Classified Index: TP317.4

U.D.C: 621

Thesis for the Master Degree in Engineering

DEVELOPMENT OF PMSM CONTROL USING MATLAB, IMPLEMENTED WITH A DIGITAL SIGNAL PROCESSOR

Candidate:	Hui Liu
Supervisor:	Prof. Zexiang Li
Associate Supervisor:	Hong Wu Googol Ltd.
Academic Degree Applied for:	Master of Engineering
Specialty:	Control Science and Engineering
Affiliation:	Shenzhen Graduate School
Date of Defence:	December, 2007
Degree-Conferring-Institution:	Harbin Institute of Technology

国内图书分类号：TP317.4

国际图书分类号：621

工学硕士学位论文

基于 MATLAB 和 DSP 的永磁同步电机控制

硕士研究生：刘辉

导　　师：李泽湘教授

副 导 师：吴宏高工

申请学位级别：工学硕士

学科、专业：控制科学与工程

所在单位：深圳研究生院

答辩日期：2007 年 12 月

授予学位单位：哈尔滨工业大学

ABSTRACT

This thesis develops the PMSM control system using MATLAB and implements the system with a DSP. The measurement of current is improved by using current instead of resistance. The code was loaded into DSP through SCI loader and was executed to drive the motor. RTDX was used to collect the experiment data under the MATLAB environment. After analyzing the experiment result, some improvement method was provided to consummate the drive system.

The coordinate transformation and the FOC control strategy are introduced in detail. An improved methodology for embedded software development was used. Graphical block programming allows engineers to simplify algorithm development and avoid duplication of effort in deploying these algorithms to the end hardware.

The PMSM speed control system was built in Simulink with blocks. The code generated for the model was loaded into the memory of DSP. The computation processor of duty cycle was described in detail. And the data collected by RTDX is displayed in MATLAB figure. M-file, C-MEX S-function and TLC file was developed to complete the block function and data collection. The feedback of the current loop was improved. Instead of resistance sensor, a current sensor is used to measure the 3-phase stator current. It improves the precision of the system. The result shows that the system has good performance.

The experiment results and error analyses were laid out in the thesis. The entire design process and deployment to a digital signal processor took less time and effort than the conventional methods. The merit for data analyses and other function can be utilized. And the model has well compatibility and maintainability. More jobs can be done based on this project such as vision system and velocity planning and so on.

Keywords: PMSM control, graphical block programming, DSP

摘 要

本课题采用 Hardware-in-Loop 设计方法在 MATLAB 环境下基于磁场耦合方法设计交流电机驱动器的软件模块，改进了原功率板中电流采样的方法，并将代码下载到 DSP 里实现了对交流电机的控制。开发了 SCI Loader 及利用 RTDX 对实验数据进行采集，并分析了实验结果和对现有实验条件提出对应的改进措施。

本文基于定子坐标系变换和 FOC 控制策略对永磁同步电机建立数学模型。将图形代码转化成为 DSP 可执行代码 COFF 格式,并在 SIMULINK 开发环境下建立整个控制系统的模型，实现了占空比的求解和三相逆变桥的按照特定规律的导通。在 MATLAB 下，将生成的可执行代码通过仿真器下载到 DSP 里。撰写了 C-MEX S-FUNCTIO 和 Target Language File 来完成必需的功能模块，并撰写了 M-file 文件实现 RTDX 的数据采集。 利用 RTDX 实时采集相关参数信息和传递数据，并给出了实验结果和误差分析以及相应的针对改进办法。硬件方面使用 SEEDDSP2812 作为控制板，使用电流传感器取代传统的低精度的测量电阻，对电流环的反馈环节作了改进。

实验数据和结果分析显示本课题基于 MATLB 对永磁同步电机的控制是可行的，相比于传统的设计方法，整个设计过程花费更少的时间，并且可以利用 MATLAB 对数据处理功能和图像处理功能强大的优点，节省一些搭建处理数据等方面的步骤,使得交流电机驱动器的高级算法的转化和验证更加容易，减少由算法到硬件实现所花费的时间，借助本课题所搭建的平台系统，可以有效避免控制算法研究者投入过多的精力在硬件实现上。所建立的模型具有很好的可移植性和可维护性，可以更方便的实现驱动器功能扩展和性能提升。

关键词 永磁同步电机控制；快速模型模块化编程； DSP

ACKNOWLEDGEMENTS

During my graduate studies in HITSZ and practice in GOOGOL Ltd., several persons collaborated directly and indirectly with my research. Without their support it would be impossible for me to finish my work. That is why I wish to dedicate this section to recognize their supports.

I want to start expressing a sincere acknowledgement to my advisor, Professor Li and Mr. Wu Hong in Googol Ltd. They give me the chance to study under their guidance. I received motivation and support from them during my research work. I can’t complete this thesis without help of theirs. I also want to express my gratitude to Dong Liang, Sun Shuai-hua, and Lin Cheng-xi in Googol ltd. for their valuable advice.

At last, but the important, I would like to thank my family, for their unconditional support and love.

CONTENTS

NOMENCLATURE

BLDC	brushless direct current motor
CCS	code composer studio
DM	data memory
DSP	digital signal processor
FOC	field oriented control
GUI	graphical user interface
JTAG	joint test action group
PM	program memory
PI	proportion integral
PMSM	permanent magnet synchronous motor
PWM	pulse width modulator
QEP	quartered encoder pulse
RPM	resolution per minute
RTDX	real-time data exchange
RTW	real-time workshop
TLC	target language file

LIST OF TABLES

Table **Page**

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

With in the framework of factory automation and machine manufacture, increasingly powerful and more flexible systems are needed in industrial drive engineering. From these years, China is recognized as the world factory- so many kinds of machines were signed “made in China”. But the development of drives machine made in China still drop behind the Japan and Europe country.

1.1 Background

Everyone recognizes the vital role played by electrical motors in the development of industrial systems. Fig1-1 shows the typical motion control application [1]. Usually, the manufacture machine system is composed of position controller, servo system, sensor and the mechanism. We can usually call the former two parts as drive controller. Drive controllers are actuators that operate without feedback and automatic control equipment having a speed, current or position feedback.

Being the most important part in the system, the choice of electrical motor depends on application type and the performance requirement. After the motor is selected, a particular power electronic converter is specified. All AC and DC drives use power semiconductor devices to convert and the devices operate in the switching mod. The common power switch elements are MOSFET, IGBT and thyristor. And then particular algorithms of power converter, torque control and motion control were end up as selected. Motion controller includes the closed loops (usually position loop and velocity loop) that send command to the drive. The drive includes the current loop or torque loop. Also the torque loop and the other closed loop can be in one controller with dealing them in one processor. The sensor and interface is also important for the performance and function of the system.

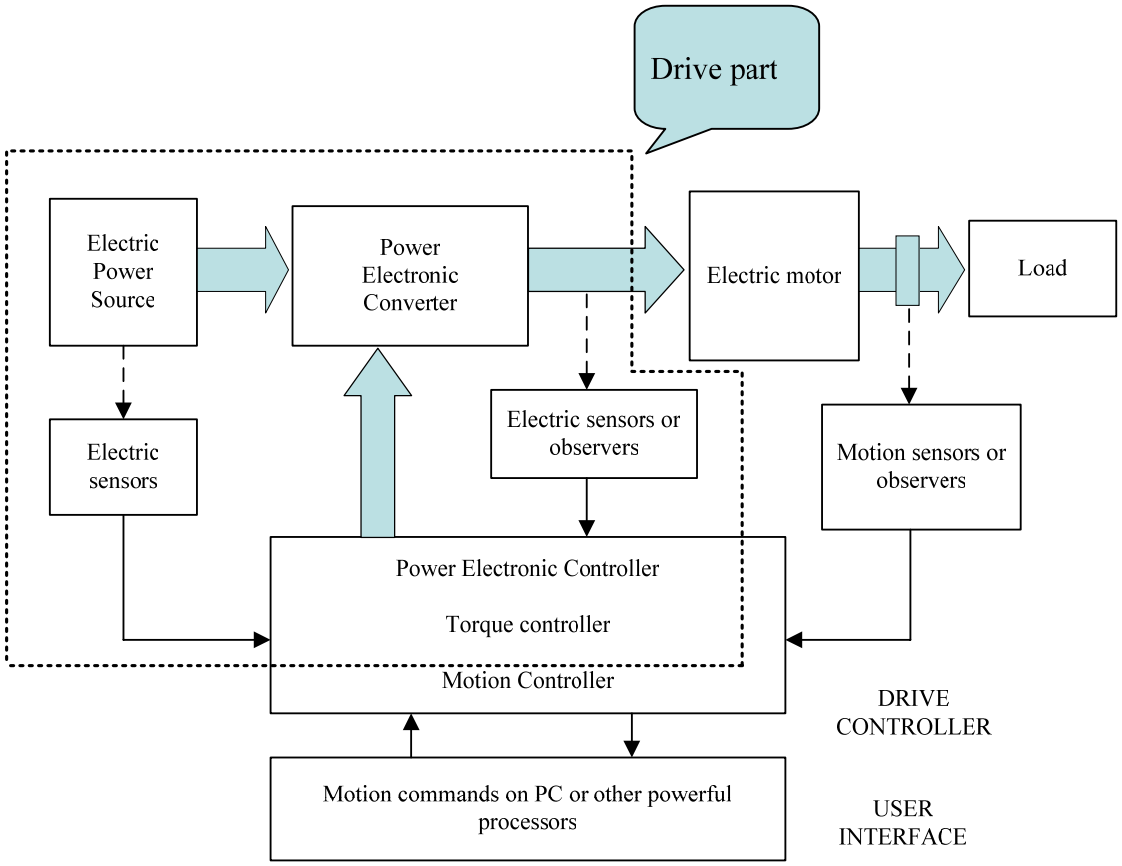


Figure 1.1 Typical structure of motion control system.

Servo motors are classified into DC servo motor, AC servo motor, and stepping motors. There are two varieties of AC servo motors; synchronous motor and induction type servo motor.

Since the DC motor was invented by W.V. Siemens in the late 1800s, brush DC motors have been the most prominent variable speed technology. The AC induction motor was invented by Nicolas Tesla in 1924.In 1962 two engineers T.G. Wilson and P.H. Trickey published a paper in which they described a “brushless DC motor” [2]. It was not a practical technology until the late 1980 for the lag of the magnet and power switching device technology. But now, we know that brushless motor is the most popular motor in the modern factories.

What’s the reason machinery manufacturer and users choose brushless over the time-proven DC drive or the now widely accepted vector induction motor system?

Compare with brush DC motor and induction motor, the permanent magnet motor has the following advantages.

• Performance: Brushless systems offer the highest available dynamic accuracy of any of the three major types of drive systems.

• Size: Brushless motors are the smallest available motors for a given power rating.

• Efficiency: The brushless motor is the most efficient motor technology available today for industry application. Formal tests were conducted by Ontario Hydro(Canada) in a before/after comparison of a machine conversion from brush DC to brushless DC in which efficiency improved 13% minimal and a whopping 78% at 25% speed.

Table 1.1 The comparison between AC & DC motors [2]

	AC SERVO	DC SERVO
Life	<Nearing life> 20,000 h or up	<Brush life> Normally, 3,000 to 5,000h varies considerably due to load and environmental conditions
Maintenance	<not required> No mechanical contact(no brushes, communicators)	<required> Required periodical check and replacement of brushes
Sound noise	<Quiet>	<Noisy> Due to brush contacting noise
Electrical noise	<None> No noise as no brushes	<Exist> Noise occurs due to actuation of brushes
Efficiency	<Excellent> Good cooling efficiency as heat radiates from stator	<Good> Rectification loss occurs Bad cooling efficiency due to rotor heat
Against Overload	<Good> Large thermal time constant High speed and large torque	<Medium> Small thermal time constant Limited current due to brush flashover
Response Characteristics	<Very quick > Large power rate	<Quick> Small power rate
Cleanness	<Good> Clean as no brush powder occurs	<Bad> Brush powder occurs

The drive system receive the signal (may be analog or digital signal) from position controller (or motion controller) , and then deal with it to use the information to control the power equipment and to make the motor rotor round so the drive system is very important to improve the whole system's performance.

About 80-90% of the controllers used in the industry application are PID controller [3]. It doesn't need the precise mathematic model and easy programmed using computer language. People have done a lot of attempts to avoid the influence by the disturbance (certain or uncertain disturbance). J. Solsona and M. I. Valla provided the nonlinear method in their paper [5]. J. Zhou and Y. wang used the adaptive method in their paper [6]. F-J Lin and R-J wai designed an on-line trained fuzzy neural network controller [33]. Shun-zhi Xiao used the Fuzzy-PI control based on the adaptive genetic algorithm in the paper [7]. A. A. Hassan and M. Azzam applied the Linear Quadretic Gaussian algorithm to realize the robust control of PMSM in their paper [31].

The advantage of those strategies is that with robust character, good performance with precise mathematics model. The disadvantage is that they are all not easily or impossible realized on the DSP or other processor. Because of complicated structure and the slow response, those algorithms usually were only tested in the laboratory.

1.2 Objective and method

Normally, people develop the motor control system based on the principle and algorithm with mathematics expression. Then they'll do some simulation with some special software, such as MATLAB, LABVIEW and so on. Based on the simulation result, they change and improve their control algorithm or parameters. Also they use these tools to treat the control method with the ideal condition. After getting the satisfying result, then they'll realize their algorithm by hardware with program language. At the last step, they can use the oscillograph to observe the waves cape and then change the control algorithm based on the experiment result, still need to do some simulation, still to do program. So they have to repeat this process until to get the right export. This is a complicate process.

With the tool provided by MATLAB and TI, people can reduce some steps to do these jobs. They can do the hardware-loop test of the motor control system. Some repeat steps can be avoided in the development process. So we use the Embedded target for TIC2000 (a toolbox integrated in MATLAB) to complete our project. This method also can be called Hardware-In-Loops design methods or model-based design method. The model is used to define specifications, evaluate design and system performance, automatically generate code, perform hardware-in-the-loop testing, and create a software-based test harness for testing production hardware. This approach can substantially reduce development time by rapidly leading to complete and functional proof-of-concept designs and enabling rapid design iterations and parameter optimization through a unified design, simulation, and test environment [5].

The use of model-based design for motor control applications has been aided by the introduction of blocksets that include pre-configure blocks to handle all elements of vector control systems, such as Park and Clarke transforms, PWMs, speed estimators ,flux estimators and others. These new tools enable designers to quickly build a graphical model using pre-built blocks representing primitives and advanced algorithms, incorporating their own c-code only when required. All of the integration between the various blocks and peripherals is carried out automatically. The following figure illustrates the develop method.

For our application, the processor we choose is TI TMS320LF2812 serial, so the toolbox we used for rapid code generation is embedded target for TIC2000 toolbox.

In the Simulink environment, we developed our model. The TIC2000 provided some basic blocks such as AD, PWM block. But we need develop some other blocks to make the model built successfully. The block development process can be done with the system function or so-called S-function that can be written in C programming language (C-MEX S-function). S-function defines number of the inputs, numbers of outputs, algorithm for calculation of the block output value, sample time, etc for the custom simulink block. These information can be linked by TLC file. RTW (Real-time workshop) also generates makefile (model.mk) from the system template makefile. After that, the target language compiler is invoked into the code generation process and it converts

simulink model to the C code. Then it is possible for the MATLAB to call the communication function. In the MATLAB toolbox, there's a tool named Link or CCS (code composer studio). With this tool, we can use MATLAB function to communicate with CCS and with information stored in memory and registers on a target. With the link we can transfer information to and from CCS and with the embedded objects we get information about data and functions stored in our signal processor memory and registers, as well as information about functions in our project.

After the file is transformed into COFF, it needs to be executed in the DSP. We provide two methods to load the executed file into the memory of DSP: using SCI and using RTDX.

Using emulator (based on JTAG), we can load the COFF file into the DSP memory. With the RTDX technology, we can easily observe the result and tune the parameter. RTDX can send data from the DSP to the host computer while the DSP is running. A few m-files are used to operate the DSP, load different project files, configure RTDX channels, and manage the exchange of information between DSP and host PC. Originally, the intention was to have the DSP send speed data, current data and controller parameter, in real time. And the result can be shown in the figure and GUIs.

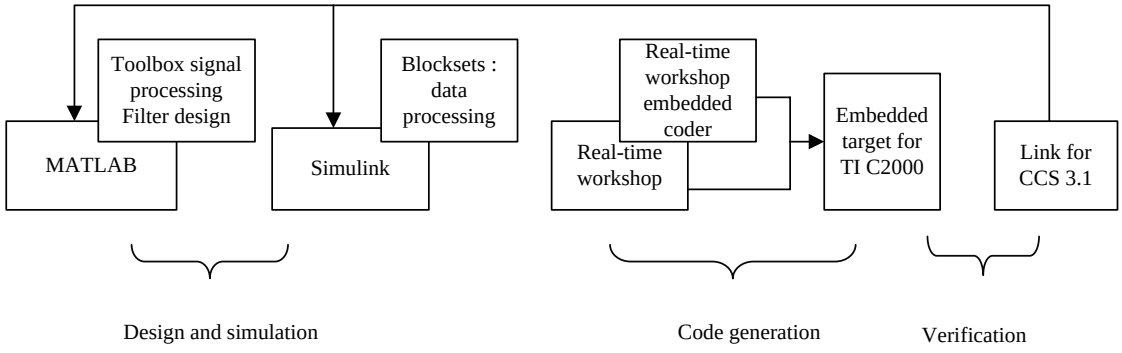


Figure1.2 Auto-code generation processor

1.3 System Structure

The system can be divided into software and hardware part. The model was developed under Simulink, and then the code was generated automatically in RTW. Through emulator, the code was loaded into the memory of DSP. the controller board and the power board combines the hardware part. TMS320LF2812 is chosen as the main processor. The controller board we used is SEED2812 DSP starter board. It's mainly used for the people who want to learn DSP. We used it to generate the right PWM and receive AD. After the model was completed developed under SIMULIUNK environment, the auto generated code can be loaded into the memory of DSP. Then the power board can transform the low voltage into high voltage to drive the motor.

CHAPTER 2

MODELING AND SIMULATION

The PMSM (Permanent magnet synchronous motor) has a wound 3-phase stator, and magnet rotor. In generally, as its name revealed, the rotor is made by permanent magnet material. Usually there're sense equipment were assembled with the rotor to detect the rotor angle and the speed can be computed based on the information provided by the sensor, in general, the equipment is optical encoder, hall , or transformer. These sensors were fixed on the shaft of the motor, and some part of them round with the shaft. Then the signal from the sensor (analog signal or digital signal) was sent to the processor. We called this loop as velocity feedback loop.

Two configurations of permanent magnet synchronous motor are considered: sinusoidal type brushless motor and trapezoidal type wave brushless wave motor (BLDC). We usually called the former as AC servo motor. In our experiment, we used sinusoidal type brushless motor.

2.1 FOC Strategy

The sinusoidal voltage waveform applied to this motor is created by using the space vector modulation technique. The field oriented control (FOC） algorithm will enable real-time control of torque and rotation speed. As this control method is accurate in every mode of operation. No torque ripple appears when driving this sinusoidal BEMF motor with sinusoidal currents.

Operation of a brushless PM motor relies on the conversion of electrical energy to magnetic energy and then from magnetic energy to mechanical energy. It is possible to generate a magnetic rotating field by applying sinusoidal voltages to the 3 stator phases of a 3 phase motor. A resulting sinusoidal current flows in the coils and generates the rotating stator flux.

The rotation of the rotor shaft is then created by attraction of the permanent rotor flux with the stator flux. Usually the motor used in the industry application has more than one pole. So for the measurement equipment, the angle is mechanical angle and the angle what we used in the coordinate transformation is electrical angle.

Clarke and Park transforms are used in the high performance drive architectures related to permanent magnet synchronous motors.

The Clarke transformation will be used to transform the abc0 axis coordinate to d-q (consider d axis as real axis and q axis as imaginary axis) coordinate. The park transform can be used to realize the transformation of stationary to rotary reference frame [9]. Then Id will be kept zero in order to achieve the field orientation condition. So the permanent magnet flux linkage is aligned to d-axis, and the stator current is kept along the d-axis direction. Because the permanent magnet flux is constant, the electromagnetic torque is linear to q-axis current. So we realize the decoupling between the electric and magnet field. Finally we can get the required torque to control the current through stator.

2.2 Coordinate transformation

The following assumptions are made in the derivation [8]:

(1) Saturation is neglected although it can be taken into account by parameters changes.

(2) The induced electromotive force is sinusoid.

(3) There are no field current dynamics.

(4) Eddy currents and hysteresis losses are neglected.

(5) There is no cage on the rotor.

Key point 1: Clarke transforms

Generally speaking, for the stator, we build the coordinates as Fig 2-1; a-b-c is the physical 3-phase stator input. And α-β coordinates is the vertical coordinate.

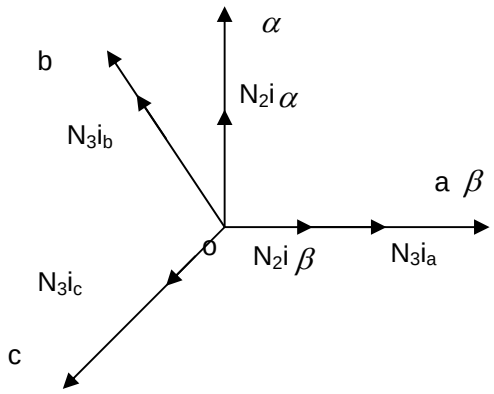


Figure 2.1 The CLARK transformation.

Because of the vector transform doesn't change the magnet flux, and then we get:

$$N_2 i\alpha = N_3 i_a + N_3 i_B \cos\frac{2}{3}\pi + N_3 i_c \cos(-\frac{2}{3}\pi) \tag{2.1}$$

$$N_2 i_\beta = 0 + N_3 i_b \sin\frac{2}{3}\pi + N_3 i_c \sin(-\frac{2}{3}\pi) \tag{2.2}$$

$$i_0 = N_3 k(i_a + i_b + i_c) \tag{2.3}$$

In the matrix form, in fact i_0 equal zero, in other word, there doesn't exist i_0.

$$\begin{pmatrix} i_\alpha \\ i_\beta \\ i_0 \end{pmatrix} = \frac{N_3}{N_2} \begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ k & k & k \end{pmatrix} \begin{pmatrix} i_a \\ i_b \\ i_c \end{pmatrix} \tag{2.4}$$

For convenience, we use the character c express the transform matrix.

$$C = \frac{2N_2}{3N_3}\begin{pmatrix} 1 & 0 & \frac{1}{2K} \\ -\frac{1}{2} & \frac{\sqrt{3}}{2} & \frac{1}{2K} \\ -\frac{1}{2} & -\frac{\sqrt{3}}{2} & \frac{1}{2K} \end{pmatrix} \tag{2.5}$$

Following the principle of power conservation, we know:

$$C^{-1} = C^{T} \tag{2.6}$$

So:

$$\frac{N_2}{N_3} = \sqrt{\frac{3}{2}}, \quad K = \frac{1}{\sqrt{2}} \tag{2.7}$$

Then

$$\begin{pmatrix} i_\alpha \\ i_\beta \\ i_0 \end{pmatrix} = \sqrt{\frac{2}{3}}\begin{pmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} & \sqrt{\frac{1}{2}} \end{pmatrix}\begin{pmatrix} i_a \\ i_b \\ i_c \end{pmatrix} \tag{2.8}$$

Because

$$i_a + i_b + i_c = 0 \tag{2.9}$$

So we can get rid of i_C, the transform can be changed into 2-2 coordinate transform.

$$\begin{pmatrix} i_\alpha \\ i_\beta \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ \frac{1}{\sqrt{3}} & \frac{2}{\sqrt{3}} \end{pmatrix}\begin{pmatrix} i_a \\ i_b \end{pmatrix} \tag{2.10}$$

Key point 2: Park transforms:

Fixed coordinate transform to the rotary coordinate,

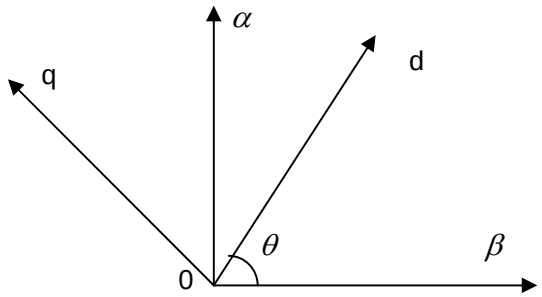


Figure 2.2 Park transformation

$$\begin{pmatrix} i_d \\ i_q \end{pmatrix} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} i_\beta \\ i_\alpha \end{pmatrix} \tag{2.11}$$

So

$$\begin{pmatrix} i_d \\ i_q \end{pmatrix} = \begin{pmatrix} \cos\theta + \frac{\sqrt{3}}{3}\sin\theta & \frac{2\sqrt{3}}{3}\sin\theta \\ -\sin\theta + \frac{\sqrt{3}}{3}\cos\theta & \frac{2\sqrt{3}}{3}\cos\theta \end{pmatrix} \begin{pmatrix} i_a \\ i_b \end{pmatrix} \tag{2.12}$$

2.3 Motor Dynamics Equation

After the coordinate transformation, the stator coordinate was transformed into d-q coordinate, which is similar to the DC motor. The equivalent electric circuit of the permanent magnet synchronous motor is just like the DC motor circuit. The dynamic model of PMSM in d-q coordinate can be described as following:

$$u_d = (R_d i_d + \frac{d}{dt} L_d i_d) - p\omega_r L_q i_q \tag{2.13}$$

$$u_q = R_q i_q + \frac{d}{dt} L_q i_q + p\omega_r L_d i_d + \phi \tag{2.14}$$

$$\varphi_q = L_q i_q \tag{2.15}$$

$$\varphi_d = L_d i_d + \phi \tag{2.16}$$

$$\omega_e = \omega_r p \tag{2.17}$$

Eq (2.13) and (2.14) result from applying Kirchoff's law for voltage drops around the equivalent circuit. The mechanical motion can be expressed as:

$$Te = K_f (L_d - L_q) i_q i_d \tag{2.18}$$

And the electromagnet torque equation derived from the motor is:

$$Te = J\frac{d\omega_r}{dt} + B\omega_r + T_L \tag{2.19}$$

From the expression function, we can get that this is still a nonlinear system. For nonlinear system, it's not easy to analysis and the model need to be linearization. So we can use the state feedback to cancel the nonlinearly part.

To decouple this system, we can use the following expression of Ud, Uq to replace uq and ud:

$$Ud = ud + p\omega_r L_q i_q \tag{2.20}$$

$$Uq = uq - p\omega_r L_d i_d - \phi \tag{2.21}$$

So now the system is coupled and also a linearly system:

$$Ud = R_d i_d + \frac{d}{dt} L_d i_d \tag{2.22}$$

$$Uq = R_q i_q + \frac{d}{dt} L_q i_q \tag{2.23}$$

$$\begin{bmatrix} \frac{d}{dt} i_d \\ \frac{d}{dt} i_q \end{bmatrix} = \begin{pmatrix} -\frac{R_d}{L_d} & 0 \\ 0 & -\frac{R_q}{L_q} \end{pmatrix} \begin{bmatrix} i_d \\ i_q \end{bmatrix} + \begin{bmatrix} \frac{U_d}{L_d} \\ \frac{U_q}{L_q} \end{bmatrix} \tag{2.24}$$

Laplace transforming Eqs provides the basis for constructing the block diagram. So, we can get the block like this:

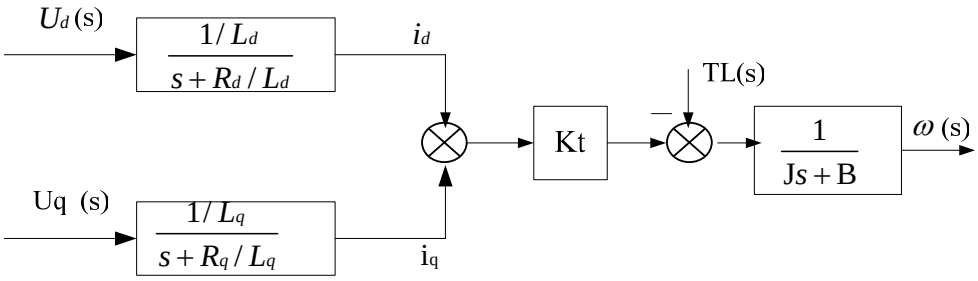


Figure 2.5 d-q axis model

Where ud is the direct voltage, uq is the quadrature voltage, id is the direct current, iq is the quadrature current, ω_e is the electrical angle velocity, ω_r is the mechanical angle velocity. p is the pole pair number. In this coordinate system, the transformed currents id and iq vary approximately at the mechanical frequency of the motor. These variables typically have bandwidth in the range of 0-100 HZ compared with 0-5kHz bandwidth for ud, uq, id, iq. And in this block diagram, we haven't added the controller and the feedback part.

Usually, for different Ud rang, there are different control methods: Ud=0 to make control simply, Ud <0 to improve the range of velocity. In this condition, we make Ud=0,.

2.4 Simulation and parameter tuning

Mechanical time constant: $\tau_s = 23.3ms$, Back-EMF constant $K_b = 2.325\ mV/rpm$, rotor inertia J=$7.82\times10^{-7}\ kg\cdot m^2$, the phase resistance R=5.25*ohm*, the phase inductance L=0.46*mH*, the torque constant Kt =25*mNm/A*

The open-loop transfer function can be expressed as:

$$G_\omega(s) = \frac{\omega(s)}{U(s)}|_{U(s)=0} = \frac{\left(\frac{1}{Js+B}\right)Kt\left(\frac{1}{Js+B}\right)}{1+Kb\left(\frac{1}{L_qs+R}\right)Kt\left(\frac{1}{Js+B}\right)}$$
$$= \frac{Kt}{(L_qs+R)(Js+B)+KtKb} \tag{2.25}$$

$$G_T(s) = -\frac{\omega(s)}{T_L(s)}|_{U(s)=0} = -\frac{\frac{1}{Js+B}}{1+Kb\left(\frac{1}{L_qs+R}\right)Kt\left(\frac{1}{Js+B}\right)} \tag{2.26}$$
$$= -\frac{L_qs+R}{(L_qs+R)(Js+B)+KtKb}$$

Then, by superposition, a property of linear systems, the motor response to voltage and load torque is

$$\mathrm{G(s)} = G_\omega(s)\mathrm{U(s)} + G_T(s)T_L(s) \tag{2.27}$$

The motor is modeled as a second order system with characteristic polynomial:

$$\Delta(s) = (L_qs+R)(Js+B)+KtKb$$

We know the standard form of the second order system transfer function

$$\mathrm{G_s(s)} = \frac{K\omega^2}{S^2+2\xi\omega s+\omega^2} \tag{2.28}$$

Solving for the steady-state gain (from voltage to angular speed) K, the n ature frequency ω and the damping ratio ξ in terms of the motor parameters re sults in

$$K = \frac{K_t}{BR+K_bK_t} \tag{2.30}$$

$$\omega = \left(\frac{BR+KbKt}{JL_q}\right)^{1/2} \tag{2.31}$$

$$\text{And } \xi = \frac{(BL + JR)}{2\left[JL_q(BR + KbKt)\right]^{1/2}} \tag{2.32}$$

The characteristic roots [poles of $G_\omega(s)$] are obtained by solving $\Delta(s) = 0$.

$$s_1, s_2 = -\xi\omega \pm \sqrt{\xi^2 - 1}\omega \tag{2.33}$$

The transfer function $G_\omega(s)$ is expressible in terms of the characteristic roots and motor time constants by

$$G_\omega(s) = \frac{K\omega^2}{(s - s_1)(s - s_2)} = \frac{K\omega^2\tau_1\tau_2}{(\tau_1 s+1)(\tau_2 s+1)} \tag{2.34}$$

Where:

$$\tau_1 = \frac{1}{s_1},\ \tau_2 = \frac{1}{s_2} \tag{2.35}$$

So comparing the Eq (2.25) with Eq (2.34), we can get the following result:

$$\tau_1 = \frac{Lq}{R},\quad \tau_2 = \frac{JR}{BR + KbKt} \tag{2.36}$$

And τ_1 is the electrical time constant, τ_2 is the mechanical time constant, here

$$\tau_1 = 87.6\text{e-}6\text{s},\ \tau_2 = 23.3\text{e-}3\text{s}$$

$$G_0(s) = \frac{1.72\times10^3}{(23.3\times10^{-3}s + 1)(87.6\times10^{-6}s + 1)} \tag{2.37}$$

Because $\tau_1\tau_2$ is so small, so it just like a first-order system.

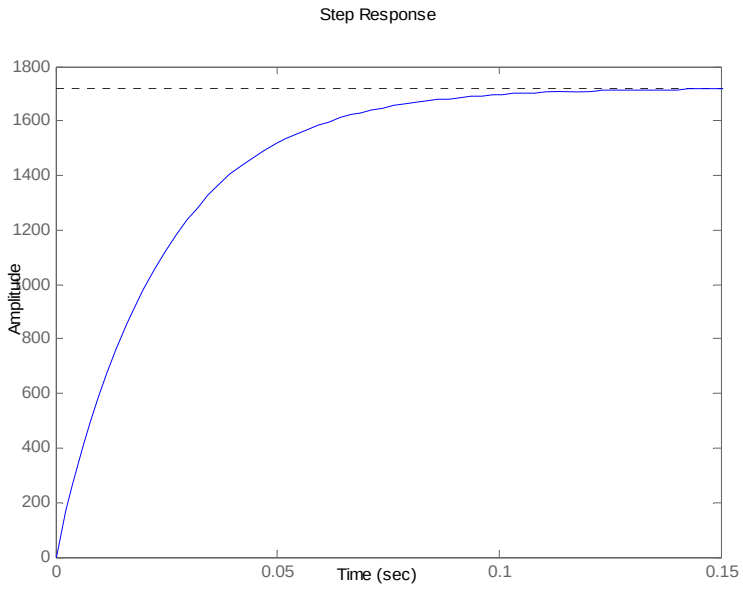


Figure 2.7 Open loop step response

A conventional three term P-I-D controller will be used to control the motor speed. The continuous form is described by

$$G(s) = \frac{G_c(s)G_0(s)}{1+G_c(s)G_0(s)} = \frac{1732(K_p s + K_I)}{(23.3e-3)s^2 + (1732K_p + 1)s + 1732K_I} \qquad (2.38)$$

The integral component makes the closed-loop system assures zero offset (steady-state error) for a step change in commanded motor speed. The derivative component is available id needed to improve stability by adding damping to the system. Then the transfer function can be expressed as

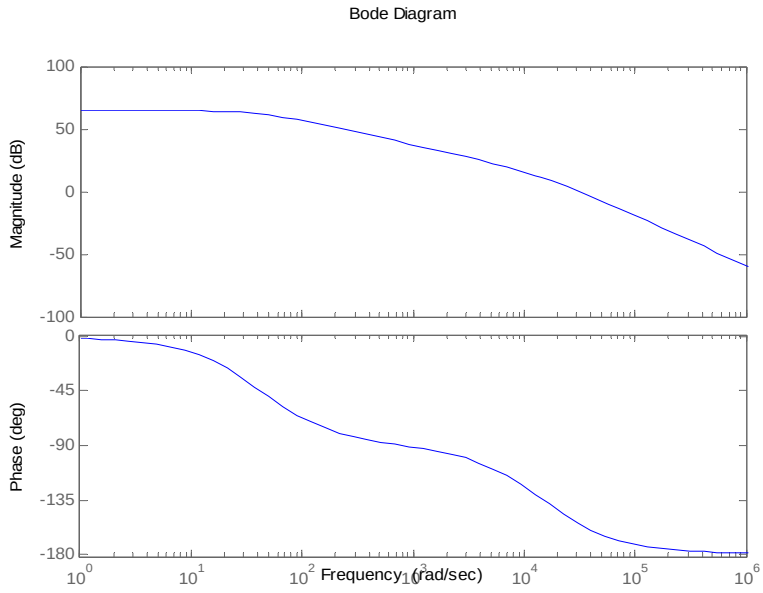


Figure 2.8 Open loop BODE diagram

$$G(s)=\frac{G_c(s)G_0(s)}{1+G_c(s)G_0(s)}=\frac{1732(K_ps+K_I)}{(23.3e-3)s^2+(1732K_p+1)s+1732K_I} \tag{2.39}$$

Here, we use the PI controller, so let K_D =0. Comparing the character equation with the standard character equation, we know

$$\Delta(s)=(23.3e-3)s^2+(1732K_p+1)s+1732K_I=0 \tag{2.40}$$

This function can be transferred into the following expression:

$$1+K_pG(s)H(s)=1+K_p[\frac{1732s}{(23.3\cdot10^{-3}s+1)s+s+1732K_I}]=0 \tag{2.41}$$

Then we can draw the root locus of $G(s)H(s)$ choose K_I =0.01,

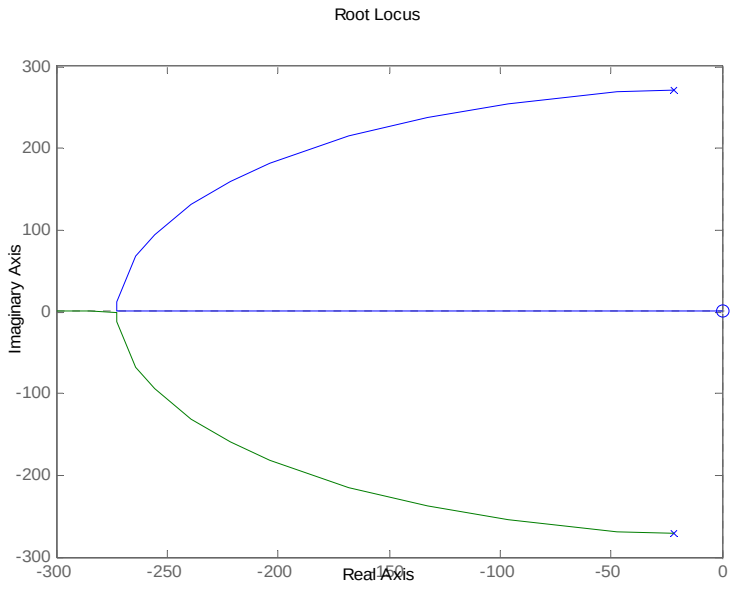


Figure 2.9 Root locus when K_I =0.01

Here, we can compute $\omega = \sqrt{\frac{1732K_I}{0.0233}}$, $\xi = \frac{1732K_P + 1}{2(40K_I)^{\frac{1}{2}}}$

For the common design, to get the quick response, ξ is usually chase as 0.707, and we get K_P =0.0046, K_I =0.01, then the system can be described as

$$G(s) = \frac{7.94s + 1732}{(23.3e-3)s^2 + 8.94s + 1732} \tag{2.42}$$

Step response is like the following figure

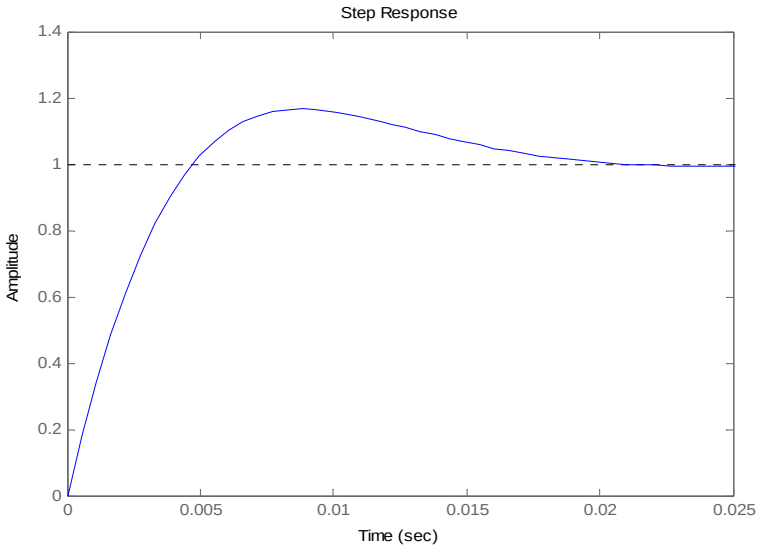


Fig 2-10 step response K_P =0.0046, K_I =0.01

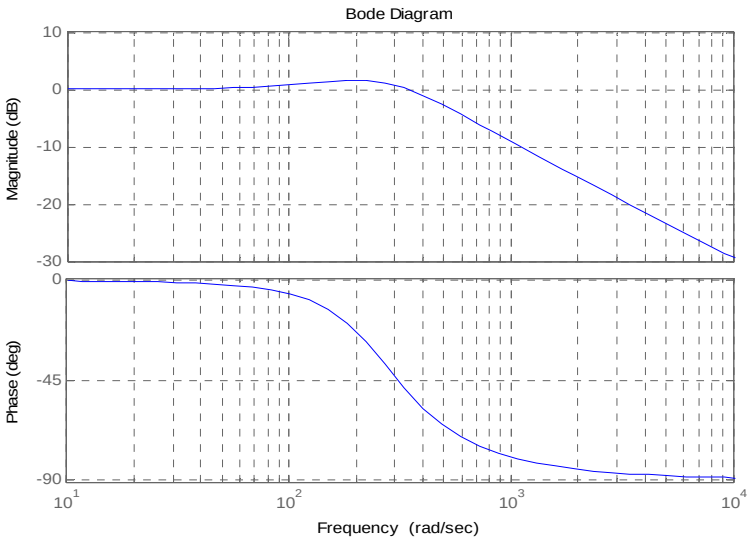


Figure 2.11 BODE figure plot K_P =0.0046, K_I =0.01

Comparing between the system without controller and with controller, we can find that after adding the controller the system had good response: less raise time and error. But this just the simulation, we can't get the precise model for the motor. In the experiment, the controller parameter may be different.

CHAPTER 3

TOOLS USED AND HARDWARE CONFIGURATION

It is a very common challenge for engineers to develop and test software to carry out various tasks on embedded hardware. This can often be a very tedious process. Engineers usually plan out the algorithm graphically or with pseudo code. They must then translate that into high level language, which involves writing many lines of code. In the present time block programming of the microcontrollers and DSP's is more and more important and it plays a key role in the time to market of the new products. Block programming is much closer to the engineering and the scientists that other types of programming. Block programming enables that they can force their energy to the system design and not into the code development process. The code generation process must be quick and transparent to the system designer. If so, they can quickly test and validate designed algorithms on the intended target.

Though this has been the common methodology for many yeas, there is an emerging technique that involves developing algorithms with graphical tools. This improves development time, decrease programming complexity, and facilitates sharing of the design with coworkers. The Math Works provide some tools to solve this problem.

The model was developed using MATLAB. And we build the model using blocks provided by TIC2000 and other blocks developed by ourselves. This process makes it suitable for simulation, data manipulation and hardware targeting, so somebody called this method as hardware-in-loop method. Then the closed-loop system was designed and simulated with MATLAB. They were then implemented using a TMS320F2812 digital signal processor (DSP) from Texas Instruments. MATLAB has special tools included that allow it to compile code for this type DSP, program it, and transfer data to and from it in real-time.

3.1 Tools used to generate the code

Several tools or technology were used to complete the process of auto code generation. They are MATLAB, simulink, embedded target for TIC2000 and RTDX. The following is the basic introduction of them.

In fact, the model was built directly in simulink. Simulink is a software package for modeling, simulating, and analyzing dynamic systems. It supports liner and nonlinear systems, modeled in continuous time, sampled time or a hybrid of the two. We can easily build models from scratch, or take an existing model and added to it. The model we used should be built in simulink. And then we can change it to the code. The following figure is the basic PWM generation function. We just put the existing model PWM 1 and other elements from the simulink library browser, and then use the line to connect them. So simulink is a very convenient environment to develop our model. Fig4-1 shows a simple model built in simulink.

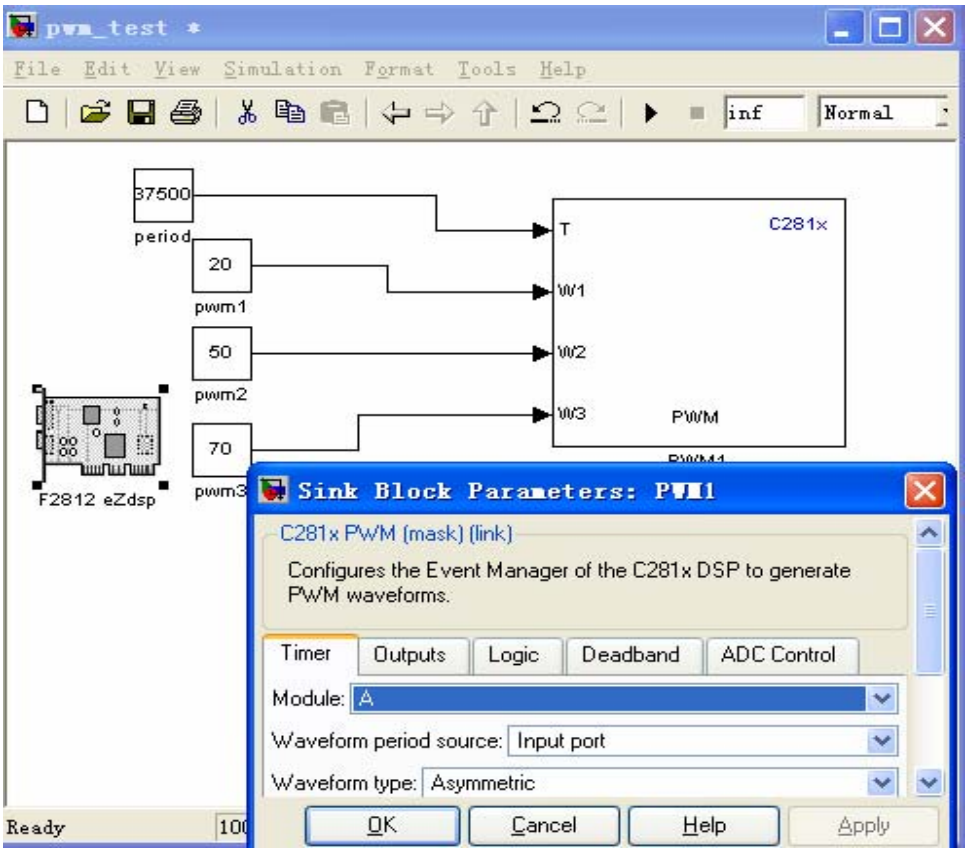


Figure 3.1 An example model using TIC2000 toolbox

EMBEDDED TARGET FOR TI C2000

The embedded target for TI C2000 DSP integrates MATLAB and SIMULINK with Texas Instrument C2000 DSP processors. It lets engineer perform automatic code generation, prototyping, and embedded system development on TI C2000 processors. There are custom blocks that can be added to Simulink model to target the C2000DSP's peripherals. Figure shows some examples of these blocks. When these blocks are added to a model, the necessary code will be generated by MATLAB.

RTDX: Real-Time Data Exchange

Real time data exchange provides real-time, continuous visibility into the way target applications operate in the real word. RTDX allows system developers to transfer data between a host and target devices without interfering with the target applications. The data can be analyzed and visualized on the host using any host client. This shortens time by giving you a realistic representation o the way your system actually operates.

RTDX consists of both target and host components. A small RTDX software library runs on the target application. The target application makes functions calls to this library's API in order to pass data to or from the host platform via a JTAG interface. Data transfer to the host occurs in real-time while the target application is running.

On the host platform, an RTDX Host library operates in conjunction with code composer. Displays and analysis tools communicate with RTDX via an easy-to-use COM API to obtain the target data and/or to send data to the target application. Designers may use their choice of standard software display packages. With this tool, we can read data and write data more directly. Instead of focusing on obtaining the data, we can concentrate on designing the display to visualize the data in the most meaningful way.

3.2 Hardware configuration

The hardware was composed by three parts: controller board, power board, motor and encoder. The controller board is just a common DSP starter board: SEED-DSK2812. It just prepare for those people who want to learn DSP. The power board is used to transfer the command from the DSP to motor. In general, the controller provider the right

PWM, and this PWM was sent to the power transfer equipment, such as MOSFET and IGBT. These equipments are just switch element. And the voltage can be signed the wave what the motor need from them. The motor we used is produced by PITTMAN Co. The serial is 3441E023-R1 brushless motor. The following table is the motor parameters provided by the manufacturer:

Table 3.1 Motor parameter

Phase resistance	5.25*ohm*
Phase inductance	0.46*mH*
Back-EMF constant	2.62V/1000*rpm*
Torque constant	25*mNm/A*
Dynamic parameter	
Rate voltage	19*V*
Max. voltage	36*V*
No-load current	72*mA*
Max.continuous stall current	3.64*A*
Max. continuous torque	29*mNm*
Max. recommended speed	8000*rpm*
Peak current	1.3*A*
Peak torque	94*mNm*
Mechanical parameters	
Rotor Interti	9.9 kgm^2 .10^{-7}
Mechanical time constant	8*ms*

3.2.1 Controller board

The controller board has the following features:

- TMS320F2812 Digital Signal processor
- 150MIPS operating speed
- 18K words on-chip RAM
- 128K words on-chip Flash memory
- 64K words off-chip SRAM memory
- 30MHZ clock
- 2 Expansion Connectors (analog, I/O)
- Onboard IEEE 1149.1 JTAG controller
- 5-volt only operation with supplied AC adapter
- On board IEEE1149.1 JTAG emulation connector
- Three- phase synchronous pulse width modulator
- Symmetrical output pulses；66ns time resolution
- RS232 full duplex interface with fixed Baud Rate

The following figure is the basic information of the SEED 2812 starter board. This board can satisfy our need.

The controller collected the feedback current information and sampled it. Then the processor compares it with the given or computed current. The result was sent to the controller. After the adjusting of the controller, the command then transform into the duty cycle, then to generate the right PWM pulse. This is the process to deal with current loop for the controller board. And the process for the velocity loop is similar to this. The only difference is that the sensor is different. For the velocity loop, the sensor is encoder. In experiment the encoder is fixed on the draft of the motor. So we also can call this type as half-loop. The reason is that for the system the sensor can’t get the information of the moving body, it just got the information of the motor.

Figure 3.2 SEED2812 Starter board

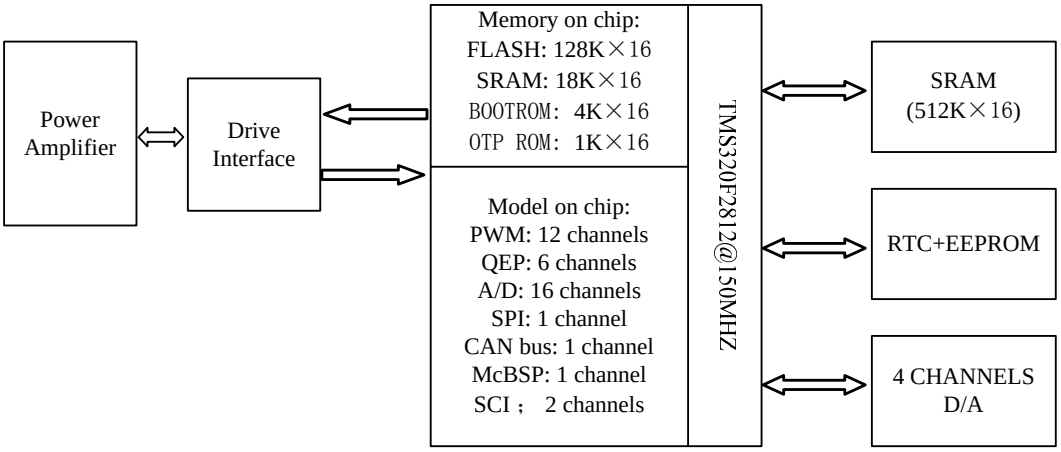


Figure 3.3 Basic configuration of the SEED DSP2812 start board

3.2.2 POWER BOARD

The power board was made by Googol Ltd. It's a 24 supplied board. This board can transform the PWM low voltage to the high voltage which is needed to drive the motor. Of course the transform need to be islanded by opto-coupler or other kinds of similar elements. The PWM low voltage signal was received by the drive chip-IR2132. It's a kind of 3-phase bridge drive chip. It receives the control or switch signal, and then conversed it into the type to drive the gate power element. Here, we used the MOSFET, a

kind of power element. The next Fig is designed by the engineer of Googol to control the BLDC motor.

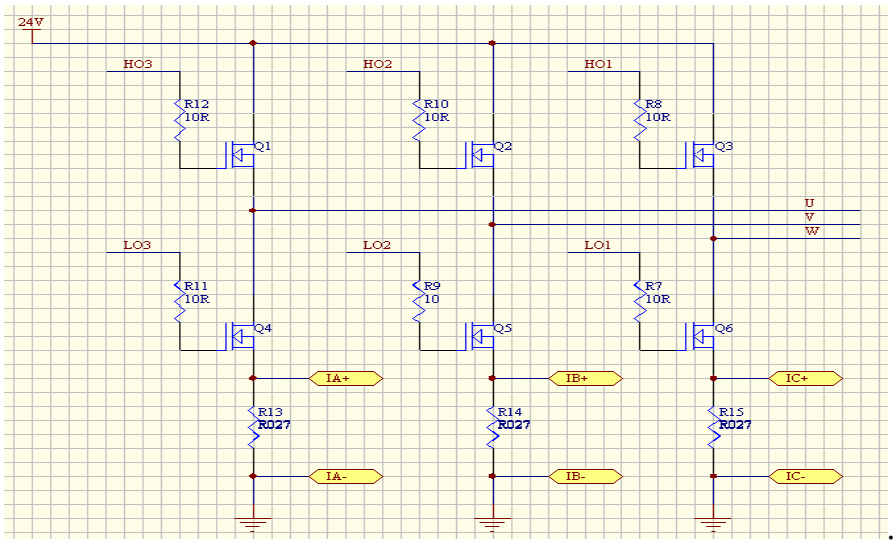


Figure 3.4 3-phase drive bridge using sample resistance

We had developed this circuit to control the PMSM. The problem is on the current sample part [21]. For the control of PMSM, the current part is very important; the reason is that the current loop is the inner loop. The precision of this out-loop is low the precision of the inner-loop. In the foregone design, the current was sampled by the resistance. This is a simple, low-cost method, but it's not a prior method for the improvement for the precision o the whole system. So we redesigned this part .We used

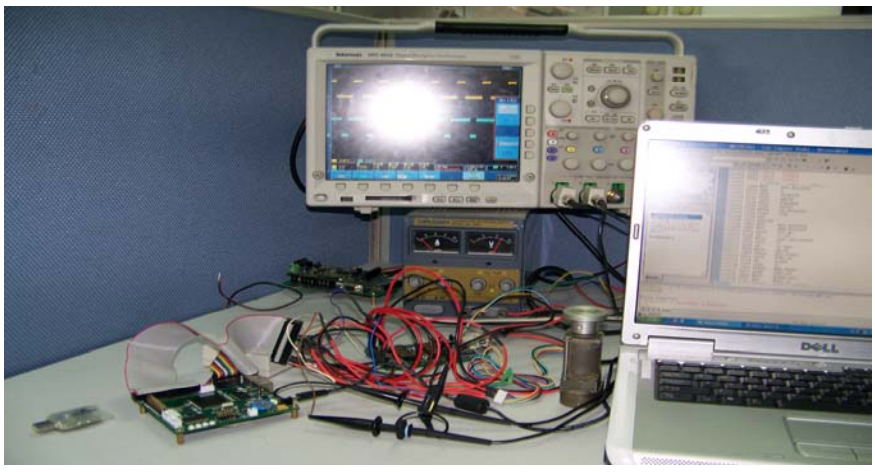


Figure 3.5 Experiment equipment

the current sensor to replace the resistance to improve the precision. In other case, though the motor had three stator phases, there's relationship between these three phases. In the Chapter 2 we had got the relationship Eq2.9.

So based on this information, we just need to measure two phase current; because the third one can be calculated in the program.

.

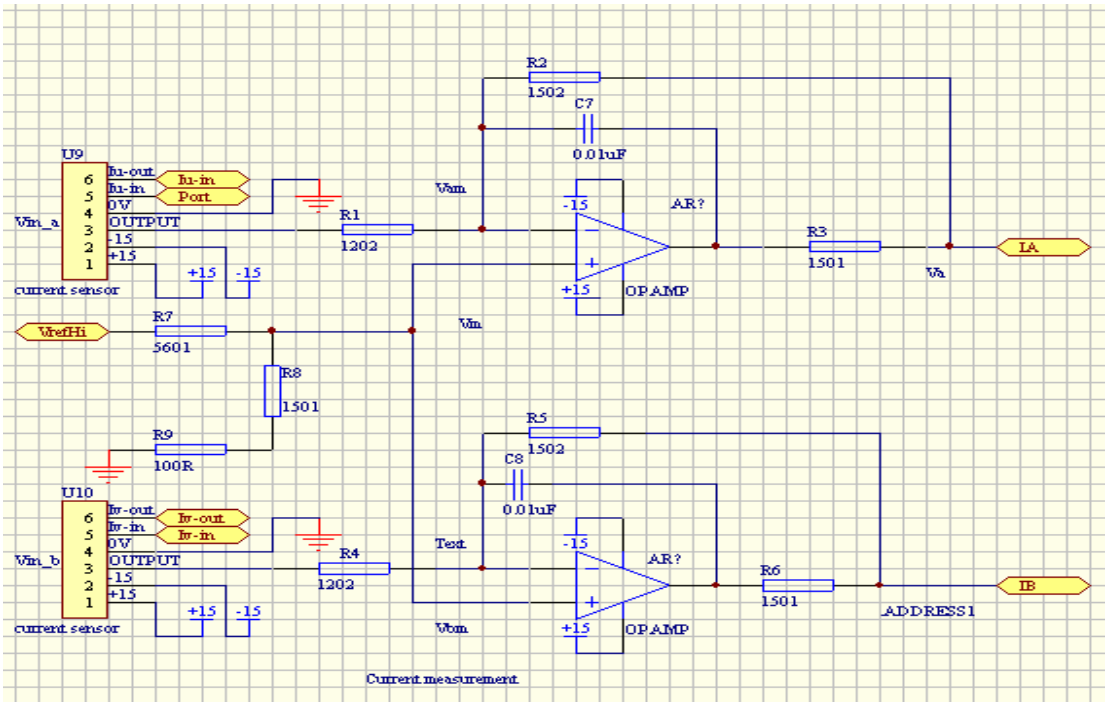


Figure 3.6 3-phase drive bridge using current sensor

Based on this circuit, we can get the expression development the principle of OP. Then input current of OP is equal to zero and Vam=Vbm=Vm, so for the A phase circuit:

$$\frac{V_{refHi}-V_m}{R_7}=\frac{V_m}{R_8+R_9} \tag{3.1}$$

$$\frac{V_{in\_a}-V_m}{R_1}=\frac{V_m-V_a}{R_2} \tag{3.2}$$

We can compute

$$V_a=\frac{(R_1+R_2)(R_8+R_9)}{R_1(R_7+R_8+R_9)}V_{refHi}-\frac{R_2}{R_1}V_{in\_a} \tag{3.3}$$

Because V_a will send to the DSP directly, it should be in 0-5V. Then through tuning V_{refHi}, we can insure V_a in this range whenever V_{in_a} is plus or negative. The similar principle can be used on the B phase circuit.

CHAPTER 4

DEVELOPMENT PROCESS AND ANALYSIS

Control algorithms and system model were previously simulated and tested in the MATLAB/SIMULINK. And then we can use the code auto-generation method to generate the code and test on the DSP board. Some block can be daggled into simulink from the c2000lib. And other blocks should be developed with c-mex s-function. Real-time workshop is the simulink add-on software that enables automatics C or assembly code generation from the simulink model. So we should take advantage of the MATLAB, Simulink, Real-time Workshop, and then develop some simulink blocks, that will enable programming of the development board.

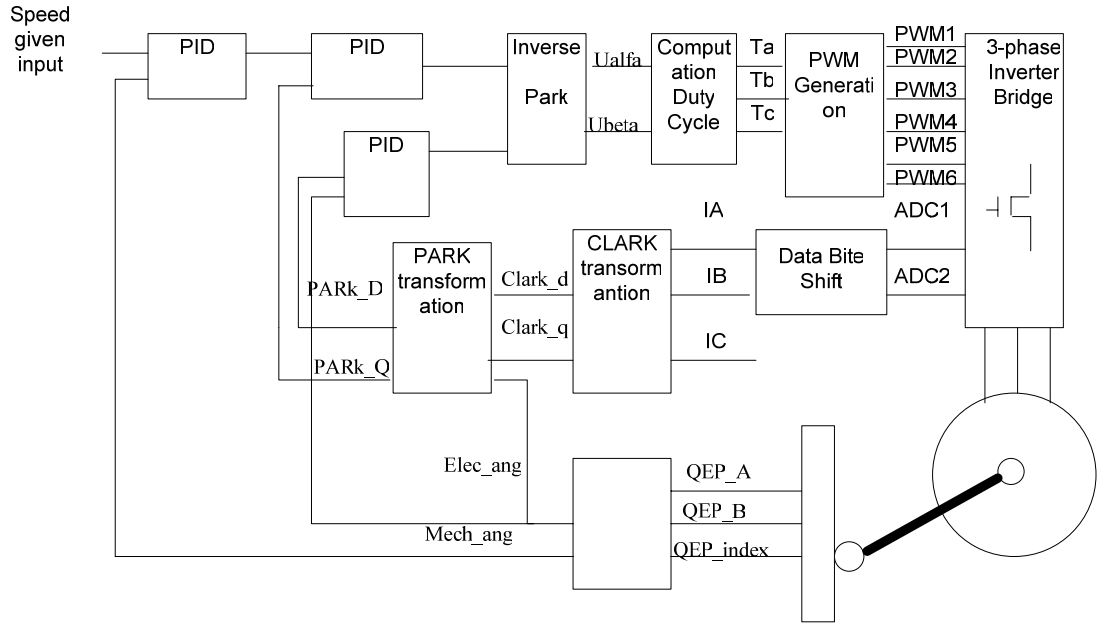


Figure4.1 PMSM control structure

The code generated in MATLAB should be loaded into the memory of DSP. Then we run the program in the PM of DSP. The controller board can send the PWM to the power board, and the power board collected the current and QEP to the controller board.

This results to the rolling of the shaft of the motor. Then, in this process, the information of the course should be collected. We provided two methods to solve this problem: SCI loader and RTDX. The collected data can be shown with MATLAB or other form.

Based on these tools, we can know the performance of the system. Then the parameters of the controller were tuned to make the system more stably and having better performance.

4.1 block development process and code generation

The block development process can be done with the system function or so-called S-function that can be written in C programming language (C-MEX S-function). S-function defines number of the inputs, numbers of outputs, algorithm for calculation of the block output value, sample time, etc for the custom simulink block. RTW (Real-time workshop) also generates makefile (model.mk) from the system template makefile. After that, the target language compiler is invoked into the code generation process and it converts simulink model to the C code. Then it is possible for the MATLAB to call the communication function. In the MATLAB toolbox, there's a tool named Link or CCS (code composer studio). With this tool, we can use MATLAB function to communicate with CCS and with information stored in memory and registers on a target. With the link we can transfer information to and from CCS and with the embedded objects we get information about data and functions stored in our signal processor memory and registers, as well as information about functions in our project.

The target for TI c2000 toolbox doesn't support the continuous time model. Only the discrete system can be transformed the code running in the processor. This is decided by the principle of dealing with the data for the CPU. The following is the suitable applications: Fixed-point arithmetic, Single rate, Multirate, Multistage, adaptive and frame based. And to workspace and from workspace toolbox can 't be used be together with target for TI C2000.

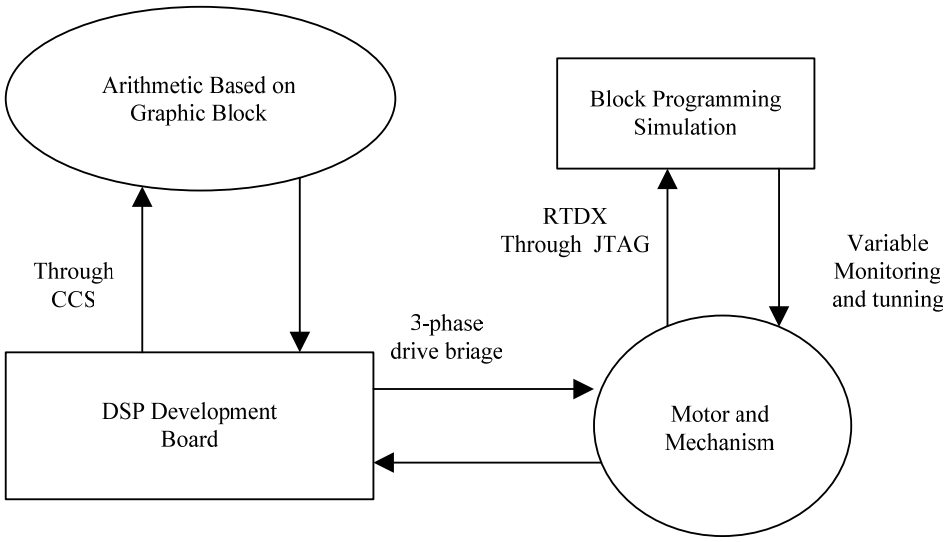


Figure 4.2 The frame that auto-code generation

From the forenamed process, the model built in Simulink can be transferred into the C language that can be compiled in the CCS. So, the next step is similar to developing program directly in C language. The C language is then can be compiled into assembly language. Then the code can be transferred into COFF file and HEX file. These file can be directly executed with the digital signal process. The whole code transferring process can be illuminated with the Figure 4.2.

From the figure, the graphical block was transformed into the binary format file. Using emulator (based on JTAG), we can load the COFF file into the DSP memory. With the RTDX technology, we can easily observe the result and tune the parameter. RTDX can send data from the DSP to the host computer while the DSP is running. A few m-files are used to operate the DSP, load different project files, configure RTDX channels, and manage the exchange of information between DSP and host PC. Originally, the intention was to have the DSP send speed data, current data and controller parameter, in real time. In fact, because of the difference between the processor speed and the sensor, the method of down-sample time and buffer is necessary. The rate can be decided for different sensor speed. However, RTDX read and write commands don’t take exactly the same amount of time to execute for each function call. This made reading and storing data erratic, as well as unnecessarily slow. Instead, the data recovery scheme was changed so that the DSP reported its speed reading only after the experiment trial had completed.

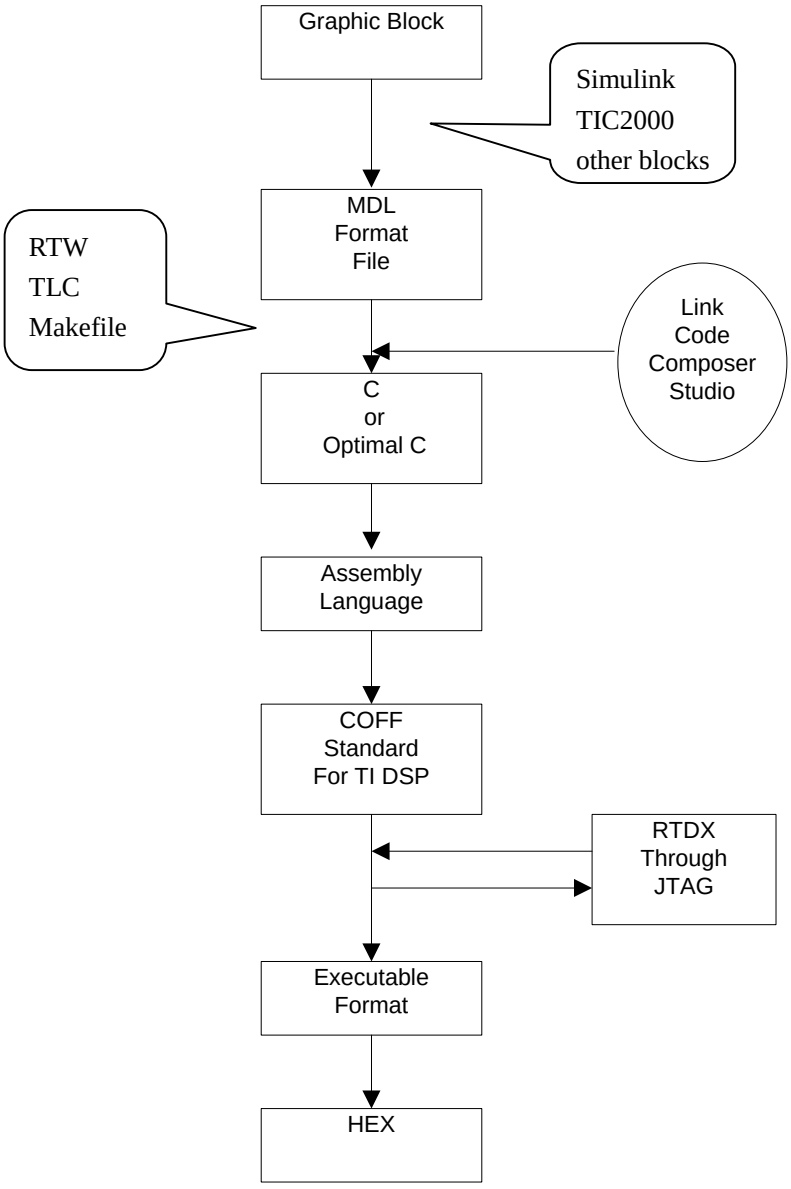


Figure 4.3 The language format transformation process

First, we should do the experiment to test what's the appropriately rate be choose to satisfied the need. The model is very simple. Just a QEP block and the RTDX, as the following fig;

After loading the auto-generated code into the DSP through JTAG interface, we can write M-file to execute the data collection and display. When the program is executing, we turn the shaft of motor, then the QEP pulse will be sent to the DSP, this

information can be collected with RTDX, the result is seen from the fig. And the corresponding M-file can be seen from the appendix.

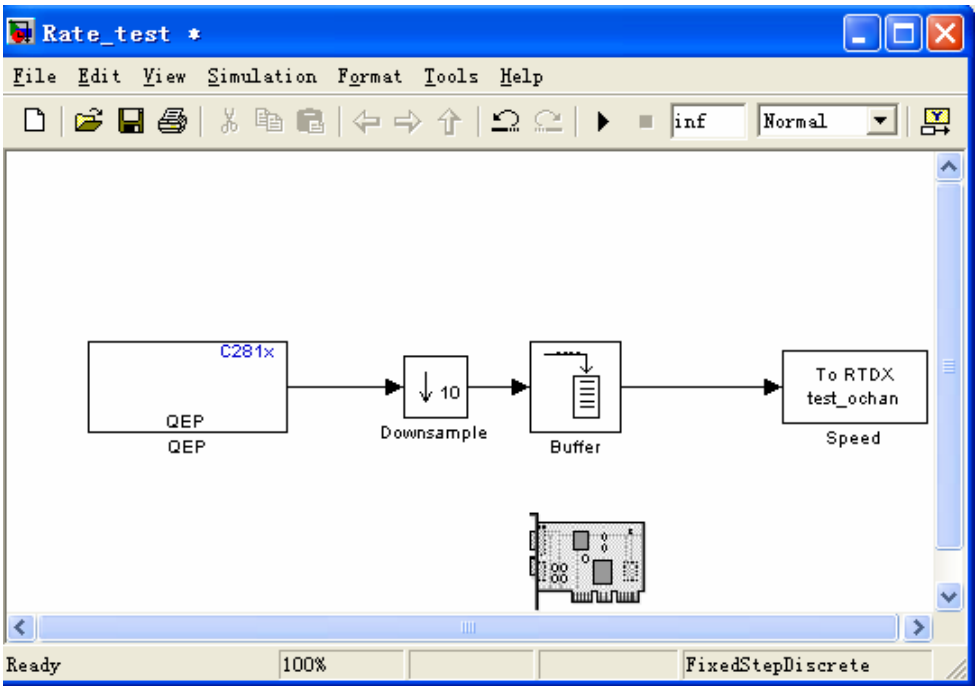


Figure 4.4 The down-sample-rate application for RTDX

In addition, we developed another method that got the information from DSP through SCI (referred to the Technosoft monitor tools) [5]. it just a section program resident in the flash, when the DSP was powered or reset, this program was configured to auto run. So in our application exercise, we can use it to communicate with the memory of DSP. So the basic loader is just a communication function that exchange information with host PC through SCI. Here we also need some other operation such as read/write PM (program memory), DM (data memory), run the program and other operation. All of these operations were operated by several characters. So we developed the command interpreter to make these command can be recognized by the host PC.

The basic structure can be explained by the following figure. In attention, this program was developed on MSK2407 board.

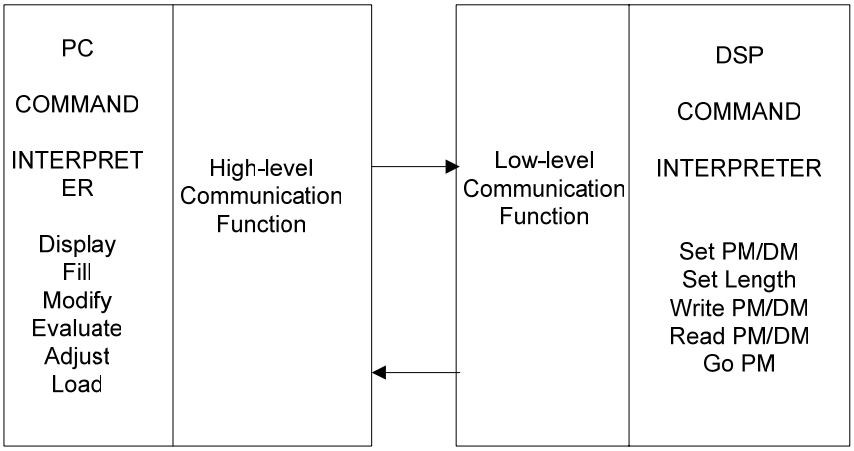


Figure 4.5 The structure of the SCI loader

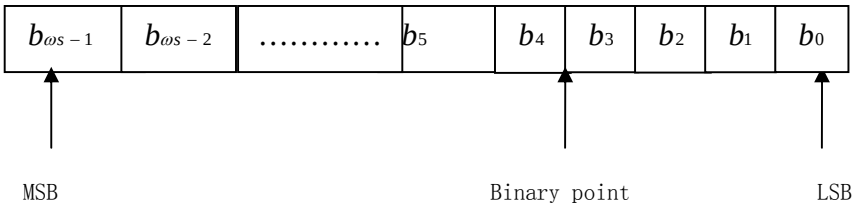
The advantage of this method is: no need emulator can tune the inner parameter very easily. The disadvantage is the program is always running in the flash, and so many interrupts slow the speed for dealing the control methods. So on the new 2812 board; we use the RTDX method to replace the SCI loader.

4.2 Fixed-point arithmetic

We know that TMS320LF2812 is fixed-point DSP. It means that the data to be deal with is 32 or 16 bit number. In digital hardware, numbers are stored in binary words. A binary word is a fixed-length sequence of binary digits (1's and 0's). How hardware components or software functions interpret this sequence of 1's and 0's is defined by the data type.

Binary numbers are used to represent either fixed-point or floating-point data types. A fixed-point data type is characterized by the word size in bits, the binary point, and whether it is signed or unsigned. The position of the binary point is the means by which fixed-point values are scaled and interpreted.

For example, a binary representation of a fractional fixed-point number is show below.



Where

- b_i is the i th binary digit.
- ωs is the word size in bits
- $b_{\omega s-1}$ is the location of the most significant(highest) bit(MSB).
- b_0 is the location of the least significant(lowest) bit(LSB).
- the binary point is shown four places to the left of the LSB. In this example, therefore, the number is said to have four fractional, or a fraction length of four.

Signed fixed-point numbers

Signed binary fixed-point numbers are typically represented in one of three ways:

- Sign/magnitude
- One's complement
- Two's complement

Two's complement is the most common representation of signed fixed-point numbers and is used by TI digital signal processors.

Negation using signed two's complement representation consists of a bit inversion (translation into one's complement) followed by the binary addition of a 1. For example, the two's complement of 000101 is 111011, as follows:

000101->111010(bit inversion) ->111011(binary addition of a 1 to the LSB)

Q Format Notation

The position of the binary point in a fixed-point number determines how you interpret the scaling of the number. When it performs basic arithmetic such as addition or subtraction, hardware uses the same logic circuits regardless of the value of the scale

factor. In essence, the logic circuit has no knowledge of a binary point. They perform signed or unsigned integer arithmetic- as if the binary point is to the right of bit.

In the IQmath Library, the position of the binary point in the signed, fixed-point data types is expressed in and designated by Q format notation. This fixed-point notation takes the form: $Q_{m.n}$

Where

• Q designates that the number is in Q format notation-the Texas Instruments representation for signed-point numbers.

• m is the number of bits used to designate the two's complement integer portion o the number.

• n is the number of bits used to designate the two's complement fractional portion of the number, or the number of bits to the right of the binary point.

In Q format, the most significant bit is always designated as the sign bit. Representing a signed fixed-point data type in Q format always requires m+n+1 bits to account for the sign. we usually use Q_n to replace $Q_{m.n}$.

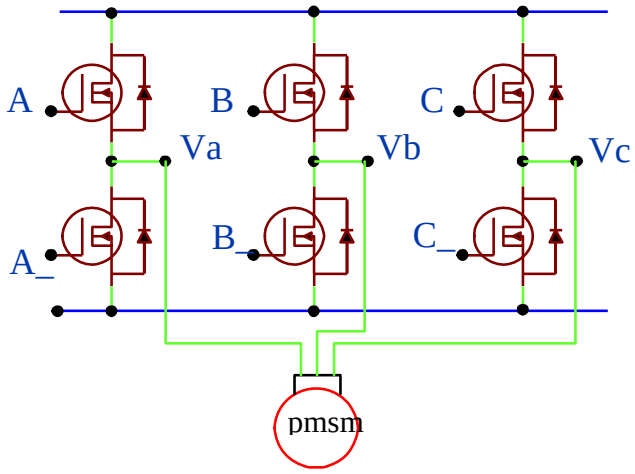
We used Q format to deal with decimal fraction part to improve the precision. But the range of the number was reduced. So the precision restrict the range, and the same to the reverse.

The coordinate transformation used fixed algorithm. In fact, it is just a shift bit process. What we need to is just to shift the bit to the origin position before and after the computation.

4.3 Computation of the duty cycle

The key point of the PMSM control is the computation of duty cycle. When the MOSFET is fired, for example, A phase upper arm is fired, we sign it with '1'. Because the upper arm and the under arm can't be fire at the same time. So, we sign the under arm with'0'. Then, there are 8 kinds of states for the 3-phase drive bridge shown in Figure 4.5. These states decide the 3-phase stator current which determined the stator magnet field. So to make the motor run, what we need to do is to give the right firing logic and

the right firing time length. Then we should insure the right duty cycle and the right configuration for 3-phase voltage.



N	ON	Va	Vb	Vc
0	000	0	0	0
1	110	+	+	-
2	100	+	-	-
3	101	+	-	+
4	001	-	-	+
5	011	-	+	+
6	010	-	+	-
7	111	0	0	0

Figure 4.6 The MOSFET fire logic chart

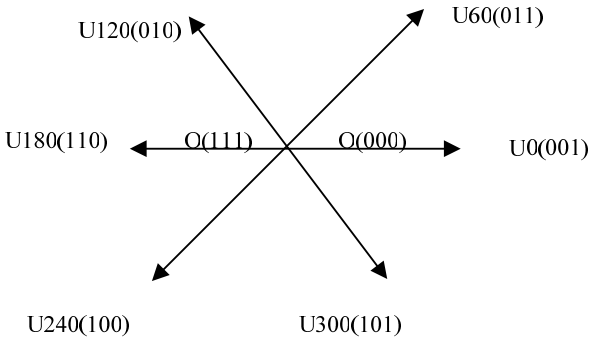


Figure 4.7 Basic space vectors

We use the MATLAB block to realize this function. The code development process was introduced forenamed method. For the given voltage which is a vector separated into the vertical coordinate, we know it will be in which sector. The near two vectors will be used to form it. This decide which MOSFET will be fired and the magnitude of it determined the fire time.

4.4 PMSM model development

Now, we will start the PMSM model developed with simulink and target for TI C2000 tool. Before the building of the model, several blocks had been prepared using the methods referred to pre-chapter. These blocks included hall block, duty cycle blocks, data shift blocks and so on. We can separate the develop process into the following steps：

(1) Configuration of the hardware and software development platform

(2) Generation of the basic PWM and test the basic model

(3) Building the current loop system and tuning the controller parameters

(4) Adding the velocity feedback and position feedback

(5) Tuning the controller parameter

Now, the first thing that we should do is to make the hardware communicate with software development correctly. The following step based on that we have install the CCS and MATLAB 7.0 and have correctly configured the board and CCS. The detail of this can be seen from the reference document provided by TI and the emulator company (this project use the emulator produced by SEED Co.). Step 1 PWM generation, the simulink model is following:

In this step, it confirms operation of peripheral and target independent modules on forward control path and the code framework. Utility functions RampCntl and freqRampGen are used to generate a saw-tooth waveform. This waveform is an emulation of the rotor angle. Its frequency is the running frequency.

The blocks RampCntl implement the following equation:

Case 1: when target_value > setpt_value

Setpt_value = setpt_value +1, for t=n.*Td, n=1.2.3…

Where Td = sampling time period* delay time

Case 2: when target_value< setpt_value

Setpt_value = setpt_value -1, for t=n.*Td, n=1.2.3…

Where Td = sampling time period* delay time

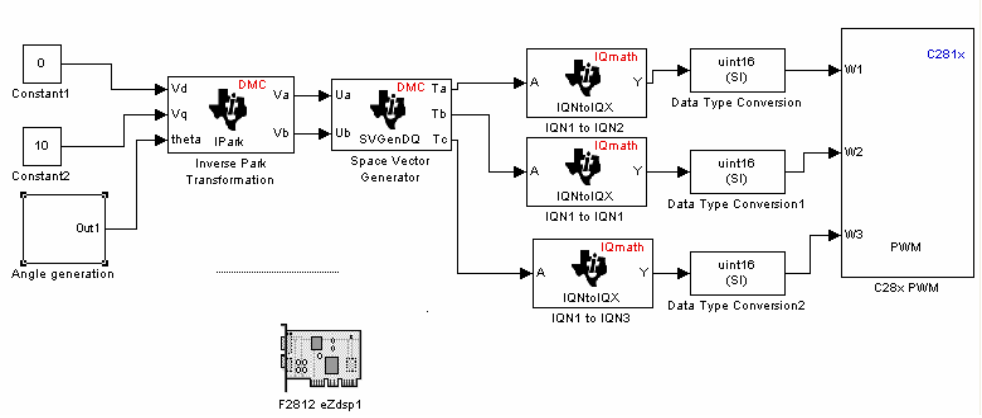


Figure 4.8 Open-loop model using for testing basic PWM output

For the blocks freq RampGen, the frequency of the ramp output is controlled by a precision frequency generation algorithm which relies on the wrap_around of finite length variables.

The wave that generate by the RampCntl and RampGen is just like as the following figure:

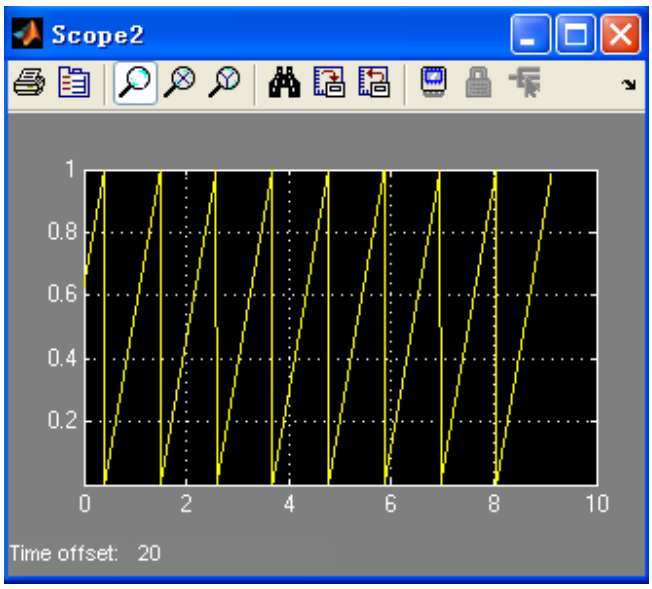


Figure 4.9 The toothed sawtooth wave

The parameters is target offset =0.01, Ramp gain = 1, RamGen offset=0. So, we can use this wave line to replace the encoder in the case we have not the enough information of the encoder. This is a necessary step that used to test the system in the ideal condition. The test result is that there's PWM output but this output is not available to control the motor. And the wave seen from the oscillograph like this Fig4-7

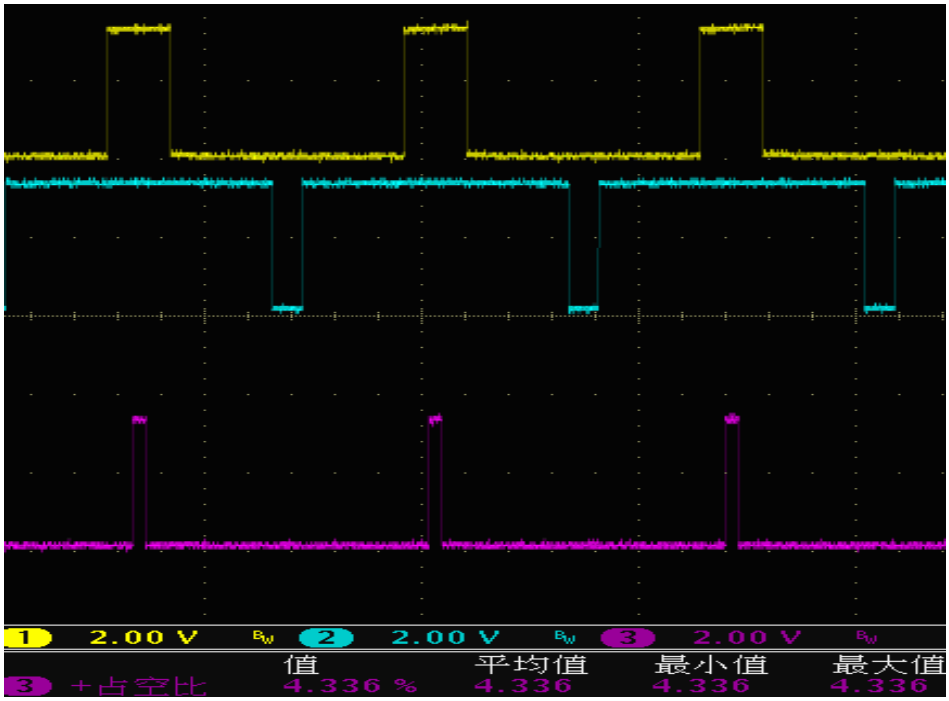


Figure 4.10 The PWM wave generated from the model

In fact, the truth PWM wave generated by the PMSM is changed in real-time because of the change of duty cycle.

Building the current loop

In this step, we added the current feedback: the sampled current was sent to the DSP. The AD was embedded in the DSP. It was 10-bit precision. But the data in the DSP is 16-bit or 32-bit. So the data collected from the AD input should be shifted and converted into the Q-forma data.

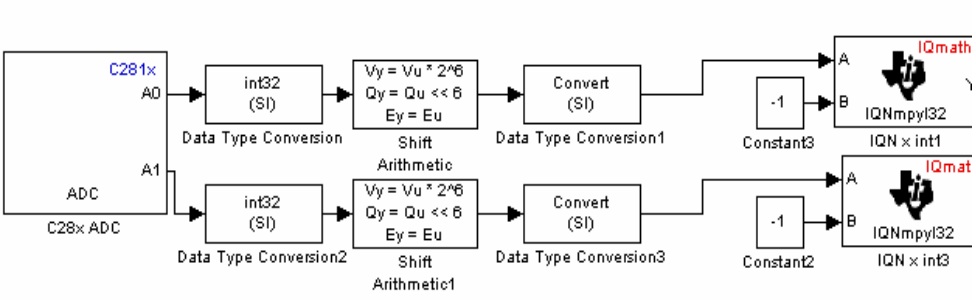


Figure 4.11 AD model for sampling the current

So after combining the AD part and transformation, we got the closed current-loop without controller. But the motor still can't run because of less of controller. Then we add the velocity loop. The given speed input was compared with the feedback of the QEP. So we got the following model in the next figure. In this model, the velocity loop was added following the PID controller. But we still use the toothed wave generator to replace the index of encoder. But in this step, we can run the motor cursory.

The next step we should to do is to use the true index signal and tune the PID parameter to make the motor running smoothly. So we got the final model with RTDX to transfer data. And the mile to configure the RTDX and start CCS can be seen in the appendices.

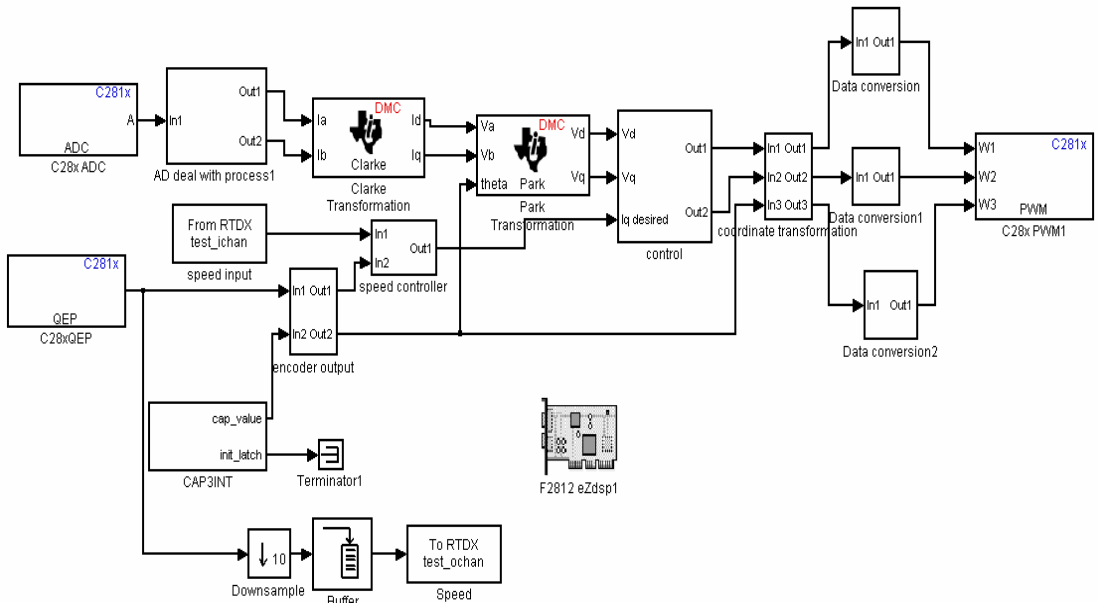


Figure 4.12 Speed control model with RTDX for collecting data

So, based on this model, the code was generated and loaded into memory of the controller board. The motor was running .And we can get the experiment data. The design of the speed loop assumes that the current loop is at least 10 times faster than speed loop.

4.5 Experiment result

Fig 4-13 is the voltage on the MOSFET when running the motor by hand. It's not the standard trapezoid or sinusoid. Because of this special back-EMF, this kind of motors can be used as BLDC and PMSM. We can't measure the phase voltage because there's no zero line from the motor. So we can measure the line voltage, Figure 4.14, Figure 4.15. Figure 4.16 is the speed display in MATLAB. Where P1, I1 are the current loop controller parameter, and the P2, I2 are the velocity loop controllers parameters, P3, I3 are the velocity loop controller parameters.

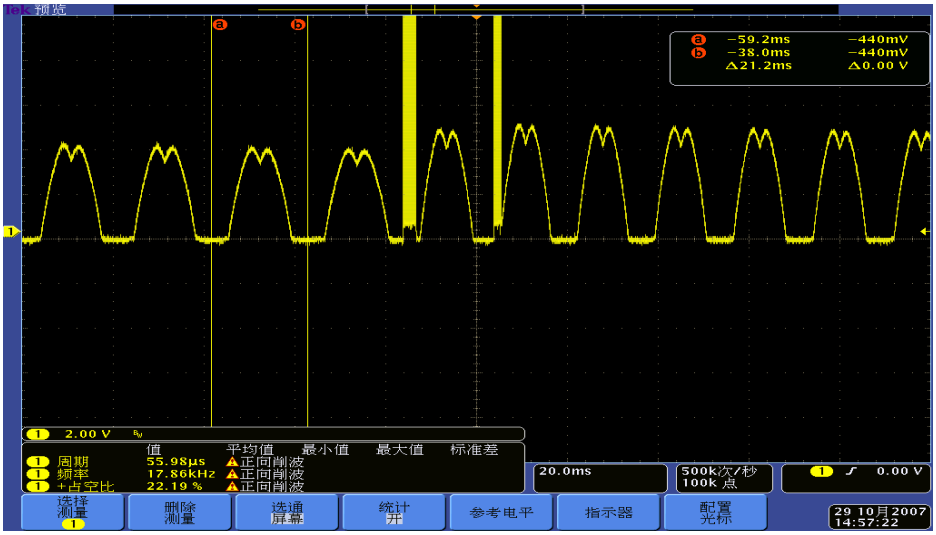


Figure 4.13 Back-EMF

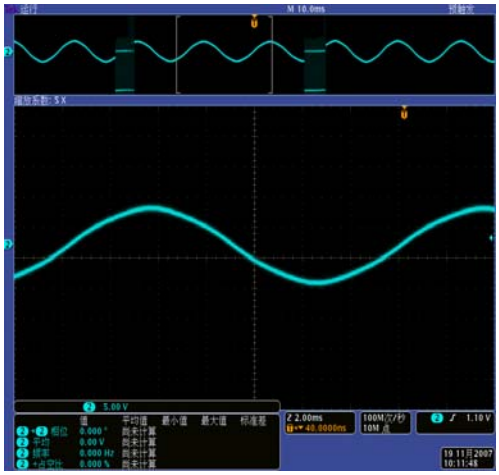


Figure 4.14 Line voltage of stator phase

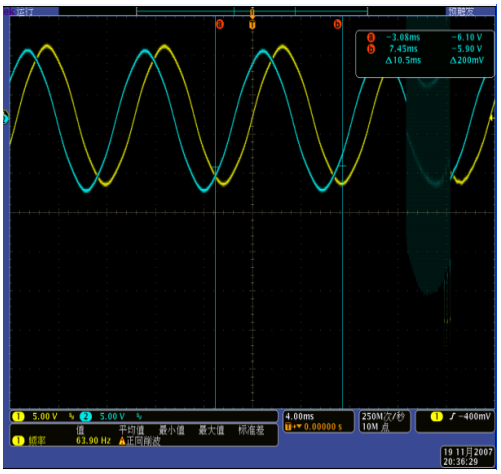


Figure 4.15 2-phase stator line voltage

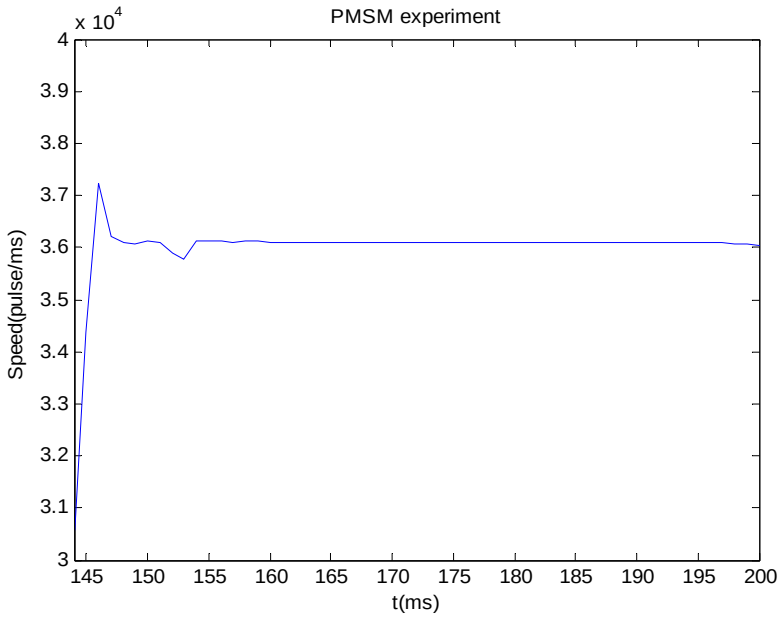


Figure 4.16 Speed responses at P1=0.75, I1=0.01

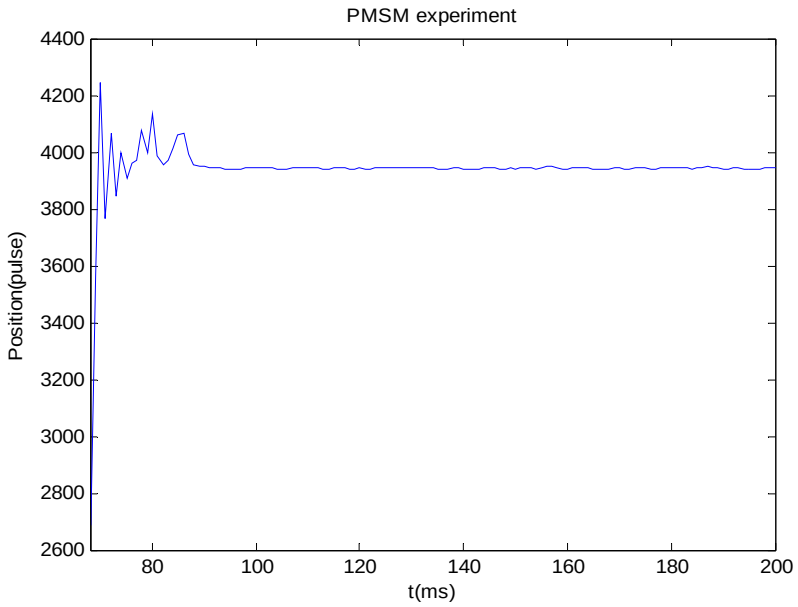


Figure 4.17 Position responses at given input at 3982 pulse

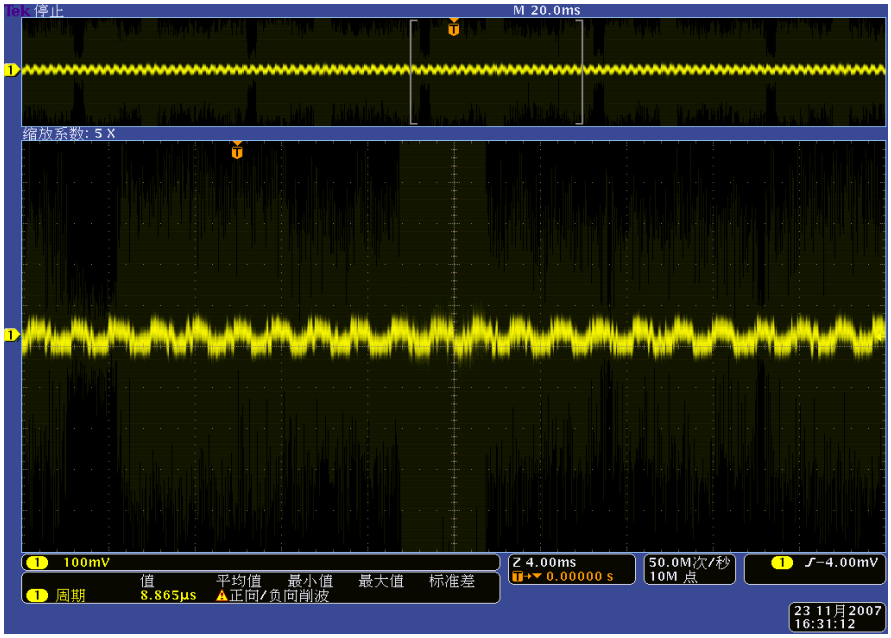


Figure 4.18 The signal phase current voltage

4.5 analyses

The error of the speed response at given 3.62e4 (HEX) pulse/ms is about 2.7%, and the raise time is about 5ms. The error of position response is about 6.8%， raise time is about 12ms.

Experiment result analyses:

(1) The detection of rotor position brings the error in the start part, because it is determined by the hall. The hall sensor just can detect the rotor position cursory. So we can improve it by using the high precision encoder.

(2) Data transformation in RTDX and MATLAB increased the response period. We can subtract it for the real system response through repeat test.

(3) The connection between hardware results the error of line voltage. Because the controller board is a starter board, we can redesign it more reliably.

(4) Without observe equipment for the current wave, so the parameters tuning for the current loop is not very good. This part may be try the High-RTDX, but here our chip don’t support high-RTDX now.

(5) There’s some conflict when using RTDX to collect the data of the current loop. The motor will be stop when collecting data from current loop. This may be arisen by the process velocity conflict.

But we got the good response cycle of speed. It had little error and quick small rise time. More important is all of process and result were implemented under MATLAB. This makes things easily.

CHAPTER 5

CONCLUSION

From the development process of PMSM control system, using MATLAB and simulink to develop an embedded control system offers many advantages over the typical method of hand translation from pseudo code to source code. There is an inherent advantage in using Simulink to model the control system. It saves time and effort. This allows the engineer to design a system in a straightforward manner, rather than to waste time writing source code from scratch.

We use the SEEDDSP2812 as our controller board. The F2812 had the appropriate peripherals to properly interface with the motor and encoder. In addition, the Embedded Target for TI C2000 DSP blocksets in simulink allows for simple software interfacing to these peripherals. It was possible to create simulink models for motor testing.

It was soon determined that some code was necessary for acquiring data from the DSP. RTDX can send data from the DSP to the host computer while the DSP is running. A few m-files are used to operate the DSP, load different project files, configure RTDX channels, and manage the exchange of information between DSP and host PC. Originally, the intention was to have the DSP send speed data in real-time. However, RTDX read and write commands don't take exactly the same amount of time to execute for each function call. This made reading and storing data erratic. So，the data recovery scheme was changed so that the DSP reported it's speed reading only after the experiment trial had completed. Acute readings of the speed were available. The sample time of the system is 10ms.

Some more work could be done to combine the planning velocity with in this system. It needs other hardware to complete the experiment. Also some work can be done to analyze the difference between the continuous and discrete system. Experimentation could let the user determine how fast the sampling frequency of the discrete system must

be to approach the performance of the continuous system. Also, other robust algorithm can be added into the system to improve the stability and robust characters.

REFERENCES

1. L. Romeral, Motion control for Electric Drives，Springler-Verlag,1990
2. E. Lee， Article for Drive system group
3. J. Astrom and T. Hagglund, PID controllers: Theory, Design and Tuning, 2nd Edition, *Instrument Society of America,* 1995
4. M. Ouassaid, M. Cherkaoui, Nonlinear Torque Control of PMSM: A Lyapunov Technique Approach, *Transaction on engineering, computing and technology* V6 June 2005
5. J. Solsona, M. I. Valla, Nonlinear control of a permanent magnet synchronous motor with disturbance torque estimation, *IEEE Transaction on Energy Conversion*, June , 2000
6. J. Zhou ,Y. wang, Adaptive back stepping speed controller design for a permanent magnet synchronous motor
7. Shun-zhi Xiao ,Design for fuzzy logic current controller of PMSM,Chung Yuan Christian University,2001
8. A.Mulpur, Z.Galijasevic, Model-based Design shortens Development Time of Motor Control Applications
9. Target for TIC2000 use guide, *the MathWorks Inc.*
10. R.Valentine ,Motor control electronics handbook, *McGraw-Hill,* 1998
11. P.Pillay, R.Krishnan, Modeling, Simulation, and Analysis of Permanent Magnet Synchronous-Motor Drives, Part I: The Permanent Magnet Synchronous-motor Drive, *IEEE Transaction on Industry Applications*, 1989
12. Clarke & Park Transforms on the TMS320C2xx, *TI Co. Application Notes*
13. Dai-Fuyu, Cai-Xing'an, SOPC-Based Servo Control System for the XYZ-Table, *Southern Taiwan University/ Motor Engineering Research Institute,* 2005
14. A.Klee, Development of motor speed control system using MATLAB and simulink, implemented with digital signal processor, *Spring term*,2005
15. Implementation of a speed Field oriented control of a 3-phase PMSM motor using TMS320F2812, *Application notes of Texas Instruments*
16. 交流马达控制模拟及空间向量波宽调变 SIMULINK 模拟方块之建立，蔡明发，明新科技大学电机工程系，2002
17. L.Carrilo ,Modeling and simulation of permanent magnet synchronous motor drive system, *University of Puerto Rico Mayaguez Campus*, 2006
18. MCK2407 User manual, *Technosoft*, 2001
19. 李永东著,交流电机数字控制系统，机械工业出版社，2002 年
20. 王晓明 王玲等著， 电动机的 DSP 控制-TI 公司 DSP 应用，北京航空航天大学出版社，200 年 7 月

21. E.Simon, Implementation of a speed Field Oriented Control of 3-phase PMSM motor using TMS320F240, *TI application report*
22. D.Hercog, M.Curkovic, G.Edelbaher, Programming of the DSP2 board with the MATLAB/Simulink, *University of Maribor Faculty of Electrical Engineering and Computer Science, Slovenia*
23. 俊原科技股份有限公司技术应用报告，DSP C 语言与 Simulink 界面连接操作手册
24. 胡凯， 基于 DSP56F80 的交流电机驱动控制试验平台，国防科技大学机电工程与自动化学院，2002 年 10 月
25. R.Safanc, MTruntic, Control and robotics remote laboratory for engineering education, *Control and robotics laboratory for engineering education*
26. K.Boateng, C. Badu, Integration of MATLAB Tools for DSP Code Generation, *Bradley University ECE department*,12.06.2005
27. Power study commissioned by Ontario Hydro to third party testing lab, *Ortech International Energy Technologies*, 1994
28. Gross, Hans, Electrical feed drives for machine tools, Siemens Aktiengesell schaft, Berlin and Munchen, *John Wiley and Sons Ltd*.,1983
29. IEEE industry application society, Introduction to field orientation and high performance AC drives, *Tutorial course presented at the 1985 IEEE industry application society annual meeting*, October 6-7. Toronto. Canada.
30. 董良,基于 DSP 的无刷直流电机驱动器的设计，哈尔滨工业大学硕士研究生论文，2006
31. A. A. Hassan and M. Azzam, Robust Control of a Speed Sensorless Permanent Magnet Synchronous Motor Drive
32. Clarke & Park Transforms on the TMS320C2xx, *TI Co. Application Notes*
33. Faa-Jeng Lin, Rong-Jong Wai, A PM Synchronous Servo Motor Drive with an On-line Trained Fuzzy Neural Network Controller, *IEEE Transaction on Energy Conversion*,1998
34. 夏长亮，李正军，基于自抗扰控制器的无刷直流电机控制系统，中国电机工程学报，2005
35. 萧顺治，永磁同步马达的模糊电流控制器之设计，私立中原大学电机工程系硕士论文集，2003
36. 李钟明， 刘卫国 等著，稀土永磁电机， 国防工业出版社，2001
37. J. Streeter, A. Keane, and John R. Koza, Automatic Synthesis Using Genetic Programming of Improved PID Tuning Rules, *IFAC*,2002

APPENDICES

Part of program

1 M-file：test.m

The following program is used to record the data collection through the RTDX and plot it in the figure.

```
   %declare the function
function void=test1()
cc=ccsdsp;
r= cc.rtdx;
r.disable;
r.configure(6400,4,'continuous');
r.open('test_ochan','r');
r.enable
timeout_msg='Timeout';
NOMOREDATAMSG = 'no more data is available!';
errmsg=NaN;
run(cc)


pause(5)
   while(isrunning(cc))
    %count and read data
    numMsgstest_ochan = r.msgcount('test_ochan');
    if (numMsgstest_ochan>1);
   %flush frames as necessary to maintain real-time display
    r.flush('test_ochan',numMsgstest_ochan-1);
    end
     if (numMsgstest_ochan)
      speed =r.readmsg('test_ochan','int32');
      disp(speed);
     end
```

```
   end
   x_axis =0:0.2:9.8;
    length(x_axis)
     figure;
     length(speed)
     plot(x_axis,speed);
     %subplot(x_axis,speed);
   title('measure');
      xlabel('t(s)');
      ylabel('Speed(rpm)');
      grid('off');
      XMIN = 0;
      XMAX = 10;
      YMIN = 0;
      YMAX = 530;
      axis([XMIN XMAX YMIN YMAX]);
  %    end
% call to the main updatae loop
r.disable('test_ochan');
r.disable
function void = why_u()
cc= ccsdsp;
r= cc.rtdx;
r.disable;
r.configure(64000,4,'continuous');
r.open('test_ochan','r');
r.enable
timeout_msg='Timeout!';
NOMOREDATAMSG = 'no more data is available!';
errmsg= NaN;
```

```
run(cc)
pause(5)
while(isrunning(cc))
 numMsgtest_ochan=r.msgcount('test_ochan');
 if (numMsgtest_ochan>1);
    r.flush('test_ochan',numMsgtest_ochan-1);
 end
 if(numMsgtest_ochan)
    data=r.readmsg('test_ochan','int32');
    disp(data);
 end
end
x_axis= 0:0.2:19.8
length(x_axis)
length(data)
figure;
plot(data)
title('Test of PMSM speed when p1=0.62,i1=0.02;p2=0.89,i2=0.01');
xlabel('t(s)');
ylabel('speed(pulse)');
grid('off');
XMIN= 0;
XMAX= 50;
YMIN= 0;
YMAX=500000;
axis([XMIN XMAX YMIN YMAX])
```

2 CMD file:

MEMORY

```
{
PAGE 0:    /* Program Memory */
   Z6      : origin = 0x100000, length = 0x008000
   OTP         : origin = 0x3D7800, length = 0x000800
   BEGIN       : origin = 0x3F7FF6, length = 0x000002
   RAMH0       : origin = 0x3F8000, length = 0x002000
   ROM         : origin = 0x3FF000, length = 0x000FC0
   RESET       : origin = 0x3FFFC0, length = 0x000002
   VECTORS     : origin = 0x3FFFC2, length = 0x00003E
PAGE 1 :   /* Data Memory */
   RAMM0       : origin = 0x000000, length = 0x000400
   RAMM1       : origin = 0x000400, length = 0x000400
   DEV_EMU     : origin = 0x000880, length = 0x000180
   PIE_VECT    : origin = 0x000D00, length = 0x000100
   FLASH_REGS  : origin = 0x000A80, length = 0x000060
   CSM         : origin = 0x000AE0, length = 0x000010
   XINTF       : origin = 0x000B20, length = 0x000020
   CPU_TIMER0  : origin = 0x000C00, length = 0x000008
   PIE_CTRL    : origin = 0x000CE0, length = 0x000020
   ECANA       : origin = 0x006000, length = 0x000040
   ECANA_LAM   : origin = 0x006040, length = 0x000040
   ECANA_MOTS  : origin = 0x006080, length = 0x000040
   ECANA_MOTO  : origin = 0x0060C0, length = 0x000040
   ECANA_MBOX  : origin = 0x006100, length = 0x000100
   SYSTEM      : origin = 0x007010, length = 0x000020
   SPIA        : origin = 0x007040, length = 0x000010
   SCIA        : origin = 0x007050, length = 0x000010
   XINTRUPT    : origin = 0x007070, length = 0x000010
   GPIOMUX     : origin = 0x0070C0, length = 0x000020
   GPIODAT     : origin = 0x0070E0, length = 0x000020
```

```
   ADC         : origin = 0x007100, length = 0x000020
   EVA         : origin = 0x007400, length = 0x000040
   EVB         : origin = 0x007500, length = 0x000040
   SCIB        : origin = 0x007750, length = 0x000010
   MCBSPA      : origin = 0x007800, length = 0x000040
   RAML0       : origin = 0x008000, length = 0x001000
   RAML1       : origin = 0x009000, length = 0x001000
   Z6      : origin = 0x108000, length = 0x008000
   CSM_PWL     : origin = 0x3F7FF8, length = 0x000008
}
SECTIONS
{
   codestart          : > BEGIN       PAGE = 0
   .text              : > Z6       PAGE = 0
   .cinit             : > Z6       PAGE = 0
   .pinit             : > Z6,      PAGE = 0
   .reset             : > RESET,      PAGE = 0, TYPE = DSECT
   IQmath             : > Z6       PAGE = 0
   IQmathTables       : > ROM         PAGE = 0, TYPE = NOLOAD
   .stack             : > RAML1       PAGE = 1
   .ebss              : > Z6       PAGE = 1
   .econst            : > Z6       PAGE = 1
   .esysmem           : > Z6       PAGE = 1
    ramfuncs          : > Z6       PAGE = 1
   PieVectTableFile  : > PIE_VECT,     PAGE = 1
/*** Peripheral Frame 0 Register Structures ***/
   DevEmuRegsFile    : > DEV_EMU,      PAGE = 1
   FlashRegsFile     : > FLASH_REGS,   PAGE = 1
   CsmRegsFile       : > CSM,          PAGE = 1
   XintfRegsFile     : > XINTF,        PAGE = 1
```

```
   CpuTimer0RegsFile : > CPU_TIMER0,   PAGE = 1
   PieCtrlRegsFile   : > PIE_CTRL,     PAGE = 1
/*** Peripheral Frame 1 Register Structures ***/
   SysCtrlRegsFile   : > SYSTEM,       PAGE = 1
   SpiaRegsFile      : > SPIA,         PAGE = 1
   SciaRegsFile      : > SCIA,         PAGE = 1
   XIntruptRegsFile  : > XINTRUPT,     PAGE = 1
   GpioMuxRegsFile   : > GPIOMUX,      PAGE = 1
   GpioDataRegsFile  : > GPIODAT       PAGE = 1
   AdcRegsFile       : > ADC,          PAGE = 1
   EvaRegsFile       : > EVA,          PAGE = 1
   EvbRegsFile       : > EVB,          PAGE = 1
   ScibRegsFile      : > SCIB,         PAGE = 1
   McbspaRegsFile    : > MCBSPA,       PAGE = 1
/*** Peripheral Frame 2 Register Structures ***/
   ECanaRegsFile     : > ECANA,        PAGE = 1
   ECanaLAMRegsFile  : > ECANA_LAM     PAGE = 1
   ECanaMboxesFile   : > ECANA_MBOX    PAGE = 1
   ECanaMOTSRegsFile : > ECANA_MOTS    PAGE = 1
   ECanaMOTORegsFile : > ECANA_MOTO    PAGE = 1
/*** Code Security Module Register Structures ***/
   CsmPwlFile        : > CSM_PWL,      PAGE = 1
}
```

哈尔滨工业大学硕士学位论文原创性声明

本人郑重声明：此处所提交的硕士学位论文《Development of PMSM Control Using MATLAB, Implemented with a Digital Signal Processor》，是本人在导师指导下，在哈尔滨工业大学攻读硕士学位期间独立进行研究工作所取得的成果。据本人所知，论文中除已注明部分外不包含他人已发表或撰写过的研究成果。对本文的研究工作做出重要贡献的个人和集体，均已在文中以明确方式注明。本声明的法律结果将完全由本人承担。

作者签字　　日期：　年 月 日

哈尔滨工业大学硕士学位论文使用授权书

《Development of PMSM Control Using MATLAB, Implemented with a Digital Signal Processor》系本人在哈尔滨工业大学攻读硕士学位期间在导师指导下完成的硕士学位论文。本论文的研究成果归哈尔滨工业大学所有，本论文的研究内容不得以其它单位的名义发表。本人完全了解哈尔滨工业大学关于保存、使用学位论文的规定，同意学校保留并向有关部门送交论文的复印件和电子版本，允许论文被查阅和借阅。本人授权哈尔滨工业大学，可以采用影印、缩印或其他复制手段保存论文，可以公布论文的全部或部分内容。

保 密□，在　年解密后适用本授权书。

本学位论文属于

不保密□。

（请在以上相应方框内打“ √ ”）

作者签名：　　日期：　年 月 日

导师签名：　　日期：　年 月 日

基于MATLAB和DSP的永磁同步电机控制

作者：刘辉
学位授予单位：哈尔滨工业大学