Application Note

Vector Control of a 3-Phase AC Induction Motor Using FMC16100 MCU

AN024702-1006
This publication is subject to replacement by a later edition. To determine whether a later edition exists, or to request copies of publications, contact:

**ZiLOG Worldwide Headquarters**
532 Race Street
San Jose, CA 95126
Telephone: 408.558.8500
Fax: 408.558.8300
www.zilog.com

ZiLOG is a registered trademark of ZiLOG Inc. in the United States and in other countries. All other products and/or service names mentioned herein may be trademarks of the companies with which they are associated.

**Information Integrity**
The information contained within this document has been verified according to the general principles of electrical and mechanical engineering. Any applicable source code illustrated in the document was either written by an authorized ZiLOG employee or licensed consultant. Permission to use these codes in any form, besides the intended application, must be approved through a license agreement between both parties. ZiLOG will not be responsible for any code(s) used beyond the intended application. Contact the local ZiLOG Sales Office to obtain necessary license agreements.

**Document Disclaimer**
©2006 by ZiLOG, Inc. All rights reserved. Information in this publication concerning the devices, applications, or technology described is intended to suggest possible uses and may be superseded. ZiLOG, INC. DOES NOT ASSUME LIABILITY FOR OR PROVIDE A REPRESENTATION OF ACCURACY OF THE INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED IN THIS DOCUMENT. ZiLOG ALSO DOES NOT ASSUME LIABILITY FOR INTELLECTUAL PROPERTY INFRINGEMENT RELATED IN ANY MANNER TO USE OF INFORMATION, DEVICES, OR TECHNOLOGY DESCRIBED HEREIN OR OTHERWISE. Except with the express written approval ZiLOG, use of information, devices, or technology as critical components of life support systems is not authorized. No licenses or other rights are conveyed, implicitly or otherwise, by this document under any intellectual property rights.

**Version Disclaimer**
Application Notes are tested with the version of ZDS II available at the time of release. Subsequent releases of ZDS II may require you to modify Application Note code to restore its function.
Revision History

Each instance in the Revision History reflects a change to this document from its previous revision. For more details, refer to the corresponding pages or appropriate links given in the table below.

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision Level</th>
<th>Section\Description</th>
<th>Page No</th>
</tr>
</thead>
<tbody>
<tr>
<td>October 2006</td>
<td>02</td>
<td>Table 6, replaced eZ80® CPU with Z8 Encore!® and eZ80® CPU User Manual (UM0077) with eZ8 CPU User Manual (UM0128).</td>
<td>29</td>
</tr>
<tr>
<td>July 2006</td>
<td>01</td>
<td>All</td>
<td>All</td>
</tr>
</tbody>
</table>
# Table of Contents

**List of Figures** ................................................................. v  
**List of Tables** ................................................................. vi  
**Abstract** ............................................................................. 1  
  Z8 Encore® MC Flash Microcontrollers ........................................ 1  
**Discussion** ........................................................................... 1  
  Theory of Operation ............................................................... 2  
  Description of Components .................................................... 3  
**Application Hardware** ........................................................... 5  
**Software Implementation** ....................................................... 11  
  Variables and Declarations ..................................................... 11  
  Initialization .......................................................................... 12  
  Main Event Loop ................................................................... 14  
  Subfunctions ......................................................................... 24  
  Software Metrics .................................................................... 25  
**Summary** ............................................................................. 28  
**Appendix A—References** ......................................................... 29  
**Appendix B—Glossary** ............................................................ 30  
**Appendix C—Schematic Diagrams** .......................................... 31  
**Appendix D—Flowcharts** ......................................................... 38
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Washing Machine Application</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Washing Machine Motor with Tachogenerator</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Modular Controller and XP Command Module</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Hardware Block Diagram</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>Current Sense Circuit</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Overcurrent Sense Circuit</td>
<td>9</td>
</tr>
<tr>
<td>7</td>
<td>Functional Block Diagram</td>
<td>11</td>
</tr>
<tr>
<td>8</td>
<td>Sampling Space Vector PWM Current (State = 3)</td>
<td>15</td>
</tr>
<tr>
<td>9</td>
<td>Space Vector PWM States</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>Phase Current Reconstruction</td>
<td>17</td>
</tr>
<tr>
<td>11</td>
<td>Park Transform</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>Inverse Park Transform</td>
<td>21</td>
</tr>
<tr>
<td>13</td>
<td>Inverse Clarke Transform</td>
<td>22</td>
</tr>
<tr>
<td>14</td>
<td>PWM Space Vector Modulation (Ground Reference)</td>
<td>23</td>
</tr>
<tr>
<td>15</td>
<td>PWM Space Vector Modulation (Neutral Reference)</td>
<td>24</td>
</tr>
<tr>
<td>16</td>
<td>High-Voltage Main Board Schematic</td>
<td>31</td>
</tr>
<tr>
<td>17</td>
<td>FMC16 Control Module Schematic</td>
<td>32</td>
</tr>
<tr>
<td>18</td>
<td>HV Gate Drive Module Schematic</td>
<td>33</td>
</tr>
<tr>
<td>19</td>
<td>HV Power Module Schematic</td>
<td>34</td>
</tr>
<tr>
<td>20</td>
<td>Washing Machine Interface Module Schematic</td>
<td>35</td>
</tr>
<tr>
<td>21</td>
<td>DAC Module Schematic</td>
<td>36</td>
</tr>
<tr>
<td>22</td>
<td>XP Command Module Schematic (Connects to Washing</td>
<td>37</td>
</tr>
<tr>
<td></td>
<td>Machine Module)</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Flowchart for the Application</td>
<td>38</td>
</tr>
<tr>
<td>24</td>
<td>Flowchart for the PWM ISR</td>
<td>39</td>
</tr>
</tbody>
</table>
List of Tables

Table 1. Sample Interpretation by PWM State ........................................ 16
Table 2. PWM Duty Cycle by State .......................................................... 22
Table 3. Initialization Execution Times ..................................................... 25
Table 4. Main Loop (ISR) Execution Times .............................................. 26
Table 5. Subfunction Execution Times ....................................................... 27
Table 6. List of References ...................................................................... 29
Table 7. Glossary ..................................................................................... 30
Abstract

The 3-phase alternating current (AC) induction motors are mechanically simple, rugged, highly reliable, lower in cost per horsepower than DC motors, and capable of more torque and efficiency than single-phase AC motors. A 3-phase AC induction motor can be controlled by varying its inputs according to a mathematical model of the rotor flux field in a complex vector space (Vector Control). Vector Control provides efficient and accurate control of the motor's speed and torque.

Until now, Vector Control has been the domain of digital signal processors (DSPs), digital signal controllers (DSCs), and a few 32-bit and 16-bit microcontrollers. Constant cost pressure and increased consumer expectations have driven design engineers to seek minimal hardware solutions that extract maximum performance from motors used in consumer goods.

This application note demonstrates how ZiLOG's Z8 Encore!® MC 8-bit microcontroller is used to implement Vector Control of an AC induction motor.

Z8 Encore!® MC Flash Microcontrollers

ZiLOG's Z8FMC16100 Series Flash MCU is based on ZiLOG's advanced eZ8 8-bit CPU core and is optimized for motor control applications. It supports control of single and multiphase variable speed motors. Target applications are consumer appliances, HVAC, factory automation, refrigeration, and automotive applications, among others.

To rotate a 3-phase motor three AC voltage signals must be supplied, phase-shifted 120° from each other. To control a 3-phase motor the MCU must provide 6 Pulse Width Modulation (PWM) outputs. The Z8FMC16100 Series Flash MCU features a flexible PWM module with three complementary pairs or six independent PWM outputs supporting deadband operation and fault protection trip input. These features provide multiphase control capability for various motor types and ensure safe operation of the motor by providing immediate shutdown of the PWM pins during a fault condition.

Discussion

An electric motor consists of a stationary frame (stator) in which a rotating component (rotor) is mounted on a shaft and bearings. In a 3-phase AC induction motor, the stator is laced with three sets of inductor windings energized by three AC voltage inputs that are phase-offset 120° from each other, producing a rotating field of magnetic flux inside the motor. This field creates an induced magnetic flux in short passive windings on the rotor. The stator field exerts magnetic force on the rotor flux field, resulting in torque on the output shaft.
In a 3-phase motor control application, the input to the motor is produced by a 3-phase inverter bridge. A bridge contains three complementary source/drain transistor pairs which connect either ground or high-voltage DC to each of its three outputs in response to digital control signals from the microcontroller. The microcontroller uses PWM on the bridge control signals to generate three approximately sinusoidal AC waveforms on the bridge outputs, with the required 120° phase offset.

The duty cycle of each microcontroller PWM output is varied to control the period and amplitude of the generated AC signal, which in turn determines the speed and torque of the motor. The microcontroller senses the resulting AC current and motor speed and periodically adjusts the AC signal waveform, increasing or decreasing torque to maintain the appropriate motor speed. For more information on motor control, see documents listed in Appendix A—References on page 29.

**Theory of Operation**

In this application, the FMC16100 microcontroller's PWM module is configured as three complementary output pairs. Each output pair controls one complementary source/drain transistor pair in the inverter bridge. The PWM module is configured to insert a 0.6 µS deadband between ON states to prevent leakage that might occur if one transistor in a pair turns on before the other is fully off.

Each PWM output pair produces a stream of complementary on/off pulses to activate the corresponding source or drain transistor in the inverter bridge. The voltage of each bridge output varies with the source/drain pulse duty cycle.

The period of each PWM cycle is configured to be 50 µS and the PWM module generates an interrupt request at the end of each cycle. The PWM interrupt service routine (ISR) leaves the PWM interrupt disabled while it executes, and is tuned to execute within 200 µS to 250 µS. Therefore, the PWM ISR is executed every five PWM cycles or 250 µS.

The PWM ISR controls all of the ongoing application program functions after initialization. The primary goal of the ISR is to update the duty cycle value for each PWM channel as needed to produce the appropriate AC waveforms at the inverter bridge outputs.

The PWM duty cycle values are derived from a feedback loop based on rotor speed and rotor flux position. The stator current waveform is determined by sampling two phases of the inverter output current and reconstructing the third phase.

The rotor speed is sampled by a counter/timer configured to measure the period of a magnetic position sensor on the motor. The measured speed is periodically compared to the requested speed received through UART from an external controller. The resulting speed command value is used to create a rotation reference frame expressed as a two-phase direct/quadrature (DQ) vector.
The resulting vector is regulated to produce the necessary flux and torque amplitude in the output vector.

An inverse Park transform is used to rotate the reference frame relative to the rotor position. Then an inverse Clarke transform is used to convert the rotated two-phase reference frame back into a 3-phase expression. At the same time, the vector’s position is classified into one of six states, each corresponding to one sixth of the vector circle.

When the ISR calculates duty cycles for the three PWM channels, one phase is left unmodulated (ON or OFF for the whole PWM cycle), and the flux vector is encoded by modulating the other two phases. This reduces switching losses by one third. The two phases to be modulated depend on the flux vector’s state position. After the PWM space vector calculation is complete, the corresponding PWM registers are updated. For more details on the software implementation, see Software Implementation on page 11.

Description of Components

In addition to the Z8 Encore!® MC microcontroller, this design uses the following major components:

- Z8 Encore! XP® microcontroller
- Insulated-gate bipolar transistors (IGBTs)
- I2C interfaced 8-bit digital-to-analog converter (DAC)
- Washing Machine Motor

Z8 Encore! XP® Microcontroller

This application employs the Z8 Encore! XP® 8-pin MCUs internal precision oscillator, 10-bit analog-to-digital converter (ADC), I/O, and UART features to demonstrate the washing machine application user interface. User commands are transmitted to the Z8 Encore!® MC MPU as serial data. In theory, this allows very complex control commands to be transmitted. The demonstrated user interface consists of a potentiometer for speed control, a pushbutton command input, and a pushbutton to reset the Z8 Encore! XP® MCU.

Two 6N137 single-channel optocouplers transmit UART command signals while electrically isolating the user command module from the high-voltage modules.

Insulated-Gate Bipolar Transistors

Six 6SG5N60RUFD IGBTs are used under PWM control to generate the required 3-phase AC motor drive current from a high-voltage DC supply. Although these IGBTs are rated for momentary short-circuit operation, the Z8 Encore!® MC PWM deadband feature is used to reduce leakage when the IGBTs are switched.
Three IR21064S high-side and low-side gate drivers are used to convert the Z8 Encore!® MC microcontroller's CMOS PWM outputs into signals with the voltage needed to drive the IGBT gate inputs.

**Digital-to-Analog Converter**

The Z8 Encore!® MC microcontroller’s I²C interface is used with a DAC5574 digital-to-analog-converter to generate up to 4 analog test outputs based on internal program values. This allows the internal values to be directly compared to the motor controller’s output waveform on an oscilloscope.

**Motor**

The motor is a 3-phase AC Induction motor used in washing machine applications as illustrated in Figure 1.

The application motor is illustrated in Figure 2 on page 5. The motor has the following specifications:

- Rated Power: 500 W
- RPM max: 15,000 rpm
- Max current: 2.8 Arms
- Continuous current: 1.5 Arms
- DC Bus Voltage: 350 Vdc

The motor windings are connected in the Wye configuration for this application. The back side of the motor has an integrated Tachogenerator, which consists of an 16-pole magnet mounted on the shaft and a single-phase winding mounted on the housing. This produces a sine wave signal whose voltage and frequency are proportional to the motor speed. The interface to the Z8FMC16100 is a simple transistor circuit that squares off the sine wave into 3.3 V digital pulses going to Timer 0 in Capture Restart mode. This gives 8 pulses per revolution.
Application Hardware

The application electronic components and circuitry are divided among several printed circuit modules that attach to a main board mounted on a heat sink and a separate user command module, as illustrated in Figure 3.
Figure 4 is a block diagram of the application hardware design, showing the logical position of each module in the controller.

The following sections briefly describe each module of the application hardware. The schematics are shown in Appendix C—Schematic Diagrams on page 31.

**High-Voltage Main Board and Power Stage**

The main board serves as the connector backplane for all the other controller modules. It also provides the high-voltage power stage that converts the single-phase AC supply to the 3-phase motor control output under PWM control. The schematic for the main board module is illustrated in Figure 16 on page 31.
The power stage section rectifies and filters the single-phase AC supply to provide a high-voltage DC bus supply. This section also contains the six IGBTs used under PWM control to generate the 3-phase AC motor drive current.

The MPU is powered after the AC supply is connected, but a normally-open relay withholds DC power from the IGBTs until the MPU has started and the application software has completed its necessary startup functions. The application then uses an I/O port output to enable a simple transistor driver that closes the relay, powering the IGBTs.

**High-Voltage Power Supply Module**

The schematic for the power supply module is shown in Figure 19 on page 34. This module uses an LNK304G IC to convert rectified AC from the power stage to generate the application’s 120 mA, 12 V supply. This 12 V supply is then regulated by a UA78M33 IC to provide the 3.3 V supply.

**High-Voltage Gate Drive Module**

The schematic for the gate drive module is shown in Figure 18 on page 33. This module contains three IR21064S high-side and low-side gate drivers that convert the Z8 Encore!® MC microcontroller PWM outputs into the voltage needed to drive the IGBT gate inputs.

**FMC16100 MPU Control Module**

The schematic for the MPU module is shown in Figure 17 on page 32. This module contains the Z8 Encore!® MC microcontroller, a 20 MHz ceramic resonator, ZiLOG 6-pin DBG interface, and the off-chip components of the high-voltage current sense and overcurrent detection circuits.
Figure 5 illustrates the overall current sense circuit, including some power stage components and components internal to the MPU. Vref is the internal reference voltage of the MPU analog-to-digital converter and the full scale input range. At zero current (zero voltage across Rsense) the output of the Op Amp is at 1/2Vref, this centers the current waveform at one half of the full input range of the ADC.

The Resistive Divider also forms a Low Pass Filter (LPF) to take out the diode reverse recovery current spike in the current sense signal. Since this is usually in the MHz frequency range a single pole LPF is sufficient set at a couple of hundred kHz. As the Operational Amplifier circuit already has a LPF this is more important for the following comparator circuit.
The Cutoff Frequency is calculated as follows:

$$f_c = \frac{1}{2\pi \times R1 \parallel R2 \times C2} = \frac{1}{2\pi \times 5 \text{ k}\Omega \times 120 \text{ pF}} = 256 \text{ kHz}$$

Figure 6 illustrates the overall schematic for the overcurrent sense circuit, likewise including some power stage components and components internal to the MPU.

R3 and R4 set the overcurrent trip threshold. Since this is a DC value, a bypass capacitor (C3) of 0.1 μF is used.

For R4 = 10, R3 is selected as:

$$R3 = R4 \times \frac{(V_{\text{ref}} + I_{\text{max}} \times R_{\text{sense}})}{(V_{\text{ref}} - I_{\text{max}} \times R_{\text{sense}})} = 10 \text{ k}\Omega \times \frac{(2V + 10\text{A} \times 0.02 \text{ \Omega})}{(2V - 10\text{A} \times 0.02 \text{ \Omega})} \approx 12.4 \text{ k}\Omega$$
Washing Machine Interface Module

This module provides the speed control interface between the modular controller and the washing machine. It consists of the following parts:

- Optoisolated UART command interface
- Motor tachometer/turn sensor

The optoisolator section ensures the electrical isolation of the user command module from the high-voltage module, and also serves to multiplex the FMC16100 RXD and TXD signals onto a single isolated DATA signal that connects to the remote Z8 Encore! XP® user command module. Power and ground for the user side of the optoisolator section are provided by the user command module.

The tachometer/turn sensor section converts the analog rotor-sense signal from the AC motor to a logic-level DC pulse stream suitable for the Z8 Encore!® MC timer/counter input.

DAC Test Output Module

This module contains a DAC5574IDGS DAC whose I²C serial interface is driven by the Z8 Encore!® MC microcontroller. The controller software transmits selected internal values to the DAC, which converts those digital values to one of four analog outputs. This allows, for example, the internal reconstructed waveform to be compared on an oscilloscope to the actual AC output from the power stage.

Z8 Encore! XP® Command Module (External)

The Z8 Encore! XP® user command module is separate and remote from the modular controller. It requires a separate 12 V power supply, which it regulates to 5 V and 3.3 V for its own use. The module’s clock source is the Z8 Encore! XP® MPU’s internal precision oscillator. The module includes a 6-pin DBG header, a reset pushbutton switch, an input pushbutton, and an input potentiometer for speed control. Command values derived from the input are transmitted to the controller main board through a single data line connected to both the TXD and RXD pin of the Z8 Encore! XP® UART. The main board connector also supplies 5 V and GND signals to power the user side of the optoisolator section of the washing machine interface module.
Software Implementation

Figure 7 illustrates the functional block diagram of the Vector Control application.

Variables and Declarations

Declarations set the PWM frequency, the Timer 0 scaling for the Tachogenerator, and the modulation values. All variables are declared as global for speed of execution and simplicity of code.

All variables are unsigned. All signed values have separate variables of the form Name for positive and nName for negative contents. One of these variables is always zero. Using unsigned math speeds up execution but makes the algorithms longer in code.

The ADC inputs are treated as 8 bits and the PWM output is 8 bits. Internal computations generally have 4 extra bits of precision especially in the control loops that use PI (Proportional Integral) compensators, this prevents rounding from affecting integrator precision. For Inverse Park Transform (vector rotation) the concept of a quadrant is used to determine calculations on the unsigned values.
The program uses a 128-value sine lookup table, \texttt{Sine\_Table[128]}. This table represents sine values from 0 to 90 Degrees, or approximately 0.7 Degree steps. Stepping through the table in the forward direction gives sine values from 0 to 90 Degrees and stepping through the table in the reverse direction gives cosine values from 0 to 90 Degrees. This along with knowledge of the quadrant of the angle of interest (Theta) allows full reconstruction of sine and cosine values for any angle between 0 and 360 Degrees.

The polarity table, \texttt{Polarity\_table[8]}, contains bit masks used to set the polarity bits in the PWMCTL1 register. A 1 bit causes the corresponding channel to start the next PWM cycle in the ON state. A 0 bit causes the channel to start in the OFF state.

The ripple compensation table, \texttt{ripple\_table[256]}, contains pre-calculated bus voltage ripple compensation factors. See Bus Ripple Compensation on page 20.

**Initialization**

The following initializations are performed when \texttt{main()} is executed:

**Option Bits**

The options bits set the PWM Polarity and the WDT default mode.

**Oscillator**

Oscillator initialization consists of sending an unlock sequence to the oscillator control register then enabling the external oscillator circuit. After allowing a setting time of 50 ms the oscillator control register is unlocked again 20 MHz External Ceramic Resonator

**Digital Output**

PB7 is used as a digital test pin for code development. This allows testing of the speed of routines and to check out conditional branching of routines.

**Comparator**

Used for Overcurrent shutdown.

**Op Amp**

For current sensing using a single current sense resistor. The op Amp is used to offset the ground referenced current sense signal to 1 V (half of the 2 V reference) and the voltage is gained up by a factor of 10. For a sense resistor value of 20 mOhms and a current of +/-4A gained up by 10 this represents a 1 V signal that can swing +/- 0.8 V.
ADC
Used for sampling current and DC Bus Voltage.

Relay
The main relay is normally open so the high-voltage power supply is disabled while the microcontroller starts up. The relay initialization pauses for three seconds before closing the relay to make sure the high-voltage bus capacitors can precharge through a resistor. After closing the relay, the program waits another second to allow the relay contacts to settle.

Current Calibration
Once the peripherals are enabled and have settled out 8 samples of the current sense channel are taken with zero current. The average of these samples are used to zero out any offsets in the circuit.

UART
The UART is configured for 9600 baud, 1 start bit, 8 data bits, and 1 stop bit. UART transmit and receive are tied together for single wire communication. The 3 wires are ground, data, and 5 V. The UART service routine updates every 2 ms. The service routine is set up for just receiving data in this application. A more elaborate communication link could be implemented with this hardware.

I2C
I2C is used for an external EEPROM for logging product data. For testing purposes the I2C interface is used to send data to a 4 Channel DAC. In this implementation a single channel of the DAC is updated every 250 μs so that one parameter can be monitored in real time. This provides a method for troubleshooting and code development, and field orientation tuning using an oscilloscope. To control the DAC the Z8FMC16100 is set up as the I2C Master at 384 kbps, 2.6 μs per bit. All data is sent to DAC channel A.

Timer0
Timer 0 is used in Capture Restart Mode, the encoder input rising edge is used to trigger the Capture. If the timer rolls over then the maximum timer count of 0xFFFF is used setting it to the minimum speed.

PWM Module
The 3-phase PWM module is set up in Edge Aligned PWM and the PWM polarity is controlled to Allow Space Vector Modulation and current sampling and reconstruction. Each PWM pair is set up in complementary mode with 12 clock
cycles of deadband (0.6 μs). The PWM reload event triggers an ADC conversion. This is used for current reconstruction.

**PWM Interrupt**

The PWM reload interrupt is enabled. The PWM ISR constitutes the program control loop. PWM interrupts are disabled during the ISR until the ISR is complete.

**Main Event Loop**

After initializations are complete, program control drops into an infinite ‘do nothing’ loop. All subsequent program functions are performed by the PWM interrupt service routine (ISR), so this ISR is in effect the main event loop. The PWM ISR is executed on every fifth PWM cycle.

**Current Sample**

The PWM interrupt is configured to automatically start an ADC conversion that samples the high-voltage current, so the PWM ISR begins with an ADC conversion already in progress. The PWM interrupt is edge-aligned, and the sample takes place before the PWM state change propagates to the inverter bridge gates. Thus, this first sample captures the current at the end of the just-completed PWM cycle.

After reading the first sample, the ISR immediately begins another ADC conversion. This second sample captures the current at the beginning of the new PWM cycle. This is illustrated in Figure 8 on page 15. Phase currents are then reconstructed from these samples.
**I²C Send Address and Start Bit**

Sends the address byte used by the DAC to establish communication. This action begins a side-process of the PWM ISR. I²C-related instructions are inserted at intervals to set up and transmit one test value per ISR loop to an external DAC.

**Phase Current Reconstruction**

Phase current reconstruction is based on the two current samples previously obtained plus the PWM space vector State. The State value is updated at the end of each ISR pass to reflect the stator flux vector angle arrived at in that pass. State positions are shown in Figure 9 on page 16. The State determines how each current sample is interpreted, as shown in Table 1 on page 16. In this table, the variables $I_a, nI_a, I_b, nI_b, I_c$, and $nI_c$ represent the positive and negative current variables for phases A, B, and C.

![Figure 8. Sampling Space Vector PWM Current (State = 3)](image-url)
Vector Control of a 3-Phase AC Induction Motor Using FMC16100 MCU

Using Kirchoff’s current Law and assuming no zero sequence currents (zero sequence currents are only possible in an imbalanced Delta wound motor not in a

Table 1. Sample Interpretation by PWM State

<table>
<thead>
<tr>
<th>PWM State</th>
<th>Current Sample 1 Variable</th>
<th>Current Sample 2 Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>nI_c</td>
<td>nI_b</td>
</tr>
<tr>
<td>1</td>
<td>I_b</td>
<td>I_a</td>
</tr>
<tr>
<td>2</td>
<td>nI_a</td>
<td>nI_c</td>
</tr>
<tr>
<td>3</td>
<td>I_c</td>
<td>I_b</td>
</tr>
<tr>
<td>4</td>
<td>nI_b</td>
<td>nI_a</td>
</tr>
<tr>
<td>5</td>
<td>I_a</td>
<td>I_c</td>
</tr>
<tr>
<td>6</td>
<td>(Not Used)</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>(Not Used)</td>
<td></td>
</tr>
</tbody>
</table>

Figure 9. Space Vector PWM States
Wye winding), the sum of three phase currents equals zero. The third current can be derived from the other two samples using the following equation:

\[ I_a + I_b + I_c = 0 \]

The oscillograph in Figure 10 compares a reconstructed waveform as output to the DAC at 1 A/330 mV (top trace) to the original waveform as read by a current probe at 1 A/10 mV.

---

**Test for End of PWM Period**

This action begins another side-process of the PWM ISR. PWM interrupts remain disabled until the main loop ISR returns. However, at several times during the main loop, the program polls the IRQ0 register to see if a PWM request is pending. If so, the program clears the request and increments a counter. Near the end of the main loop, this event counter is used to delay completion of the process until four PWM request events have occurred, in addition to the one that began the loop. This process ensures a five-event main loop cycle (250 μs total) without the need for precise timing of each main loop routine.
Clarke Transform

A Clarke transform produces a 2-phase (direct and quadrature) representation of the reconstructed 3-phase stator flux vector. Then a quadrant test is applied to the reference currents $I_{ds}$ and $I_{qs}$ to determine which quadrant the currents are in. This result is used later in the Park Transform to rotate the 2-phase vector. The equation for the Clarke Transform is:

$$I_{ds} = I_a - \frac{1}{2} (I_b + I_c)$$

$$I_{qs} = \frac{2}{\sqrt{3}} (I_b + I_c)$$

Angle Update

The angle Theta is updated through the following equation:

$$Theta = Theta + Speed + Slip$$

Where:

$Theta$ = rotor flux angle relative to the stator phase A winding

$Speed$ = speed (frequency) of the rotor in terms of the update rate of the control loop

$Slip$ = slip speed (frequency) required to produce the proper rotor flux

Sine and Cosine Lookup

Sine and Cosine values are generated from an unsigned table of sine values from 0 to 90 degrees contained in a 128 value table ($90 \text{ Degrees/128steps} = 0.7$ Degrees/step) Two values are taken from the table one at angle stored in the variable Sin_1 and one at 128-angle stored in the variable Sin_2.

I2C Send Control Byte to DAC

Sends the control byte used by the DAC to select the DAC output channel.

Park Transform—Vector Rotation

The Park transform rotates the two phase currents referred to the stationary reference frame of the stator to the rotating reference frame of the rotor flux. This is illustrated by Figure 11 on page 19. The equation for the Park Transform is:

$$I_{dr} = I_{ds} \times \cos(Theta) + I_{qs} \times \sin(Theta)$$

$$I_{qr} = -I_{ds} \times \sin(Theta) + I_{qs} \times \cos(Theta)$$
I\textsuperscript{2}C Send MSB to DAC

Sends the 8 bit data used by the DAC. This is the internal value that is sent to the DAC board to allow display of internal values as voltages on the oscilloscope. The implemented routine sends data to one channel of the DAC updated at the loop rate. This allows real time display of reference or feedback values.

I\_d Regulator

Regulator that controls the direct component of the current vector. This is regulated by a simple PI regulator.

I\_q Regulator

Regulator that controls the quadrature component of the current vector. This is regulated by a simple PI regulator.
Quadrant Test
A quadrant test is applied to the reference currents $I_{dr}$ and $I_{qr}$ to determine which quadrant the currents are in. This result is used later in the Inverse Park Transform to rotate the 2-phase vector. Reference currents $I_{dr}$ and $I_{qr}$ are calculated in the `Speed_reg()` subroutine.

I²C Send LS Byte to DAC
The DAC is only 8-bits resolution so this is a don’t care byte required by the DAC.

Read Bus Voltage
The bus voltage is sampled through the microcontroller’s ADC. Currently this is used for ripple compensation, but it could also be included in field weakening calculations or used for over-voltage shutdown protection.

Bus Ripple Compensation
This routine tracks changes in the bus voltage and looks up a precalculated ripple compensation factor which is inversely proportional to the ADC sample of the bus voltage. The PWM duty cycle is scaled by this factor to compensate for variations in the DC bus voltage, so that a smaller factor value corresponds to a reduced PWM duty cycle. This clamps the output waveform to a nominal voltage.

This application uses a nominal bus voltage of 225 VDC. At this voltage, the output factor is 255 (full scale, no compensation). The factor scales down to 0 at 375 VDC or higher, providing an overvoltage shutdown. Between 225 VDC and 375 VDC, the lookup table values are calculated using the following equation:

$$
pwm\_correction = \frac{ADC\_vbus\_nom}{ADC\_vbus} \times 255
$$

Inverse Park Transform—Vector Rotation
The Inverse Park transform rotates the two phase voltages referred to the rotating reference frame of the rotor flux to the stationary reference frame of the stator. This is illustrated by Figure 12 on page 21. The equation for the Inverse Park Transform is:

$$
V_{ds} = V_{dr} \times \cos(Theta) - V_{qr} \times \sin(Theta)
$$
$$
V_{qs} = V_{dr} \times \sin(Theta) + V_{qr} \times \cos(Theta)
$$
I²C Send Stop Bit to DAC
A stop bit is sent to terminate the I²C communication.

Inverse Clarke Transform
The Inverse Clarke Transform converts the two phase voltages of the rotated DQ reference frame to the ABC reference frame of the stator windings. This is illustrated by Figure 13 on page 22. The equation for the Inverse Clarke Transform is:

\[
\begin{align*}
V_a &= V_{ds} \\
V_b &= \frac{1}{2} (V_{ds}) + \frac{2}{\sqrt{3}} (V_{qs}) \\
V_c &= \frac{1}{2} (V_{ds}) - \frac{2}{\sqrt{3}} (V_{qs})
\end{align*}
\]
PWM Space Vector Modulation

Space Vector Modulation starts with the 6 states that represent the 6 voltage vectors VA, –VC, VB, –VA, VC, and –VB. Table 2 shows the PWM duty cycle calculation used for each state. States 6 and 7 represent the unmodulated phase (OFF or ON).

**Table 2. PWM Duty Cycle by State**

<table>
<thead>
<tr>
<th>PWM State</th>
<th>Phase A</th>
<th>Phase B</th>
<th>Phase C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100%</td>
<td>100%– V_b</td>
<td>100%– V_c</td>
</tr>
<tr>
<td>1</td>
<td>V_a</td>
<td>V_b</td>
<td>0%</td>
</tr>
<tr>
<td>2</td>
<td>100%– V_a</td>
<td>100%</td>
<td>100%– V_c</td>
</tr>
</tbody>
</table>
Figure 14 illustrates an example of PWM space vector modulation at a PWM magnitude of 75 out of 125, referenced to DC Ground.

<table>
<thead>
<tr>
<th>PWM State</th>
<th>Phase A</th>
<th>Phase B</th>
<th>Phase C</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>0%</td>
<td>V_b</td>
<td>V_c</td>
</tr>
<tr>
<td>4</td>
<td>100%−V_a</td>
<td>100%−V_b</td>
<td>100%</td>
</tr>
<tr>
<td>5</td>
<td>V_a</td>
<td>0%</td>
<td>V_c</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>

Figure 14. PWM Space Vector Modulation (Ground Reference)
Figure 15 illustrates an example of PWM space vector modulation at a PWM magnitude of 75 out of 125, referenced to Neutral.

![Figure 15. PWM Space Vector Modulation (Neutral Reference)](image)

**Watchdog Timer Refresh**

Refreshes the Watchdog Timer once every loop update (250 \(\mu\)s). A reset occurs if the oscillator or a fault in the code disrupts operation for more than 200 ms.

**Subfunction Service Routine**

The subfunctions are subroutines that don’t need to be updated every time the PWM ISR executes. There are ten subroutine calls arranged in a round-robin sequence so that only one call is executed per ISR loop. Therefore, each call is executed once every 2 ms. Not all round robin slots are needed for the currently implemented subfunctions. The remaining slots are available for future expansion.

**Subfunctions**

**Tachometer Update**

The Tachometer update routine looks at the latest Capture Value from Timer 0. The routine checks for a roll-over, if it detects a roll-over then the maximum counter value of \(0xFFFF\) is used to calculate speed. Also a rate limit is put on the variable `Speed_step` to make sure noise doesn’t cause erratic speed readings. In case of an optical encoder you can have the phenomena of edge jitter, when and encoder line is on the edge of the optical pick-up and the motor is at stand still, and movement in the motor can cause fast edges that can be interpreted as
sensing high speed. This is filtered out by not allowing a jump in speed. The Tachometer update function is divided into three subfunction calls to limit the execution time for each call.

Field Weakening
Currently not developed for this application.

Slip Update (Current Model)
Currently not developed for this application

Speed Regulator
The speed regulator is a standard Proportional Integral (PI) regulator. The command comes from the UART and the speed feedback comes from the Tachometer.

UART Communication
The UART communicates with the Z8 Encore! XP® module that simulates a washing machine control panel. The control module has a potentiometer and two momentary switches. The potentiometer setting determines the transmitted speed command. Switch 1 stops/starts communication (stopping the communication stops motor).

The control module sends a single command byte that is read at each UART update. Bits 7:1 of the command byte are the speed command (MSB of the control module’s ADC sample). Bit 0 reflects the controller module Switch 2 status, which is used to control the motor direction.

Acknowledge byte from motor controller: AAh
The UART baud rate is 9600 Baud.

Software Metrics
Table 3 illustrates the execution time for each program initialization routine, executed once at startup.

<table>
<thead>
<tr>
<th>Routine</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>init_osc</td>
<td>50 ms</td>
</tr>
<tr>
<td>init_out</td>
<td>1 µs</td>
</tr>
<tr>
<td>init_comp</td>
<td>1 µs</td>
</tr>
<tr>
<td>init_amp</td>
<td>2 µs</td>
</tr>
</tbody>
</table>
Table 3. Initialization Execution Times (Continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>init_adc</td>
<td>2 µs</td>
</tr>
<tr>
<td>init_relay</td>
<td>4 s</td>
</tr>
<tr>
<td>init_current</td>
<td>32 µs</td>
</tr>
<tr>
<td>init_uart</td>
<td>2 µs</td>
</tr>
<tr>
<td>init_i2c</td>
<td>3 µs</td>
</tr>
<tr>
<td>init_timer0</td>
<td>3 µs</td>
</tr>
<tr>
<td>init_pwm</td>
<td>5 µs</td>
</tr>
<tr>
<td>init_pwmint</td>
<td>1 µs</td>
</tr>
</tbody>
</table>

Table 4 illustrates the execution time for each routine in the PWM ISR, executed every 250 µs (4 kHz loop).

Table 4. Main Loop (ISR) Execution Times

<table>
<thead>
<tr>
<th>Routine</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current Sample</td>
<td>5</td>
</tr>
<tr>
<td>I2C Send Address and Start Bit</td>
<td>1</td>
</tr>
<tr>
<td>Current Reconstruction</td>
<td>14</td>
</tr>
<tr>
<td>Clarke Transform (3=&gt;2 Conversion)</td>
<td>18</td>
</tr>
<tr>
<td>PWM Period Test</td>
<td>1</td>
</tr>
<tr>
<td>Theta Update</td>
<td>6</td>
</tr>
<tr>
<td>Sine and Cosine Lookup</td>
<td>3</td>
</tr>
<tr>
<td>I2C Send Control Byte</td>
<td>1</td>
</tr>
<tr>
<td>Park Transform (Vector Rotation)</td>
<td>28</td>
</tr>
<tr>
<td>PWM Period Test</td>
<td>1</td>
</tr>
<tr>
<td>I2C Send MSB</td>
<td>1</td>
</tr>
<tr>
<td>I_d Regulator</td>
<td>13</td>
</tr>
<tr>
<td>I_q Regulator</td>
<td>13</td>
</tr>
<tr>
<td>Quadrant Test</td>
<td>5</td>
</tr>
<tr>
<td>I2C Send LSB</td>
<td>1</td>
</tr>
<tr>
<td>Bus Voltage Sample</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 4. Main Loop (ISR) Execution Times (Continued)

<table>
<thead>
<tr>
<th>Routine</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bus Ripple Compensation</td>
<td>3</td>
</tr>
<tr>
<td>PWM Period Test</td>
<td>1</td>
</tr>
<tr>
<td>Inverse Park Transform (Vector Rotation)</td>
<td>28</td>
</tr>
<tr>
<td>PWM Period Test</td>
<td>1</td>
</tr>
<tr>
<td>I2C Send Stop Bit</td>
<td>1</td>
</tr>
<tr>
<td>Inverse Clarke Transform (2=&gt;3 Conversion)</td>
<td>18</td>
</tr>
<tr>
<td>PWM Period Test</td>
<td>1</td>
</tr>
<tr>
<td>PWM Space Vector Modulation</td>
<td>33</td>
</tr>
<tr>
<td>PWM Period Test</td>
<td>1</td>
</tr>
<tr>
<td>WDT refresh</td>
<td>1</td>
</tr>
<tr>
<td>Sub Function (stepped through list)</td>
<td>26</td>
</tr>
<tr>
<td><strong>Subtotal</strong></td>
<td><strong>224</strong></td>
</tr>
<tr>
<td>End of PWM Period Test</td>
<td>Variable</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>250</strong></td>
</tr>
</tbody>
</table>

Table 5 illustrates the execution time for each subfunction routine, executed every 2 ms.

Table 5. Subfunction Execution Times

<table>
<thead>
<tr>
<th>Subroutine</th>
<th>Time (µs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tachometer Update 1 of 3</td>
<td>7</td>
</tr>
<tr>
<td>Tachometer Update 2 of 3</td>
<td>26</td>
</tr>
<tr>
<td>Tachometer Update 3 of 3</td>
<td>2</td>
</tr>
<tr>
<td>Field Weakening</td>
<td>Unused</td>
</tr>
<tr>
<td>Slip Update</td>
<td>Unused</td>
</tr>
<tr>
<td>Speed Regulator</td>
<td>26</td>
</tr>
<tr>
<td>UART Communication</td>
<td>4</td>
</tr>
</tbody>
</table>
The purpose of this application is to demonstrate how a Z8 Encore!® MC microcontroller is used for efficient rotor flux field oriented vector control of a 3-phase AC induction motor. The following microcontroller features make this part particularly suited to motor control applications:

- The eZ8 processor core runs at up to 10 MIPS with a 20 MHz clock.
- The integrated PWM interface module provides the necessary timing and logic outputs for synthesizing 3-phase voltage in motor control applications.
- The 2.5 μs successive approximation register (SAR) ADC can be triggered by PWM events for sampling and reconstructing phase currents. The integrated Op Amp amplifies and provides offset adjustment of a single ground-referenced sense resistor, centered to one-half of the built-in ADC reference. The integrated comparator provides for fast overcurrent shutdown.
- The integrated communication peripherals (UART, I²C, and SPI) provide for system level communication. For example, the UART in this application provides a highly capable user command interface, yet simple enough to be optically coupled for electrical isolation.

In addition to the Z8 Encore!® MC microcontroller’s native advantages, the software design uses the following techniques to manage the processing task:

- To make time for the waveform calculations, the main loop operates at one fifth of the 20 kHz PWM chop rate (five 50 μs cycles, 250 μs per loop). This preserves most of the advantages of a high-frequency PWM rate, while still producing a good approximation of the appropriate output waveform.
- To free up additional time for main loop calculations, low-priority subfunctions are called in a 10-position round-robin schedule, so that each subfunction is executed only once every 2.5 ms. The lengthy tachometer update operation is further divided into three subfunction calls to limit execution time per call.
- Converting current measurements to a 2-phase vector for the rotation transform reduces calculation time compared to rotating a 3-phase vector.
- All signed values are expressed as pairs of unsigned variables. This requires extra code to perform an if-then-else for each calculation, but the eZ8 processor’s short-range branch instructions are efficient enough so the unsigned calculations can execute faster than the equivalent library routines, which are written for general-case signed arithmetic.

The example application and techniques described in this document should prove helpful for anyone who intend to develop motor control applications based on the Z8 Encore!® MC family of microcontrollers.
Appendix A—References

Further details on this topic and the eZ80® family of products is available in the references listed in Table 6.

Table 6. List of References

<table>
<thead>
<tr>
<th>Topic</th>
<th>Document Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Z8 Encore® CPU</td>
<td>eZ8 CPU User Manual (UM0128)</td>
</tr>
<tr>
<td>Z8FMC16100 MCU</td>
<td>Z8FMC16100 Series Product Specification (PS0246)</td>
</tr>
<tr>
<td>Motor Control</td>
<td>Hsu, P., &quot;A Short Course on Vector Control&quot;. San Jose State University, pp. 49-54, 1996.</td>
</tr>
</tbody>
</table>
Appendix B—Glossary

Definitions for terms and abbreviations used in this application note that are not commonly known are listed in Table 7.

Table 7. Glossary

<table>
<thead>
<tr>
<th>Term/Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADC</td>
<td>Analog-to-Digital Converter</td>
</tr>
<tr>
<td>DAC</td>
<td>Digital-to-Analog Converter</td>
</tr>
<tr>
<td>IGBT</td>
<td>Insulated-gate bipolar transistor</td>
</tr>
<tr>
<td>ISR</td>
<td>Interrupt Service Routine</td>
</tr>
<tr>
<td>PI</td>
<td>Proportional plus Integral</td>
</tr>
<tr>
<td>PWM</td>
<td>Pulse Width Modulation</td>
</tr>
<tr>
<td>Rotor</td>
<td>Rotating cage and windings of AC motor</td>
</tr>
<tr>
<td>Stator</td>
<td>Stationary windings of AC motor</td>
</tr>
</tbody>
</table>
Vector Control of a 3-Phase AC Induction Motor Using FMC16100 MCU

Appendix C—Schematic Diagrams

Figure 16 through Figure 22 are the schematic diagrams for the Vector Control application modules.

Figure 16. High-Voltage Main Board Schematic
Figure 17. FMC16 Control Module Schematic
Figure 18. HV Gate Drive Module Schematic
Figure 19. HV Power Module Schematic
Figure 20. Washing Machine Interface Module Schematic

- **Title**: Washing Machine Interface Module
- **Number**: 96C1021-001
- **Revision**: A
- **Date**: 26-Jun-2006
- **File**: C:\Program Files\Design Explorer 99 SE\PROJECTS\APPLICATIONS\Rex\HiVoltReferences\Washing Machine Interface Module
Figure 21. DAC Module Schematic
Figure 22. XP Command Module Schematic (Connects to Washing Machine Module)
Appendix D—Flowcharts

Figure 23 illustrates the overall program flow. After `main()` initializes the application, it drops into an infinite loop. Meanwhile, the PWM timer generates an event every 50 μs. The PWM ISR is timed to execute on every fifth PWM event, once every 250 μs. The ISR calls one of ten low-priority subfunctions on every loop cycle, so each subfunction is called once every 2.5 ms.

![Flowchart for the Application](image-url)
Figure 24 illustrates the PWM ISR main loop.

Figure 24. Flowchart for the PWM ISR