NEC

Application Note

78F0714

8-Bit Single-Chip Microcontroller

Permanent Magnet Synchronous Motor Control

µPD78F0714

Document No. U19166EE1V0AN00
Date published February 2008

Printed in Germany

Legal Notes

- **The information in this document is current as of November, 2007. The information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.**

- No part of this document may be copied or reproduced in any form or by any means without the prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

- NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.

- Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of a customer's equipment shall be done under the full responsibility of the customer. NEC Electronics assumes no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

- While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.

- NEC Electronics products are classified into the following three quality grades: "Standard", "Special" and "Specific".

- The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated "quality assurance program" for a specific application. The recommended applications of an NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.
  "Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.
  "Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime

systems, safety equipment and medical equipment (not specifically designed for life support).
"Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is "Standard" unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact an NEC Electronics sales representative in advance to determine NEC Electronics' willingness to support a given application.

(Note)
(1) "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
(2) "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

Regional Information

Some information contained in this document may vary from country to country. Before using any NEC product in your application, please contact the NEC office in your country to obtain a list of authorized representatives anddistributors. They will verify:

- Device availability
- Ordering information
- Product release schedule
- Availability of related technical literature
- Development environment specifications (for example, specifications for third-party tools and components, host computers, power plugs, AC supply voltages, and so forth)
- Network requirements

In addition, trademarks, registered trademarks, export restrictions, and otherlegal issues may also vary from country to country.

NEC Electronics Corporation
1753, Shimonumabe, Nakahara-ku,
Kawasaki, Kanagawa 211-8668, Japan
Tel: 044 4355111
http://www.necel.com/

[America]

NEC Electronics America, Inc.
2880 Scott Blvd.
Santa Clara, CA 95050-2554,
U.S.A.
Tel: 408 5886000
http://www.am.necel.com/

[Europe]

NEC Electronics (Europe) GmbH
Arcadiastrasse 10
40472 Düsseldorf, Germany
Tel: 0211 65030
http://www.eu.necel.com/

United Kingdom Branch
Cygnus House, Sunrise Parkway
Linford Wood, Milton Keynes
MK14 6NP, U.K.
Tel: 01908 691133

Succursale Française
9, rue Paul Dautier, B.P. 52
78142 Velizy-Villacoublay Cédex
France
Tel: 01 30675800

Tyskland Filial
Täby Centrum
Entrance S (7th floor)
18322 Täby, Sweden
Tel: 08 6387200

Filiale Italiana
Via Fabio Filzi, 25/A
20124 Milano, Italy
Tel: 02 667541

Branch The Netherlands
Steijgerweg 6
5616 HS Eindhoven,
The Netherlands
Tel: 040 2654010

[Asia & Oceania]

NEC Electronics (China) Co., Ltd
7th Floor, Quantum Plaza, No. 27
ZhiChunLu Haidian District,
Beijing 100083, P.R.China
Tel: 010 82351155
http://www.cn.necel.com/

NEC Electronics Shanghai Ltd.
Room 2511-2512, Bank of China
Tower,
200 Yincheng Road Central,
Pudong New Area,
Shanghai 200120, P.R. China
Tel: 021 58885400
http://www.cn.necel.com/

NEC Electronics Hong Kong Ltd.
12/F., Cityplaza 4,
12 Taikoo Wan Road, Hong Kong
Tel: 2886 9318
http://www.hk.necel.com/

NEC Electronics Taiwan Ltd.
7F, No. 363 Fu Shing North Road
Taipei, Taiwan, R.O.C.
Tel: 02 27192377

NEC Electronics Singapore Pte. Ltd.
238A Thomson Road,
#12-08 Novena Square,
Singapore 307684
Tel: 6253 8311
http://www.sg.necel.com/

NEC Electronics Korea Ltd.
11F., Samik Lavied'or Bldg., 720-2,
Yeoksam-Dong, Kangnam-Ku, Seoul,
135-080, Korea Tel: 02-558-3737
http://www.kr.necel.com/

Table of Contents

Chapter 1 Introduction

This application note presents a 3-phase permanent magnet synchronous motor control software developed for NEC 8-bit microcontroller uPD78F0714 with sinusoidal waveform. The 78F0714 microcontroller facilitates a dedicated peripheral for 3-phase motor control, enabling easier motor drive with AC induction motors and/or permanent magnet DC/AC motors (BLDC/PMAC).

The presented software library is written in standard C language and provides a set of sample functions for sinusoidal waveform generation, the synchronization mechanism and closed loop control of PMSM drive. The source code is compatible with IAR (http://www.iar.com/) Embedded Workbench C/C++ compiler and debugger tools.

This software library can be taken as a demonstration tool together with NEC Motor Control Starter Kit (MC-LVKIT) and a Maxon EC motor. By using the sine wave generation and speed regulation algorithms provided in the library, user can concentrate on the application development with a few parameter adaption.

A PMSM usually consists of a magnetic rotor and wound stator. The magnetic rotor rotates as the magnetic field produced by the wound stator changes. Such construction requires no brushes in between, producing greater efficiency and power density. It provides high torque-to-inertia ratios and also reduces the maintenance cost. A PMSM generates magnetic flux using permanent magnets on the rotor, which generates torque most effectively when it is perpendicular to flux generated by the stators. To maintain near-perpendicularity between stator flux and rotor flux, a control method with position-speed feedback loop are popularly used for controlling a PMSM.

Prerequisite The prerequisite for using this library is the basic knowledge of C programming, AC motor drives and power inverter hardware. In-depth know-how of motor control functions is only required for customizing existing modules and when adding new ones for complete application development.

Note This control software only functions with NEC Library for Motor Control in the preliminary version.

Disclaimer The demo control software described in this application note is used for demostration purpose only. Please do not use it for any purpose beside demostration and evaluation purpose.

Chapter 2 Working System

2.1 System Feature

- The motor control algorithm employs PI closed-loop control. The power switches are controlled by means of sinusoidal pulse width modulation (PWM).
- The rotor position feedback hardware elements are Hall sensors.
- The motor is capable of rotating in both direction and has a speed range from 500 rpm to 9000 rpm.
- Rotation direction, control profile, speed and current overshoot can be controlled with help of NEC Motor Control Visualizing GUI.

2.2 Development Tools

2.2.1 In-Circuit Emulator

The user software can be downloaded to the target device with on-chip debugging (OCD) emulator NEC 78K0MINI, a separately sold device, which supports pseudo real-time RAM monitoring and C-Spy debugging.

Figure 2-1 NEC MINICUBE Connection

For more information about this OCD emulator, please refer to document U17029EJ3V0UM00.

2.2.2 Integrated Development Environment

This library is compiled with C/C++ compilers and debuggers for NEC Electronics 78K0 of IAR Embedded Workbench 4.40 or higher version.

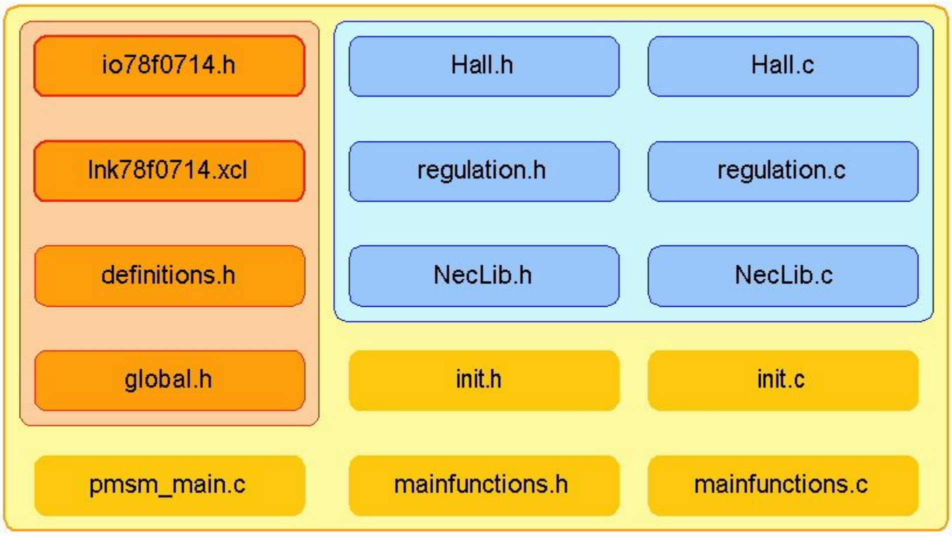
Note You can download the 4K limited free version of IAR Embedded Workbench to compile the software library.

2.3 Source Code

2.3.1 Downloads

The source code of this control software can be downloaded on *http://www.eu.necel.com/docuweb/*

2.3.2 File Structure



Kernel Files **io78f0714.h**

- Microcontroller specific files, registers addresses

lnk78f0714.xcl

- Microcontroller specific files, segment definition

definitions.h

- Macro definitions

global.h

- Global variables definitions

Main Programs

pmsm_main.c

- Entry program

init.h, init.c

- Hardware initializing functions

mainfunctions.h, mainfunctions.c

- Sine waveform output function
- ADC measured data storage function
- Current overshoot precaution function
- Software variables reset function

Control Programs

Hall.h, Hall.c

- Interrupt sub-routines for hall sensor signal change

regulation.h, regulation.c

- PI controller function

NecLib.h, NecLib.c

- NEC Library for Motor Control

Chapter 3 PMSM Motor Fundamentals

This chapter will explain the basic process of developing a control software for PMSM.

3.1 Target Motor

PMSM

A PMSM rotates at a fixed speed in synchronization with the frequency of the power source independent of the load, provided an adequate field current is applied on the motor windings.

A 3-phase PMSM drive commutates the phase windings sinusoidally such that the stator magnetic field is at 90 degrees to the PM rotor magnetic field, producing a maximum torque on the rotor.

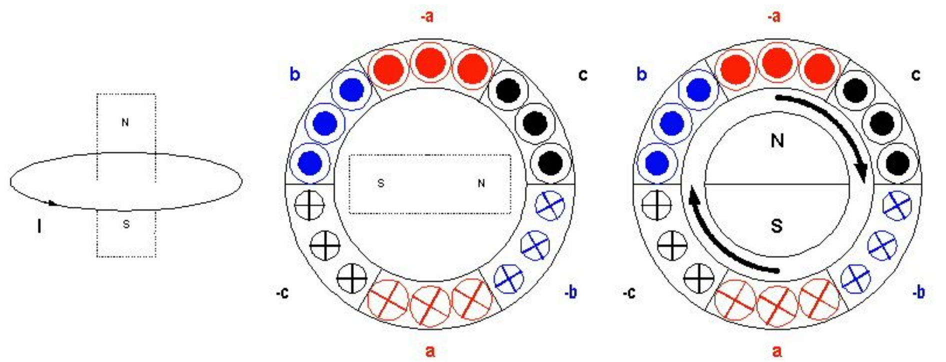


Figure 3-1 PM Motor Cross Section and Electromagnetism

PMSM and BLDC

By design, the basic architecture of a BLDC motor and a PMAC motor have no intrinsic differences. A BLDC motor can normally be driven by alternating current and PMAC motor by direct current. However, the windings are specially designed to retain a trapezoid form of current for BLDC motor and a sinusoidal form for PMAC motor.

	BLDC motor	PMAC motor
Phase Voltage and Phase Current	rectangular	sinusoidal
Current Peak Value	high	low
Torque	with commutation ripples	smooth
Noise	high	low
Core Power Loss	high	low
Switching Power Loss	low in inverter	high in inverter
Implementation	simple	relatively complicated

In a BLDC motor, only two windings carry current at any given time. This reduces the winding utility by 33%. On the other hand, in a three-phase PMAC motor, three-phase sinusoidal voltages are applied on all three windings and all three windings carry current at all times. This will naturally increase switching loss in inverter.

PMSM Model

Stator voltage differential equations:

$$u_A = R_w i_A + \Psi_A$$

$$u_B = R_w i_B + \Psi_B$$

$$u_C = R_w i_C + \Psi_C$$

R_A, R_B, R_C are the resisitence in the stator

Stator and rotor flux linkages:

$$\Psi_A = L_{AA} i_A + L_{AB} i_B + L_{AC} i_C + \Psi_{A,rotor}$$

$$\Psi_B = L_{BA} i_A + L_{BB} i_B + L_{BC} i_C + \Psi_{B,rotor}$$

$$\Psi_C = L_{CA} i_A + L_{CB} i_B + L_{CC} i_C + \Psi_{C,rotor}$$

$$\Psi_{A,rotor} = \Psi_{flux} \cdot \cos\theta$$

$$\Psi_{B,rotor} = \Psi_{flux} \cdot \cos(\theta + \frac{2\pi}{3})$$

$$\Psi_{C,rotor} = \Psi_{flux} \cdot \cos(\theta + \frac{4\pi}{3})$$

L_{AA}, L_{BB}, L_{CC} are the stator inductances.

L_{AB}, L_{AC}, L_{BA}, L_{BC}, L_{CA}, L_{CB} are the mutual inductances.

3.2 Sine PWM Digital Control

NEC 78F0714 8-bit micro-controller contains a 10-bit inverter control timer. This timer consists of an 8-bit dead-time generation timer, and allows non-overlapping active-level output, as in the figure depicted below.

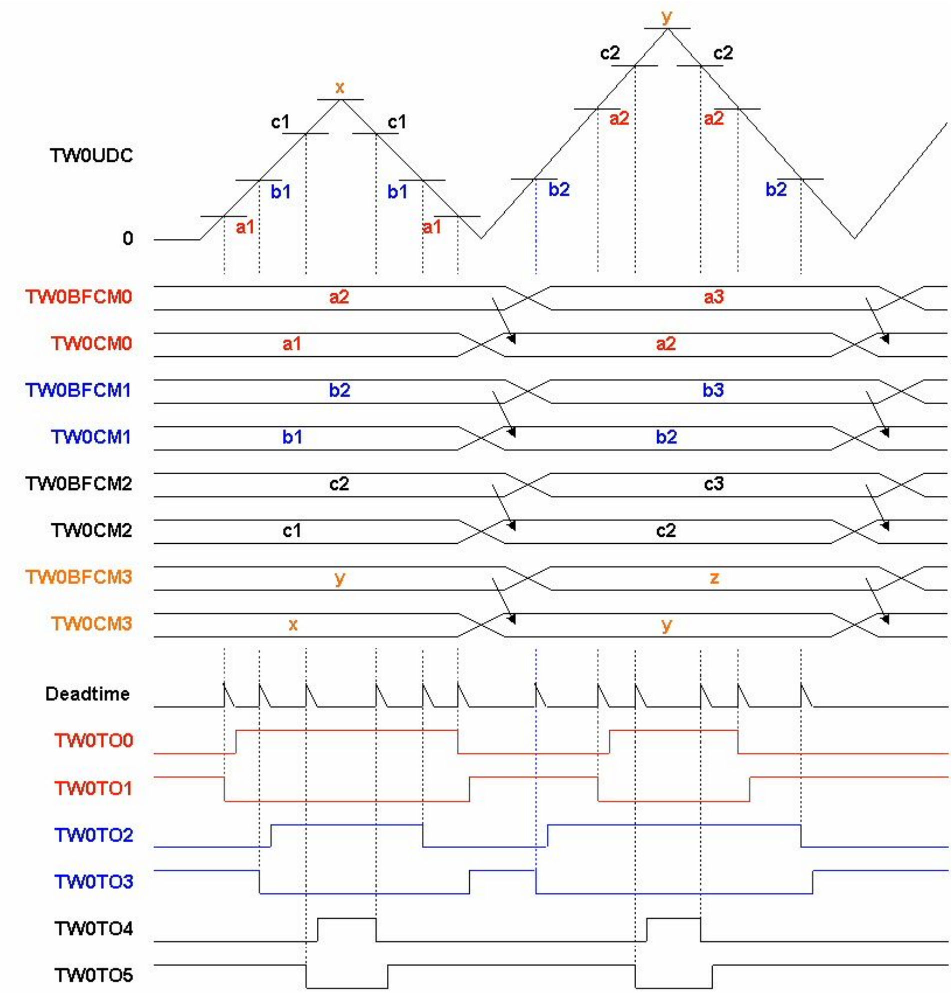


Figure 3-5 Inverter Timer Outputs

A sinusoidal waveform can be represented by a number of relative values compared to the triangular carrier signal. The frequency of the sinusoidal waveform is dependent of the number of relative values within a period. The ratio of sine wave peak value to triangular wave peak value determines the output waveform amplitude.

To gain a precise control of output waveform amplitude, the maximum of relative value (resolution) should be as large as possible. But higher resolution will lead to lower PWM frequency. Frequencies of lower than 20KHz can produce noises within human audible threshold. On the other hand, the faster the PWM frequency is, the less time the MCU will have to execute commands. To minimize acoustic disturbance while having more time for program routines, 20KHz is usually selected as PWM cycle.

The NEC micro-controller 78F0714 works with a 20MHz oscillator. The compare counter maximum (half of the PWM cycle) is thus 20MHz / 20 KHz / 2 = 500.

The base frequency of a sine wave is dependent on the number of sampling points. In other words, it depends on the size of the sine look-up table. Too few sampling points will lead to a so-called staircase effect. The staircase effect will cause excessive motor current distortion, which causes higher heat dissipation. Superfluous sampling points consume precious memory in the micro-controller.

A good rule is to divide the PWM carrier frequency by the maximum desired sine wave frequency. The nominal speed of the motor in case is around 10000 RPM, corresponding to a sine wave frequency of 166.67Hz. The number of look-up table sampling points is 120.

The generated PWMs are then transmitted to transistor switches of an inverter in the control hardware. DC power will thus be converted to AC power at the required frequency and amplitude.

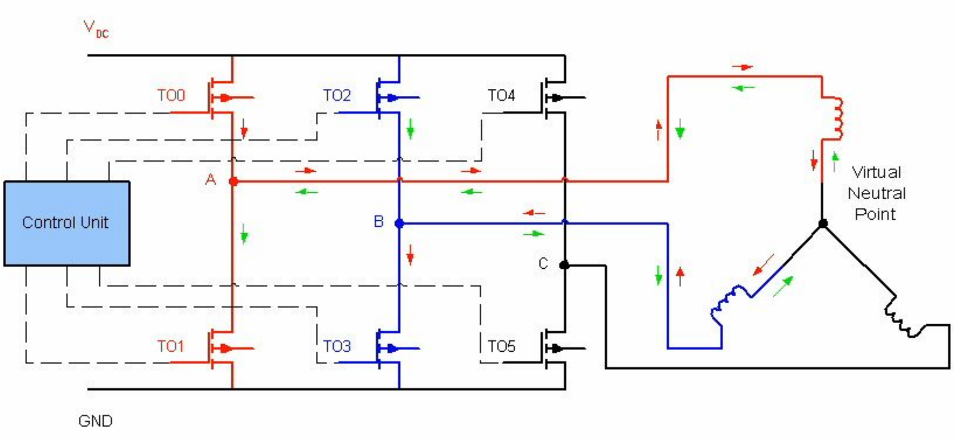


Figure 3-6 Half-bridge Circuits and Motor Connection

The inverter is composed of three equivalent half-bridge circuits. Only one side of the half-bridge circuit, either the high side or the low side, should be switched on at any given time. Otherwise the control hardware will be short-circuited. Dead-time is then inserted to avoid such situation.

At the outputs of the half-bridge circuits, 120-degree-shifted three-phase sinewave waveforms can be detected with a simple RC-circuit.

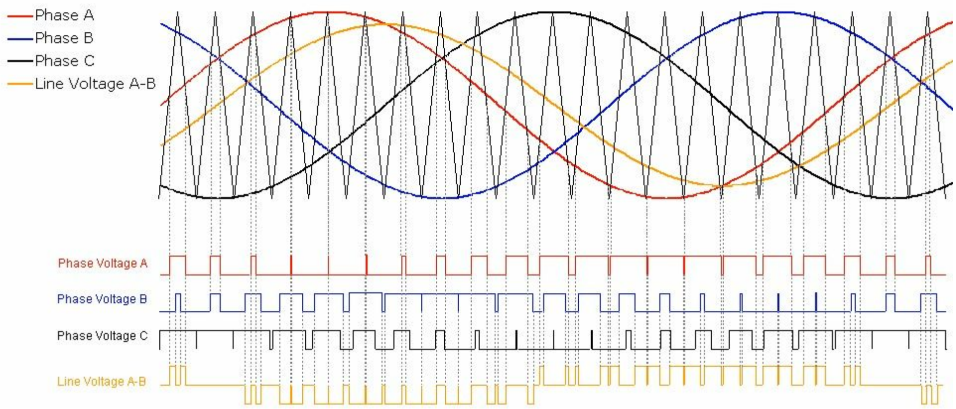


Figure 3-7 Sinusoidal Control Waveform and corresponding PWM Signals

The amplitude of the line-voltage is always less than that of the output waveform, around 88.6% of the peak value. This effect is an intrinsic limitation of the sine PWM control method.

3.3 Synchronization with Hall Sensors

Determining Rotor Position

Hall sensors are used to determine rotor position during operation. Using the built-in Hall sensors in phase winding can simplify control logic and does not require extra circuit to process the signals. Electricity carried through the phase winding conductor will produce a magnetic field that varies with current, and such a Hall sensor can be used to measure the current without interrupting the circuit.

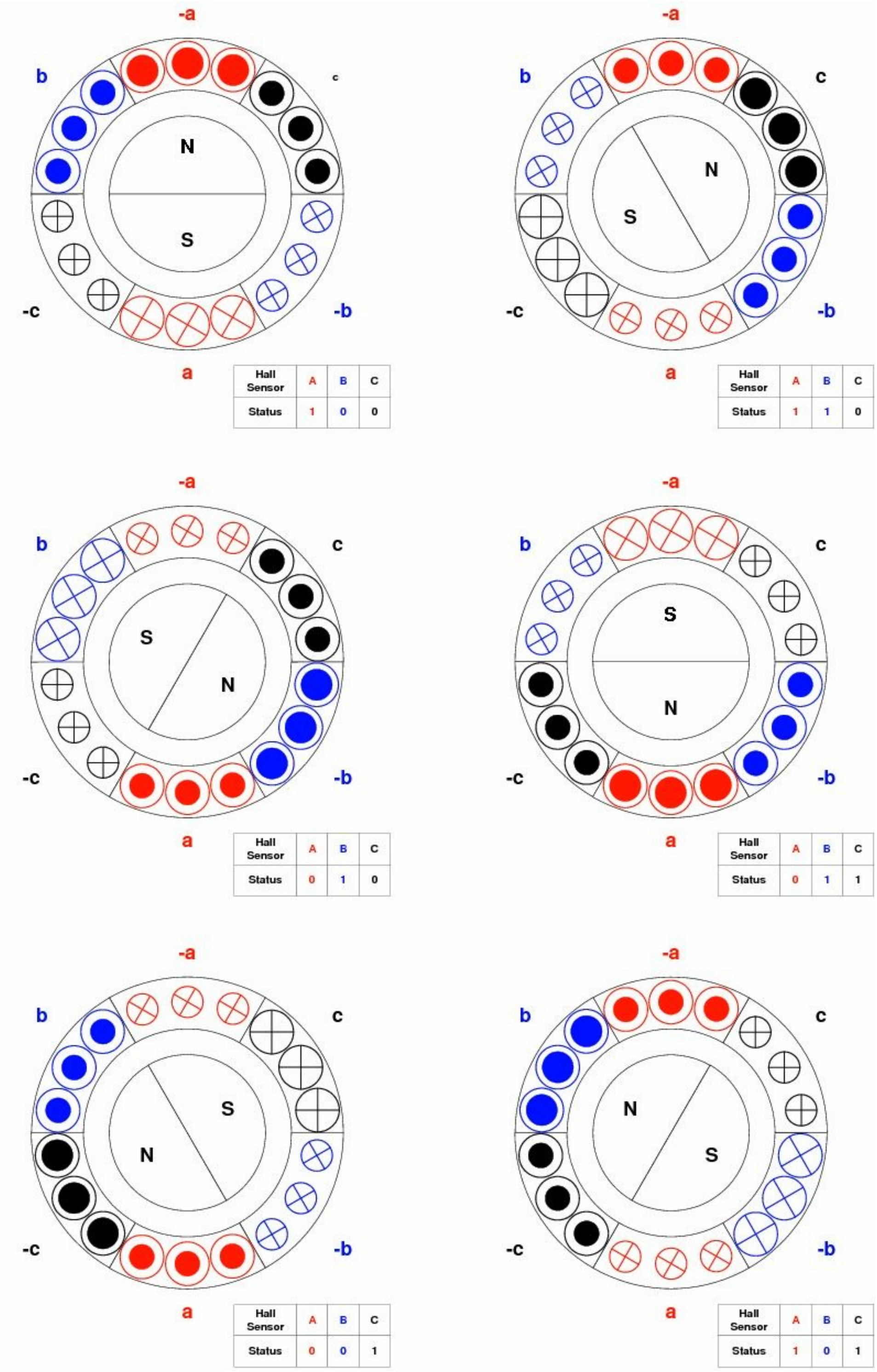


Figure 3-8 Rotor Position and Hall Sensors Outputs

By connecting Hall sensor signals to microcontroller interrupt input pins, the interrupt request will be generated when the signals change. The corresponding subroutine can thus estimate the rotor position according to turing direction and update the output waveform angle accordingly.

Table 3-1 Section allocation by Hall sensor signals combination

Hall C	Hall B	Hall A	Section
0	0	0	Invalid
0	0	1	1
0	1	0	3
0	1	1	2
1	0	0	5
1	0	1	6
1	1	0	4
1	1	1	Invalid

The sections are numbered according to the rotor position illustrated in *Figure 3-5* .

The relationship between Hall sensor signals and the desired control signal waveform can be depicted in the figure below by resorting the section orders.

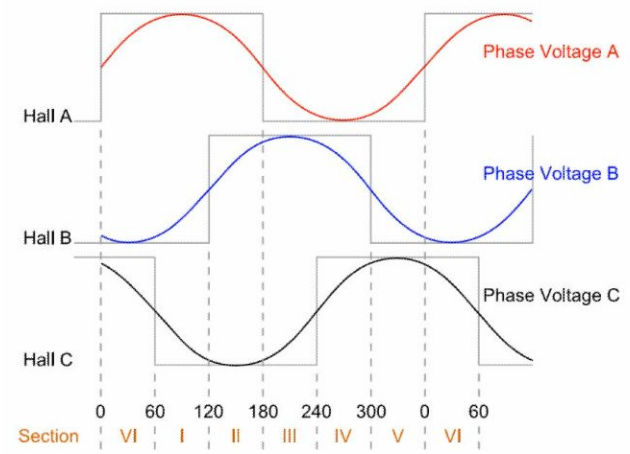


Figure 3-9 Motor Currents and Hall Sensor Signals rotating clockwise

3.4 Close Loop Control

Introduction

PI controller is a generic feedback controller commonly used to implement closed-loop control. A PI controller responds to an error created by subtracting desired value from output quantity. Then it adjusts the controlled quantity to achieve the desired system response. The controlled value can be any measurable system quantity such as speed, torque or flux. The parameters of a PI controller can be adjusted empirically by tuning one or more gain values and observing the change in system response. A digital PI controller is executed at a periodic sampling interval. It is assumed that the controller is executed frequently enough so that the system is under proper control.

The proportional (P) contribution of the controller results from multiplying the error by a P gain value. A larger error results in larger proportional contribution. This P term contribution tends to reduce the overall error as time elapses. However, the P term has less effect as the error approaches zero. Most systems with P term only controller have a steady error and do not converge. Large P gain value imposes a tight control on the output value but an excessively large proportional gain will lead to process instability.

The integral (I) contribution of the controller is used to eliminate small steady error but always brings along larger overshoot. Errors of the system are multiplied by I gain value and accumulated over time into an error buffer. This error buffer forms the I term output of the PI controller.

Speed Measurement

A closed-loop PI speed controller requires actual motor speed for controller input to eliminate the speed difference. The actual motor speed is in this software is realized by computing the time elapsed between two consecutive Hall signal changes.

Hall sensor C signal is connected to interrupt pin INTP5. Interrupt request will thus be generated each time the Hall sensor C signal changes. 16-bit Timer Counter 01 is set to run at 78KHz and is read during this interrupt service. The timer counts multiplying the Timer Counter 01 interval is the time elapsed since the last Hall sensor signal change.

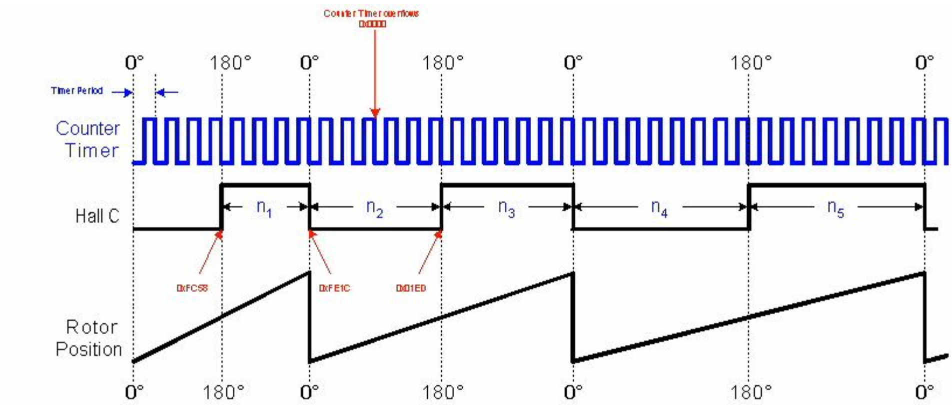


Figure 3-10 Speed Measurement with Hall Sensor C Signal

PI Controller

The digital PI controller is called at fixed time interval by checking the overflow flag of a dedicated timer counter.

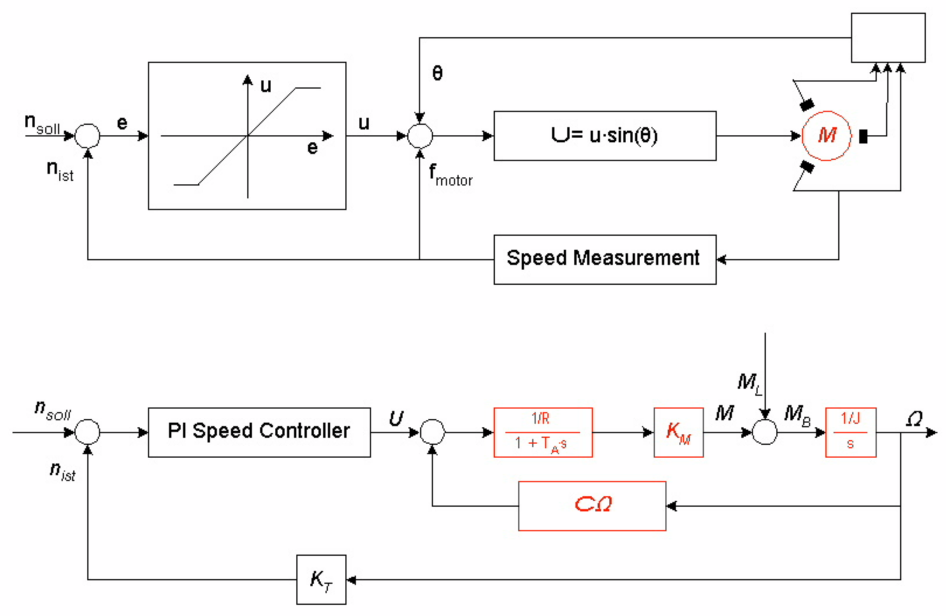


Figure 3-11 PI Speed Controller Implementation

The speed difference is computed by subtracting actual motor speed from setpoint. The difference will be multiplied by P-gain and I-gain factor. The sum of the two gains will be normalized to duty cycle of the inverter timer to update the output waveform amplitude.

The parameters of the PI controller can be determined by using the Ziegler-Nichols closed-loop tuning method.

Table 3-2 Ziegler-Nichols closed-loop parameter tuning method

Controller Type	P-Gain	Reset Time
P Controller	0,5 x K_C	-
PI Controller	0,45 x K_C	K_C / T_C

K_C is the critical value of P-gain factor, with which the P-only Controller results in an ultimate periodic oscillation of output response.

T_C is the period of the oscillation.

3.5 System Overview

The sensorred control system presented in this application note has the following implementation:

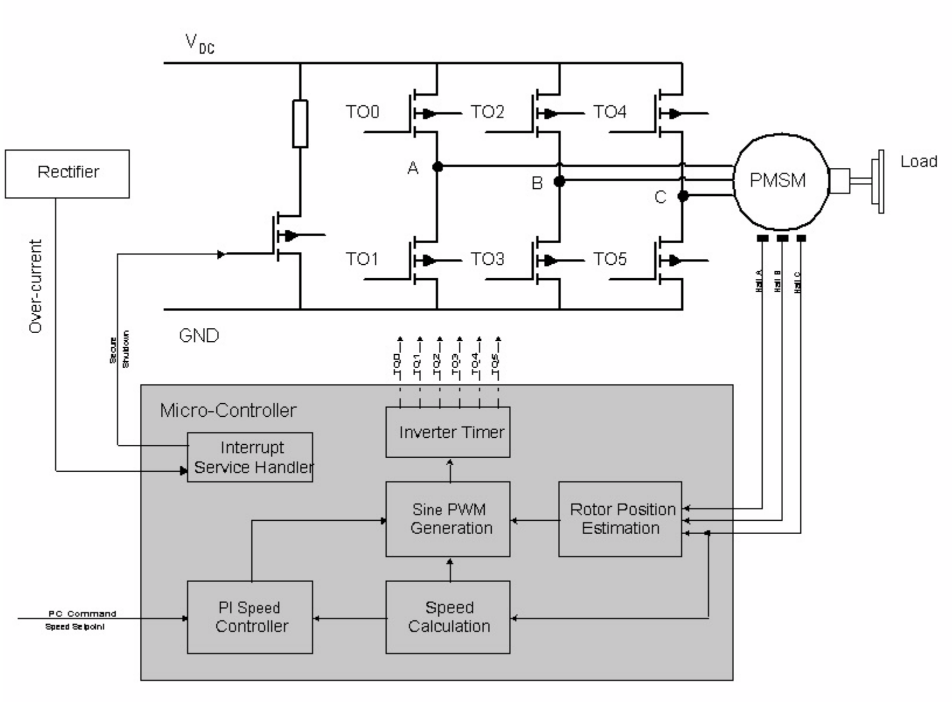


Figure 3-12 Implementation of the control software

The setpoint of motor speed is specified with user input made on the NEC Visualizing GUI for Motor Control.

The sine angle of the output waveform is updated on Hall sensor signal change and the measured frequency f_{motor} will impose on the motor to keep it in synchronization.

When the current in the motor overshoots the threshold value of the rectifier, an interrupt request will be made to the microcontroller. The demo software will shut off the MOSFET with a switch signal during the interrupt service.

The control system has the following states

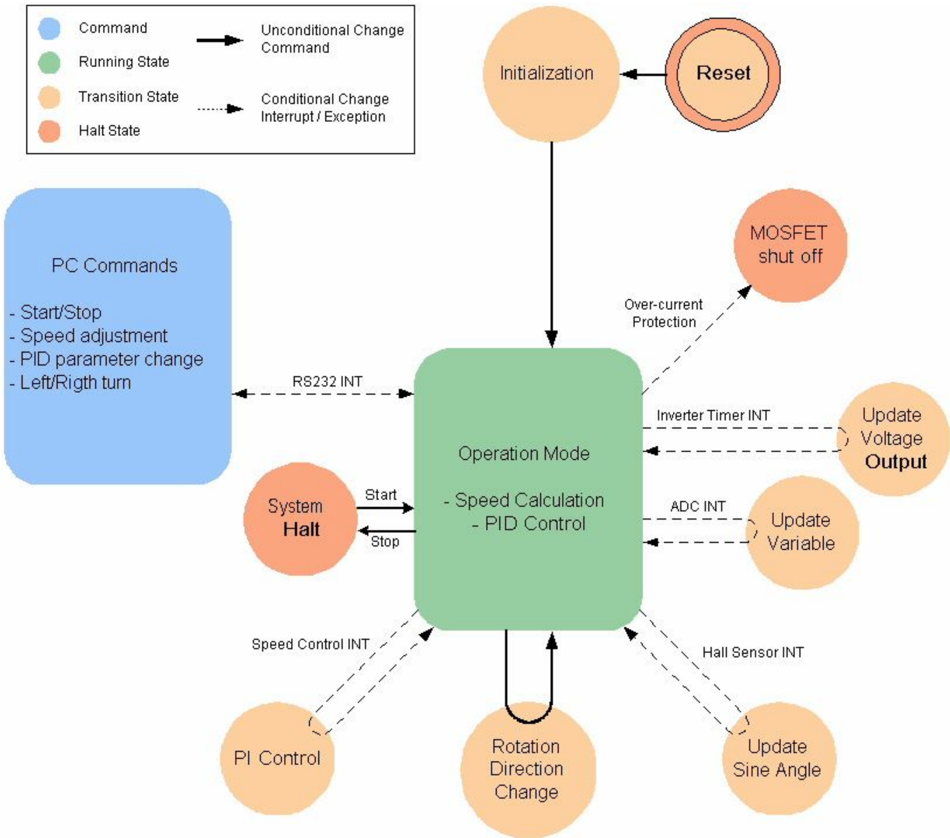


Figure 3-13 System States of the Control Software

When **Stop** command is sent to the control software, the motor will be stopped even when the setpoint is not zero.

AD converter can be used to measure system variables such as motor shunt current and phase currents. Such features are not implemented in the demo software presented in this application note.

Chapter 4 Getting Started

To debug the user program, the NEC 78K0MINI in-circuit emulator (ICE) should be connected with the micro board through the 2JP7 on-chip debugging connector as shown in *Figure* .

Figure 4-1 MINICUBE Connection with NEC StarterKit for Motor Control

To debug software using desired working frequency, an external oscilltor must be mounted on the ICE oscillator slot.

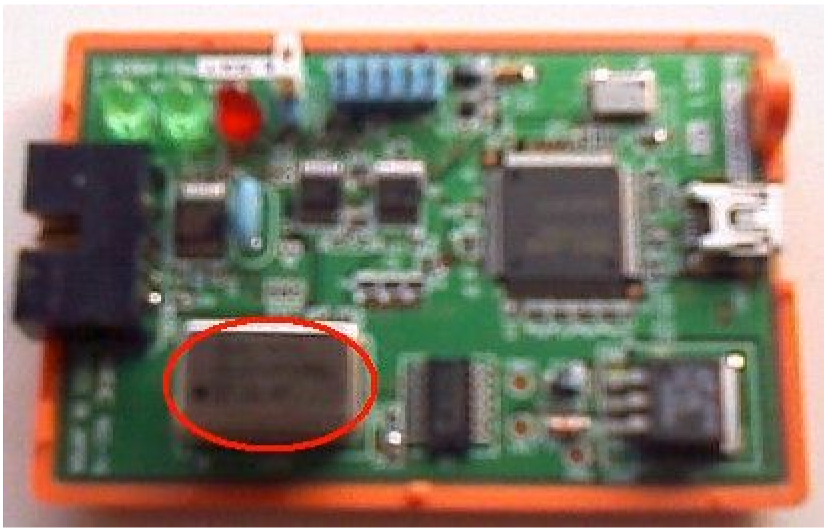


Figure 4-2 External Oscillator Installation

Chapter 5 Software Configuration

This chapter will describe the possibilities of enabling / disabling some features implemented in the demo control software.

5.1 Control Registers

Some features implemented in this demo control software can be enabled or disabled by manipulating the macro definitions in header file **definitions.h** in the sample source code.

The major control variables are defined in the header file **global.h**. Possible values of the control variables are described below.

definitions.h **ENABLE_UART**

This macro definition enables UART communication.

To reduce flash memory consumption, user can disable UART communication, by commenting out the definition.

CLOSE_LOOP

This macro definition indicates the compiler include the close loop control code.

To exclude closed-loop control function, user can comment out the definition.

SENSORRED

This macro definition indicates the compiler include the close loop control code. To exclude this code, comment out the definition.

norm_constant

This constant is used to calculate the motor actual speed from timer counts.

actual_count_min, **actual_count_max**

These two constants are used to limit the timer count value. Value exceeding the limit will be ignored.

global.h **onoff**

This flag stores information of system on / off state.

0 = system stopped, 1 = system set to start, 2 system running

disconnx

This flag indicates if the motor system is disconnected from visualizing module.

0 = normal, 1 = system disconnected

OverCurrent

This flag indicates if motor current overshoot occurred.

0 = normal, 1 = motor current oveshot

Motor_Dir

This flag indicates the motor rotation direction.

0 = anti-clockwise, 1 = clockwise

Loop_Sel

This flag indicates the motor drive control method.

0 = open-loop, 1 =closed-loop

SecureSel

This flag indicates if hardware over-current protection is used.

0 = disabled, 1 = enabled

SensorSel

This flag indicates the motor drive synchronization method.

0 = sensored control, 1 = sensorless control

BemfA_sel, **BemfB_sel**, **BemfC_sel**

These three flags determine which channel of the Back-EMF should be measured by A/D-Converter.

0 = not selected, 1 = selected.

ovf

This flag indicates the number of times the timer overflows.

After the third time of timer overflow, the motor speed will be set to 0, and system will be stopped.

5.2 Program Area Consumption

By changing the macro definitions in the header file **definitions.h**, program size of the demo software varies as shown in the table below.

Table 5-1 Program size with different settings

	ROM Area	RAM Area
Close Loop with UART	3,820 bytes	446 bytes
Open Loop with UART	2,870 bytes	430 bytes
Close Loop without UART	2,787 bytes	248 bytes
Open Loop without UART	1,834 bytes	232 bytes

Chapter 6 Sample Result

This chapter will show some sample results using the control software described in this application note.

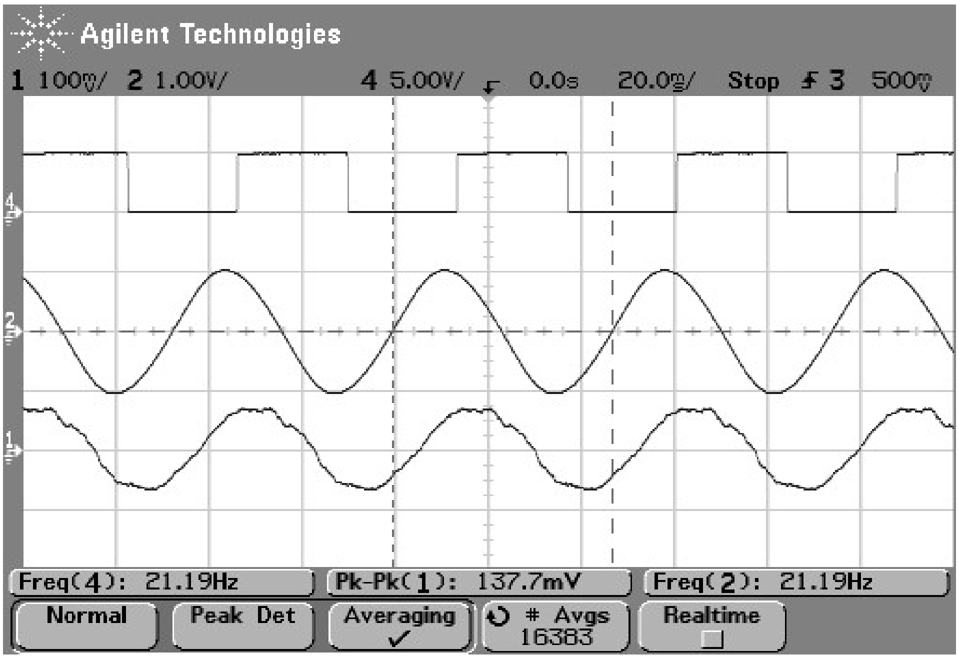


Figure 6-1 Phase Voltage, Current and Hall Sensor Signal at 1272 rpm (21.19Hz)

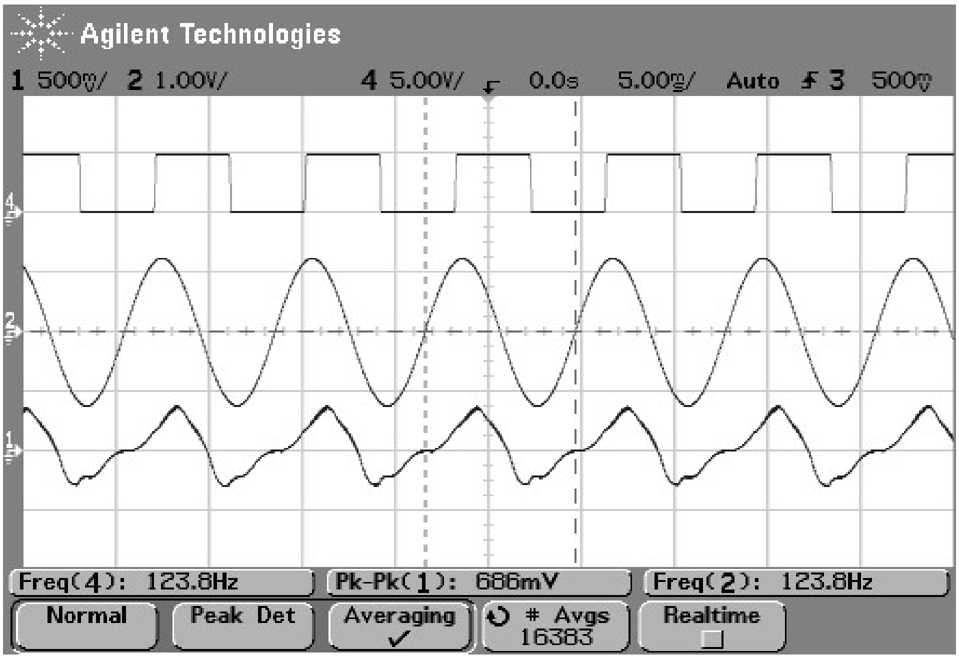


Figure 6-2 Phase Voltage, Current and Hall Sensor Signal at 7428 rpm (123.8Hz)

Channel 1 depicts the motor phase current.

Channel 2 depicts the phase control signal waveform.

Channel 4 is the Hall sensor signal of the correspondent phase.

The figures below display the system dynamics of the in-use PI speed controller under different circumstances.

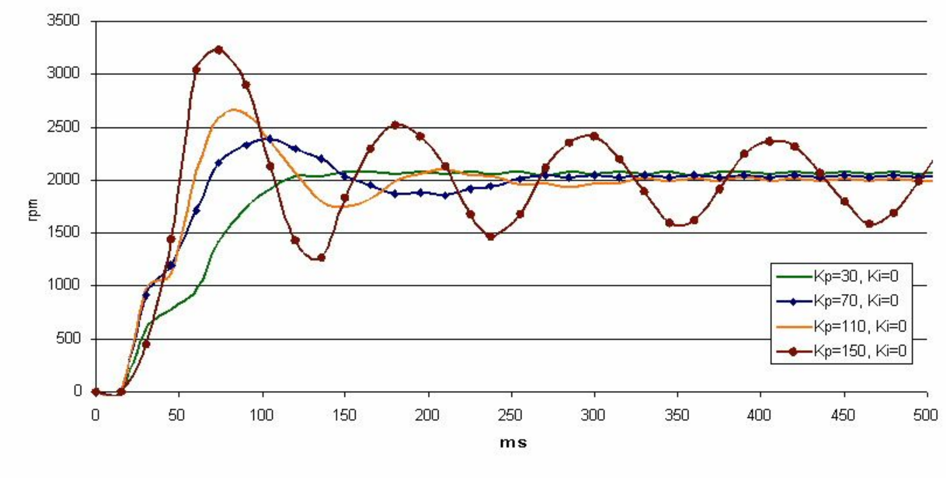


Figure 6-3 P-Gain-only Dynamic Performance with Kp varying from 0.03 to 0.146

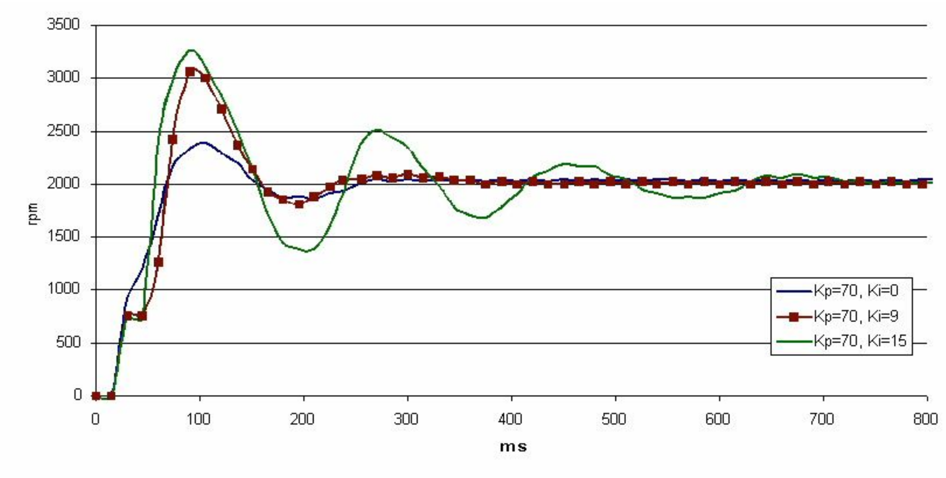


Figure 6-4 PI Controller Dynamic Performance with Ki varying from 0 to 0,015

Figure 6-3 displays the system response with proportional error control.

Figure 6-4 displays the system response under a PI speed controller with different settings of I-gain.

Chapter 7 Source Code

This chapter will list the source code of the demo control software described in this application note.

7.1 Marco Definitions

```
#ifndef __DEFINITIONS_H__
#define __DEFINITIONS_H__

#include <io78f0714.h>

#define SENSORRED
#define ENABLE_UART
#define CLOSE_LOOP

#define CLEAR           0
#define SET             1

/* Regulator constants: */
#define norm_constant       2343750    // constant to norm the timer count
#define actual_count_min        213   // minimum valid count value  (11003 rpm)
#define actual_count_max      15625   // maximum valid count value  (150 rpm)

#endif
```

7.2 Global Variable Definitions

```
#ifndef __GLOBAL_H__
#define __GLOBAL_H__

#include <io78f0714.h>
#include <migration.h>
#include <intrinsics.h>
#include "definitions.h"

unsigned int MaxSpeed = 9540;

/* exchange variables for communication */
/* must be defined as unsigned integer  */

unsigned int motor_rpm, reg_Y, DeltaX;                  // measured values
unsigned int Kp, Ki;                               // controller parameters
unsigned int PWM_CYCLE, DUTY_CYCLE, setpoint, deadtime;  // control variables

// state variable
unsigned int onoff;                      // 0 = off, 1 = on, 2 = running
unsigned int disconnx;
unsigned int OverCurrent, SecureSel;
unsigned int Motor_Dir;                 // 0: anti-clockwise, 1: clockwise
unsigned int Loop_Sel;                   // 0: closed loop 1: open loop

/* control variables */
unsigned int DCLevel = 249;
unsigned int phase_A;                    // angle variable
unsigned int steps;                      // frequency variable
unsigned char newspeed;
unsigned char ovf;

/* regulation variables */
```

```
unsigned int actual_count = 0;         // actual timer count value for speed
                                         // (16 Bit = [0 ... 65535])
long Integrator;

/* motor position control variables */
unsigned char Motor_Pos;

unsigned int PHASE_A_LEFT[]  = {0, 0, 8130, 0, 49090, 0, 59330,
                                  0, 28610, 0, 18370, 0, 38850};
unsigned int PHASE_A_RIGHT[] = {0, 0, 49090, 0, 8130, 0, 59330,
                                  0, 28610, 0, 38850, 0, 18370};

#endif
```

7.3 Main Entry Program

```
/* pmsm_main.c */

#pragma language = extended

#include "global.h"
#include "definitions.h"

#include "init.h"
#include "regulation.h"
#include "mainfunctions.h"

#ifdef ENABLE_UART
  #include "NecLib.h"
#endif

//-----------------------------------------------------------------------------
// Option Byte
//-----------------------------------------------------------------------------
#pragma constseg = OPTBYTE
__root const unsigned char option = 0x00;
#pragma constseg = default

#pragma constseg = SECUID
__root const unsigned char secuid[] =
    {0x00, 0xFF, 0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF,0xFF};
#pragma constseg = default

//===========================================================================
// MAIN
//===========================================================================

#pragma language = extended
void main(void)
{
  _DI();                    // Disable all interrupts

  /* uPD init */
  init_Interrupt();
  system_init();
  var_init();

#ifdef ENABLE_UART
  extern unsigned int* CommVarAddr[variable_no];
  ClearAddresses ();

  CommVarAddr[RPM]        = &motor_rpm;
  CommVarAddr[REGY]       = &reg_Y;
  CommVarAddr[XD]         = &DeltaX;
  CommVarAddr[KP]         = &Kp;
  CommVarAddr[KI]         = &Ki;
  CommVarAddr[PWMCYCLE]   = &PWM_CYCLE;
  CommVarAddr[DUTYCYCLE]  = &DUTY_CYCLE;
  CommVarAddr[SETPOINT]   = &setpoint;
```

```
  CommVarAddr[DEADTIME]  = &deadtime;
  CommVarAddr[ONOFF]     = &onoff;
  CommVarAddr[DISCONNX]  = &disconnx;
  CommVarAddr[OC]        = &OverCurrent;
  CommVarAddr[MOTORDIR]  = &Motor_Dir;
  CommVarAddr[LOOP]      = &Loop_Sel;
  CommVarAddr[SSD]       = &SecureSel;
  CommVarAddr[MAXSPEED]  = &MaxSpeed;

  SaveDefaultValue ();

  CommStart();
  start_51;                 // transmission interval timer
#endif

#ifdef CLOSE_LOOP
  start_50;                 // regulation interval timer
#endif

  _EI();                    // Enable all interrupts

  while (1)
  {
    if (onoff)
    {
      if (onoff == 1)                    // system starts
      {
        Motor_Pos    = P0 & 0x0E;
        phase_A      = PHASE_A_LEFT[Motor_Pos];
        steps        = 62;               // reset frequency variable
        system_start();
        onoff        = 2;
      }

#ifdef CLOSE_LOOP
      if (onoff == 2)
      {
        if ((!Loop_Sel) && TMIF50) PIRegulation();
      }
#endif
    }
    else
    {
     system_stop();              // setpoint changed, and =0, system stops
      actual_count  = 0;
      motor_rpm     = 0;
      Integrator    = 0;
      newspeed      = 0;
    }

    if (newspeed > 1)
    {
     motor_rpm     = norm_constant / actual_count;     // normalize speed
      steps         = motor_rpm / 10;                   // synchronization
      newspeed      = 1;
      ovf           = 0;
    }
    else if (newspeed == 1 && OVF00 == 1)
    {
      ovf++;
      OVF00 = 0;
      if (ovf > 2)            // after 3 * 78.125KHz * 65536 = 2,5s
      {
        ovf = 0;
        motor_rpm = 0;
        if (setpoint)
        {
          disconnx = 1;       // device error, shutdown system
        }
      }
    }

#ifdef ENABLE_UART
```

```
    if (disconnx)
    {
      var_init();
    }

    if (TMIF51)
    {
      TMIF51 = 0;
      CommUpdate ();
    }
#endif
  }
}
```

7.4 System Initialization

```
/* init.c */
#include "init.h"
#include "definitions.h"

extern unsigned int deadtime;
extern unsigned int PWM_CYCLE;

/*********************************************/
/****       H/W µPD INIT                ****/
/*********************************************/
void system_init(void)
{
  init_PORT();
  init_OSC();
  init_TW0();
  init_TM00();
  init_AD();
  init_50();
  init_51();
  init_UART00();
}

/*************************/
/****   System start   ****/
/*************************/
void system_start(void)
{
  start_TW0;
  start_TM00;
  //start_AD;
  INTTW0UD_on;
}

/************************/
/****   System stop   ****/
/************************/
void system_stop(void)
{
  INTTW0UD_off;
  stop_TW0;
  stop_TM00;
  stop_AD;
}

/************************/
/****   System stop   ****/
/************************/
void init_PORT(void)
{
                          // 7 6 5 4 3 2 1 0
  EGP = 0x0E;             // 0 0 0 0 1 1 1 0
  EGN = 0x0F;             // 0 0 0 0 1 1 1 1
                          //         | | | |___TOFF7 Secure shut off
```

```
                        //          |_|_|_____Hall Ext. IRQ both edges

  /*
  Low voltage power module TRIP signal shuts off power to MOSFETs
  It is active high and driven by P5.4
  */
  PM5   = 0xEF;                         /* P53 input for CR01 */
  P54   = CLEAR;                        /* Clear Trip Signal  */
}

void init_OSC(void)
{
  // IMS - Internal Memory Size
  // 7 6 5 4 3 2 1 0
  // | | | 0 |_|_|_|____ROM3 ROM2 ROM1 ROM0
  // | | |                 1    0    0    0   32KB Internal ROM
  // |_|_|___RAM2 RAM1 RAM0
  //            1    1    0   1024 bytes interal high-speed RAM
  IMS = 0xC8;                           /* Memory size switching */

  PCC   = 0x00;                         /* Sets division ratio */

  // MCM bit 7 6 5 4 3 2 1 0
  //         0 0 0 0 0 0 0 |____MCM0    0: Ring OSC   1: X1
  MCM0  = 1;                            /* X1 as input clock */
}

/*****************************/
/****  Inverter timer    ****/
/*****************************/
void init_TW0(void)
{
  TW0M     = 0;
  TW0TRGS  = 0;
  TW0C     = 0x01;                      // underflow every second time
  TW0OC    = 0;
  TW0CM3   = PWM_CYCLE;               /* buffer - PWM carrier frequency */
  TW0CM0   = 0;                         /* buffer - PWM duty phase A */
  TW0CM1   = 0;                         /* buffer - PWM duty phase B */
  TW0CM2   = 0;                         /* buffer - PWM duty phase C */
  TW0DTIME = deadtime;                  /* Dead time  */
  TW0BFCM4 = PWM_CYCLE/2;               /* ADC Trigger */
  TW0BFCM3 = PWM_CYCLE;                 /* PWM carrier frequency */
  TW0BFCM2 = 0;                         /* initiial PWM duty phase C */
  TW0BFCM1 = 0;                         /* initiial PWM duty phase B */
  TW0BFCM0 = 0;                         /* initiial PWM duty phase A */
}

/******************************/
/****  16-bit timer 00    ****/
/******************************/
void init_TM00(void)
{
  CRC00  = 0x07;                        /* CR01 compare register */
  PRM001 = SET;
  PRM000 = CLEAR;                       /* Count clock 78.125 kHz */
  ES001  = SET;
  ES000  = SET;                         /* TI000 pin Both Edges */
}

/***************************/
/****  8-bit timer 50  ****/
/***************************/
void init_50(void)
{
  // TCL50 bit 7 6 5 4 3 2 1 0
  //           0 0 0 0 0 |_|_|____ TCL502 TCL501 TCL500
  //                                1      1      1     fx/8196 =   2.44  KHz
  TCL50 = 0x07;
  CR50  = 17;                           /* 7ms */
}

/******************************/
```

```
/****   8-bit timer 51    ****/
/******************************/
void init_51(void)
{
  // TCL51 bit 7 6 5 4 3 2 1 0
  //           0 0 0 0 0 |_|_|____ TCL512 TCL511 TCL501
  //                              1      1      0     fx/8196 =   2.44  KHz
  TCL51 = 0x06;
  CR51  = 12;                         /* 5ms */
}

/*********************/
/****   UART00    ****/
/*********************/
void init_UART00(void)
{
  PM10    = SET;                       // FLMD0
  PM13    = SET;
  PM14    = CLEAR;
  P14     = SET;
  BRGC00  = 0x56;                     /* 115200 */

  // ASIM00 bit 7 6 5 4 3 2 1 0
  //            | | | | | | | 1
  //            | | | | | | |___SL00    0: 1 stop bit
  //            | | | | | |_____CL00    1: 8 data bits
  //            | | | |_|_______PS001 PS000
  //            | | |___________RXE00   1: enable reception
  //            | |_____________TXE00   1: enable transmission
  //            |_______________POWER00
  //ASIM    = 0xE5;
  PS001   = CLEAR;
  PS000   = CLEAR;
  CL00    = SET;                      /* 8-bit */
  SL00    = CLEAR;
  TXE00   = SET;
  RXE00   = SET;
  POWER00 = SET;

  STIF00  = CLEAR;
  SRIF00  = CLEAR;
  SRMK00  = CLEAR;                    /* Enables receive interrupt */
}

/*************************/
/****   Interrupts    ****/
/*************************/
void init_Interrupt (void)
{
  // interrupt definition
  IF0L = 0x00;                        // INT request
  IF0H = 0x00;                        // INT request
  IF1L = 0x00;                        // INT request
  IF1H = 0x00;                        // INT request

                                      // 7 6 5 4 3 2 1 0
  MK0L = 0xE1;                        // 1 1 1 0 0 0 0 1
  MK0H = 0xFD;                        // 1 1 1 1 1 1 0 1
  MK1L = 0x87;                        // 1 0 0 0 0 1 1 1
  MK1H = 0xEF;                        // 1 1 1 0 1 1 1 0

  /* EXT 1,2,3 enabled, INTP0(TOFF) enabled */
  /* TWO enabled */
  /* TX, RX, RXE, TM01 enabled */
  /* ADIF enabled */

  PR0L = 0xFF;                        // INT low priority
  PR0H = 0xFD;                        // INT low priority
  PR1L = 0xFF;                        // INT low priority
  PR1H = 0xFF;                        // INT low priority
}
```

7.5 Main Functions

```
/* mainfunctions.c */
#include "mainfunctions.h"
#include "definitions.h"

// motor drive variables
extern unsigned int motor_rpm, reg_Y, DeltaX;
extern unsigned int Kp, Ki, Kd, PWM_CYCLE, DUTY_CYCLE, setpoint, deadtime;

extern unsigned int DCLevel;
extern unsigned int phase_A;
extern unsigned int steps;
extern unsigned char Motor_Pos;

// control variables
extern unsigned int onoff;
extern unsigned int disconnx;
extern unsigned int OverCurrent;
extern unsigned int Motor_Dir;
extern unsigned int Loop_Sel;
extern unsigned int SecureSel;
extern unsigned int SensorSel, BemfA_sel, BemfB_sel, BemfC_sel;

// sine angle variables
unsigned char phA, phase_B, phase_C;

// sinusoidal signal generating signal
const unsigned char SinTable[] = {0x00, 0x0D, 0x19, 0x25, 0x32, 0x3E, 0x4A,
0x56, 0x61, 0x6D, 0x78, 0x82, 0x8C, 0x96, 0xA0, 0xA9, 0xB2, 0xBA, 0xC1,
0xC8, 0xCF, 0xD5, 0xDA, 0xDF, 0xE3, 0xE7, 0xEA, 0xEC, 0xEE, 0xEF, 0xEF};

/*********************************************************/
/** Reload Inverter with new values -> ISR: TWOexception  **/
/*********************************************************/
#pragma vector=INTTW0UD_vect
interrupt void INTTW0UD_exception (void)
{
  phA = phase_A>>9;

  if (Motor_Dir)
  {
    phase_C = phA + 40;
    phase_B = phA + 80;
  }
  else
  {
    phase_C = phA + 80;
    phase_B = phA + 40;
  }

  // BFCM3_value = PWM Cycle;
  TW0BFCM3 = PWM_CYCLE;

  if (phase_C >= 120) phase_C -=120;
  if (phase_C < 30)
    TW0BFCM2 = DCLevel + ((DUTY_CYCLE * SinTable[phase_C]) >> 8);
  else if (phase_C < 60)
    TW0BFCM2 = DCLevel + ((DUTY_CYCLE * SinTable[60-phase_C]) >> 8);
  else if (phase_C < 90)
    TW0BFCM2 = DCLevel - ((DUTY_CYCLE * SinTable[phase_C-60]) >> 8);
  else
    TW0BFCM2 = DCLevel - ((DUTY_CYCLE * SinTable[120-phase_C]) >> 8);

  if (phase_B >= 120) phase_B -=120;
  if (phase_B < 30)
    TW0BFCM1 = DCLevel + ((DUTY_CYCLE * SinTable[phase_B]) >> 8);
  else if (phase_B < 60)
    TW0BFCM1 = DCLevel + ((DUTY_CYCLE * SinTable[60-phase_B]) >> 8);
```

```
  else if (phase_B < 90)
    TW0BFCM1 = DCLevel - ((DUTY_CYCLE * SinTable[phase_B-60]) >> 8);
  else
    TW0BFCM1 = DCLevel - ((DUTY_CYCLE * SinTable[120-phase_B]) >> 8);

  if (phA < 30)
    TW0BFCM0 = DCLevel + ((DUTY_CYCLE * SinTable[phA]) >> 8);
  else if (phA < 60)
    TW0BFCM0 = DCLevel + ((DUTY_CYCLE * SinTable[60-phA]) >> 8);
  else if (phA < 90)
    TW0BFCM0 = DCLevel - ((DUTY_CYCLE * SinTable[phA-60]) >> 8);
  else
    TW0BFCM0 = DCLevel - ((DUTY_CYCLE * SinTable[120-phA]) >> 8);

  phase_A += steps;
  if (phase_A >= 61440) phase_A -= 61440;
}

/****************************************************/
/** ISR: Over-current Signal                      **/
/** Function: Trigger safety shutdown             **/
/****************************************************/
#pragma vector=INTP0_vect
interrupt void INTP0_exception (void)
{
  if (SecureSel)
  {
   P54 = SET;           // set P54 -> TRIP high (@ 40-pin Ribbon connector)
    OverCurrent = SET;
  }
}

/****************************************************/
/** Communication Variable Initialization         **/
/****************************************************/
void var_init ()
{
  motor_rpm   = 0;    // RPM,
  reg_Y       = 0;    // REGY
  DeltaX      = 0;    // XD,
  Kp          = 50;   // KP,
  Ki          = 1;    // KI,
  PWM_CYCLE   = 500;  // PWMCYCLE,
  DUTY_CYCLE  = 0;    // DUTYCYCLE,
  setpoint    = 0;    // SETPOINT,
  deadtime    = 0;    // DTIME
  onoff       = 0;
  disconnx    = 0;
  OverCurrent = 0;
  Motor_Dir   = 0;
  Loop_Sel    = 0;
  SecureSel   = 1;
}
```

7.6 Hall Sensor Signals Control

```
/* hall.c */
#include "hall.h"
#include "definitions.h"

extern unsigned char newspeed;
extern unsigned char Motor_Pos;
extern unsigned int Motor_Dir;
extern unsigned int phase_A;
extern unsigned int PHASE_A_LEFT[];
extern unsigned int PHASE_A_RIGHT[];
extern unsigned int actual_count;

unsigned int last_hall_time = 0;        // Count value (n-1) from TM00
```

```
unsigned int this_hall_time = 0;        // Count value (n) from TM00

//====================================================================
//  Speed measurement of the motor
//====================================================================
#pragma vector=INTTM01_vect
interrupt void INTTM01_exception (void) //Speed_Meassurment()
{
  this_hall_time  = CR01;
  if (this_hall_time <= last_hall_time)
    actual_count = 0x10000 - last_hall_time + this_hall_time;
  else
    actual_count = this_hall_time - last_hall_time;

  last_hall_time = this_hall_time;
 if ((actual_count > actual_count_min) && (actual_count<actual_count_max))
    newspeed++;
}

#ifdef SENSORRED

/*********************************************/
/****      Ext_ISR Detect Hall A          ****/
/*********************************************/
#pragma vector=INTP1_vect
interrupt void INTP1_exception (void)
{
  Motor_Pos = P0 & 0x0E;
  if (Motor_Dir) phase_A = PHASE_A_RIGHT[Motor_Pos];
  else           phase_A = PHASE_A_LEFT[Motor_Pos];
}

/*********************************************/
/****      Ext_ISR Detect Hall B          ****/
/*********************************************/
#pragma vector=INTP2_vect
interrupt void INTP2_exception (void)
{
  Motor_Pos = P0 & 0x0E;
  if (Motor_Dir) phase_A = PHASE_A_RIGHT[Motor_Pos];
  else           phase_A = PHASE_A_LEFT[Motor_Pos];
}

/*********************************************/
/****      Ext_ISR Detect Hall C          ****/
/*********************************************/
#pragma vector=INTP3_vect
interrupt void INTP3_exception (void)
{
  Motor_Pos = P0 & 0x0E;
  if (Motor_Dir) phase_A = PHASE_A_RIGHT[Motor_Pos];
  else           phase_A = PHASE_A_LEFT[Motor_Pos];
}

#endif
```

7.7 PI Controller

```
/* regulation.c */
#include <io78f0714.h>
#include "regulation.h"
#include "definitions.h"

extern unsigned int setpoint;
extern unsigned int motor_rpm;
extern unsigned int Kp;
extern unsigned int Ki;
extern unsigned int reg_Y;
extern unsigned int DeltaX;
```

```
extern unsigned int PWM_CYCLE;
extern unsigned int DUTY_CYCLE;
extern long Integrator;

#ifdef CLOSE_LOOP

/* local variables */
static long lbuf; // local long type calculation buffer
int ERR;          // speed error
long Yp;          // Y proportional part
long Yi;          // Y integral part
int Ytotal;       // Y result

/* value range limits */
#define Yp_max             9768960  // +9540 * 1024
#define Yp_min            -9768960  // -9540 * 1024
#define Yi_max             9768960  // +9540 * 1024
#define Yi_min            -9768960  // -9540 * 1024
#define Integrator_max     9768960  // +9540 * 1024
#define Integrator_min    -9768960
#define Y_max                 9540
#define Y_min                    0

void PIRegulation(void)
{
  TMIF50 = 0;

  /*      PI Regulator
  Range of values:

  setpoint        =    0%...+100% = (integer) =          0...+9540
  motor_rpm       =    0%...+100% = (integer) =        600...+9540
  ERR             = -100%...+100% = (integer) =  -9540...0...+9540

  Kp              = -100%...+100% = (integer) =  -1024...0...+1024
  Ki              = -100%...+100% = (integer) =  -1024...0...+1024
  */

  // calculate ERR
  ERR = setpoint - motor_rpm;
  DeltaX = ERR;

//  if (ERR < 0) DUTY_CYCLE--;
//  if (ERR > 0) DUTY_CYCLE++;

  // calculate Yp and limit
  // calculate Yp = ERR * Kp;
  lbuf  = ERR;
  lbuf *= Kp;

  // limit Yp
  if      (lbuf > Yp_max) lbuf = Yp_max;
  else if (lbuf < Yp_min) lbuf = Yp_min;

  Yp = lbuf;

  // calculate Yi and limit
  // calculate Yi(t) = ERR * Ki * t;
  lbuf  = ERR;
  lbuf *= Ki;

  // limit Yi
  if      (lbuf > Yi_max) lbuf = Yi_max;
  else if (lbuf < Yi_min) lbuf = Yi_min;

  Integrator = (Integrator + lbuf);

  if      (Integrator > Integrator_max) Integrator = Integrator_max;
  else if (Integrator < Integrator_min) Integrator = Integrator_min;

  Yi = Integrator;

  // calculate Y and limit */
```

```
  lbuf = Yp + Yi;

  Ytotal = lbuf >> 10;    // normalizing by divide by 1024 (2^10)

  Ytotal = reg_Y + Ytotal;

  // limit Y
  if      (Ytotal > Y_max)  Ytotal = Y_max;
  else if (Ytotal < Y_min)  Ytotal = Y_min;

  reg_Y = Ytotal;
  lbuf  = reg_Y;
  //lbuf  = (lbuf * PWM_CYCLE / Y_max) >> 1;
  lbuf  = (lbuf * PWM_CYCLE / 8200) >> 1;

  // limit DUTY_CYCLE
  if      (lbuf > PWM_CYCLE/2)  lbuf = PWM_CYCLE/2;
  else if (lbuf < 0)  lbuf = 0;

  DUTY_CYCLE  = lbuf;
}

#endif
```