MOTOROLA
intelligence everywhere™

digital dna™

BLDC Motor Control Board for Industrial and Appliance Applications

Designer Reference Manual

M68HC08
Microcontrollers

DRM007/D
2/2003

MOTOROLA.COM/SEMICONDUCTORS

BLDC Motor Control Board for Industrial and Appliance Applications Reference Design

By: Jorge Zambada
Email: jorge.zambada@motorola.com
Applications Engineer — Mexico Applications Lab

Diego Garay
Email: diego.garay@motorola.com
Applications Engineer — Mexico Applications Lab

Maurizio Acosta
Email: m.acosta.duran@motorola.com
Applications Engineer — Mexico Applications Lab

Revision History

To provide the most up-to-date information, the revision of our documents on the World Wide Web will be the most current. Your printed copy may be an earlier revision. To verify you have the latest information available, refer to:

http://motorola.com/semiconductors

The following revision history table summarizes changes contained in this document. For your convenience, the page number designators have been linked to the appropriate location.

Revision History

Date	Revision Level	Description	Page Number(s)
February, 2003	N/A	Initial release	N/A

List of Sections

List of Sections

Table of Contents

Section 1. Introduction and Setup

Section 2. Operational Description

Section 3. Schematics and Bill of Materials

Section 4. Hardware Design Considerations

Section 5. Software Design Considerations

List of Figures

List of Figures

List of Tables

List of Tables

Section 1. Introduction and Setup

1.1 Contents

1.2 Introduction

Motorola's BLDC (brushless dc motor) control board for industrial and appliance applications is a system for controlling a 3-phase BLDC motors with three Hall-effect position sensors. The system consists of hardware and software tools for controlling this type of motor.

Hardware consists of:

- Three-phase inverter
- Sensing circuitry for current, voltage, and temperature
- User interface: 16 x 2 character display and two push buttons
- On-board power supply: 15 Vdc or 5 Vdc
- Optoisolated RS-232 interface for external microcontroller communication and for in-application programming.

There are two board versions available, one for operating at 110–127 Vac and the other for operating at 220–240 Vac. The 3-phase inverter of the 110–127 Vac board operates at a nominal voltage of 180 Vdc and 8 A RMS with 11 A peak. The inverter of the 220–240 Vac board operates at a nominal voltage of 320 Vdc driving the same current.

The example software consists of the following, but may be easily modified to perform other process cycles.

- PI speed controller for closed loop control
- Six-step BLDC commutation control based on three Hall-effect position sensors
- User interface control
- Two washing machine process implementations: wash process and spin process

The wash process consists of generating a sine wave of speed references, including positive and negative reference speeds. The spin process consists of generating a start up curve of reference speeds and maintaining a fixed reference speed for a certain time. The PI speed controller operates in the 200 rpm up to 4000 rpm range.

1.3 MC68HC908MR8

Motorola offers several 8-bit and 16-bit microcontroller families that are perfectly adapted to the requirements of modern industrial and household applications, combining high-performance and low cost.

This development is based on an MC68HC908MR8 microcontroller, a member of the M68HC08 Family. The MC68HC908MR8 incorporates a fault tolerant and flexible 6-channel, 12-bit pulse-width modulator (PWM) designed to support center and edge-aligned modes with automatic deadtime insertion and patented deadtime compensation capability.

Write-once protection of key configuration parameters further enhances motor and consumer safety, the MC68HC908MR8 is appropriate for cost and space conscious applications including smart appliances, blowers, fans, refrigeration compressors, office automation products, and electric lawn equipment.

Refer to **Figure 1-1** for a block diagram of the MC68HC908MR8. **Table 1-1** summarizes the MC68HC908MR8 peripherals and memory.

The MC68HC908MR8 is a member of the low-cost, high-performance M68HC08 Family of 8-bit microcontroller units (MCU). The M68HC08 Family is based on the customer-specified integrated circuit (CSIC) design strategy. All MCU's in the family use the enhanced M68HC08 central processor unit (CPU08) and are available with a variety of modules, memory sizes and types, and package types. The central processor unit can address 64 Kbytes of memory space.

Table 1-1. MC68HC908MR8 Peripherals and Memory

RAM (Bytes)	FLASH (Bytes)	Timer	I/O	Serial	A/D	PWM	Operating Voltage	Maximum Bus Frequency
256	8 K	2-ch + 2-ch 16-bit IC, OC, or PWM	14	SCI	4-ch to 7-ch 10 bit	6-ch 12 bit	5.0 V	8.0 MHz

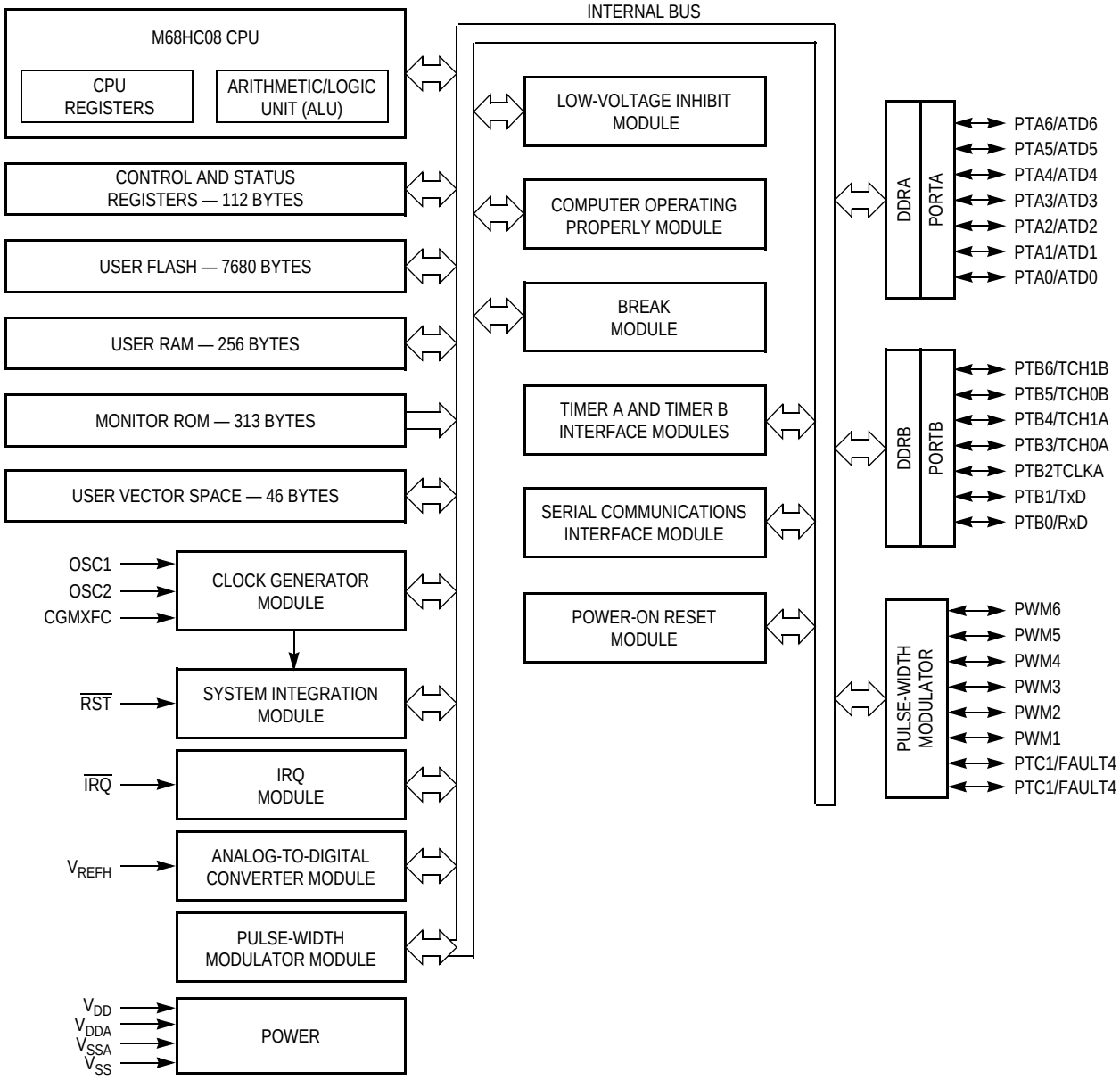


Figure 1-1. MC68HC908MR8 Block Diagram

Features of the MC68HC908MR8 include:

- High-performance M68HC08 architecture
- Fully upward-compatible object code with M6805, M146805, and M68HC05 Families
- 8-MHz internal bus frequency
- 8 Kbytes of on-chip FLASH
- On-chip programming firmware for use with host PC
- On-chip random-access memory (RAM) 256 bytes
- 12-bit, 6-channel center-aligned or edge-aligned PWMMC
- Serial communications interface module (SCI)
- Two 16-bit, 2-channel timer interface modules (TIMA and TIMB)
- Eight high current sink and source pins (PTA1/ATD1, PTA0/ATD0, PTB6/TCH1B, PTB5/TCH0B, PTB4/TCH1A, PTB3/TCH0A, PTB2/TCLKA, and PTB1/TxD)
- Clock generator module (CGM)
- Digitally filtered low-voltage inhibit (LVI), software selectable for ±5 percent or ±10 percent tolerance
- 10-bit, 4- to7-channel analog-to-digital converter (ADC)
- System protection features:
  - Optional computer operating properly (COP) reset
  - Low-voltage detection with optional reset
  - Illegal opcode detection with optional reset
  - Illegal address detection with optional reset
- Fault detection with optional PWM disabling
- Available packages:
  - 32-pin low-profile quad flat pack (LQFP)
  - 28-pin dual in-line package (PDIP)
  - 28-pin small outline package (SOIC)

- Low-power design, fully static with stop and wait modes
- Break (BRK) module allows single breakpoint setting during in-circuit debugging
- Master reset pin and power-on reset (POR)

Features of the CPU include:

- Fully upward, object-code compatibility with M68HC05 Family
- 16-bit stack pointer with stack manipulation instructions
- 16-bit index register with X-register manipulation instructions
- 8-MHz CPU internal bus frequency
- 64-Kbyte program/data memory space
- Sixteen addressing modes
- Memory-to-memory data moves without using the accumulator
- Fast 8-bit by 8-bit multiply and 16-bit by 8-bit divide instructions
- Enhanced binary-coded decimal (BCD) data handling
- Modular architecture with expandable internal bus definition for extension of addressing range beyond 64 Kbytes
- Low-power stop and wait modes

The MC68HC908MR8 PWM module can generate three complementary PWM pairs or six independent PWM signals. These PWM signals can be center-aligned or edge-aligned.

A 12-bit timer PWM counter is common to all six channels. PWM resolution is one clock period for edge-aligned operation and two clock periods for center-aligned operation. The clock period is dependent on the internal operating frequency (f_{op} of the MCU) and a programmable prescaler.

The highest resolution for edge-aligned operation is 125 ns (f_{op} = 8 MHz). The highest resolution for center-aligned operation is 250 ns (f_{op} = 8 MHz).

When generating complementary PWM signals, the module features automatic deadtime insertion to the PWM output pairs.

1.4 MC68HC908MR8 Pulse-Width Modulator

The pulse-width modulator module (PWMMC) resident on the MC68HC908MR8 is specifically designed to provide pulse-width modulated outputs to drive a power stage connected to a dc servo, brushless dc, or 3-phase ac motor system. The PWMMC module can be partitioned and configured in several ways, depending on the specific motor control application. **Figure 1-2** shows a block diagram of the PWMMC module and is referenced throughout this explanation of the PWMMC generator.

Features of the PWM include:

- Three complementary PWM pairs or six independent PWM signals
- Complementary mode features include:
  - Deadtime insertion
  - Separate top/bottom pulse-width correction via current sensing or programmable software bits
- Edge-aligned PWM or center-aligned PWM signals
- PWM signal polarity
- 20-mA current sink capability on all PWM outputs
- Manual PWM output control through software
- Programmable fault protection.

One of the most important features of the PWMMC is its ability to “shut itself down” when a system fault is detected. When dealing with a system that potentially could have hundreds of amps of peak current, reacting to faults such as Overcurrent or Overvoltage conditions is an absolute necessity. Fault protection is discussed first. Then, we will work our way from the outputs of the PWM inward.

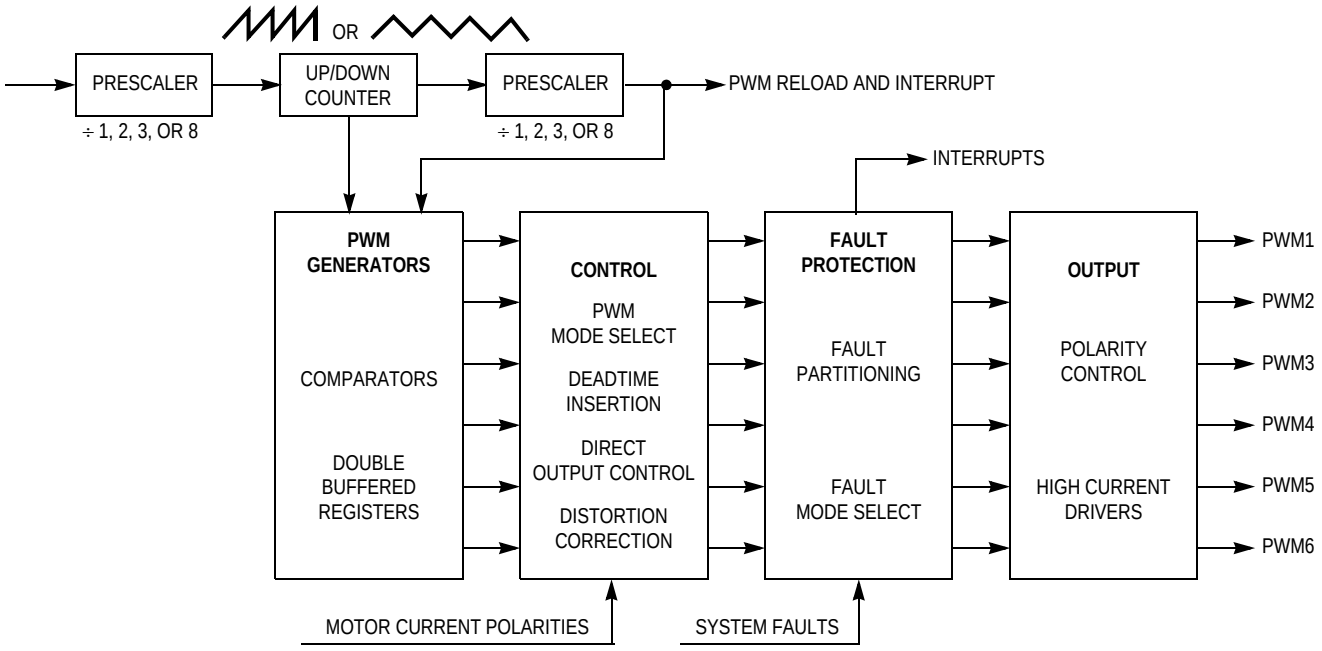


Figure 1-2. PWMMC Module Block Diagram

The six outputs of the PWMMC generator can be configured as individual pulse-width modulated signals where each output can be controlled as an independent output. Another option is to configure the outputs in pairs, with the outputs complementary or not, so driving complementary top and bottom transistors on a power stage becomes an easy task. The outputs of the PWMMC are capable of sinking up to 20 mA. That drive capability allows for direct drive of optocouplers without the need of additional drivers.

To prevent erroneous signals from being output from the PWMMC module while loading new values, the bulk of the registers are double buffered. New output is inhibited until the load okay (LDOK) bit in the PWM control register is set indicating that it is okay to output the new values.

1.4.1 Fault Protection

Conditions can arise in the external drive circuitry, requiring that the PWM signals become inactive immediately. These conditions include Overcurrent, Overvoltage, Overtemperature, or other error conditions. Upon detection of a fault, the two fault input pins on the MC68HC908MR8's PWMMC module can be configured to react in a number of different ways.

Each fault input has its own interrupt vector. In all fault conditions, the output of the PWM generator is forced to a known inactive state. A number of fault control and recovery options are available to the systems architect. In some cases, it may be desirable to selectively disable PWM(s) solely with software. Manual and automatic recovery mechanisms are available that allow certain acceptable fault situations to occur, such as starting a motor and using a fault input to limit the maximum startup current. The fault inputs can be partitioned if the MC68HC908MR8 is used to control multiple motors.

1.4.2 PWM Output Alignment

Depending on the system design, there is a choice between edge- or center-aligned PWM signals output from the MC68HC908MR32's PWM generator. The PWM counter uses the value in the timer modulus register to determine its maximum count. In center-aligned mode, a 12-bit up/down counter is used to create the PWM period. The PWM resolution in center-aligned mode is two clock periods (highest resolution is 250 ns at a processor speed of 8 MHz). The PWM period will be equal to:

[(Timer modulus) x (PWM clock period) x 2]

In edge-aligned mode, a 12-bit up-only counter is used to create the PWM period. Therefore, the PWM resolution in edge-aligned mode is one clock (highest resolution is 125 ns at a processor speed of 8 MHz). Again, the timer modulus register is used to determine the maximum count. The PWM period will be equal to:

[(Timer modulus) x (PWM clock period)]

1.4.3 PWM Counter Timebase

To permit lower PWM frequencies, a prescaler is provided which will divide the PWM clock frequency by 1, 2, 4, or 8. This prescaler is buffered and will not be used by the PWM generator until the LDOK bit located in a PWM control register is set and a new PWM reload cycle begins.

1.4.4 PWM Load Operations

When generating sine waves to a motor, an interrupt routine is typically used to step through a sine table located in FLASH memory, scale that sine value, and output the result to the system from the PWM generator. The rate at which the sine table is scanned can be derived from an interrupt from the PWM generator. The PWM module can be programmed to provide an interrupt rate of every 1, 2, 3, or 8 PWM reload cycles.

1.4.5 Direct Output Control

In some cases, the user may desire to bypass the PWM generator and directly control the PWM outputs. A mechanism exists to disconnect the PWM generator from its outputs and directly control the six PWM outputs. When this mode is used, the PWM generator continues to run; however, it's normal PWM output is disabled as it is overridden by direct output.

1.4.6 Deadtime Insertion

When the PWM generator is used in complementary mode, automatic deadtime insertion can be provided to prevent turning on both top and bottom inverter transistors in the same phase leg at the same time. When controlling dc-to-ac inverters, the top and bottom PWMs in one pair must never be active at any given time.

CAUTION: *If the top and bottom transistors are turned on simultaneously, large currents will flow through the two transistors as they attempt to discharge the bus supply voltage. The transistors could be weakened or destroyed.*

Simply forcing the two PWMs to be inversions of each other is not always sufficient. Since a time delay is associated with turning off the transistors in the motor drive, there must be a "deadtime" between the deactivation of one PWM power transistor and the activation of the opposite transistor in a top and bottom pair. Deadtime can be specified in the deadtime write-once register. This 8-bit value specifies the number of CPU clock cycles to use for the deadtime.

1.5 Brief Overview to Brushless DC Motors

A brushless dc motor is a rotating electric machine where the stator is a classic 3-phase stator like that of an induction motor and the rotor has surface-mounted permanent magnets. There are no brushes on the rotor and the commutation is performed electronically at certain rotor positions. The stator is usually made from magnetic steel sheets. The stator phase windings are inserted in the slots (distributed winding) as shown on **Figure 1-3**.

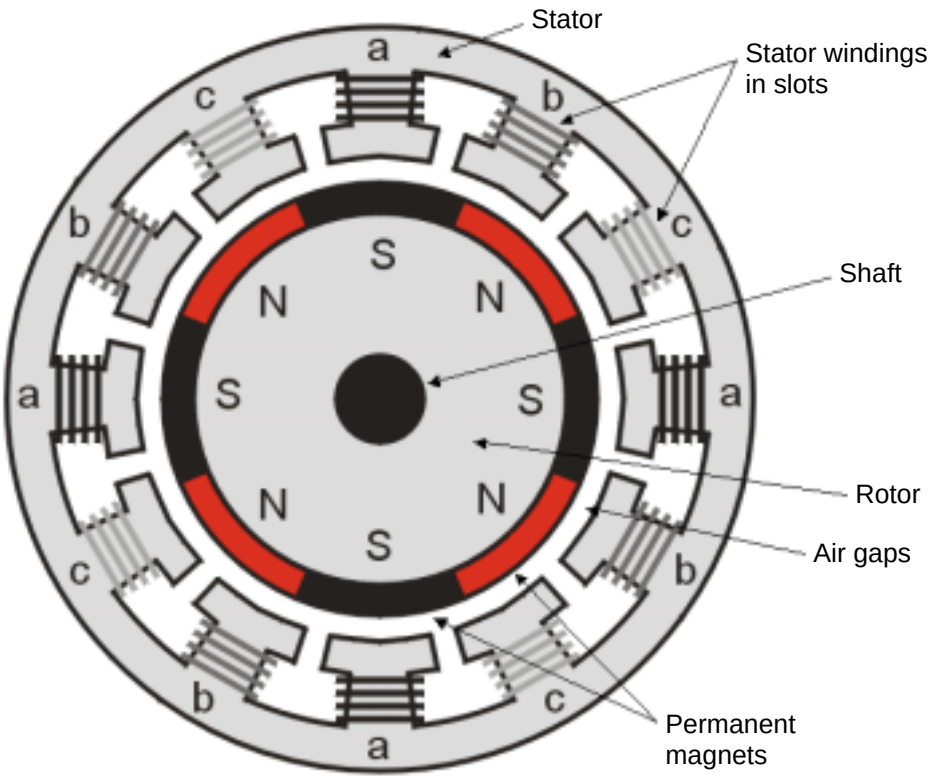


Figure 1-3. BLDC Motor – Cross Section

Brushless dc motors are named in different ways:

- Permanent magnet synchronous motors
- Brushless permanent magnet
- Permanent magnet ac motors, etc.

A BLDC motor is equivalent to an inverted dc commutation motor, where the magnet rotates while the conductors remain stationary. In the dc commutation motor, the commutator and brushes reverse the current polarity. But, in the brushless dc motor, a power transistor (which must be switched in synchronization with the rotor position) performs the polarity reversal. The BLDC motor often has either internal or external position sensors to sense actual rotor position so that synchronization can be performed.

The motor can have more than one pole-pair per phase. The pole-pair per phase defines the ratio between the electrical revolution and the mechanical revolution. For example, the BLDC motor shown in **Figure 1-3** has four pole-pairs per phase; which leads to four electrical revolutions; per one mechanical revolution.

Advantages of the brushless dc motors are:

- No electrical noise due to brushes and commutator
- No tachometer needed for speed control
- High starting torque and high no load speed
- Good power output to size ratio
- Higher efficiency than ac induction motors
- Reversible
- Precise speed control
- Variable speed
- Oil-less operation
- Rapid acceleration and deceleration
- Very low torque ripple

The presented application uses three Hall effect sensors to sense actual position. The Hall effect sensors' signals together give the six output values. These outputs are read by the microcontroller and the corresponding output voltage is generated by PWM outputs, as shown in **Figure 1-4**.

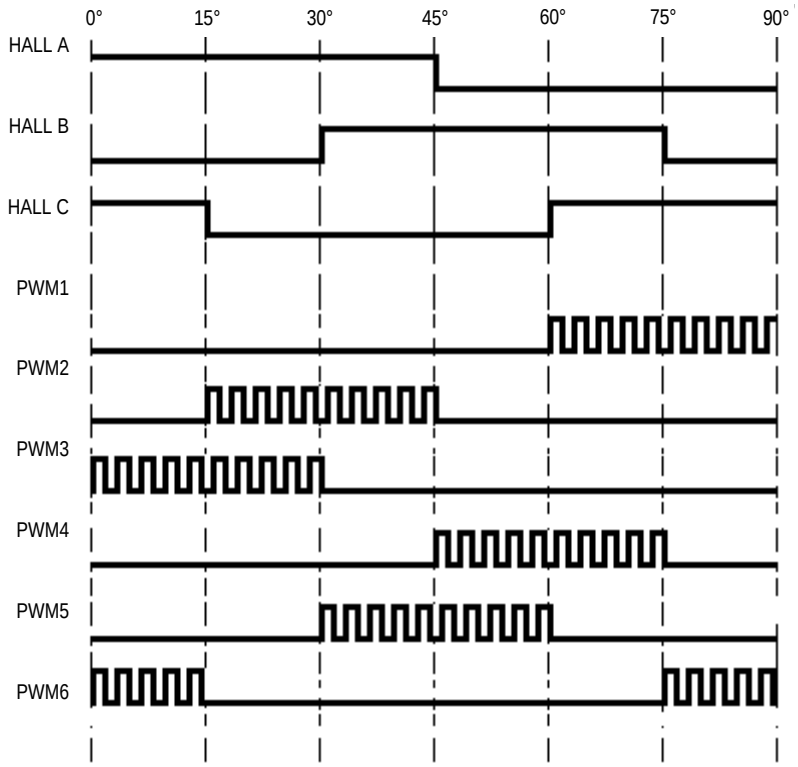


Figure 1-4. BLDC Motor Commutation Signals

These six PWM outputs are direct inputs to the 3-phase inverter. The motor windings are connected to the inverter. The three Hall effect sensors are connected to independent input capture channels of the microcontroller. See **Figure 1-5**.

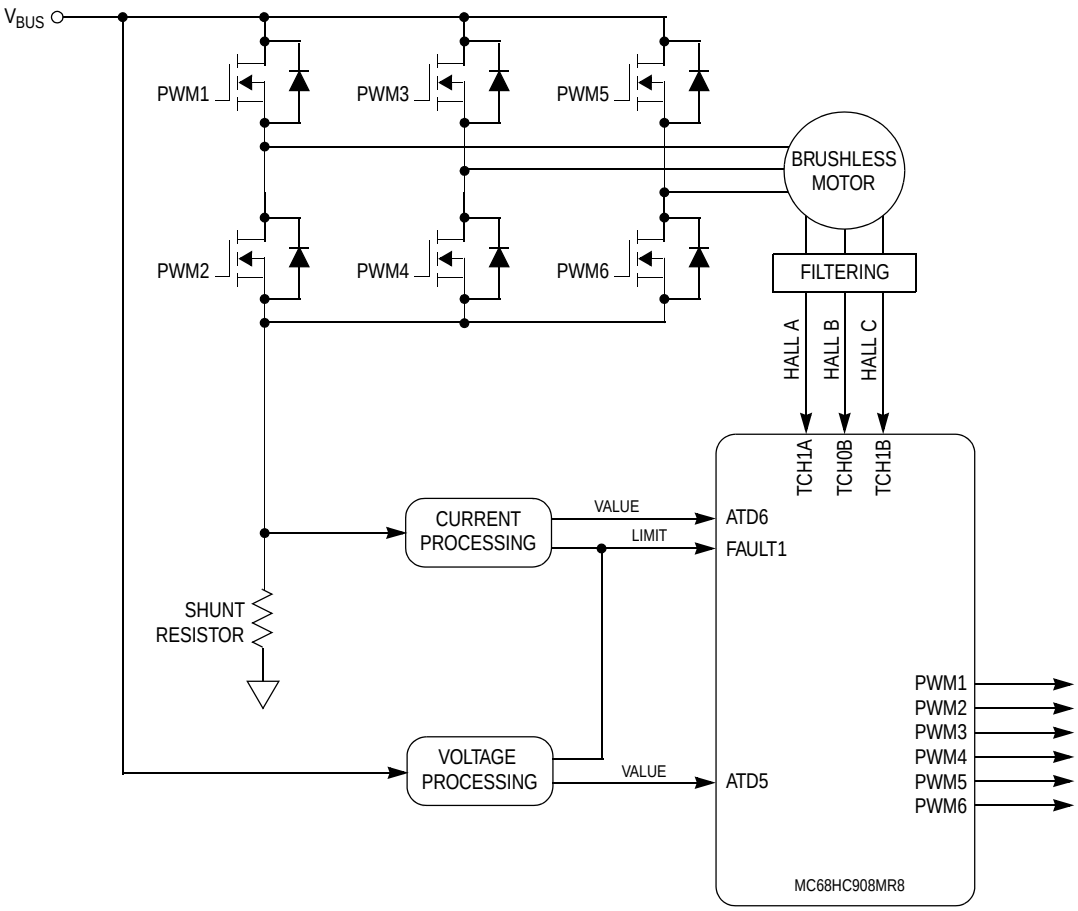


Figure 1-5. BLDC Motor Controller

1.6 Washing Machine Application's Overview

This reference design has many possible applications and can be easily reconfigured to suit industrial or appliance needs. The provided source code example emulates a basic washing machine as discussed in the following subsections.

1.6.1 Movement Patterns of the Washer

In washing machines there is a trade-off between clothes washability and clothes damage. One important consideration in the design is the agitator movement in the washer. The agitator movement pattern is given by a look up table of desired speeds. This look up table could

follow different shapes, such as square, trapezoidal or sinusoidal shapes. That is why the reference speeds in this design are taken from a table, leaving the user to customize the movement and test different patterns. From a mechanical point of view, a sinusoid agitator movement has less clothes damage, due to the smooth movement of the washer.

1.6.2 Agitator Hits

When washing, there are two important design considerations on each hit of the agitator:

- One is the angular displacement of the agitator in each hit. Modifying the reference speeds curve and calculating the integral of the entire hit can change this displacement.
- The other parameter is the frequency at which the table of reference speeds is accessed, giving different hits per minute in the washer.

1.6.3 Software

The software for this reference design drives a brushless dc motor in the four quadrants, which means that the motor can be reversed without any need of stopping the motor first. This driver capability is very useful in washers because of the water inertia in the washing machine.

1.6.4 User's Menu

A user menu with a 16 x 2 character display and two push buttons was included in the reference design board. This menu provides useful information during operation.

1.6.5 Control Scheme

The closed loop control scheme becomes necessary in this application to have more robustness in the washer operation, such as load change, input voltage variations, or mechanical degradations.

1.6.6 Target Washer

The targeted washers for this application example are direct drive washing machines. These washers have the following advantages over the classic ones:

- No belts between the motor shaft and the agitator of the washer.
- Different speed ranges, allowing different patterns of agitator movement.
- Powerful microcontroller, which makes possible the implementation of digital controllers.

1.7 System Concept

The system is designed to drive a 4-pole 3-phase BLDC star connected motor with a 5 to 1 speed gearbox. The microcontroller runs the main control algorithm. According to the user interface input and feedback signals, it generates 3-phase PWM output signals for the motor inverter.

The system incorporates all of the application in one board. **Figure 1-6** shows the system concept, including the following hardware:

- On-board power supply
- Feedback network
- Three-phase inverter
- Microcontroller unit
- User interface
- Optoisolated RS-232 interface

The motor used for this application is based on a ½ HP BLDC and a maximum speed of 4000 rpm.

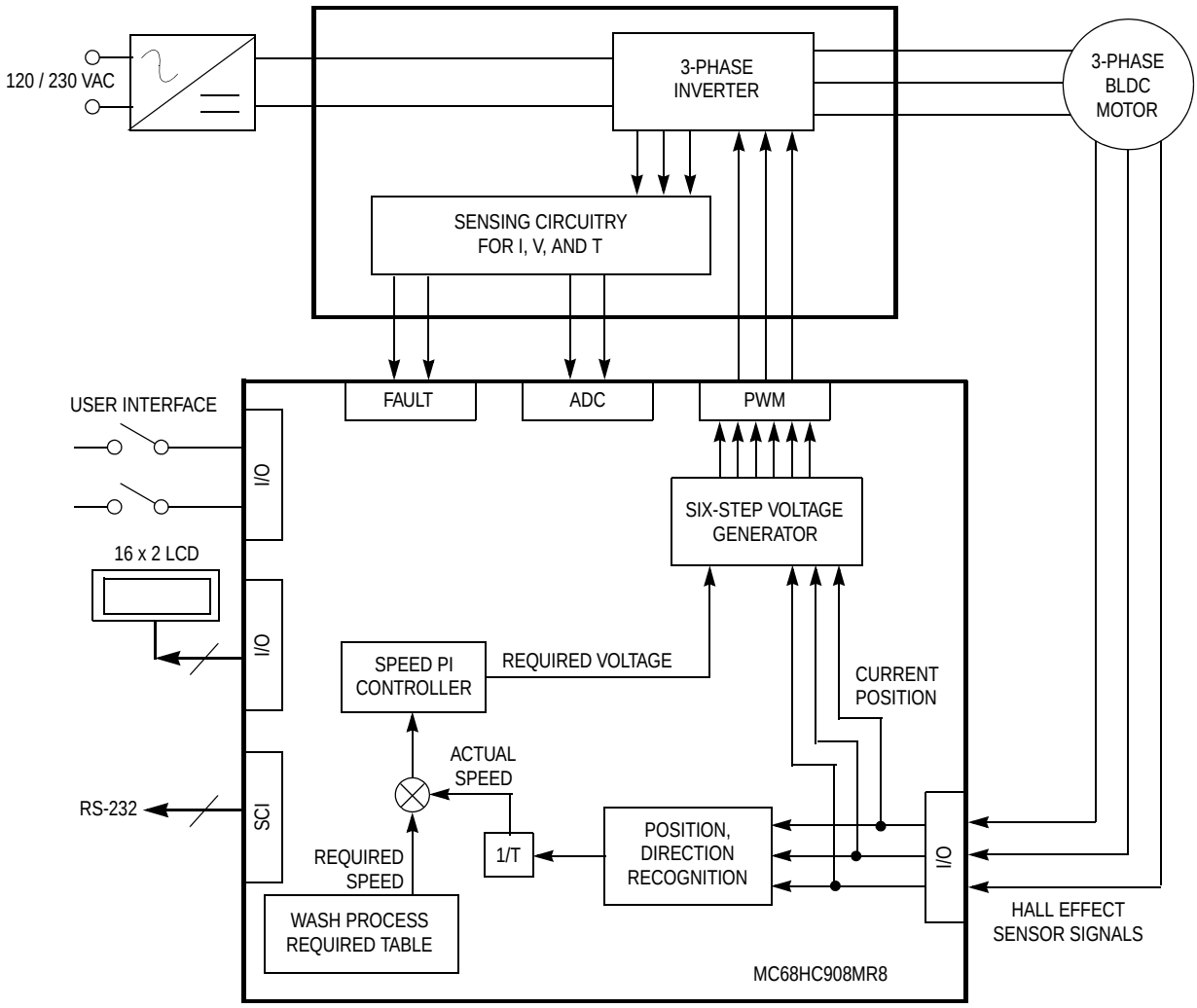


Figure 1-6. System Concept

The control process is as follows:

The state of the Hall sensor's inputs is periodically scanned, while the speed of the motor is measured on each new incoming edge from the Hall sensors. According to the user menu, the speed reference is calculated and controlled based upon the current and desired speed. The comparison between the actual speed and the desired speed generates a speed error. The speed error is brought to the speed PI controller that generates a new corrected applied voltage. There are two independent modules in software, one for commutating the motor and other for controlling the speed, which gives us a four-quadrant BLDC motor drive.

The Hall sensor signals are scanned independently of the speed controller. Each new incoming edge of any Hall sensor signal calls an interrupt routine, which calculates a new voltage shape, applied to the BLDC motor. This process is called commutation. The PWM transistors work in complementary mode, when the upper transistor is on, the lower transistor is off and vice versa.

1.8 Warnings

This reference board operates in an environment that includes dangerous voltages and rotating machinery.

Due to the high-voltage power stage operating directly from an ac line, oscilloscope grounds and power stage grounds are at different potentials, unless the oscilloscope is floating. Note that probe grounds and, therefore, the case of a floated oscilloscope, are subjected to dangerous voltages.

- Before moving scope probes, making connections, etc., you must turn off the main switch.
- Operation in lab setups that have grounded tables and/or chairs should be avoided.
- Wearing safety glasses, avoiding ties and jewelry, using shields, and operation by personnel trained in high-voltage lab techniques are advisable.
- Never turn on the board in running mode if it is not known if the code is downloaded.
- To reduce the cost of the board, optoisolation circuitry was not included: the microcontroller's ground is tied to a power stage ground. For this reason, special care must be taken when handling the board. Touching its components when it is turned on must be avoided.

1.9 Setup Guide

This board operates in two different modes: programming mode and running mode. Programming mode allows downloading code to the microcontroller. In running mode the microcontroller executes the downloaded code.

Out of the box conditions suppose the board is programmed with “BLDC CODE V1.s19”. Default position of Jumper JP1 is between 2 and 3 pins.

The board contains its own dc power supply for the power stage, besides a 15 Vdc regulated power supply and a 5 Vdc regulated power supply. The 15 Vdc and the 5 Vdc power supplies can be sourced by the dc power supply for power stage or by and external source of 18 Vdc at 200 mA. Input for this external source is the connector labeled J6. Selecting internal or external sourcing of 15 Vdc and 5 Vdc regulated power supplies, is done by means of switch S5. Then, if the user wants to use an external power supply, connect its terminals to connector J6 and slide the switch S5 to the position labeled “EXT”.

1.9.1 Programming Mode Setup

The following procedure describes programming mode setup. Before starting you must turn off the main switch. Auxiliary external power supply usage is recommended.

A PC computer is required having Metrowerks CodeWarrior Development Studio for HC08 Microcontrollers or PEMICRO PROG08SZ — FLASH programmer for M68HC908MR. The PC serial port baud rate should be set up at 9600 bps with no DTR signal.

The reference board works as a Class III — direct serial to target with MON08 serial port circuitry built in. The programmers software should be configured to match this.

To program the MCU perform the following steps:

1. Unplug the active cord.
2. Install a shorting jumper on pins 1 and 2 of JP1 to enter the microcontroller to monitor mode.
3. Connect a serial cable from a PC RS-232 serial port to the reference board's DB9 connector J5.
4. Connect external 18 Vdc power supply to J6 and slide switch S5 to position labeled "EXT". Or, plug ac line cord into jack J1 and turn on the main switch S4.
5. Continue with the FLASH programming procedure of the software used by the computer.

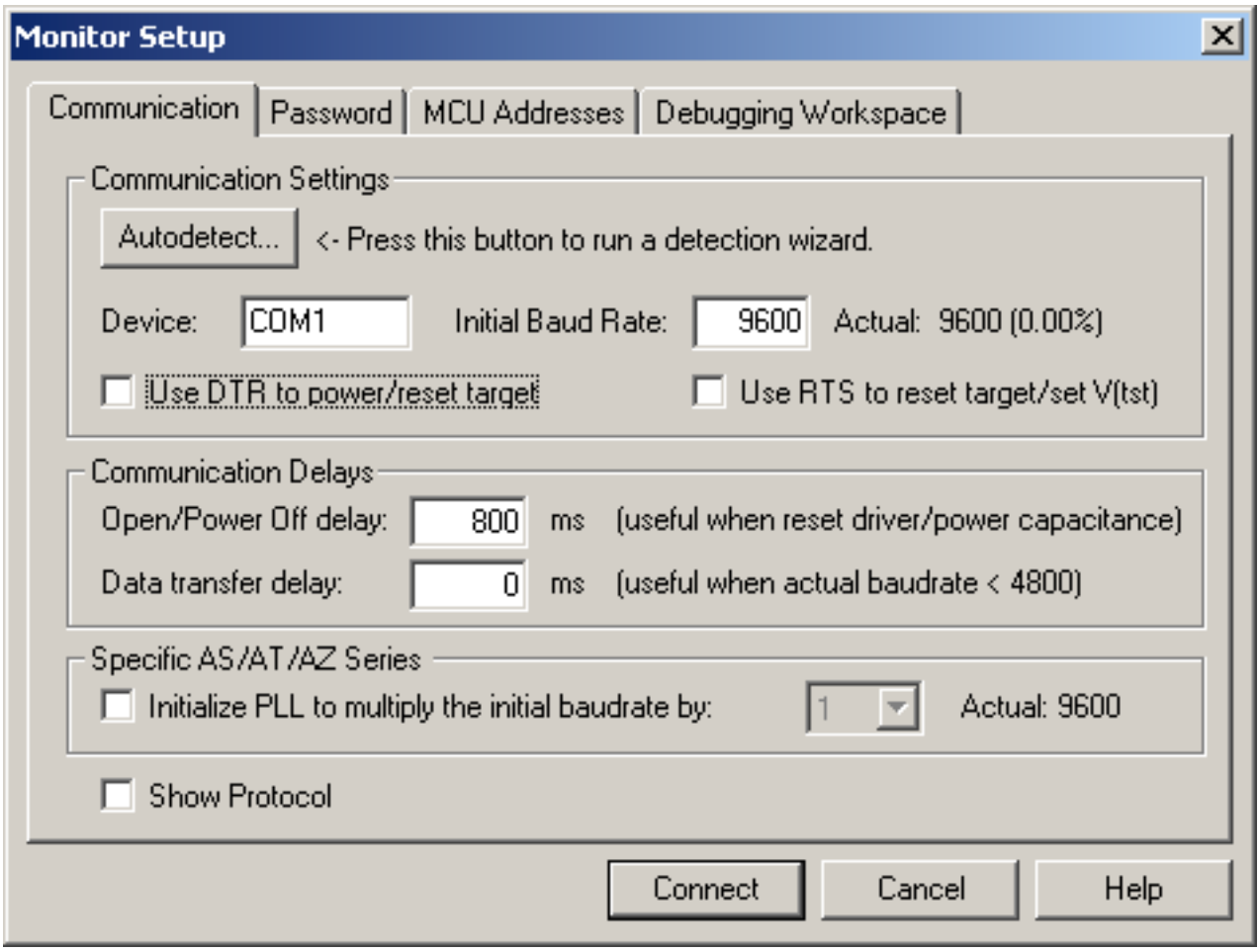


Figure 1-7. Monitor Setup

1.9.2 Running Mode Setup

Setup procedure for running mode is described here. This procedure supposes the microcontroller is programmed with a valid version of code. Before starting you must turn off the main switch S4.

1. Unplug the ac line cord.
2. Install a shorting jumper on pins 2 and 3 of JP1 to entry microcontroller to user mode.
3. Connect motor phase terminals to connector J2 according to labels near the connector.
4. Connect motor Hall sensor terminals to header J8 according to its label.
5. Slide switch S5 to position labeled “INT”.
6. Plug ac line cord into jack J1.
7. Turn on the main switch S4.

Alternatively to steps 5 through 7, you can connect an external 18 Vdc power supply to J6 and slide switch S5 to position labeled “EXT”.

The green LED, D21, must be turned on indicating that the 5 Vdc regulated power supply is working properly.

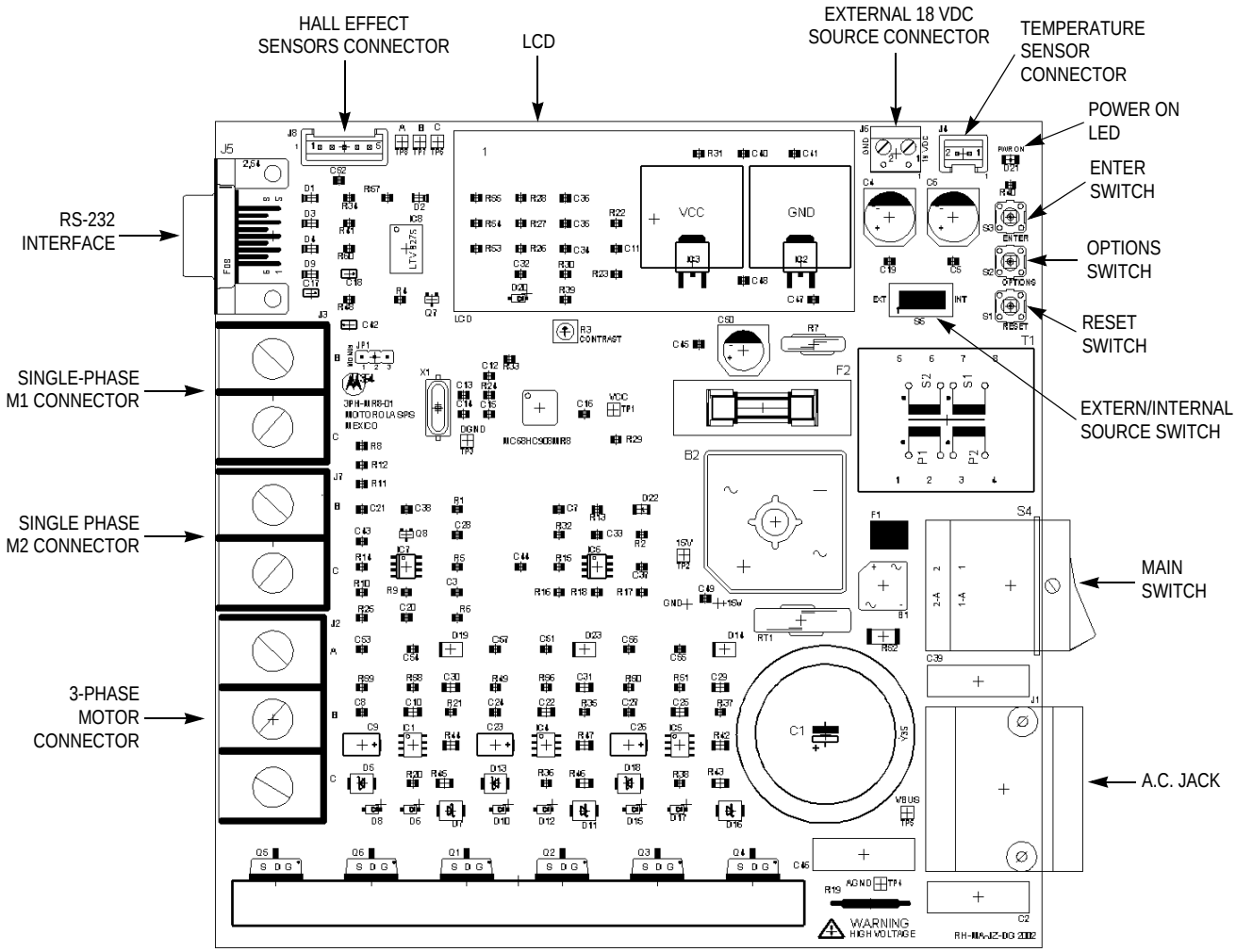


Figure 1-8. Board Layout

Section 2. Operational Description

2.1 Contents

2.2 Introduction

This section describes the electrical characteristics, user interfaces, and connections for the BLDC (brushless dc motor) control board.

2.3 Electrical Characteristics

The electrical characteristics in Table 2-1 and Table 2-2 apply to operation of the BLDC reference board at 25°C.

Table 2-1. Electrical Characteristics for 127 Vac Board Version

Inputs	Min	Typ	Max	Unit
AC input voltage	110	120	127	V RMS
AC input current	—	—	9	A RMS
Auxiliary dc input voltage	16	18	20	V
Auxiliary dc input current	—	—	150	mA
Minimum logic 1 input voltage	3.5	—	—	V
Maximum logic 0 input voltage	—	—	1.5	V
Motor output voltage	—	—	180	V RMS
Motor output current	—	—	8	A RMS
RS-232 connection speed	9504	9600	9696	Baud

Table 2-2. Electrical Characteristics for 230 Vac Board Version

Inputs	Min	Typ	Max	Unit
AC input voltage	210	220	230	V RMS
AC input current	—	—	9	A RMS
Auxiliary dc input voltage	16	18	20	V
Auxiliary dc input current	—	—	150	mA
Minimum logic 1 input voltage	3.5	—	—	V
Maximum logic 0 input voltage	—	—	1.5	V
Motor output voltage	—	—	320	V RMS
Motor output current	—	—	8	A RMS
RS-232 connection data rate	9504	9600	9696	Baud

2.4 User Interfaces

The BLDC board user interface consists of a 16 x 2 line character liquid crystal display (LCD), a LCD contrast potentiometer, a reset switch, a jumper, two push buttons, a slide switch, an indicator light-emitting diode (LED), and an optoisolated RS-232 interface.

- **D21: PWR ON —** D21, labeled PWR ON, illuminates when power is applied to the board.
- **JP1 —** Jumper JP1 is a 3-position jumper header. When shorted between position 1 and 2 the microcontroller is set to enter the HC08 monitor mode. For more detailed information, refer to the *MC68HC908MR8 Technical Data* (Motorola document order number MC68HC908MR8/D).
- **LCD —** A 16 characters per 2 lines liquid crystal display.
- **S5 —** S5 is a slide switch located on the top-right side of the board. It is used to select between external or internal input of power for 15 Vdc and 5 Vdc power supplies.
- **S1: RESET —** S1, the RESET switch, is a push button located near the right border of the board. It resets the microcontroller of the board.
- **S2: OPTIONS —** Push-button labeled OPTIONS scrolls all the washing machine cycles programmed.
- **S3: ENTER —** Push-button labeled ENTER selects the options showed in the LCD.
- **J5 —** An Optoisolated RS-232 interface, for monitor mode communication with a host computer, is available via DB-9 connector J5.

After turning on the board, when the board is programmed with code version “BLDC CODE V1.s19”, the first message displayed on the LCD is “BLDC WASH”.

By pressing the push button labeled OPTIONS (S2) the following menu options (defined in the following paragraphs) are displayed on the LCD:

- “Fault Occurred!!!”
- “Motor Stalled!!!”
- “BLDC WASH”
- “BLDC SPIN CW”
- “BLDC SPIN CCW”
- “SPEED DES +1980 CU +000”
- “BLDC STOP”

“Fault Occurred!!!” is a message display when an over voltage or over current has activated the FAULT1 input signal. The motor is stopped when this happens and the message is displayed.

“Motor Stalled!!!” is a message displayed when the motor is stalled.

“BLDC WASH” option is the typical washing cycle. The motor rotates in both directions, clockwise and counterclockwise. To produce this movement of the motor a defined look-up table of desired speeds is accessed continuously.

“BLDC SPIN CW” option makes the motor rotate in a clockwise direction. It is applied as a starting curve table and then the speed is maintained at a desired value programmed in software.

“BLDC SPIN CCW” option behaves similar to “BLDC SPIN CW” but in counterclockwise direction.

“SPEED” option displays the desired speed (‘DES’) programmed in software and the current speed (‘CU’), both in RPMs with a direction sign (‘+’ or ‘–’) corresponding to either clockwise or counterclockwise direction.

“BLDC STOP” option is intended to stop the motor.

When the push button labeled ENTER (S3) is pressed, the option showed on the LCD is executed. For example, if the option “BLDC SPIN CW” is displayed on the LCD and this button is pressed then the spin

clockwise cycle starts. Stopping a washing cycle is accomplished by selecting the option “BLDC STOP” by mean of OPTIONS button and then pressing the ENTER button.

2.5 Connectors Pin Descriptions

The following subsections describe the connector pins.

2.5.1 J1 — AC Jack

Table 2-3. AC Jack Connector (J1)

Pin Number	Name	Description s
1	Line	Line signal
2	Neutral	Neutral signal
3	GND	Chassis ground

2.5.2 J2 — 3-Phase Motor Connector

Table 2-4. 3-Phase Motor Connector (J2)

Pin Number	Name	Description
1	Phase A	Signal for phase A motor terminal
2	Phase B	Signal for phase B motor terminal
3	Phase C	Signal for phase C motor terminal

2.5.3 J3 — Single Phase Motor 1 Connector

Table 2-5. Single-Phase Motor 1 Connector (J3)

Pin Number	Name	Description
1	Phase B	Signal for phase B motor terminal
2	Phase C	Signal for phase C motor terminal

2.5.4 J4 — Temperature Sensor Connector

Table 2-6. Temperature Sensor Connector (J4)

Pin Number	Name	Description
1	V_{CC}	5 Vdc output signal
2	TEMPERATURE_SENSE	DC input signal from temperature sensor

2.5.5 J5 — RS-232 Interface Connector

Table 2-7. Optoisolated RS-232 DB-9 Connector (J5)

Pin Number	Name	Description
1	Unused	N/A
2	RxD	Data received by the PC from the control board
3	TxD	Data transmitted from the PC to the control board
4	DTR	Positive or negative voltage for communication
5	GND	Common ground reference
6	Unused	N/A
7	RTS	Negative or positive voltage for communication
8	Unused	N/A
9	Unused	N/A

2.5.6 J6 — External 18 Vdc Source Connector

Table 2-8. External 18 Vdc Source Connector (J6)

Pin Number	Name	Description
1	18 Vdc	18 Vdc signal from external source
2	GND	Common ground reference

2.5.7 J7 — Single Phase Motor 2 Connector

Table 2-9. Single-Phase Motor 2 Connector (J7)

Pin Number	Name	Description
1	Phase B	Signal for phase B motor terminal
2	Phase C	Signal for phase C motor terminal

2.5.8 J8 — Motor Hall Effect Sensor Connector

Table 2-10. Motor Hall Effect Sensors Connector (J8)

Pin Number	Name	Description
1	GND	GND
2	V_{CC}	5 Vdc output signal
3	HALL_A	Input signal from motor Hall sensor A
4	HALL_B	Input signal from motor Hall sensor B
5	HALL_C	Input signal from motor Hall sensor C

Section 3. Schematics and Bill of Materials

3.1 Contents

3.2 Schematics

A set of schematics for the BLDC (brushless dc motor) control board appears in **Figure 3-1** through **Figure 3-5**. Interrupted lines coded with the same letters are electrically connected.

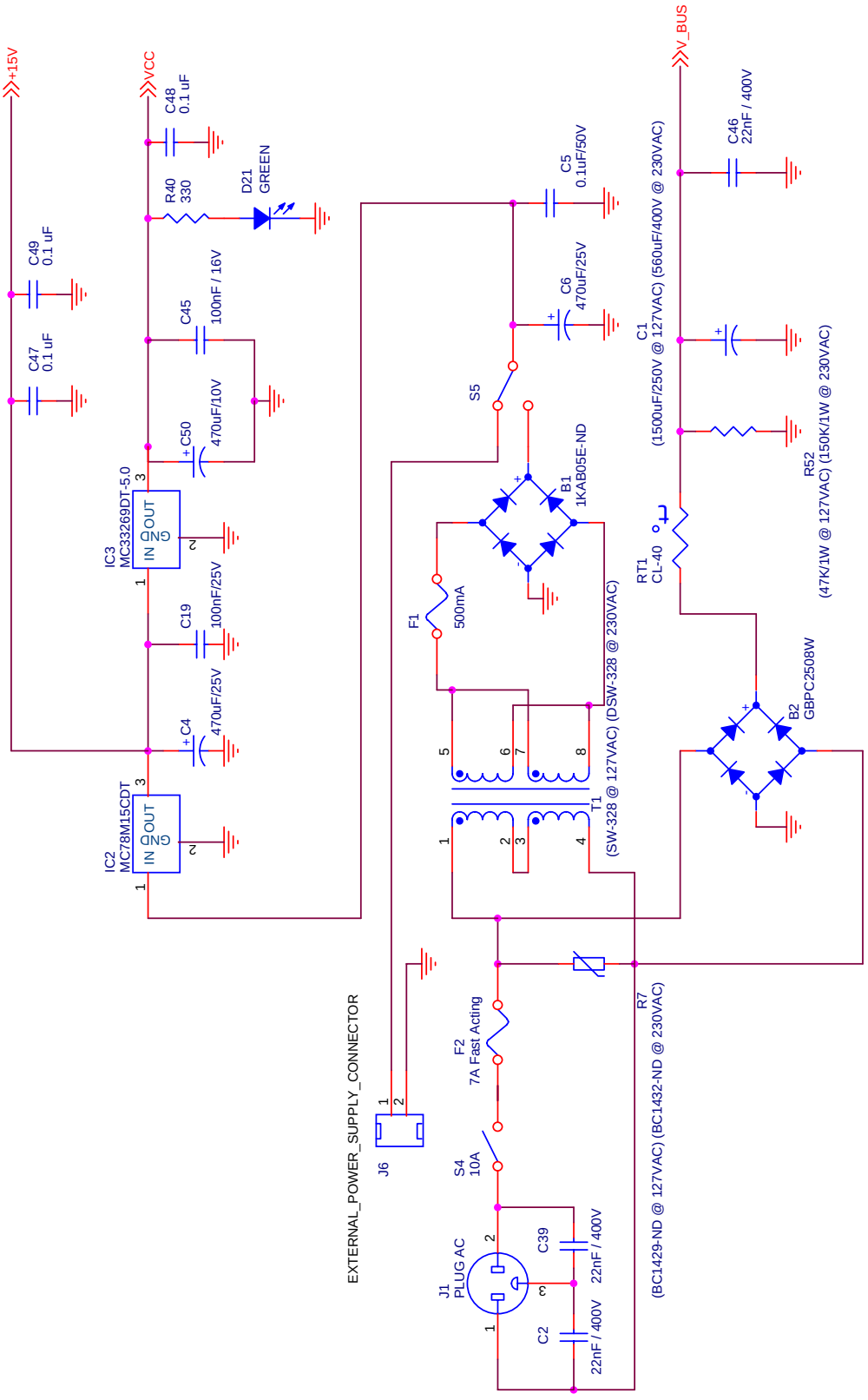


Figure 3-1. Power Supply

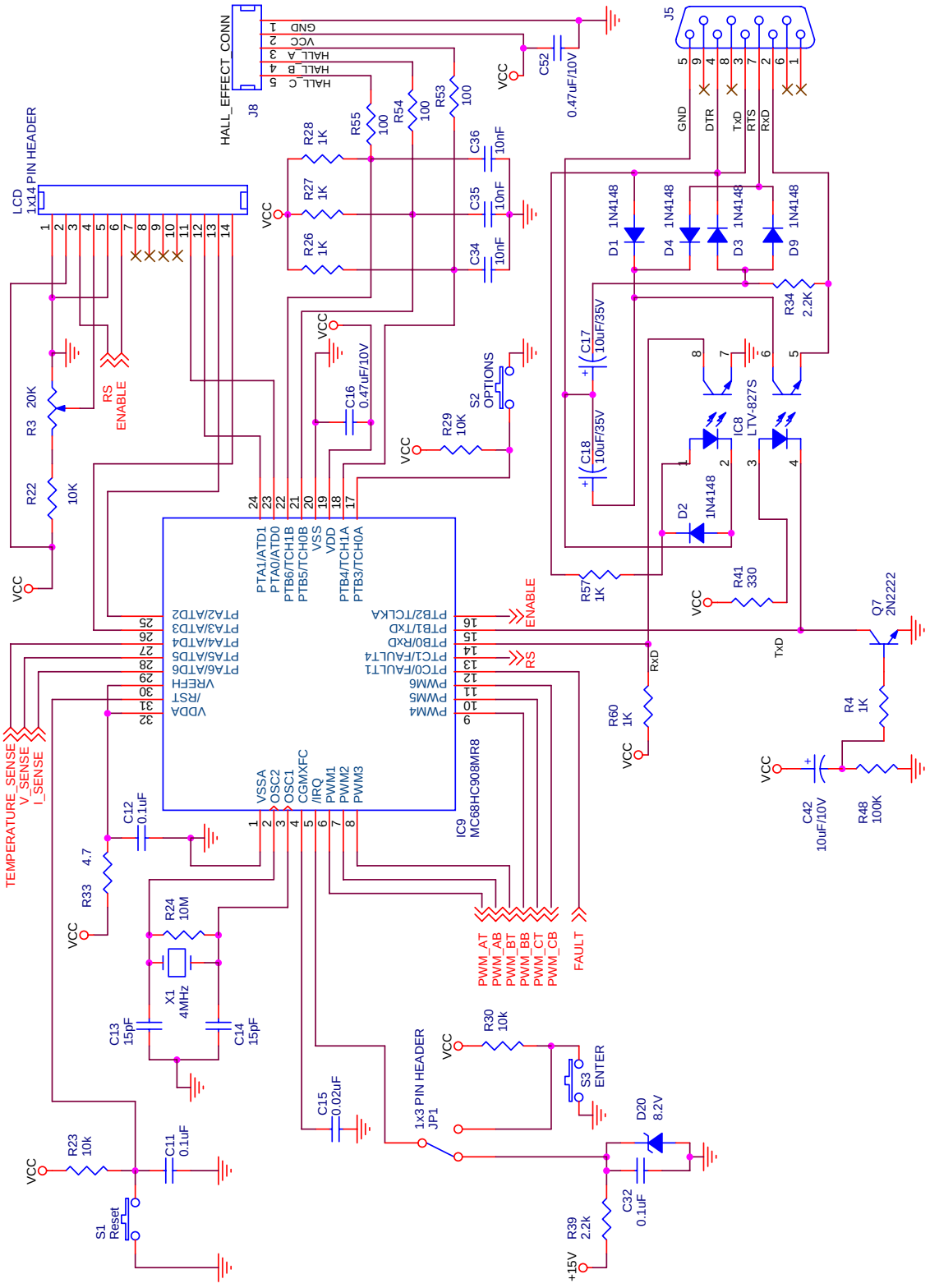


Figure 3-2. MCU

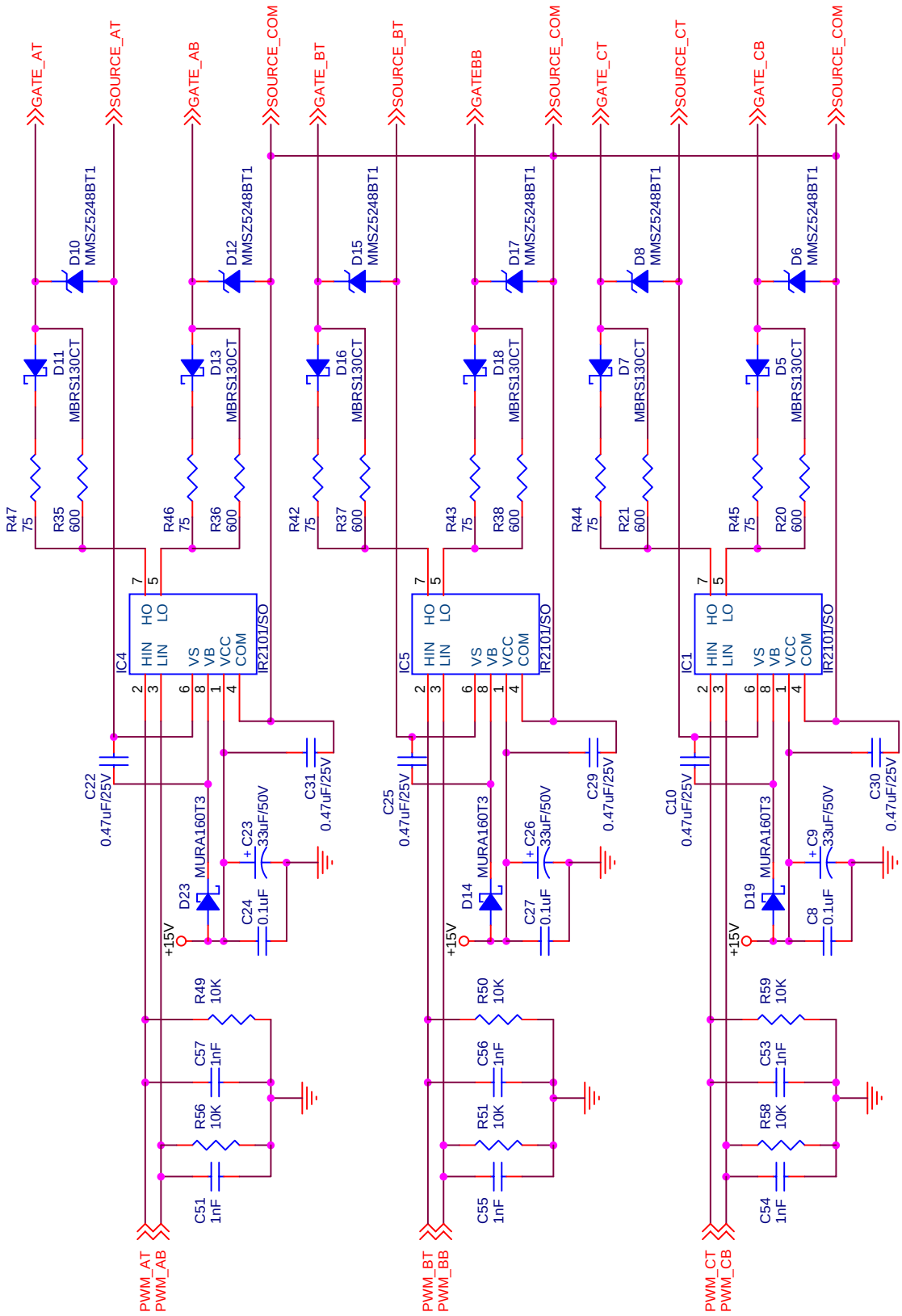


Figure 3-3. Gate Driver

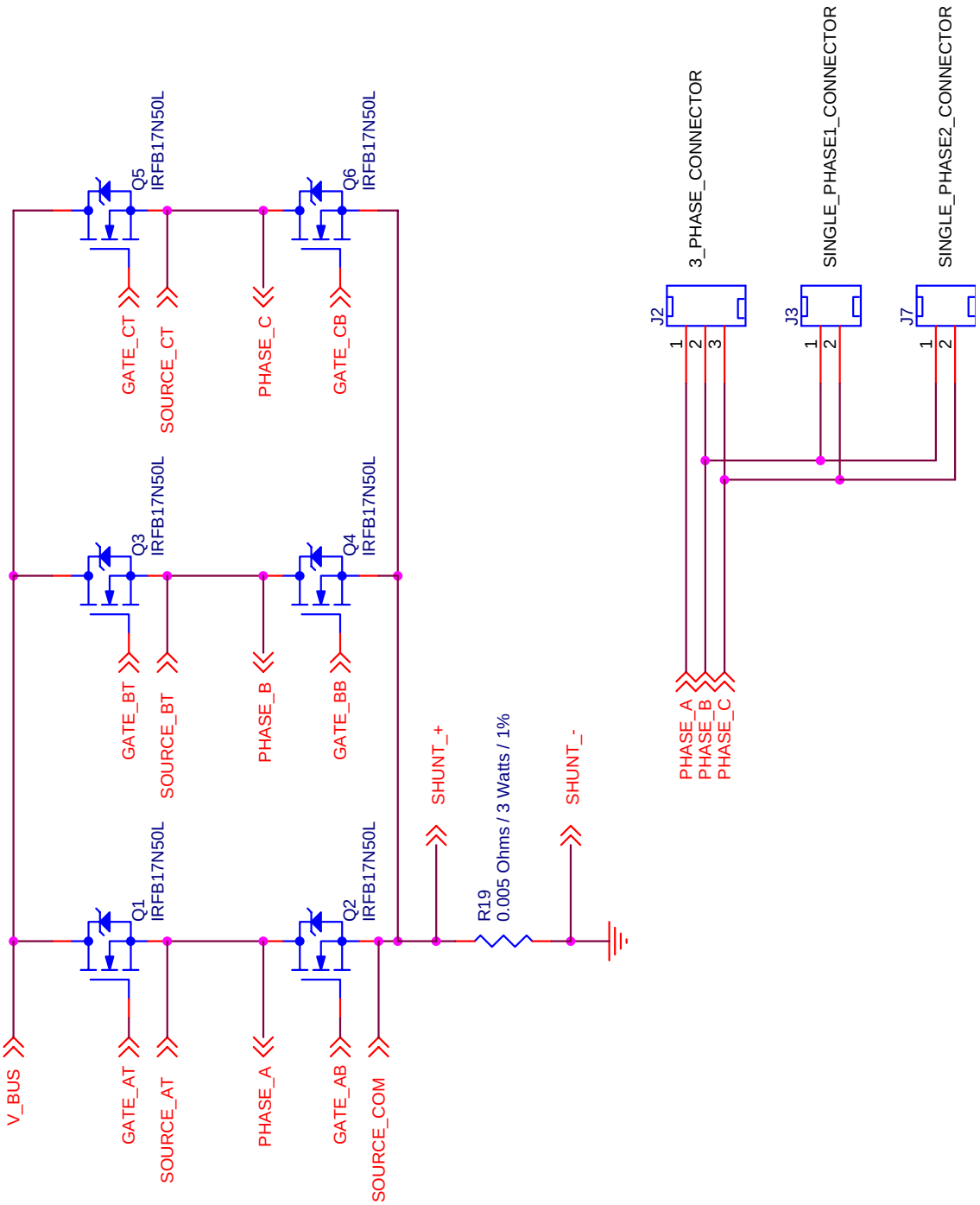


Figure 3-4. 3-Phase H-Bridge

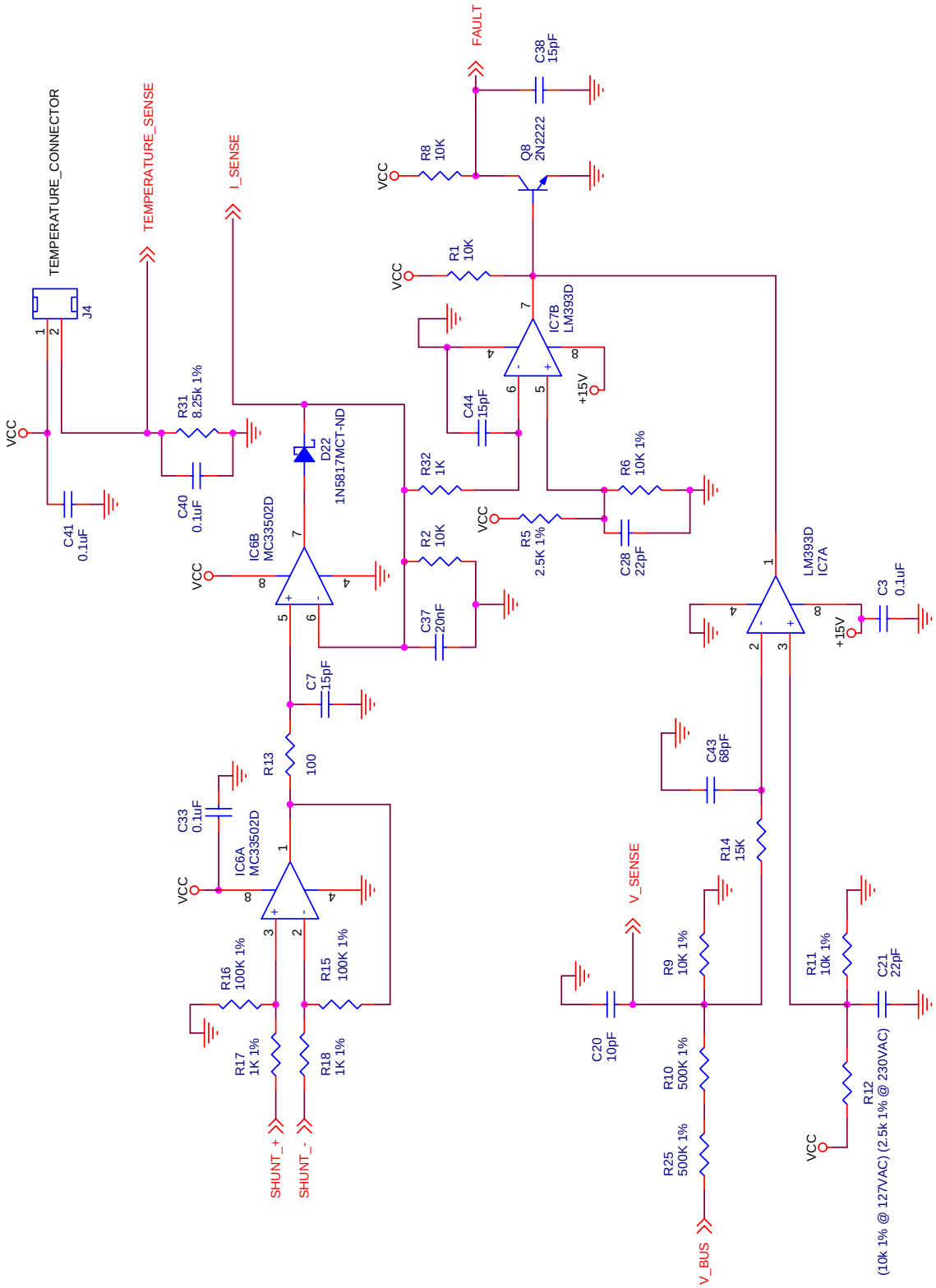


Figure 3-5. Current and Voltage Sense

3.3 Bill of Materials

The BLDC for Washing Machines Motor Controller Board Bill of Materials (BOM) 127 Vac version is described in **Table 3-1.** The 230 Vac board version has only five components different from 127 Vac version, **Table 3-2** shows those changes.

Table 3-1. Bill of Materials for 127 Vac Board (Sheet 1 of 5)

Qty	Value	Description	Label	Manufacturer	Part Number	Distributor	Distributor Part Number
			Diode Bridges				
1	1.2 A	1.2 A Rectifier	B1	International Rectifier	1KAB05E	Digikey	1KAB05E-ND
1	25 A	25 A Rectifier	B2	International Rectifier	GBPC2508W	Digikey	GBPC2508W-ND
			Capacitors				
1	1500 uF / 250v	Large Can Aluminum Electrolytic Capacitors	C1	Panasonic	ECOS2EP152EA	Digikey	P7413-ND
6	0.47 uF / 25v	Ceramic Capacitor (1206)	C10, C22, C25, C29, C30, C31	Panasonic - ECG	ECJ-3YB1E474K	Digikey	PCC1891TR-ND
5	15 pF	Ceramic Capacitor (0805)	C7, C13, C14, C38, C44	Yageo America	0805CG150J9B200	Digikey	311-1101-1-ND
2	0.02 uF	Ceramic Capacitor (0805)	C15, C37	Panasonic - ECG	ECJ-2VB1H223K	Digikey	PCC223BGCT-ND
1	0.47 uF/10v	Ceramic Capacitor (0805)	C16	Panasonic - ECG	ECJ-2YB1C474K	Digikey	PCC1818CT-ND
2	10 uF / 35v	CPOL-USCT3216	C17, C18	Panasonic - ECG	EEV-HA1V100WR	Digikey	PCE3299TR-ND
3	0.022 uF / 400v	Large Ceramic Capacitor	C2, C39, C46	Vishay / Sprague	225P22394XD3	Newark	47F143
1	10 pF	Ceramic Capacitor (0805)	C20	Yageo America	0805CG100J9B200	Digikey	311-1099-1-ND
2	22 pF	Ceramic Capacitor (0805)	C21, C28	Yageo America	0805CG220J9B200	Digikey	311-1103-1-ND
15	0.1 uF	Ceramic Capacitor (0805)	C3, C8, C11, C12, C19, C24, C27, C32, C33, C40, C41, C45, C47, C48, C49	Panasonic - ECG	ECJ-2VB1E104K	Digikey	PCC1828TR-ND
3	10 nF	Ceramic Capacitor (0805)	C34, C35, C36	Panasonic - ECG	ECJ-2VB1H103K	Digikey	PCC103BNCT-ND

Table 3-1. Bill of Materials for 127 Vac Board (Sheet 2 of 5)

Qty	Value	Description	Label	Manufacturer	Part Number	Distributor	Distributor Part Number
6	1 nF	Ceramic Capacitor (0805)	C51, C53, C54, C55, C56, C57	Yageo America	0805CG102J9B200	Digikey	311-1122-1-ND
2	470 uF / 25v	Electrolitic Capacitor	C4, C6	Panasonic - ECG	EEV-FK1V471Q	Digikey	PCE3464CT-ND
1	10 uF / 10v	Electrolitic Capacitor	C42	Panasonic - ECG	ECE-V1AA100NR	Digikey	PCE3125CT-ND
1	68 pF	Ceramic Capacitor (0805)	C43	Panasonic - ECG	ECJ-2VC1H680J	Digikey	PCC680CGCT-ND
1	0.1 uF / 50v	Ceramic Capacitor (0805)	C5	Panasonic - ECG	ECJ-2YB1H104K	Digikey	PCC1840CT-ND
1	470 uF / 10v	POL-CAPF	C50	Panasonic - ECG	EEV-FK1A471P	Digikey	PCE3392CT-ND
1	0.47 uF / 10v	Ceramic Capacitor (0805)	C52	Panasonic - ECG	ECJ-2YF1E474Z	Digikey	PCC1857CT-ND
3	33 uF / 50v	CPOL-USCT7343	C9, C23, C26	Kemet	T491X336K025AS	Newark	
			Diodes				
5	LL4148	LL4148	D1, D2, D3, D4, D9	Diodes Inc.	LL4148		
3	MURA160T3	SCHOTTKY_SMA	D14, D19, D23	ON	MURA160T3		
1	MMSZ5237BT1	Zener Diode 8.2 v	D20	ON	MMSZ5237BT1		
1	Green	SMD Green Led	D21	Stanley Electric Sales of America	DG1112H-TR	Digikey	404-1026-2-ND
1	1N5817MCT	Schottky - 20v / 1A	D22	Diodes Inc.	1N5817M	Digikey	1N5817MCT-ND
6	MBRS130LT	SCHOTTKY_SMB	D5, D7, D11, D13, D16, D18	International Rectifier	MBRS130LTR	Digikey	MBRS130LCT-ND
6	MMSZ5248BT1	Zener Diode 18 v	D6, D8, D10, D12, D15, D17	ON	MMSZ5248BT1	Diodes Inc	SMAZ18-13
			Fuses				
1	500 mA SMT	SM-FUSESM	F1	Bourns	MF-SM050		
1	10 Amp	FUSE22	F2	Schurterinc	OGD 0031.8231		
			Integrated Circuits				
3		IR2101S	IC1, IC4, IC5	International Rectifier	IR2101S	Digikey	IR2101S-ND
1	MC78M15CDT	Voltage Regulator 15v / 500mA	IC2	ON	MC78M15CDT		
1	MC33269DT-5.0	Voltage Regulator 5v / 800mA	IC3	ON	MC33269DT-5.0		
1	MC33502D	Dual Operational Amplifier	IC6	ON	MC33502D		

Table 3-1. Bill of Materials for 127 Vac Board (Sheet 3 of 5)

Qty	Value	Description	Label	Manufacturer	Part Number	Distributor	Distributor Part Number
1	LM393D	Low Offset Voltage Comparator	IC7	ON	LM393D		
1	LTV-827S	Optoisolator SMD	IC8	Lite-On Inc.	LTV-827S	Digikey	160-1369-5-ND
			Connectors				
1	AC_jack	AC Power Connector	J1	SCHURTER	GSP2.9213.13	Newark	32C1691
1	66503	66503	J2	MOLEX/WALDOM	66503	Newark	29B3093
2	6650202	6650202	J3, J7	MOLEX/WALDOM	66502	Newark	29B3092
1		S02P	J4	TYCO ELECTRONICS	640456-2	Newark	90F4250
1	FDB9	DB9 / Female connector	J5	CINCH	DEKL-9SAT-F	Newark	95F4126
1		W237-102	J6	TYCO ELECTRONICS	796949-2	Newark	34C9478
1		S05P	J8	TYCO ELECTRONICS	640456-5	Newark	90F5643
			Jumpers				
1		JP2E	JP1	SPC CONNECTORS	8431-0721	Newark	16N2602
			LCD				
1		LCD_OPTREXN	LCD	LUMEX	LCM-S01602DTR/A	Digikey	67-1779-ND
			Microcontroller				
1	HC908MR8	Microcontroller	MC68HC908MR8	Motorola	MC68HC908MR8		
			Transistors				
6	IRFPC40VH	Power Mosfet 500V 17A	Q1, Q2, Q3, Q4, Q5, Q6	International Rectifier	IRFB17N50L	Newark	33C4970
2	MMBT2222AL	NPN transistor 2N2222AL	Q7, Q8	ON	MMBT2222AL		
			Resistors				
16	10 K	Resistor (0805)	R1, R2, R8, R22, R23, R26, R27, R28, R29, R30, R49, R50, R51, R56, R58, R59	Yageo America	9C08052A1002FKHFT	Digikey	311-10.0KCCT-ND
2	500 k / 1%	Resistor (0805)	R10, R25	Yageo America	9C08052A4993FKHFT	Digikey	311-499KCCT-ND
1	10 k / 1%	Resistor (0805)	R11	Yageo America	9C08052A1002FKHFT	Digikey	311-10.0KCCT-ND

Table 3-1. Bill of Materials for 127 Vac Board (Sheet 4 of 5)

Qty	Value	Description	Label	Manufacturer	Part Number	Distributor	Distributor Part Number
4	100	Resistor (0805)	R13, R53, R54, R55	Yageo America	9C08052A1000FKHFT	Digikey	311-100CCT-ND
1	15 K	Resistor (0805)	R14	Yageo America	9C08052A1502FKHFT	Digikey	311-15.0KCCT-ND
2	100 K / 1%	Resistor (0805)	R15, R16	Yageo America	9C08052A1652FKHFT	Digikey	311-16.5KCTR-ND
2	1 K / 1%	Resistor (0805)	R17, R18	Yageo America	9C08052A1001FKHFT	Digikey	311-1.00KCCT-ND
1	.005 / 3w / 1%	Shunt Resistor	R19	IRC	OAR-3 0.005 1%	Future Electronics	
4	1K	Resistor (0805)	R4, R32, R57, R60	Yageo America	9C08052A1001FKHFT	Digikey	311-1.00KCCT-ND
6	600	Resistor (0805)	R20, R21, R35, R36, R37, R38	Yageo America	9C08052A6040FKHFT	Digikey	311-604CCT-ND
1	10 M	Resistor (0805)	R24	Yageo America	9C08052A1005FKHFT	Digikey	311-10.0MCCT-ND
1	8.25 K / 1%	Resistor (0805)	R31	Yageo America	9C08052A8251FKHFT	Digikey	311-8.25KCCT-ND
1	4.7	Resistor (0805)	R33	Yageo America	9C08052A4R70JLHFT	Digikey	311-4.7ACT-ND
2	2.2 K	Resistor (0805)	R34, R39	Yageo America	9C08052A2201FKHFT	Digikey	311-2.20KCCT-ND
2	330	Resistor (0805)	R40, R41	Yageo America	9C08052A3300FKHFT	Digikey	311-330CCT-ND
6	75 - 1/4 w	Resistor (1206)	R42, R43, R44, R45, R46, R47	Yageo America	9C12063A1200FKHFT	Digikey	311-120FCT-ND
1	100 K	Resistor (0805)	R48	Yageo America	9C08052A1003FKHFT	Digikey	311-100KCTR-ND
1	2.5 K / 1%	Resistor (0805)	R5	Yageo America	9C08052A2501FKHFT	Digikey	311-2.50KCCT-ND
1	47k / 1w	Resistor (2512)	R52	Panasonic - ECG	ERJ-1TYJ473U	Digikey	PT47KXCT-ND
3	10 K / 1%	Resistor (0805)	R6, R9, R12	Yageo America	9C08052A1002FKHFT	Digikey	311-10.0KCCT-ND
			Varistor				
1		Varistor 150v RMS	R7	BC Components	2322 594 51516	Digikey	BC1429-ND
			NTC				
1	CL40	Disc thermistor	RT1	NTC Thermistors	CL40		
			Potentiometer				
1	20 K	Trimmer	R3	Copal Electronics	ST4TA203	Digikey	ST4A203TR-ND
			Switches				
1	RESET	Push Button	S1	E-switch	TL59FF260Q	Newark	
2		Push Button	S2, S3	E-switch	TL59FF260Q	Newark	
1	CKDFA	Main Switch Power Supply	S4	C&K COMPONENTS	DF62J12S2APQF	Newark	91F4835
1		Slide Switch	S5	C&K COMPONENTS	CK1101M2S3CQE2		

Table 3-1. Bill of Materials for 127 Vac Board (Sheet 5 of 5)

Qty	Value	Description	Label	Manufacturer	Part Number	Distributor	Distributor Part Number
			Transformer				
1	328SW	Side-Winder Transformer	T1	Stancor	SW-328		
			Test Points				
1	VCC	Test Point - Vcc	TP1	Keystone Electronics	5000	Newark	52F7277
1	15V	Test Point - 15v	TP2	Keystone Electronics	5000	Newark	52F7277
1	DGND	Test Point - DGND	TP3	Keystone Electronics	5001	Newark	52F7278
1	AGND	Test Point - AGND	TP4	Keystone Electronics	5001	Newark	52F7278
1	VBUS	Test Point - VBUS	TP5	Keystone Electronics	5000	Newark	52F7277
1	C	Test Point - Hall Sensor C	TP6	Keystone Electronics	5002	Newark	52F7279
1	B	Test Point - Hall Sensor B	TP7	Keystone Electronics	5003	Newark	52F7280
1	A	Test Point - Hall Sensor A	TP8	Keystone Electronics	5004	Newark	52F7281
			Heat Sink				
1		Heatsink	U1	Aavid Thermalloy	780103B04500		
			Crystal				
1	4 MHz	4 MHz crystal	X1	CTS-Frequency Controls	ATS040SM	Digikey	CTX502-ND

Table 3-2. Bill of Material Changes for 230 Vac Board

Qty	Value	Description	Label	Manufacturer	Part Number	Distributor	Distributor Part Number
			Capacitor				
1	560 mF/400 V	Large Can Aluminum Electrolytic Capacitors	C1	Panasonic	ECOS2GP1561EA	Digikey	P6157-ND
			Resistors				
1	2.5 K/1%	Resistor (0805)	R12	Yageo America	9C08052A2501FKHFT	Digikey	311-2.50KCCT-ND
1	150 K/1 W	Resistor (2512)	R52	Panasonic – ECG	ERJ-1TYJ154U	Digikey	PT150KXCT-ND
			Varistor				
1		Varistor 250 V RMS	R7	BC Components	2322 594 52516	Digikey	BC1432-ND
			Transformer				
1	328 DSW	Dual Side-Winder Transformer	T1	Stancor	DSW-328		

Schematics and Bill of Materials

Section 4. Hardware Design Considerations

4.1 Contents

4.2 Introduction

The hardware for motor control developed for the reference design has the power output for the motor, and the microcontroller on the same board. In addition to the hardware that is needed to run the motor, a variety of feedback signals that facilitate control algorithm development are included.

4.3 Power Supply

The main power input to the board is through a power jack (J1). From this power input, V_BUS signal is generated. This voltage (V_BUS) is generated through a rectifier bridge (B2). To minimize the effects of the in-rush current when S4 is turned on, a NTC (RT1) was placed to slowly charge V_BUS capacitor (C1). When S4 is turned OFF, C1 is sometimes charged (depending on last system operation). To avoid any risk, a discharge resistor (R52) is connected in parallel to C1. See **Figure 4-1**.

NOTE: *There is also an Overvoltage (R7) and an Overcurrent (F2) protection.*

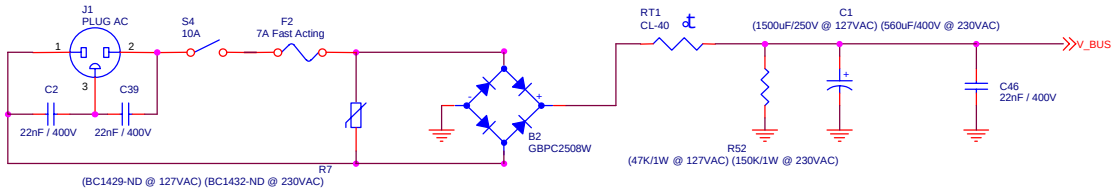


Figure 4-1. V_BUS Power Supply

From the line input jack (J1) the low voltage power supplies (5 Vdc and 15 Vdc) are derived. These power supplies are generated using voltage regulators (IC2 and IC3). To help developers vary V_BUS voltage using a variable transformer in J1 and also let them program the microcontroller without having the power-stage turned on (V_BUS), an alternate Vdc power supply can be connected (J6) to keep 5 Vdc and 15 Vdc on the board when varying AC voltage in J1. To enable this external power supply, S5 slide switch must be turned to “EXT” position. A green LED (D21) was included to show proper +5 Vdc power supply operation. See **Figure 4-2**.

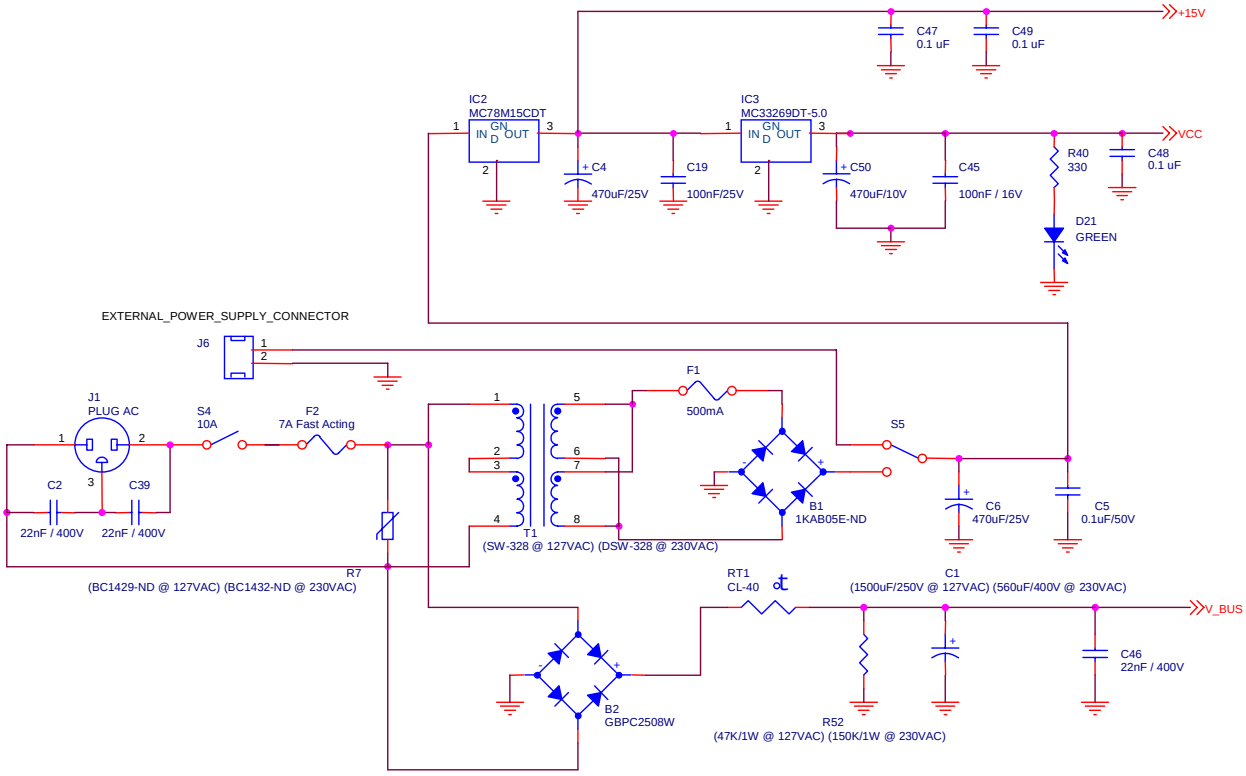


Figure 4-2. 15 Vdc and 5 Vdc Power Supplies

4.4 RS-232 interface and MON08 Hardware Interface

The board provides an RS-232 interface by the use of an optoisolator referenced at 5 Vdc voltage level (IC8). This topology lets the user program the microcontroller using the MON08 interface, and communicate via the RS-232 interface when operating in run mode. This topology also, allows operating the board ground at a different level than the PC (or RS-232 device), avoiding the risk of damaging the board or the PC. See **Figure 4-3**.

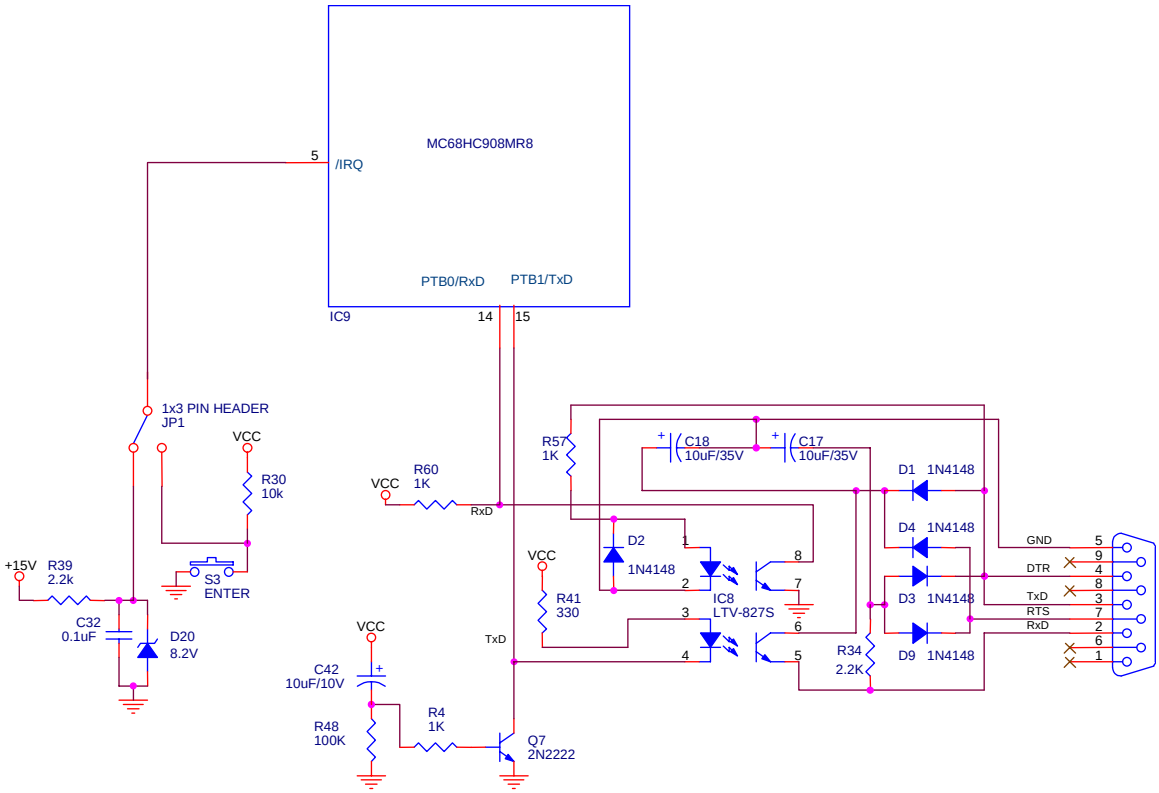


Figure 4-3. RS-232 and MON08 Interfaces

4.5 Clock Source

The board uses a 4.00-MHz crystal (X1) connected to microcontroller's oscillator inputs (OSC1 and OSC2). The MC68HC908MR8 uses its internal phase-locked loop (PLL) to multiply the input frequency in order to achieve its 8 MHz maximum operating frequency. See **Figure 4-4**.

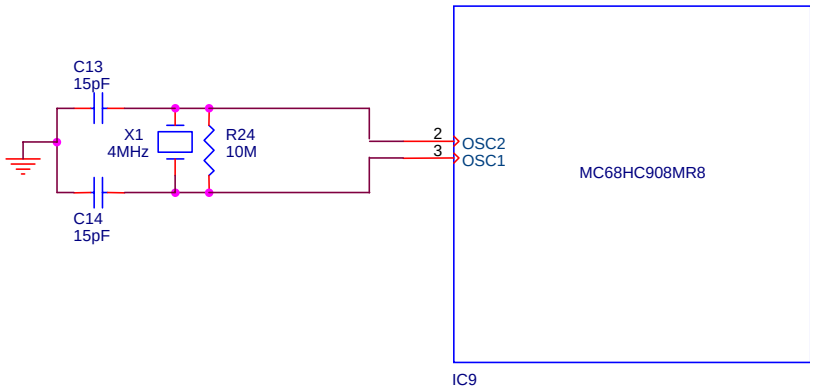


Figure 4-4. Clock Source

4.6 Hall-Effect Sensors Interface

The board contains a Hall-effect interface connected to the microcontroller's timer A (channel 1) and timer B (channel 0 and channel 1) port signals, TCH1A, TCH0B, and TCH1B. The circuit is designed to accept +5.0 V Hall-effect sensor inputs. Input noise filtering is supplied on the input path for the Hall-effect interface. **Figure 4-5** shows the hardware interface.

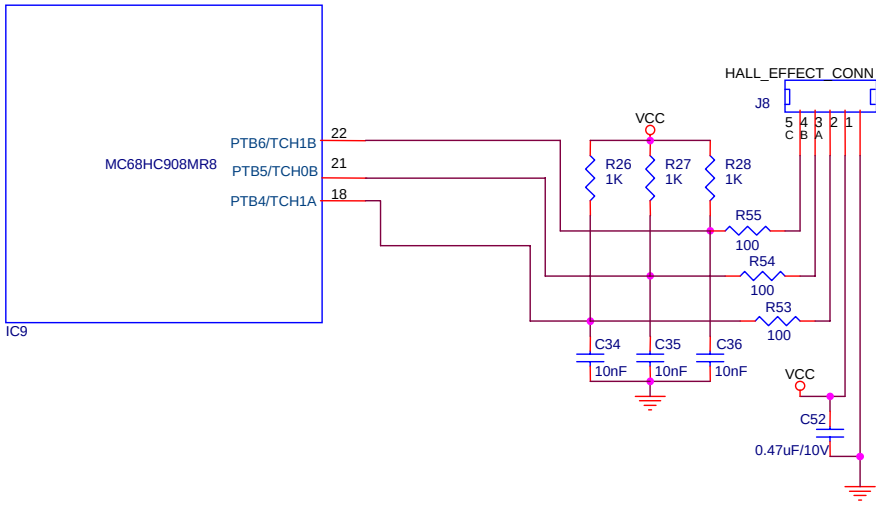


Figure 4-5. Hall-Effect Sensors Interface

4.7 LCD Interface

The board contains an LCD as main user interface feedback. The LCD contains an internal driver. The display is controlled and managed by the microcontroller through it's port signals. **Figure 4-6** shows the hardware interface.

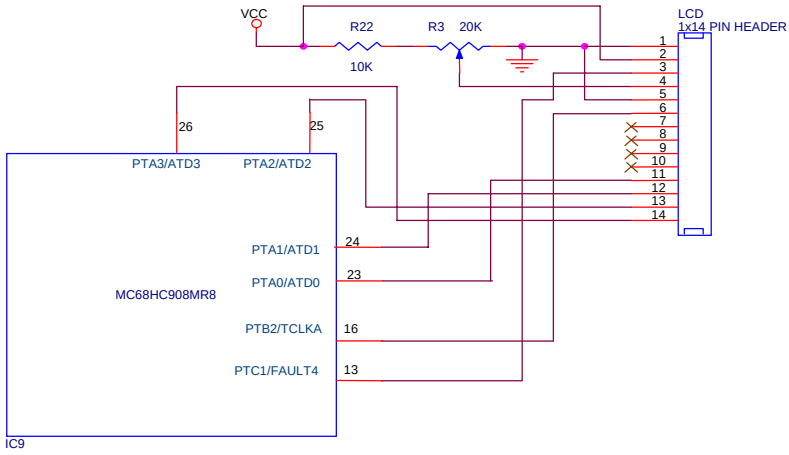


Figure 4-6. LCD Interface

4.8 Reset Button

The board contains a reset button (RESET). This button is directly connected to the microcontroller's reset pin which causes an external pin reset to the microcontroller. **Figure 4-7** shows the hardware interface.

Pulling the asynchronous $\overline{\text{RST}}$ pin low halts all processing. The PIN bit of the SIM reset status register (SRSR) is set as long as $\overline{\text{RST}}$ is held low for a minimum of 67 CGMXCLK cycles, assuming that neither the power-on reset (POR) nor the low-voltage inhibit (LVI) was the source of the reset. Refer to **Table 4-1** detailed information on PIN bit set timing and to **Figure 4-8** for the relative timing.

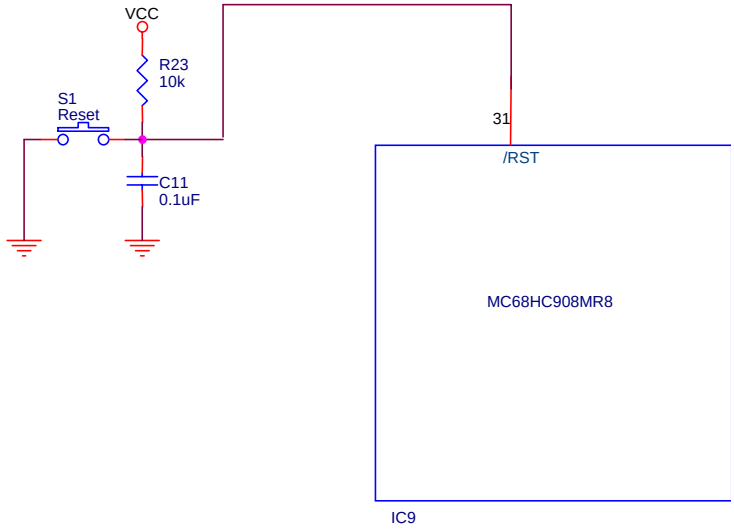


Figure 4-7. Reset Button

Table 4-1. PIN Bit Set Timing

Reset Type	Number of Cycles Required to Set PIN
POR/LVI	4163 (4096 + 64 + 3)
All Others	67 (64 + 3)

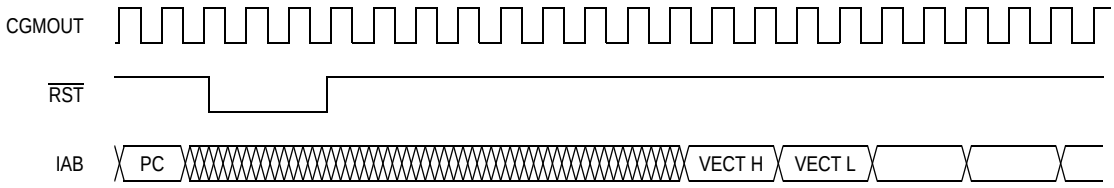


Figure 4-8. External Reset Timing

4.9 3-Phase H-Bridge

The power output is configured as a 3-phase MOSFET inverter with free-wheeling diodes. The gate drivers of the MOSFETs are integrated circuits for high and low side gate drivers with high voltage capability. The gate drivers have a minimum logic 1 input of 3 volts and a maximum logic 0 input voltage of 0.8 volts. A schematic of one of the three phases and its corresponding gate driver circuitry is shown in **Figure 4-9.**

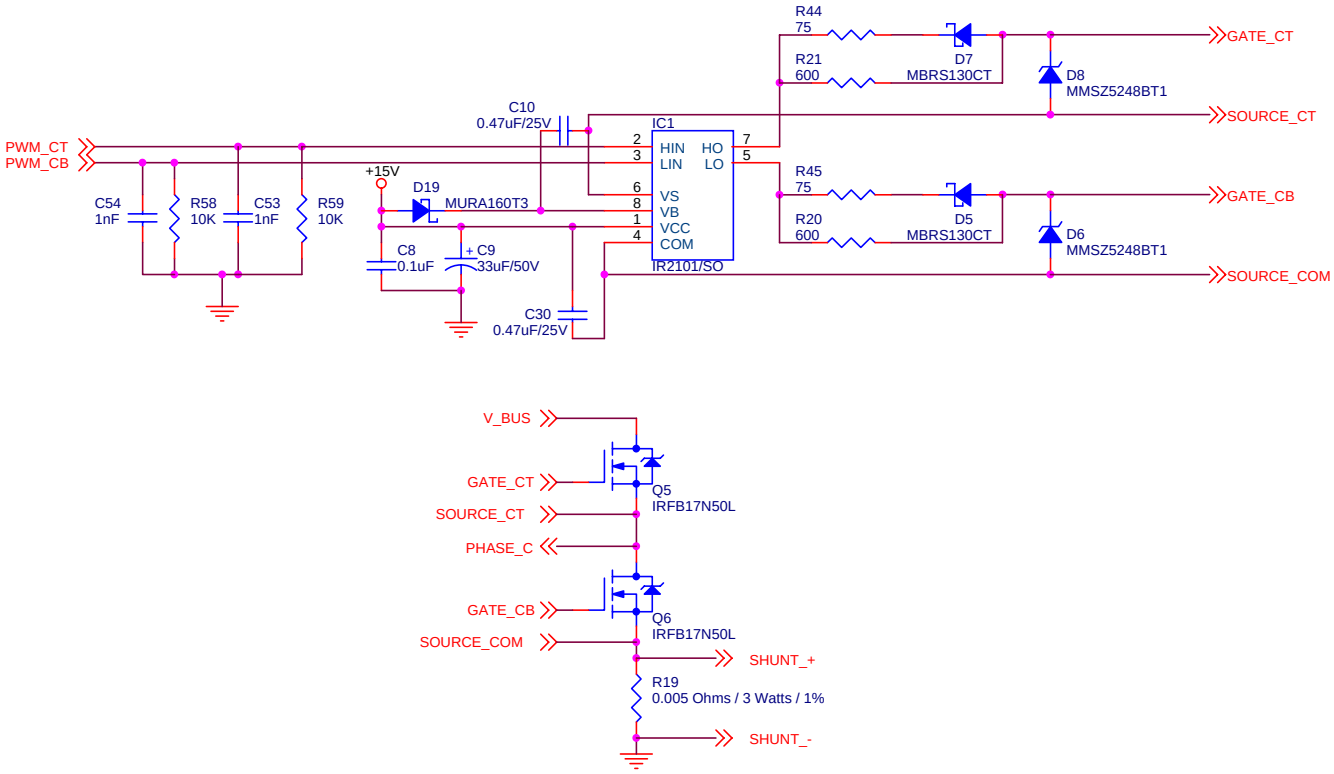


Figure 4-9. Phase C Output and Gate Driver

As a protection for power transitions of the microcontroller's power supply, there are pull-down resistors R58 and R59. So, the MOSFETs are not triggered during transitions.

The gate drive circuit has two different impedance output values, one for turn-on time and other for turn-off time for each of the power transistors, TOP and BOTTOM in each phase. This is possible using D7 and D5 for the turn-off impedances of the transistors per phase. The turn-on impedance is given by R20 and R21 respectively, and the turn-off impedance is given by the parallel connection of R44||R21 and R45||R20 respectively. With the values displayed in the schematic, the turn-on time is 800 ns, and the turn-off time is 600 ns with the IRFB17N50L MOSFET.

In the software for this reference design, deadtime is fixed to 2 μs. This gives enough time for the transistors to change their state of conductance with no short circuit of the phase output.

The bootstrap capacitor C10 is used to turn-on the TOP transistor without a charge pump circuitry. Turning on the lower transistors first is recommended in order to charge this bootstrap capacitor each time the motor is initially energized.

4.10 Current Feedback and Cycle-by-Cycle Limiting

The 3-phase current is sensed by resistor R19 in **Figure 4-9**, and amplified by a differential amplifier shown in **Figure 4-10**. The circuit provides an amplified voltage of the chopped current of the inverter.

The output of the amplifier represents 0.5 volts per ampere in the shunt resistor (R19). The MC33502 OPAMP was used for this amplifier circuit.

At this point, for current sensing within the microcontroller the ADC conversion must be synchronized with the PWM module. That is why a peak detector circuit was implemented to have a suitable current waveform for sensing.

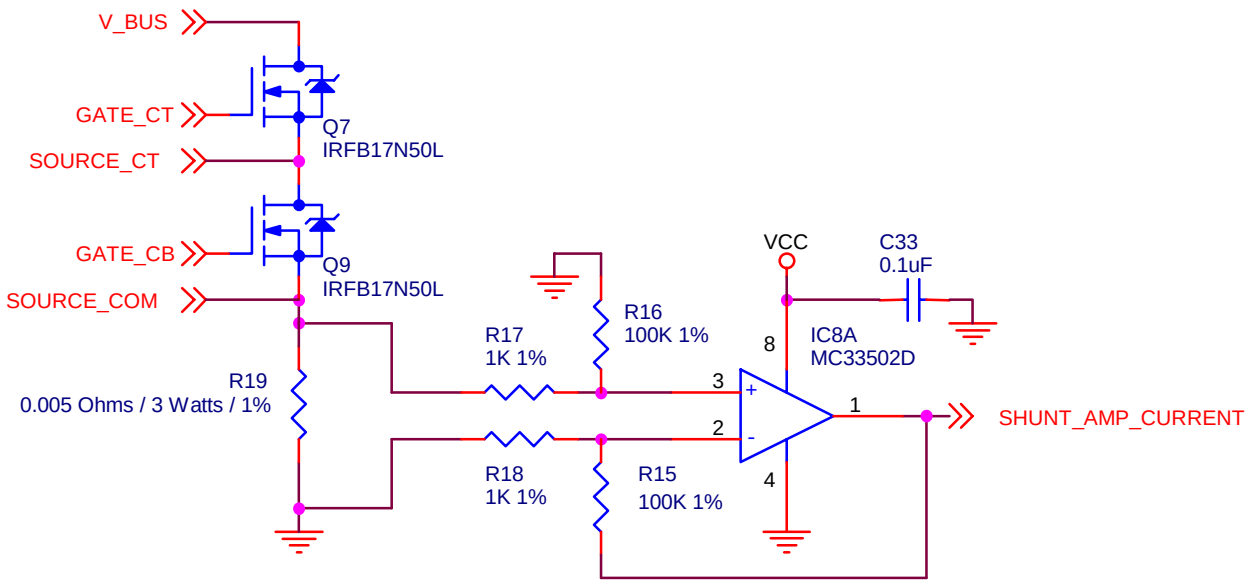


Figure 4-10. Current Differential Amplifier

This peak detector is shown in **Figure 4-11**. Consisting of a voltage follower configuration with diode output for detecting peaks in the input signals.

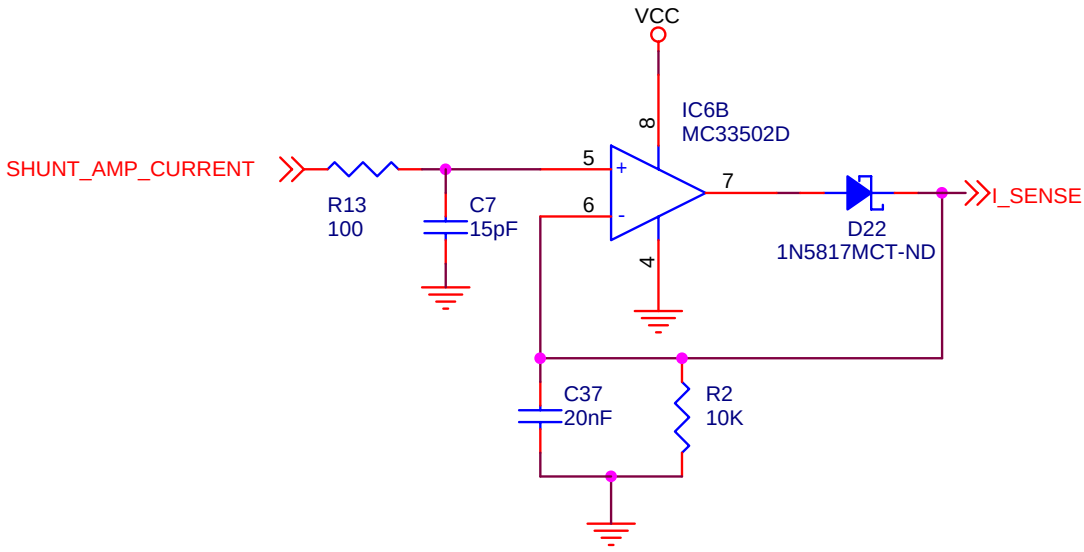


Figure 4-11. Current Peak Detector for Current Sensing

This peak is stored in capacitor C37 when current flows through R19. When the MOSFETs are switched off, the voltage stored in C37 starts to discharge through R2.

The output of the peak detector is connected to a comparator for the cycle-by-cycle current limiting. The FAULT1 input signal of the microcontroller is used for limiting the current. The FAULT configuration in the MCU CONFIG register is set to automatic operation; so, cycle-by-cycle current limiting is accomplished.

The current limiter is shown in **Figure 4-12**. A LM393 was used for this purpose. The output of this current limiter is an open collector, so multiple inputs of limiting can be possible using only one FAULT input signal of the microcontroller.

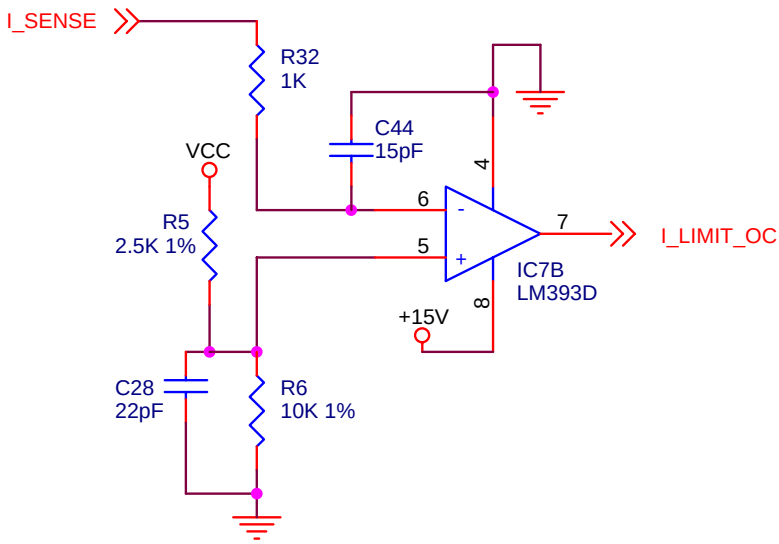


Figure 4-12. Cycle-by-Cycle Current Limiter

4.11 Voltage Feedback

Bus voltage is scaled down by a voltage divider consisting of R25, R10, and R9. The values are chosen such that a 500-volt maximum bus voltage corresponds to 5 volts at output V_SENSE. So, V_SENSE = V_BUS / 100.

For V_BUS FAULT there are two different values, depending on the reference board. For the 115 Vac reference design board, the value is chosen for 250 Vdc maximum, and 400 Vdc maximum for the 230 Vac reference design board. The LM393 is used for the voltage FAULT signal, which is shared with the current FAULT signal of the circuit shown in **Figure 4-12**. The voltage feedback circuitry and voltage FAULT detector (V_LIMIT_OC) is shown in **Figure 4-13**.

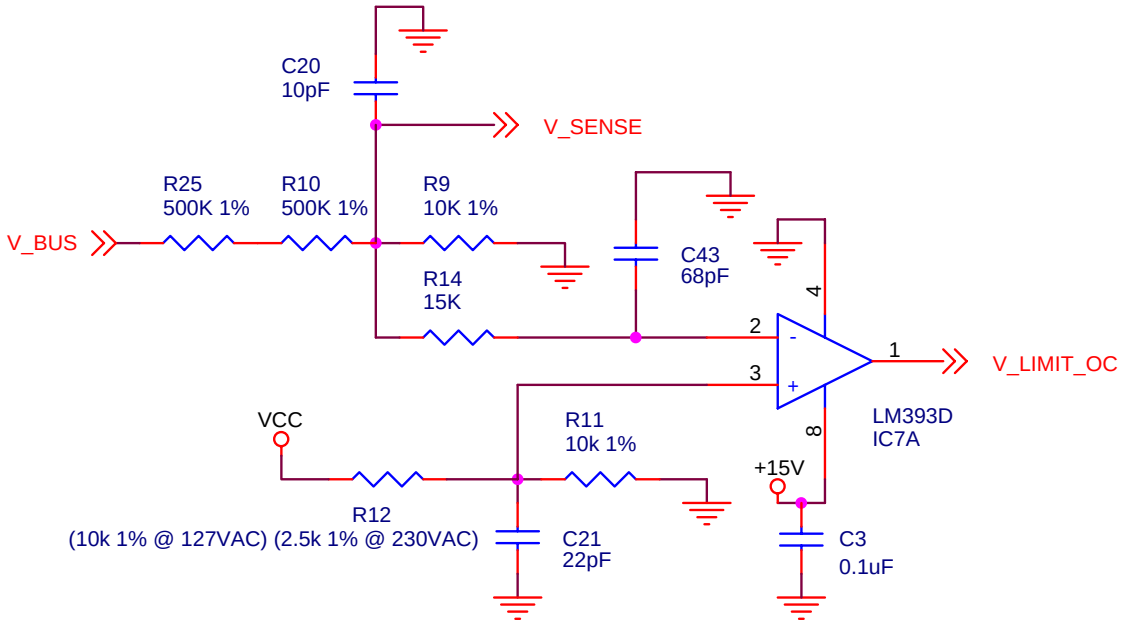


Figure 4-13. Voltage Feedback and Fault Detector

4.12 Current and Voltage Limiter

The circuit is shown in **Figure 4-14**. FAULT is signal connected to the FAULT1 pin of the microcontroller. This input of the microcontroller is used for limiting current and voltage. When either input of the FAULT is in logic 0 state, the transistor Q8 is switched off and the FAULT signal will be set to logic 1.

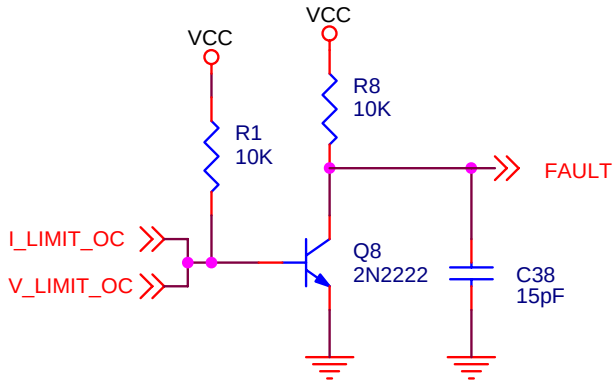


Figure 4-14. Current and Voltage Limiter

4.13 Heat Sink Selection

A recommended application note written by the manufacturer of the heat sink used in this board for selecting a heat sink can be found on the World Wide Web at:

http://www.aavidthermalloy.com/technical/papers/pdfs/select.pdf

The thermal model of a semiconductor with heat sink is:

$$R_{\vartheta DA} = \frac{\left(T_{J_{MAX}} - T_A\right)}{P_D} - R_{\vartheta JC} - R_{\vartheta CD}$$

Where:

R_{SDA}	Thermal impedance of selected heat sink
T_{JMAX}	MOSFET junction maximum temperature
T_A	Ambient temperature
P_D	MOSFET power
R_{SJC}	MOSFET thermal impedance junction to case
R_{SCD}	Thermal impedance of the thermal conductive tape

The values for the components selected on this board are:

Heat sink (part number: 780103B04500):

R_{SDA} = 1.45 °C-in^2 /W

MOSFET (part number: IRF17N50L):

R_{SJC} = 0.75 °C-in^2 /W

Thermally conductive tape (part number: 8805):

R_{SCD} = 0.50 °C-in^2 /W

If we suppose that every MOSFET can be as hot as 110°C and ambient temperature is 25°C, we will get:

P_D = 31.48 W

This is the maximum total power allowed for the six MOSFETs with this heat sink.

The formula to obtain P_D for a single MOSFET is:

$$P_D = (I_{eff})^2 \cdot (Rds_{on})$$

Where:

P_D	Power dissipated by a single MOSFET when conducting
I_{eff}	Effective MOSFET current
Rds_{on}	MOSFET drain-source impedance when it is conducting (0.28 Ω for this MOSFET)

Section 5. Software Design Considerations

5.1 Contents

5.2 Introduction

This section describes data flow of the software implemented for this reference design. The microcontroller is mastering all inputs from the user interface and the Hall effect sensors. From the user interface, functionality (washing machine process) and desired speed for the motor can be set. This data is input for the speed controller that is also detailed in this section. Another input for the speed controller is the actual speed of the motor that is calculated based on the Hall effect sensors values. The controller processes this information and calculates the most suitable value for the MOSFET's PWM signals. Using PWM modules, the microcontroller triggers the MOSFET through a power stage.

NOTE: *The commutation algorithm and speed control for the motor are driven by input capture and timer interrupts.*

5.3 Controller Design

The motor system to be controlled was considered as a first order system, with a time constant of 10 milliseconds. For a robust operation of the washing machine, a PI controller was implemented with a controller period of 1 millisecond. The actual motor speed is calculated from input capture channels, and the desired speed is generated in the microcontroller depending on the washing machine process being executed.

The system has the following transfer function in the continuous time domain.

$$G(s) = \frac{1/\tau}{s + 1/\tau}$$

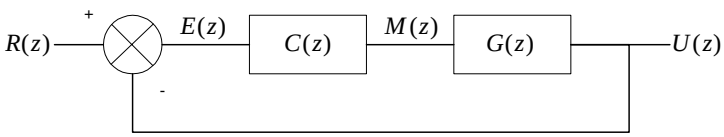
Taking the Z transformation and considering the zero-order-hold of the PWM module, the system's transfer function becomes:

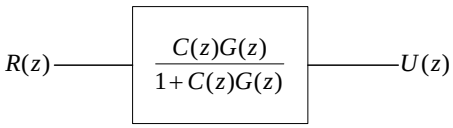
$$G(z) = \frac{\left(1 - e^{-T/\tau}\right) z^{-1}}{1 - e^{-T/\tau} \cdot z^{-1}}$$

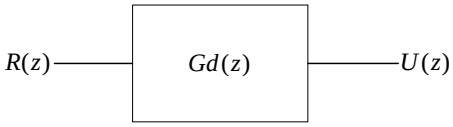
The PI controller transfer function in the Z domain is:

$$C(z) = \frac{(Kp + Ki) - Kp \cdot z^{-1}}{1 - z^{-1}}$$

Closing the loop:







Where:

$$Gd(z) = \frac{\left(1 - e^{-T/\tau d}\right) z^{-1}}{1 - e^{-T/\tau d} \cdot z^{-1}}$$

Then, the controller:

$$C(z) = \frac{Gd(z)}{G(z)\left[1 - Gd(z)\right]}$$

$$C(z) = \frac{1 - e^{-T/\tau d}}{1 - e^{-T/\tau}} \cdot \frac{1 - e^{-T/\tau} \cdot z^{-1}}{1 - z^{-1}} = \frac{(Kp + Ki) - Kp \cdot z^{-1}}{1 - z^{-1}}$$

Solving for *Ki*

$$Ki = 1 - e^{-T/\tau d}$$

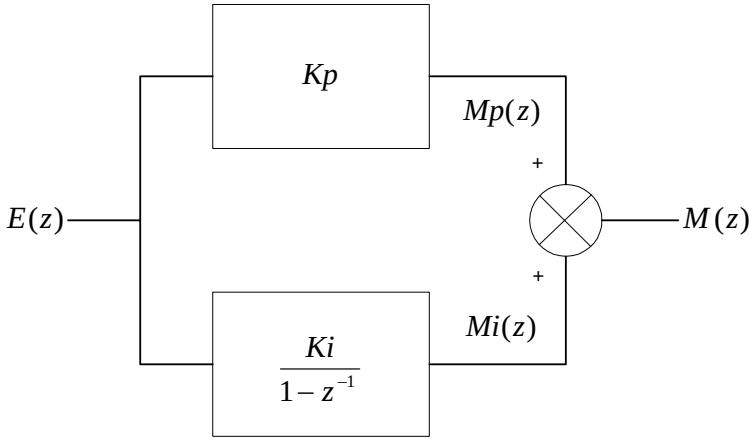
And for *Kp*

$$Kp = \frac{Ki}{1 - e^{-T/\tau}} - Ki$$

Where:

- *T* — Controller period
- τ — Time constant of motor speed in open loop
- τd — Desired time constant of motor speed in closed loop
- *Kp* — Proportional gain of the controller
- *Ki* — Integral gain of the controller

The implementation of the PI controller using parallel programming is given in this diagram:



Converting into equations in discrete time domain:

Mp(K) = Kp • E(K)
Mi(K) = Kp • E(K)
M(K) = Mp(K) + Mi(K)

The targeted motor for the application has a time constant of 10 milliseconds. Based on that, a controller period is defined as 1 millisecond (10 times bigger frequency). Thus, the system has this transfer function:

$$G(z) = \frac{0.095163 \cdot z^{-1}}{1 - 0.904837 \cdot z^{-1}}$$

The desired time constant is 100 milliseconds for the closed loop system. That gives the following values for the controller parameters:

$Ki = 0.00995$
$Kp = 0.094609$

In the microcontroller implementation of this controller, a scale factor is defined. It is better if the scale value is a power of two. So, 256 is our scale value.

$I\_Gain = 0.00995 \bullet 256 = 2.54 \approx 3$
$P\_Gain = 0.094609 \bullet 256 = 24.22 \approx 24$

Once the controller parameters are calculated, it is possible to implement them into the microcontroller.

The PI controller implementation is shown in **Figure 5-1**.

5.4 Speed Control Algorithm

The speed control algorithm consists of three main parts: the actual speed calculation, the speed regulator by a PI controller, and a motor stalled protection. This algorithm is executed by a timer overflow interrupt handler each millisecond. The flowchart of this interrupt handler is shown in **Figure 5-2**.

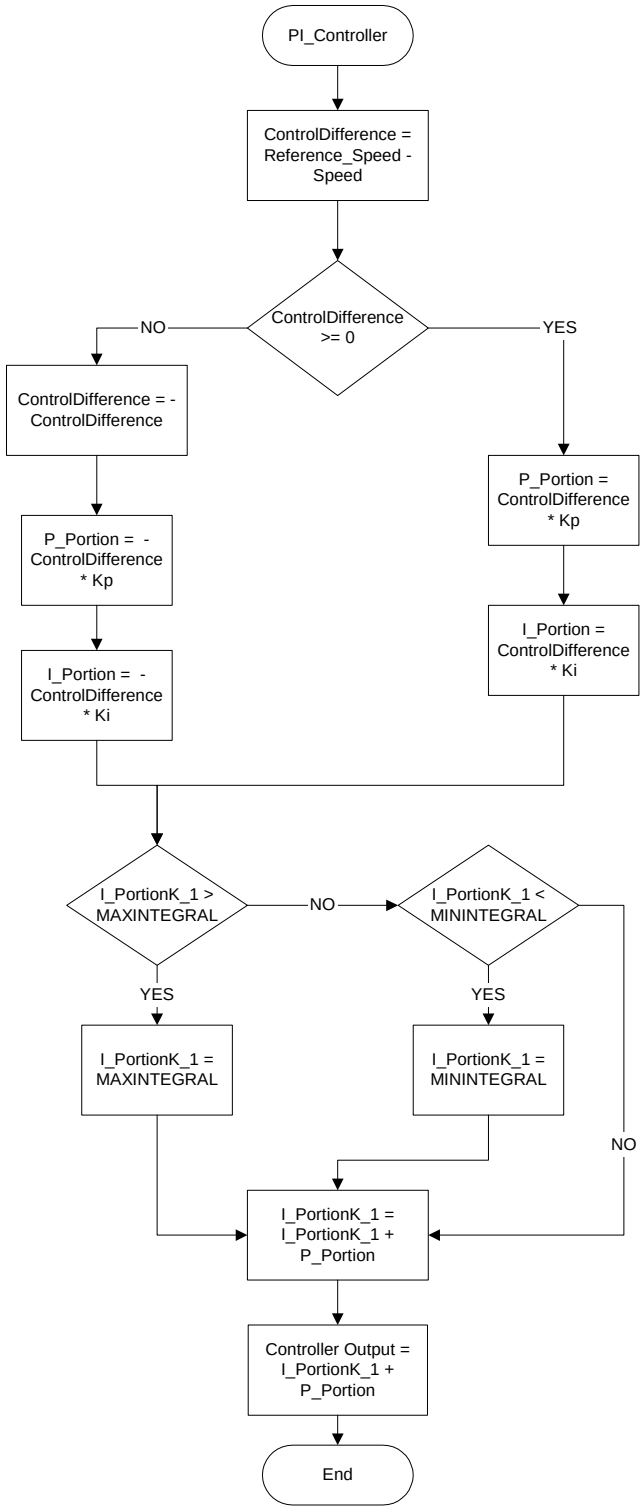


Figure 5-1. PI Controller Flowchart

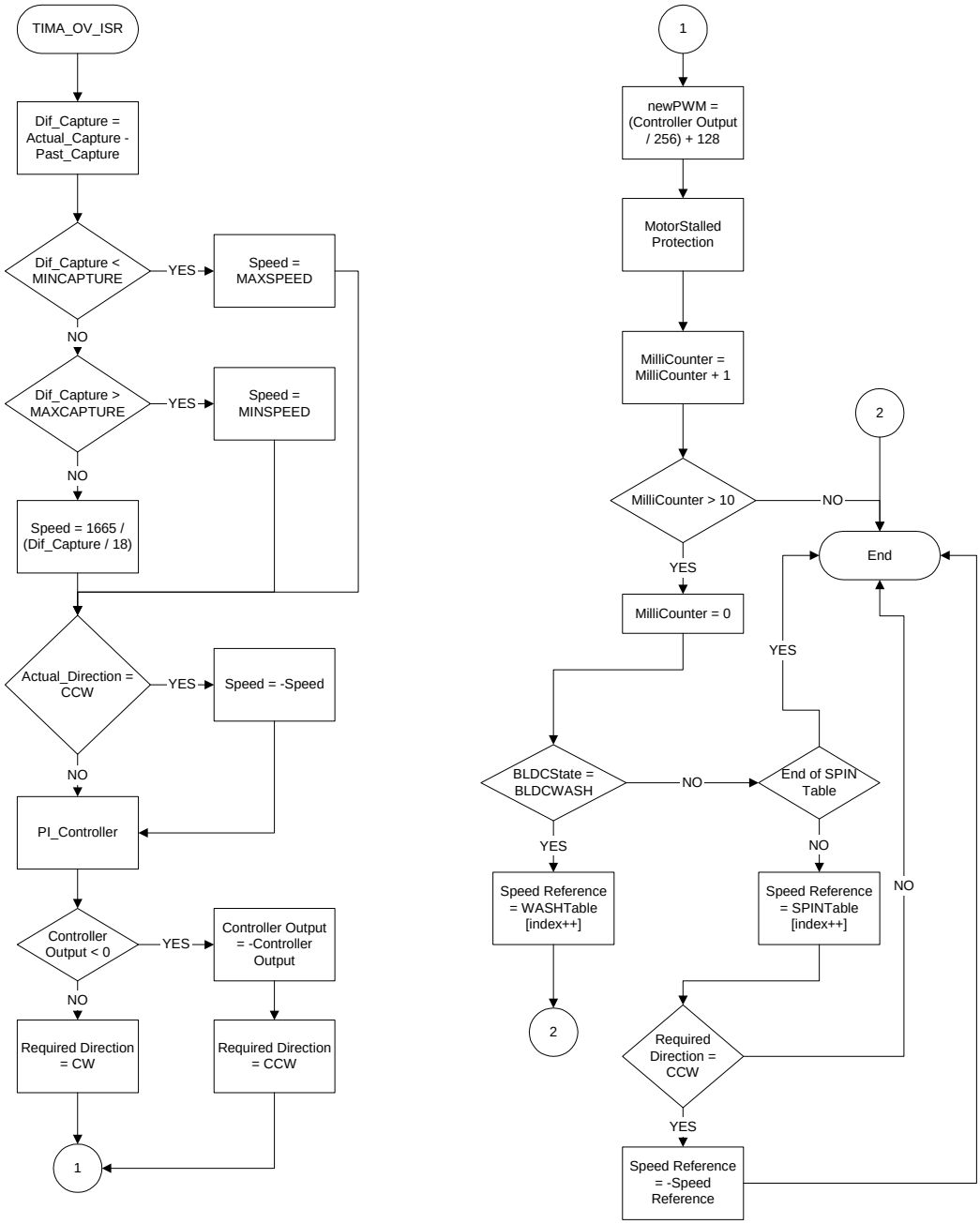


Figure 5-2. Speed Control Algorithm Flowchart

5.4.1 Motor Stalled Protection

The motor stalled protection subroutine is used for commutating the motor windings if the motor hasn't moved to a new angular position. If the motor doesn't change its angular position in a period of 250 milliseconds, the motor is completely stopped.

The motor stalled subroutine's flowchart is the following:

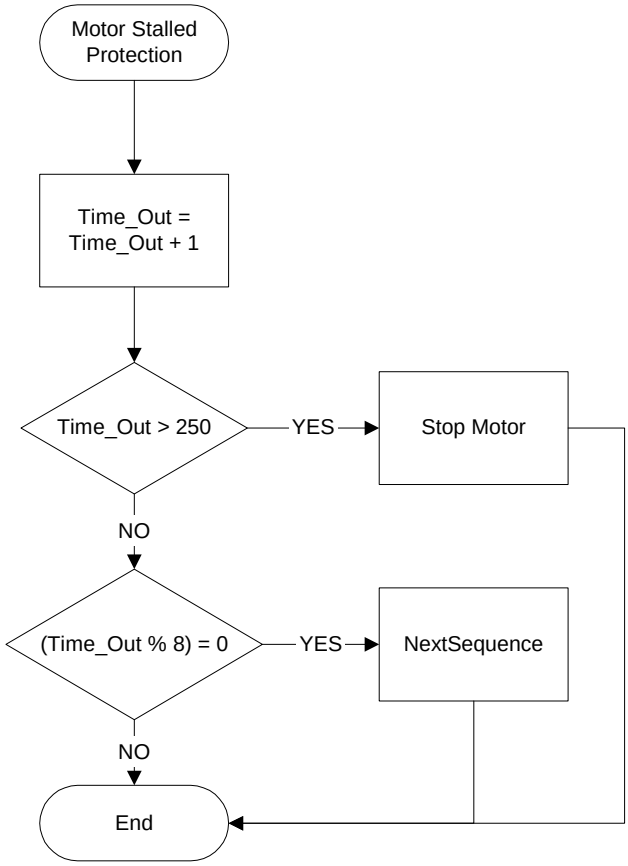


Figure 5-3. Motor Stalled Protection Flowchart

5.5 Commutation Algorithm

The commutation algorithm provides the generation of a rotational field according to rotor position. This algorithm uses the Hall sensors to obtain the rotor position. Outputs from the Hall sensors are connected to three independent input-capture channels through an analog filter. The timers are set to catch each input signal edge and call an interrupt routine, which provides the commutation algorithm.

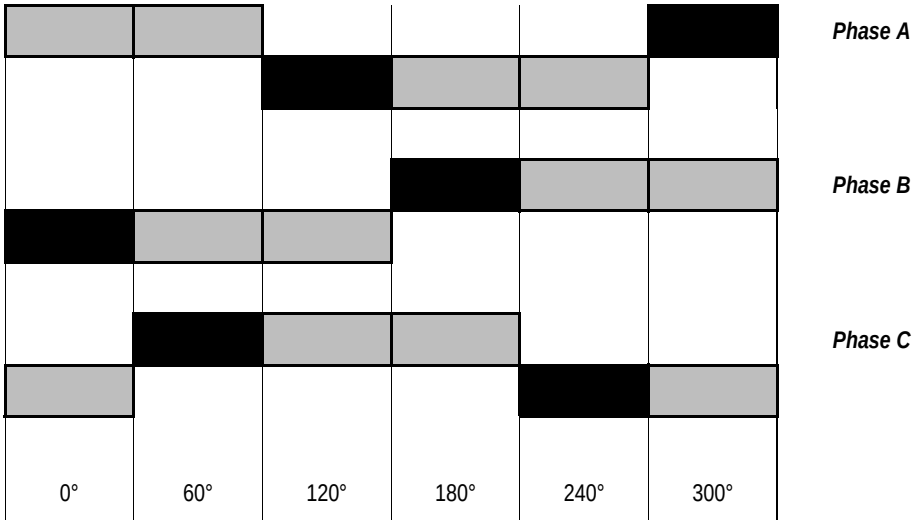
The Hall sensor consists of three sensors (sensor A, sensor B, and sensor C). These sensors comprise six states (001, 010, 011, 100, 101, and 110). Each state determines which motor phase the 3-phase inverter should power. The interrupt routine reads the state of the Hall sensors from the MCU port. This value is used as pointer to the commutation table (see **Table 5-1** and **Table 5-2**), which includes information about the power MOSFETs gating. **Figure 5-4** shows the resultant voltage which is applied to a BLDC motor per one electrical revolution.

Table 5-1. Commutation Sequence for Clockwise Rotation

Hall Sensor Inputs			Two MOSFET Scheme			Three MOSFET Scheme		
Hall Sensor A	Hall Sensor B	Hall Sensor C	Phase A	Phase B	Phase C	Phase A	Phase B	Phase C
1	1	0	+Vdc	NC	–Vdc	+Vdc	–Vdc	–Vdc
1	0	0	+Vdc	–Vdc	NC	+Vdc	–Vdc	+Vdc
1	0	1	NC	–Vdc	+Vdc	–Vdc	–Vdc	+Vdc
0	0	1	–Vdc	NC	+Vdc	–Vdc	+Vdc	+Vdc
0	1	1	–Vdc	+Vdc	NC	–Vdc	+Vdc	–Vdc
0	1	0	NC	+Vdc	–Vdc	+Vdc	+Vdc	–Vdc

Table 5-2. Commutation Sequence for Counterclockwise Rotation

Hall Sensor Inputs			Two MOSFET Scheme			Three MOSFET Scheme		
Hall Sensor A	Hall Sensor B	Hall Sensor C	Phase A	Phase B	Phase C	Phase A	Phase B	Phase C
1	0	1	NC	+Vdc	–Vdc	–Vdc	+Vdc	–Vdc
1	0	0	–Vdc	+Vdc	NC	–Vdc	+Vdc	+Vdc
1	1	0	–Vdc	NC	+Vdc	–Vdc	–Vdc	+Vdc
0	1	0	NC	–Vdc	+Vdc	+Vdc	–Vdc	+Vdc
0	1	1	+Vdc	–Vdc	NC	+Vdc	–Vdc	–Vdc
0	0	1	+Vdc	NC	–Vdc	+Vdc	+Vdc	–Vdc



Note: Use black area for three MOSFET commutation scheme.

Figure 5-4. 3-Phase Voltage System Applies to BLDC Motor

The generation of the PWM voltage waveforms is done by the complementary mode when using a three MOSFET commutation scheme, and by loading 0 to the corresponding phases and configuring the microcontroller to have a TOPNEG PWM when using a two MOSFET commutation scheme. This is done because the M68HC908MRx microcontrollers don't have the PWM MASK option, so

the complementary mode with a two MOSFET commutation scheme is done by software. The deadtime is fixed to 2 microseconds for both commutation schemes. This method allows independence of commutation and speed control. See **Figure 5-5**.

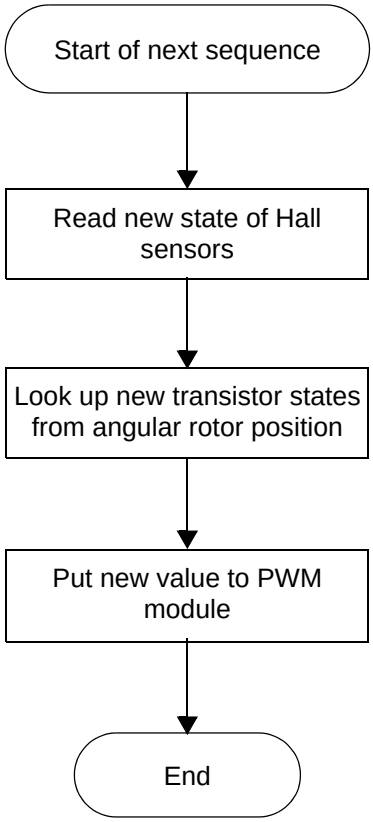


Figure 5-5. Commutation Algorithm for Hall Sensors

5.6 Data Flow

The control algorithm of a closed loop BLDC drive for washing machines is described in **Figure 5-6**.

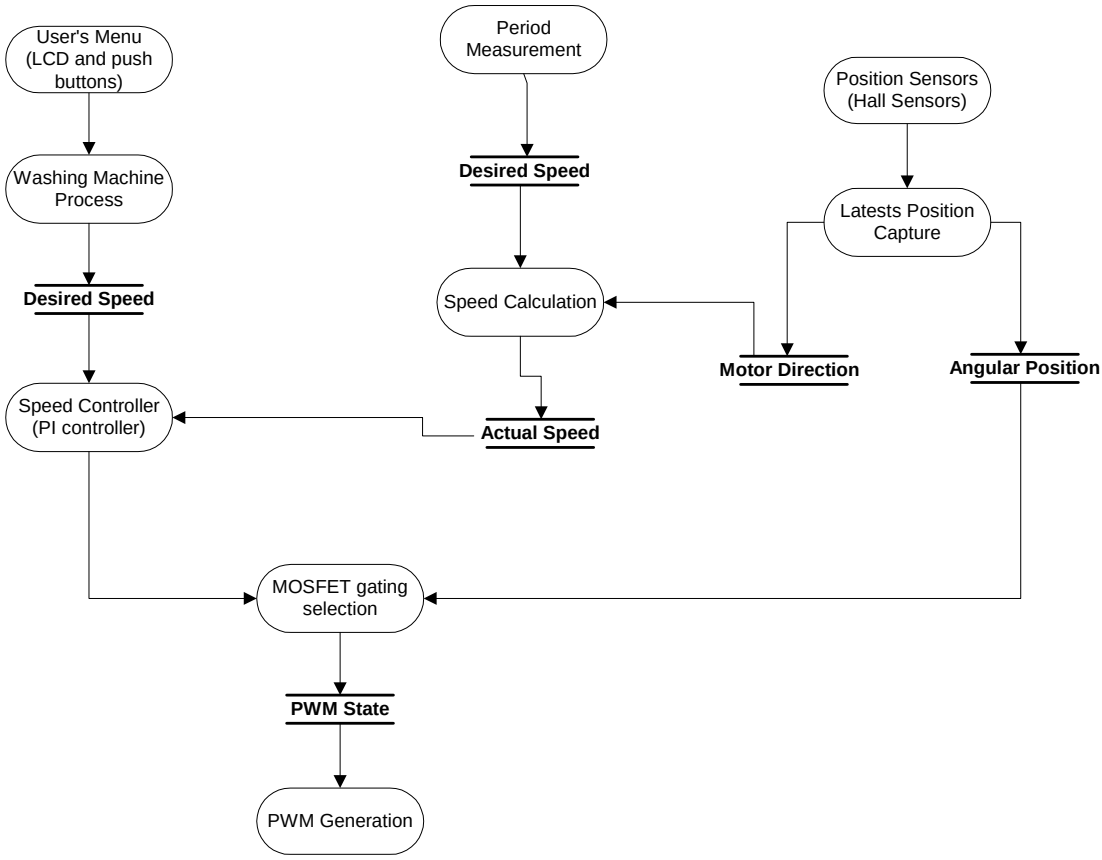


Figure 5-6. Main Data Flow

5.6.1 Processes: Latest Position Capture, Period Measuring, and Speed Calculation

The latest position capture, period measuring, and speed calculation processes relate to the inputs of the Hall sensors. The sensors generate streams of pulses that are captured (separately for each sensor) by the input capture (IC) function. The process latest position capture captures the latest state of the Hall sensors.

The processes period measuring and speed calculation read the time between the adjacent rising edges of Hall sensor output and calculate the actual motor speed variable speed.

5.6.2 Process Speed Controller

This process calculates the duty cycle of the PWM based on the output of the speed controller (the PI controller).

5.6.3 Process MOSFET Gating Selection

This process calculates which PWM channel is enabled for PWM generation. Two commutation schemes are present here. In the file main.h, a compiler directive allows the programmer to select between the two MOSFET scheme and the three MOSFET scheme. For the deadtime insertion there are things which should be noted.

If the three MOSFET commutation scheme is selected by the directive:

#define MOS_3_COM

#undef MOS_2_COM

The PWM module automatically makes deadtime insertion by hardware.

If the two MOSFET commutation scheme is selected by the directive:

#undef MOS_3_COM

#define MOS_2_COM

Deadtime insertion is done by software and PWM module configuration.

As an example, the algorithm for 50% of duty cycle on phase A and the two MOSFET commutation scheme is:

- In the microcontroller CONFIG register the PWM write once register is configured as:
  - Center aligned PWM
  - Independent mode
  - TOPNEG enabled
- The required duty cycle is directly loaded into the PVAL register for the TOP transistor.
- The value loaded into the PVAL register for the BOT transistor is calculated as:

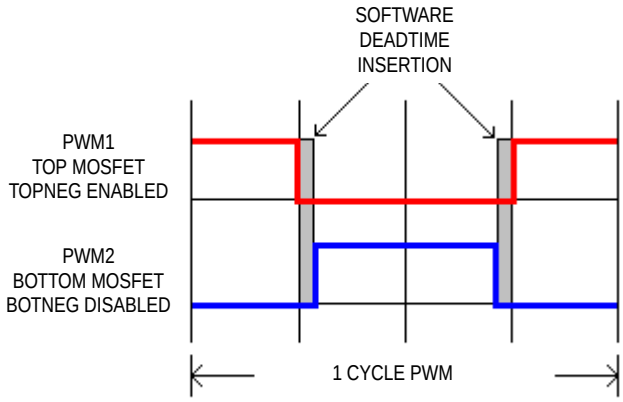
  #define DEADTIME 0x10

  PMOD = 0x100;

  PVAL1 = 0x80;

  PVAL2 = PVAL1 – DEADTIME;

The output signal for one PWM cycle is shown in **Figure 5-7**.



Note: The PWMMC is configured with independent mode and center aligned operation

Figure 5-7. Software Deadtime Insertion

5.6.4 Process Washing Machine

The process generates reference speeds, depending on the process phase being executed of the washer. The user selects the washer process by a user's menu.

5.7 Application State Diagram

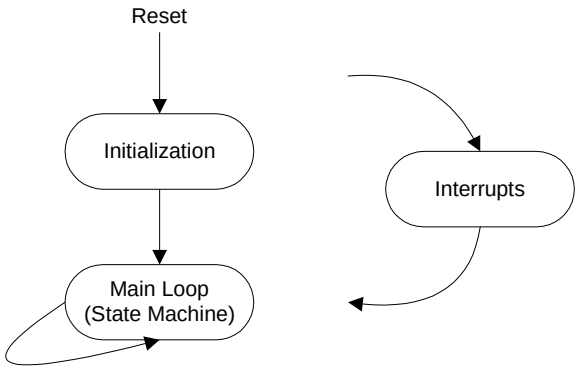


Figure 5-8. Application State Diagram

As Figure 5-8 shows, the application state consists of the initialization routine, followed by a main loop with background tasks. The time critical functions are calculated by the interrupt routines.

A brief description of the 3-phase BLDC motor control follows:

- Initialization routine:
  - PWM initialization
  - System timer initialization
  - Input capture initialization for position feedback
  - Variable initialization for speed measurement
  - Character display initialization
  - I/O ports initialization
  - PLL initialization
  - MCU initialization

- Main loop:
  - Application state machine
  - Check push buttons
  - Display messages for user menu
  - Display actual and desired motor speed
- Initialize motor for running state:
  - Load desired speed from look up table
  - Charge bootstrap capacitors
  - Resume timers for speed control
- Timer A overflow interrupt handler:
  - Speed calculation
  - Speed PI controller calculation
  - Setting of new duty cycle to PWM
  - Motor stalled protection
  - Load new desired speed from look up table depending on the washer process being executed
- Timer A Ch1, Timer B Ch0 and Ch1 interrupt handlers:
  - Reading the angular motor position
  - Spin direction calculation
  - Selecting gating signals for MOSFETs
- Timer B Ch1 interrupt handler
  - Calculation of period between edges for one Hall effect sensor

5.8 Drive State Machine

The drive can be one of the states shown in **Figure 5-9** (which also shows transition conditions among the drive states).

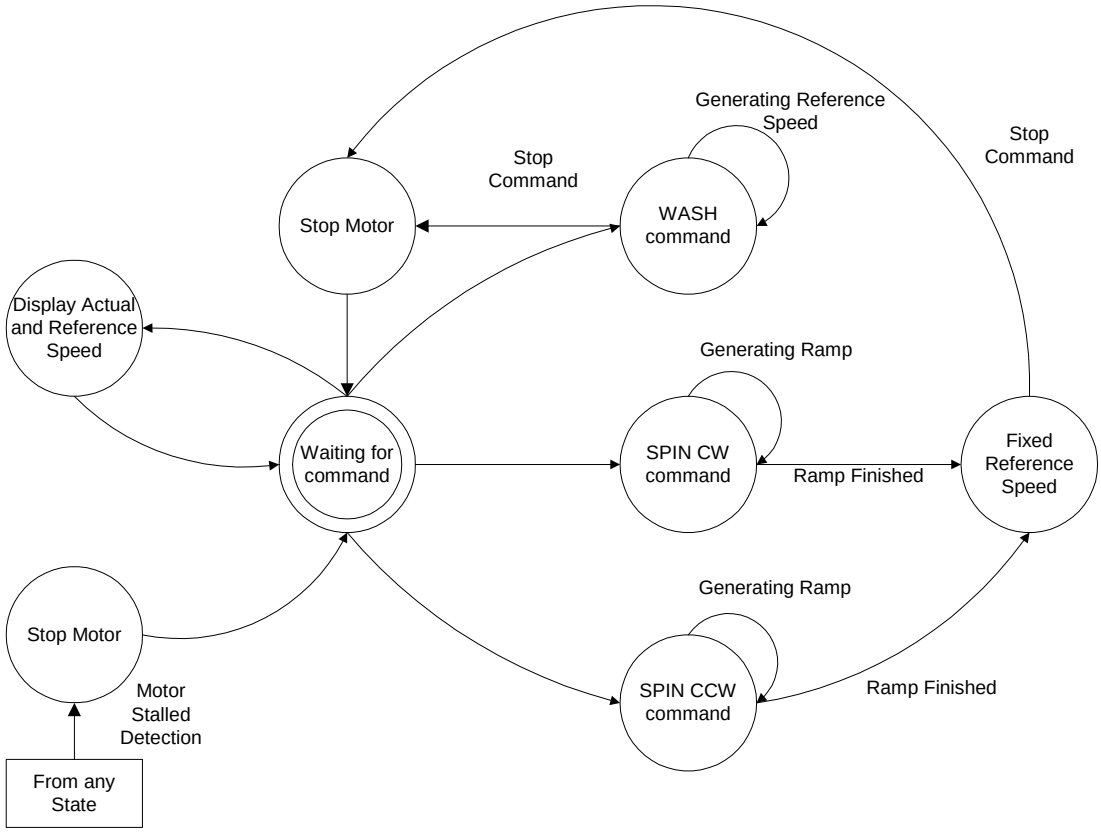


Figure 5-9. Drive State Machine and Transitions

5.9 Description of Routines

The following subsections provide a description of each routine.

5.9.1 Main(void)

This routine contains the principal state machine of the application. It includes initialization and user's menu for selecting two different processes of the washing machine: SPIN and WASH. It also includes two additional options: STOP and Speed display, where the actual and measured speeds are displayed in the LCD.

5.9.1.1 Stop Motor

The application arrives in this state by two different ways: first if there is no Hall sensor changes for more than 250 milliseconds, and second if the user selects the option STOP from the menu.

5.9.1.2 Waiting for Command

This is the idle state of the application. Only the LCD and the push buttons are processed in this state. The UPPER button is used for changing the message displayed; thus, the command to be executed, and the LOWER button is used for executing the currently displayed message command, except for the SPEED message, which displays the actual and desired speed of the motor.

5.9.1.3 Displaying Actual and Reference Speed

In this state, the actual and reference speed are continuously displayed.

5.9.1.4 Wash

When the user selects the WASH process from the user's menu, the application starts running the motor. First an initialization routine is called for charging bootstrap capacitors, resume timers for speed control, and the first reference speed for the Wash process look up table is loaded into variable RefSpeed. Once the motor is running, a timer overflow

interrupt handler is used for the speed control and continuous generation of reference speeds, including positive and negative values, so the agitator moves in both directions of rotation.

5.9.1.5 Spin CW and Spin CCW

When the Spin process is selected in either direction, the motor initialization is called, and then an acceleration ramp is loaded from a Spin look up table, and the sign of the reference speed is set according to the direction of rotation selected.

5.9.1.6 Fixed Reference Speed

When the acceleration ramp table of the Spin process is fully loaded, the reference speed remains constant.

5.9.2 InitPLL(void)

This function is called once in the application. It sets the bus frequency to 8 MHz with an external crystal of 4 MHz.

5.9.3 InitPWMMC(void)

This function initializes the PWM module for motor control with the following settings:

- PWM frequency of 15.625 kHz
- Two microseconds of deadtime
- Reload every PWM cycle

5.9.4 InitTimerA(void)

Timer A and timer A channel 1 are initialized for speed control and commutation control. The overflow interrupt is enabled for speed control each millisecond. Channel 1 is configured as an input capture channel with interrupt enabled on any edge. This channel is connected to Hall sensor A.

5.9.5 InitTimerB(void)

Timer B channel 0 and channel 1 are configured as input capture channels with interrupts enabled on any edge. Channel 0 is connected to Hall sensor B and channel 1 to Hall sensor C. These two channels are used also for commutation control. Channel 1 is used for period calculation between two Hall sensor edges.

5.9.6 Byte ResolveButtons(void)

The state of the input pins, where the push buttons are continuously checked for any change, are tested here. A debounce delay is included in the routine. If there is no change on the push buttons, and the Speed message is being displayed, the respective value of the actual speed and reference speed are displayed in this routine.

5.9.7 InitMotor(Byte Commanded_Operation)

This subroutine is called from main to perform one of the two of the washing machine processes. The process is selected by the parameter value, Commanded_Operation.

Parameters:
BLDCWASH — Wash process of the washing machine.
BLDCSPIN — Spin process

Depending on the process selected from the user's menu, the Speed reference is loaded from the respective look up table. The speed controller integral portion is set to 0, the bootstrap capacitors are charged and the timers are resumed.

5.9.8 TimerAOverflow_ISR(void)

Refer to **5.4 Speed Control Algorithm**.

5.9.9 Signed Word 16 PIController(void)

Refer to **5.3 Controller Design**.

5.9.10 MotorStalledProtection(void)

Refer to **5.4.1 Motor Stalled Protection**.

5.9.11 HALLA_ISR(void) and HALLB_ISR(void)

Interrupt handler routines to drive Hall sensors A and B for BLDC motor commutation. Direction is computed from the last Hall sensor input state.

5.9.12 HALLC_ISR(void)

Interrupt handler routines to drive Hall sensor C for BLDC motor commutation. Direction is computed from the last Hall Sensor input state. In this routine, the period between edges is measured for speed calculation.

5.9.13 Fault1_ISR(void)

Interrupt handler subroutine for Fault1. The motor is stopped when a FAULT occurs. The FAULT is asserted when the current limit or voltage limit has been reached by the power stage.

5.9.14 NextSequence(void)

In this routine, the MOSFET selection is performed based on the commutation scheme and the Required_Direction of the motor. Refer to **5.5 Commutation Algorithm**.

5.9.15 InitLCD(void)

This function initializes the character display with these settings:

- 4-bit operation mode
- 2-line display
- No display shift and move right
- Clear display and return to home position
- Display on, blink off, and cursor off

5.9.16 CtrlLCD(Byte ctrl)

This subroutine is used for sending control bytes to the LCD. Because the function is called in 4-bit operation mode, this routine sends the 8-bit value in two parts.

Parameters:

ctrl — An 8-bit value for different control of the LCD, such as number of lines, blink on or off, etc.

5.9.17 Ctrl8LCD(Byte ctrl)

This subroutine is used for sending control bytes to the LCD in 8-bit mode. The function is used only to enter 4-bit mode, since the other four data pins have no connection.

Parameters:

ctrl — An 8-bit value for different control of the LCD, such as number of lines, blink on or off, etc.

5.9.18 MovCursorLCD(Byte places, Byte dir)

Function used to move the LCD cursor to right or left the number of desired places.

Parameters:

places — Number of places wanted to move the LCD cursor without affecting any LCD actual message.

dir — Direction in which the cursor is to be moved, right or left.

5.9.19 DataLCD(Byte data)

ASCII symbol to be displayed on the LCD, at the current cursor position.

Parameters:

data — 8-bit value representing the ASCII code of the symbol to be displayed in the LCD at current position.

5.9.20 StringLCD(Byte *msgLCD)

This function displays a string in the LCD at current cursor position. If a '&' character is present in the string, a new line feed is commanded to the LCD. The function sends all the bytes in the string until a presence of an End Of String, EOS or 0x00 byte.

Parameters:

*msgLCD — Pointer to the string to be displayed on the LCD.

5.9.21 WaitMs(Byte milis)

Delay routine that waits for a number of milliseconds to send in the parameter milis. The delay is calculated for an 8 MHz f_{BUS} operation.

Parameters:

milis — An 8-bit value representing the number of milliseconds the delay will take.

5.9.22 Wait40μs(void)

Fixed delay of 40 microseconds.

5.10 MCU Usage

Table 5-3 shows how much memory is needed to run the 3-phase BLDC motor drive in a speed closed loop using Hall sensors, washing machine functions, and user's interface. A part of the MCU memory is still available for other tasks.

Table 5-3. RAM and FLASH Memory Usage

Memory (In 8-Bit Words)	Available (MC68HC908MR8)	Used (Application + Stack)
Program FLASH	7680	2820
Data RAM	256	36 + 96

Software Design Considerations

Section 6. Practical Results

Figure 6-1 shows the motor power output versus the motor torque with drives for the two commutation schemes developed in the reference design — consisting of switching two MOSFETs at each angular position or three MOSFETs at each angular position.

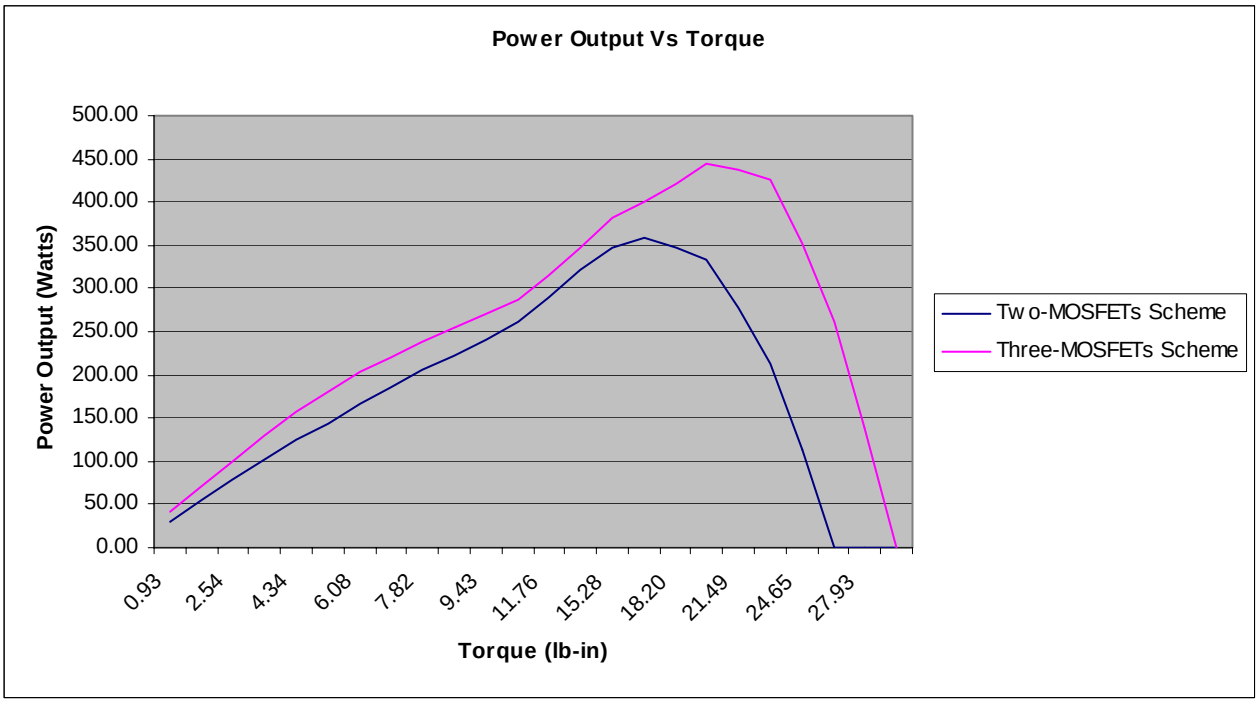


Figure 6-1. Power Output versus Torque Motor Characteristic

Figure 6-2 shows the motor torque output versus motor maximum speed for the two commutation algorithms developed in the reference design.

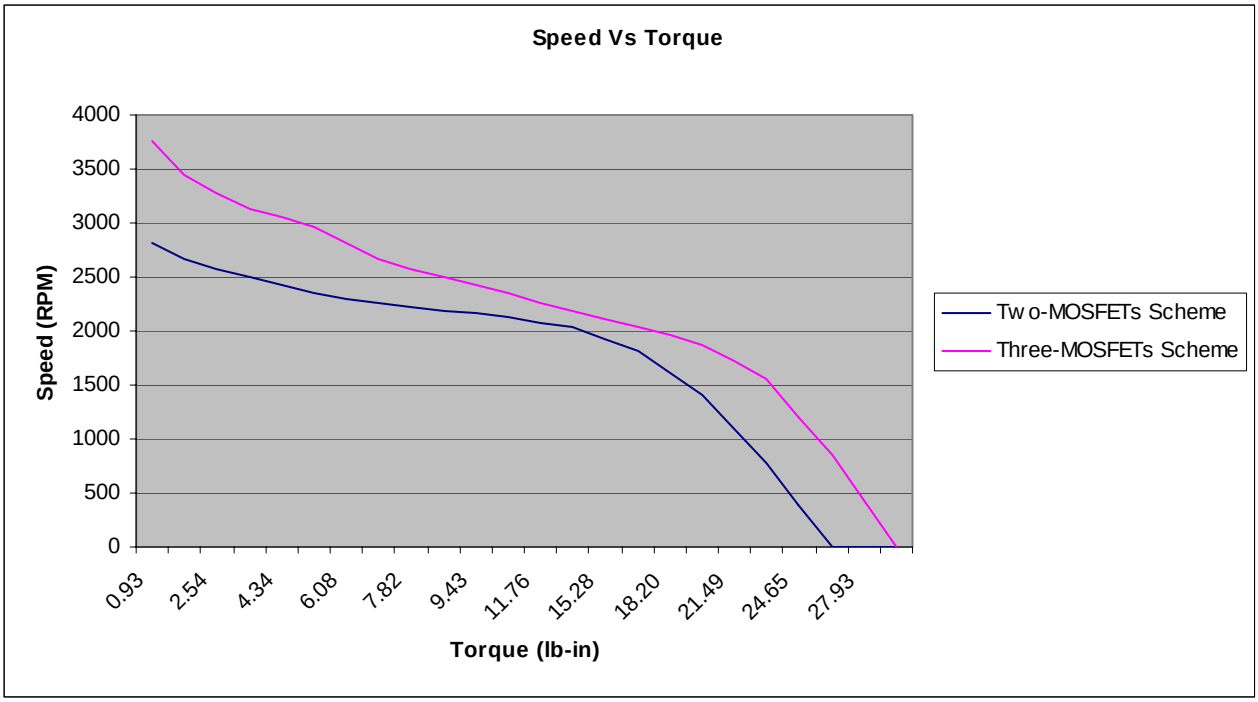


Figure 6-2. Speed versus Torque Motor Characteristic

Current waveforms are shown in the two oscilograms:

- Figure 6-3 for the commutation scheme switching two MOSFETs at a time
- Figure 6-4 for the commutation scheme switching three MOSFETs at a time

NOTE: *There is less torque ripple, which is dependent on the current, for the commutation algorithm switching three MOSFETs.*

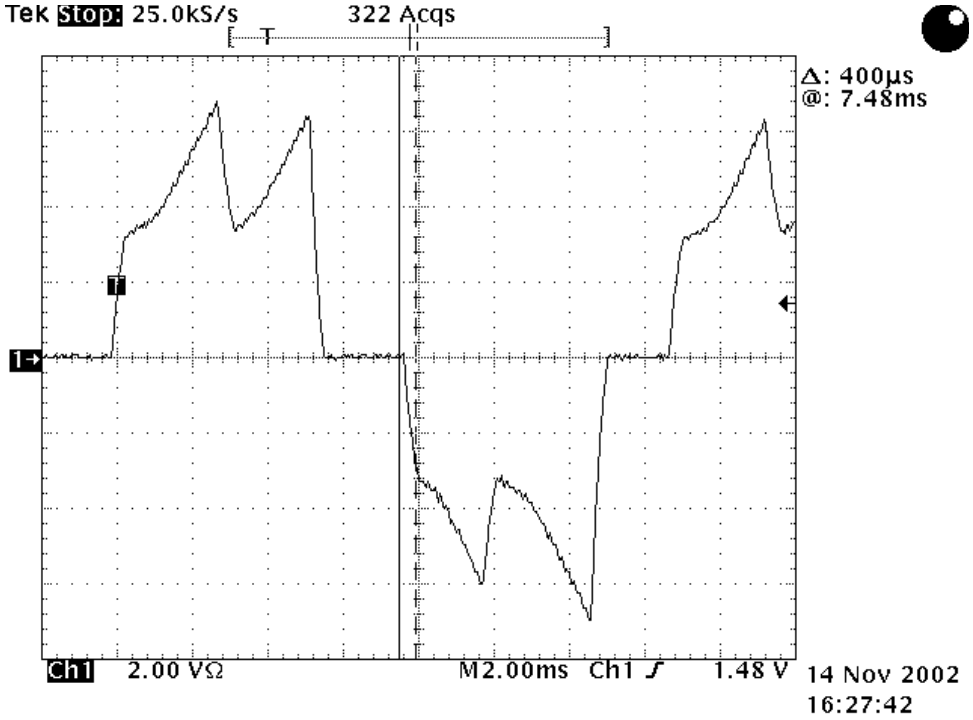


Figure 6-3. Current Waveform for Two MOSFET Commutation Scheme

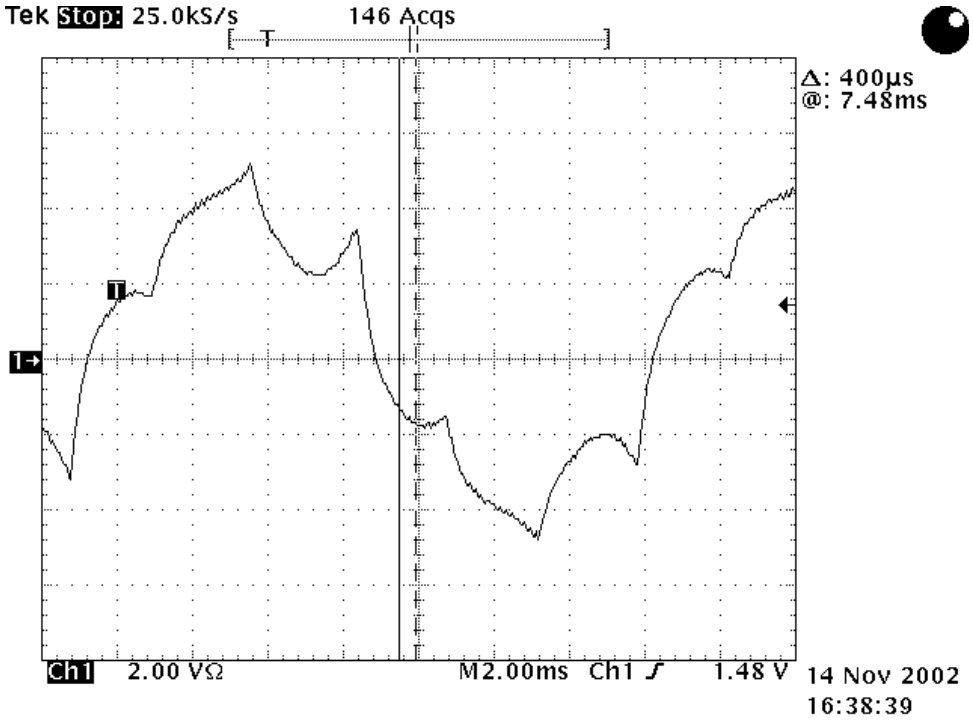


Figure 6-4. Current Waveform for Three MOSFET Commutation Scheme

Taking the rectified current of the three-phase inverter, the torque ripple in the motor can be seen assuming that torque is proportional to current. This is shown in **Figure 6-5** and **Figure 6-6** for the two MOSFET commutation scheme and the three MOSFET commutation scheme, respectively.

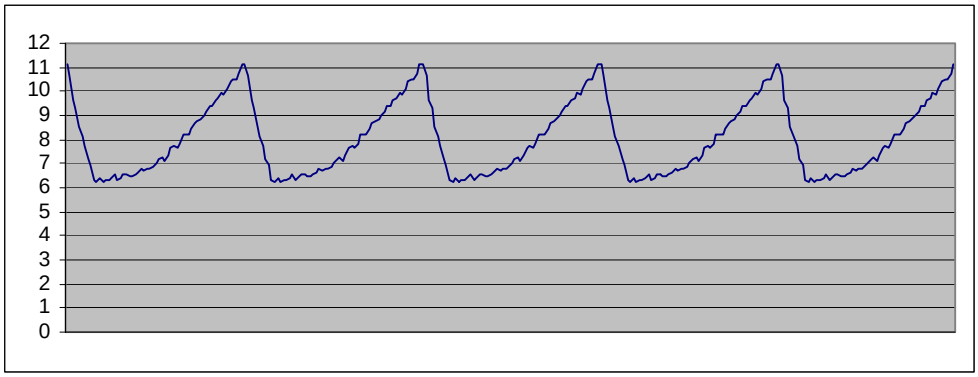


Figure 6-5. Torque Waveform for Two MOSFET Commutation Scheme

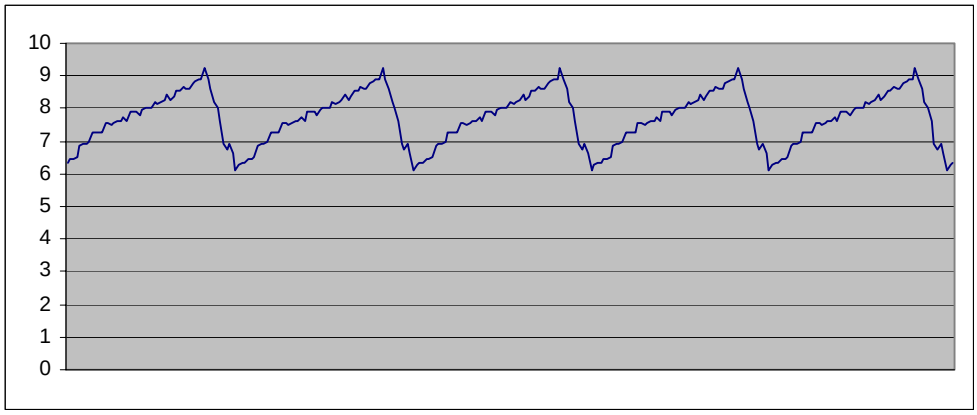


Figure 6-6. Torque Waveform for Three MOSFET Commutation Scheme

The speed control algorithm results are listed in **Table 6-1** for this reference design. This data includes:

- Steady-state error of the controller for different speeds
- Minimum and maximum controllable speed ranges

Table 6-1. Speed Results

	Full Load (13.4 lb-in)	Full Load (13.4 lb-in)	No Load	No Load
	Maximum Speed (RPM)	***Minimum Speed (RPM)***	***Maximum Speed (RPM)***	***Minimum Speed (RPM)***
3 MOSFET	2189.7 (–31.3, +62.6)	218.9 (± 31.3)	3440.9 (–187.7, +31.3)	218.9 (± 31.3)
2 MOSFET	2033.3 (± 31.3)	218.9 (± 31.3)	2658.9 (± 62.6)	218.9 (± 31.3)

Practical Results

Section 7. Source Code

7.1 Contents

7.2 Include Files

7.2.1 MR8IO.H

```
;
/*****************************************************************************\
* Copyright (c) 2002, Motorola Inc.
*
* Motorola Confidential Proprietary
*
* ------------------------------------------------------------------------- *
* File name :     mr8io.h                                                   *
* Project name:   Brushless DC Motor Drive with the MR8 Microcontroller
*
* ------------------------------------------------------------------------- *
* Author :        Jorge Zambada                                             *
* Email :         Jorge.Zambada@motorola.com                                *
* Department :    Mexico Applications Lab - SPS                             *
*                                                                           *
* Description :   All the MCU registers and some bit mask values are declared
*
*                 in this document as defines to interface with most of the   *
*                 microcontroller registers and peripherals
*
\*****************************************************************************/

/*      PORTS section
 */
#define PORTA   (*(volatile char*)(0x00))  /* port A */
#define PORTB   (*(volatile char*)(0x01))  /* port B */
#define PORTC   (*(volatile char*)(0x02))  /* port C */
#define DDRA    (*(volatile char*)(0x04))  /* data direction port A */
#define DDRB    (*(volatile char*)(0x05))  /* data direction port B */
#define DDRC    (*(volatile char*)(0x06))  /* data direction port C */

/*      A TIMER section
 */
#define TASC    (*(volatile char*)(0x0E))  /* timer A status/ctrl register */
#define TACNT   (*(volatile int*)(0x0F))   /* timer A counter register */
#define TACNTH  (*(volatile char*)(0x0F)   /* timer A counter high */
#define TACNTL  (*(volatile char*)(0x10))  /* timer A counter low */
#define TAMOD   (*(volatile int*)(0x11))   /* timer A modulo register */
#define TAMODH  (*(volatile char*)(0x11))  /* timer A modulo high */
#define TAMODL  (*(volatile char*)(0x12))  /* timer A modulo low */
#define TASC0   (*(volatile char*)(0x13))  /* timer A channel 0 status/ctrl */
#define TACH0   (*(volatile int*)(0x14))   /* timer A channel 0 register */
#define TACH0H  (*(volatile char*)(0x14))  /* timer A channel 0 high */
#define TACH0L  (*(volatile char*)(0x15))  /* timer A channel 0 low */
#define TASC1   (*(volatile char*)(0x16))  /* timer A channel 1 status/ctrl */
```

```
#define TACH1   (*(volatile int*)(0x17))   /* timer A channel 1 register */
#define TACH1H  (*(volatile char*)(0x17))  /* timer A channel 1 high */
#define TACH1L  (*(volatile char*)(0x18))  /* timer A channel 1 low */

/*      OPTION section
 */
#define MOR     (*(volatile char*)(0x1F))  /* CONFIG Configuration Write-Once Regis-
ter */

/*      PWM section
*/
#define PCTL1   (*(volatile char*)(0x20))  /* PWM control register 1 */
#define PCTL2   (*(volatile char*)(0x21))  /* PWM control register 2 */

#define FCR     (*(volatile char*)(0x22))  /* Fault control register */
#define FSR     (*(volatile char*)(0x23))  /* Fault Status register */
#define FTAC    (*(volatile char*)(0x24))  /* Fault acknowledge register */
#define PWMOUT  (*(volatile char*)(0x25))  /* PWM output control register */
#define PCNT    (*(volatile int*)(0x26))   /* PWM counter register */
#define PCNTH   (*(volatile char*)(0x26))  /* PWM counter register high */
#define PCNTL   (*(volatile char*)(0x27))  /* PWM counter register low */
#define PMOD    (*(volatile int*)(0x28))   /* PWM counter Modulo register */
#define PMODH   (*(volatile char*)(0x28))  /* PWM counter Modulo reg. high */
#define PMODL   (*(volatile char*)(0x29))  /* PWM counter Modulo reg. low */
#define PVAL1   (*(volatile int*)(0x2a))   /* PWM 1 value register */
#define PVAL1H  (*(volatile char*)(0x2a))  /* PWM 1 value register high */
#define PVAL1L  (*(volatile char*)(0x2b))  /* PWM 1 value register low */
#define PVAL2   (*(volatile int*)(0x2c))   /* PWM 2 value register */
#define PVAL2H  (*(volatile char*)(0x2c))  /* PWM 2 value register high */
#define PVAL2L  (*(volatile char*)(0x2d))  /* PWM 2 value register low */
#define PVAL3   (*(volatile int*)(0x2e))   /* PWM 3 value register */
#define PVAL3H  (*(volatile char*)(0x2e))  /* PWM 3 value register high */
#define PVAL3L  (*(volatile char*)(0x2f))  /* PWM 3 value register low */
#define PVAL4   (*(volatile int*)(0x30))   /* PWM 4 value register */
#define PVAL4H  (*(volatile char*)(0x30))  /* PWM 4 value register high */
#define PVAL4L  (*(volatile char*)(0x31))  /* PWM 4 value register low */
#define PVAL5   (*(volatile int*)(0x32))   /* PWM 5 value register */
#define PVAL5H  (*(volatile char*)(0x32))  /* PWM 5 value register high */
#define PVAL5L  (*(volatile char*)(0x33))  /* PWM 5 value register low */
#define PVAL6   (*(volatile int*)(0x34))   /* PWM 6 value register */
#define PVAL6H  (*(volatile char*)(0x34))  /* PWM 6 value register high */
#define PVAL6L  (*(volatile char*)(0x35))  /* PWM 6 value register low */

#define DEADTM  (*(volatile char*)(0x36))  /* Dead Time Write-once register */
#define DISMAP  (*(volatile char*)(0x37))  /* PWM Disable Mapping Write-once register
*/

/*      SCI section
 */
#define SCC1    (*(volatile char*)(0x38))  /* SCI control register 1 */
#define SCC2    (*(volatile char*)(0x39))  /* SCI control register 2 */
#define SCC3    (*(volatile char*)(0x3A))  /* SCI control register 3 */
```

```
#define SCS1    (*(volatile char*)(0x3B))  /* SCI status register 1 */
#define SCS2    (*(volatile char*)(0x3C))  /* SCI status register 2 */
#define SCDR    (*(volatile char*)(0x3D))  /* SCI data register */
#define SCBR    (*(volatile char*)(0x3E))  /* SCI baud rate */

/*      INTERRUPT section
 */
#define ISCR    (*(volatile char*)(0x3F))  /* IRQ status/control register */
/*      A/D section
 */
#define ADSCR   (*(volatile char*)(0x40))  /* ADC status and control reg. */
#define ADR     (*(volatile int*)(0x41))   /* ADC data register */
#define ADRH    (*(volatile char*)(0x41))  /* ADC data register high */
#define ADRL    (*(volatile char*)(0x42))  /* ADC data register low */
#define ADCLK   (*(volatile char*)(0x43))  /* ADC clock register */

/*      B TIMER section
 */
#define TBSC    (*(volatile char*)(0x51))  /* timer B status/ctrl register */
#define TBCNT   (*(volatile int*)(0x52))   /* timer B counter register */
#define TBCNTH  (*(volatile char*)(0x52))  /* timer B counter high */
#define TBCNTL  (*(volatile char*)(0x53))  /* timer B counter low */
#define TBMOD   (*(volatile int*)(0x54))   /* timer B modulo register */
#define TBMODH  (*(volatile char*)(0x54))  /* timer B modulo high */
#define TBMODL  (*(volatile char*)(0x55))  /* timer B modulo low */
#define TBSC0   (*(volatile char*)(0x56))  /* timer B channel 0 status/ctrl */
#define TBCH0   (*(volatile int*)(0x57))   /* timer B channel 0 register */
#define TBCH0H  (*(volatile char*)(0x57))  /* timer B channel 0 high */
#define TBCH0L  (*(volatile char*)(0x58))  /* timer B channel 0 low */
#define TBSC1   (*(volatile char*)(0x59))  /* timer B channel 1 status/ctrl */
#define TBCH1   (*(volatile int*)(0x5A))   /* timer B channel 1 register */
#define TBCH1H  (*(volatile char*)(0x5A))  /* timer B channel 1 high */
#define TBCH1L  (*(volatile char*)(0x5B))  /* timer B channel 1 low */

/*      PLL section
 */
#define PCTL   (*(volatile char*)(0x5C))   /* PLL control register */
#define PBWC   (*(volatile char*)(0x5D))   /* PLL bandwidth register */
#define PPG    (*(volatile char*)(0x5E))   /* PLL programming register */

/*      SIM section
 */
#define SBSR    (*(volatile char*)(0xFE00))/* SIM break status register */
#define SRSR    (*(volatile char*)(0xFE01))/* SIM reset status register */
#define SBFCR   (*(volatile char*)(0xFE03))/* SIM break control register */
#define FLCR    (*(volatile char*)(0xFE08))/* FLASH control register */

#define LVISCR (*(volatile char*)(0xFE0F)) /* LVI status/control register */
#define FLBPR  (*(volatile char*)(0xFF7E)) /* FLASH block protect register */
#define COPCTL (*(volatile char*)(0xFFFF)) /* COP control register */
```

```
/*          ADC Flags and bit masks
*/
#define ATD8_PTC0                       0x06
#define Continuous_Conversion 0x20
#define ADC_Input_Clock_by_8  0x60
#define Internal_Bus_Clock      0x10
#define COCO                            0x80

/*          PLL Flags and bit masks
*/
#define BCS    0x10
#define PLLON  0x20
#define AUTO   0x80
#define LOCK   0x40
/*          IRQ Flags and bit masks
*/
#define IMASK  0x02

/*          PWM Flags and bit masks
*/
*/
#define PWMEN     0x01
#define LDOK      0x02
#define PWMINT    0x20
#define PWMF      0x10
#define FTACK1    0x01

/*          TIM Flags and bit masks
*/
#define TRST   0x10
#define TSTOP  0x20
#define TOIE   0x40
#define CHIE   0x40
#define TOF    0x80
#define CHF    0x80

/*****************************************************************************\
* End mr8io.h                                                                 *
*****************************************************************************/
;
```

7.2.2 START08.H

```
;
/*****************************************************************************
  FILE    : start08.h
  PURPOSE : datastructures for startup
  LANGUAGE: ANSI-C
*/
/*****************************************************************************/

#ifndef START08_H
#define START08_H

#ifdef __cplusplus
extern "C" {
#endif

#include "hidef.h"

/*
  the following datastructures contain the data needed to
  initialize the processor and memory
*/

typedef struct{
  unsigned char *_FAR beg;
  int size;      /* [beg..beg+size] */
} _Range;


typedef struct _Copy{
  int size;
  unsigned char *_FAR dest;
} _Copy;

typedef void (*_PFunc)(void);

typedef struct _LibInit{
  _PFunc  *startup;      /* address of startup desc */
} _LibInit;

typedef struct _Cpp{
  _PFunc  initFunc;      /* address of init function */
} _Cpp;

#define STARTUP_FLAGS_NONE          0
#define STARTUP_FLAGS_ROM_LIB       (1<<0)  /* if module is a ROM library */
#define STARTUP_FLAGS_NOT_INIT_SP   (1<<1)  /* if stack pointer has not to be initial-
ized */
```

```
#pragma DATA_SEG FAR _STARTUP

#ifdef __ELF_OBJECT_FILE_FORMAT__

/* ELF/DWARF object file format */

/* attention: the linker scans the debug information for this structures */
/* to obtain the available fields and their sizes. */
/* So dont change the names in this file. */

extern struct _tagStartup {
     unsigned char   flags;           /* STARTUP_FLAGS_xxx */
     _PFunc          main;            /* top level procedure of user program */
#ifndef __NO_STACK_OFFSET
     unsigned short  stackOffset;     /* initial value of the stack pointer */
#endif
     unsigned short  nofZeroOuts;     /* number of zero out ranges */
    _Range         *_FAR pZeroOut;  /* vector of ranges with nofZeroOuts elements */
     _Copy          *_FAR toCopyDownBeg;  /* rom-address where copydown-data begins
*/
#if 0  /* switch on to implement ROM libraries */
     unsigned short  nofLibInits;     /* number of library startup descriptors */
     _LibInit        *_FAR libInits;  /* vector of pointers to library startup
descriptors */
#endif
#if defined(__cplusplus)
    unsigned short  nofInitBodies;   /* number of init functions for C++ constructors
*/
     _Cpp           *_FAR initBodies; /* vector of function pointers to init functions for
C++ constructors */
#endif
} _startupData;

#else

extern struct _tagStartup{
     unsigned   flags;
    _PFunc      main;            /* top procedure of user program */
     unsigned   dataPage;        /* page where data allocation begins */
     long       stackOffset;
     int        nofZeroOuts;
    _Range      *_FAR pZeroOut;       /* pZeroOut is a vector of ranges with nofZe-
roOuts elements */
     long       toCopyDownBeg;   /* rom-address where copydown-data begins */
    _PFunc     *_FAR mInits;         /* mInits is a vector of function pointers, ter-
minated by 0 */
    _PFunc      *_FAR libInits;       /* libInits is a vector of function pointers,
terminated by 0x0000FFFF */
} _startupData;

#endif
```

```
#pragma DATA_SEG DEFAULT

extern void _Startup(void);    /* execution begins in this procedure */
/*--------------------------------------------------------------------*/
#ifdef __cplusplus
 }
#endif

#endif
```

7.2.3 MAIN.H

```
/*****************************************************************************\
* Copyright (c) 2002, Motorola Inc.
*
* Motorola Confidential Proprietary
*
* ---------------------------------------------------------------------------*
* File name :     main.h                                                     *
* Project name:   Brushless DC Motor Drive with the MR8 Microcontroller
*
* ---------------------------------------------------------------------------*
* Author :        Jorge Zambada                                              *
* Email :         Jorge.Zambada@motorola.com                                 *
* Department :    Mexico Applications Lab - SPS                              *
*                                                                            *
* Description :   File subroutines and State Flags values are defined in this *
*                 document. Macro definition and new type definition where   *
*                 added here                                                 *
\*****************************************************************************/
#define MOS_2_COM
#undef MOS_3_COM

// New Data type definitions
typedef unsigned short int  UINT16;       // 16 bit unsigned integer    (0, 65535)
typedef signed short int    SINT16;       // 16 bit signed integer (-32768, 32767)
typedef unsigned char       UBYTE;  // 8 bit unsigned byte  (0, 255)
typedef signed char         SBYTE;  // 8 bit signed byte    (-128, 127)

// Function Headers
UBYTE ResolveButtons(void);

// Macro Definitions
#define Forever()                       while(1)
#define EnableInterrupts()              {__asm CLI;}
#define DisableInterrupts()             {__asm SEI;}
```

```
#define DebounceDelay()                    WaitMs(30)
#define WaitUntilUpButtonIsReleased()      while((PORTB & OPTIONS_BUTTON) == 0x00)

// General Boolean defines
#define TRUE  1
#define FALSE 0

// Buttons Definition
#define OPTIONS_BUTTON  0x08
#define ENTER_BUTTON    0x04

// MCU Configuration
#define EDGE_ALIGNED                0x80
#define CENTER_ALIGNED              0x00
#define INDEPENDENT_PWMS            0x10
#define COMPLEMENTARY_MODE          0x00
#define COP_DISABLE                 0x01
#define TOPNEG                      0x20
#define FAULT_1_AUTOMATIC           0x01
#define FAULT_1_MANUAL              0x00
#define FAULT_1_INT                 0x02

/*****************************************************************************\
* End main.h                                                                  *
*****************************************************************************/
```

7.2.4 TIMER.H

```
/*****************************************************************************\
* Copyright (c) 2002, Motorola Inc.
*
* Motorola Confidential Proprietary
*
* ---------------------------------------------------------------------------*
* File name :     timer.h                                                    *
* Project name:   Brushless DC Motor Drive with the MR8 Microcontroller
*
* ---------------------------------------------------------------------------*
* Author :        Jorge Zambada                                              *
* Email :         Jorge.Zambada@motorola.com                                 *
* Department :    Mexico Applications Lab - SPS                              *
*                                                                            *
* Description :   File subroutines and State Flags values are defined in this
*
*                 document. Also Macro definitions are placed here.          *
\*****************************************************************************/
```

Source Code

```
// Function Headers
void InitTimerA(void);
void InitTimerB(void);
void StopMotor(void);
void WaitMs(UBYTE number_of_miliseconds);
void InitPWMMC(void);
void InitPLL(void);
void NextSequence(void);
SINT16 PIController (void);
void MotorStalledProtection(void);
void InitMotor(UBYTE commanded_operation);

// Macro Definitions
#define HallSensorInputs()                (PORTB & 0x70)
#define TurnOffAllPWMOutputs()            (PWMOUT = 0x40)
#define Turn_On_Low_Side_MOSFETs()              (PWMOUT = 0x6A)
#define ResumeTimerA()                    (TASC  &= ~TSTOP)
#define ResumeTimerB()                    (TBSC  &= ~TSTOP)
#define Reset_TimerA()                    (TASC  |= TRST)
#define Reset_TimerB()                    (TBSC  |= TRST)

// Timer Flags
#define Prescaler_by_1  0x00
#define Prescaler_by_2  0x01
#define Prescaler_by_4  0x02
#define Prescaler_by_8  0x03
#define Prescaler_by_16 0x04
#define Prescaler_by_32 0x05
#define Prescaler_by_64 0x06
#define _1milli         0x007D
#define _100milis       0xC350
#define IC_any_Edge     0x0C
#define Port_Control    0x00
#define MAXPERIOD       4605
#define MINPERIOD       237
#define MAXSPEED        126
#define MINSPEED        7
#define MAXINTEGRAL     25000
#define MININTEGRAL     -25000

// Brushless Status and Control
#define HALL_A          0x10
#define HALL_B          0x20
#define HALL_C          0x40
#define CW              0
#define CCW             1
#define BLDCSTOP        0
#define BLDCSPIN        1
#define BLDCWASH        2
#define WASHTABLEPOINTS 256
#define SPINTABLEPOINTS 256
```

```
#define NO_FAULT        0x00
#define MOTOR_STALLED   0x01
#define FAULT_OCCURRED  0x02

// PWM Module
#define _15_625KHz      0x100
#define ZEROPWM         0x80
#define DEADTIME        0x10
#define PWMOFF          0x0000
#define PWMFREQ         _15_625KHz
#define PWMON           PWMFREQ
#define RELOAD_1        0x00
#define RELOAD_2        0x40
#define RELOAD_4        0x80
#define RELOAD_8        0xC0

/*****************************************************************************\
* End timer.h                                                                 *
*****************************************************************************/
```

7.2.5 LCD.H

```
/*****************************************************************************\
* Copyright (c) 2002, Motorola Inc.
*
* Motorola Confidential Proprietary
*
* ---------------------------------------------------------------------------*
* File name :     lcd.h                                                      *
* Project name:   Brushless DC Motor Drive with the MR8 Microcontroller
*
* ---------------------------------------------------------------------------*
* Author :        Jorge Zambada                                              *
* Email :         Jorge.Zambada@motorola.com                                 *
* Department :    Mexico Applications Lab - SPS                              *
*                                                                            *
* Description :   The functions prototypes and some usefull #defines where   *
*                 placed in this document for a better understanding of LCD  *
*                 interface                                                  *
\*****************************************************************************/

// Function Declaration Headers
void WaitMs(UBYTE number_of_miliseconds);
void Wait40us(void);
void InitLCD(void);
void DataLCD(UBYTE data_to_be_displayed);
void StringLCD(UBYTE *pointer_to_string);
```

```
void CtrlLCD(UBYTE control_byte);
void Ctrl8LCD(UBYTE control_byte);
void MovCursorLCD(UBYTE number_of_places, UBYTE direction);

// Macro Definitions
#define Set_E()         (PORTB |= E)
#define Clear_E()       (PORTB &= ~E)
#define Set_RS()        (PORTC |= RS)
#define Clear_RS()      (PORTC &= ~RS)


#define EnableInterrupts()    {__asm CLI;}

// General Defines
#define CLEARLCD              0x01
#define MOVECURSORCOMMAND     0x10
#define MAXLCDMSGS            5
#define RIGHT                 0x04
#define LEFT                  0x00
#define EOS                   0
#define EOL                   '&'
#define First_Column          16

// Control Pins
#define RS                    0x02
#define E                     0x04

// LCD States
#define BLDC_WASH       0
#define BLDC_SPINCW     1
#define BLDC_SPINCCW    2
#define SPEED           3
#define BLDC_STOP       4

/*****************************************************************************\
* End lcd.h                                                                   *
*****************************************************************************/
;
```

7.2.6 TABLES.H

```
;
/* Table used for WASH process of the washing machine */

const SBYTE WASHTable[WASHTABLEPOINTS] =
{14,15,16,18,19,20,22,23,24,25,27,28,29,30,32,33,34,35,36,37,38,40,41,42,43,44,
 45,46,47,48,48,49,50,51,52,53,53,54,55,56,56,57,58,58,59,59,60,60,61,61,61,62,
 62,62,63,63,63,63,64,64,64,64,64,64,64,64,64,64,64,63,63,63,63,63,62,62,62,61,
 61,60,60,59,59,58,58,57,56,56,55,54,54,53,52,51,50,50,49,48,47,46,45,44,43,42,
 41,40,39,38,37,35,34,33,32,31,29,28,27,26,24,23,22,21,19,18,17,15,14,13,-13,
 -15,-16,-17,-19,-20,-21,-22,-24,-25,-26,-28,-29,-30,-31,-32,-34,-35,-36,-37,
 -38,-39,-40,-41,-42,-43,-44,-45,-46,-47,-48,-49,-50,-51,-52,-52,-53,-54,-55,
 -55,-56,-57,-57,-58,-59,-59,-60,-60,-61,-61,-61,-62,-62,-62,-63,-63,-63,-63,
 -64,-64,-64,-64,-64,-64,-64,-64,-64,-64,-64,-64,-63,-63,-63,-63,-62,-62,-62,
 -61,-61,-60,-60,-59,-59,-58,-58,-57,-57,-56,-55,-55,-54,-53,-52,-52,-51,-50,
 -49,-48,-47,-46,-45,-44,-43,-42,-41,-40,-39,-38,-37,-36,-35,-33,-32,-31,-30,
 -29,-27,-26,-25,-24,-22,-21,-20,-18,-17,-16,-14,-13};

/* table used for SPIN process of washing machine */

const SBYTE SPINTable[SPINTABLEPOINTS] =
{14,15,15,15,16,16,16,16,17,17,17,18,18,18,19,19,19,19,20,20,20,21,21,21,22,22,
 22,23,23,23,23,24,24,24,25,25,25,26,26,26,26,27,27,27,28,28,28,28,29,29,29,30,
 30,30,30,31,31,31,32,32,32,32,33,33,33,34,34,34,34,35,35,35,36,36,36,36,37,37,
 37,37,38,38,38,39,39,39,39,40,40,40,40,41,41,41,41,42,42,42,42,43,43,43,43,44,
 44,44,44,45,45,45,45,46,46,46,46,47,47,47,47,47,48,48,48,48,49,49,49,49,49,50,
 50,50,50,51,51,51,51,51,52,52,52,52,52,53,53,53,53,53,53,54,54,54,54,54,55,55,
 55,55,55,55,56,56,56,56,56,56,57,57,57,57,57,57,58,58,58,58,58,58,58,59,59,59,
 59,59,59,59,59,60,60,60,60,60,60,60,60,61,61,61,61,61,61,61,61,61,61,62,62,62,
 62,62,62,62,62,62,62,62,63,63,63,63,63,63,63,63,63,63,63,63,63,63,63,63,63,64,
 64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64,64 };
;
```

7.3 Source Code Files

7.3.1 START08.C

```
;
/*****************************************************************************
  FILE        : start08.c
  PURPOSE     : 68HC08 standard startup code
  LANGUAGE    : ANSI-C / INLINE ASSEMBLER
  ----------------------------------------------------------------------------
  HISTORY
    22 oct 93         Created.
    04/17/97          Also C++ constructors called in Init().
 *****************************************************************************/

#include "start08.h"

/*****************************************************************/
struct _tagStartup _startupData;    /* read-only:
                                       _startupData is allocated in ROM and
                                       initialized by the linker */


#define USE_C_IMPL 0 /* for now, we are using the inline assembler implementation for
the startup code */

#if !USE_C_IMPL
#pragma MESSAGE DISABLE C20001 /* Warning C20001: Different value of stackpointer
depending on control-flow */
/* the function _COPY_L releases some bytes from the stack internally */

#pragma NO_ENTRY
#pragma NO_EXIT
#pragma NO_FRAME
static void near loadByte(void) {
  asm {
            PSHH
            PSHX
            LDA     5,SP
            PSHA
            LDX     7,SP
            PULH
            LDA     0,X
            AIX     #1
            STX     6,SP
            PSHH
            PULX
            STX     5,SP
            PULX
```

```
            PULH
            RTS
  }
}


#endif

extern void _COPY_L(void);
/* DESC:    copy very large structures (>= 256 bytes) in 16 bit address space (stack
incl.)
   IN:      TOS count, TOS(2) @dest, H:X @src
   OUT:
   WRITTEN: X,H */


#ifdef __ELF_OBJECT_FILE_FORMAT__
#define toCopyDownBegOffs 0
#else
#define toCopyDownBegOffs 2 /* for the hiware format, the toCopyDownBeg field is a
long. Because the HC08 is big endian, we have to use an offset of 2 */
#endif
static void Init(void) {
/* purpose:     1) zero out RAM-areas where data is allocated
                2) init run-time data
                3) copy initialization data from ROM to RAM
 */
  unsigned int i;
  int *p;
#if USE_C_IMPL   /* C implementation of ZERO OUT and COPY Down */
  int j;
  char *dst;
  _Range *r;

  r = _startupData.pZeroOut;

  /* zero out */
  for (i=0; i != _startupData.nofZeroOuts; i++) {
    dst = r->beg;
    j = r->size;
    do {
      *dst = 0; /* zero out */
      dst++;
      j--;
    } while(j != 0);
    r++;
  }
#else /* faster and smaller asm implementation for ZERO OUT */
  asm {
```

Source Code

```
ZeroOut:     ;
             LDA    _startupData.nofZeroOuts:1 ; nofZeroOuts
             INCA
             STA    i:1                  ; i is counter for number of zero outs
             LDA    _startupData.nofZeroOuts:0 ; nofZeroOuts
             INCA
             STA    i:0
             LDHX   _startupData.pZeroOut      ; *pZeroOut
             BRA    Zero_5
Zero_3:    ;
            ; CLR    i:1 is already 0
Zero_4:    ;
             ; { HX == _pZeroOut }
             PSHX
             PSHH
             ; { nof bytes in (int)2,X }
             ; { address in (int)0,X   }
             LDA    0,X
             PSHA
             LDA    2,X
             INCA
             STA    p               ; p:0 is used for high byte of byte counter
             LDA    3,X
             LDX    1,X
             PULH
             INCA
             BRA    Zero_0
Zero_1:    ;
            ;  CLRA   A is already 0, so we don't have to clear it
Zero_2:    ;
             CLR    0,X
             AIX    #1
Zero_0:    ;
             DBNZA  Zero_2
Zero_6:
             DBNZ   p, Zero_1
             PULH
             PULX                           ; restore *pZeroOut
             AIX    #4                      ; advance *pZeroOut
Zero_5:    ;
             DBNZ   i:1, Zero_4
             DBNZ   i:0, Zero_3
             ;
CopyDown:    ;

  }

#endif

  /* copy down */
```

```
  /* _startupData.toCopyDownBeg  --->  {nof(16) dstAddr(16) {bytes(8)}^nof} Zero(16)
*/
#if USE_C_IMPL /* (optimized) C implementation of COPY DOWN */
  p = (int*)_startupData.toCopyDownBeg;
  for (;;) {
    i = *p; /* nof */
    if (i == 0) {
      break;
    }
    dst = (char*)p[1]; /* dstAddr */
    p+=2;
    do {
      /* p points now into 'bytes' */
      *dst = *((char*)p); /* copy byte-wise */
      ((char*)p)++;
      dst++;
      i--;
    } while (i!= 0);
  }
#elif defined(__OPTIMIZE_FOR_SIZE__)
  {

  asm {
              LDA    _startupData.toCopyDownBeg:(1+toCopyDownBegOffs)
              PSHA
              LDA    _startupData.toCopyDownBeg:(0+toCopyDownBegOffs)
              PSHA
Loop0:
              JSR    loadByte  ; load high byte counter
              TAX              ; save for compare
              INCA
              STA    i
              JSR    loadByte  ; load low byte counter
              INCA
              STA    i:1
              DECA
              BNE    notfinished
              CBEQX  #0, finished
notfinished:

              JSR    loadByte  ; load high byte ptr
              PSHA
              PULH
              JSR    loadByte  ; load high byte ptr
              TAX              ; HX is now destination pointer
              BRA    Loop1
Loop3:
Loop2:
              JSR    loadByte  ; load data byte
              STA    0,X
              AIX    #1
```

```
Loop1:
            DBNZ   i:1, Loop2
            DBNZ   i:0, Loop3
            BRA    Loop0

finished:
            AIS #2
    }
  }


#else /* optimized asm version. Some bytes (ca 3) larger than C version (when consid-
ering the runtime routine too), but about 4 times faster */
  asm {
            LDX    _startupData.toCopyDownBeg:(0+toCopyDownBegOffs)
            PSHX
            PULH
            LDX    _startupData.toCopyDownBeg:(1+toCopyDownBegOffs)
next:
            LDA   0,X    ; list is terminated by 2 zero bytes
            ORA   1,X
            BEQ copydone
            PSHX         ; store current position
            PSHH
            LDA   3,X    ; psh dest low
            PSHA
            LDA   2,X    ; psh dest high
            PSHA
            LDA   1,X    ; psh cnt low
            PSHA
            LDA   0,X    ; psh cnt high
            PSHA
            AIX   #4
            JSR  _COPY_L ; copy one block
            PULH
            PULX
            TXA
            ADD   1,X    ; add low
            PSHA
            PSHH
            PULA
            ADC   0,X    ; add high
            PSHA
            PULH
            PULX
            AIX   #4
            BRA next
copydone:
  }
#endif
```

```
  /* FuncInits: for C++, this are the global constructors */
#ifdef __cplusplus
#ifdef __ELF_OBJECT_FILE_FORMAT__
  i = _startupData.nofInitBodies - 1;
  while ( i >= 0) {
    (&_startupData.initBodies->initFunc)[i]();  /* call C++ constructors */
    i--;
  }
#else
  if (_startupData.mInits != NULL) {
    _PFunc *fktPtr;
    fktPtr = _startupData.mInits;
    while(*fktPtr != NULL) {
      (**fktPtr)(); /* call constructor */
      fktPtr++;
    }
  }
#endif
#endif
  /* LibInits: used only for ROM libraries */
}

#pragma NO_EXIT
#ifdef __cplusplus
  extern "C"
#endif

void _Startup (void) { /* To set in the linker parameter file: 'VECTOR 0 _Startup' */
/*  purpose:    1)  initialize the stack
                2)  initialize run-time, ...
                    initialize the RAM, copy down init dat etc (Init)
                3)  call main;
    called from: _PRESTART-code generated by the Linker
*/
#ifdef __ELF_OBJECT_FILE_FORMAT__
//asm{
//    mov #$40,$25
//    }
  DisableInterrupts;  /* in HIWARE format, this is done in the prestart code */
#endif
  for (;;) { /* forever: initialize the program; call the root-procedure */
    if (!(_startupData.flags&STARTUP_FLAGS_NOT_INIT_SP)) {
      /* initialize the stack pointer */
      INIT_SP_FROM_STARTUP_DESC();
    }
    Init();
    (*_startupData.main)();
  } /* end loop forever */
}
;
```

7.3.2 MAIN.C

```
;
/*****************************************************************************\
* Copyright (c) 2002, Motorola Inc.
*
* Motorola Confidential Proprietary
*
* ------------------------------------------------------------------------- *
* File name :     main.c                                                    *
* Project name:   Brushless DC Motor Drive with the MR8 Microcontroller
*
* ------------------------------------------------------------------------- *
* Author :        Jorge Zambada                                             *
* Email :         Jorge.Zambada@motorola.com                                *
* Department :    Mexico Applications Lab - SPS                             *
*                                                                           *
* Description :   In this file, the MCU configuration, data initialization and*
*                 an endless loop is implemented. Also a subroutine to  sense
*
*                 push button  changes  and  an algorithm for calculating the
*
*                 desired and actual motor speed.                           *
\*****************************************************************************/

#ifndef  _MAIN_H
#define  _MAIN_H
#include "main.h"
#include "timer.h"
#include "MR8IO.h"
#include "lcd.h"
#endif
/************************** LCD MESSAGES ***********************************/
const UBYTE MSGS[MAXLCDMSGS][13] = {

            {"   BLDC WASH"},
            {"BLDC SPIN CW"},
            {"BLDC SPI CCW"},
            {"   SPEED   &"},
            {"   BLDC STOP"}
};

#pragma DATA_SEG DATA_ZEROPAGE

UBYTE       LCDState = BLDC_WASH,  // Variable for LCD command pointer
            BLDCState = BLDCSTOP,  // State variable for Brushless DC motor
            FAULTState = NO_FAULT; /* State of the FAULT. Motor was stalled or
                                      FAULT 1 occurred */
```

```
/*************************************************************************\
* void main(void): This function includes MCU and its peripherals        *
*                  configuration. Also an endless loop for the main menu *
*                  in the LCD display for user interface                 *
*                                                                        *
* Parameters: None.                                                      *
*                                                                        *
* Return: None.                                                          *
\*************************************************************************/

void main(void) {

      extern UBYTE Required_Direction;

      UBYTE botpressed;             /* This variable is used to store the key
                                       pressed by the user */

      // MCU init

      #ifdef MOS_3_COM
            MOR = CENTER_ALIGNED | COMPLEMENTARY_MODE | COP_DISABLE;
      #endif

      #ifdef MOS_2_COM
            DISMAP = 0x20;
            MOR = CENTER_ALIGNED | TOPNEG | INDEPENDENT_PWMS | COP_DISABLE;
      #endif

      ISCR = IMASK;

      FCR = FAULT_1_MANUAL | FAULT_1_INT;

      InitPLL();

      InitPWMMC();

      // Port init

      PORTA = 0x00;
      PORTB = 0x00;
      PORTC = 0x00;
      DDRA  = 0x0F;
      DDRB  = 0x04;
      DDRC  = 0x02;

      WaitMs(250);
      InitLCD();

      InitTimerA();

      InitTimerB();
```

```
EnableInterrupts();

do
{

      /* At this point of the endless main loop,
         a new string of the main menu is displayed
         in the LCD for user interface */

      CtrlLCD(CLEARLCD);
      StringLCD((UBYTE *)(MSGS[LCDState]));

      /* This function call doesn't return until
         one of the two buttons is pressed and
         released */

      botpressed = ResolveButtons();


      /* The LEFT button is used for changing the LCD
         message for other system functions, such as
         varying BLDC and FAN DC speed, starting and
         stopping both motors, etc. */

      if (botpressed == OPTIONS_BUTTON)
      {
            LCDState = (UBYTE)(LCDState + 1);
            if (LCDState == MAXLCDMSGS) LCDState = BLDC_WASH;
      }

      /* The RIGHT button is used for selecting the
         current function displayed in the LCD */

      else if (botpressed == ENTER_BUTTON)
      {

            /* Function 1. Wash function for a washing machine
                           is selected here. */

            if(LCDState == BLDC_WASH)
            {
                  if(BLDCState == BLDCSTOP)
                  {
                        LCDState = BLDC_STOP;
                        InitMotor(BLDCWASH);
                  }
            }
```

```
                  /* Function 2. Spin CW function for the washing machine */

                  else if(LCDState == BLDC_SPINCW)
                  {
                        if(BLDCState == BLDCSTOP)
                        {
                              LCDState = BLDC_STOP;
                              Required_Direction = CW;
                              InitMotor(BLDCSPIN);
                        }
                  }

                  /* Function 3. Spin CCW function for the washing machine */

                  else if(LCDState == BLDC_SPINCCW)
                  {
                        if(BLDCState == BLDCSTOP)
                        {
                              LCDState = BLDC_STOP;
                              Required_Direction = CCW;
                              InitMotor(BLDCSPIN);
                        }
                  }

                  /* Function 4. At any time, when this function is selected,
                                 the brushless dc motor is stopped and all the
                                 values are reinitialized for another start */

                  else if(LCDState == BLDC_STOP)
                        StopMotor();
            }

      }Forever();

}

UBYTE ResolveButtons(void)
{

      extern      SBYTE RefSpeed,
                        Speed;

      #pragma DATA_SEG DATA_ZEROPAGE

      static      UBYTE buffer = 0; /* used for buffer temporal calculations
                                       of motor actual speed */
      do
      {
            if((PORTB & OPTIONS_BUTTON) == 0x00)
            {
                  DebounceDelay();
```

```
        if((PORTB & OPTIONS_BUTTON) == 0x00)
        {
              WaitUntilUpButtonIsReleased();
              return OPTIONS_BUTTON;
        }
  }
  else
  {
        asm BIH no_button_pressed;
        DebounceDelay();
        asm BIH no_button_pressed;

        asm button_pressed:  /* Wait until DOWN button is released */
              asm BIL button_pressed;
              return ENTER_BUTTON;
        asm no_button_pressed:

  }

  /* For displaying the actual and desired speed select this message.
  This algorithm converts a UBYTE value to ASCII values suitable for
  the LCD display */

  if ((LCDState == SPEED))
  {
        if (RefSpeed < 0)
        {
              buffer = (UBYTE)(-RefSpeed);
              StringLCD("DES-");
        }
        else
        {
              buffer = (UBYTE)RefSpeed;
              StringLCD("DES+");
        }
        DataLCD((UBYTE)(((buffer * 31) / 100) / 10) + '0');
        DataLCD((UBYTE)(((buffer * 31) / 100) % 10) + '0');
        DataLCD((UBYTE)(((buffer * 31) % 100) / 10) + '0');
        if (Speed < 0)
        {
              buffer = (UBYTE)(-Speed);
              StringLCD("0 CU-");
        }
        else
        {
              buffer = (UBYTE)Speed;
              StringLCD("0 CU+");
        }
        DataLCD((UBYTE)(((buffer * 31) / 100) / 10) + '0');
        DataLCD((UBYTE)(((buffer * 31) / 100) % 10) + '0');
        DataLCD((UBYTE)(((buffer * 31) % 100) / 10) + '0');
```

```
            DataLCD('0');
            MovCursorLCD(First_Column, LEFT);
      }

      if (FAULTState != NO_FAULT)
      {
            CtrlLCD(CLEARLCD);
            if (FAULTState == MOTOR_STALLED)
                  StringLCD("Motor Stalled!!!");
      else
                  StringLCD("Fault Occured!!!");
            FAULTState = NO_FAULT;
            LCDState = BLDC_STOP;
      }


}Forever();
}

/*****************************************************************************\
* End main.c                                                                  *
*****************************************************************************/
```

7.3.3 TIMER.C

```
/*****************************************************************************\
* Copyright (c) 2002, Motorola Inc.
*
* Motorola Confidential Proprietary
*
* ---------------------------------------------------------------------------*
* File name :     timer.c                                                    *
* Project name:   Brushless DC Motor Drive with the MR8 Microcontroller
*
* ---------------------------------------------------------------------------*
* Author :        Jorge Zambada                                              *
* Email :         Jorge.Zambada@motorola.com                                 *
* Department :    Mexico Applications Lab - SPS                              *
*                                                                            *
* Description :   The implementation of different motor control algorithms are*
*                 in this document. Also the interrupt handler subroutines are*
*                 here in timer.c                                            *
\*****************************************************************************/
#ifndef  _TIMER_H
#define  _TIMER_H
#include "main.h"
#include "timer.h"
#include "tables.h"
```

```
#include "MR8IO.h"
#include "lcd.h"
#endif

#pragma DATA_SEG DATA_ZEROPAGE

SINT16      newPWM = PWMOFF,        /* variable that indicates the duty cycle
                                       of the BLDC motor windings, and the
         output of the speed controller */
           _newPWM = PWMOFF,/* Negative value of newPWM for
                                       complementary mode */
            P_Portion = 0,          /* Proportional portion of the controller*/
            I_Portion = 0,          /* Integral portion of the controller */
            I_PortionK_1 = 0,       /* Integral portion in last control
                                       action */
            ControllerOutput = 0;   /* Output of the controller */

SBYTE       Speed = MINSPEED,       /* Actual Speed of the motor */
            RefSpeed = MINSPEED,    /* Reference Speed of the motor */
            ControlDifference = 0;  /* Error signal of the controller */

UBYTE       Required_Direction = CW,/* Required direction of motor rotation */
            Actual_Direction = CW,  /* Actual direction of motor rotation */
           Time_Out = 0,/* Used for detecting motor stalled
                                       condition*/
            TempHalls = 0,          /* Used for temporal storage of Hall
                                       sensors */
            P_Gain = 24,            /* Proportional parameter of the
                                       controller */
            I_Gain = 3,             /* Integral parameter of the controller */
            SPINTable_Index = 0,    /* Index used for SPIN process table */
            WASHTable_Index = 0,    /* Index used for WASH process table */
           Milli_Counter = 0;/* Counter of milliseconds to change
                                       reference speed value in the two processes of
                                       the washing machine */

UINT16      Past_Capture = 0,       /* Past value of the capture value in one
                                       of the timer channels */
            Actual_Capture = 0,     /* Actual value of the capture value in one
                                       of the timer channels */
            Dif_Capture = 0;        /* Actual period between captures for speed
                                       calculation */

/*************************************************************************\
* void Init_Motor(UBYTE Commanded_Operation): This subroutine is called from  *
*           main to perform one of the three washing machine processes. The   *
*           process is selected by the parameter value, Commanded_Operation.  *
*                                                                             *
* Parameters: Commanded_Operation.                                            *
*           BLDCWASH. Wash process of the washing machine.                    *
*           BLDCSPIN. Spin process.                                           *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/
```

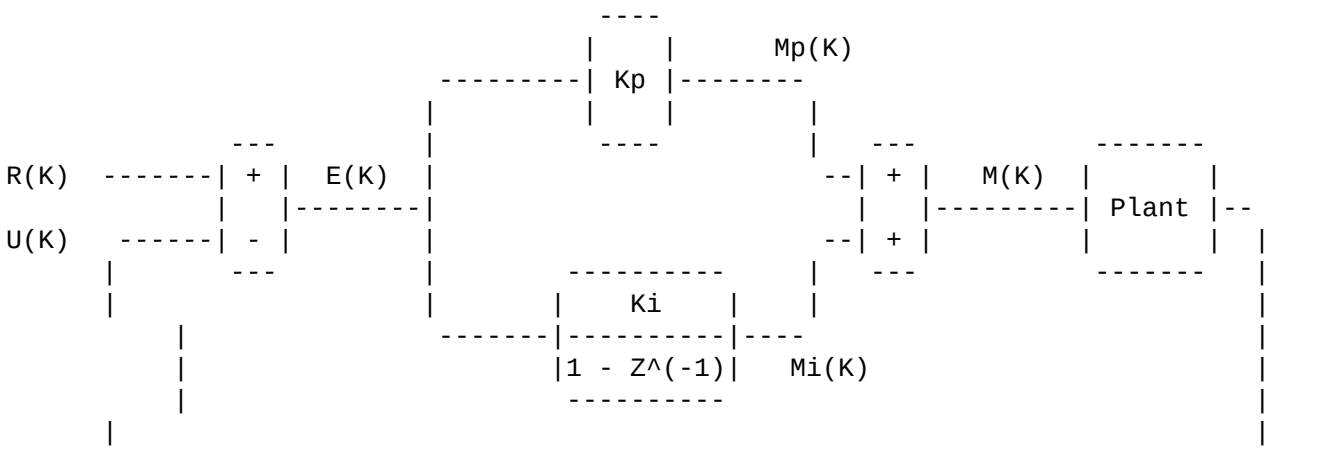
```
void InitMotor(UBYTE Commanded_Operation)
{
      extern UBYTE BLDCState;
      BLDCState = Commanded_Operation;

      /* Initialize Reference speed and pointers to tables */
      if (BLDCState == BLDCWASH)
      {
            WASHTable_Index = 0;
            RefSpeed = WASHTable[WASHTable_Index++];
      }
      else
      {
            SPINTable_Index = 0;
            RefSpeed = SPINTable[SPINTable_Index++];
            if (Required_Direction == CCW)
                  RefSpeed = -RefSpeed;
      }
      /* Initialize variables used for motor control and speed calculation */
      Actual_Capture = MAXPERIOD;
      Past_Capture = 0;
      I_PortionK_1 = 0;
      Milli_Counter = 0;
      Time_Out = 0;

      /* Charge bootstrap capacitors*/
      #ifdef MOS_3_COM
            PVAL1 = PWMOFF;
            PVAL3 = PWMOFF;
            PVAL5 = PWMOFF;
            PCTL1 |= LDOK;
            Turn_On_Low_Side_MOSFETs();
            WaitMs(10);

            PWMOUT = 0x00;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = PWMON;
            PVAL3 = PWMON;
            PVAL5 = PWMON;
            PVAL2 = PWMON;
            PVAL4 = PWMON;
            PVAL6 = PWMON;
            PCTL1 |= LDOK;
            WaitMs(10);

            PVAL2 = PWMOFF;
            PVAL4 = PWMOFF;
            PVAL6 = PWMOFF;
            PCTL1 |= LDOK;
```

```
      #endif

      /* Initialize timers for capture operation and interrupt every 1 ms */

      InitTimerB();
      InitTimerA();
      ResumeTimerB();
      ResumeTimerA();

      newPWM = ZEROPWM;

      return;
}

/*****************************************************************************\
* void InitTimerA (void): This subroutine is called from main and from the    *
*                       subroutine for executing any washing machine process. * *
Its function is to initialize timer A.                *
*                                                                             *
* Parameters: None.                                                          *
*                                                                             *
* Return: None.                                                              *
\*****************************************************************************/

void InitTimerA (void)
{

      /*
      Used for:
            1 Speed control
            2 Commutation
      */

      TASC;
      TASC  = TOIE | TSTOP | TRST | Prescaler_by_64;
      TAMOD = _1milli;
      TASC1 = CHIE | IC_any_Edge; //      HALL A

      return;
}

/*****************************************************************************\
* void InitTimerB (void): This subroutine is called from main and from the    *
*                       subroutine for executing any washing machine process.
*
*                       Its function is to initialize timer A.                *
*                                                                             *
* Parameters: None.                                                          *
*                                                                             *
* Return: None.                                                              *
\*****************************************************************************/
```

```
void InitTimerB (void)
{

      /*
      Used for:
            1 Speed Calculation
            2 Commutation
      */

      TBSC;
      TBSC  = TSTOP | TRST | Prescaler_by_64;
      TBMOD = 0xFFFF;
      TBSC0 = CHIE | IC_any_Edge;   //    HALL B
      TBSC1 = CHIE | IC_any_Edge;   //    HALL C

      return;
}

/*************************************************************************\
* interrupt void TIMA_OV_ISR (void): Interrupt handler subroutine for motor   *
*                       control, motor stalled protection and application     *
*                       management. This interrupt occurs every millisecond.  *
*                                                                             *
* Parameters: None.                                                           *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/

interrupt void TimerAOverflow_ISR (void)  // 519 max, 403 typ
{
      extern UBYTE BLDCState;

      TASC;
      TASC &= ~TOF;

      Dif_Capture = Actual_Capture - Past_Capture;

      if (Dif_Capture > MAXPERIOD)
            Speed = MINSPEED;
      else if (Dif_Capture < MINPERIOD)
            Speed = MAXSPEED;
      else
      {
            /*
                              1665
            Speed =   ------------------
                          (Dif_Capture / 18)
            */
            asm{
                  TXA
```

```
                LDX         #0x12
                DIV
                LDHX  #0x0681
                PSHA
                TXA
                PULX
                DIV
                STA         Speed
            }
        }
        if (Actual_Direction == CCW)
            Speed = -Speed;

        ControllerOutput = PIController();

        if (ControllerOutput < 0)
        {
            ControllerOutput = -ControllerOutput;
            Required_Direction = CCW;
        }
        else
            Required_Direction = CW;

        /*
                     ControllerOutput
        newPWM = ---------------- + 128
                           256
        */
        newPWM = (UBYTE)((UBYTE)(ControllerOutput >> 8) + 0x80);

        MotorStalledProtection();

        Milli_Counter++;

        /* Enters if Milli_Counter > 10 milliseconds */
        if (Milli_Counter > 10)
        {
            Milli_Counter = 0;
            /* Wash Process */
            if (BLDCState == BLDCWASH)
                RefSpeed = WASHTable[WASHTable_Index++];
            /* Spin Process */
            else if (SPINTable_Index != 0)
            {
                RefSpeed = SPINTable[SPINTable_Index++];
                if (Required_Direction == CCW)
                    RefSpeed = -RefSpeed;
            }
        }

        return;
}
```

```
/*****************************************************************************\
* SINT16 PI_Controller (void): This subroutines contains the PI controller    *
*                              implementation.                                *
*                                                                             *
* Parameters: None.                                                           *
*                                                                             *
* Return: SINT16. Controller output.                                          *
\*****************************************************************************/

/*
```




```
      E(K) = R(K) - U(K)

      Mp(K) = E(K) * Kp

      Mi(K) = Mi(K - 1) + E(K) * Ki

      M(K) = Mp(K) + Mi(K)

      where:

            Symbol            Variable Name

            E(K):             ControlDifference
            R(K):             RefSpeed
            U(K):             Speed
            Mp(K):            P_Portion
            Mi(K):            I_Portion
            Mi(K - 1):  I_PortionK_1
            M(K):             ControllerOutput
            Kp:         P_Gain
            Ki:         I_Gain
```

```
*/

SINT16 PIController (void) // 171 max, 152 typ
{

      ControlDifference = RefSpeed - Speed;

      if (ControlDifference >= 0)
      {
            P_Portion = (SINT16)((UBYTE)(ControlDifference) * (UBYTE)(P_Gain));
            I_Portion = (SINT16)((UBYTE)(ControlDifference) * (UBYTE)(I_Gain));
      }
      else
      {
            ControlDifference = -ControlDifference;
            P_Portion = (SINT16)(-((UBYTE)(ControlDifference) * (UBYTE)(P_Gain)));
            I_Portion = (SINT16)(-((UBYTE)(ControlDifference) * (UBYTE)(I_Gain)));
      }

      if (I_PortionK_1 > MAXINTEGRAL)
            I_PortionK_1 = MAXINTEGRAL;
      else if (I_PortionK_1 < MININTEGRAL)
            I_PortionK_1 = MININTEGRAL;

      I_PortionK_1 = I_PortionK_1 + I_Portion;

      return (I_PortionK_1 + P_Portion);
}

/*************************************************************************\
* void Motor_Stalled_Protection (void): This subroutines doesn't let the      *
*                       motor to stop. It calls NextSequence if a period of   *
*                       time has passed and no hall sensor changes have
*
*                       arrived. If a longer period of time has passed with   *
*                       no hall sensor changes, the motor is stopped.         *
*                                                                             *
* Parameters: None.                                                           *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/

void MotorStalledProtection(void) // 140 max, 43 typ
{
extern UBYTE FAULTState;

      Time_Out++;

      /* If no hall sensor interrupt has occured in a timeframe of 250 milli
      second, stop the motor and quit process execution */
      if (Time_Out > 250)
```

```
	{
		StopMotor();
		FAULTState = MOTOR_STALLED;
	}
else
	{
		/* If 8 timeout has completed, a motor stalled protection action is
		taken, calling subroutine NextSequence instead of being called from
		a Hall Effect Sensor Interrupt */
		if ((Time_Out & 0x07) == 0)
		{
			TempHalls = HallSensorInputs();
			NextSequence();
		}
	}

	return;
}

/*************************************************************************\
* interrupt void HALL_A_ISR(void): Interrupt handler subroutine for driving   *
*                                  Hall A input signal. in this interrupts    *
*                                  is called NextSequence Subroutine for      *
*                                  commuting the BLDC motor                   *
*                                                                             *
* Parameters: None.                                                           *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/

interrupt void HallA_ISR (void) // 160 max, 148 typ
{
	TASC1;
	TASC1 &= ~CHF;

	TempHalls = HallSensorInputs();

	/* Compute actual rotor direction from hall effect sensor changes */
	if ( (TempHalls == (HALL_C)) || (TempHalls == (HALL_A | HALL_B)) )
		Actual_Direction = CW;
	else
		Actual_Direction = CCW;

	Time_Out = 0;

	NextSequence();

	return;
}
```

Source Code

```
/*************************************************************************\
* interrupt void HALL_B_ISR(void): Interrupt handler subroutine for driving   *
*                                  Hall B input signal. in this interrupts    *
*                                  is called NextSequence Subroutine for      *
*                                  commuting the BLDC motor                   *
*                                                                             *
* Parameters: None.                                                           *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/

interrupt void HallB_ISR (void) // 160 max, 148 typ
{
      TBSC0;
      TBSC0 &= ~CHF;

      TempHalls = HallSensorInputs();

      /* Compute actual rotor direction from hall effect sensor changes */
      if ( (TempHalls == (HALL_A)) || (TempHalls == (HALL_B | HALL_C)) )
            Actual_Direction = CW;
      else
            Actual_Direction = CCW;

      Time_Out = 0;

      NextSequence();

      return;
}

/*************************************************************************\
* interrupt void HALL_C_ISR(void): Interrupt handler subroutine for driving   *
*                                  Hall C input signal. in this interrupts    *
*                                  is called NextSequence Subroutine for      *
*                                  commuting the BLDC motor. Othe function    *
*                                  of this interrupt handler is to provide    *
*                                  to the overflow interrupt two consecutive  *
*                                  periods of hall changes, to calculate      *
*                                  actual speed.                              *
*                                                                             *
* Parameters: None.                                                           *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/

interrupt void HallC_ISR (void) // 189 max, 177 typ
{
      TBSC1;
      TBSC1 &= ~CHF;
```

```
	/* This hall effect sensor is used as a period feedback for control input
	of the speed of the motor */
	Past_Capture = Actual_Capture;
	Actual_Capture = TBCH1;

	TempHalls = HallSensorInputs();

	/* Compute actual rotor direction from hall effect sensor changes */
	if ( (TempHalls == (HALL_B)) || (TempHalls == (HALL_A | HALL_C)) )
		Actual_Direction = CW;
	else
		Actual_Direction = CCW;

	Time_Out = 0;

	NextSequence();

	return;
}

/*************************************************************************\
* void NextSequence (void): This subroutine has all the posible combinations  *
*                           of hall effect sensor inputs and direction of the
*
*                           motor, to properly commutate it.                  *
*                                                                             *
* Parameters: None.                                                           *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/

void NextSequence(void) //108 max, 96 typ
{

	#ifdef MOS_3_COM
		_newPWM = PWMFREQ - newPWM;
	#endif

	#ifdef MOS_2_COM
		#pragma DATA_SEG DATA_ZEROPAGE
		static SINT16 backupnewPWM;
		backupnewPWM = newPWM;

		_newPWM = newPWM;
		newPWM = PWMFREQ - newPWM;
	#endif

	/* This commutation truth table is based on "Commutate truth table.xls"*/
	if (Required_Direction == CW)
	{
		if (TempHalls == (HALL_A))
```

```
{
      #ifdef MOS_3_COM
            PVAL1 = newPWM;
            PVAL3 = _newPWM;
            PVAL5 = newPWM;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = newPWM;
            PVAL2 = PVAL1 - DEADTIME;

            PVAL3 = _newPWM;
            PVAL4 = PVAL3 - DEADTIME;

            PVAL5 = PWMON;
            PVAL6 = PWMOFF;
      #endif
}
else if (TempHalls == (HALL_A | HALL_C))
{
      #ifdef MOS_3_COM
            PVAL1 = _newPWM;
            PVAL3 = _newPWM;
            PVAL5 = newPWM;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = PWMON;
            PVAL2 = PWMOFF;

            PVAL3 = _newPWM;
            PVAL4 = PVAL3 - DEADTIME;

            PVAL5 = newPWM;
            PVAL6 = PVAL5 - DEADTIME;
      #endif
}
else if (TempHalls == (HALL_C))
{
      #ifdef MOS_3_COM
            PVAL1 = _newPWM;
            PVAL3 = newPWM;
            PVAL5 = newPWM;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = _newPWM;
            PVAL2 = PVAL1 - DEADTIME;

            PVAL3 = PWMON;
            PVAL4 = PWMOFF;
```

```
            PVAL5 = newPWM;
            PVAL6 = PVAL5 - DEADTIME;
      #endif
}
else if (TempHalls == (HALL_B | HALL_C))
{
      #ifdef MOS_3_COM
            PVAL1 = _newPWM;
            PVAL3 = newPWM;
            PVAL5 = _newPWM;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = _newPWM;
            PVAL2 = PVAL1 - DEADTIME;

            PVAL3 = newPWM;
            PVAL4 = PVAL3 - DEADTIME;

            PVAL5 = PWMON;
            PVAL6 = PWMOFF;
      #endif
}
else if (TempHalls == (HALL_B))
{
      #ifdef MOS_3_COM
            PVAL1 = newPWM;
            PVAL3 = newPWM;
            PVAL5 = _newPWM;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = PWMON;
            PVAL2 = PWMOFF;

            PVAL3 = newPWM;
            PVAL4 = PVAL3 - DEADTIME;

            PVAL5 = _newPWM;
            PVAL6 = PVAL5 - DEADTIME;
      #endif
}
else if (TempHalls == (HALL_A | HALL_B))
{
      #ifdef MOS_3_COM
            PVAL1 = newPWM;
            PVAL3 = _newPWM;
            PVAL5 = _newPWM;
      #endif
```

```
                #ifdef MOS_2_COM
                        PVAL1 = newPWM;
                        PVAL2 = PVAL1 - DEADTIME;

                        PVAL3 = PWMON;
                        PVAL4 = PWMOFF;

                        PVAL5 = _newPWM;
                        PVAL6 = PVAL5 - DEADTIME;
                #endif
        }
    }
    else
    {
        if (TempHalls == (HALL_A))
        {
                #ifdef MOS_3_COM
                        PVAL1 = _newPWM;
                        PVAL3 = newPWM;
                        PVAL5 = newPWM;
                #endif

                #ifdef MOS_2_COM
                        PVAL1 = _newPWM;
                        PVAL2 = PVAL1 - DEADTIME;

                        PVAL3 = newPWM;
                        PVAL4 = PVAL3 - DEADTIME;

                        PVAL5 = PWMON;
                        PVAL6 = PWMOFF;
                #endif
        }
        else if (TempHalls == (HALL_A | HALL_C))
        {
                #ifdef MOS_3_COM
                        PVAL1 = _newPWM;
                        PVAL3 = newPWM;
                        PVAL5 = _newPWM;
                #endif

                #ifdef MOS_2_COM
                        PVAL1 = PWMON;
                        PVAL2 = PWMOFF;

                        PVAL3 = newPWM;
                        PVAL4 = PVAL3 - DEADTIME;

                        PVAL5 = _newPWM;
                        PVAL6 = PVAL5 - DEADTIME;
                #endif
```

```
}
else if (TempHalls == (HALL_C))
{
      #ifdef MOS_3_COM
            PVAL1 = newPWM;
            PVAL3 = newPWM;
            PVAL5 = _newPWM;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = newPWM;
            PVAL2 = PVAL1 - DEADTIME;

            PVAL3 = PWMON;
            PVAL4 = PWMOFF;

            PVAL5 = _newPWM;
            PVAL6 = PVAL5 - DEADTIME;
      #endif
}
else if (TempHalls == (HALL_B | HALL_C))
{
      #ifdef MOS_3_COM
            PVAL1 = newPWM;
            PVAL3 = _newPWM;
            PVAL5 = _newPWM;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = newPWM;
            PVAL2 = PVAL1 - DEADTIME;

            PVAL3 = _newPWM;
            PVAL4 = PVAL3 - DEADTIME;

            PVAL5 = PWMON;
            PVAL6 = PWMOFF;
      #endif
}
else if (TempHalls == (HALL_B))
{
      #ifdef MOS_3_COM
            PVAL1 = newPWM;
            PVAL3 = _newPWM;
            PVAL5 = newPWM;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = PWMON;
            PVAL2 = PWMOFF;
```

```
                        PVAL3 = _newPWM;
                        PVAL4 = PVAL3 - DEADTIME;

                        PVAL5 = newPWM;
                        PVAL6 = PVAL5 - DEADTIME;
                  #endif
            }
            else if (TempHalls == (HALL_A | HALL_B))
            {
                  #ifdef MOS_3_COM
                        PVAL1 = _newPWM;
                        PVAL3 = _newPWM;
                        PVAL5 = newPWM;
                  #endif

                  #ifdef MOS_2_COM
                        PVAL1 = _newPWM;
                        PVAL2 = PVAL1 - DEADTIME;

                        PVAL3 = PWMON;
                        PVAL4 = PWMOFF;

                        PVAL5 = newPWM;
                        PVAL6 = PVAL5 - DEADTIME;
                  #endif
            }
      }

      PCTL1 |= LDOK;

      #ifdef MOS_2_COM
            newPWM = backupnewPWM;
      #endif

      return;
}

/*************************************************************************\
* void init_PWMMC (void):Initialization of the PWM module is implemented      *
*                        in this subrouine and the frequency is set to
*
*                        15.625 kHz.                                          *
*                                                                             *
* Parameters: None.                                                           *
                                                                              *
* Return: None.                                                               *
\*************************************************************************/
```

```
void InitPWMMC(void)
{
      PMOD = PWMFREQ;   // Frequency of 15.625 KHz

      #ifdef MOS_3_COM
            PVAL1 = PWMOFF;
            PVAL3 = PWMOFF;
            PVAL5 = PWMOFF;
            DEADTM = DEADTIME;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = PWMON;
            PVAL3 = PWMON;
            PVAL5 = PWMON;
            PVAL2 = PWMOFF;
            PVAL4 = PWMOFF;
            PVAL6 = PWMOFF;
      #endif

      PCTL2 = RELOAD_1;       /* Reload every 4 PWM cycle. Fop=Fbus=8000000 Hz.
                                 PWMFreq = 8MHz / (2*256) = 15.625 kHz
                                 Reload Freq = 15.625 kHz / 4 = 3.90625 kHz */

      PCTL1 = PWMEN;          // Turn on PWM module

      PCTL1 |= LDOK;

      return;
}

/*************************************************************************\
* void stop_motor (void): The motor is stopped in this subroutine, either     * *
for user command or motor stalled.                    *
*                                                                           *
* Parameters: None.                                                         *
*                                                                           *
* Return: None.                                                             *
\*************************************************************************/

void StopMotor(void)
{

      extern      UBYTE BLDCState;

      InitTimerA();
      InitTimerB();

      BLDCState = BLDCSTOP;

      #ifdef MOS_3_COM
```

```
            TurnOffAllPWMOutputs();
            PVAL1 = PWMOFF;
            PVAL3 = PWMOFF;
            PVAL5 = PWMOFF;
      #endif

      #ifdef MOS_2_COM
            PVAL1 = PWMON;
            PVAL3 = PWMON;
            PVAL5 = PWMON;
            PVAL2 = PWMOFF;
            PVAL4 = PWMOFF;
            PVAL6 = PWMOFF;
      #endif

      PCTL1 |= LDOK;

      return;
}

/*************************************************************************\
* void init_PLL (void): PLL is initialized to run at 8 MHz of Bus frequency   *
*                                                                             *
* Parameters: None.                                                           *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/

void InitPLL(void)  // Fbus = 8000000 +/- 2% Hz
{
      PCTL &= ~BCS;                 // select external reference as base clock
      PCTL &= ~PLLON;         // turn off PLL
      PPG  = 0x86;                  // program N and L
      PBWC |= AUTO;                 // enable automatic bandwidth control
      PCTL |= PLLON;          // turn on PLL
      while((PBWC & LOCK)==0);      // wait for PLL to lock
      PCTL |= BCS;
      return;
}

/*************************************************************************\
* interrupt void Fault1_ISR(void): Interrupt handler subroutine for Fault1.   *
*                                 The motor is stopped when a FAULT occurs.  *
*                                 The FAULT is asserted when the current     *
*                                 limit or voltage limit has been reached by *
*                                 the power stage.                           *
*                                                                       *
* Parameters: None.                                                       *
*                                                                       *
* Return: None.                                                         *
\*************************************************************************/
```

```
interrupt void Fault1_ISR (void)
{
      extern UBYTE FAULTState;
      StopMotor();
      FTAC |= FTACK1;
      FAULTState = FAULT_OCCURED;
      return;
}

interrupt void Error_Trap (void)
{
      return;
}

/*****************************************************************************\
* End timer.c                                                                 *
*****************************************************************************/
```

7.3.4 LCD.C

```
/*****************************************************************************\
* Copyright (c) 2002, Motorola Inc.
*
* Motorola Confidential Proprietary
*
* ---------------------------------------------------------------------------*
* File name :     lcd.c                                                      *
* Project name:   Brushless DC Motor Drive with the MR8 Microcontroller
*
* ---------------------------------------------------------------------------*
* Author :        Jorge Zambada                                              *
* Email :         Jorge.Zambada@motorola.com                                 *
* Department :    Mexico Applications Lab - SPS                              *
*                                                                            *
* Description :   The LCD interface and delay subroutines are implemented in  *
*                 this file.                                                 *
\*****************************************************************************/
#ifndef  _LCD_H
#define  _LCD_H
#include "main.h"
#include "MR8IO.h"
#include "lcd.h"
#endif
```

```
/*************************************************************************\
* void init_LCD(void): Subroutine to initialize the LCD character display for *
*                      4-bit operation, blink off, display on.            *
*                                                                         *
* Parameters: None.                                                       *
*                                                                         *
* Return: None.                                                           *
\*************************************************************************/

void InitLCD(void)
{

      /* Sequence followed for LCD initialization */

      // In 8 bit operation mode
      WaitMs(15);
      Ctrl8LCD(0x03);   // Set 8 bit operation
      WaitMs(5);
      Ctrl8LCD(0x03);   // Set 8 bit operation
      WaitMs(1);
      Ctrl8LCD(0x03);   // Set 8 bit operation
      Ctrl8LCD(0x02);   // Set 4 bit operation

      // In 4 bit operation mode
      CtrlLCD(0x28); // 4 bit operation with 2 line display
      CtrlLCD(0x06);    // No display shift and move right
      CtrlLCD(0x01);    // Clear display and return home position
      CtrlLCD(0x0C);    // Display on, cursor off and blink off
      return;
}

/*************************************************************************\
* void ctrl_LCD(void): Subroutine for sending control bytes to the LCD. This
     *
*                      routine send the 8 bit value in two parts, since this  *
*                      function is called in 4 bit operation mode.            *
*                                                                             *
* Parameters: ctrl. An 8 bit value for different control of the LCD, such as  *
*                   number of lines, blink on or off, etc.                    *
*                                                                             *
* Return: None.                                                               *
\*****************************************************************************/

void CtrlLCD(UBYTE ctrl)
{
      // Upper Nibble

      PORTA &= 0xF0;          // puting pin states of the LCD in PORTA pins
      PORTA |= (ctrl >> 4) & 0x0F;
      Set_E();
      Clear_E();
```

```
      Wait40us();

      // Lower Nibble

      PORTA &= 0xF0;          // puting pin states of the LCD in PORTA pins
      PORTA |= ctrl & 0x0F;
      Set_E();
      Clear_E();

      if ((ctrl==0x01) || (ctrl==0x02)) WaitMs(2);
      Wait40us();

      return;
}

/*************************************************************************\
* void ctrl8LCD(void): Subroutine for sending control bytes to the LCD in 8   *
*                      bit mode. use this function only to enter 4-bit mode,  *
*                      since the other 4 data pins have no connection         *
*                                                                             *
* Parameters: ctrl. An 8 bit value for different control of the LCD, such as  *
*                   number of lines, blink on or off, etc.                    *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/

void Ctrl8LCD(UBYTE ctrl)
{

      PORTA &= 0xF0;          // puting pin states of the LCD in PORTA pins
      PORTA |= ctrl & 0x0F;
      Set_E();
      Clear_E();

      Wait40us();
      return;
}

/*************************************************************************\
* void mov_cursor_LCD(UBYTE places, UBYTE dir): subroutine to move the LCD    *
*                                               cursor to RIGHT or LEFT the   *
*                                               the number of places the user *
*                                               wants specyfied in 'places'   *
*                                                                             *
* Parameters: places. Number of places wanted to move the LCD cursor without  *
*                     affecting any LCD actual message.                       *
*             dir. Direction in which the cursor is to be moved. RIGHT or     *
*                  LEFT.                                                      *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/
```

Source Code

```
void MovCursorLCD(UBYTE places, UBYTE dir)
{
      UBYTE ctrl_byte = 0x10 | dir;
      do
      {
            CtrlLCD(ctrl_byte);
      }while((--places)>0);

      return;
}

/*****************************************************************************\
* void data_LCD(UBYTE data): ASCII symbol to be displayed in the LCD in the   *
*                            current cursor position.                         *
*                                                                             *
* Parameters: data. 8-bit value representing the ASCII code of the symbol     *
*                   to be displayed in the LCD at current position            *
*                                                                             *
* Return: None.                                                               *
\*****************************************************************************/

void DataLCD(UBYTE data)
{
      // Upper Nibble
      PORTA &= 0xF0;          // puting pin states of the LCD in PORTA pins
      PORTA |= (data >> 4) & 0x0F;
      Set_RS();
      Set_E();
      Clear_E();

      // Lower Nibble
      PORTA &= 0xF0;          // puting pin states of the LCD in PORTA pins
      PORTA |= data & 0x0F;
      Set_E();
      Clear_E();

      Wait40us();

      Clear_RS();

      return;
}
```

```
/*************************************************************************\
* void string_LCD(UBYTE *msgLCD): A function that displays a string in the LCD*
*                                 at current cursor position. If a '&' cha-   *
*                                 racter is present in the string, a new line
*
*                                 is commanded in the LCD. the function send  *
*                                 all the bytes in the string until a presense*
*                                 of a EndOfString, EOS or 0x00 byte.         *
*                                                                             *
* Parameters: *msgLCD. Pointer to the string to be displayed in the LCD
*
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/

void StringLCD(UBYTE *msgLCD)
{
      while(*msgLCD != EOS)
      {
            if(*msgLCD == EOL) MovCursorLCD(29,RIGHT);      // new line
            else DataLCD(*msgLCD);
            msgLCD++;
      }
      return;
}

/*************************************************************************\
* void wait_ms(UBYTE milis): Delay routine that waits for a number of milli-  *
*                            seconds send in the parameter milis. the delay   *
*                            is calculated for a 8 MHz Fbus operation.        *
*                                                                             *
* Parameters: milis. A 8 bit value representing the number of milliseconds the*
*                    delay will wait.                                         *
*                                                                             *
* Return: None.                                                               *
\*************************************************************************/

void WaitMs(UBYTE milis)
{

      UBYTE wait40usCount = 0;      // used for counting wait40us delay

      do{
            for(wait40usCount = 0; wait40usCount < 24; wait40usCount++)
                  Wait40us();
      }while((--milis) != 0);
      return;
}
```

Source Code

```
/*****************************************************************************\
* void wait40us(void): An instant of time of which the wait_ms() subroutine is*
*                      based on.                                              *
*                                                                             *
* Parameters: None                                                            *
*                                                                             *
*                                                                             *
* Return: None.                                                               *
\*****************************************************************************/

void Wait40us(void)
{

      UBYTE count = 103;            // Value for 40us delay at Fbus = 8 MHz

      do{
      }while(--count);
      return;
}

/*****************************************************************************\
* End lcd.c                                                                   *
*****************************************************************************/
;
```

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://motorola.com/semiconductors

MOTOROLA

DRM007/D
Rev. 0
2/2003