally with the ramp in progress. Because this is a pipeline related issue, the solution is similar to flushing the pipe, which is common in other pipelined architectures.

3.7 Interrupt Timing

As illustrated in Figure 13, there are several sources of interrupts in the MC3PHAC code. As with any software with interrupt capability, there are certain checks and balances that must be carefully maintained, as discussed below.

3.7.1 PWM ISR

The PWM ISR is central to all timing operations in the MC3PHAC that pertain to motor dynamics, such as PWM frequency, motor acceleration, and motor velocity. During execution of this ISR, interrupts remain disabled. Referring to Figure 23, we see that the profiler is invoked at the end of the seventh PWM ISR cycle, where cycle 1 is arbitrarily chosen to be where the profiler data is loaded. At the end of the seventh cycle, the PWM ISR does not exit normally, but instead transfers execution to the profiler via a subroutine call. In the profiler, interrupts are immediately enabled to allow the PWM ISR to respond to future interrupts from the PWM module. When the profiler completes, it returns execution back to its point of origin in cycle 7, where an RTI instruction returns execution to the background task in progress when the cycle 7 interrupt occurred.

The PWM ISR runs at a higher frequency than any other periodic ISR, and for this reason, the dc bus monitoring function is performed here. The other ADC channel conversions are also performed in this ISR during the seventh PWM ISR cycle, so that they will be fresh when the profiler uses them. Because this ADC data will not be overwritten until after the profiler has finished, the data will be coherent throughout the execution of the profiler code. This is not the case, however, with the dc bus reading, which is updated each time the PWM ISR is executed. Therefore, care must be taken to establish the V_{Bus} variable only once within a background calculation that uses multiple occurrences of the variable.

3.7.2 SCI Rx ISR

Whenever a character is received in the SCI receive buffer (presumably as a result of communications with an external master via the FreeMASTER protocol), an interrupt will be generated and the SCI Rx ISR will be invoked. To prevent starving the PWM ISR, the SCI Rx ISR enables interrupts, with the understanding that if it is interrupted, execution privilege will be given back before the receive buffer can overflow. However, referring to Figure 23, what if the next PWM interrupt corresponds to cycle 7? Because

interrupts were enabled at the time, the SCI Rx ISR would be treated like a background routine, and the profiler execution would take precedence. The result is that the SCI Rx ISR could be starved instead.

To reciprocate the kindness of the SCI Rx ISR enabling interrupts, the PWM ISR will check at the end of cycle 7 to determine whether the SCI Rx ISR was running when the PWM interrupt occurred. If so, it will make a note to launch the profiler on the cycle following the completion of the SCI Rx ISR, and instead return execution to the SCI Rx ISR. By this action, a priority chain is established which consists of the PWM ISR, followed by the SCI Rx ISR, followed by the profiler, followed by other background tasks.

3.7.3 Other ISRs

The three remaining ISRs are fairly generic in nature with no special timing requirements. The PLLCheck ISR is fatal, as it indicates that a problem has occurred with the PLL to cause it to lose phase lock. Variables shared between the remaining ISRs and background routines must be guarded for coherency, especially when there are multiple occurrences of the variable in each background calculation.

3.8 Step Invariant Digital Filter

To enhance reference speed stability, the ADC input used for the speed signal is processed with a single pole, step invariant digital IIR filter, as shown in Figure 26. It gets its name from the fact that it is designed using the Z-transforms of the input and output waveforms of an RC filter when an input step function is applied.

This filter topology is actually a sampled approximation of how an analog RC filter really works, and consequently, its frequency and time domain responses closely resemble those of the analog filter. For more information, see Appendix A. The piecewise equations governing the operation of an analog single-pole, low-pass filter have been derived in the appendix and match the difference equation in Figure 26.

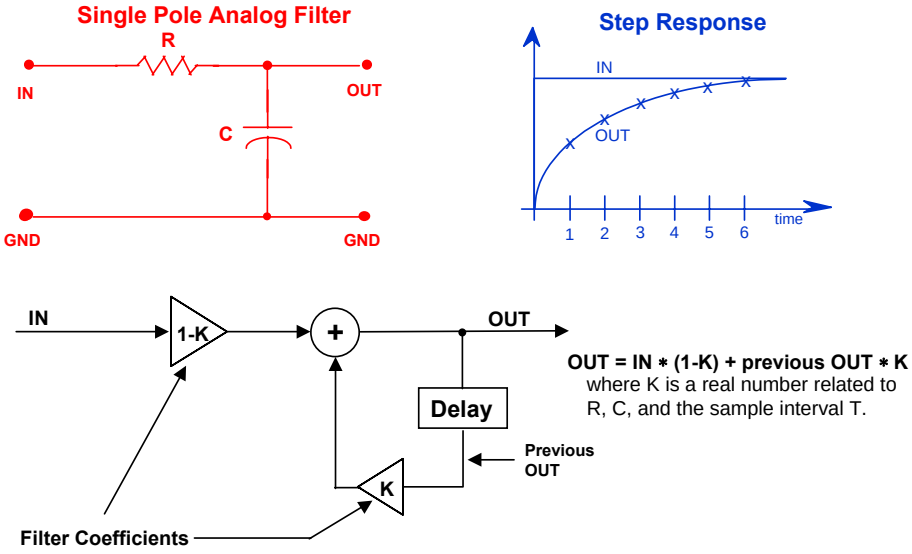


Figure 26. Step Invariant IIR Digital Filter Topology

As can be seen, each new output of the filter is calculated as the summation of a weighted sample of the input, and a weighted sample of the previous output. From an intuitive perspective, the operation of the filter can be understood by asking, “How much filtering is desired?” If little filtering is needed, we simply turn up the gain on the input term, and lower the gain on the feedback term. Indeed, if the input gain is 1 and the feedback gain is 0, then the output will simply track the input, and no filtering will be performed. On the other hand, if a lot of filtering is needed, we turn up the feedback gain, and lower the input gain. If the input gain reaches 0, and the feedback gain is 1, this is analogous to having an infinite size capacitor on the analog filter counterpart. In all cases, the sum of the feedback gain and the input gain must always be 1.

The trick to implementing this filter quickly on a machine other than a DSP is in the judicious selection of the value of k. If chosen wisely, all the multiplications associated with the filter can be avoided altogether and the filter can be implemented as a series of shifts and adds. The value chosen for k on the MC3PHAC is 0.9921875, which will result in a heavily filtered output. The reason behind the selection of such an odd number should become apparent in the next few paragraphs.

k = 1 – 1 / (128). If we substitute this value into the difference equation presented in Figure 26, we obtain the following equation:

Eqn. 5

$$Y(n) = \left(1 - \frac{1}{128}\right) \cdot Y(n-1) + \frac{1}{128} \cdot X(n)$$

or

$$Y(n) = Y(n-1) - \frac{1}{128} \cdot Y(n-1) + \frac{1}{128} \cdot X(n)$$

where:

Y(n) = the present filter output

Y(n-1) = the previous filter output

X(n) = the present filter input

Both the previous filter output and the present filter input must be divided by 128. Because all summed intermediate values are implemented in 24-bit, multi-precision arithmetic, this could result in plenty of cycles to perform these operations. However, on a computer, this is equivalent to shifting right by seven bits. Unfortunately, even this will take quite a while to perform on two 24-bit values. The main trick is to realize that seven shifts to the right is equivalent to one shift to the left, followed by simply changing the byte boundary by which the variable is accessed by one byte to the left.

Figure 27 shows an excerpt from the MC3PHAC code that actually implements this filter. Notice that as much of a variable as possible is loaded into the X and A registers prior to shifting, because shift operations performed in these registers use only one bus clock cycle.

Page 1

```
ldx       out
lda       out+1    ; X:A = last filter output value

clr       ptemp5

lsla
rolx
rol       ptemp5
stx       ptemp6   ; ptemp5:ptemp6:A = out/128

nega
sta       ptemp7

lda       out+1
sbc       ptemp6
sta       ptemp6

lda       out
sbc       ptemp5
sta       ptemp5   ; ptemp5:ptemp6:ptemp7 =
                   ; out * (1 - 1/128)
```

Page 2

```
lda       ADChan1+1
ldx       ADChan1     ; X:A = filter input

clr       ADChan1

lsla
rolx
rol       ADChan1     ; ADCHAN1:X:A = filter input / 128

add       ptemp7
txa
adc       ptemp6
sta       out+1
lda       ADChan1
adc       ptemp5
sta       out         ; out = out * (1 - 1/128) +
                      ;       filter input * (1/128)
```

This filter executes in 9.2 uS for an 8 MHz CPU08

Figure 27. Implementation of Digital IIR Filter on the CPU08

When the final output is calculated, the two most significant bytes are saved. They will become the last filter output values the next time the filter is executed. As indicated in Figure 27, this implementation of the filter requires less than 10 μs on an 8 MHz HC08 device.

Because this filter consists of a single pole, it will have a 20 dB/decade frequency attenuation response, just like its analog counterpart. To achieve further attenuation, several such stages may be cascaded together, again, just like its analog counterpart.

3.9 Waveform Generation

The algorithm used to calculate the PVALx PWM register values was created because of a need to develop a fast PWM update technique for distortion correction on the MC68HC708MP16. To understand how the technique works, it is appropriate to review the principles of PWM modulation.

Figure 28 shows a half-bridge power configuration driving a PWM signal. If we assume dead-time is 0, and sinusoidal modulation is employed, the averaged or filtered output for unity power supply voltage is given as:

Eqn. 6

$$\overline{V_o(t)} = \frac{t_h(t)}{T} = \frac{1}{2} + \frac{M}{2}\sin(\omega_o t + \theta)$$

where:

$V_o(t)$= the averaged output voltage

$t_h(t)$ = high time of the PWM signal

T = the PWM period

M = the modulation index (from 0 to 1)

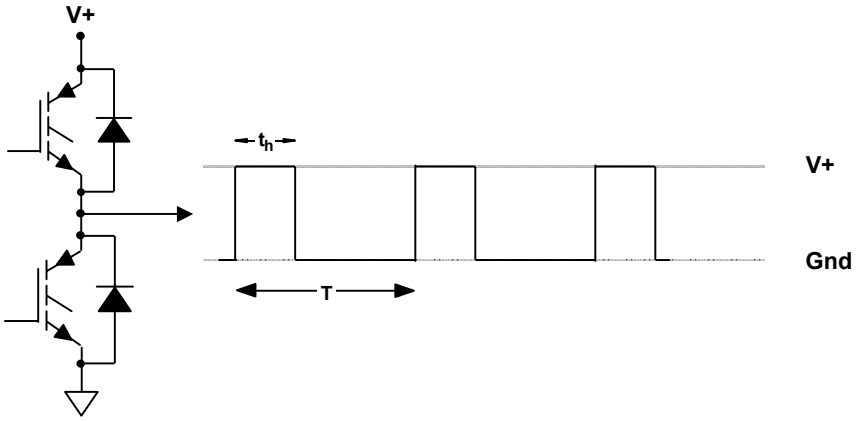


Figure 28. PWM Output of a Typical Half-Bridge

Solving for the high time:

Eqn. 7

$$t_h(t) = \frac{T}{2} + \frac{TM}{2}\sin(\omega_o t + \theta)$$

We can also establish the following relationships:

$$T = PMOD \cdot 250\,nS$$

$$t_h(t) = PVAL(t) \cdot 250\,nS$$

where:

PMOD is the register value that sets the PWM frequency.

PVAL(t) is the register value that sets the pulse width.

Substituting, we obtain:

Eqn. 8

$$PVAL(t) = \frac{PMOD}{2} + \frac{PMOD \cdot M}{2}\sin(\omega_o t + \theta)$$

In defining a waveform table to implement the above equation, the most obvious choice is to define a sinewave table that can then be scaled and biased appropriately. However, a pure sinewave table consists of signed entries, which will require a signed multiply during the scaling process. Because signed multiplications are not directly supported by the CPU08, it makes sense to avoid signed values.

Instead of signed values, let's scale and bias the waveform so that it occupies a range between 0 and 1 instead of –1 and +1. In other words:

Eqn. 9

$$wave(t) = \frac{1}{2} + \frac{\sin(\omega_o t + \theta)}{2}$$

If we multiply both sides of the above expression by the product of PMOD and M, we obtain:

$$wave(t) \cdot PMOD \cdot M = \frac{PMOD \cdot M}{2} + \frac{PMOD \cdot M}{2} \cdot \sin(\omega_o t + \theta) \quad \textit{Eqn. 10}$$

This operation very nearly results in the correct expression for PVAL(t) on the right side of the equation. Unfortunately, the first term on the right side has the modulation index M in it. If we could remove this, we would have an expression for PVAL(t), which can be directly loaded into the PVALx register. The question then becomes, “What term Z can be added to the first term on the right side to result in the correct value of PMOD/2”? In other words:

$$\frac{PMOD \cdot M}{2} + Z = \frac{PMOD}{2} \quad \textit{Eqn. 11}$$

Solving for Z, we obtain:

$$Z = \frac{PMOD}{2}(1 - M) \quad \textit{Eqn. 12}$$

So if Z is added to both sides of Equation 10, the expression on the right side results in a value that equals PVAL(t). Performing this operation, and substituting PVAL(t) for the right side of the expression, Equation 10 can be rewritten as:

$$wave(t) \cdot PMOD \cdot M + \frac{PMOD}{2}(1 - M) = PVAL(t) \quad \textit{Eqn. 13}$$

Recall that because the modulation index M can be a value only between 0 and 1, we now have an expression that requires no signed multiplications. Also, because the PMOD term is not anticipated to change very frequently, the term PMOD/2 can be calculated ahead of time and is treated as a constant in Equation 13. Unfortunately, the first term in Equation 13 requires three terms to be multiplied together. To get around this, another trick is employed. Instead of creating a waveform table as defined by Equation 9, let’s include the PMOD multiplication as part of the table. In other words:

$$wave(t) = PMOD \cdot \left(\frac{1}{2} + \frac{\sin(\omega_o t + \theta}{2} \right) \quad \textit{Eqn. 14}$$

The waveform table now consists of values between 0 and PMOD. Equation 13 can then be rewritten as:

$$PVAL(t) = wave(t) \cdot M + HalfPMOD \cdot (1 - M) \quad \textit{Eqn. 15}$$

To preserve memory, and to simplify the calculation of Equation 15, it is desired to limit the wave(t) entries to 8-bit values. However, for best resolution, the waveform should use as much of the 8-bit range as possible. We also want

the waveform table to be fixed, so it can occupy space in program memory instead of RAM. As might be expected, all these requirements place a limitation on the selected value of PMOD.

Strict interpretation of Equation 15 would suggest that if the waveform table is fixed, then only one value of PMOD, and thus only one PWM frequency, is possible. For example, if the waveform table is scaled to span the full-8 bit range from 0 to 255 (PMOD = 255), this would mean that the PWM frequency must be 15.7 kHz. However, if we are willing to live with discrete steps in the PWM frequency options, we can employ another trick to accommodate this. To illustrate, assume the waveform table contains values from 0 to 252, implying a PMOD value of 252, or 15.9 kHz. However, if PMOD is really set to 378, or 10.6 kHz, then the first term on the right side of Equation 15 will be wrong. To correct this term, it must be multiplied by 1.5. On a digital computer, this is equivalent to shifting the term right one time and adding it back to the original value (both operations can be performed very quickly and efficiently). So, by judicious selection of allowable PMOD values, the resulting error in Equation 15 can be easily corrected.

Table 7 shows the four allowable PMOD values in the MC3PHAC code, the resulting PWM frequencies, the PWM interrupt rate, and the correction operation which must be performed to the first term on the right side of Equation 15.

Table 7. Allowable PMOD Values

PMOD Value	PWM Frequency	Correction Operation	PWM Interrupt Rate
756 ($2F4)	5.291 kHz	Left shift, multiply by 1.5	PWM period (189 uS)
378 ($17A)	10.582 kHz	Multiply by 1.5	PWM period x 2 (189 uS)
252 ($FC)	15.873 kHz	none	PWM period x 4 (252 uS)
189 ($BD)	21.164 kHz	Right shift, multiply by 1.5	PWM period x 4 (189 uS)

The last entry in the table (PMOD = 189) requires that PMOD be shifted right (126), and then be multiplied by 1.5. To multiply 126 by 1.5 using the technique described above requires that it be shifted right once more. Therefore, 126 must be an even number, or the last right shift will result in a value with a fractional part. This is why the waveform table assumes PMOD = 252 instead of 254.

Also note that for every PMOD value except $FC, the PWM interrupt period is 189 μs. This is due to the ability of the MC3PHAC PWM module to change its interrupt period to be integral power-of-2 multipliers of the PWM period. Unfortunately, when PMOD equals $FC, the PWM period is 63 μs. To generate a 189 μs interrupt period from this would require a multiplier of 3. While this ability has been added to the PWM module on our DSP devices, it is not possible to achieve with the MC3PHAC. As a result of this different interrupt period, the scale factor associated with the frequency and acceleration variables must also be different.

The waveform table consists of one cycle of data, as shown in Figure 16. The actual waveform used is a fundamental sinewave with third harmonic injection. Velocity information received from the profiler is integrated to create an angle variable, and this angle is used as a pointer into the waveform. If the waveform consists of one complete cycle that is expected to repeat end to end, and the number of entries in the waveform table is selected appropriately, the modulo nature of binary arithmetic can be used to our advantage to simplify the pointer calculation. For example, if the waveform table has 256 entries, and the pointer is an 8-bit variable, then whenever the pointer update calculation overflows, it wraps around using modulo 256 arithmetic, which points to the correct value in the waveform table.

The MC3PHAC uses a pointer that also incorporates a 16-bit fractional part. This permits very smooth scrolling of the pointer, allowing for fine velocity resolution, which is especially important at lower speeds. Because it is impossible to define a fractional entry number in the waveform table, the fractional part of the pointer is rounded to the nearest integer value to fetch the appropriate table entry.

Figure 16 also shows that because this is a three-phase motor controller, three waveform values must be fetched each time the pointer is updated. This is accomplished by adding the equivalent of 120 degrees to the pointer angle value for each additional wavetable fetch.

In actuality, the MC3PHAC incorporates a 512-point waveform table, for better angular resolution of the waveform. However, the modulo principles discussed in the last few paragraphs still apply. To accommodate a 512-point table, the angle pointer is simply shifted left one bit before being used to fetch the waveform data. A 256-point table was discussed for tutorial purposes only, to help understand the concept of modulo arithmetic applied to pointer addressing.

3.10 Bus Ripple Cancellation

In the derivation of the equations related to waveform generation, one of the first assumptions we made was that the power supply voltage (or bus voltage) was unity. This is rarely the case. Depending on the size of the bus capacitor with respect to the motor load, and also whether the input mains are single phase or three phase, there may be significant ripple on the dc bus. There may also be much lower-frequency, higher-amplitude variations in bus voltage under aggressive deceleration or brownout conditions. For these reasons, it is often necessary to sense the bus voltage and compensate the PWMs to correct for these perturbations to prevent them from being passed on to the motor.

In Figure 28, if we assume that V+ = $V_{Bus}(t)$ instead of unity, we must rewrite Equation 6 to include $V_{Bus}(t)$, as shown in Equation 16.

Eqn. 16

$$\overline{V_o(t)} = \frac{t_h(t)}{T} \cdot V_{bus}(t) = \left(\frac{1}{2} + \frac{M}{2}\sin(\omega_o t + \theta)\right) \cdot V_{bus}(t)$$

By substituting PVAL(t) and PMOD in Equation 16 for $t_h(t)$ and T respectively, we obtain:

Eqn. 17

$$\overline{V_o(t)} = \frac{PVAL(t)}{PMOD} \cdot V_{bus}(t)$$

We can now substitute the expression for PVAL(t) from Equation 15 into Equation 17, resulting in:

Eqn. 18

$$\overline{V_o(t)} = \frac{wave(t) \cdot M + HalfPMOD - HalfPMOD \cdot M}{PMOD} \cdot V_{bus}(t)$$

When the bus voltage is at its nominal value, which we will call V_{norm}, the resulting phase output voltage will be at its correct value of $\overline{V_o(t)}_{norm}$, as determined by the PWM modulation. However, when the bus

voltage is not at its nominal value, we must adjust the PWM modulation to drive $\overline{V_o(t)}$ to $\overline{V_o(t)}_{norm}$. This is illustrated in Equation 19, where the correction is placed in square brackets.

Eqn. 19

$$\overline{V_o(t)}_{norm} = \frac{\left(wave(t)\cdot M + HalfPMOD - HalfPMOD\cdot M\right)\cdot\left[\frac{V_{norm}}{V_{bus}(t)}\right]}{PMOD}\cdot V_{bus}(t)$$

As seen in Equation 19, any deviation in $V_{Bus}(t)$ from V_{norm} will be corrected, because the $V_{Bus}(t)$ terms cancel out, leaving only V_{norm}.

Equation 19 results in perfect correction of bus voltage perturbations, as the output voltage is impervious to changes in the bus voltage. However, this is not the optimal situation. To allow for the maximum modulation index without clipping either the top or bottom of the output waveform, it is required that the waveform be centered in the middle of the V_{Bus} voltage range. Equation 19 will always center the waveform around ½ V_{norm} instead. For low values of V_{Bus}, this will result in the top portion of the output waveform being clipped before the bottom portion of the waveform. To center the waveform around ½ of the V_{Bus} reading, we first must realize that the numerator in Equation 19 consists of three terms; two of which involve the modulation index (M), and one that is simply a bias term. Therefore, the correction should be applied only to the terms containing the modulation index, thus correcting the gain of the waveform while leaving the bias term untouched. The final expression for the PVAL(t) calculation, which includes bus ripple correction, is presented in Equation 20.

Eqn. 20

$$PVAL_x(t) = wave_x(t)\cdot M\cdot\left[\frac{V_{norm}}{V_{bus}(t)}\right] + HalfPMOD\cdot\left(1 - M\cdot\left[\frac{V_{norm}}{V_{bus}(t)}\right]\right)$$

Implementation of Equation 20 will result in any noise associated with $V_{Bus}(t)$ being reflected in the output waveform. However, because it will show up on all motor phases simultaneously, it will be seen as a common mode distortion, and thus be rejected by the motor.

The above expression must be calculated for each phase of the motor. Fortunately, the second term in the expression is common for each phase, so it needs to be calculated only once. Also, the bus ripple correction of the modulation index is common to both terms, so it is calculated once and then applied to each term.

The value selected for V_{norm} is $02CD, which corresponds to an ADC reading of 3.5 V, assuming V_{REFH} is 5 V. A new V_{Bus} reading is taken every time the PWM ISR is executed. Because the reading is 10 bits, and the CPU08 can only support an 8-bit divisor, another division technique must be employed. The algorithm selected for use with the MC3PHAC is called a block divide technique. For each possible ADC reading with the exception of two values, the technique completes in four iterations or less. For these two special cases, the quotient is stipulated ahead of time, and the divide algorithm is bypassed altogether. For more information on the operation of this divide algorithm, refer to the series of articles by Jack W. Crenshaw in Embedded Systems between September and December of 1997.

Appendix A

Referring to Figure 29, the voltage on the capacitor in the analog filter is directly related to the amount of charge (q) deposited on the capacitor plates, and inversely proportional to the capacitance, as defined by Equation 21:

Eqn. 21

$$v_c(t) = \frac{q_c(t)}{C}$$

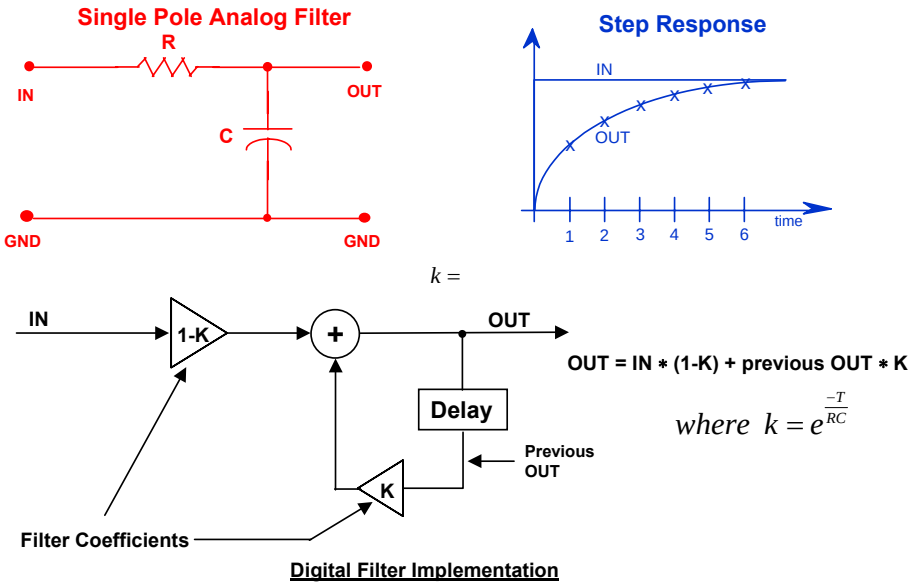


Figure 29. Comparison Between a Single Pole Analog and Digital Filter

Assuming that nothing is connected to the output of the analog filter, all of the current flowing through the resistor R results in charge build up on the capacitor. In fact, the capacitor integrates the current over time to develop the charge, as stated in Equation 22:

$$q_c(t) = \int_{-\infty}^{t} i(t)\,dt$$

Eqn. 22

If we substitute Equation 22 into Equation 21, we obtain:

$$v_c(t) = \frac{1}{C}\int_{-\infty}^{t} i(t)\,dt$$

Eqn. 23

We will now define a region in time T where this integration will be performed, such that $T = t_2 - t_1$. As a result, we must now include the previous state of the capacitor by accounting for its voltage prior to the integration interval:

$$v_c(t_2) = \frac{1}{C}\int_{t_1}^{t_2} i(t)\,dt + v_c(t_1)$$

Eqn. 24

If we expand the expression for the current, we see that it is the difference between the input voltage and the output voltage divided by the resistor value, as shown in Equation 25.

$$v_c(t_2) = \frac{1}{C}\int_{t_1}^{t_2} \frac{v_{in}(t) - v_c(t)}{R}\,dt + v_c(t_1)$$

Eqn. 25

For our piecewise analysis, we will assume that the current is a constant in the interval between t_1 and t_2 . Therefore, it doesn't matter whether we define the current at t_1, t_2, or anywhere in between. So, let's define the current at t_1, as indicated in Equation 26.

$$v_c(t_2) = \frac{1}{C}\int_{t_1}^{t_2} \frac{v_{in}(t_1) - v_c(t_1)}{R}\,dt + v_c(t_1)$$

Eqn. 26

Because the voltages at t_1 correspond to specific values, we can now treat them as constants. We solve Equation 26 by pulling R out of the integral, and performing the integration:

$$v_c(t_2) = \frac{1}{RC}\left[T \cdot v_{in}(t_1) - T \cdot v_c(t_1)\right] + v_c(t_1)$$

Eqn. 27

This can be further simplified to:

$$v_c(t_2) = \frac{T}{RC} v_{in}(t_1) + \left(1 - \frac{T}{RC}\right) \cdot v_c(t_1)$$

Eqn. 28

If we compare the topology of this equation with the discrete difference equation of Figure 29, we see that they have the same form. However, in Figure 29, the correct value of k that is multiplied by the previous output term is given as:

$$k = e^{-\frac{T}{RC}}$$

Eqn. 29

This is clearly different than the result presented in Equation 28. However, as T/RC approaches 0 (i.e., the sampling frequency approaches infinity, or a continuous system), we can rely on another identity to resolve this conflict:

$$\lim_{\frac{T}{RC}\to 0} e^{-\frac{T}{RC}} = 1 - \frac{T}{RC} \qquad \textit{Eqn. 30}$$

Finally, it can be seen that unlike the difference equation of Figure 29, Equation 28 uses the previous value of v_{in} (defined at t_1) instead of the present value (defined at t_2). This is the mathematically correct way to implement the filter. However, by using the present value instead of the previous value, the filter has a minimal effect of shifting the output in time by a small advance. It also requires less memory to execute because the last input sample doesn't have to be stored in RAM.

In summary, the difference equation defining the digital filter is shown to discretely model the process by which charge is collected on a capacitor to create a filtering effect.

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

AN2988
Rev. 1.2, 11/2005

freescale™ semiconductor