Product Not Recommended for New Designs

XILINX®

FPGA Motor Control Reference Design

Author: Craig Hackney

XAPP808 (v1.0) September 16, 2005

Summary

With the growing complexity of motor and motion control applications, it becomes apparent that a Field Programmable Gate Array (FPGA) offers significant advantage over the off the shelf Application Specific Standard Product (ASSP) solutions in the areas of performance, flexibility and inventory control. With an FPGA, calculations that would normally consume large amounts of CPU time when implemented in software may be hardware accelerated. Using hardware acceleration allows for more functionality within the system software. Custom motor drive interfaces such as PWM can be developed easily, quickly and at low cost. Additionally, because of full configurability, the same FPGA can be used in various product ranges, reducing the need to maintain inventory for multiple devices.

This application note describes the Spartan™-3 device based FPGA Motor Control Reference Design. This design utilizes the MicroBlaze™ 32-bit CPU soft core, floating point unit, and associated memory subsystems, SPI communications interface, and IP specifically designed to control brushless DC (BLDC) and three phase AC induction motors.

Design files can be found at www.xilinx.com/bvdocs/appnotes/xapp808.zip.

Figure 1: **Motor Control Design Assembly**

Functional Overview

A Windows based Graphical User Interface (GUI) is used to transfer commands to the embedded development boards controlling the motors. These commands include the speed at which the motor should rotate, how often the speed of the motor should be sampled, and the gain values for the control loop used to maintain the speed of the motors. The GUI displays the current speed and drive voltage as well as the average, and standard deviation of the speed and drive voltage calculated from the last 256 samples received from the motor controllers.

The software running on the embedded motor controller boards calculates the current speed of the motors, and using a Proportional Integral Derivative (PID) control loop and the gain values from the GUI, attempts to maintain the speed of the motor. Each time the speed of the motor is

sampled, the motor speed and drive voltage information is transmitted back to the host based GUI where it is displayed in a graphical format.
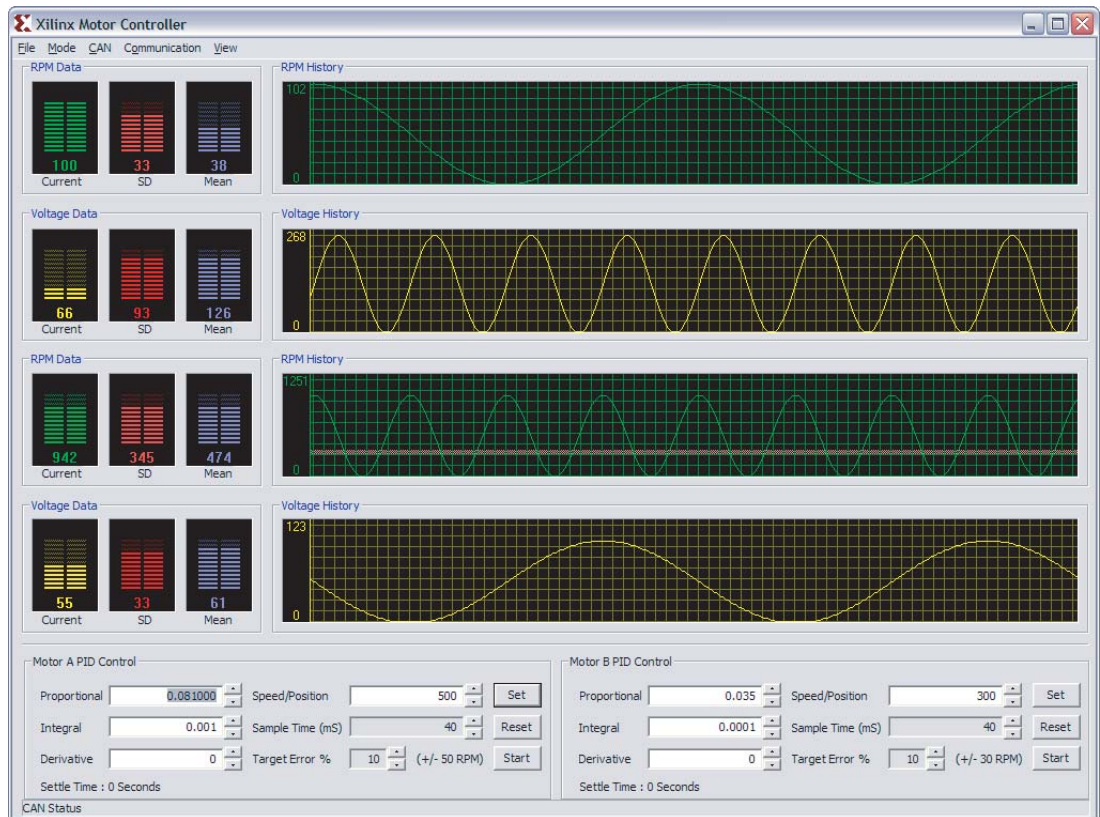

*Figure 2:* **Windows Based Graphical User Interface**

Hardware Overview

The reference system consists of two motors, one BLDC motor (IB23810) from MCG and one three phase AC induction motor (2IK6A-SW) from Oriental Motor U.S.A. Corp. Each motor is fitted with an optical shaft encoder of type HEDS-564X-AXX from Agilent Technologies. The BLDC motor also contains three Hall effect sensors used to determine the commutation for the motor.

The BLDC motor utilizes a 12VDC power stage (ECLOVACBLDC) from Freescale Semiconductor Inc., while the AC induction motor utilizes a 110-220VAC power stage from International Rectifier based on the IRADK10 reference design kit. For this design the PIC was removed from the IRADK10 board and pins 16 (RESETSC) and 22 (ENABLE) of U1 were tied to +5V (pin 20).

Isolation circuits were used to separate the low-voltage controller circuitry from the high-voltage motor power stages, the schematics for which are located in the *Datasheets* directory within the ZIP file for this reference design.

Xilinx SP305 embedded development boards were used as the controllers for the motors. These boards utilize a Spartan-3 (XC3S1500) FPGA, a stand-alone CAN controller with an SPI interface, an eight channel stand-alone ADC also with an SPI interface, Ethernet and RS232 communication interfaces, an abundance of IO, static and dynamic memory systems available for use by the FPGA, and more.

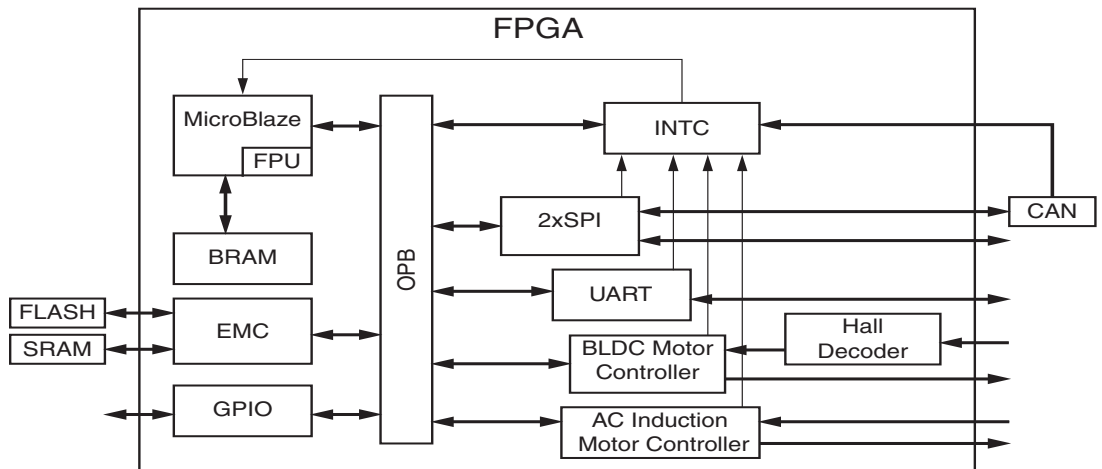Figure 3 depicts a high-level overview of the reference design.
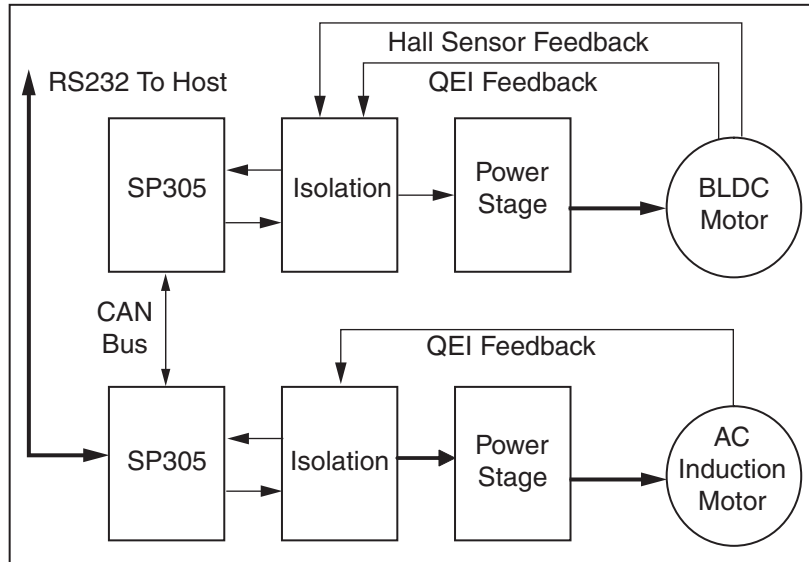
xapp408_03

Figure 3: **High-Level System Overview**

EDK Reference System

The [Embedded Development Kit](#) ([EDK](#)) Reference System can be found in the ZIP file under the *Designs/SP305_RevB* directory. The figure below depicts the IP instantiated within the FPGA for this design.

xapp408_04

Figure 4: **EDK Reference Design Block Diagram**

In this reference design the motor controller application is stored in the Flash memory on the SP305 embedded development board. Once the FPGA is configured, a small boot loader application located within the Block RAM is executed; this application copies the main application code from the Flash to SRAM before executing it. The [EDK](#) reference system contains both the motor controller application and the boot loader.

Software Memory Map

Table 1 lists the memory ranges utilized by the IP instantiated for the reference design. Refer to the IP modules' product specifications' for detailed software register descriptions, and **XAPP448** for detailed information relating to PWM and sine wave generation methods.

Table 1: **Software Memory Map**

Address Range	Utilization
0x81002000 – 0x81003fff	Induction Motor Sine Table
0x81000c00 – 0x81000cff	Induction Motor Controller
0x81000b00 – 0x81000bff	BLDC Motor Controller
0x81000a00 – 0x81000aff	Interrupt Controller
0x81000900 – 0x810009ff	SPI_1
0x81000800 – 0x810008ff	SPI_0
0x81000600 – 0x810007ff	GPIO_2
0x81000400 – 0x810005ff	GPIO_1
0x81000200 – 0x810003ff	GPIO_0
0x81000000 – 0x810000ff	UART
0x80800000 – 0x80ffffff	Flash
0x80000000 – 0x000fffff	SRAM
0x00000000 – 0x00003fff	Block RAM

Boot Loader

The source code for the boot loader is located in the *Designs/code/bootloader_0* directory of the reference design zip file. This is the standard boot loader generated by the EDK; it should be used as the initial image in the Block RAM that MicroBlaze uses to boot from. Its purpose is to read the application image programmed into the Flash memory on the development board and transfer it to SRAM. When the transfer is complete, execution is passed to the image in SRAM.

This boot procedure is used for two reasons; first, the main application image is too large to fit into the Block RAM on the FPGA device; second, Flash access is typically slower than SRAM access; therefore, in order to speedup the overall execution speed of the application, the application image is copied from Flash to SRAM before it is executed.

Motor Controller Application Code

The motor controller application source code is located in the *Designs/code/motor_controller* directory of the reference design Zip file. When the application code runs, one of the first things it does is to read the "DIP Switches" to determine the desired configuration of the motor controller.

If configured to control an AC induction motor, the application generates 90° of a sine wave and stores it into the space allocated for the induction motor sine table (see the "Software Memory Map" section) before going on to initializing the Motor Controller, SPI, CAN, UART interfaces (see Figure 5). The application is entirely interrupt driven, meaning that once the system initialization is complete, the main() function does nothing more than flash an LED as an indication that the application is running.
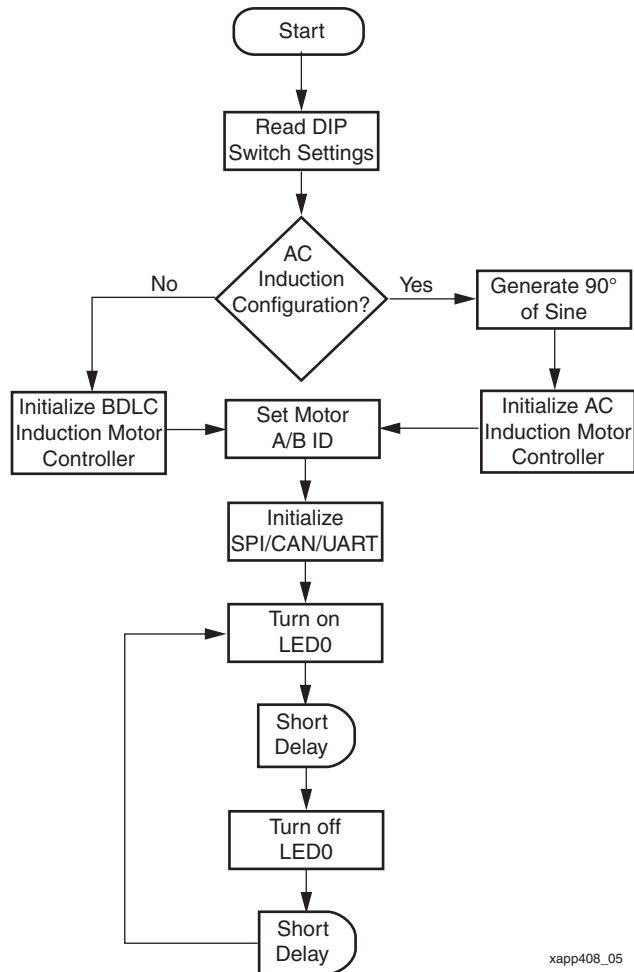
```
                        ┌──────────┐
                        │  Start   │
                        └────┬─────┘
                             │
                    ┌────────┴────────┐
                    │ Read DIP        │
                    │ Switch Settings │
                    └────────┬────────┘
                             │
                         ╱───┴───╲
                        ╱   AC    ╲
          No           ╱ Induction ╲          Yes      ┌──────────────┐
        ◄─────────────◄ Configuration?├───────────────►│ Generate 90° │
                        ╲           ╱                   │ of Sine      │
                         ╲─────────╱                    └──────┬───────┘
                             │                                 │
  ┌───────────────┐   ┌──────┴──────┐              ┌──────────┴────────┐
  │ Initialize BDLC│  │ Set Motor   │              │ Initialize AC     │
  │ Induction Motor├─►│ A/B ID      │◄─────────────┤ Induction Motor   │
  │ Controller    │   └──────┬──────┘              │ Controller        │
  └───────────────┘          │                     └───────────────────┘
                    ┌────────┴────────┐
                    │ Initialize      │
                    │ SPI/CAN/UART    │
                    └────────┬────────┘
                             │
                    ┌────────┴────────┐
              ┌────►│ Turn on         │
              │     │ LED0            │
              │     └────────┬────────┘
              │        ┌─────┴─────┐
              │        │ Short     │
              │        │ Delay     │
              │        └─────┬─────┘
              │     ┌────────┴────────┐
              │     │ Turn off        │
              │     │ LED0            │
              │     └────────┬────────┘
              │        ┌─────┴─────┐
              └────────┤ Short     │        xapp408_05
                       │ Delay     │
                       └───────────┘
```

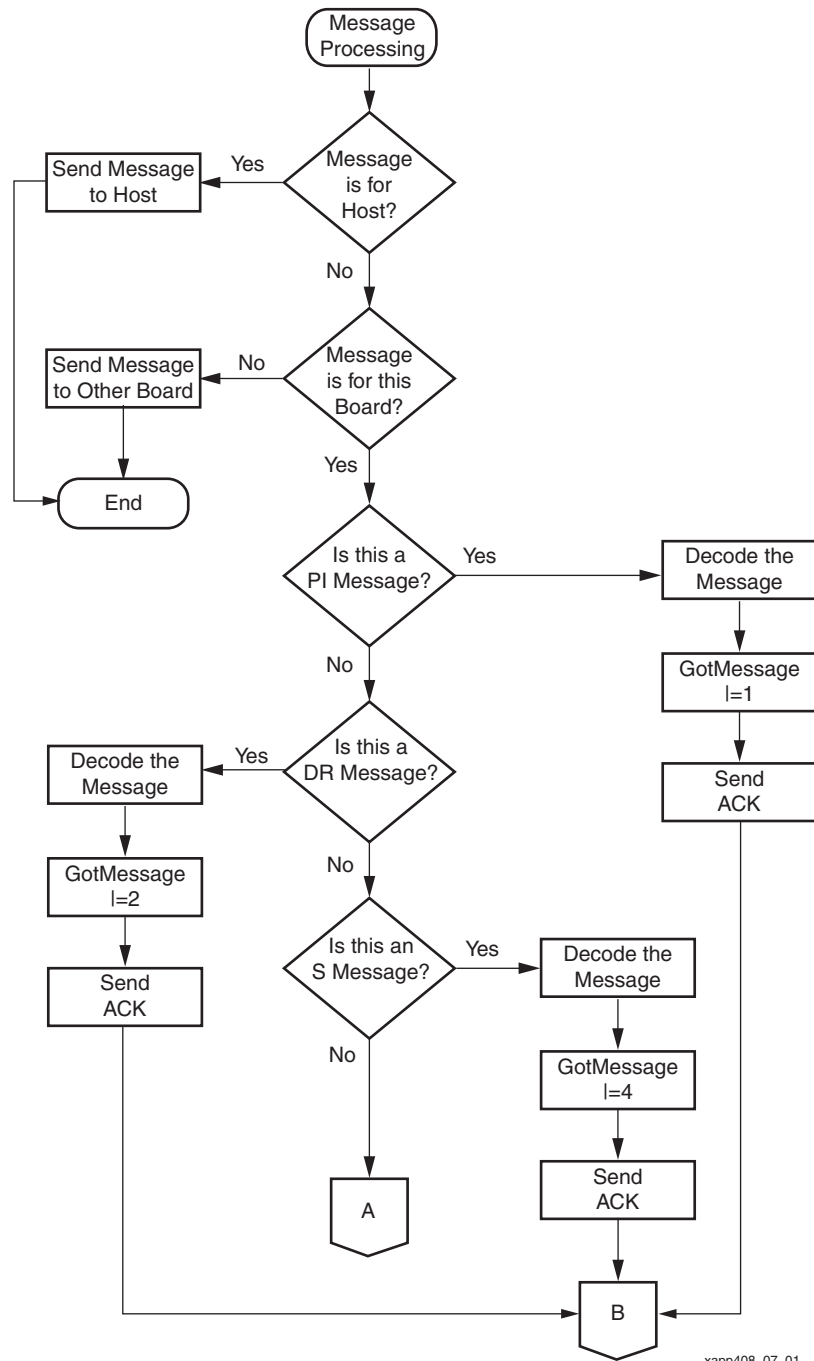*Figure 5:* **Initialization Sequence Flow Chart**

The motor controller IP contains a *Speed Unit* that is configured to generate a processor interrupt every 10ms. The idea behind the speed unit is to count the number of optical shaft encoder lines within a fixed period of time, which in this case is 10ms. Given that the total number of lines on the shaft encoder is known to be 2000, and the sample period is fixed at 10ms, the rotational speed of the motor can be calculated. The state of LED1 is toggled each time a *Speed Unit* interrupt is detected when the motor is enabled, when the motor is disabled, LED1 is turned off.
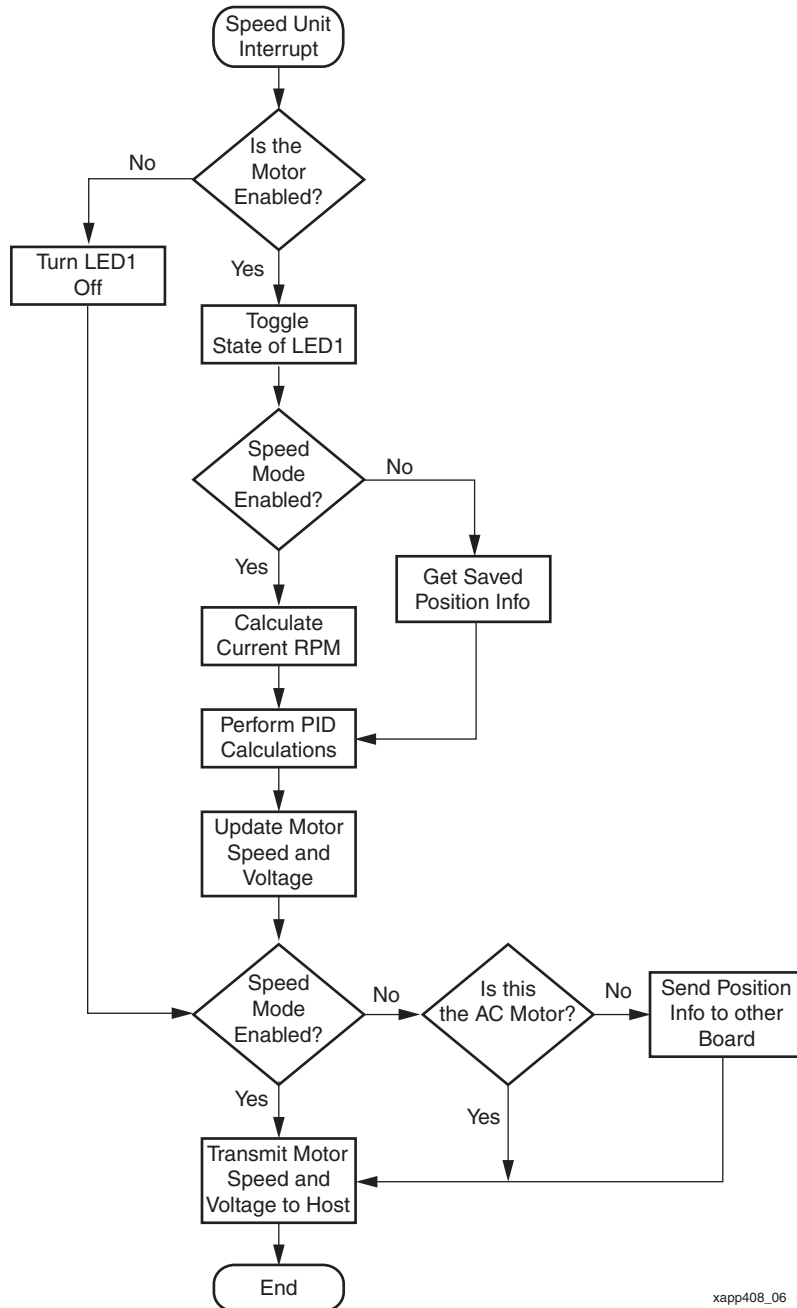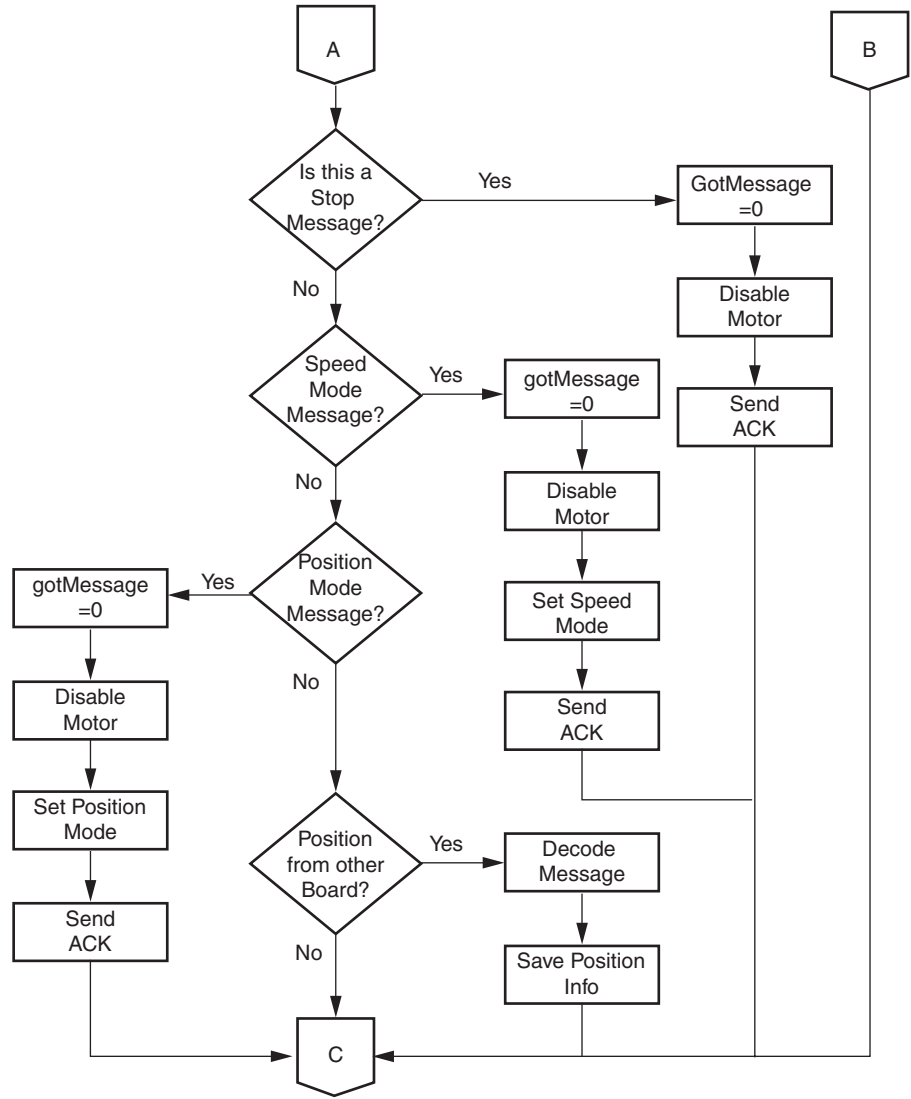
Figure 6: **Speed Unit Interrupt Flow Chart**

Every time a communication message is received via the CAN or RS232 communication interfaces a message processing function is called to determine what to do with the received message. The flow chart for this message processing function is shown below in Figure 7.

Message
Processing

Message
is for
Host?

Yes → Send Message
to Host

No

Message
is for this
Board?

No → Send Message
to Other Board → End

Yes

Is this a
PI Message?

Yes → Decode the
Message → GotMessage
l=1 → Send
ACK → B

No

Is this a
DR Message?

Yes → Decode the
Message → GotMessage
l=2 → Send
ACK → B

No

Is this an
S Message?

Yes → Decode the
Message → GotMessage
l=4 → Send
ACK → B

No → A

xapp408_07_01

Figure 7: **Message Processing Flow Chart (Part 1)**

xapp408_07_02

Figure 8: **Message Processing Flow Chart (Part 2)**
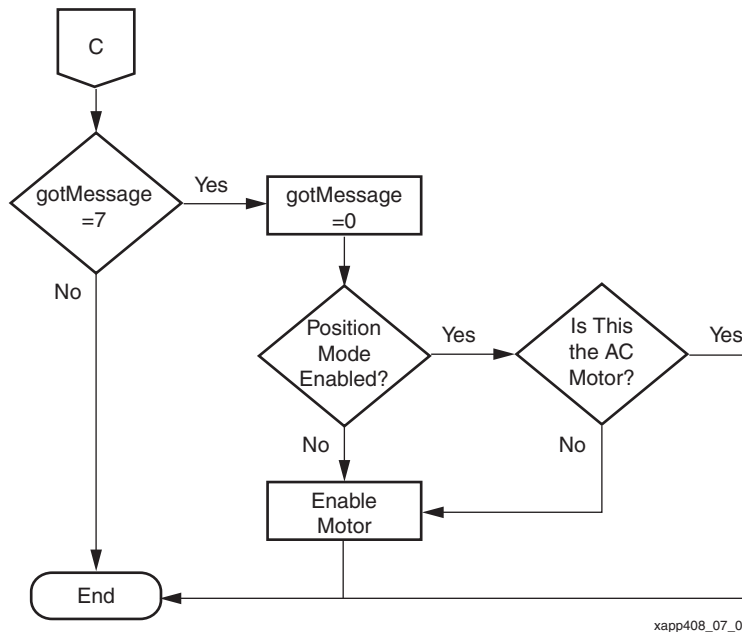
Figure 9: **Message Processing Flow Chart (Part 3)**

Communication

The following paragraphs describe the protocol used to communicate between the motor controllers and the host computer. Table 2 shows the configuration of the RS232 line characteristics.

Table 2: **UART Line Characteristics**

Parameter	Value
Baud	115200bps
Parity	None
Data Bits	8
Stop Bits	1
Flow Control	None

Communication Protocol

The communication protocol used is a simple ASCII based protocol which communicates over a CAN bus. This protocol is used by the system in both board-to-board, and also board-to-computer (PC host).

Because this protocol is used over a CAN bus the amount of data that a single message can contain is limited to eight bytes. See Figure 10 for the exact format of the communication packets.
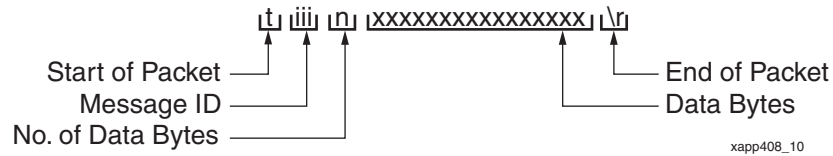
Figure 10: **Communication Protocol Format**

Each message begins with an ASCII 't' followed by 3 ASCII hex digits that represent the message ID. Next, a single ASCII digit in the range of '0' – '8' that represents the number of data bytes to follow, note that this is the number of BYTES, not the number of ASCII characters in the data; each BYTE requires two ASCII digits. Up to 16 ASCII hex digits follow representing the message data followed by the end of message delimiter '\r' (0x0a). A typical message is represented below.

```
t100812efe57300000080\r
```

PI Message

This message is sent from the controlling computer to the motor controllers and contains Proportional and Integral values used in the PID control loop. This message is formatted as follows.

```
tiii8PPPPPPPPIIIIIIII\r
```

iii – Message ID

> 100 – PI Message for Motor-A

> 200 – PI Message for Motor-B

PPPPPPPP – Proportional Gain Value

> 32 bits representing a single precision floating point value

IIIIIIII – Integral Gain Value

> 32 bits representing a single precision floating point value

When successfully received the motor controllers will respond with a PI-ACK message, formatted as follows.

```
t3020\r
```
 – Motor-A PI-ACK

```
t3040\r
```
 – Motor-B PI-ACK

DR Message

This message is sent from the controlling computer to the motor controllers and contains the Derivative value used for the PID control loop, and the speed (in RPM) of the motors. This message is formatted as follows.

```
tiii8DDDDDDDDRRRRRRRR\r
```

iii – Message ID

> 101 – DR Message for Motor-A

> 201 – DR Message for Motor-B

DDDDDDDD – Derivative Gain Value

> 32 bits representing a single precision floating point value

RRRRRRRR – Motor Speed in RPM

Product Not Recommended for New Designs

32-bit unsigned value representing the speed of the motors in RPM

When successfully received, the motor controllers will respond with a DR-ACK message, formatted as follows.

t3030\r – Motor-A DR-ACK

t3050\r – Motor-B DR-ACK

S Message

This message is sent from the controlling computer to the motor controllers and contains the time in ms used to sample the motor speed. This message is formatted as follows.

tiii4ssssssss\r

iii – Message ID

102 – S Message for Motor-A

202 – S Message for Motor-B

ssssssss – Motor Speed Sample Time (ms)

32-bit unsigned value representing the motor speed sample time in ms

When successfully received, the motor controllers will respond with a S-ACK message, formatted as follows.

t3060\r – Motor-A S-ACK

t3070\r – Motor-B S-ACK

Start Message

There is no specific start message; the motors are started by sending a "PI Message" followed by a "DR Message", followed by an "S Message".

Stop Message

This message is send from the controlling computer to the motor controllers to stop the motors. This message is formatted as follows.

tiii0\r

iii – Message ID

104 – Stop Message for Motor-A

204 – Stop Message for Motor-B

When successfully received, the motor controllers will respond with a Stop-ACK message, formatted as follows.

t3090\r – Motor-A Stop-ACK

t30b0\r – Motor-B Stop-ACK

Speed Mode Message

This message is sent from the controlling computer to the motor controllers to switch their operational mode to maintaining a constant speed. The message is formatted as follows.

tiii0\r

iii – Message ID

105 – Speed Mode Message for Motor-A

205 – Speed Mode Message for Motor-B

When successfully received, the motor controllers will respond with a Speed-ACK message, formatted as follows.

`t30c0\r` – Motor-A Stop-ACK

`t30d0\r` – Motor-B Stop-ACK

Position Mode Message

This message is sent from the controlling computer to the motor controllers to switch their operational mode to maintaining a specific position. The message is formatted as follows.

`tiii0\r`

`iii` – Message ID

`106` – Speed Mode Message for Motor-A

`206` – Speed Mode Message for Motor-B

When successfully received, the motor controllers will respond with a Position-ACK message, formatted as follows.

`t30e0\r` – Motor-A Stop-ACK

`t30f0\r` – Motor-B Stop-ACK

Physical Connections

The following sections describe the physical connections required by this reference design.

RS232 Communications

A NULL modem cable should be connected between the UART Host port (P3) on the development board configured to be the AC induction motor controller and the host computer. The AC induction motor controller board acts as a CAN/RS232 converter, receiving messages from the UART and transmitting them on the CAN bus and receiving messages from the CAN bus and transmitting them via the UART. The development board configured to be the BLDC motor controller does not perform this operation.

CAN Bus

Create a CAN bus between the two SP305 embedded development boards by connecting J15 from one board to J15 on the other board using twisted pair cabling. Ensure the jumpers listed in Table 3 are installed to enable the CAN bus termination, external CAN MAC and CAN PHY.

Table 3: **CAN Bus Jumper Settings**

Jumper	Description	Connection
J23	120Ω CAN bus termination	1-2 (enabled)
J16	CAN Phy enable	1-2 (enabled)
J27	CAN MAC Clock select	2-3 (external 16MHz oscillator)
J35	CAN_TXCAN_MAC	1-2 (connect to CAN_TXCAN)
J36	CAN_RXCAN_MAC	1-2 (connect to CAN_RXCAN)

Isolation Cards

Connect the two female 96-pin connectors on the isolation cards to the two male 96-pin connectors on the SP305 embedded development boards so that the isolation card lays across the SP305 (see Figure 11).
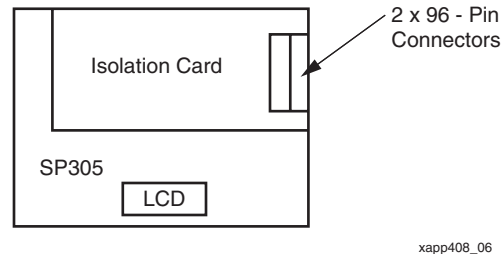
Figure 11: **Isolation Card Connection**

The isolation cards are powered from the SP305 and the motor power stages; no additional power source is required and no jumpers need to be fitted to the isolation cards.

Power Stages

For the BLDC motor controller, connect J1 on the BLDC power stage to H3 on the isolation card using a 40-pin ribbon cable. For the AC induction motor controller, connect J3 on the AC induction power stage to H4 on the isolation card using a 24-pin ribbon cable. It is vital when connecting the AC induction power stage that pin 1 on the isolation card is connected to pin 1 on the power stage, the power stage connector is a single row of 12 pins whilst the connector on the isolation card is two rows of 12 pins.

Shaft Encoders

Table 4 should be referenced when connecting both the BLDC and AC induction motor shaft encoders to their respective isolation cards.

Table 4: **Shaft Encoder Connections**

Isolation Card Connector		Shaft Encoder Pin Number	Signal Name
AC Induction (H5) Pin Number	BLDC (H9) Pin Number		
1	1	4	+5V
2	2	1	GND
3	3	3	Ch A
4	4	5	Ch B
5	5	2	Ch I

BLDC Hall Effect Sensors

Table 5 should be referenced when connecting the Hall effect sensors of the BLDC motor to the isolation card.

Table 5: **Hall Sensor Connections**

Isolation Card (H6) Pin #	Motor Signal
1	+5V
2	GND
3	Hall A
4	Hall B
5	Hall C

Motor Power

Connect phase A, B, and C of the BLDC motor to the appropriate phase connections (J2) on BLDC power stage, and likewise with J4 on the AC induction motor power stage for the AC induction motor, to reduce the risk of electric shock, the case of the AC induction motor should be connected to an earth ground. For more information with regard to the motor power stage connections refer to the power stage documentation located in the *Datasheets* directory of the ZIP file for this reference design.

DIP Switches

Two switches on SW1 are used configure the motor control software running on the MicroBlaze processor. The DIP switch settings are described in Table 6.

Table 6: **DIP Switch Settings**

DIP Switch	State	Description
SW1-1	OFF	Represents *Motor A* on the Motor Controller GUI
	ON	Represents *Motor B* on the Motor Controller GUI
SW1-2	OFF	Controller controlling AC induction motor
	ON	Controller controlling BLDC motor

Host Based GUI

Source code for the Windows based GUI used to control and monitor the motion of the motors is located under the *Visual Studio Projects/Motor Controller* directory of the ZIP file for this reference design. A pre-build version is located in the *Bin* directory of the reference design ZIP file.

To change the serial communications port used by the application select *Communication -> Serial Setup…*, the *Serial Setup* dialog will be displayed, enter the name of the communications port that the application should use in the format *comX*, where X is the number of the communication port to be used. After changing the application's communication port the application should be re-started.

The reference system has three main modes of operation selectable from the *Mode* pull-down menu on the GUI; these modes are described below.

Speed

When the *Speed* option is selected from the *Mode* pull-down menu within the GUI the motor control parameters are used to maintain the speed of the motors. This is the default operation mode when the Motor Controller application is started.

After setting appropriate PID control loop, speed, and sample time values, click the **Start** button to start the motors; to stop the motors, click the **Stop** button. The **Reset** button should be used to reset the motor's parameters to the application starting values, these values are saved when the application is terminated and re-loaded when the application is started. The **Set** button should be used to change the motor control parameters without having to stop and re-start the motor.

Position

When the *Position* option is selected from the *Mode* pull-down menu the AC induction motor does not turn. The optical shaft encoder attached to the AC induction motor is used to send position information to the BLDC motor controller which in turn uses the PID control loop parameters to try to keep the position of the BLDC in sync with the position of the AC induction motor.

The BLDC motor will not begin to track the movements of the AC induction motor until the **Start** button for the BLDC motor is clicked.

Auto

Auto mode is used to demonstrate the design without user intervention. After random periods of time, random speed information is sent to each motor.

Conclusion

This document has detailed the design of the FPGA Motor Control Reference Design. Though this design has been extensively verified in simulations, Xilinx assumes no responsibility for the accuracy or the functionality of this design.

Revision History

The following table shows the revision history for this document.

Date	Version	Revision
9/16/05	1.0	Initial Xilinx release.