

“DEVELOPMENT OF A PC NUMERICAL CONTROL SYSTEM FOR HIGH VOLTAGE
SPHERE GAP CONTROL”

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ONUR KASAP

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

JUNE 2005

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. İsmet Erkmen
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Mirzahan Hızal
Supervisor

Examining Committee Members:

Prof. Dr. Ahmet Rumeli (METU EE) _____

Prof. Dr. Mirzahan Hızal (METU EE) _____

Prof. Dr. Muammer Ermiş (METU EE) _____

Prof. Dr. Nevzat Özay (METU EE) _____

Msc. Ulaş Karaağaç (TÜBİTAK BİLTEN) _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name:

Signature :

ABSTRACT

DEVELOPMENT OF A PC NUMERICAL CONTROL SYSTEM FOR HIGH VOLTAGE SPHERE GAP CONTROL

KASAP, Onur

M.Sc. , Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Mirzahan Hızal

June 2005, 153 Pages

In this thesis, a high precision motion and position control system has been developed and applied to a high voltage sphere gap control and measurement system. The system is able to support up to 3-axes position and motion control. The control system includes a microcontroller card, three DC servo motor driver cards and a data storage unit. To provide communication between computer and motion control system, the Universal Serial Bus (USB) port is used.

The microcontroller card is equipped with an USB interface and a PIC (Peripheral Interface Controllers) microcontroller. This microcontroller controls the dedicated motion control processors (LM629), on servo motor driver cards and read/write operations of data storage unit, which consists of a Multi Media Card.

Keywords: High voltage sphere gap, DC servo motor motion control, PIC microcontroller, LM629, Multi Media Card (MMC)

ÖZ

YÜKSEK VOLTAJ KÜRE ARALIĞINI KONTROL AMAÇLI BİR BİLGİSAYAR SAYISAL KONTROL SİSTEMİNİN GELİŞTİRİLMESİ

KASAP, Onur

Yüksek Lisans , Elektrik-Elektronik Mühendisliği Bölümü

Tez Yöneticisi: Prof. Dr. Mirzahan Hızal

Haziran 2005, 153 sayfa

Bu tezde, yüksek hassasiyetli bir hareket ve pozisyon kontrol sistemi geliştirilmiş ve bir yüksek gerilim küre aralığının ayarlanmasında ve ölçümünde kullanılmıştır.. Sistem üç eksene kadar hareket ve pozisyon kontrolünü desteklemektedir. Kontrol sistemi bir adet mikrodenetleyici kartı, üç adet DC servo motor sürücü kartı ve bilgi depolama ünitesi içermektedir. Bilgisayar ve hareket kontrol sistemi arasındaki iletişimi sağlamak için USB portu kullanılmıştır.

Mikrodenetleyici kartı USB arayüzü ve PIC mikrodenetleyicisi ile donanmıştır. Bu mikroişlemci servo motor sürücü kartları üzerindeki, hareket için adanmış işlemcileri (LM629) ve Multi Media kartı içeren depolama ünitesinin okuma yazma işlemlerini kontrol etmektedir.

Anahtar Kelimeler: Yüksek voltaj küre aralığı , DC servo motor hareket kontrolü, PIC mikrodenetleyici, LM629, Multi Media Kart

To My Family

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation to Prof. Dr. Mirzahan Hızal for his encouragements, guidance and supervision.

I wish to thank Mehmet Akyüz for his suggestions on software.

Finally, I would like to thank Seren Yüksel for her precious help, great support and understanding. I believe without her this thesis would not have been completed.

TABLE OF CONTENTS

PLAGIARISM	iii
ABSTRACT	iv
ÖZ	v
ACKNOWLEDGEMENTS	vii
TABLE OF CONTENTS	viii
LIST OF TABLES	xi
LIST OF FIGURES	xii
 CHAPTERS	 1
1.INTRODUCTION	1
1.1 Sphere Gaps	1
1.2 Computer Numerical Control (CNC) Application	5
1.2.1 CNC Machine Tools	7
1.3 Control Systems	7
1.3.1 Open-Loop Control	8
1.3.2 Closed-Loop Control	9
1.3.3 Position Control Systems	10
1.3.3.1 Position Transducers	10
2.CONTROL MOTOR TYPES	14
2.1 Step Motors	14
2.1.1 Drive Circuits	16
2.1.1.1 Flux Direction Control	17
2.1.1.2 Current Control	17
2.1.2 Torque-Speed Characteristics	18
2.2 Servo Motors	19
2.2.1 DC Servo Motors	19
2.2.1.1 Drive circuit	19
2.2.1.2 Speed Control	21
2.2.1.3 Torque-Speed Characteristics	22
2.2.2 AC Servo Motor	23
2.2.2.1 Construction and Operating Principle	23
2.2.2.2 Torque Control	24
2.2.2.3 Speed Control	25
2.2.2.4 Torque- Speed Characteristic	25
3. MOTOR CONTROL HARDWARE	27
3.1 General Overview to PIC Microcontroller	30
3.1.1 PIC 16F877 Microcontroller	31
3.1.2 General Architecture	32
3.1.3 Clocking Scheme/Instruction Cycle	34
3.1.4 CPU AND ALU	35
3.1.4.1 CPU Registers	37
3.1.5 Program Memory Organization and Program Counter	38
3.1.6 Data Memory	40
3.1.7 Hardware Features	41

3.1.7.1	I/O Ports	41
3.1.7.2	Timers	42
3.1.7.3	USART	42
3.1.7.4	Capture/Compare/PWM Modules	42
3.1.7.5	A/D Converter	42
3.1.7.6	Synchronous Serial Port (SSP) Module	43
3.2	Digital I/O Interface	45
3.2.1	USB Hardware	46
3.2.1.1	Connectors	46
3.2.1.2	Electrical	47
3.2.1.3	Speed Identification	48
3.2.1.4	Power Distribution	48
3.2.2	USB Protocols	49
3.2.2.1	Common USB Packet Fields	49
3.2.2.2	USB Packet Types	50
3.2.3	Endpoints	51
3.3	Microcontroller Card	52
3.4	Precision Motion Controller LM629	56
3.4.1	Hardware Architecture	58
3.4.2	Motor Position Decoder	59
3.4.3	Velocity Profile (Trajectory) Generation	59
3.4.4	PID Compensation Filter	61
3.4.5	Motor Outputs	63
3.4.6	LM629 Reading and Writing Operations	63
3.4.7	User Command Set	64
3.4.8	Programming LM629	65
3.5	Data Storage	71
3.6	Data Storage Circuit	77
3.7	Motor driver circuit	79
3.8	Buffer Circuit	80
4.	MOTOR CONTROL SOFTWARE	83
4.1	PIC Programming	85
4.1.1	PIC Basic Pro	85
4.2	Firmware Description	88
4.2.1	Storage Part	89
4.2.2	LM629 Part	91
4.2.3	Data processing Part	94
4.2.4	Subroutines	97
4.2.4.1	MMC Subroutines	97
4.2.4.2	LM629 subroutines	98
5.	CONCLUSION	101
	REFERENCES	103
	APPENDIX A: MACHINE CODES	107
	APPENDIX B: DLP-IO26 USB Interface and Schematic	108
B.1)	FTU245AM	108
B-2)	SCHEMATIC DIAGRAM	113
	APPENDIX C: MOTOR DRIVER	114
	APPENDIX D: BUFFER CIRCUIT	116
D.1)	SN54/74LS245 OCTAL BUS TRANSCEIVER	116

D.2) DM74LS241 Octal TRI-STATE... Buffers/Line Drivers/Line Receivers	117
APPENDIX E: PIC BASIC PRO	118
E.1) PicBasic Pro Basics	118
E.2) PicBasic Pro Statement Reference	124
APPENDIX F: PIC Pin Variables and Functions	125
APPENDIX G: FIRMWARE.....	126
APPENDIX H: SPI Command Set.....	150

LIST OF TABLES

Table 1-1	Sphere gap with one terminal grounded breakdown voltages.....	2
Table 2-1	H-BRIDGE modes of operation.....	20
Table 3-1	Basic features of PIC 16F877.....	31
Table 3-2	Direct and Indirect Addressing of Banks.....	40
Table 3-3	USB pin functions.....	47
Table 3-4	Description of DLP-IO26 40 pin header pins.....	55
Table 3-5	LM629 User Command Set.....	64
Table 3-6	Initialization Module (with Hardware Reset).....	67
Table 3-7	Filter Programming Module.....	69
Table 3-8	Trajectory Programming Module.....	70
Table A-1	Mid-Range MCU Instruction Set.....	107
Table D-1	Truth Table of LS245.....	116
Table E-1	PIC Basic Pro Math Operators.....	122
Table E-2	Comparison Operators.....	123
Table E-3	Logical Operators.....	123
Table F-1	PIC Pin Variables and Functions.....	125
Table H-1	Command Classes in SPI Mode.....	150
Table H-2	SPI Bus command description.....	151

LIST OF FIGURES

Figure 1-1	Vertical and Horizontal Arrangements of Spheres.....	3
Figure 1-2	A Typical Microprocessor-based Open-loop Control.....	8
Figure 1-3	Block diagram of a closed-loop control of a stepping motor.....	9
Figure 1-4	Generalized positioning control system.....	10
Figure 1-5	Basic rotary photo electric transducer.....	12
Figure 1-6	Output signals of a photo electric encoder.....	12
Figure 1-7	Magnetic Encoder.....	13
Figure 2-1	Step motor drive circuit scheme.....	16
Figure 2-2	A typical torque-speed characteristics of a stepping motor.....	18
Figure 2-3	Bidirectional BDC motor drive (H-BRIDGE) circuit.....	20
Figure 2-4	Speed – Torque Characteristic of PMDC motor.....	23
Figure 2-5	AC Servo motor control block diagram.....	24
Figure 2-6	A typical torque-speed Characteristic of AC servo motor.....	26
Figure 3-1	General Overview of the Design.....	27
Figure 3-2	Sphere Gap Setting.....	28
Figure 3-3	3-axes Position and Motion Controller.....	29
Figure 3-4	XY Positioning Table.....	29
Figure 3-5	PIC 16F877 Block Diagram.....	32
Figure 3-6	Harvard vs. von Neumann Block Architectures.....	33
Figure 3-7	Clock/Instruction Cycle.....	34
Figure 3-8	Instruction Pipeline Flow.....	35
Figure 3-9	Operation of the ALU and W Register.....	36
Figure 3-10	Status Register	37
Figure 3-11	Architectural Program Memory Map and Stack.....	38
Figure 3-12	Loading of PC in Different Situations.....	39
Figure 3-13	PIC16F877 Pin Diagram.....	41
Figure 3-14	SSPSTAT: Synchronous Serial Port Status Register.....	44
Figure 3-15	SSPCON: Synchronous Serial Port Control Register.....	44
Figure 3-16	USB connector Types.....	47
Figure 3-17	Full and Low Speed Device with pull up resistor	48
Figure 3-18	Block diagram of DLP-IO26.....	53
Figure 3-19	The interface between FT8U245AM and Microcontroller.....	54
Figure 3-20	DLP-IO26 40 pin header.....	56
Figure 3-21	LM628 and LM629 Typical System Block Diagram.....	57
Figure 3-22	Hardware Architecture of LM629.....	59
Figure 3-23	Typical Velocity Profiles.....	60
Figure 3-24	Position, Velocity and Acceleration Registers.....	61
Figure 3-25	LM629 PID control block diagram.....	61

Figure 3-26	PWM Output Signal Format	63
Figure 3-27	Status Byte Bit Allocation.....	66
Figure 3-28	Interrupt Mask/Reset Bit Allocations.....	67
Figure 3-29	Filter Control Word Bit Allocation.....	68
Figure 3-30	Trajectory Control Word Bit Allocation.....	70
Figure 3-31	MMC Pinout and pad assignment.....	72
Figure 3-32	MultiMediaCard/RS-MultiMediaCard Block Diagram.....	72
Figure 3-33	Data Transfer Formats.....	73
Figure 3-34	MMC Read and Write operations.....	74
Figure 3-35	I2C EEPROM Pinout diagram and Pin functions.....	75
Figure 3-36	I2C Device Address Format.....	75
Figure 3-37	I2C Write Operation.....	76
Figure 3-38	I2C Read Operation.....	77
Figure 3-39	MMC Data Storage Circuit.....	78
Figure 3-40	X axis Motor Driver Circuit.....	80
Figure 3-41	Buffer and line control circuit of X axis.....	82
Figure 4-1	Trajectory Example.....	84
Figure 4-2	Programming Device.....	87
Figure 4-3	General Overview of the Firmware.....	88
Figure 4-4	Flow Chart of Data Storage Part.....	90
Figure 4-5	Flow Chart of LM629 Part.....	93
Figure 4-6	Flow Chart of Data Processing Part.....	96
Figure B-1	FT8U245AM Block Diagram.....	110
Figure B-2	DLP-IO26 Schematic Diagram.....	113
Figure C-1	Functional Block Diagram of LMD18201.....	115
Figure D-1	Logic and Connection Diagram of LS245.....	116
Figure D-2	Pinout and Logic Diagram of LS541.....	117

CHAPTER 1

INTRODUCTION

In this study, the developed position and motion control system is used in two applications:

- 1) Controlling the sphere gap setting for measurement of high voltages.
- 2) Controlling the motion and position of a three axes machine.

1.1 Sphere Gaps

Sphere gaps are commonly used for the measurement of the peak value of high voltages and as a result of extensive investigations calibration tables giving breakdown voltages corresponding to various gap lengths for different sizes of spheres have been prepared.

The Table 1-1 gives the breakdown voltages for a standard sphere gap of the specified sphere size and spacing. The standard accuracy is $\pm 3\%$ for gaps less than half the sphere diameter and 5% for gaps larger than that. The Table 1-1 is for a gap with one terminal grounded, typically a vertical gap. There are two columns for each sphere size depending on whether it is a positive or negative impulse. The column A shows AC, DC, either polarity and full negative standard impulse voltages (one sphere grounded) and the column B shows positive polarity standard impulse voltages and impulse voltages with long tails

As the field gets more uniform (i.e. the gap is a smaller fraction of the sphere diameter), the difference in voltage for the two polarities becomes less. In general, voltages for spheres larger than the largest for which there is a voltage will be the same. That is, 200 cm spheres with a 0.5 cm gap will breakdown at the same voltage as a 15 cm sphere at the same spacing.

Table 1-1: Sphere gap with one terminal grounded breakdown voltages (in KV) at Standart Temperature and pressure (760 mmHg , 20 °C)

Gap Spacing cm	Sphere Diameter (cm)															
	5		10		15		25		50		100		150		200	
	A	B	A	B	A	B	A	B	A	B	A	B	A	B	A	B
0.5	17.4		16.9	16.8	16.9											
1.0	32.0		31.7		31.4		31.2	31.4								
1.5	44.7	45.5	44.7	45.1	44.7	45.1	44.7									
2.0	57.5	58.0	58.0		58.0		58.0									
2.5			71.5		71.5		71.5	71.5								
3.0			85		85		85	85								
3.5			95.5	96	97		97	97								
4.0			106	108	108	110	110	110								
5.0			123	127	127	132	135	136	136							
7.5					181	187	195	196	199	199						
10.0							257	268	259	259	262		262		262	
12.5							277	294	315	317						
15.0							309	331	367	374	383	384	384		384	
17.5							336	362	413	425						
20.0									452	472	500		500		500	
25.0									520	545	605	610				
30.0									575	610	700	715	730	735	735	740
35.0									725	755	785	800				
40.0											862	885	940	950	960	965
45.0											925	965				
50.0											1000	1020	1110	1130	1160	1170
75.0											1210	1260	1420	1460	1510	1590
100.0															1870	1900

For sphere s of particular diameter, the field in the gap becomes less uniform as the gap length increases from a sphere radius to a sphere diameter. It is recommended that the spacing should not exceed a sphere radius.

In general the spheres are mounted with the axis of the sphere gap vertical and the lower sphere is earthed as shown in Figure 1-1. When measurement of a symmetrical applied voltage has to be made, the spheres are arranged horizontally and both the spheres are insulated. The breakdown voltage characteristic will then be slightly different.

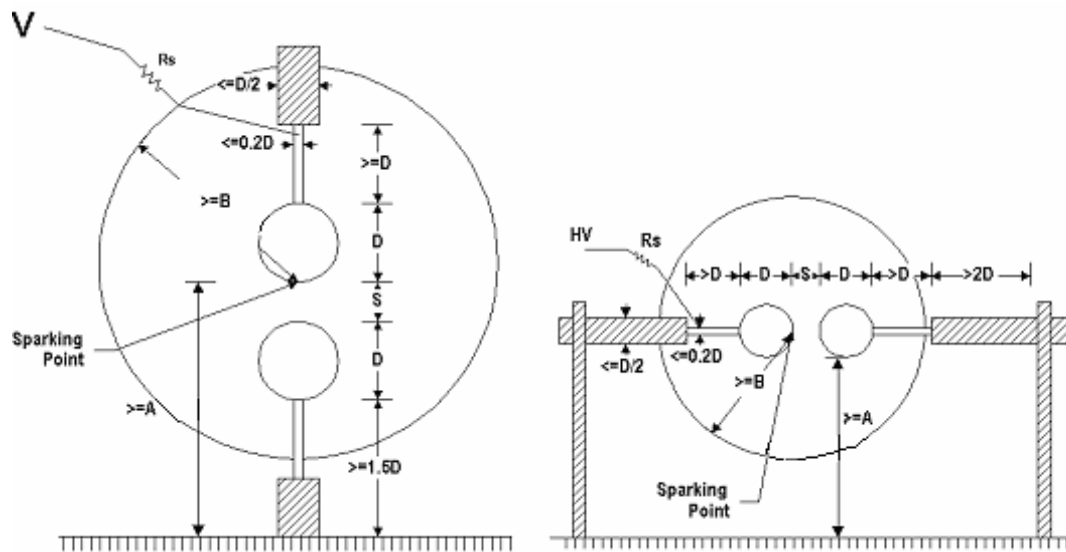


Figure 1-1: Vertical and Horizontal Arrangements of Spheres

In the figure 1-1, A is the height of the spark point above earth plane, B is the radius of space free from external structures and Rs is the series resistor.

The relationship between applied voltage to spheres, diameter of spheres and gaps is given by the below equation.

$$U = 27,75 \left(1 + \frac{0,757}{\sqrt{\delta \cdot S}} \right) \left(\frac{1}{K} \cdot \frac{D}{S} \right) \cdot S \quad (1.1)$$

U: applied voltage, KV

D: Sphere diameter, m

S: Sphere Gap, m

$$\delta = 0.386 \frac{P}{T} \quad (1.2)$$

P is air pressure, T is air temperature

K: Geometrical parameter depends upon the ratio of gap spacing to sphere radius.

The factors that affect the high voltage measurements with sphere gap:

- Standard sphere gaps
- Tolerances on size, shape and state of surface
- Construction of the shanks of the sphere
- Height of the spheres above the horizontal ground plane
- Clearance around the spheres
- High voltage conductor
- Protective series resistance in the measurement of alternating, direct and impulse voltages
- Irradiation
- Correction factors for varying atmospheric conditions

Arranging the distance between the spheres is very difficult and time consuming process. As described above for accurate measuring of high voltage, the sphere gap distance must be very precise. In modern applications, a remote controlled position controller system is used to control the sphere gap to save time and take measure to high voltage's lethal hazard.

1.2 Computer Numerical Control (CNC) Application

The operation of mechanical devices requires some form of control, whether manual, automatic, through a computer program, or by remote control. Manufacturing machines need control systems that primarily repeat well-defined movements with high precision and in a minimum of time, permitting the mass production of products of uniform quality with a minimum human intervention. [5]

For mass production such automated machinery is frequently arranged sequentially so that, at each stage, an additional operation or operations may be carried out. This type of manufacture is the most efficient yet devised.

A control system which has been designed according to traditional principle is referred to as a mechanical, electrical, pneumatic, or hydraulic control, depending on classification of its main components. These controls have a major disadvantage in that their sequence of operations is fixed. Changes in operational sequence due to product changes cause considerable downtime while the electrical circuitry and mechanical elements of the control are being changed to achieve the new control characteristics required. A major portion of setup time is needed to effect adjustment of the control devices.

For these reasons, machines with traditional controls are not appropriate for flexible manufacturing. A new control concept has been developed, which meets the following criteria:

- No manual intervention necessary during the machining process
- Storage of part programs that are quickly retrievable
- Precisely defined and simultaneous movements of as many machine axes as possible
- Quickchange tooling and autochange feed and spindle speed facilities

This new controller concept demands programmable machines that offer rapid and error-free response to changing manufacturing demands and which are “controlled by numbers,” as all information is supplied in digital format. [5]

Numeric Control (NC) is the technique of giving instructions to a machine in the format of a code which consists of numbers, letters, punctuation marks and other symbols. All NC machine responds to this coded information in precise and ordered manner to carry out various machining functions. These functions may range from the positioning of the machine spindle relative to the workpiece, to controlling the speed and direction of spindle rotation, tool selection, and so on. [6]

The combination of all this numerical information in a sequence understood by the machine tool's controller is called part program, and the process of creating data in a correctly structured format is called programming. [5]

Automatic control of NC machine tools relies on the presence of the part program in a form that is external to the machine itself. The NC machine does not possess any memory of its own and is only capable of executing a single block of information, fed to it, at a time. For this reason part program is stored on punched tape. The machine control unit (MCU) will read a block of information, then execute that block, read the next block of information and execute that block, and so on.

Since a part cannot be produced automatically without a tape being run through the machine block by block, they are often referred to as tape controlled machines. The production of repetitive, identical parts thus relies on the tape being present and the tape being in good condition. [6]

Computer Numerical Control (CNC) retains the fundamental concepts of NC but utilizes a dedicated stored-program computer within the machine control unit. CNC is largely the result of technological progress in microelectronics, rather than any radical departure in the concept of NC. CNC attempts to accomplish as many MCU functions as possible within the computer software which is programmed into the computerized control unit. This greatly simplifies the CNC hardware, significantly lowers purchase costs, and improves reliability and maintainability.

Updates and upgrades are relatively simple. In many cases it is only the stored operating program that needs to be modified.

CNC control units, like the computers on which they are based, operate according to stored program held in computer memory. This means that part programs are now able to become totally resident within the memory of the control unit, prior to their execution. No longer do the machines have to operate on the “read-block/execute-block” principle. This eliminates the dependency on slow and often unreliable, tapes and tape reading devices. [6]

1.2.1 CNC Machine Tools

Many CNC machine tools still retain many of the constructional and physical design aspects of their NC counterparts. However, many new control features are made available on CNC machines, which were impractical or uneconomical to implement on early NC machines [6]. Such new features include:

- Stored programs
- Editing facilities of part programs
- Stored patterns
- Sub-programs for repetitive machining sequences
- Enhanced cutter compensation
- Optimize machining conditions
- Communication facilities
- Diagnostic
- Management information

1.3 Control Systems

Proper control of the machine tool motions is an essential requirement of any CNC machining application. Without it we would still be reliant on the manual skills of the conventional machine tool operator. A control system may be defined as; One or more interconnected devices which work together to automatically maintain or alter the condition of the machine tool in a prescribed manner.

Such a system may be mechanical, electrical, electronic, hydraulic or pneumatic. In practice, many control systems are combinations of these and are termed hybrid systems. [6]

In theory, an input signal is generated in response to an inputted program command. This produces an output signal which turns a motor, which then moves the machine tool slide.

One important distinction that must be made in relation to control systems is between open loop and closed loop operation.

1.3.1 Open-Loop Control

In a control system, when the output quantity has no effect on the input quantity the system is called open loop control system. The adoption of open loop control requires very careful consideration since any change in external conditions may cause the output of the system to fluctuate, or drift.

The open-loop control scheme has the advantages simplicity and low cost. A typical microprocessor-based open-loop motor control system is shown in Figure 1-2. As seen in Figure 1-2, the digital phase control signals are generated by the microprocessor and amplified by the drive circuit before being applied to motor.

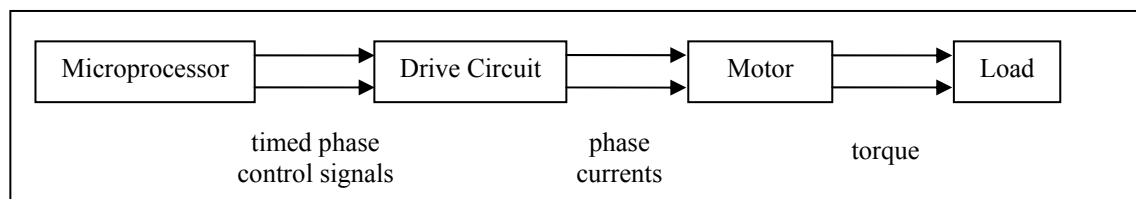


Figure 1-2: A Typical Microprocessor-based Open-loop Control

In an open-loop system there is no feedback of load position to the controller and therefore it is imperative that the motor responds correctly to each excitation change. If the excitation changes are made too quickly the motor is unable to move the load to the new demanded position and so

there is a permanent error in the actual load position compared to the expected one. Also it is important that in the applications where the load is likely to fluctuate the timings must be set for the worst conditions, i.e. the largest load, and the control scheme is then non-optimal for all other loads. As there is no feedback in this type of control, the need for expensive sensing and feedback devices such as optical encoders are eliminated. [8, 18]

1.3.2 Closed-Loop Control

When a feedback loop is introduced into the control system, the output quantity is having effect on the input quantity. This control system is classified as closed loop control system. A closed loop system can overcome the difficulties met in an open-loop control system by using feedback

A block diagram illustrating a closed-loop control scheme of a motor is shown in Figure 1-3. With the closed loop control, one not only determines the proper positions of the load, but more achieves much higher speeds and more stability in speed. Each command is issued only when the motor has responded satisfactorily to the previous command and so there is no possibility of the motor losing synchronism.

The closed loop systems, of necessity, require more component parts, and extra control circuitry, in order for them to perform the feedback function, and they are consequently more complex than their open-loop counterparts. This inevitably means higher costs for the design and implementation of closed loop systems in CNC machine tools. [8, 18]

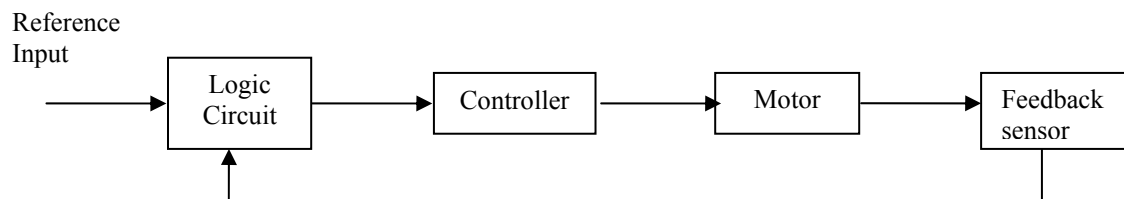


Figure1-3: Block diagram of a closed-loop control of a stepping motor

1.1.1 Position Control Systems

The degree of accuracy in positioning of machine members is directly related to the characteristics of the principal system components, which are data input, position transducer, amplifier and controlled drive elements. Figure 1-4 shows schematically generalized positioning control system. A data input device provides a signal to the input section of a comparing device. And a position feedback is provided by position transducer to the comparing device. An error signal corresponding to the difference between the data input and the physical position of the controlled members exists as an output to the amplifier block.

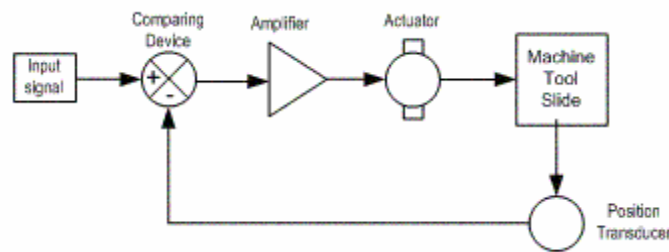


Figure 1-4: Generalized positioning control system

Appropriate amplification, dictated by the system's accuracy and repeatability requirements, provides excitation for the machine control actuator. Motion of the actuator shaft is converted to appropriate machine member position by means of gearing, or by lead screws. Other positioning systems use hydraulic or pneumatic transducers, amplifiers, and/or linear actuators.

1.1.1.1 Position Transducers

A transducer is a device that converts energy in one form into energy in another form. For example tachogenerator is a transducer that converts angular velocity into voltage. Basically three types of position measuring transducer are employed on CNC machine tools: INDUCTIVE, PHOTOELECTRIC and MAGNETIC transducers.

Inductive Transducers

Resolvers, synchro and inductosyn scales are belong to this category. These are old technology and replaced by photoelectric and magnetic transducers.

The synchro transducer utilizes the principle of magnetic induction. Its rotor is attached to lead screw and the stator windings are supplied with constant voltage which sets up the magnetic field around them. As the rotor rotates, with the lead screw a voltage is induced in the rotor winding. This induced voltage varies from a minimum to a maximum, depending on its position, in a sinusoidal fashion. The magnitude of this voltage represents the angular position of rotor. [6]

The inductosyn operates on the synchro resolver principle but is effectively laid out flat. The amplitude and phase shift of the induced voltage via the linear motion depends on the relative position of two elements, that is, the fixed scale windings corresponding to rotor windings, and the sliding scale windings corresponding to stator windings. [5]

Photoelectric Transducers

The most popular type of transducer is the photoelectric transducer, which consists of a linear or radial grating, a light source, and a photo detector (light sensor).

Photoelectric systems are available in both linear and rotary forms. The rotary transducers are mounted on the rotating shaft of the motor or on the end of the lead screw. In the rotary transducer, LED provides a light source on one side of the wheel and a photo transistor detects light on the other side of the wheel (see Figure 1-5). Light passing through the slots in the wheel will turn the photo transistor on. As the shaft turns, the photo transistor turns on and off with the passing of the slots in the wheel. The frequency at which the transistor toggles is an indication of motor speed. In the case of positioning applications, an optical encoder will also provide feedback as to the position of the motor. [7, 9, 10]

The operating principle is same in the linear encoder. The instrument is attached to the slide of a moving table. This type of transducers measures the actual movement of the carriage because they move with the carriage.

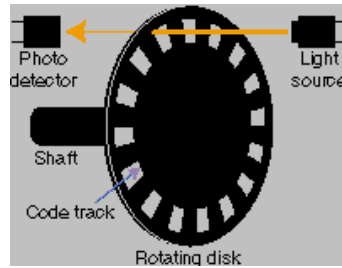


Figure 1-5: Basic rotary photo electric transducer

The photoelectric transducers are use two output channels (A and B) to sense position. Using two code tracks with sectors positioned 90 degrees out of phase; the two output channels indicate both position and direction of rotation. If A leads B, for example, the disk is rotating in a clockwise direction. If B leads A, then the disk is rotating in a counter-clockwise direction. By monitoring both the number of pulses and the relative phase of signals A and B, you can track both the position and direction of rotation. Some transducers also include a third output channel, called a zero or index or reference signal, which supplies a single pulse per revolution. This single pulse is used for precise determination of a reference position. An example for the output signals can be seen in Figure 1-6.

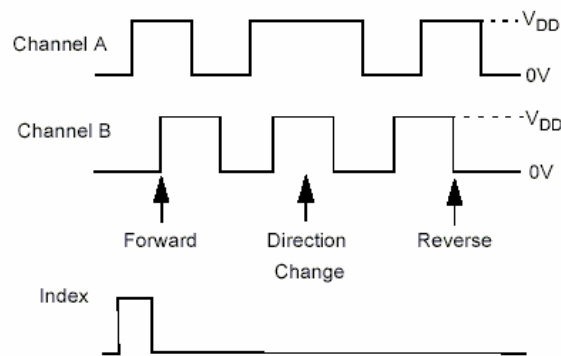


Figure 1-6: Output signals of a photo electric encoder

Magnetic Transducer

A magnetic encoder consists of a rotating gear made of ferrous metal and a magnetic pick-up that contains a permanent magnet and the sensing element. The gear, which is mounted on the rotating shaft, has precisely machined teeth that provide the code pattern. As the disk rotates, these teeth disturb the magnetic flux emitted by the permanent magnet, causing the flux field to expand and collapse. These changes in the field are sensed by the sensing element, which generates a corresponding digital or pulse signal output. [9, 10] Two kinds of magnetic pick-ups exist:

- **Hall effect:** pick-ups use a semiconducting sensing element that relies on the Hall effect to generate a pulse for every gear tooth that passes the pickup
- **Variable reluctance:** pick-ups use a simple coil of wire in the magnetic field. As the gear teeth pass by the pick-up and disturb the flux, they cause a change in the reluctance of the gear/magnet system. This induces a voltage pulse in the sensing coil that is proportional to the rate flux change.

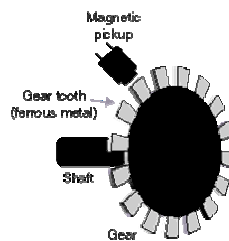


Figure 1-7: Magnetic Encoder

CHAPTER 2

CONTROL MOTOR TYPES

High-precision control of electrical motors plays a critical role in many industrial applications, such as CNC machine tools, automation, robotics, flexible manufacturing systems. Among many electric motors, step and servo motors are widely used.

2.1 Step Motors

A stepping motor is a permanent magnet or variable reluctance dc motor that has the following performance characteristics:

- rotation in both directions,
- precision angular incremental changes,
- repetition of accurate motion or velocity profiles,
- a holding torque at zero speed, and
- Capability for digital control.

It is an electromechanical device which converts electrical pulses into discrete mechanical movements. Basically, it is a synchronous motor with the magnetic field electronically switched to rotate the armature magnet around. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence.

The number and rate of the pulses control the position and speed of the motor shaft. The motor rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the direction of motor shafts rotation. The speed of the

motor shaft's rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied

Theoretically, a stepping motor is a marvel in simplicity. They are very reliable at low cost, since there are no brushes or contacts in the motor. Therefore the life of the motor is simply dependent on the life of the bearing.

Stepping motors have precise positioning and repeatability of movement since good stepper motors have an accuracy of 3 – 5% of a step and this error is non cumulative from one step to the next. They give excellent responses to starting/stopping/reversing actions and it is possible to achieve very low speed synchronous rotation with a load that is directly coupled to the shaft. Also, there is a wide range of rotational speeds that can be realized as the speed is proportional to the frequency of the input pulses. Besides all these advantages, resonances that can occur and the difficulty of operation at high speeds are the disadvantages of using a stepping motor.

Generally, stepping motors produce less than one horsepower (746W) and therefore they are frequently used in low-power position control applications. A stepper motor can be a good choice whenever controlled movement is required. They can be used to advantage in applications where you need to control rotation angle, speed, position and synchronism.

There are three basic types of stepping motors: permanent magnet, variable reluctance and hybrid. They differ in terms of construction based on the use of permanent magnets and/or iron rotors with laminated steel stators. The type of the motor determines the type of the circuit driver and the type of the translator to be used.

Permanent Magnet Motors: The permanent magnet (PM) motor has, as the name implies, a PM rotor. It is a relatively low speed, low torque device with large step angles of either 45 or 90 degrees. Its simple construction and low cost make it an ideal choice for non industrial applications. Unlike the other stepping motors, the permanent magnet motor's rotor has no teeth and is designed to be magnetized at a right angle to its axis.

Variable Reluctance Motors: The variable reluctance motor does not use a permanent magnet. As a result, the motor rotor can move without constraint or “detent” torque. This type of

construction is good in non industrial applications that do not require a high degree of motor torque, such as the positioning of a micro slide.

Hybrid Motors: Hybrid motors combine the best characteristics of the variable reluctance and permanent magnet motors. They are constructed with multi-toothed stator poles and a permanent magnet rotor. Standard hybrid motors have two hundred rotor teeth and rotate at 1.80 step angles. As they exhibit high static and dynamic torque and run at very high step rates, hybrid motors are used in a wide variety of industrial applications.

2.1.1 Drive Circuits

The stepper motor driver receives low-level signals from the indexer or control system and converts them into electrical (step) pulses to run the motor. One step pulse is required for every step of the motor shaft. This process is shown in Figure 2-1.

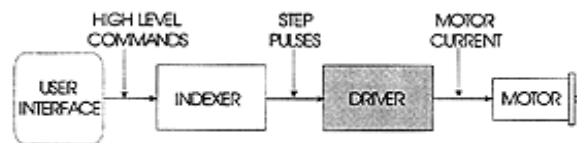


Figure 2-1: Step motor drive circuit scheme

Speed and torque performance of the step motor is based on the flow of current from the driver to the motor winding. The factor that inhibits the flow, or limits the time it takes for the current to energize the winding, is known as inductance. The lower the inductance, the faster the current gets to the winding and the better the performance of the motor. To reduce inductance, most types of driver circuits are designed to supply a greater amount of voltage than the motors rated voltage. [18]

The stepper motor driver circuit has two major tasks:

- To change the current and flux direction in the phase windings

- To drive a controllable amount of current through the windings, and enabling as short current rise and fall times as possible for good high speed performance.

2.1.1.1 Flux Direction Control

Stepping of the stepper motor requires a change of the flux direction independently in each phase. The direction change is done by changing the current direction. It may be done in two different ways, using a bipolar or a unipolar drive.

Bipolar drive refers to the principle where the current direction in one winding is changed by shifting the voltage polarity across the winding terminals. The bipolar drive method requires one winding per phase.

The unipolar drive principle requires a winding with a center-tap or two separate windings per phase. Flux direction is reversed by moving the current from one half of the winding to the other half. This method requires only two switches per phase. On the other hand, the unipolar drive utilizes only half the available copper volume of the winding. Power loss in the winding is therefore twice the loss of a bipolar drive at the same output power. [15]

2.1.1.2 Current Control

To control the torque as well as to limit the power dissipation in the winding resistance, the current must be controlled or limited. Furthermore, when half stepping a zero current level is needed, while microstepping requires a continuously variable current. There are two principles to limit the current, the resistance limited drive and the chopper drive. Any of the methods may be realized as a bipolar or unipolar driver. [15]

2.1.2 Torque-Speed Characteristics

The torque-speed characteristics, that show the maximum torque which the motor can develop at each operating speed, are the key to selecting the right motor and drive method for a specific application. These characteristics are dependent upon the motor, excitation mode and type of driver or drive method. A typical torque-speed characteristic of a stepping motor is given in Figure 2

In which there are pull-in and pull-out curves. The former defines an area referred to as the start stop region and this is the maximum frequency at which the motor can start/stop instantaneously, with a load applied, without loss of synchronism while the latter defines an area referred to as the slew region and it defines the maximum frequency at which the motor can operate without losing synchronism. Since this region is outside the pull-in area the motor must accelerated or decelerated into this region.

The pull-in characteristics vary also depending on the load as having the larger the load inertia the smaller the pull-in area. From the shape of the curve, it can be seen that the step rate affects the torque output capability of stepper motor. The decreasing torque output as the speed increases is caused by the fact that at high speeds the inductance of the motor is the dominant circuit element. The shape of the speed - torque curve can change quite dramatically depending on the type of driver used. [18]

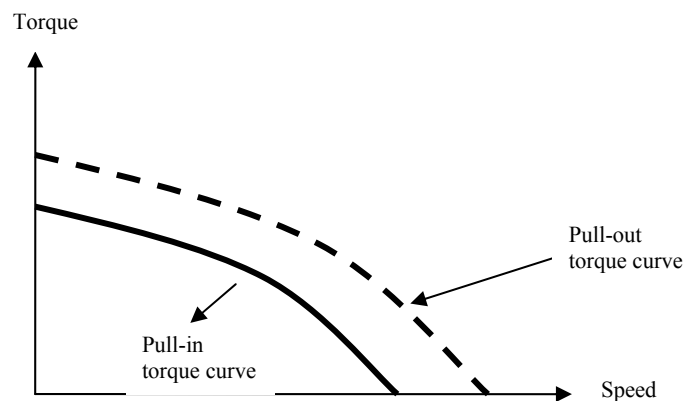


Figure 2-2: A typical torque-speed characteristics of a stepping motor

2.2 Servo Motors

A servo motor is an AC or DC powered motor that uses feedback and controllers to achieve a specific angular location. Servo motors can achieve the same levels of accuracy as stepping motors, but must use a closed loop feedback system whereas stepping motors operate using open loop systems.

A characteristic of servo motors that is appealing is their speed. When comparing servo motor speed to stepping motor speed, the servo motors are much faster. Their speed offsets their greater costs due to the control system that must accompany them. Without the use of an accompanying control system, servo motors would have very little precision, which is their only drawback. [19]

2.2.1 DC Servo Motors

DC servo motors are Permanent magnet DC (PMDC) servo motors with brushes, widely used in applications ranging from CNC machine tools, automation, and robotics. PMDC are inexpensive, easy to drive, easy to control speed and torque and are readily available in all sizes and shapes.

2.2.1.1 Drive circuit

Drive circuits are used in applications where a controller of some kind is being used and speed control is required. The purpose of a drive circuit is to give the controller a way to vary the current in the windings of the BDC (brushed dc) motor. The common drive circuits allow the controller to pulse width modulate the voltage supplied to a BDC motor. In terms of power consumption, this method of speed control is a far more efficient way to vary the speed of a BDC motor compared to traditional analog control methods. [12]

Bidirectional control of a BDC motor requires a circuit called an H-bridge. The H-bridge, named for its schematic appearance, is able to move current in either direction through the motor winding.

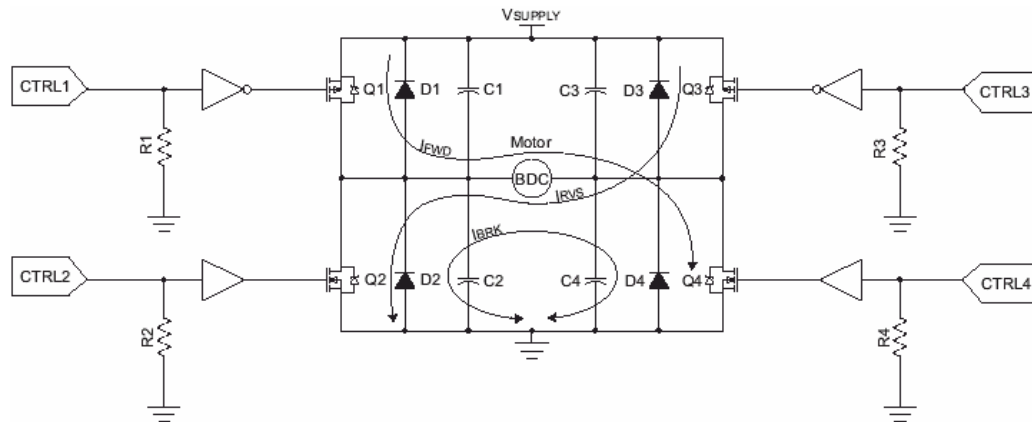


Figure 2-3: Bidirectional BDC motor drive (H-BRIDGE) circuit

The different drive modes for an H-bridge circuit are shown in Table 2-1. In Forward mode and Reverse mode one side of the bridge is held at ground potential and the other side at V_{SUPPLY} . In Figure 2-3 the IFWD and IRVS arrows illustrate the current paths during the Forward and Reverse modes of operation. In Coast mode, the ends of the motor winding are left floating and the motor coasts to a stop. Brake mode is used to rapidly stop the BDC motor. In Brake mode, the ends of the motor are grounded. The motor behaves as a generator when it is rotating. Shorting the leads of the motor acts as a load of infinite magnitude bringing the motor to a rapid halt. The IBRK arrow illustrates this.

Table 2-1: H-BRIDGE modes of operation

	Q1 (CTRL1)	Q2 (CTRL2)	Q3 (CTRL3)	Q4 (CTRL4)
Forward	on	off	off	on
Reverse	off	on	on	off
Coast	off	off	off	off
Brake	off	on	off	on

2.2.1.2 Speed Control

The speed of a BDC motor is proportional to the voltage applied to the motor. When using digital control, a pulse-width modulated (PWM) signal is used to generate an average voltage. The motor winding acts as a low pass filter so a PWM waveform of sufficient frequency will generate a stable current in the motor winding. The relation between average voltage, the supply voltage, and duty cycle is given by:

$$V_{\text{AVERAGE}} = D \times V_{\text{SUPPLY}} \quad (2.1)$$

Speed and duty cycle are proportional to one another. For example, if a BDC motor is rated to turn at 15000 RPM at 12V, the motor will (ideally) turn at 7500 RPM when a 50% duty cycle waveform is applied across the motor.

Feedback Mechanisms

Though the speed of a BDC motor is generally proportional to duty cycle, no motor is ideal. Heat, commutator wear and load all affect the speed of a motor. In systems where precise speed control is required, some sort of feedback mechanism must be included in the system.

Speed feedback is implemented in one of two ways. The first involves the use of a speed sensor of some kind. The second is back electro magnetic flux (BEMF) control method which uses the BEMF voltage generated by the motor.

Sensored Feedback

There are a variety of sensors used for speed feedback. The most common are optical encoders and Hall Effect sensors, described in section 1.3.3.1.

Back Electro Magnetic Flux (BEMF) Control Method

Another form of velocity feedback for a BDC motor is BEMF voltage measurement. BEMF voltage and speed are proportional to one another. A voltage divider is used to drop the BEMF voltage into the 0-5V range so that it can be read by an analog-to-digital converter. The BEMF voltage is measured between PWM pulses when one side of the motor is floating and the other

is grounded. At this instance in time the motor is acting like a generator and produces a BEMF voltage proportional to speed. [12]

2.2.1.3 Torque-Speed Characteristics

Torque-Speed Characteristic of a dc motor can be determined from two main equations and their connection diagrams.

$$T = K_a \cdot \Phi_d \cdot I_a \quad (2.2)$$

$$E_a = K_a \cdot \Phi_d \cdot W_m \quad (2.3)$$

T = Torque,

K_a = Winding constant,

Φ_d = Direct axis flux per pole,

I_a = Armature current,

E_a = armature voltage,

W_m = Speed of rotation

The PMDC motor has a fixed source of field-winding flux which is supplied by permanent magnet. Consequently, increased torque must be accompanied by a very nearly proportional increase in armature current. These motors respond to changes in voltage very quickly because the stator field is always constant. The PMDC motor is substantially a constant speed motor. The speed is relatively independent of load. [16]

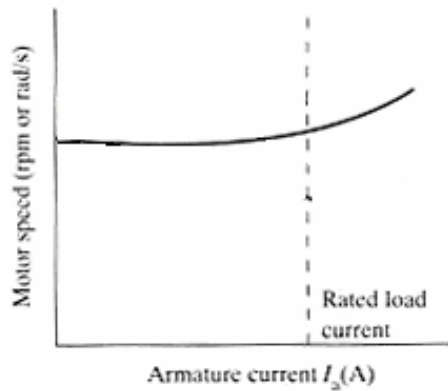


Figure 2-4: Speed – Torque Characteristic of PMDC motor

2.2.2 AC Servo Motor

AC servo motors are polyphase synchronous motors with permanent magnet rotors, referred to as brushless dc motors (BLDC) [16]. The AC servo motors can be used in any application where torque, speed or position control is required. AC servo are electronically commutated. These motors have many advantages over brushed DC motors. A few of these are:

- Better speed versus torque characteristics
- High dynamic response
- High efficiency
- Long operating life
- Noiseless operation
- Higher speed ranges

2.2.2.1 Construction and Operating Principle

AC servo motors are a type of synchronous motor. This means the magnetic field generated by the stator and the magnetic field generated by the rotor rotates at the same frequency. AC servo motors come in single-phase, 2-phase and 3-phase configurations. Corresponding to its type, the

stator has the same number of windings. Out of these, 3-phase motors are the most popular and widely used.

Unlike a brushed DC motor, the commutation of an AC servo motor is controlled electronically. To rotate these motors, the stator windings should be energized in a sequence. It is important to know the rotor position in order to understand which winding will be energized following the energizing sequence. Rotor position is sensed using Hall effect sensors embedded into the stator.

Based on information supplied by the rotor position sensors, the electronic controller decides which stator phases should be energized at any instant. The controller consists of a set of power electronic devices which are controlled by low-level logic or a microprocessor. [11]

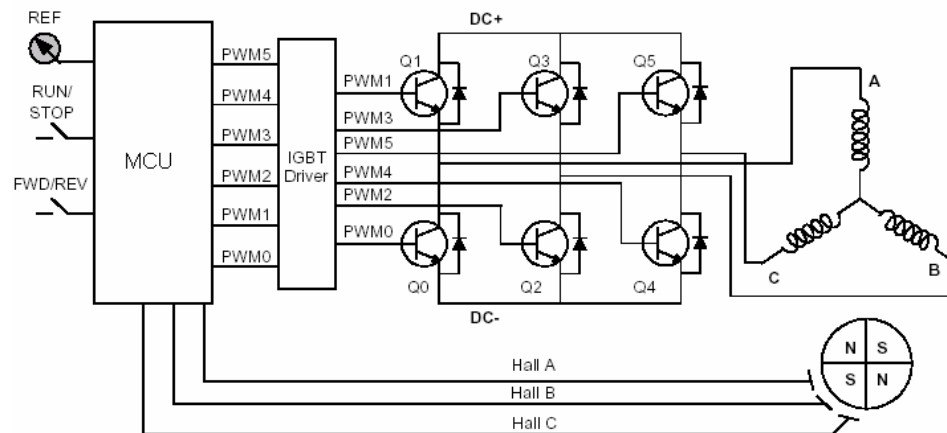


Figure 2-5: AC servo motor control block diagram

2.2.2.2 Torque Control

Torque control is usually accomplished by direct control of armature winding current. The rotor permanent magnets produce a constant flux that links the armature windings. Since output torque is nearly equal to the product of flux and current, it is sufficient to use armature as a measure of output torque. In a control system the current can be sensed and fed back to the

electronics that control the power switching devices. The feedback of current is also used to limit the peak current by chopping the input voltage available at the input of the motor terminals. When negative or braking torque is required, the motor current must be reversed, otherwise the motor would decelerate only due to losses. This requires that the controller be capable of allowing the motor as a generator.[21]

2.2.2.3 Speed Control

Because of the constant armature flux, the speed of a BLDC motor at a steady state condition is proportional to applied armature voltage. For applications requiring speed control it is sufficient to control the voltage applied to the motor terminals by chopping the dc voltage supply. However, this form of open-loop control results in the requirement of some means of current limiting or the current will exceed the component current ratings. This is because the motor back emf is substantially less than the applied voltage. Also, the actual speed achieved will vary with load since there will be no compensation for the resistance voltage drop. A more accurate speed-control system will require a speed feedback from some form of tachometer or rotary transducer.[21]

2.2.2.4 Torque- Speed Characteristic

Figure 2-6 shows an example of torque/speed characteristics. There are two torque parameters used to define a BLDC motor, peak torque (TP) and rated torque (TR). During continuous operations, the motor can be loaded up to the rated torque. As discussed earlier, in a BLDC motor, the torque remains constant for a speed range up to the rated speed. The motor can be run up to the maximum speed, which can be up to 150% of the rated speed, but the torque starts dropping. Applications that have frequent starts and stops and frequent reversals of rotation with load on the motor, demand more torque than the rated torque. This requirement comes for a brief period, especially when the motor starts from a standstill and during acceleration. During this period, extra torque is required to overcome the inertia of the load and the rotor itself. The

motor can deliver a higher torque, maximum up to peak torque, as long as it follows the speed torque curve. [11]

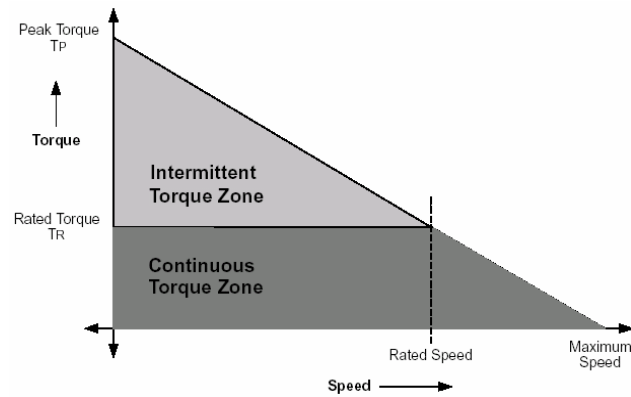


Figure 2-6: A typical torque-speed Characteristic of AC Servo motor

CHAPTER 3

MOTOR CONTROL HARDWARE

Contemporary electrical drives are usually equipped with a motion controller utilizing one or more micro-processors/micro-controllers and/or digital signal processors (DSPs). Such motor drive interfaces are mostly through RS-485 and motor drive control is mostly performed by a dedicated digital-motion controller.

The aim of this thesis is to design and implementation of a DC motor drive system suitable for controlling a sphere gap setting for high voltage measurement, Figure 3-2 and controlling position and motion of a three axes machine, Figure 3-3 and Figure 3-4, requiring high precision motion control. In this study, a three axis motion controller is designed with a microcontroller, a data storage unit and three servo motor controllers. The general overview of the design is shown below.

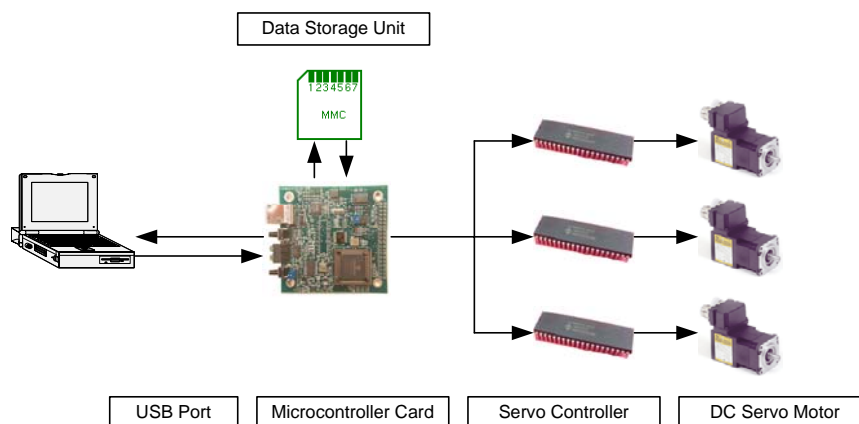


Figure 3.1 General overview of the design



Figure 3-2: Sphere Gap Setting



Figure 3-3: 3-axes Position and Motion Controller



Figure 3-4: XY Positioning Table

3.1 General Overview to PIC Microcontroller

A microcontroller is an inexpensive single-chip computer. The entire computer system lies within the confines of the integrated circuit chip. Primarily, the microcontroller is capable of storing and running a program. It contains a central processing unit (CPU), RAM, ROM, input/output lines, serial and parallel ports, timers, and sometimes other built-in peripherals such as A/D (analog to digital) and D/A (digital to analog) converters.

The microcontroller's ability to store and run unique programs makes it extremely versatile. For example, one can program microcontroller to make decisions based on predetermined situations and selections. The microcontroller's ability to perform math and logic functions allows it to mimic sophisticated logic and electronic circuits. One of the most popular and easy to use microcontroller families available on the market today is the MICROCHIP "PICmicro[®] microcontroller". Originally known as the "PIC" or Peripheral Interface Controller consists of over two hundred variations, each designed to be optimal in different applications. These variations consist of a number of memory configurations, different I/O pin arrangements, amount of support hardware required, packaging and available peripheral functions. [27, 24]

What has made the PICmicro[®] MCU successful is:

- Speed : PIC executes most of its instructions in 0.2 microseconds;
- Programmable timer options
- Instruction set simplicity: the instruction set consist of 35 instructions;
- Integration of operational features: power on reset, brown-out protection, watchdog timer, four clock options and low power options.
- Interrupt control
- Powerful output pin control
- I/O port expansion; The built-in serial peripheral interface can make use of standard 16 pin shift registers parts to add any number of I/O
- Serial programming via two pins
- EPROM/OTP/ROM options
- The availability of excellent, low cost (free) development tools

- The largest and strongest user interne based community of probably any silicon chip family
- An outstanding distributor family
- A wide range of devices with various features that is suitable for any application

PICmicro devices are grouped by the size of their Instruction Word. The three current PICmicro families are:

1. Base-Line: 12-bit Instruction Word length
2. Mid-Range: 14-bit Instruction Word length
3. High-End: 16-bit Instruction Word length

3.1.1 PIC 16F877 Microcontroller

In this study a Mid-range microcontroller 16F877 is used. Figure 3-1 shows the layout of architecture of the PIC 16F877 microcontroller and key features of the PIC 16F877 microcontroller are shown in the Table 3-1 below.

Table 3-1 Basic features of PIC 16F877

Key Features	PIC16F877
Operating Frequency	DC-20Mhz
Resets (and Delays)	POR, BOR(PWRT, OST)
FLASH Program Memory	8K x 14 bit
Data Memory	368 Bytes
EEPROM Data Memory	256 Bytes
Interrupts	14
I/O Ports	Ports A, B, C, D, E
Timers	3
Serial Communication	MSSP, USART
Capture/Compare/PWM	8 input channels
Instruction Set	35 instructions

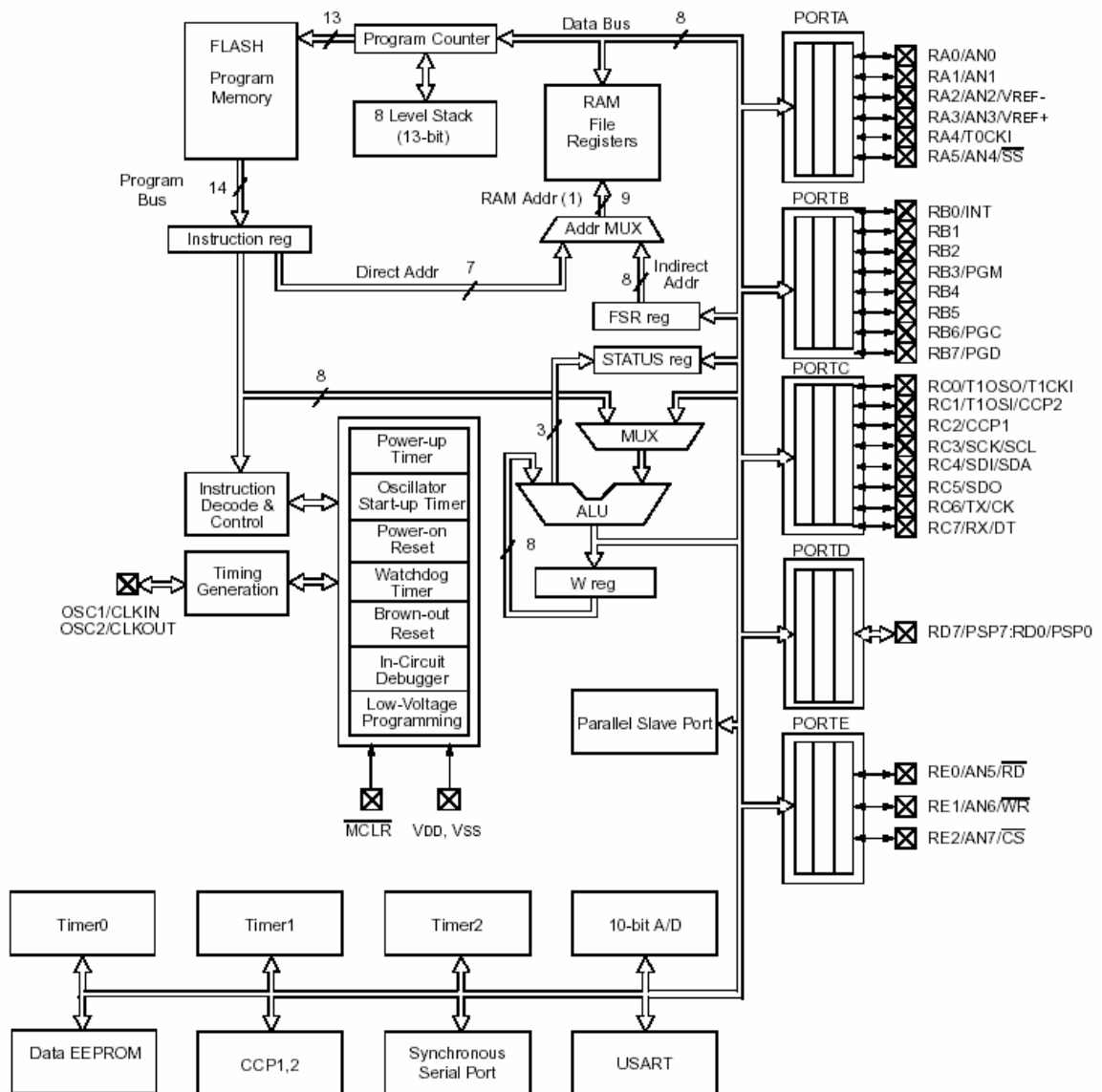


Figure 3.5: PIC 16F877 Block Diagram

3.1.2 General Architecture

PICmicro microcontrollers use RISC instruction set and Harvard architecture to have an exceptionally fast execution speed for a given clock rate.

RISC stands for Reduced Instruction Set Computer. This means that the instruction set that the chip supports is small, lean, and mean. RISC architectures allow for a useful set of instructions that are typically executed in a single clock.

Harvard architecture has the program memory and data memory as separate memories and are accessed from separate buses. This improves bandwidth over traditional von Neumann architecture in which program and data are fetched from the same memory using the same bus. To execute an instruction, a von Neumann machine must make one or more (generally more) accesses across the 8-bit bus to fetch the instruction. Then data may need to be fetched, operated on, and possibly written. As can be seen from this description, that bus can be extremely congested. While with a Harvard architecture, the instruction is fetched in a single instruction cycle (all 14-bits). While the program memory is being accessed, the data memory is on an independent bus and can be read and written. These separated buses allow one instruction to execute while the next instruction is fetched. A comparison of Harvard vs. von-Neumann architectures is shown in Figure 3-6. [23]

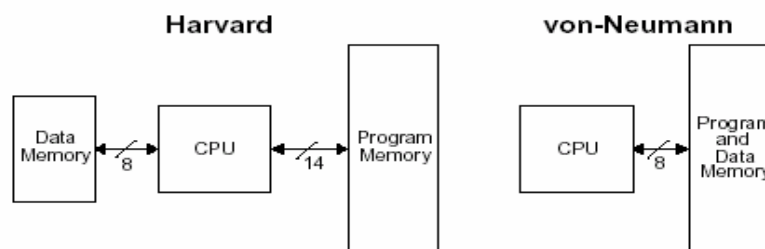


Figure 3-6: Harvard vs. von Neumann Block Architectures

3.1.3 Clocking Scheme/Instruction Cycle

The clock input is internally divided by four to generate four non-overlapping quadrature clocks, namely Q1, Q2, Q3, and Q4. Internally, the program counter (PC) is incremented every Q1, and the instruction is fetched from the program memory and latched into the instruction register in Q4. The instruction is decoded and executed during the following Q1 through Q4. The clocks and instruction execution flow are illustrated in Figure 3-7.

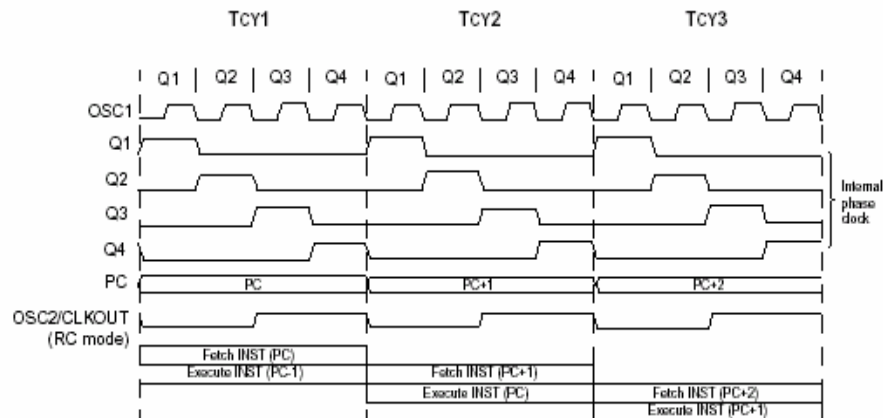


Figure 3-7: Clock/Instruction Cycle

The CPU executes each instruction during the cycle following its fetch, pipelining instruction fetches and instruction executions to achieve the execution of one instruction every cycle. This is illustrated in Figure 3-8. It can be seen that while each instruction requires two cycles (a fetch cycle and followed by a execute cycle), the overlapping of the execute cycle of one instruction with the fetch cycle of the next instruction leads to the execution of a new instructions every cycle.

This lockstep progression is broken whenever an instruction includes a branch operation, as illustrated in Figure 3-8. An instruction is fetched during the fourth cycle, **CALL SUB_1**, whose job is to change the normal flow of instruction fetches from one address to the next address. During the fifth cycle, the CPU carries out the sequential fetch from address 4. At the

end of that fifth cycle, the CPU executes the goto new address instruction by changing the program counter to new address instead of simply incrementing program counter and it ignores the instruction automatically fetched from address 4.

All instructions are single cycle, except for any program branches. These two cycles since the fetch instructions is flushed from the pipeline while the new instruction is being fetched and then executed. [24]

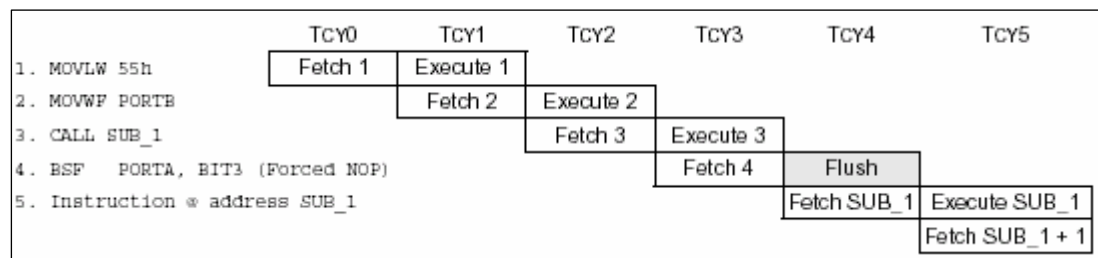


Figure 3-8: Instruction Pipeline Flow

3.1.4 CPU and ALU

The Central Processing Unit (CPU) is responsible for using the information in the program memory (instructions) to control the operation of the device. The CPU controls the program memory address bus, the data memory address bus, and accesses to the stack. Many of these instructions operate on data memory. To operate on data memory, the Arithmetic Logical Unit (ALU) is required. In addition to performing arithmetical and logical operations, the ALU controls status bits (which are found in the STATUS register). The results of some instructions force status bits to a value depending on the state of the result. The machine codes that the CPU recognizes are show in Appendix A

PICmicro MCUs contain an 8-bit ALU and an 8-bit working register. The ALU is a general purpose arithmetic and logical unit. It performs arithmetic and Boolean functions between the data in the working register and any register file.

The ALU is 8-bits wide and is capable of addition, subtraction, shift and logical operations. Unless otherwise mentioned, arithmetic operations are two's complement in nature. In two-operand instructions, typically one operand is the working register (W register). The other operand is a file register or an immediate constant. In single operand instructions, the operand is either the W register or a file register.

The W register is an 8-bit working register used for ALU operations. It is not an addressable register. Depending on the instruction executed, the ALU may affect the values of the Carry (C), Digit Carry (DC), and Zero (Z) bits in the STATUS register. The C and DC bits operate as a borrow bit and a digit borrow out bit, respectively, in subtraction. See the SUBLW and SUBWF instructions for examples.

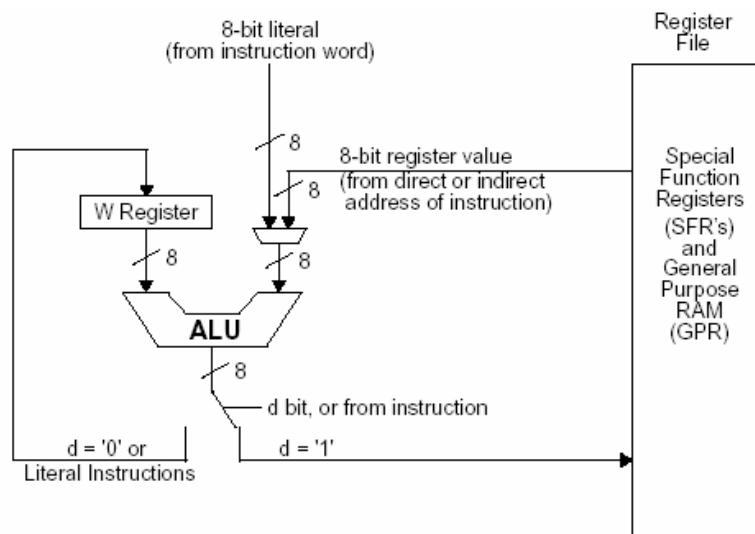


Figure 3-9: Operation of the ALU and W Register

3.1.4.1 CPU Registers

Status Register

The STATUS register, shown in Figure 3-10, contains the arithmetic status of the ALU, the RESET status and the bank select bits for data memory. Since the selection of the Data Memory banks is controlled by this register, it is required to be present in every bank. Also, this register is in the same relative position (offset) in each bank.

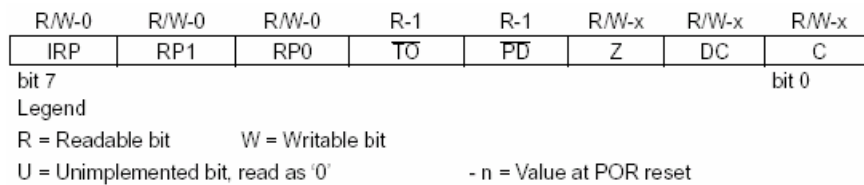


Figure 3-10: Status Register

Bit 7: IRP: Register Bank Select bit (used for indirect addressing)

Bit 6:5: RP1:RP0: Register Bank Select bits (used for direct addressing)

Bit 4: TO: Time-out bit

Bit 3: PD: Power-down bit

Bit 2: Z: Zero bit

Bit 1: DC: Digit carry/borrow bit (ADDWF, ADDLW, SUBLW, SUBWF)

Bit 0: C: Carry/borrow bit (ADDWF, ADDLW, SUBLW, SUBWF instructions)

Working Register

W, the working register, is used by many instructions as the source of operand. It may also serve as the destination for the result of instruction execution. It serves a function similar to that of accumulator in many other microcontrollers.

3.1.5 Program Memory Organization and Program Counter

There are two memory blocks program memory and data memory. Each block has its own bus, so that access to each block can occur during the same oscillator cycle. The data memory can further be broken down into General Purpose RAM and the Special Function Registers (SFRs).

Program memory is addressed word-wise, so the first instruction is located at 0x000, the second at 0x001, etc. Program memory is organized into pages. Memory pages can be selected using the PCLATH register. PIC16F877 microcontrollers have a 13-bit program counter capable of addressing an 8K x 14 program memory space. The width of the program memory bus (instruction word) is 14-bits. Since all instructions are a single word, a device with an 8K x 14 program memory has space for 8K of instructions.

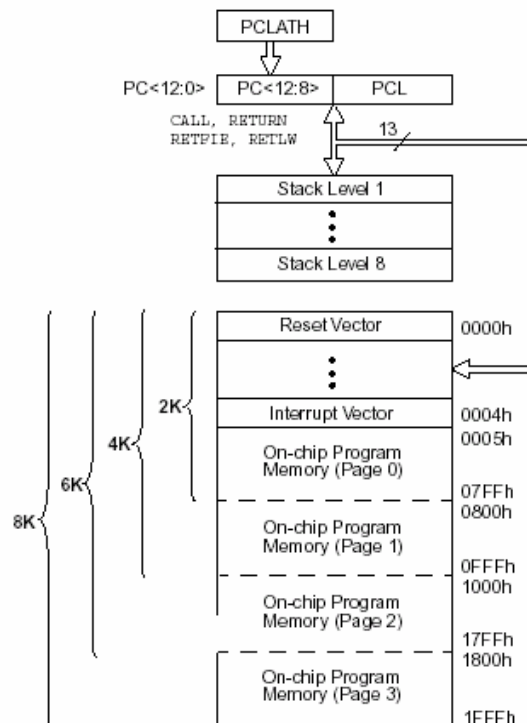


Figure 3-11: Architectural Program Memory Map and Stack

Figure 3-11 shows the program memory map as well as the 8 level deep hardware stack. To jump between the program memory pages, the high bits of the Program Counter (PC) must be modified. This is done by writing the desired value into a SFR called PCLATH (**P**rogram **C**ounter **L**atch **H**igh). If sequential instructions are executed, the program counter will cross the page boundaries without any user intervention

The program counter (PC) specifies the address of the instruction to fetch for execution. The PC is 13-bits wide. The low byte is called the PCL register. This register is readable and writable. The high byte is called the PCH register. This register contains the PC<12:8> bits and is not directly readable or writable. All updates to the PCH register go through the PCLATH register.

Figure 3-12 shows the four situations for the loading of the PC. Situation 1 shows how the PC is loaded on a write to PCL. Situation 2 shows how the PC is loaded during a GOTO instruction. Situation 3 shows how the PC is loaded during a CALL instruction, with the PC loaded (PUSHed) onto the Top of Stack. Situation 4 shows how the PC is loaded during one of the return instructions where the PC loaded (POPed) from the Top of Stack. PCLATH is never updated with the contents of PCH.

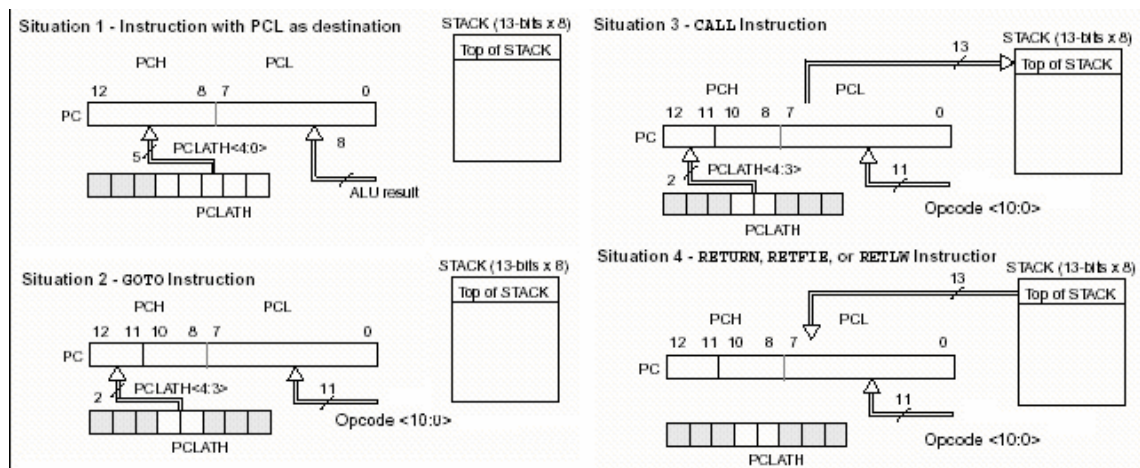


Figure 3-12: Loading of PC in Different Situations

3.1.6 Data Memory

Data Memory is organized as file registers (usually 8 bit) which are both readable and writable. Data RAM can be used to hold data for calculations, temporary variable storage, or user settings. The data memory of the PIC 16F877 microcontroller is partitioned into multiple banks each of which has 128 bytes long. A specific bank can be selected by setting special bits (RP0 and RP1) of the STATUS register. All implemented banks contain Special Function Registers, which may be mirrored in another bank for code reduction and quicker access. Special Function Registers are registers used by CPU and peripheral modules for controlling the desired operation of the device.

Data memory is made up of the Special Function Registers (SFR) area, and the General Purpose Registers (GPR) area. The SFRs control the operation of the device, while GPRs are the general area for data storage and scratch pad operations. The data memory is banked for both the GPR and SFR areas. The GPR area is banked to allow greater than 96 bytes of general purpose RAM to be addressed. SFRs are for the registers that control the peripheral and core functions. Banking requires the use of control bits for bank selection.

3.1.6.1 Banking

The data memory is partitioned into four banks. Each bank contains General Purpose Registers and Special Function Registers. Switching between these banks requires the RP0 and RP1 bits in the STATUS register to be configured for the desired bank when using direct addressing. The IRP bit in the STATUS register is used for indirect addressing.

Table 3-2: Direct and Indirect Addressing of Banks

Accessed Bank	Direct (RP1:RP0)	Indirect (IRP)
0	0 0	0
1	0 1	
2	1 0	1
3	1 1	

3.1.7 Hardware Features

3.1.7.1 I/O Ports

General purpose I/O pins can be considered the simplest of peripherals. They allow the PICmicro™ to monitor and control other devices. To add flexibility and functionality to a device, some pins are multiplexed with an alternate function(s). These functions depend on which peripheral features are on the device. In general, when a peripheral is functioning, that pin may not be used as a general purpose I/O pin.

For most ports, the I/O pin's direction (input or output) is controlled by the data direction register, called the TRIS register. TRIS<x> controls the direction of PORT<x>. A '1' in the TRIS bit corresponds to that pin being an input, while a '0' corresponds to that pin being an output. An easy way to remember is that a '1' looks like an I (input) and a '0' looks like an O (output). The PORT register is the latch for the data to be output. When the PORT is read, the device reads the levels present on the I/O pins (not the latch). This means that care should be taken with read-modify-write commands on the ports and changing the direction of a pin from an input to an output.

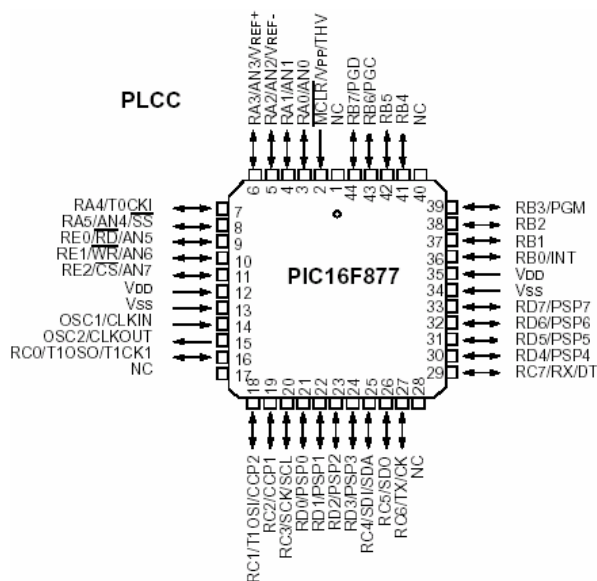


Figure 3-13: PIC16F877 Pin Diagram

The PIC 16F877 microcontroller is 40-pin device as shown in Figure 3-10. Vdd and Vss pins provide power and ground references, respectively. There are two ports for external oscillator connection. The Master Clear/Reset pin (MCLR) is used to reset the PIC device externally. This could be tied to a system reset circuit. If no external reset is used, the pin should be tied to Vdd. The PIC16F877 microcontroller has A (6 pins), B (8 pins), C (8 pins), D (8 pins) and E (3 pins) ports, totally 33 pins.

3.1.7.2 Timers

Timers can be used to time events. Timers have an advantage over merely using wait loops in that other processing can take place while the timer is running. 16F877 has three independent timers. Timer0 and Timer2 are 8 bit timer/counter whereas Timer1 is 16 bit. All of them are readable and writable and all have software prescaler, and Timer2 also has a postscaler.

3.1.7.3 USART (Universal Synchronous/Asynchronous Receiver/Transmitter)

This device allows the microcontroller to interface serially with other devices through protocols such as RS232.

3.1.7.4 Capture/Compare/PWM Modules

This module allows the microcontroller to watch an input until it reaches a certain value, and then take some action. It is especially useful for implementing Pulse Width Modulation (PWM) schemes. The PIC 16F877 microcontroller has two CCP modules; each contains 16-bit register which can operate as a Capture, Compare and PWM master/slave Duty Cycle register.

3.1.7.5 A/D Converter

The analog-to-digital (A/D) converter module can have up to eight analog inputs for a device. The analog input charges a sample and hold capacitor. The output of the sample and hold

capacitor is the input into the converter. The converter then generates a digital result of this analog level via successive approximation. This A/D conversion, of the analog input signal, results in a corresponding 10-bit digital number. The analog reference voltages (positive and negative supply) are software selectable to either the device's supply voltages (AVDD, AVss) or the voltage level on the AN3/VREF+ and AN2/VREFpins. The A/D module has four registers. These registers are:

- A/D Result High Register (ADRESH)
- A/D Result Low Register (ADRESL)
- A/D Control Register0 (ADCON0)
- A/D Control Register1 (ADCON1)

The ADCON0 register controls the operation of the A/D module. The ADCON1 register configures the functions of the port pins. The port pins can be configured as analog inputs (AN3 and AN2 can also be the voltage references) or as digital I/O.

3.1.7.6 Synchronous Serial Port (SSP) Module

The Synchronous Serial Port (SSP) module is a serial interface useful for communicating with other peripheral or microcontroller devices. These peripheral devices may be serial EEPROMs, shift registers, display drivers, A/D converters, etc. The SSP module can operate in one of two modes:

- Serial Peripheral Interface (SPI)
- Inter-Integrated Circuit (I²C)

SPI Mode

The SPI mode allows 8-bits of data to be synchronously transmitted and received simultaneously. All four modes of SPI are supported, as well as Microwire™ (sample edge) when the SPI is in the master mode. To accomplish communication, typically three pins are used:

- Serial Data Out (SDO)
- Serial Data In (SDI)

- Serial Clock (SCK)

Additionally a fourth pin may be used when in a slave mode of operation:

- Slave Select (SS)

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/ \overline{A}	P	S	R/ \overline{W}	UA	BF
bit 7							bit 0

Figure 3-14: SSPSTAT: Synchronous Serial Port Status Register

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
bit 7							bit 0

Figure 3-15: SSPCON: Synchronous Serial Port Control Register

Operation

When initializing the SPI, several options need to be specified. This is done by programming the appropriate control bits in the SSPCON register (SSPCON<5:0>) and SSPSTAT<7:6>. These control bits allow the following to be specified:

- Master Mode (SCK is the clock output)
- Slave Mode (SCK is the clock input)
- Clock Polarity (Idle state of SCK)
- Clock edge (output data on rising/falling edge of SCK)
- Data Input Sample Phase
- Clock Rate (Master mode only)
- Slave Select Mode (Slave mode only)

To enable the serial port, SSP Enable bit, SSPEN (SSPCON<5>) must be set. To reset or reconfigure SPI mode, clear bit SSPEN, re-initialize the SSPCON registers, and then set bit SSPEN. This configures the SDI, SDO, SCK and SS pins as serial port pins.

SSP I²C Operation

The SSP module in I²C mode fully implements all master and slave functions (including general call support) and provides interrupts-on-start and stop bits in hardware to determine a free bus (multi-master function). The SSP module implements the standard mode specifications, as well as 7-bit and 10-bit addressing.

A "glitch" filter is on the SCL and SDA pins when the pin is an input. This filter operates in both the 100 kHz and 400 kHz modes. In the 100 kHz mode, when these pins are an output, there is a slew rate control of the pin that is independent of device frequency.

Two pins are used for data transfer. These are the SCL pin, which is the clock, and the SDA pin, which is the data. The SDA and SCL pins are automatically configured when the I2C mode is enabled.

The SSPCON register allows control of the I2C operation. Four mode selection bits (SSPCON<3:0>) allow one of the following I2C modes to be selected:

- I²C Slave mode (7-bit address)
- I²C Slave mode (10-bit address)
- I²C Master mode, clock = OSC/4 (SSPADD +1)

Before selecting any I²C mode, the SCL and SDA pins must be programmed to inputs by setting the appropriate TRIS bits. Selecting an I2C mode, by setting the SSPEN bit, enables the SCL and SDA pins to be used as the clock and data lines in I2C mode. The CKE bit (SSPSTAT<6:7>) sets the levels of the SDA and SCL pins in either master or slave mode. When CKE = 1, the levels will conform to the SMBUS specification. When CKE = 0, the levels will conform to the I²C specification.

3.2 Digital I/O Interface

As computer power and the number of peripherals have increased, the older interfaces (serial and parallel port) have become a bottleneck of slow communications, with limited options of expansion. The Universal Serial Bus (USB) was designed from the ground up to be a simple

and efficient way to communicate with many types of peripherals, without the limitations and frustrations of existing interfaces. And today every PC has several USB interface on it. [28, 29].

Following features make USB so popular;

- **Ease of use:**
 - One interface for many devices
 - Automatic configuration
 - No user settings
 - Frees hardware resources for other devices
 - Easy to connect
 - Hot pluggable
 - No power supply required
- **Speed:** USB supports three bus speeds: high speed at 480 Megabits per second, full speed 12 Megabits per second and low speed at 1.5 Megabits per second. Every USB capable PC supports low and full speeds. High speed was added in version 2.0 of the specification, and requires USB 2.0 capable hardware on the motherboard or an expansion card.
- **Reliability:** The reliability of USB results from the hardware design and the data-transfer protocols. The hardware specifications for USB drivers, receivers, and cables eliminate most noise that could otherwise cause data errors. In addition, the USB protocol enables detecting of data errors and notifying the sender so it can retransmit.
- **Low Cost**
- **Low Power Consumption**

3.2.1 USB Hardware

3.2.1.1 Connectors

All devices have an upstream connection to the host and all hosts have a downstream connection to the device. Upstream and downstream connectors are not mechanically interchangeable, thus eliminating illegal loopback connections at hubs such as a downstream port connected to a downstream port. There are commonly two types of connectors, called type A and type B which are shown below.

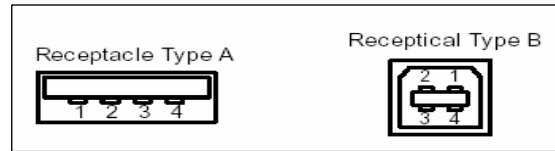


Figure 3-16: USB connector Types

Type A plugs always face upstream. Type A sockets will typically find themselves on hosts and hubs. For example type A sockets are common on computer main boards and hubs. Type B plugs are always connected downstream and consequently type B sockets are found on devices.

Table 3-3: USB pin functions

Pin Name	Cable Colour	Function
1	Red	Vbus (5 volts)
2	White	D-
3	Green	D+
4	Black	Ground

3.2.1.2 Electrical

USB uses a differential transmission pair for data. This is encoded using Non-Return to Zero Inverted (NRZI) and is bit stuffed to ensure adequate transitions in the data stream. On low and full speed devices, a differential '1' is transmitted by pulling D+ over 2.8V with a 15K ohm resistor pulled to ground and D- under 0.3V with a 1.5K ohm resistor pulled to 3.6V. A differential '0' on the other hand is a D- greater than 2.8V and a D+ less than 0.3V with the same appropriate pull down/up resistors. The receiver defines a differential '1' as D+ 200mV greater than D- and a differential '0' as D+ 200mV less than D-. The polarity of the signal is inverted depending on the speed of the bus. Therefore the terms 'J' and 'K' states are used in signifying the logic levels. In low speed a 'J' state is a differential 0. In high speed a 'J' state is a differential 1. USB transceivers will have both differential and single ended outputs. Certain bus states are indicated by single ended signals on D+, D- or both. [28]

3.2.1.3 Speed Identification

A USB device must indicate its speed by pulling either the D+ or D- line high to 3.3 volts. A full speed device, pictured below will use a pull up resistor attached to D+ to specify itself as a full speed device. These pull up resistors at the device end will also be used by the host or hub to detect the presence of a device connected to its port. Without a pull up resistor, USB assumes there is nothing connected to the bus. Some devices have this resistor built into its silicon, which can be turned on and off under firmware control, others require an external resistor.

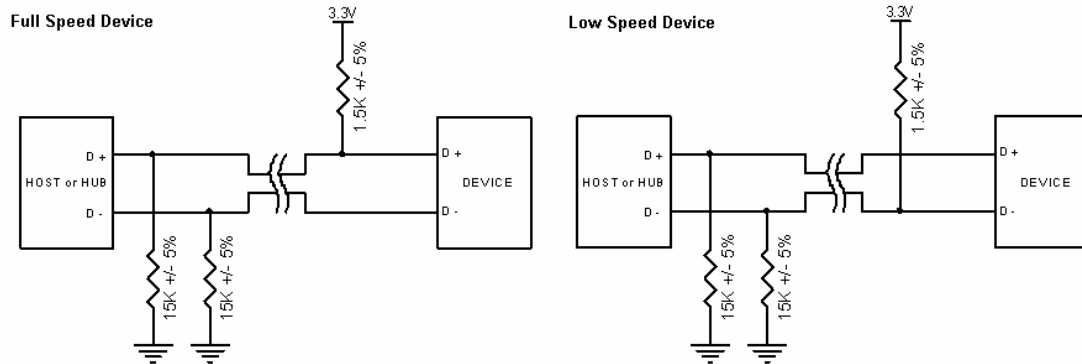


Figure 3-17: Full and Low Speed Device with pull up resistor

High speed devices will start by connecting as a full speed device (1.5k to 3.3V). Once it has been attached, it will do a high speed chirp during reset and establish a high speed connection, if the hub supports it. If the device operates in high speed mode, then the pull up resistor is removed to balance the line. [24]

3.2.1.4 Power Distribution

USB Devices have three forms of power classification:

- The Self-powered device has its own power supply and do not sink current from the USB cable.

- The bus-powered device will take all their power from the USB bus.
- A hybrid powered device takes power from both the USB bus as well as from it's own power supply.

3.2.2 USB Protocols

Unlike RS-232 and similar serial interfaces where the format of data being sent is not defined, USB is made up of several layers of protocols. In fact most USB controller I.C.s will take care of the lower layer, thus making it almost invisible to the end designer.

Each USB transaction consists of a

- Token Packet (Header defining what it expects to follow), an
- Optional Data Packet, (Containing the payload) and a
- Status Packet (Used to acknowledge transactions and to provide a means of error correction)

USB is a host centric bus. The host initiates all transactions. The first packet, also called a token is generated by the host to describe what is to follow and whether the data transaction will be a read or write and what the device's address and designated endpoint is. The next packet is generally a data packet carrying the payload and is followed by an handshaking packet, reporting if the data or token was received successfully, or if the endpoint is stalled or not available to accept data.

3.2.2.1 Common USB Packet Fields

Data on the USBus is transmitted LSBit first. USB packets consist of the following fields,

- Sync: All packets must start with a sync field. The sync field is 8 bits long at low and full speed or 32 bits long for high speed and is used to synchronize the clock of the receiver with that of the transmitter. The last two bits indicate where the PID fields starts.

- **PID:** PID stands for Packet ID. This field is used to identify the type of packet that is being sent.
- **ADDR:** The address field specifies which device the packet is designated for. Being 7 bits in length allows for 127 devices to be supported. Address 0 is not valid, as any device which is not yet assigned an address must respond to packets sent to address zero.
- **ENDP:** The endpoint field is made up of 4 bits, allowing 16 possible endpoints. Low speed devices, however can only have 2 additional endpoints on top of the default pipe. (4 endpoints max)
- **CRC:** Cyclic Redundancy Checks are performed on the data within the packet payload. All token packets have a 5 bit CRC while data packets have a 16 bit CRC.
- **EOP:** End of packet. Signaled by a Single Ended Zero (SE0) for approximately 2 bit times followed by a J for 1 bit time

3.2.2.2 USB Packet Types

USB has four different packet types. Token packets indicate the type of transaction to follow, data packets contain the payload, handshake packets are used for acknowledging data or reporting errors and start of frame packets indicate the start of a new frame.

1. Token Packets

There are three types of token packets:

- **In** - Informs the USB device that the host wishes to read information.
- **Out** - Informs the USB device that the host wishes to send information.
- **Setup** - Used to begin control transfers.

2. Data Packets

There are two types of data packets, Data0 and Data1, each capable of transmitting up to 1024 bytes of data. High Speed mode defines another two data PIDs, DATA2 and MDATA. Data packets have the following format,

- Maximum data payload size for low-speed devices is 8 bytes.
- Maximum data payload size for full-speed devices is 1023 bytes.
- Maximum data payload size for high-speed devices is 1024 bytes.
- Data must be sent in multiples of bytes.

3. Handshake Packets

There are three types of handshake packets which consist simply of the PID

- ACK - Acknowledgment that the packet has been successfully received.
 - NAK - Reports that the device temporary cannot send or received data. Also used during interrupt transactions to inform the host there is no data to send.
 - STALL - The device finds it's in a state that it requires intervention from the host.
- Handshake

4. Start of Frame (SOF)Packets

The SOF packet consisting of an 11-bit frame number is sent by the host every 1ms \pm 500ns on a full speed bus or every 125 μ s \pm 0.0625 μ s on a high speed bus.

3.2.3 Endpoints

Endpoints can be described as sources or sinks of data. As the bus is host centric, endpoints occur at the end of the communications channel at the USB function. At the software layer, your device driver may send a packet to your devices EP1 for example. As the data is flowing out from the host, it will end up in the EP1 OUT buffer. Your firmware will then at its leisure read this data. If it wants to return data, the function cannot simply write to the bus as the bus is controlled by the host. Therefore it writes data to EP1 IN which sits in the buffer until such time when the host sends a IN packet to that endpoint requesting the data. Endpoints can also be seen as the interface between the hardware of the function device and the firmware running on the function device.

All devices must support endpoint zero. This is the endpoint which receives all of the devices control and status requests during enumeration and throughout the duration while the device is operational on the bus.

3.2.4 Pipes

While the device sends and receives data on a series of endpoints, the client software transfers data through pipes. A pipe is a logical connection between the host and endpoint(s). Pipes will also have a set of parameters associated with them such as how much bandwidth is allocated to it, what transfer type (Control, Bulk, Iso or Interrupt) it uses, a direction of data flow and maximum packet/buffer sizes. For example the default pipe is a bi-directional pipe made up of endpoint zero in and endpoint zero out with a control transfer type.

3.3 Microcontroller Card

In this study, DLP-IO26 microcontroller card with USB interface is chosen from Dlpdesign [30] to control the servo motor driver through the PC's USB port. The DLP-IO26 consists of a USB interface, a PIC 16F877 (target) microcontroller and a Flash programming system. In Figure 3-18 the block diagram of DLP-IO26 card is shown. The USB interface is designed around FTDI's FT8U245AM and is used for both the Flash download process and host communications (16F877 to Host PC) at run time.

The Flash programming system is comprised of a PIC 16F872 and a 12.5-volt DV-DC converter and is used exclusively to perform the download process. Hex file data can be written directly into the 16F877's Flash memory without the need for an external device programmer. PC application software that runs under Windows 98/2000/XP is available from dlpdesign.com as a free download that performs the Flash download process.

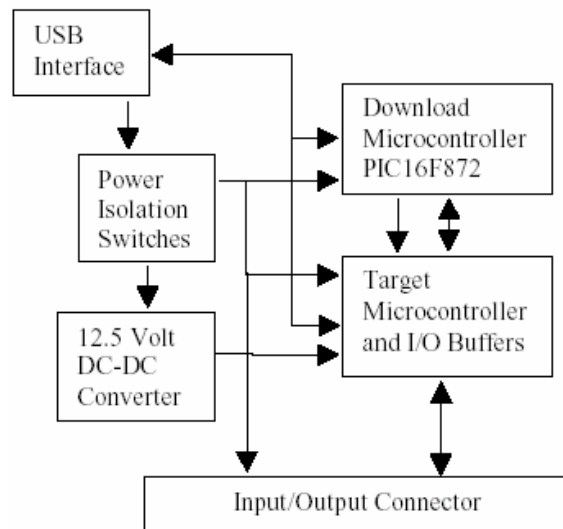


Figure 3-18: Block diagram of DLP-IO26

The DLP-IO26 connects to the user's hardware via 8 buffered digital inputs and 18 general purpose I/O lines. The 18 general purpose I/O lines are directly connected to the 16F877 microcontroller.

Download Microcontroller

The download microcontroller serves two purposes: the reset function and the firmware download process.

USB Interface

FTDI's FT8U245AM is a USB FIFO - Fast Parallel Data Transfer IC. No in-depth knowledge of USB required as all USB Protocol is handled automatically by the USB protocol engine within the I.C. The USB Protocol Engine manages the data stream from the device USB control endpoint. It handles the low level USB protocol requests generated by the USB host controller. The FT8U245AM provides an easy cost-effective method of transferring data to / from a peripheral and a host P.C. at up to 8 Million bits (1 Megabyte) per second. Its simple FIFO-like design makes it easy to interface to any CPU (MCU) either by mapping the device into the Memory / IO map of the CPU, using DMA or controlling the device via IO ports. [31]

FTDI's Virtual COM port and "D2XX Direct Drivers" for Windows drivers eliminate the need for USB driver development in most cases. In this study DLL version of the drivers (D2XX direct drivers) is used.

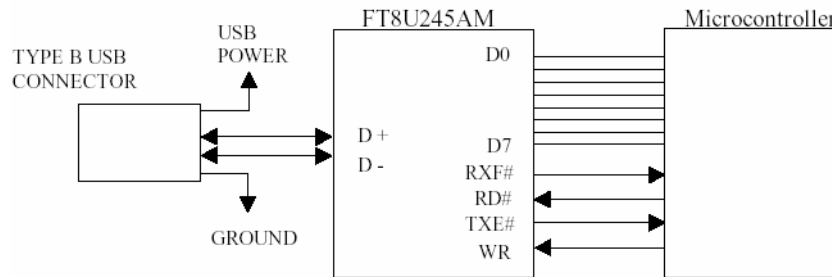


Figure 3-19: The interface between FT8U245AM and Microcontroller

Four basic hand-shaking lines and eight data lines D [7...0] are provided to interface with the chip, as shown in Figure 3-19. The FT8U245AM's (refer to Appendix B) internal FIFO is comprised of two buffers which can hold 128 bytes of received data coming from the host PC and 384 bytes of data to be transmitted to the host. Connection to a microcontroller is made with these 12 lines.

RD# (input) When pulled low, RD# takes the 8 data lines from a high impedance state to the current byte in the FIFO's receive buffer. Taking RD# high returns the data pins to a high impedance state and prepares the next byte (if available) in the FIFO to be read.

WR (input) When taken from a high state to a low state, WR# reads the 8 data lines and writes the byte into the FIFO's transmit buffer. Data written to the transmit buffer is immediately sent to the host PC and placed in the RS-232 buffer opened by the application program.

TXE# (output) When high, the FIFO's 384-byte transmit buffer is full or busy storing the last byte written. Do not attempt to write data to the transmit buffer when TXE# is high.

RXF# (output) When low, at least 1 byte is present in the FIFO's 128-byte receive buffer and is ready to be read with RD#. RXF# goes high when the receive buffer is empty.

Since both the download and target microcontrollers need to access the USB interface, a pair of AND gates were added to handle the read and write functions. During the firmware download process, the target microcontroller is disabled, and its outputs to the AND gates are left high. This allows the download microcontroller to perform both the USB read and write functions. When the target microcontroller is running, the download microcontroller leaves its USB read and writes outputs high so that the target microcontroller can have access to the USB port.

Inputs and Outputs

The DLP-IO26 provides 8 latched digital outputs (D0-D7) and 18 bi-directional, general-purpose digital I/O lines, 8 of which can be configured as 10-bit analog-to-digital converter inputs for measuring voltages in the range of 0-5 volts.

Table 3-4: Description of DLP-IO26 40 pin header pins

Pin Number	Description
1	External Reset for the USB interface (input)
2	Target Microprocessor Reset (active high)(output)
3	USB Port Ground
4	USB Port VCC
13, 14, 24, 33, 34	Switched Ground
23	Switched Vcc
25	SLEEP# USB Standby Indicator
26	SW2 Test Switch
5, 6, 7, 8, 9, 10, 11, 12	Digital Outputs
15,16,17,18,19,20,21,22	Port C General Purpose I/O, Timer1 I/O, PWM1 Output, SPI/I2C, Synchronous Serial Data output, and USART I/O.
27	Port B General Purpose I/O and External Interrupt Input with programmable weak pull-up.
28,30,32	Port E General Purpose I/O and A/D channels 5, 6, 7
35, 36, 37, 38, 39, 40	Port A General Purpose I/O, Open Drain Output (A4 only), and A/D channels 0, 1, 2, 3, 4

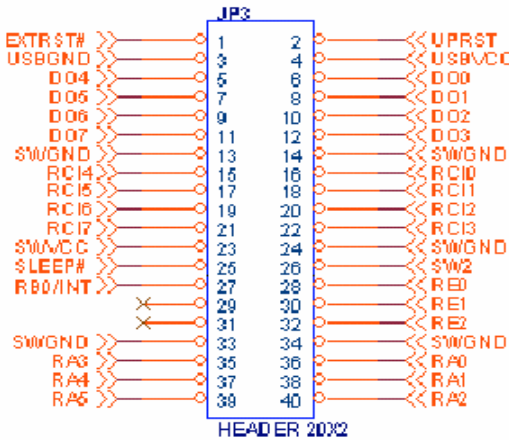


Figure 3-20: DLP-IO26 40 pin header

The External reset pin (pin 1) can be used by the target electronics to reset the USB interface on the DLP-IO26. The target microcontroller reset signal (pin 2) is made available such that target electronics can also be held in the reset state, if desired, during the firmware download process. Five-volt power is made available for target electronics but care must be taken to not exceed a total of 500mA drawn from the USB port.

On DLP-IO26 PORTD of PIC16F877 is used as 8-bit I/O data bus by the USB port. And it can be also used as digital output by the help of the D flip flop. See Appendix B, for DLP-IO26 card schematic diagram.

3.4 Precision Motion Controller LM629

LM629 is a microcontroller peripheral that incorporates in one device all the functions of a sample-data motion control system controller. Using the LM628/629 makes the potentially complex task of designing a fast and precise motion control system much easier. Both position and velocity motion control systems can be implemented with the LM629.[32]

LM629 is itself a purpose designed microcontroller that implements a position decoder, a summing junction, a digital PID loop compensation filter, and a trajectory profile generator,

Figure 3-21. The LM629 provides a 7-bit plus sign PWM signal with sign and magnitude outputs. Interface to the host microcontroller is via an 8-bit bi-directional data port and six control lines which include host interrupt and hardware reset.

In operation, to start a movement, a host microcontroller downloads acceleration, velocity and target position values to the LM629 trajectory generator. At each sample interval these values are used to calculate new demand or “set point” positions which are fed into the summing junction. Actual position of the motor is determined from the output signals of an optical incremental encoder. Decoded by the LM629’s position decoder, actual position is fed to the other input of the summing junction and subtracted from the demand position to form the error signal input for the control loop compensator. The compensator is in the form of a “three term” PID filter (proportional, integral, derivative); this is implemented by a digital filter. The coefficients for the PID digital filter are downloaded from the host before commencing a move. For a load that varies during a movement more coefficients can be downloaded and used to update the PID filter at the moment the load changes. All trajectory parameters except acceleration can also be updated while a movement is in progress.

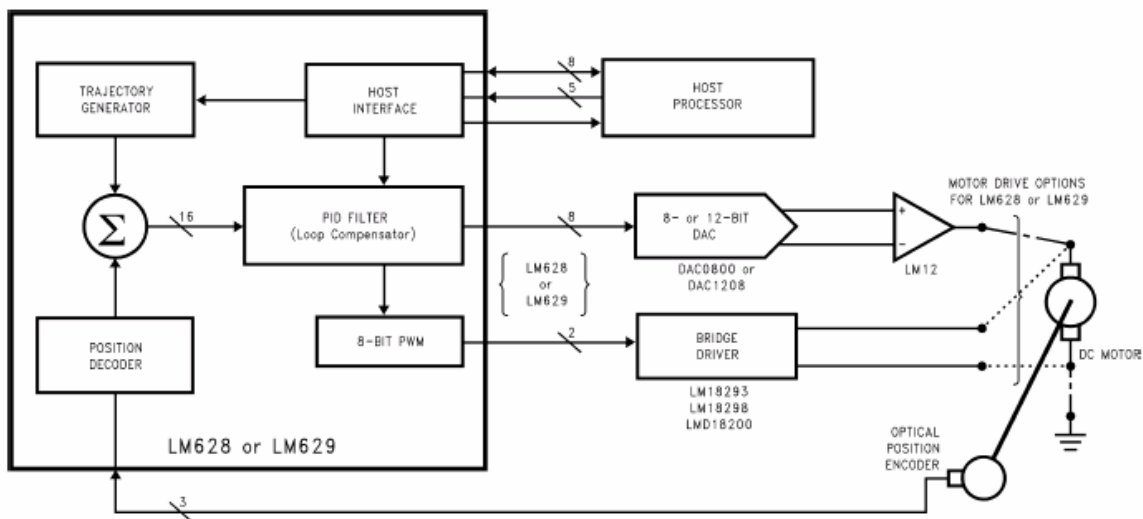


Figure 3-21: LM628 and LM629 Typical System Block Diagram

Features:

- 32-bit position, velocity, and acceleration registers
- Programmable digital PID filter with 16-bit coefficients
- Programmable derivative sampling interval
- 8-bit sign-magnitude PWM output data (LM629)
- Internal trapezoidal velocity profile generator
- Velocity, target position, and filter parameters may be changed during motion
- Position and velocity modes of operation
- Real-time programmable host interrupts
- 8-bit parallel asynchronous host interface
- Quadrature incremental encoder interface with index pulse input

3.4.1 Hardware Architecture

Four major functional blocks make up the LM629 in addition to the host and output interfaces. These are the Trajectory Profile Generator, Loop Compensating PID Filter, Summing Junction and Motor Position Decoder. Details of how LM629 is implemented by a purpose designed microcontroller are shown in Figure 3-22.

The control algorithm is stored in a 1k x 16-bit ROM and uses 16-bit wide instructions. A PLA decodes these instructions and provides data transfer timing signals for the single 16-bit data and instruction bus. User variable filter and trajectory profile parameters are stored as 32-bit double words in RAM. To provide sufficient dynamic range a 32-bit position register is used and for consistency. 32 bits are also used for velocity and acceleration values. A 32-bit ALU is used to support the 16 x 16-bit multiplications of the error and PID digital filter coefficients.

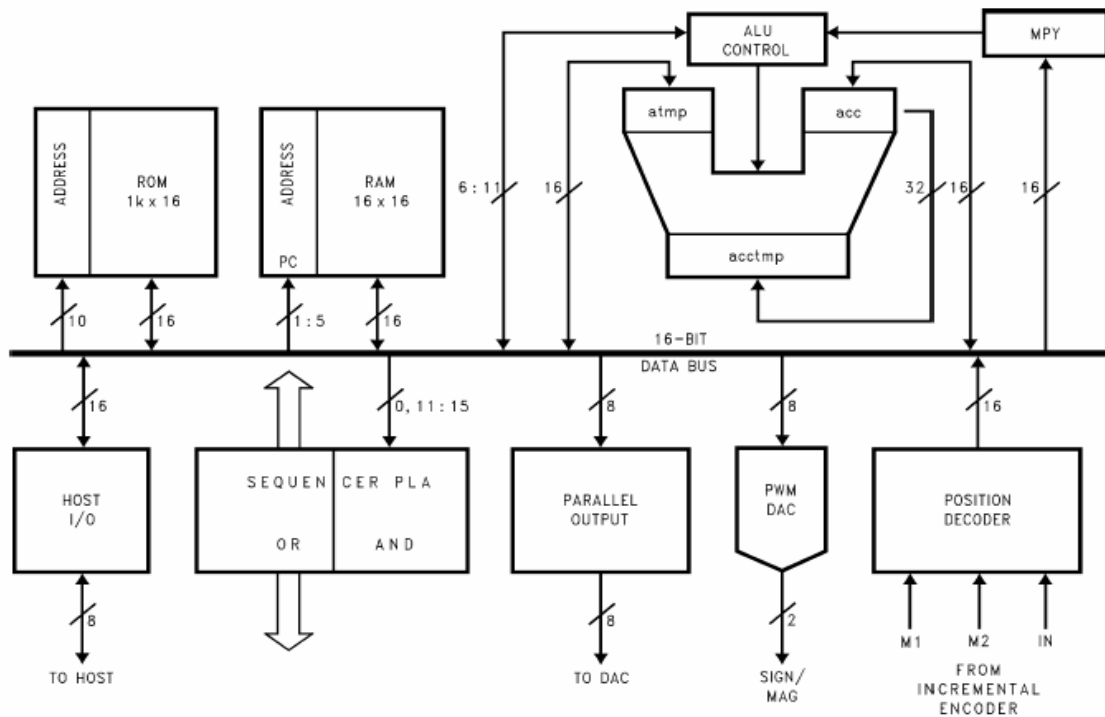


Figure 3-22: Hardware Architecture of LM629

3.4.2 Motor Position Decoder

LM629 provides an interface for an optical position shaft encoder, decoding the two quadrature output signals to provide position and direction information. From these the decoder PLA determines if the motor has moved forward, backward or stayed still and then drives a 16-bit up-down counter that keeps track of actual motor position.

3.4.3 Velocity Profile (Trajectory) Generation

The trapezoidal velocity profile generator computes the desired position of the motor versus time. In the position mode of operation, the host processor specifies acceleration, maximum velocity, and final position. The LM629 uses this information to affect the move by accelerating as specified until the maximum velocity is reached or until deceleration must begin to stop at the specified final position. The deceleration rate is equal to the acceleration rate. At any time during the move the maximum velocity and/or the target position may be changed, and the

motor will accelerate or decelerate accordingly. Figure 3-23 illustrates two typical trapezoidal velocity profiles. Figure 3-23(a) shows a simple trapezoid, while Figure 3-23(b) is an example of what the trajectory looks like when velocity and position are changed at different times during the move.

When operating in the velocity mode, the motor accelerates to the specified velocity at the specified acceleration rate and maintains the specified velocity until commanded to stop. [33]

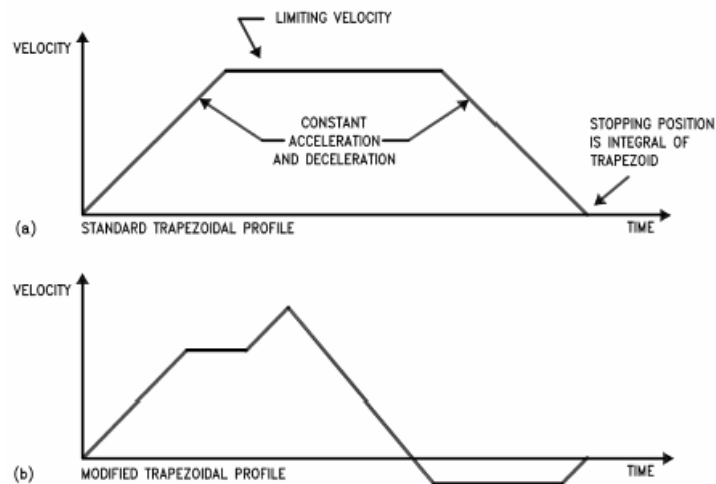


Figure 3-23: Typical Velocity Profiles

All trajectory parameters are 32-bit values. Position is a signed quantity. Acceleration and velocity are specified as 16-bit, positive-only integers having 16-bit fractions. The integer portion of velocity specifies how many counts per sampling interval the motor will traverse. The fractional portion designates an additional fractional count per sampling interval. Although the position resolution of the LM629 is limited to integer counts, the fractional counts provide increased average velocity resolution. Acceleration is treated in the same manner. Each sampling interval the commanded acceleration value is added to the current desired velocity to generate a new desired velocity (unless the command velocity has been reached).

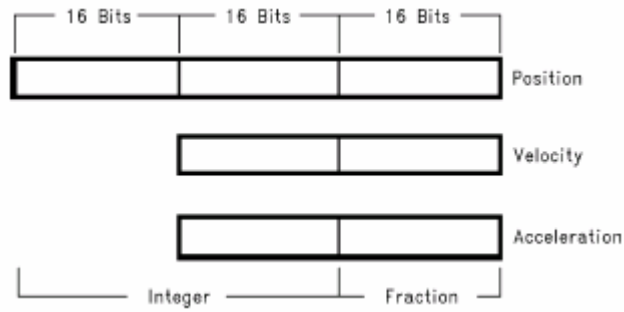


Figure 3-24: Position, Velocity and Acceleration Registers

3.4.4 PID Compensation Filter

The LM629 uses a digital Proportional Integral Derivative (PID) filter to compensate the control loop. The motor is held at the desired position by applying a restoring force to the motor that is proportional to the position error, plus the integral of the error, plus the derivative of the error. The following discrete-time equation and Figure 3-25 illustrates the control performed by the LM629:

$$u(n) = k_p e(n) + k_i \sum_{N=0}^n e(n) + k_d [e(n') - e(n'-1)] \quad (3.1)$$

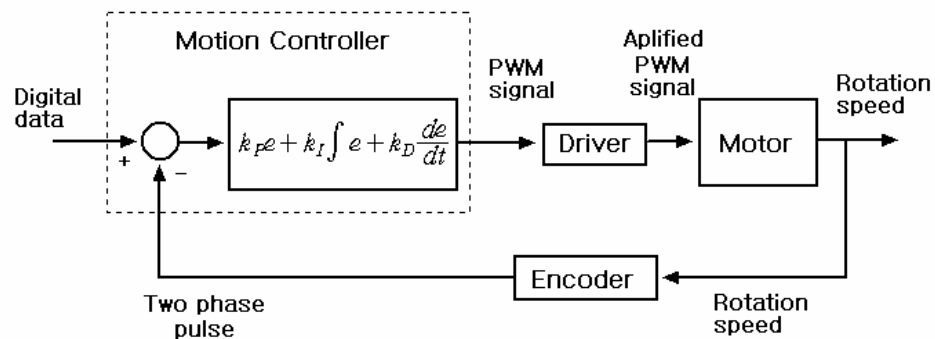


Figure 3-25: LM629 PID control block diagram

where $u(n)$ is the motor control signal output at sample time n , $e(n)$ is the position error at sample time n , n' indicates sampling at the derivative sampling rate, and k_p , k_i , and k_d are the discrete-time filter parameters loaded by the users. [33]

The first term, the proportional term, provides a restoring force proportional to the position error, just as does a spring obeying Hooke's law.

The second term, the integration term, provides a restoring force that grows with time, and thus ensures that the static position error is zero. If there is a constant torque loading, the motor will still be able to achieve zero position error.

The third term, the derivative term, provides a force proportional to the rate of change of position error. It acts just like viscous damping in a damped spring and mass system (like a shock absorber in an automobile). The sampling interval associated with the derivative term is user-selectable; this capability enables the LM629 to control a wider range of inertial loads (system mechanical time constants) by providing a better approximation of the continuous derivative. In general, longer sampling intervals are useful for low-velocity operations.

In operation, the filter algorithm receives a 16-bit error signal from the loop summing-junction. The error signal is saturated at 16 bits to ensure predictable behavior. In addition to being multiplied by filter coefficient k_p , the error signal is added to an accumulation of previous errors (to form the integral signal) and, at a rate determined by the chosen derivative sampling interval, the previous error is subtracted from it (to form the derivative signal). All filter multiplications are 16-bit operations; only the bottom 16 bits of the product are used.

The integral signal is maintained to 24 bits, but only the top 16 bits are used. This scaling technique results in a more usable (less sensitive) range of coefficient k_i values. The 16 bits are right-shifted eight positions and multiplied by filter coefficient k_i to form the term which contributes to the motor control output. The absolute magnitude of this product is compared to coefficient i_l , and the lesser, appropriately signed magnitude then contributes to the motor control signal. The derivative signal is multiplied by coefficient k_d each derivative sampling interval. This product contributes to the motor control output every sample interval, independent of the user-chosen derivative sampling interval. The k_p , limited k_i , and k_d product terms are

summed to form a 16-bit quantity. Depending on the output mode (word size), either the top 8 or top 12 bits become the motor control output signal.

3.4.5 Motor Outputs

The LM629 provides 8-bit, sign and magnitude PWM output signals for directly driving switch-mode motor-drive amplifiers. Figure 3-26 shows the format of the PWM magnitude output signal.

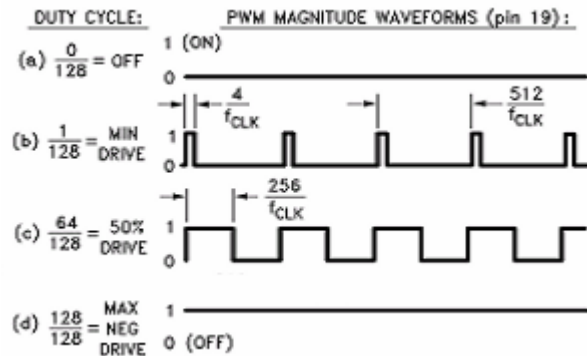


Figure 3-26: PWM Output Signal Format

3.4.6 LM629 Reading and Writing Operations

The host processor writes commands to the LM629 via the host I/O port when Port Select (PS) input (Pin 16) is logic low. The desired command code is applied to the parallel port line and the Write (WR) input (Pin 15) is strobed. The command byte is latched into the LM629 on the rising edge of the WR input. When writing command bytes it is necessary to first read the status byte and check the state of a flag called the “busy bit” (Bit 0). If the busy bit is logic high, no command write may take place. The busy bit is never high longer than 100 μs , and typically falls within 15 μs to 25 μs . The host processor reads the LM629 status byte in a similar manner: by strobing the Read (RD) input (Pin 13) when PS (Pin 16) is low; status information remains valid as long as RD is low.

Writing and reading data to/from the LM629 (as opposed to writing commands and reading status) are done with PS (Pin 16) logic high. These writes and reads are always an integral number (from one to seven) of two-byte words, with the first byte of each word being the more significant. When transferring data words (byte-pairs), it is necessary to first read the status byte and check the state of the busy bit. When the busy bit is logic low, the user may then sequentially transfer both bytes comprising a data word, but the busy bit must again be checked and found to be low before attempting to transfer the next byte pair (when transferring multiple words). If a command is written when the busy bit is high, the command will be ignored. The busy bit goes high immediately after writing a command byte, or reading or writing a second byte of data.

3.4.7 User Command Set

The Table 3-5 describes the user command set of the LM629. Some of the commands can be issued alone and some require a supporting data structure. Commands are categorized by function: initialization, interrupt control, filter control, trajectory control, and data reporting. [33]

Table 3-5: LM629 User Command Set

Command	Type	Description	HEX	Data Bytes
RESET	Initialize	Reset LM628	0 0	0
DFH	Initialize	Define Home	0 2	0
SIP	Interrupt	Set Index Position	0 3	0
LPEI	Interrupt	Interrupt on Error	1B	2
LPES	Interrupt	Stop on Error	1A	2
SBPA	Interrupt	Set Breakpoint, Absolute	20	4
SBPR	Interrupt	Set Breakpoint, Relative	21	4
MSKI	Interrupt	Mask Interrupts	1C	2
RSTI	Interrupt	Reset Interrupts	1D	2
LFIL	Filter	Load Filter Parameters	1E	2 to 10
UDF	Filter	Update Filter	0 4	0
LTRJ	Trajectory	Load Trajectory	1F	2 to 14

Table 3-5: LM629 User Command Set (Continued)

Command	Type	Description	HEX	Data Bytes
STT	Trajectory	Start Motion	0 1	0
RDSTAT	Report	Read Status Byte	None	1
RDSIGS	Report	Read Signals Register	0 C	2
RDIP	Report	Read Index Position	0 9	4
RDDP	Report	Read Desired Position	0 8	4
RDRP	Report	Read Real Position	0A	4
RDDV	Report	Read Desired Velocity	0 7	4
RDRV	Report	Read Real Velocity	0B	2
RDSUM	Report	Read Integration Sum	0D	2

3.4.8 Programming LM629

As indicated in the Figure 3-22, the LM629 is a bus peripheral and must be programmed by a host processor. Breaking programs for the LM629 into sets of functional blocks simplifies the programming process; each block executes a specific task. [34]

1. Busy-Bit Check Module

The first module required for successful programming of the LM629 is a busy-bit check module. The busy-bit, bit zero of the status byte, is set immediately after the host writes a command byte, or reads or writes the second byte of a data word. While the busy-bit is set, the LM629 will ignore any commands or attempts to transfer data. A busy-bit check module that polls the Status Byte and waits until the busy-bit is reset will ensure successful host/LM629 communications.

Reading the Status Byte is accomplished by executing a RDSTAT command. RDSTAT is directly supported by LM629 hardware and is executed by pulling CS, PS, and RD logic low.

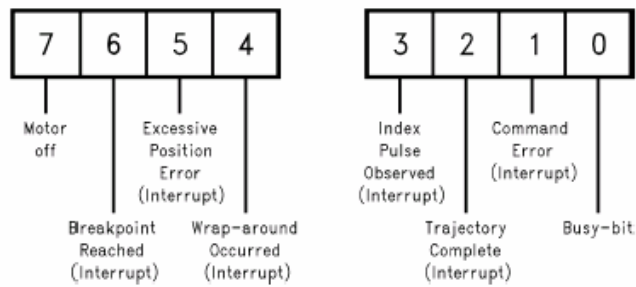


Figure 3-27: Status Byte Bit Allocation

2. Initialization Module

In general, an initialization module contains a reset command and other initialization; interrupt control, and data reporting commands. The example initialization module, detailed in Table 3-6, contains a hardware reset block

Hardware Reset Block

Immediately following power-up, a hardware reset **must** be executed. Hardware reset is initiated by strobing RST (pin 27) logic low for a **minimum of eight LM628 clock periods**. The reset routine begins after RST is returned to logic high. During the reset execution time, **1.5 ms** maximum, the LM629 will ignore any commands or attempts to transfer data.

Reset Interrupts

An RSTI command sequence allows the user to reset the interrupt flag bits, bits one through six of the status byte. See Figure 3-27. The RSTI command initiates resetting the interrupt flag bits. Command RSTI also resets the host interrupt output pin (pin 17). Immediately following the RSTI command, a single data word is written. The first byte is not used. Logical zeros in bits one through six of the second byte reset the corresponding interrupts. See Figure 3-28.

Mask Interrupts

An MSKI command sequence allows the user to determine which interrupt conditions result in host interrupts; interrupting the host via the host interrupt output (pin 17). It contains an MSKI command and one data word. The MSKI command initiates interrupt masking. See Figure 3-28. Any zeros in this 6-bit field mask (disable) the corresponding interrupts while any ones unmask (enable) the corresponding interrupts.

Table 3-6: Initialization Module (with Hardware Reset)

Port	Bytes	Command	Comments
		hardware reset	Strobe RST, pin 27, logic low for eight clock periods minimum.
		wait	The maximum time to complete hardware reset tasks is 1.5 ms. During this reset execution time, the LM628 will ignore any commands or attempts to transfer data.
command	xx	RDSTAT	This command reads the status byte. It is directly supported by LM628 hardware and can be executed at any time by pulling CS, PS, and RD logic low. Status information remains valid as long as RD is logic low.
		decision	If the status byte is C4 hex or 84 hex, continue. Otherwise loop back to hardware reset.
command	1D	RSTI	This command resets only the interrupts indicated by zeros in bits one through six of This command resets only the interrupts indicated by zeros in bits one through six of interrupt output pin (pin 17).
Busy-bit Check Module			
data	xx	HB	don't care
data	0 0	LB	Zeros in bits one through six indicate all interrupts will be reset.
Busy-bit Check Module			
command		RDSTAT	This command reads the status byte.
		decision	If the status byte is C0 hex or 80 hex, continue. Otherwise loop back to hardware reset.

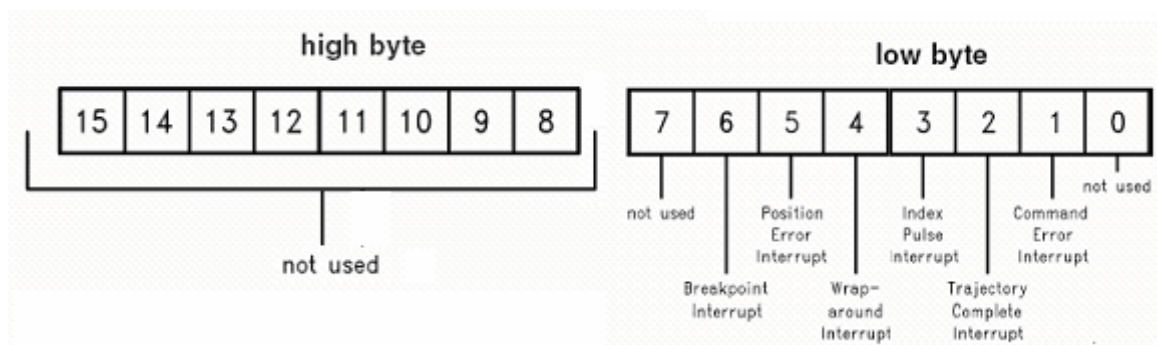


Figure 3-28: Interrupt Mask/Reset Bit Allocations

3. Filter Programming Module

Load Filter Parameters (Coefficients)

An LFIL (Load FILTER) command sequence includes command LFIL, a filter control word, and a variable number of data words. The LFIL command initiates loading filter coefficients into input buffers. The two data bytes, written immediately after LFIL, comprise the filter control word. The first byte programs the derivative sampling coefficient, ds (i.e. selects the derivative sampling interval). The second byte indicates, with logical ones in respective bit positions, which of the remaining four filter coefficients will be loaded. See Figure 3-29, Table 3-7. Any combination of the four coefficients can be loaded within a single LFIL command sequence.

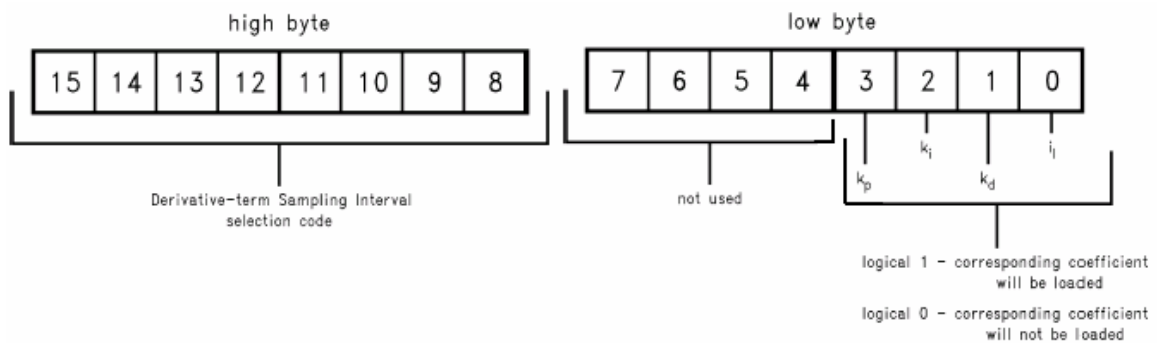


Figure 3-29: Filter Control Word Bit Allocation

In the case of the example module, the first byte of the filter control word, 00 hex, programs a derivative sampling coefficient of one. The second byte, x8 hex, indicates only the proportional gain coefficient will be loaded. The update filter command, UDF, transfers new filter coefficients from input buffers to working registers. Until UDF is executed, the new filter coefficients do not affect the transfer characteristic of the filter.

Table 3-7: Filter Programming Module

Port	Bytes	Command	Comments
command	1E	LFIL	This command initiates loading the filter coefficients input buffers.
Busy-bit Check Module			
data	0 0	HB	These two bytes are the filter control word. A 00 hex HB sets the derivative sampling interval to 2048/fCLK by setting ds to one. A x8 hex LB indicates only kp will be loaded.The other filter parameters will remain at zero, their reset default value.
data	x8	LB	
Busy-bit Check Module			
data	0 0	HB	These two bytes set kp to ten.
data	0A	LB	
Busy-bit Check Module			
command	4	UDF	This command transfers new filter coefficients from input buffers to working registers. Until UDF is executed, coefficients loaded via the LFIL command do not affect the filter transfer characteristic.
Busy-bit Check Module			

4. Trajectory Programming Module

Load Trajectory Parameters

An LTRJ (Load TRajectory) command sequence includes command LTRJ, a trajectory control word, and a variable number of data words. The LTRJ command initiates loading trajectory parameters into input buffers. The two data bytes, written immediately after LTRJ, comprise the trajectory control word. The first byte programs, with logical ones in respective bit positions, the trajectory mode (velocity or position), velocity mode direction, and stopping mode. The second byte indicates, with logical ones in respective bit positions, which of the three trajectory parameters will be loaded. It also indicates whether the parameters are absolute or relative. See Figure 3-30. Immediately following the trajectory control word, the trajectory parameters are written. The start motion control command, STT (STarT), transfers new trajectory parameters from input buffers to working registers and begins execution of the new trajectory. Until STT is executed, the new trajectory parameters do not affect shaft motion.

Table 3-8: Trajectory Programming Module

Port	Bytes	Command	Comments
command		LTRJ	This command initiates loading the trajectory parameters input buffers.
			Busy-bit Check Module
data	xx	HB	These two bytes are the trajectory control word. A 0A hex LB indicates velocity and position will be loaded and both parameters are absolute.
data		LB	
			Busy-bit Check Module
data	xx	HB	Velocity is loaded in two data words. These two bytes are the high data word.
data	0 0	LB	
			Busy-bit Check Module
data	xx	HB	Velocity data word (low)
data	0 0	LB	
			Busy-bit Check Module
data	xx	HB	Position is loaded in two data words. These two bytes are the high data word.
data	0 0	LB	
			Busy-bit Check Module
data	Xx	HB	Position data word (low)
data	0 0	LB	
			Busy-bit Check Module
command		STT	STT must be issued to execute the desired trajectory.
			Busy-bit Check Module

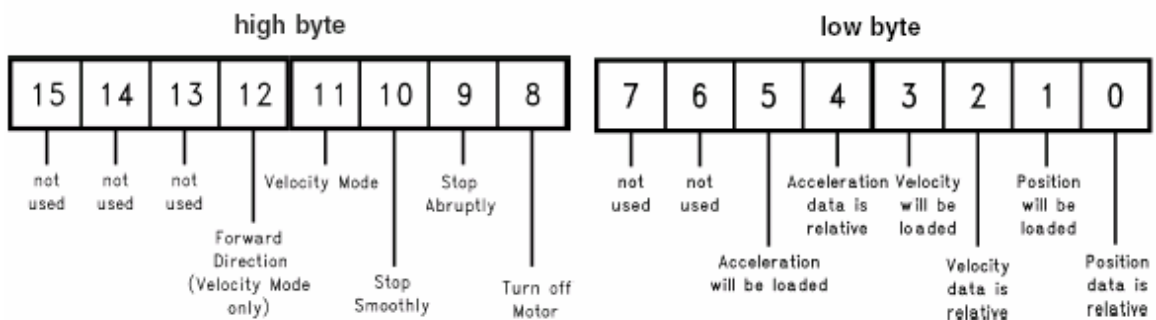


Figure 3-30: Trajectory Control Word Bit Allocation

In the case of the example module, the first byte of the trajectory control word, 00 hex, programs the LM629 to operate in position mode. The second byte, 0A hex, indicates velocity and position will be loaded and both parameters are absolute. Four data words, two for each parameter loaded, follow the trajectory control word.

3.5 Data Storage


To store the position data of the digitized object, a storage unit is added to the study. For storage MultiMediaCard (MMC) is used. The position data is first saved into MMC through the PIC. After the storage of data the positions are read from MMC by PIC. And these position data is processed and motors are activated. SPI (Serial Peripheral Interface) communication protocol is used between MMC and PIC.

All transactions between PIC and MMC must be in chunks of 512 bytes. Data to be stored or read is first saved to the external I2C (Inter-Integrated Circuit) EEPROM and is then automatically transferred to the MMC or PIC when 512 bytes accumulate. Storage is limited to a single file only, mainly due to the constraints of the PIC and available RAM on PIC (368 byte). [39]

General overview to MMC [37]

The MMC and Reduced-Size MultiMediaCard (RS-MMC) are very small, removable flash storage devices, designed specifically for storage applications that put a premium on small form factor, low power and low cost.

To support this wide range of applications, the MMC Protocol, a simple seven-pin serial interface, Figure 3-31, is designed for maximum scalability and configurability. All device and interface configuration data (such as maximum frequency, card identification, etc.) are stored on the card.



SPI Mode			
1	CS	I	Chip Select (active low)
2	DataIn	I	Host-to-card Commands and Data
3	VSS1	S	Supply Voltage Ground
4	VDD	S	Supply Voltage
5	CLK	I	Clock
6	VSS2	S	Supply Voltage Ground
7	DataOut	O	Card-to-host Data and Status

Figure 3-31: MMC Pinout and Pad Assignment

The MMC interface allows for easy integration into any design, regardless of microprocessor used. For compatibility with existing controllers, the card offers, in addition to the card interface, an alternate communication protocol, which is based on the Serial Peripheral Interface (SPI) standard.

The MMC wake up in the MultiMediaCard mode. The card will enter SPI mode if the CS signal is asserted (CS signal low) during the reception of the reset command (CMD0, refer to appendix). The module will switch to SPI mode and respond with the SPI mode R1 response.

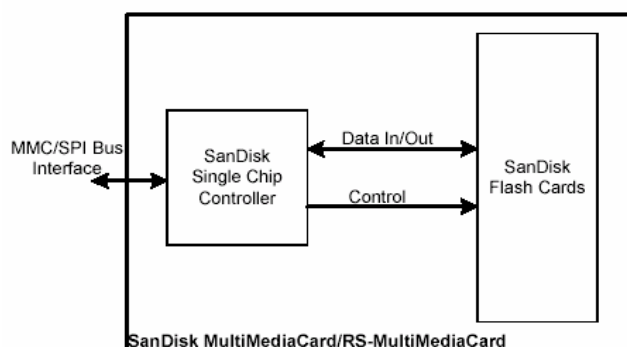


Figure 3-32: MultiMediaCard/RS-MultiMediaCard Block Diagram

The MMC memory space is byte addressable with addresses ranging from 0 to the last byte, and is divided into several structures. Memory bytes are grouped into 512-byte blocks called **sectors**. Every block can be read, written and erased individually. Sectors are grouped into **erase groups** of 16 or 32 sectors depending on card size. Any combination of sectors within one group, or any combination of erase groups can be erased with a single erase command. A write command implicitly erases the memory before writing new data into it.

Read and Write Operations

The MMC support two read/write modes as shown in Figure 3-33. In single block mode the host reads or writes one data block in a pre-specified length. The data block transmission is protected with 16-bit CRC that is generated by the sending unit and checked by the receiving unit. The block length for read operations is limited by the device sector size (512 bytes) but can be as small as a single byte. Misalignment is not allowed. Every data block must be contained in a single physical sector. The block length for write operations must be identical to the sector size and the start address aligned to a sector boundary.

In single block mode the host reads or writes one data block in a pre-specified length. The data block transmission is protected with 16-bit CRC that is generated by the sending unit and checked by the receiving unit. The block length for read operations is limited by the device sector size (512 bytes) but can be as small as a single byte. Misalignment is not allowed. Every data block must be contained in a single physical sector. The block length for write operations must be identical to the sector size and the start address aligned to a sector boundary.

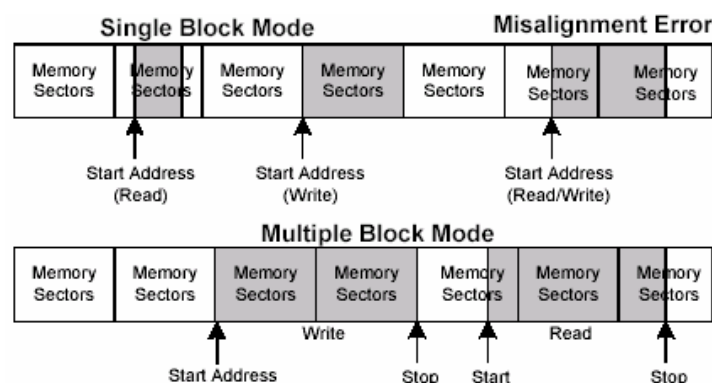


Figure 3-33: Data Transfer Formats

Multiple block mode is similar to the single block mode, except for the host can read/write multiple data blocks (all have the same length) that are stored or retrieved from contiguous memory addresses starting at the address specified in the command. The operation is terminated with a stop transmission command. Misalignment and block length restrictions apply to multiple blocks and are identical to the single block read/write operations.

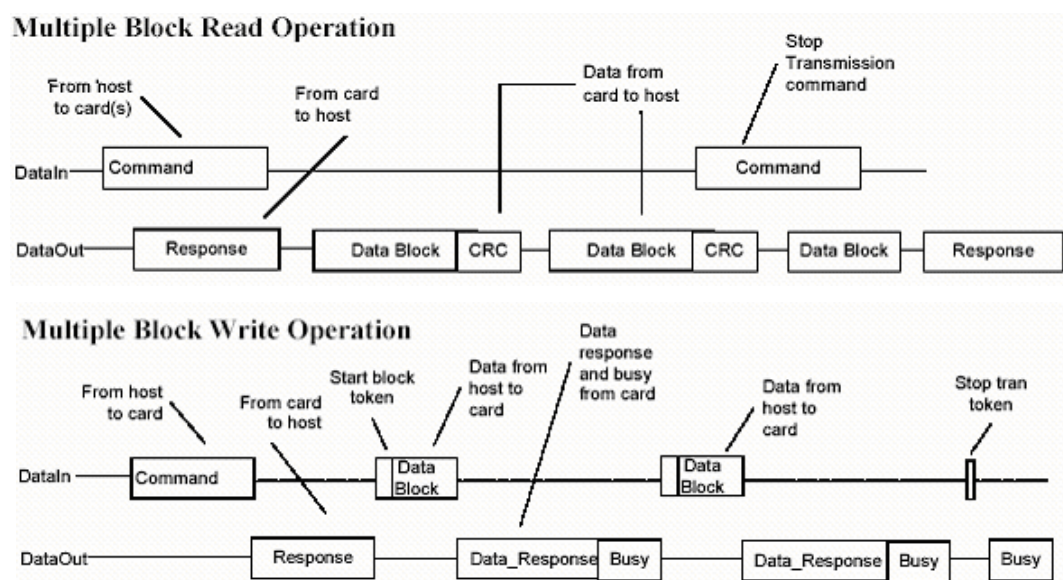


Figure 3.34: MMC Read and Write operations

I²C Serial EEPROM [40]

For the 512 byte buffer I2C two wire serial eeprom 24C512 from Atmel Corporation is used. The AT24C512 provides 524,288 bits (64 Kbytes) of serial electrically erasable and programmable read only memory (EEPROM) organized as 65,536 words of 8 bits each. The device is cascable feature allows up to 4 devices to share a common 2-wire bus.

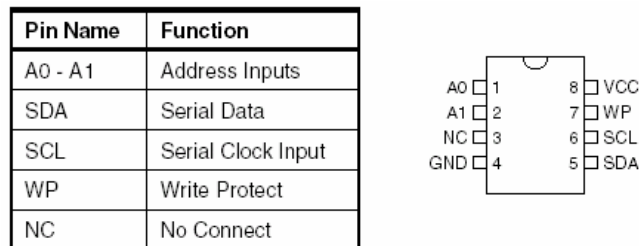


Figure 3.35: I2C EEPROM Pinout diagram and Pin functions

The serial clock (SCL) input is used to positive edge clock data into each EEPROM device and negative edge data out of each device. The Serial data (SDA) pin is bidirectional for serial data transfer.

The I2C EEPROM requires an 8-bit device address word following a start condition to enable the chip for a read or write operation (see Figure 3-36). The device address word consists of a mandatory one, zero sequence for the first five most significant bits as shown. This is common to all 2-wire EEPROM devices. The 512K uses the two device address bits A1, A0 to allow as many as four devices on the same bus. These bits must compare to their corresponding hardwired input pins.

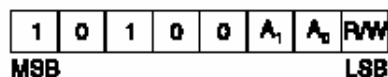


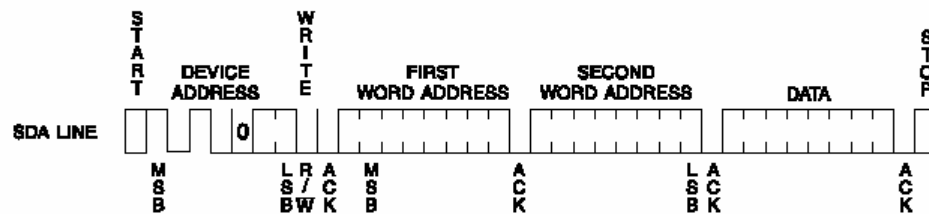
Figure3-36: I2C Device Address Format

Write and Read Operations

Byte Write: A write operation requires two 8-bit data word addresses following the device address word and acknowledgment. Following receipt of the 8-bit data word, the EEPROM will output a zero. The addressing device, such as a microcontroller, then must terminate the write sequence with a stop condition. At this time the EEPROM enters an internally-timed write cycle, t_{WR} (10 ms for 24c512), to the nonvolatile memory. See Figure 3-37

Page Write: The 512K EEPROM is capable of 128-byte page writes. A page write is initiated the same way as a byte write, but the microcontroller does not send a stop condition after the first data word is clocked in. Instead, after the EEPROM acknowledges receipt of the first data word, the microcontroller can transmit up to 127 more data words. If more than 128 data words are transmitted to the EEPROM, the data word address will “roll over” and previous data will be overwritten. See Figure 3-37

Byte Write



Page Write

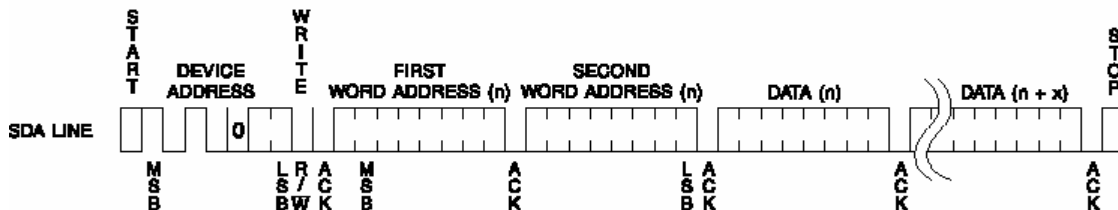


Figure 3.37: I2C Write Operation

Read operations are initiated the same way as write operations with the exception that the read/write select bit in the device address word is set to one. There are three read operations: current address read, random address read and sequential read.

Current Address Read: The internal data word address counter maintains the last address accessed during the last read or write operation, incremented by one. This address stays valid between operations as long as the chip power is maintained.

Sequential Read: Sequential reads are initiated by either a current address read or a random address read. After the microcontroller receives a data word, it responds with an acknowledge. As long as the EEPROM receives an acknowledge, it will continue to increment the data word

address and serially clock out sequential data words. When the memory address limit is reached, the data word address will “roll over” and the sequential read will continue. The sequential read operation is terminated when the microcontroller does not respond with a zero but does generate a following stop condition (see Figure 3-38).

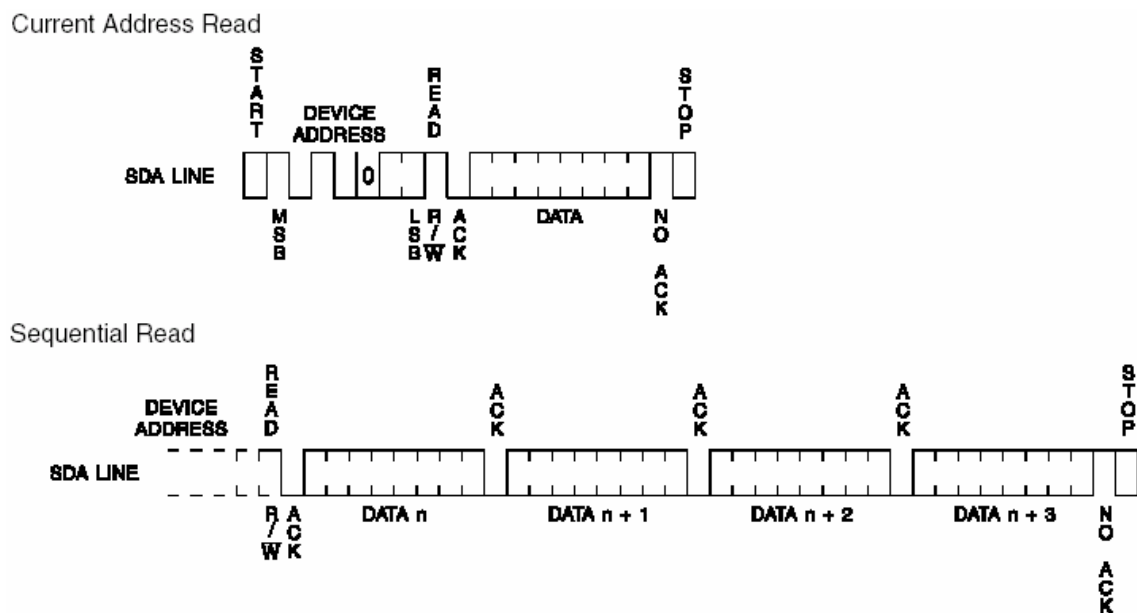


Figure 3-38: I2C Read Operation

3.6 Data Storage Circuit

As mentioned in section 3.5, all transactions between PIC and MMC must be in chunks of 512 bytes to read from and write to MMC, an external 64 KB I2C EEPROM is used. As shown in the Figure 3-39, RA4 and RA5 ports are used for the connection between I²C and PIC. RA4 and RA5 are used for serial clock and serial data (refer to Appendix F for PIC pin variables and Functions). The serial clock is generated by bus master (PIC) and 8 bit data transmitted serially, synchronized by the clock, on the bi-directional data line. Since there is no write protection on the EEPROM, WP port is connected to the

ground. A0 and A1 device address inputs are also connected to the ground. So the device address is chosen as hex A0.

For the MMC connection PORTE is used for clock, unidirectional data in and data out pins and RA3 is used for Chip select. For every command, a card (slave) is selected by asserting (active low) the CS signal. The CS signal must be continuously active for the duration of the SPI transaction (command, response and data). To supply MMC 3.3 volt, LM317 voltage regulator is used, as shown in the Figure 3-39

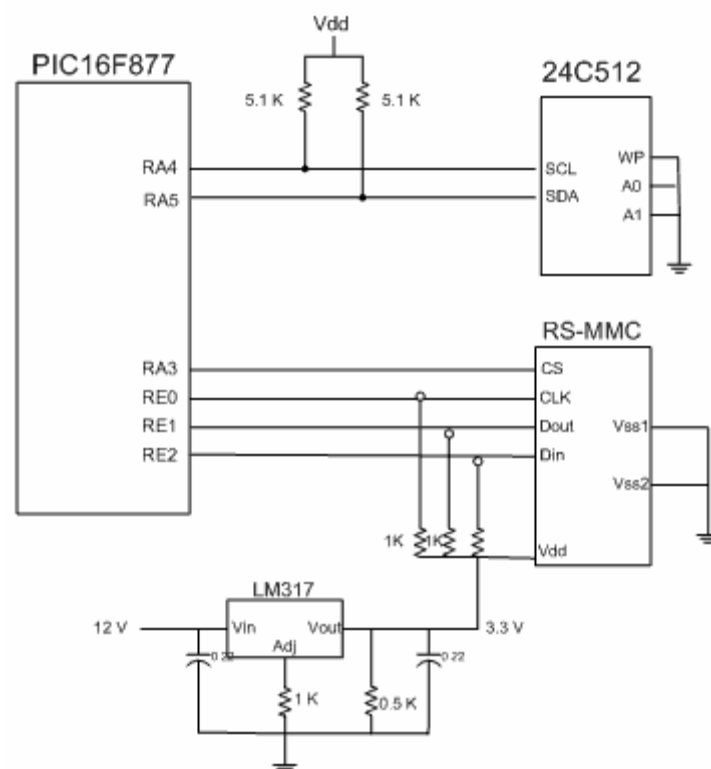


Figure 3-39: MMC storage Circuit

3.7 Motor driver circuit

As shown in Figure 3-40, to control LM629, six control line and eight I/O data bus is needed (see section 3.4). In the figure only one axis motor driver is shown. For I/O data bus PortC is selected since the other 8 pin ports (PortD and PortB) are used for USB interfacing and controlling PIC (refer to Appendix F for PIC pin variables and Functions). The host interrupt (HI) pin of LM629 is connected to RB0 pin to control whether the trajectory is complete or not. The other pins are used for controlling command writing and reading status.

For H-bridge LMD18201 is chosen from National Semiconductor. The LMD18201 is a 3A H-Bridge designed for motion control applications (see Appendix C). The device is built using a multi-technology process which combines bipolar and CMOS control circuitry with DMOS power devices on the same monolithic structure. The H-Bridge configuration is ideal for driving DC and stepper motors. The LMD18201 accommodates peak output currents up to 6A.

The LMD18201 can directly interface to any Sign/Magnitude PWM controller. The LM629 is a motion control processor that outputs a Sign/Magnitude PWM signal to coordinate either positional or velocity control of DC motors. The LMD18201 provides fully protected motor driver stage.

The Break input of LMD18201 is connected to RD7 pin of PIC. Break input is used to break a motor by effectively shorting its terminals. When braking is desired, this input is taken to a logic high level. RD7 is used to break the motors on the power on stage.

Index pulse input of LM629 is connected to logical high level because it is not used. The encoder interface must be connected to A, B input of LM629. If it is not connected in the right order the motor will run away (a condition characterized by the motor running continuously at high speed). Changing the order of A and B input solves the problem.

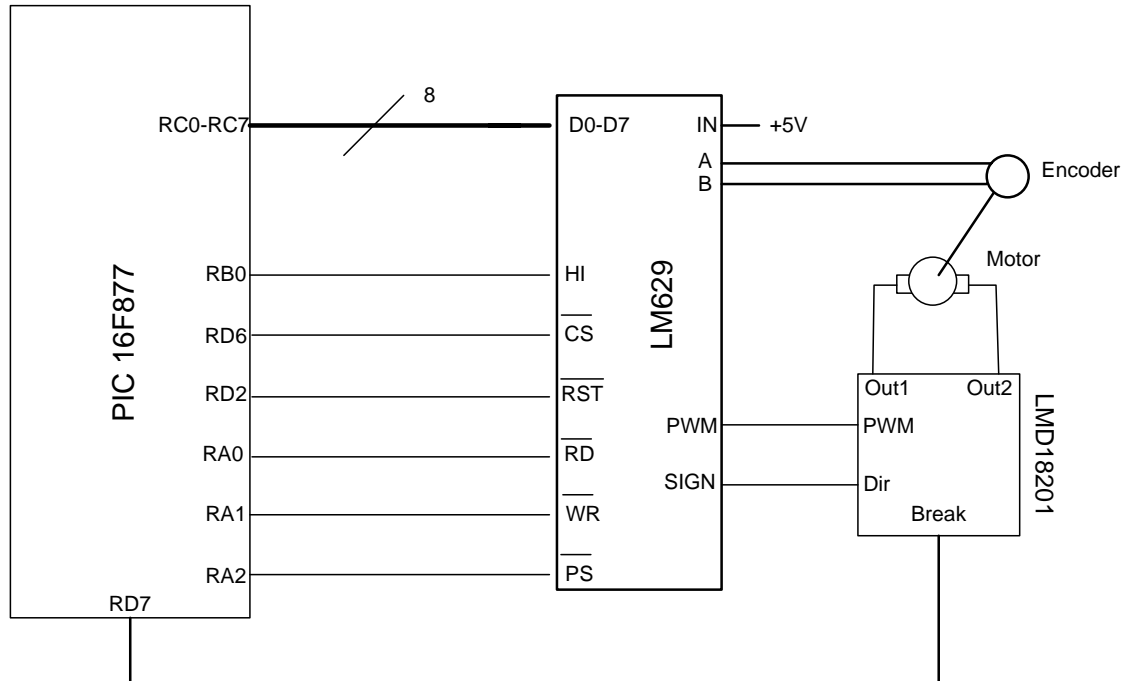


Figure 3-40: X axis Motor driver circuit

3.8 Buffer Circuit

Since there is not enough pin on PIC 16F877 to control three servo controllers, LM629, there is a need to multiplex the pins of PIC 16F877. To multiplex the control and the I/O lines, octal tri-state Buffers/Line Drivers/Line Receivers 74LS541 and octal bus transceiver 74LS245 (see Appendix D) integrated circuits are used.

The RC port of the PIC16F877 is used for 8 bit I/O data port. And the RA0-2 and RD2 ports are used for the pins which control the servo controllers (LM629). These ports are common for three servo controllers.

The RD1, RD5, RD6 and RD3 pins of the PIC16F877 are used to control the enable and direction pins of line and bus drivers. By these pins the flow of data between PIC and servo controllers is controlled. As shown in the Figure 3-41, RD6 pin is the enable pin of X axis

buffer and line I.Cs. RD1 is Y axis and RD5 is Z axis Enable pin. And direction pin RD3 is common for three axes buffer and line I.Cs. When the enable pin is logic high, the corresponding servo controller's line I/O and control line is isolated from PIC.

The selection of axis controller is done by pulling down the corresponding enable pin to logic low. When a command will be written, the data flow is from bus A (microcontroller side) to B (LM629 side) so the direction pin must be high. And to read data from LM629, the data flow is from bus B to A, the direction pin (RD3) must be low, as in the truth table in Appendix D.

To control trajectory complete interrupt from three LM629, the Host Interrupt (HI) outputs are ANDed with three input AND gate (Motorola MC74AC11N). And the output of the AND gate is connected to RB0 pin of PIC. To execute the next position, PIC waits for RB0 pin to become logical high level. When RB0 is high PIC gives the STarT command to all LM629 and until the trajectories are completed the HI pins will be logical low.

CHAPTER 4

MOTOR CONTROL SOFTWARE

In this study, Visual Basic software which digitizes two and three dimensional drawings is used, and a firmware is written for PIC16F877 by using PIC Basic Pro.

The digitizing software recognizes objects from standard image files (BMP, JPG, TIFF, etc.) and generates the points. Then the software analyzes these points and converts them to lines which constitute the object. The absolute distance of the start and end point of the lines are calculated according to the reference point (origin of a coordinate system) which is the beginning point and these distances are converted to motor's encoder count value. For example to move the XY table 4 cm the motor must turn 10 times (depends on the lead screw, drives the XY table) and if the encoder gives 2000 count per revolution the software must send 20000 count (hex4E20). Then these calculated distances converted to hexadecimal and sent through the PC's USB port in an order by the software. When PIC receives these position values, it stores to them to MMC or activate the motors according to the control command sent by the PC. The firmware is described in detail in section 4.2.

The firmware waits 3 bytes for each axis (6 bytes for two dimensional, 9 bytes for three dimensional moves). The first byte is the sign byte (sign of the axis hex0-positive, hex1-negative) and the other two bytes are the high and low byte the encoder count value of the desired position. For example to move the X axis of XY table from origin to 10 mm on the positive axis (assuming every 10 encoder count is 1 mm), the firmware must read hex00 (first byte), hex0064 (second and third bytes) .

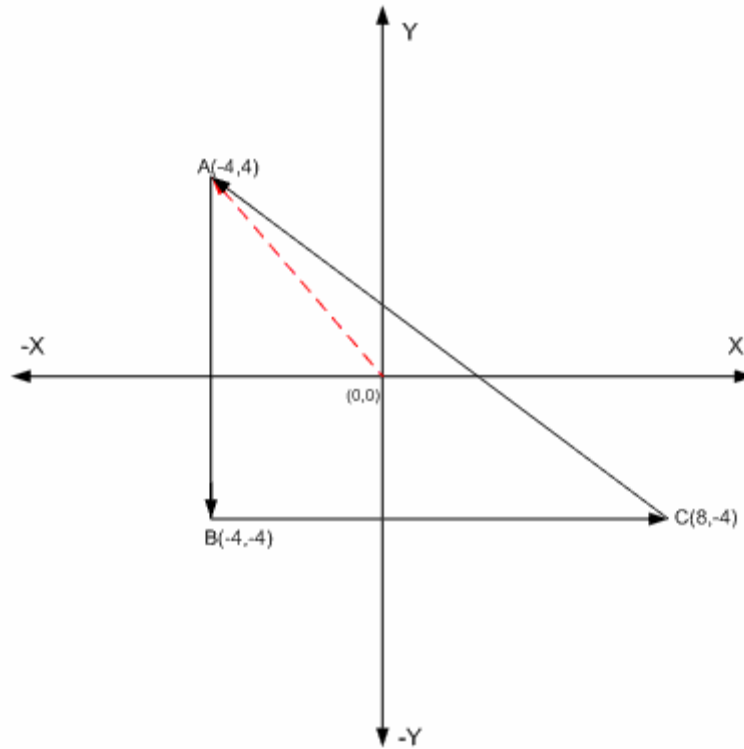


Figure 4-1: Trajectory Example

After digitizing the two dimensional object, the software calculates the start and end points of the lines A(-4,4), B(-4,-4), C(8,-4). The digitizing software sent the calculated points and PIC calculates the accurate acceleration and velocity of the motors according the absolute distances between the current and desired positions that the motors will cover. The calculated values are loaded to corresponding servo motor controller (LM629) to start the motion. When the start command is received, First the x and y axis motors moves from origin to point A and complete the trajectory, as shown in the Figure 4-1. From point A to B only the x axis motor and from point B to C x axis motor moves. But from point C to A both motors must start and finish the motion at the same time to move the workpiece through the straight line. In the Figure 4-1, x motor must move 12 cm and y motor must move 8 cm between the points C and A. So the ratio of velocity and acceleration of x and y motors must be directly proportional to the ratio of the absolute distances. For example, if the maximum velocity of x motor is 24 revolutions per second, y motor's maximum velocity must be 16 revolutions per second.

The following sections describe programming PIC microcontroller and the written firmware for this study.

4.1 PIC Programming

In this study, PICBASIC PRO [41] software is used to program the PIC. PICBASIC PRO converts BASIC programs into files that can be programmed directly into a PICmicro MCU. The PICBASIC PRO Compiler gives you direct access to all of the PICmicro MCU registers - I/O ports, A/D converters, hardware serial ports, etc. - easily and in BASIC. It automatically takes care of the page boundaries and RAM banks.

4.1.1 PIC Basic Pro

Simplicity and ease which higher programming languages bring in, as well as broad application of microcontrollers today, were reasons to incite some companies to adjust and upgrade BASIC programming language to better suit needs of microcontroller programming. By the help of BASIC programming language, developing applications is faster and easier with all the predefined routines which BASIC brings in, whose programming in assembly would take the largest amount of time. This allows programmer to concentrate on solving the important tasks without wasting his time. [41, 43]

First program written in PIC Basic

For writing BASIC program code, any text editor that can save the program file as pure ASCII text (without special symbols for formatting) can be used. For this purpose editors like Notepad or WordPad are also good. In this study CSMicro Systems CodeDesigner Lite [42] is used as the text editor. It is specially devised for program code writing. The advantage of this program package is that it takes care of the code syntax, free memory and provide more comfortable environment when writing a program.

The first step is the writing of a program code in some of enumerated text editors. Every written code must be saved on a single file with the ending .BAS exclusively as ASCII text. An example of one simple BASIC program - BLINK.BAS is given.

‘ Example of a program where the LED diode connected on PORT B pin 7 switches
‘ on and off every 0.5 seconds

Loop:

High PORTB.7	‘ Switched on LED on pin 7 of port B
Pause 500	‘ 0.5 sec pause
Low PORTB.7	‘ switched on LED on pin 7 of port B
Pause 500	‘ 0.5 sec pause
Goto Loop	‘ Go back to Loop
End	‘ End of program

When the original BASIC program is finished and saved as a single file with .BAS ending it is necessary to start PIC BASIC compiler. The compiling procedure takes place in two consecutive steps.

Step 1: In the first step compiler will convert BAS file in assembler s code and save it as BLINK.ASM file.

Step 2: In the second step compiler automatically calls assembler, which converts ASM type file into an executable HEX code ready for reading into the programming memory of a microcontroller.

In case of a syntax error of a program code, the compilation will not be successful and HEX file will not be created at all. Errors must be then corrected in original BAS file and repeat the whole compilation process.

Loading a program into the microcontroller memory

As a result of a successful compilation of a PIC BASIC program the following files will be created.

BLINK.ASM - assembler file

BLINK.LST - program listing

BLINK.MAC - file with macros

BLINK.HEX - executable file which is written into the programming memory

File with the HEX ending is in effect the program that is written into the programming memory of a microcontroller. The programming device with accessory software installed on the PC is used for this operation. Programming device is a contrivance in charge of writing physical contents of a HEX file into the internal memory of a microcontroller. The PC software reads HEX file and sends to the programming device the information about an exact location onto which a certain value is to be inscribed in the programming memory. PIC BASIC creates HEX file in a standard **8-bit Merged Intel HEX** format accepted by the vast majority of the programming software.

Besides reading of a program code into the programming memory, the programming device serves to set the configuration of a microcontroller. Here belongs the type of the oscillator, protection of the memory against reading, switching on of a watchdog timer etc. The connection between PC, programming device and the microcontroller is shown.

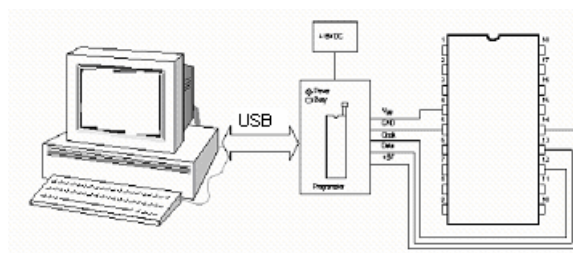


Figure 4-2: Programming Device

See Appendix E for Pic Basic Pro basics and Statement reference.

4.2 Firmware Description

PIC16F877's firmware is the code controlling the storage of data to the MMC, reading/sending data through USB and all the critical tasks of motion controller such as, calculating velocity and acceleration of each servomotor and loading the trajectory parameters to LM629. The developed firmware (see Appendix G) has five parts:

- **Initialization part:** Initialization part includes the variables and simply sets the initial conditions and activates certain peripherals of the micro-controller. For more information refer to Appendix G.
- **Storage part:** In this part, the MMC is initialized in SPI mode and position data is stored into and read from MMC.
- **Servo controller part:** In this part of the firmware, LM629 chips are initialized and the variables about trajectory like speed, acceleration and PID are updated. Also motors can be manually controlled by entering desired position.
- **Data processing part:** the stored position data is read from I2C EEPROM, processed and trajectory is started in this part.
- **Subroutine part:** this part includes the subroutines, frequently used by other parts.

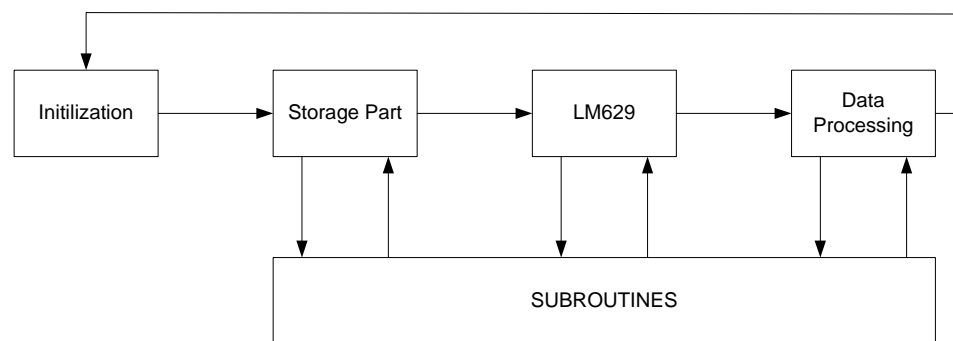


Figure 4-3: General Overview of the Firmware

4.2.1 Storage Part

This is the MMC related part of the firmware. This part includes 3 main modules which are described in the following sections. [37, 39]

As shown in the Figure 4-4, the firmware first waits for a control byte from PC. If there is data the firmware reads the control byte and checks it. According to the value of the control byte the program executes corresponding module.

MMC Initialization Module

In this module the MMC is reset and SPI mode is activated for the communication protocol between PIC and MMC. The MMC requires a defined reset sequence. After power on reset or software reset (CMD0), the card enters an idle state. By sending CMD1 repeatedly, MMC is polled until the “in-idle-state” bit in the card response indicates, by being set to 0 that the card completed its initialization processes and is ready for the next command.

To initiate the card in SPI, CS signal is asserted (negative) during the reception of the reset command (CMD0).

Read Index and MMC Module

If there is previously loaded position data in the MMC and if these data will be used, the index of the last byte is read from the address 65500 of the I2C EEPROM. The index is 6 byte long. The first 4 byte of the index is the address of the last sector and the last two byte is the number of byte loaded to the last memory sector. After reading these values, they assigned to **sector1**, **sector0** and **load_ee** variables to use in the data processing part of the firmware.

After reading index, the second part of the subroutine begins with **READMMC** label. Firmware reads the MMC memory sector (512 byte) from previously assigned 32 bit address. Firmware reads the 512 bytes one by one from the Dataout pin and write it to I2C eeprom. When all data read from I2C EEPROM, Firmware jumps to data processing part or LM269 part according to control variable **read_cntrl**.

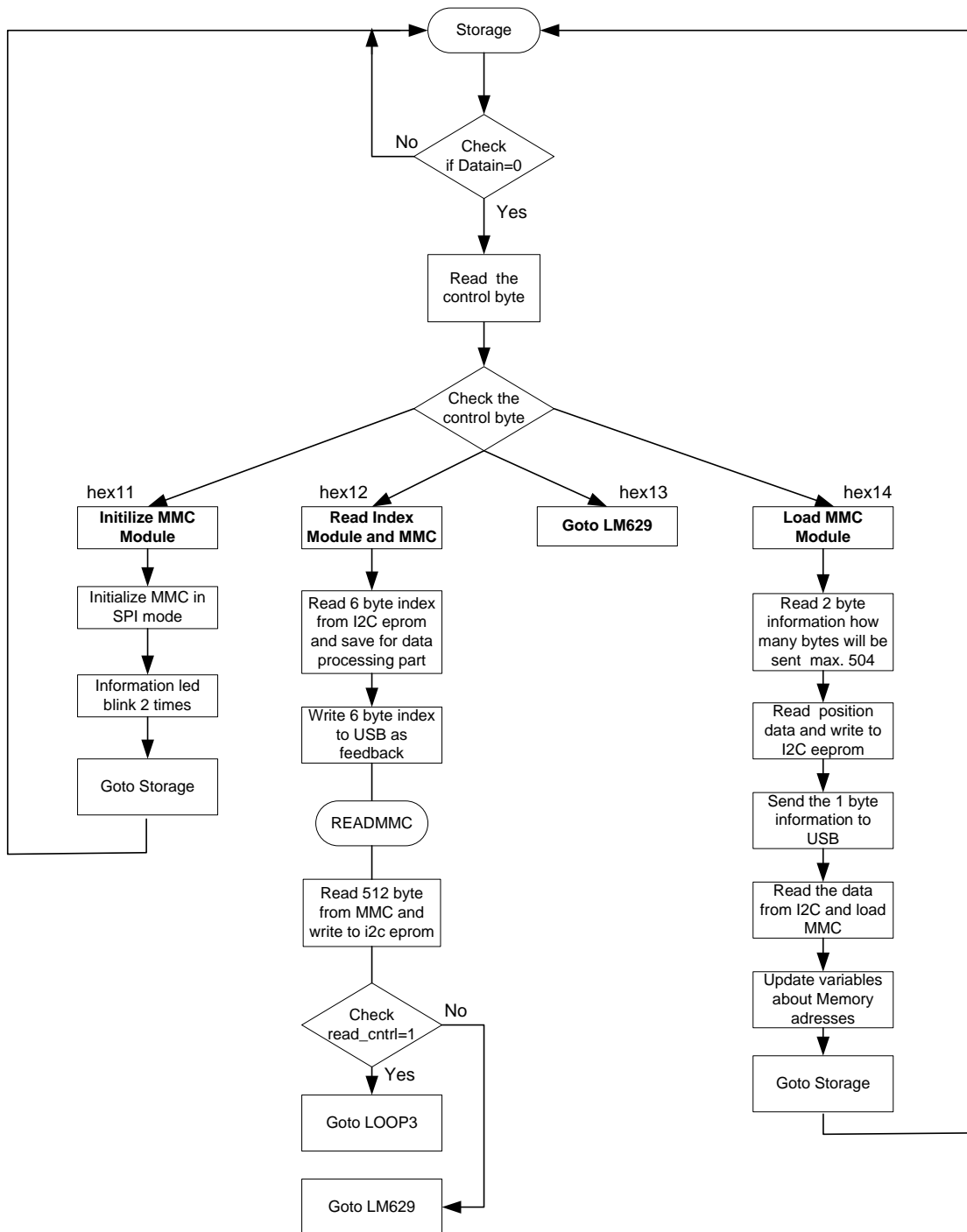


Figure 4-4: Flowchart of Storage part

Load MMC Module

In this module the number of data bytes that will be loaded to the MMC memory sector is read from USB. According to that value the firmware calculates the loop count of For..Do loop. In every loop the position data is loaded into the 64 byte array **in_array** variable and written into I2C EEPROM. After loading data into I2C EEPROM one byte feedback is sent to PC. Then the data in the EEPROM is read one by one and loaded to MMC memory sector. At the end the memory address variables are updated. And the index of the last byte is written to I2C EEPROM.

4.2.2 LM629 Part

This part is LM629 servo controller and the parameters about trajectory related part of the firmware. In this part there are eight main modules, as shown in the Figure 4-6.

Trajectory Complete Check Module

After every processing of position data in MMC part, the firmware loops back to label **LOOP3** in LM629 part. In this module first the USB interface is checked whether a control byte is send or not. If there is data, the control byte is read and executed. If there is no data, firmware waits for the interrupt that indicates the trajectory is completed by all motors. To execute the other positions, RB0 pin (see Figure 3-40) is checked by the firmware and when this pin is high the firmware jumps to Data Processing part. All saved positions are executed one by one by the help of this module.

Hardware Reset Module

The hardware reset of LM629 is described in section 3.4.8 Programming LM629 section. In this module all the servo controllers are reset and the status byte is checked in the order of X, Y, Z axis controllers. If the reset is successful, the information led blinks three times. Then the previously assigned PID parameters are loaded to servo controllers.

Stop Module

To stop the motion of motors, stop smoothly bit of the trajectory control word (See Figure 3-30) is set and sent to servo controllers. After loading the trajectory control word (hex40), by giving the STarT command to LM629, the motors stops with the current programmed acceleration. After stopping the motors, the current positions are read by the RDRP subroutine and equated to position variables (**SIGNxcurr**, **SIGNycurr**, **SIGNzcurr**, **xcurr**, **ycurr**, **zcurr**). Then read address variable of the EEPROM is taken 9 steps back. So by sending start command byte (hex26), the execution of the points starts again where it is stopped. And the stop flag (**frstop**) is set. This flag is checked at the end of the data processing part. And if it is set the firmware jumps to label **LM629** from data processing part. At the end the firmware loops to LM629 label.

Manual Load Position Module

The motors can be driven manually by sending the desired positions of X, Y, Z motors according to the home position (origin). The firmware waits for 3 bytes for each axis motor. After reading nine bytes of position data, the firmware equates these positions to **xread**, **signx**, **yread**, **signy**, **zread** and **signz** variables, then the **frusb** flag, indicates the 9 byte position data is loaded manually, is set. At the end, firmware jumps to **LTRJ2** label, in the data processing part, to execute the loaded point.

Update PID Module

To update the PID parameters of the servo controller this module is used. After the first control byte (hex24), the firmware waits for the second control byte. According to second control byte, firmware updates the corresponding servo controllers PID parameters (hex0-all, hex1-X, hex2-Y, hex3-Z axis motor). In the firmware the filter control word, refer to the section 3.4.8 Figure 3-29, is set to hex 000F. By this value the derivative sampling term is set to one and the servo controllers waits for all PID parameters. So the firmware waits 8 byte PID data to update servo controllers. After reading 8 bytes these parameters are loaded and updated by UDF command of LM629.

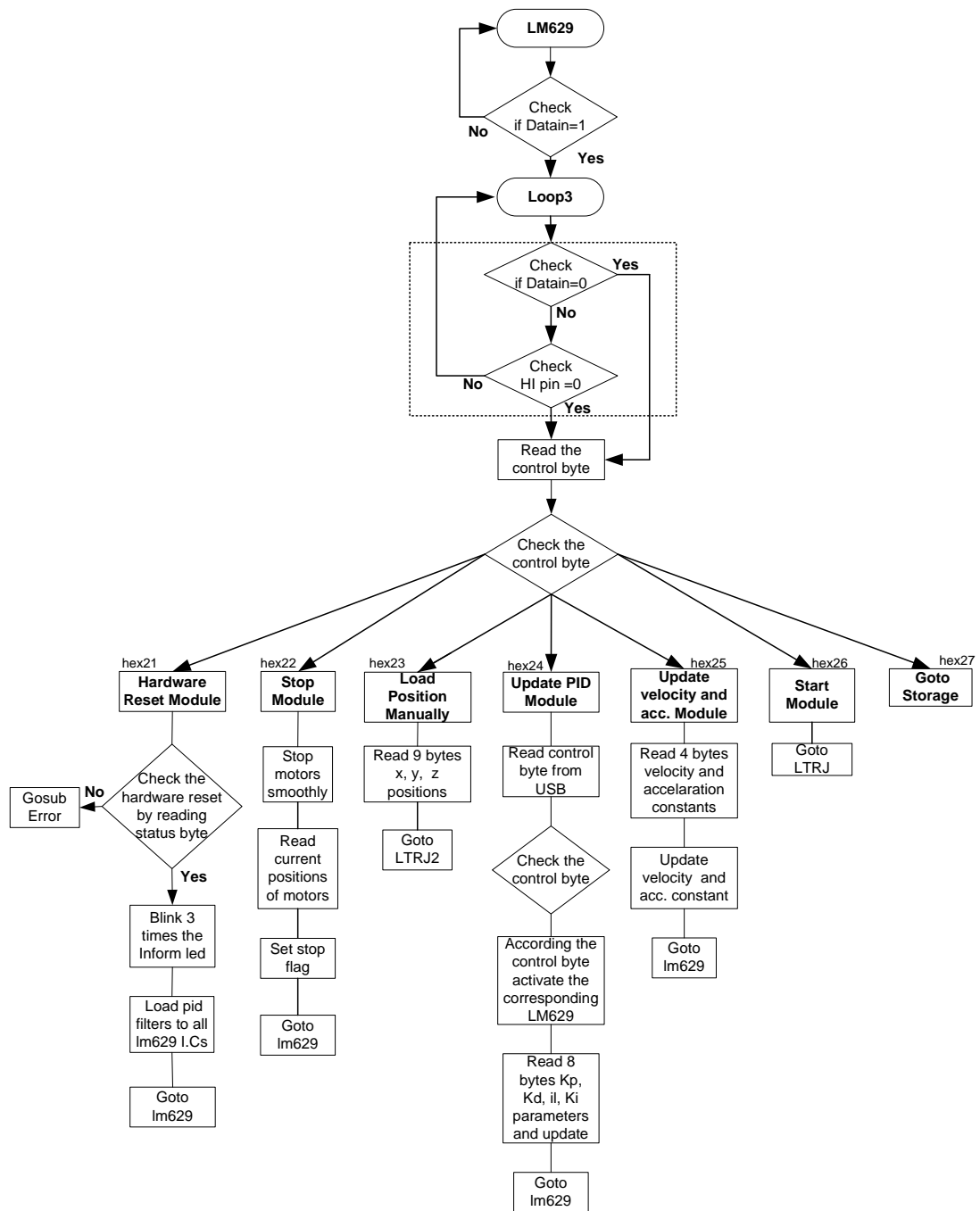


Figure 4-5: Flow chart of LM629 part

Update Velocity and Acceleration Module

The velocity and acceleration parameters are calculated in the data processing part by the help of the velocity and acceleration variables, **cv** and **acc**. The velocity of the system is directly proportional to these variables. These variables are equated to 7 and 1 in the initial part of the firmware. To update these variables this module is used. The firmware waits four data bytes, two for velocity and two for acceleration variables. After reading four data bytes, firmware equates them to **cv** and **acc** variables.

Other Modules

There is also **Start module** in the LM629 part. To process the position data loaded in to MMC the firmware jumps to data processing part by this module. And to loop back to Storage part **Goto storage module** can be used.

4.2.3 Data processing Part

In this part, previously loaded data is read from I2C EEPROM, velocity and acceleration parameters are calculated and LM629 servo controllers are programmed and at last the trajectory is started.

The 9 byte position data is read from EEPROM sequentially and equated to **signx, xread, signy, yread and signz, zread** position variables. And **rd_addr** variable is incremented by nine, for the next I2C reading. These variables are sent to position subroutine to calculate the absolute position and the direction of the motion.

The velocity and acceleration of the motors are calculate by multiplying the absolute position and the velocity and acceleration constants. the high word of the position data (hxread, hyread and hzread) is determined according to sign byte (signx, signy and signz). If sign byte is zero (positive), high word of the position is assigned to zero and if sign byte is one (negative), high word is assigned to hexFFFF. And low word of position data is subtracted from hexFFFF, if the sign byte is negative. At last the calculated trajectory values are loaded to servo controllers one by one. While loading the parameters the trajectory control word of LM629, Figure 3-30, is set

to hex2A. This means acceleration, velocity and position will be loaded. Then the trajectory is started by giving start command to all LM629 servo controllers.

Then the **frusb** flag is checked whether the position data is sent manually or not. According to the flag value, firmware jumps to label **LM629** or go on its ordinary execution. Then the index of the trajectory (sector0,sector1, load_ee) is checked whether all trajectory is completed or not. If completed hex CC is sent to USB and the firmware returns MMC part. If it is not completed, the EEPROM read address variable (**rd_addr**) is compared with the limit address (**limitaddr**) which is 504. if the **rd_addr** is equal or greater than the limit address, the read address is equated to zero and 32 bit MMC address is incremented by 512 (hex200) and firmware jumps to label **READMMC**. After checking address variables the stop flag (**frstop**) is checked.

If it is set, firmware jumps to label **LM629**. At the end of the data processing part, if there is no extraordinary situation, firmware loops to label **LOOP3** to complete trajectory loaded to MMC.

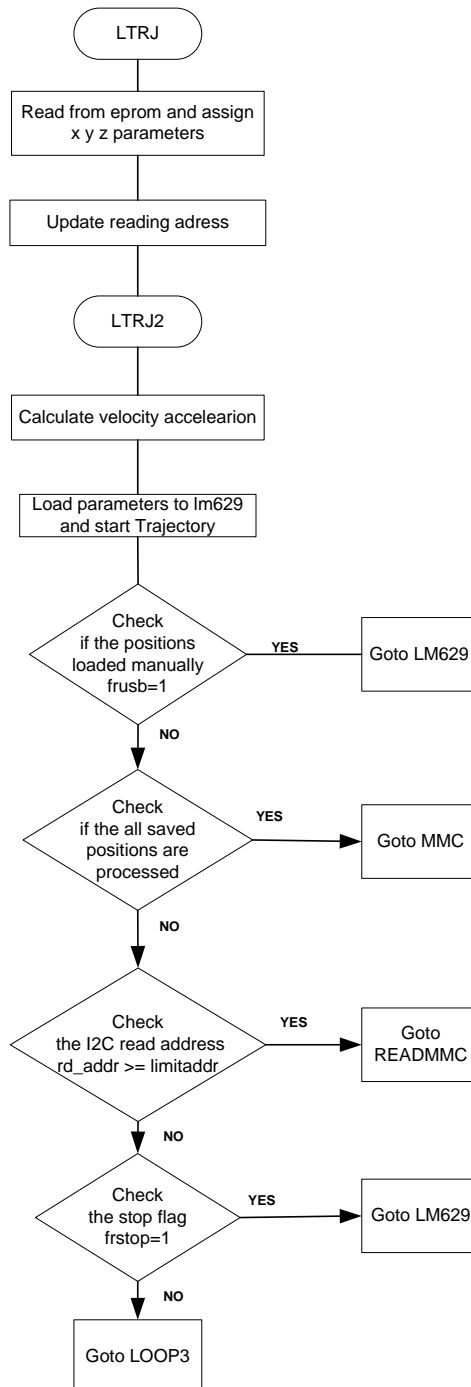


Figure 4-6: Flowchart of data processing part

4.2.4 Subroutines

To reduce redundancy and to improve readability and ease of extension of a program, some subroutines are coded in the firmware

4.2.4.1 MMC Subroutines

To initialize, read and write MMC the following subroutines are used. [37, 39]

Send_cmd

To send command to MMC this subroutine is used in the firmware. In this subroutine the command (**cmd** variable) and 32 bit address is sent by the help of the shiftout5 subroutine. At the end of the subroutine the response of the card is checked. The card sends this response (R1) token after every command [37]. This response is one-byte long, the MSB is always set to zero, and the other bits are error indications (1= error).

Shiftout_dat and shiftout5-1

To send an assigned value to MMC these subroutines are used. In shiftout_dat subroutine the **cntr** and **dat** variables are assigned in storage part. The same **dat** variable is sent from one to assigned **Cntr**.

Shiftout5-1 subroutines are called by the main program according the number of argument that will be sent. To sent 3 arguments (variable **arg**) shiftout3 is called.

Shiftin_Res

To read the response of the MMC from the Dataout pin of MMC (SO pin of PIC) this subroutine is used.

Chk_res

To check the response of MMC to the written commands this subroutine is used. The **chk** variable is compared with response (**res** variable) in a while loop. The dataout line is read in the while loop until the expected response (**chk**) is read.

4.2.4.2 LM629 subroutines

Usbread

To read the data sent from PC through the USB port this subroutine is used. In the main program the number of data byte that will be read from USB must be assigned to **readusb** variable. First DATAIN (PORTB.4) is checked, if there is data or not. When DATAIN is zero, the data is read by strobing RDUSB (PORTB.1) logical low then logical high again. The data taken by PORTD is equated to 64 byte array **in_array**.

Check

This subroutine is used to read the status byte of LM629 as described in section 3.4.8. After reading the status byte, according to variable x, firmware returns (x=1) from subroutine after hardware reset or interrupt reset of LM629, or the busy bit of status byte is checked after the host writes a command byte or reads or writes the second byte of a data word (x=2). The firmware continues to read status byte and to check busy bit, until it is reset to zero. While the busy-bit is set, the LM629 will ignore any commands or attempts to transfer data.

Hwreset

Immediately following power-up, a hardware reset must be executed. Hardware reset is initiated by strobing **rst** (PORTA.1) logic low for a minimum of eight LM629 clock periods. The reset routine begins after RST is returned to logic high. The reset execution time is maximum 1.5 ms.

Wrcomm

WRCOMM subroutine is used for writing commands to LM629. The value of COMMAND variable is assigned and then it is written to LM629 by equating COMMAND variable to PORTC and pulling **cs**, **ps** and **wr** pins logical low. At the end CHECK subroutine is called for check the busy bit.

Wwrd

When writing data, trajectory or PID filter parameters etc., WWRD subroutine is used. The **in_array[0]** and **in_array[1]** variables are also used here as high and low byte data that will be

loaded The data is assigned in the main program. The data is written by strobing **wr** pin while PS is high. At the end CHECK subroutine is called for check the busy bit.

Position

In this subroutine, the calculations about position and direction of movement are made. Since servo controllers are programmed to operate in position mode, all position values are calculated according to the home position where LM629 chips are reset. The target position and sign values, according to the origin, are read from USB and equated to **xread**, **yread**, **zread**, **signx**, **signy** and **signz** variables. Then the difference between the current and target position is taken to calculate the absolute distance that the motor will take. The actual positions are equated to **posx**, **posy** and **posz** variables. By the help of these variables the velocity and acceleration of each motor is calculated in the main program. And to keep update the current positions and sign values for the next calculations, the target positions and signs are equated to **xcurr**, **ycurr**, **zcurr**, **signxcurr**, **signycurr** and **signzcurr**.

Rdrp

To learn the real position of motors RDRP subroutine is used in trajectory module and stop module of main program. After RDRP command is written to all servo controllers, the RDRP subroutine reads the 32 bit position data from the profile generator of LM629. And the lower 16 bit position data is equated RDRP1 and RDRP2 variables and written to USB for feedback. And **rdrp3** variable is used in the stop module to determine the sign of the axis. If **rdrp3** is zero, current position is on the positive axis. If **rdrp3** is hexFF, current position is on the negative axis.

Error and Blink

If the reset procedure of servo controllers is not successful, ERROR subroutine is called to give information by blinking the D4 led of DIP-IO26 card.

BLINK subroutine is used for giving information by blinking the D4 led of DIP-IO26 card. For example, when the initialization of MMC is completed it blinks two times, when the hardware reset of LM629 is successful, it blinks three times. By assigning **bcount** variable in the firmware, the count of blink is determined.

Clock

This subroutine is used to give a clock pulse to D flip-flop which is connected to PORTD of PIC microcontroller.

CHAPTER 5

CONCLUSION

In this study, a high precision three-axes DC servo motor driver has been developed. The operating principle of the designed system and used parts such as PIC microcontroller, LM629, USB interface and the other integrated circuits are explained.

In the high voltage sphere gap application as in the machine positioning table application, the mechanical accuracy of the systems were found to be limited to the resolution of the optical position encoders used.

The PIC 16F877 microcontroller, which is used in the design, was described in detail. Additionally, the external memory usage with microcontrollers is mentioned. Also operating principle of motion controller processor LM629 and the control method by microcontroller is described. The firmware of microcontroller written in PicBasic Pro including the procedures of serial communication, SPI and I2C, between memories and PIC and programming LM629 is presented.

The driver has the ability to store the position data up to the memory capacity of used MMC card. In this study 128 MB MMC is used. By the storage ability, the system can work independent of a PC. Just giving a start command, the motor driver begins to work by reading previously loaded position data. Also the loaded data can be used several times. If the position data is below 512KB I2C EEPROM can also be used instead of MMC, by using cascade feature of I2C EEPROM. The only drawback of designed system's storage unit is waiting state of after a write operation to I2C EEPROM. If a standard I2C EEPROM is used, a 5 or 10 ms pause must be inserted after each write statement depending on the EEPROM used.

For future work, to increase the speed of communication between PC and Microcontroller USB 2.0 interface can be implemented to the system. And with PIC microcontrollers only 16 bit mathematical operations can be done. By using more powerful microcontrollers this can be increased to 32 bit. So the operating range of the driver system can be increased Finally to make the system totally independent from a PC, a controller module by the help of a microcontroller circuit with a keypad to enter commands and LCD panel can be designed.

REFERENCES

- [1] E. Kuffeland-M.Abdullah, ” High Voltag Engineering”, Pergamon Press Ltd., 1979
- [2] Sefa Akpınar, “Yüksek Gerilim Tekniğini Temelleri”, K.A.T.Ü. Basımevi, 1997
- [3] A.J. Schwab, “High Voltage Measurement Techniques”, MIT Press, 1972
- [4] Lux Jim, “High voltage experimenter’s Handbook”, <http://home.earthlink.net/~jimlux/hv/sphgap.htm>, Last Checked: 15.june.2005
- [5] Hans B. Kief – T. Frederick Warters, “Computer numerical control” , The McGraw-Hill Companies, Inc., Second Edition, 1992.
- [6] Barry Leatham-Jones, “Introduction to Computer numerical control”, Longman, 1986
- [7] G. E. Thyer, “Numeric Control of Machine Tools”, Oxford:Butterworth-Heinemann, Second Edition, 1991
- [8] Benjamin C. Kuo, “Theory and Applications of Step Motors”, West Publishing Co., 1974
- [9] National Instruments, “Fundamentals of Motion”, http://zone.ni.com/devzone/conceptd.nsf/webmain/722ECF56222AAD5086256F7B007072C4?opendocument&node=1506_US#7, Last Checked: 15.June.2005
- [10] Lepkowski J., “Motor Control Sensor Feedback Circuits”, Microchip Application Note No: AN894
- [11] Yedamale, Padmaraja, “Brushless DC (BLDC) Motor Fundamentals”, Microchip Technology Inc., Application Note No: AN885

- [12] Reston Condit, “Brushed DC (BDC) Motor Fundamentals”, Microchip Technology Inc., Application Note No: AN905

- [13] Ward Brown, “Brushless DC Motor Control Made Easy”, Microchip Technology Inc., Application Note No: AN857

- [14] Douglas W. Jones, “Control of Stepping Motors”, <http://www.cs.uiowa.edu/~jones/step#introduction>, Last Checked: 15.June.2005

- [15] Solarbotics, “Step motor drive circuit basics “, <http://library.solarbotics.net/pdflib/pdf/drive.pdf>, Last Checked: 15.June.2005

- [16] A.E. Fitzgerald, Charles Kingsley, Jr, Stephen D. Umans, “Electric Machinery”, Fifth Edition in SI Units, McGraw-Hill, 1992.

- [17] P.P. Acarnley, “Stepping Motors: A guide to modern theory and practice”, Revised Second Edition, IEE Control Engineering Series 19, 1984.

- [18] Feza Başar, “Development of a 3 axes PC numerical control system for industrial applications”, METU MS Thesis, 2003

- [19] Brian Rhoney, Chad Zimmer, Derek Murr, “Principles of AC, DC, Linear, Step, and Servo Motors”, MAE 789 C, 2000

- [20] Freescale Semiconductor, “Motion Control”, <http://www.freescale.com/webapp/sps/site/homepage.jsp?nodeId=02nQXG>, Last Checked: 15.June.2005

- [21] Richard H. Engelmann, William H. Middendorf, “Handbook of Electric Motors”, M. Dekker, 1995

- [22] Machine Design, “Electric Motors Reference Center”, <http://www.electricmotors.machinedesign.com/BDEList.aspx>, Last Checked: 15.June.2005

- [23] PIC16F877 Datasheet, Microchip, DS30292C

- [24] PICmicro™ Mid-Range MCU Family Reference Manual, Microchip, DS33023A

- [25] John B. Peatman, "Design with PIC Microcontrollers", Prentice Hall, 1998
- [26] Micheal Predko, "Programming and Customizing PICmicro Microcontrollers", Second Edition, McGraw-Hill, 2001
- [27] Firat Beştepe, "Microcontroller-Based Multiport Communication System for Digital Electricity Meters", METU MS Thesis, 2004
- [28] John Iovine, "PIC Microcontroller Project Book", McGraw-Hill, 2000
- [29] Beyondlogic, "USB in a Nutshell", <http://www.beyondlogic.org/usbnutshell/>
Last Checked: 15.June.2005
- [30] Jan Axelson, "USB Complete", Second Edition, LVR, 2001
- [31] DLP Design, "DLP-IO26", <http://www.dlpdesign.com/usb-prev>, Last Checked:
15. June.2005
- [32] Future Technology Devices International Ltd., "FT8U245AM USB FIFO",
<http://www.ftdichip.com/FTProducts.htm#FT8U245AM>, Last Checked:
15. June.2005
- [33] AN-706: Application Note 706 LM628/629 User Guide, National Semiconductor, 2002
- [34] LM628 LM629 Precision Motion Controller, National Semiconductor, 2003
- [35] AN-693: Application Note 693 LM628 Programming Guide, National Semiconductor, 2002
- [36] National Semiconductor, "LMD18201 Product Folder", <http://www.national.com/pf/LM/LMD18201.html>, Last Checked: 15.June.2005
- [37] Sandisk SanDisk MultiMediaCard and Reduced-Size MultiMediaCard Product Manual, Version 1.0, Document No. 80-36-00320, 2004

- [38] Captain's Universe, "PIC-MMC Flash Memory Extension", <http://www.captain.at/electronics/pic-mmc/>, Last Checked: 15.June.2005

- [39] Compsys System Consultation, "MMC Project", http://www.compsys1.com/workbench/On_top_of_the_Bench/MMC_Project/mmc_project.html, Last Checked: 15.June.2005

- [40] Atmel Corporation, "Serial EEPROMS AT24C512", http://www.atmel.com/dyn/products/product_card.asp?family_id=647&family_name=Serial+EEPROM&part_id=2489, Last Checked: 15.June.2005

- [41] Micro Engineering Labs Inc., "PIC Basic Pro Compiler Manual", <http://www.microengineeringlabs.com/resources/pbpmanual/>, Last Checked: 15.June.2005

- [42] CSMicro Systems, "Code Designer Lite", <http://www.csmicrosystems.com/>, Last Checked: 15.June.2005

- [43] Nebojsa Matic, "Basic for PIC microcontrollers", 2003

APPENDIX A: MACHINE CODES

Table A-1: Mid-Range MCU Instruction Set

Mnemonic, Operands	Description	Cycles	14-Bit Instruction Word				Status Bits Affected	Notes	
			MSb		LSb				
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRWF	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
BTFSC	f, b	Bit Test f, Skip if Clear	1 (2)	01	10bb	bfff-	ffff		3
BTFSS	f, b	Bit Test f, Skip if Set	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
CALL	k	Call subroutine	2	10	0kkk	kkkk	kkkk		
CLRWDI	-	Clear Watchdog Timer	1	00	0000	0110	0100	TO,PD	
GOTO	k	Go to address	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
RETFIE	-	Return from interrupt	2	00	0000	0000	1001		
RETLW	k	Return with literal in W	2	11	01xx	kkkk	kkkk		
RETURN	-	Return from Subroutine	2	00	0000	0000	1000		
SLEEP	-	Go into standby mode	1	00	0000	0110	0011	TO,PD	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

Note 1: When an I/O register is modified as a function of itself (e.g., `MOVF PORTB, 1`), the value used will be that value present on the pins themselves. For example, if the data latch is '1' for a pin configured as input and is driven low by an external device, the data will be written back with a '0'.

- 2: If this instruction is executed on the TMR0 register (and, where applicable, d = 1), the prescaler will be cleared if assigned to the Timer0 Module.
- 3: If Program Counter (PC) is modified or a conditional test is true, the instruction requires two cycles. The second cycle is executed as a NOP.

APPENDIX B: DLP-IO26 USB Interface and Schematic

B.1) FTU245AM

Features

- Single Chip Fast Data Transfer Solution
- Send / Receive Data over USB at up to 1 M Bytes / sec
- 384 byte FIFO Transmit buffer / 128 byte FIFO receive buffer for high data throughput
- Simple interface to CPU or MCU bus
- No in-depth knowledge of USB required as all USB Protocol is handled automatically within the I.C.
- FTDI's Virtual COM port drivers eliminate the need for USB driver development in most cases.
- Compact 32 pin (7mm x 7mm) MQFP package
- Integrated 6MHz - 48MHz Clock Multiplier aids FCC and CE compliance
- Integrated 3.3v Regulator – No External Regulator Required
- 4.4v.. 5.25v Single Supply Operation
- UHCI / OHCI Compliant
- USB 1.1 Specification Compliant
- USB VID, PID, Serial Number and Product Description Strings in external EEPROM.
- Virtual COM Port Drivers for – Windows 98 and Windows 98 SE, Windows 2000, Windows XP, Windows Millennium **, Apple iMAC **, Linux **
- Application Areas
 - USB ISDN and ADSL Modems
 - High Speed USB PDA Communications
 - USB I/F for Digital Cameras
 - USB I/F for MP3 players
 - High Speed USB Instrumentation
 - USB - USB data transfer cables
 - USB -USB null-modem cables

General Description

The FT8U245AM provides an easy cost-effective method of transferring data to / from a peripheral and a host P.C. at up to 8 Million bits (1 Megabyte) per second. It's simple FIFO-like design makes it easy to interface to any CPU (MCU) either by mapping the device into the Memory / IO map of the CPU, using DMA or controlling the device via IO ports.

To send data from the peripheral to the host P.C. simply write the byte wide data into the device when the transmitter empty status bit is not active. If the (384 byte) transmit buffer fills up, the device de-asserts transmit empty in order to stop further data being written to the device until some of the FIFO data has been transferred over USB. When the host P.C. sends data to the peripheral over USB, the device will assert the receiver full status bit to let the peripheral know that data is available. The peripheral then reads the data until the receiver full status bit goes inactive, indicating no more data is available to read.

By using FTDI's virtual COM Port drivers, the peripheral looks like a standard COM Port to the application software. Commands to set the baud rate are ignored – the device always transfers data at its fastest rate regardless of the application's baud rate setting.

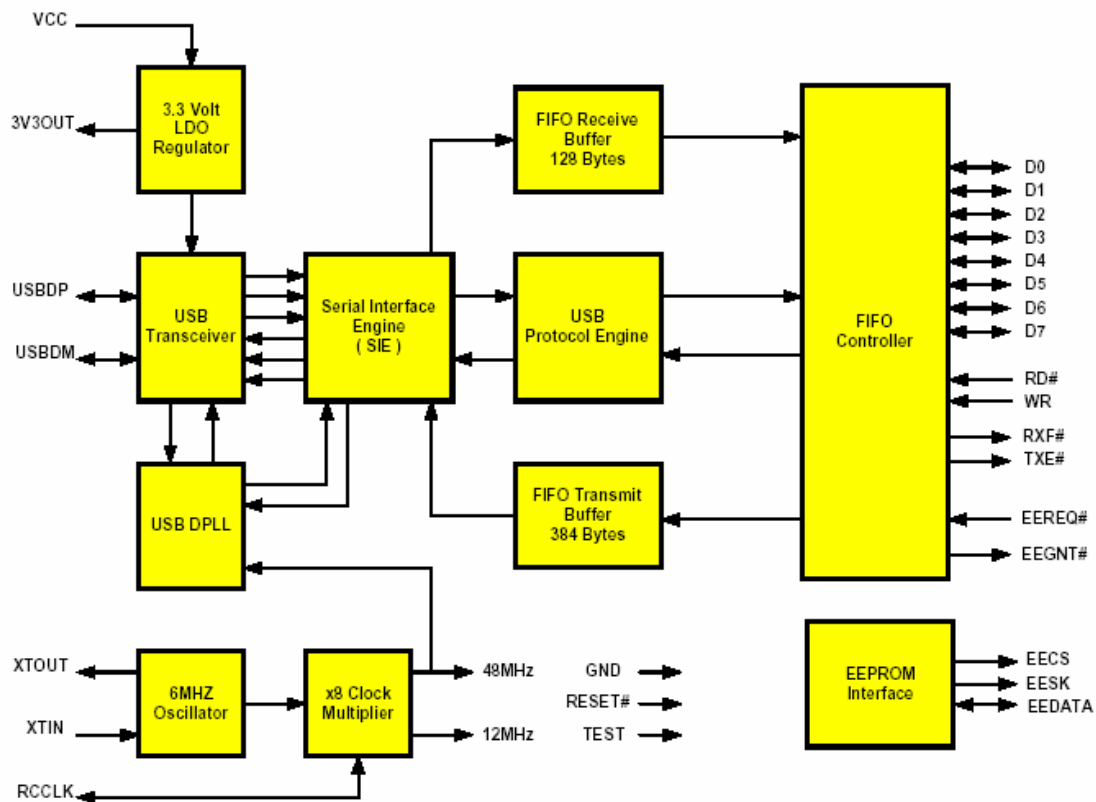


Figure B-1: FT8U245AM Block Diagram

FT8U245AM – Functional Block Description

3.3V LDO Regulator

The 3.3V LDO Regulator generates the 3.3 volt reference voltage for driving the USB transceiver cell output buffers. It requires an external decoupling capacitor to be attached to the 3V3OUT regulator output pin.

USB Transceiver

The USB Transceiver Cell provides the USB 1.1 full-speed physical interface to the USB cable. The output drivers provide 3.3 volt level slew rate control signaling, whilst a differential receiver and two single ended receivers provide USB data in, SEO and USB Reset condition detection.

USB DPPL

The USB DPPL cell locks on to the incoming NRZI USB data and provides separate recovered clock and data signals to the SIE block.

6MHz Oscillator

The 6MHz Oscillator cell generates a 6MHz reference clock input to the X8 Clock multiplier from an external 6MHz crystal or ceramic resonator.

X8 Clock Multiplier

The X8 Clock Multiplier takes the 6MHz input from the Oscillator cell and generates a 12MHz reference clock for the SIE, USB Protocol Engine and UART FIFO controller blocks. It also generates a 48MHz reference clock for the USB DPPL and the Baud Rate Generator blocks.

Serial Interface Engine (SIE)

The Serial Interface Engine (SIE) block performs the Parallel to Serial and Serial to Parallel conversion of the USB data. In accordance to the USB 1.1 specification, it performs bit stuffing / un-stuffing and CRC5 / CRC16 generation / checking on the USB data stream.

USB Protocol Engine

The USB Protocol Engine manages the data stream from the device USB control endpoint. It handles the low level USB protocol requests generated by the USB host controller and the commands for controlling the functional parameters of the UART.

FIFO Receive Buffer (128 bytes)

Data sent from the USB Host to the FIFO via the USB data out endpoint is stored in the FIFO Receive Buffer and is removed from the buffer by reading the FIFO contents using RD#.

FIFO Transmit Buffer (384 bytes)

Data written into the FIFO using WR# is stored in the FIFO Transmit Buffer. The Host removes Data from the FIFO Transmit Data by sending a USB request for data from the device data in endpoint.

FIFO Controller

The FIFO Controller handles the transfer of data between the external FIFO interface pins and the FIFO Transmit and Receive buffers.

EEPROM Interface

The FT8U245AM uses an external 93C46 EEPROM to customize the USB VID, PID, Serial Number and Strings of the FT8U245AM for OEM applications. The FT8U245 Virtual Com Port Drivers rely on a unique device serial number for to bind a unique virtual COM port to each individual device.

B-2) SCHEMATIC DIAGRAM

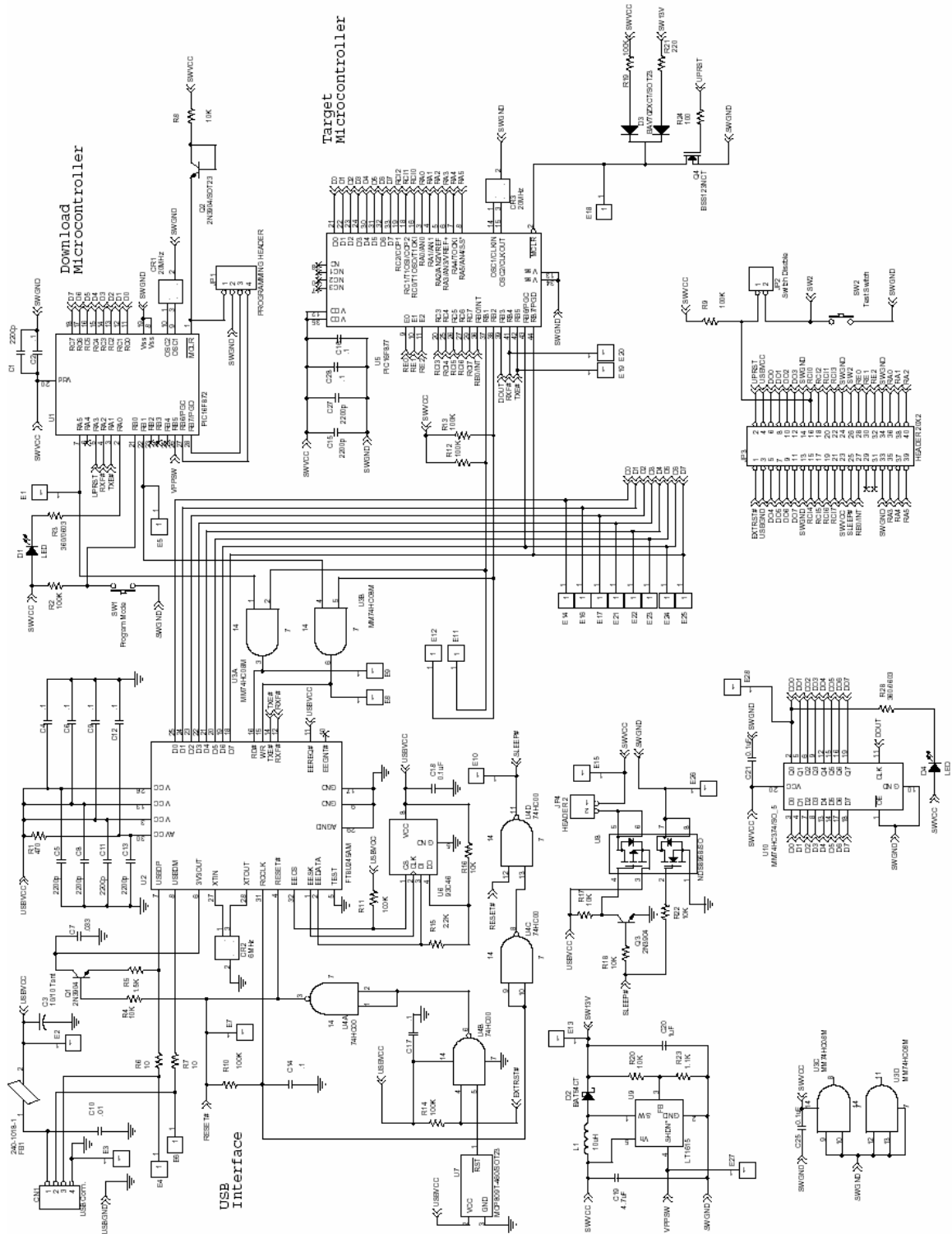


Figure B-2: DLP-IO26 Schematic Diagram

APPENDIX C: MOTOR DRIVER

LMD18201, 3A, 55V H-Bridge

General Description

The LMD18201 is a 3A H-Bridge designed for motion control applications. The device is built using a multi-technology process which combines bipolar and CMOS control circuitry with DMOS power devices on the same monolithic structure. The H-Bridge configuration is ideal for driving DC and stepper motors. The LMD18201 accommodates peak output currents up to 6A. Current sensing can be achieved via a small sense resistor connected in series with the power ground lead. For current sensing without disturbing the path of current to the load, the LMD18200 is recommended.

Features:

- Delivers up to 3A continuous output
- Operates at supply voltages up to 55V
- Low RDS(ON) typically 0.3 Ω per switch
- No “shoot-through” current
- Thermal warning flag output at 145°C
- Thermal shutdown (outputs off) at 170°C
- Internal clamp diodes
- Shorted load protection
- Internal charge pump with external bootstrap capability

Applications:

- DC and stepper motor drives
- Position and velocity servomechanisms
- Factory automation robots
- Numerically controlled machinery
- Computer printers and plotters

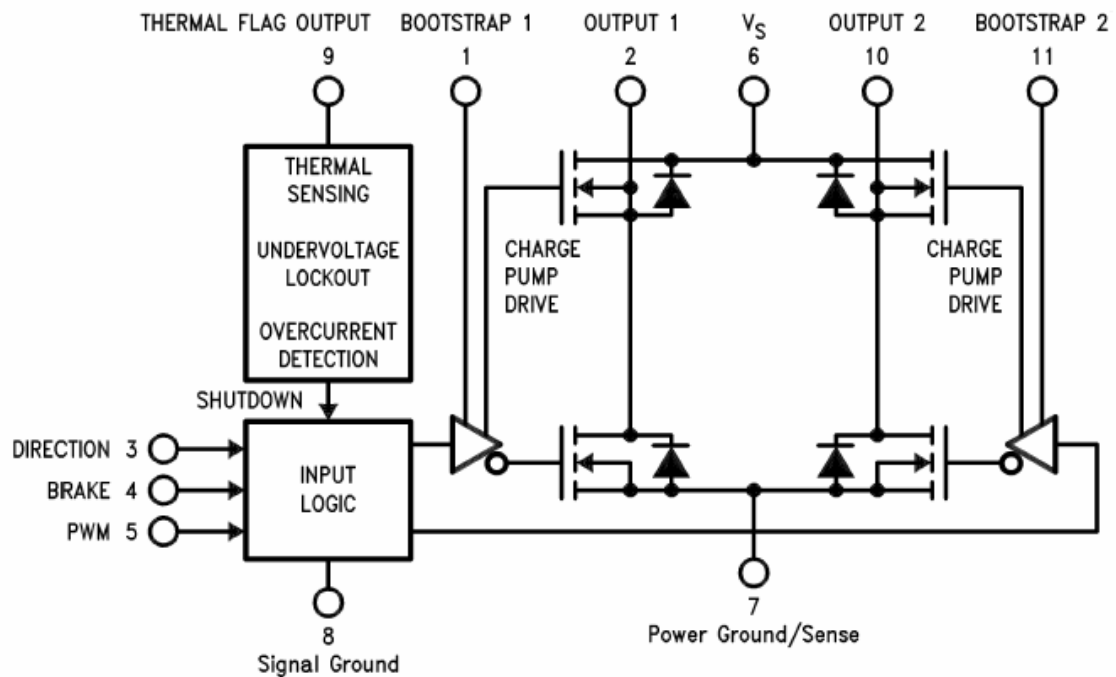


Figure C-1: Functional Block Diagram of LMD18201

Absolute Maximum Ratings

Total Supply Voltage (VS, Pin 6) 60V

Voltage at Pins 3, 4, 5 and 9 12V

Voltage at Bootstrap Pins (Pins 1 and 11) $V_{OUT} + 16V$

Peak Output Current (200 ms) 6A

Continuous Output Current (Note 2) 3A

Power Dissipation (Note 3) 25W

Sense Voltage (Pin 7 to Pin 8) +0.5V to -1.0V

Power Dissipation ($T_A = 25^\circ C$, Free Air) 3W

Junction Temperature, $T_{J(max)}$ 150°C

ESD Susceptibility (Note 4) 1500V

Storage Temperature, T_{STG} -40°C to +150°C

Lead Temperature (Soldering, 10 sec.) 300°C

APPENDIX D: BUFFER CIRCUIT

D.1) SN54/74LS245 OCTAL BUS TRANSCEIVER

74LS245 is used for control 8 bit I/O bus. The SN54/74LS245 is an Octal Bus Transmitter/Receiver designed for 8-line asynchronous 2-way data communication between data buses. Direction Input (DR) controls transmission of Data from bus A to bus B or bus B to bus A depending upon its logic level. The Enable input (E) can be used to isolate the buses.

- Hysteresis Inputs to Improve Noise Immunity
- 2-Way Asynchronous Data Bus Communication
- Input Diodes Limit High-Speed Termination Effects

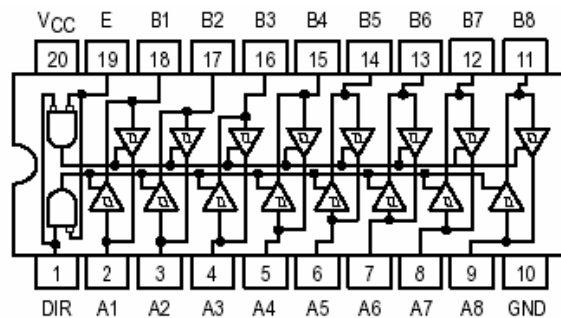


Figure D-1: Logic and Connection Diagram of LS245

Table D-1: Truth Table

INPUTS		OUTPUT
E	DIR	
L	L	Bus B Data to Bus A
L	H	Bus A Data to Bus B
H	X	Isolation

H = HIGH Voltage Level

L = LOW Voltage Level

X = Immaterial

D.2) DM74LS241 Octal TRI-STATE... Buffers/Line Drivers/Line Receivers

74LS541 is used for control the control line of the servo controller. These buffers/line drivers are designed to improve both the performance and PC board density of TRI-STATE buffers/drivers employed as memory-address drivers, clock drivers, and bus oriented transmitters/receivers

Features:

- TRI-STATE outputs drive bus lines directly
- PNP inputs reduce DC loading on bus lines
- Hysteresis at data inputs improves noise margins
- Data flow-thru Pinout (All inputs on opposite site from inputs)

The three-state control gate is a two input NOR such that if either G1 or G2 are high, all eight outputs are in the high impedance state. 74LS541 offers true data at the output.

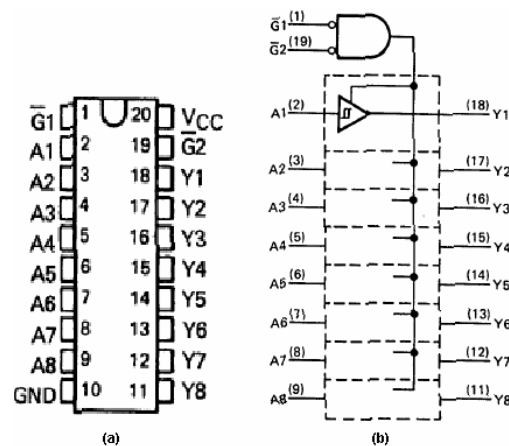


Figure D-2: Pinout and Logic Diagram of LS541

APPENDIX E: PIC BASIC PRO

E.1) PicBasic Pro Basics [35]

Line Labels

In order to mark statements that the program might wish to reference with GOTO or GOSUB commands, PBP uses line labels. Unlike many older BASICs, PBP doesn't allow line numbers and doesn't require that each line be labeled. Rather, any PBP line may start with a line label, which is simply an identifier followed by a colon (:).

```
here:   Serout 0,N2400,["Hello, World! ",13,10]
        Goto here
```

Variables

Variables are where temporary data is stored in a PicBasic Pro program. They are created using the VAR keyword. Variables may be bits, bytes or words. Space for each variable is automatically allocated in the microcontroller's RAM by PBP. The format for creating a variable is as follows:

```
Label   VAR   Size{.Modifiers}
```

Label is any identifier, excluding keywords, as described above. *Size* is **BIT**, **BYTE** or **WORD**. Optional Modifiers add additional control over how the variable is created. Some examples of creating variable are:

```
dog      var   byte
cat      var   bit
w0       var   word
```

The number of variables available depends on the amount of RAM on a particular device and the size of the variables and arrays. PBP reserves approximately 24 RAM locations for its own use. It may also create additional temporary variables for use in sorting out complex equations.

Numeric Constants

PBP allows numeric constants to be defined in the three bases: decimal, binary and hexadecimal. Binary values are defined using the prefix '%' and hexadecimal values using the prefix '\$'. Decimal values are the default and require no prefix.

```
100      ' Decimal value 100
%100     ' Binary value for decimal 4
$100     ' Hexadecimal value for decimal 256
```

Ports and Other Registers

All of the PICmicro registers, including the ports, can be accessed just like any other byte-sized variable in PicBasic Pro. This means that they can be read from, written to or used in equations directly:

```
PORTA = %01010101      ' Write value to PORTA
anyvar = PORTB & $0f     ' Isolate lower 4 bits of PORTB and
                          place result in anyvar
```

Pins

Pins may be accessed in a number of different ways. The best way to specify a pin for an operation is to simply use its PORT name and bit number:

```
PORTB.1 = 1      ' Set PORTB, bit 1 to a 1
```

To make it easier to remember what a pin is used for, it should be assigned a name using the VAR command. In this manner, the name may then be used in any operation:

```
led      var      PORTA.0      ' Rename PORTA.0 as led
High     led                               ' Set led (PORTA.0) high
```

Pins may be referenced by number (0 - 15), name (e.g. Pin0, if one of the bsdefs.bas files are included or you have defined them yourself), or full bit name (e.g. PORTA.1). Any pin or bit of the microcontroller can be accessed using the latter method.

The pin names (i.e. Pin0) are not automatically included in your program. In most cases, you would define pin names as you see fit using the VAR command:

```
led var PORTB.3
```

When a PICmicro powers-up, all of the pins are set to input. To use a pin as an output, the pin or port must be set to an output or a command must be used that automatically sets a pin to an output.

To set a pin or port to an output (or input), set its TRIS register. Setting a TRIS bit to 0 makes its pin an output. Setting a TRIS bit to 1 makes its pin an input. For example:

```
TRISA = %00000000           ' Or TRISA = 0
```

sets all the PORTA pins to outputs.

```
TRISB = %11111111           ' Or TRISB = 255
```

sets all the PORTB pins to inputs.

```
TRISB = %11111111           ' Or TRISB = 255
```

Sets all the even pins on PORTC to outputs, and the odd pins to inputs. Individual bit directions may be set in the same manner.

```
TRISA.0 = 0
```

sets PORTA, pin 0 to an output. All of the other pin directions on PORTA are unchanged.

Comments

A PBP comment starts with either the **REM** keyword, the single quote ('), or the semi-colon (;). All following characters on this line are ignored.

Include

Other BASIC source files may be added to a PBP program by using **INCLUDE**. These files may be included in programs where they are necessary, but kept out of programs where they are not needed. The included files source code lines are inserted into the program exactly where the **INCLUDE** is placed.

```
INCLUDE "modedefs.bas"
```

Define

Some elements, like the clock oscillator frequency and the LCD pin locations, are predefined in PBP. **DEFINE** allows a PBP program to change these definitions, if desired.

DEFINE may be used to change the predefined oscillator value, the DEBUG pins and baud rate and the LCD pin locations, among other things.

These definitions must be in all upper case. If not, the compiler may not recognize them. No error message will be produced for DEFINEs the compiler does not recognize.

```
DEFINE OSC 4      'Oscillator speed in MHz: 20
```

Math Operators

The PicBasic Pro Compiler performs all math operations in full hierarchal order. All math operations are unsigned and performed with 16-bit precision.

Table E-1: Pic Basic Pro Math Operators

Math Operators	Description
+	Addition
-	Subtraction
*	Multiplication
**	Top 16 Bits of Multiplication
*/	Middle 16 Bits of Multiplication
/	Division
//	Remainder (Modulus)
<<	Shift Left
>>	Shift Right
ABS	Absolute Value*
COS	Cosine
DCD	2n Decode
DIG	Digit
DIV 32	31-bit x 15-bit Divide
MAX	Maximum*
MIN	Minimum*
NCD	Encode
REV	Reverse Bits
SIN	Sine
SQR	Square Root
&	Bitwise AND
	Bitwise OR
^	Bitwise Exclusive OR
~	Bitwise NOT
&/	Bitwise NOT AND
	Bitwise NOT OR
^/	Bitwise NOT Exclusive OR

Comparison Operators

Comparison operators are used in **IF..THEN** statements to compare one expression with another. These comparisons are unsigned. They cannot be used to check if a number is less than 0.

Table E-2: Comparison Operators

Comparison Operator	Description
= or ==	Equal
<> or !=	Not Equal
<	Less Than
>	Greater Than
<=	Less Than or Equal
>=	Greater Than or Equal

Logical Operators

Logical operators differ from bitwise operations. They yield a true/false result from their operation. Values of 0 are treated as false. Any other value is treated as true. They are mostly used in conjunction with the comparison operators in an **IF..THEN** statement. The operators supported are:

Table E-3: Logical Operators

Logical Operator	Description
AND or &&	Logical AND
OR or	Logical OR
XOR or ^^	Logical Exclusive OR
NOT AND	Logical NAND
NOT OR	Logical NOR
NOT XOR	Logical NXOR

E.2) PicBasic Pro Statement Reference [35]

@	Insert one line of assembly language code.
ADCIN	Read on-chip analog to digital converter.
ASM..ENDASM	Insert assembly language code section.
CALL	Call assembly language subroutine.
CLEAR	Zero all variables.
COUNT	Count number of pulses on a pin.
DATA	Define initial contents of on-chip EEPROM.
DEBUG	Asynchronous serial output to fixed pin and baud.
DEBUGIN	Asynchronous serial input from fixed pin and baud.
EEPROM	Define initial contents of on-chip EEPROM.
END	Stop execution and enter low power mode.
FOR..NEXT	Repeatedly execute statements.
FREQOUT	Produce up to 2 frequencies on a pin.
GOSUB	Call BASIC subroutine at specified label.
GOTO	Continue execution at specified label.
HIGH	Make pin output high.
HPWM	Output hardware pulse width modulated pulse train.
HSERIN	Hardware asynchronous serial input.
HSEROUT	Hardware asynchronous serial output.
I2CREAD	Read bytes from I2C device.
I2CWRITE	Write bytes to I2C device.
IF..THEN..ELSE..ENDIF	Conditionally execute statements.
INPUT	Make pin an input.
LCDIN	Read from LCD RAM.
LCDOUT	Display characters on LCD.
LOW	Make pin output low.
OUTPUT	Make pin an output.
PAUSE	Delay (1mSec resolution).
PAUSEUS	Delay (1uSec resolution).
PULSIN	Measure pulse width on a pin.
PULSOUT	Generate pulse to a pin.
PWM	Output pulse width modulated pulse train to pin.
RCTIME	Measure pulse width on a pin.
READ	Read byte from on-chip EEPROM.
READCODE	Read word from code memory.
RETURN	Continue at statement following last GOSUB.
REVERSE	Make output pin an input or an input pin an output.
SELECT CASE	Compare a variable with different values.
SERIN	Asynchronous serial input (BS1 style).
SERIN2	Asynchronous serial input (BS2 style).
SEROUT	Asynchronous serial output (BS1 style).
SEROUT2	Asynchronous serial output (BS2 style).
SHIFTIN	Synchronous serial input.
SHIFTOUT	Synchronous serial output.
SLEEP	Power down processor for a period of time.
STOP	Stop program execution.
SWAP	Exchange the values of two variables.
TOGGLE	Make pin output and toggle state.
WHILE..WEND	Execute statements while condition is true.
WRITE	Write byte to on-chip EEPROM.

APPENDIX F: PIC Pin Variables and Functions

Table F.1: PIC Pin Variables and Functions

Pin Name	Function	Input/Output	Variable Name
RA0	Connected to RD pin of LM629, to control reading status or data	Output	RD
RA1	Connected to WR pin of LM629, to control writing command or data	Output	WR
RA2	Connected to PS pin of LM629, to control writing command or data	Output	PS
RA3	Connected to CS pin of MMC, to select the Card	Output	SS
RA4	Connected to the serial clock of I2C eeprom	Input	SCK
RA5	Connected to serial data of I2C eeprom	Input	SDA
RB0	Connected to HI output of LM629, to control whether the trajectory is completed or not	Input	HI
RB1	Connected to FTDI USB controller chip, to give clock to read data from FTDI, sent from PC	Output	RDUSB
RB2	Connected to FTDI USB controller chip, to give clock to write data to FTDI	Output	WRUSB
RB3	Connected to the clock input of d-flip flop that controls the PORTD latched output.	Output	DCLOCK
RB4	Connected to the RXF# output of FTDI, to control if there is data or not in the receive buffer of FTDI.	Input	DATAIN
RB5	Connected to the TXE# output of FTDI, to control if the transmit buffer of FTDI is full or not.	Input	DATAOUT
RB6-RB7	Connected to Download microcontroller on the DLP-IO26 card, to control firmware download	Input	
RC0-RC7	8-Bit data bus between LM629 and PIC	Input/Output	
RD0-RD7	8-Bit data bus between USB and PIC	Input/Output	
D0	Used for debugging events by blinking the led on the DLP card.	Output	INFORM
D1	Enable line and data bus driver ICs and LM629 of Y axis	Output	CSY
D2	Connected to RST pin of LM629, to reset LM629	Output	RST
D3	Connected to Direction pin of bi-direction data bus driver, to control the direction of data flow	Output	DIRECT
D5	Enable line and data bus driver ICs and LM629 of Z axis	Output	CSZ
D6	Enable line and data bus driver ICs and LM629 of X axis	Output	CSX
D7	Connected to break input of LMD18201, to break motors on power on stage	Output	BREAK
RE0	Connected to the Clock input of MMC	Output	CLK
RE1	Connected to the data output of MMC	Input	Dout
RE2	Connected to the data input of MMC	Output	Din

APPENDIX G: FIRMWARE

```

INCLUDE "modedefs.bas"
DEFINE OSC 20
ADCON1      = 7
' PIC PIN ASSIGNMENTS
rd           VAR PORTA.0      'read from LM629
wr           VAR PORTA.1      'write to LM629
ps           VAR PORTA.2      'port select 1=DATA,0=command
'I2C Eeprom Pins
scl          VAR   PORTA.4     'I2C SCL for ext eeprom
sda          VAR   PORTA.5     'I2C SDA for ext eeprom
' USB control pins
hi           VAR PORTB.0      ' lm629 interrupt control pin
rdusb        VAR PORTB.1      ' to give read clock to FTU chip
wrusb        VAR PORTB.2      ' to give write clock to FTU chip
datain       VAR PORTB.4      ' to check input buffer of usb chip
dclock       VAR PORTB.3      ' to give clock d flip-flop connected to portd
dataout      VAR PORTB.5      ' to check usb chip output buffer
dataout=1
wrusb=1
dclock=1
rdusb=1
'driver and buffer circuit control pins
inform       VAR PORTD.0      ' controls the led4 on dlp card
csy          VAR PORTD.1      ' chip select Y axis
rst          VAR PORTD.2      ' LM629 hardware reset
direct       VAR PORTD.3      ' controls the direction of data flow 0 LM-->PIC, 1 PIC-->LM
csz          VAR PORTD.5      ' chip select z axis
csx          VAR PORTD.6      ' chip select x axis
break       VAR PORTD.7      ' breaks the motor motion
'MMC Connections
SCK VAR PORTE.0      ' Clock pin MMC pin 5
SO  VAR PORTE.1      ' Data from MMC pin 7
SI  VAR PORTE.2      ' Data to MMC pin 2
SS  VAR PORTA.3      ' MMC Slave select pin 1
TRISA=%110000
TRISB.0= 1
TRISB.1= 0
TRISB.2= 0
TRISB.3= 0
TRISB.4= 1
TRISB.5= 1
TRISC=255
TRISD=0
PORTD=255
dclock=0
dclock=1
TRISE = 5

```

'I2C Variables

ctl	VAR BYTE	'EEPROM control code
edata	VAR BYTE	'Data byte to be written
rd_addr	VAR WORD	
wr_addr	VAR WORD	

' General MMC Variables used

mmc_status	VAR BYTE	
mmc_status=0		
addr	VAR WORD	
addr_H	VAR WORD	' MMC Address is a 32 bit address, addr_h is bits 31-16
addr_L	VAR WORD	' and addr_l is bits 0-15
rdaddr_H	VAR WORD	' read MMC Address is a 32 bit address, addr_h is bits 31-16
rdaddr_L	VAR WORD	
arg0	VAR BYTE	' args used
arg1	VAR BYTE	
arg2	VAR BYTE	
arg3	VAR BYTE	
arg4	VAR BYTE	
cmd	VAR BYTE	'command var
crc	VAR BYTE	'crc var
dat	VAR BYTE	'data var
y	VAR BYTE	'counter var
c	VAR BYTE	
idx	VAR BYTE	
res	VAR BYTE	'response var
reswr	VAR BYTE	'response w/r
res2	VAR WORD	'response var
hex_nibble	VAR BYTE	
erase_flag	VAR BYTE	
erase_flag = 0		
limitaddr	VAR WORD	
last_addr	VAR WORD	'index of sector and last data
load_ee	VAR WORD	'number byte that will send
load_ee0	VAR load_ee.byte0	
load_ee1	VAR load_ee.byte1	
rest_data	VAR BYTE	
in_dat	VAR BYTE	'Serial vars
chk	VAR BYTE	
cntr	VAR BYTE	
in_array	VAR BYTE[64]	'64 byte in coming array
sector0	VAR WORD	'first 16-bit number of memory sectors used
sector00	VAR sector0.byte0	
sector01	VAR sector0.byte1	
sector1	VAR WORD	'second 16-bit number of memory sectors used
sector10	VAR sector1.byte0	
sector11	VAR sector1.byte1	
carry_flag	VAR status.0	
zero_flag	VAR status.2	
addr=\$0000		'Used for eeprom etc
read_cntrl	VAR BYTE	
read_cntrl=0		
j	VAR BYTE	' reset control variable

readusb	VAR BYTE	' number of byte that will be read from usb
setd	VAR BYTE	
stop_flag	VAR BYTE	
bcount	VAR BYTE	'blink count
bcount=1		
command	VAR BYTE	
stat	VAR BYTE	
frusb	VAR BYTE	
frstop	VAR BYTE	
frstop=0		
'motor position variables		
rdrpw	VAR WORD	'current motor position read from LM629
rdrp1	VAR RDRPW.BYTE0	
rdrp2	VAR RDRPW.BYTE1	
rdrp3	VAR BYTE	
Xcurr	VAR WORD	' keeps the current position of x
Ycurr	VAR WORD	
Zcurr	VAR WORD	
Xcurr = 0		
Ycurr = 0		
Zcurr = 0		
hxread1	VAR BYTE	' high byte of next positions that motor will go
hxread0	VAR BYTE	
Xread	VAR WORD	' low byte of next positions that motor will go
Xread1	VAR xread.BYTE1	
Xread0	VAR xread.BYTE0	
hyread1	VAR BYTE	
hyread0	VAR BYTE	
yread	VAR WORD	
yread1	VAR yread.BYTE1	
yread0	VAR yread.BYTE0	
hzread1	VAR BYTE	
hzread0	VAR BYTE	
zread	VAR WORD	
zread1	VAR zread.BYTE1	
zread0	VAR zread.BYTE0	
posx	VAR WORD	' absolute distances
posx1	VAR posx.BYTE1	
posx0	VAR posx.BYTE0	
hposx	VAR BYTE	
hposy	VAR BYTE	
posy	VAR WORD	
posy1	VAR posy.BYTE1	
posy0	VAR posy.BYTE0	
posz	VAR WORD	
posz1	VAR posz.BYTE1	
posz0	VAR posz.BYTE0	
hposz	VAR BYTE	
signx	VAR BYTE	' sign of axis
signxcurr	VAR BYTE	
signxcurr=0		
signy	VAR BYTE	
signycurr	VAR BYTE	

```

signycurr=0
signz          VAR BYTE
signzcurr      VAR BYTE
signzcurr=0
'velocity and accelaration variables
hv             VAR WORD
hv0            VAR hv.BYTE0
hv1            VAR hv.BYTE1
lvx            VAR WORD
lvx1           VAR lvx.BYTE1
lvx0           VAR lvx.BYTE0
hvx            VAR WORD
hvx1           VAR hvx.BYTE1
hvx0           VAR hvx.BYTE0
hvy            VAR WORD
hvy1           VAR hvy.BYTE1
hvy0           VAR hvy.BYTE0
lvy            VAR WORD
lvy1           VAR lvy.BYTE1
lvy0           VAR lvy.BYTE0
lvz            VAR WORD
lvz1           VAR lvz.BYTE1
lvz0           VAR lvz.BYTE0
hvz            VAR WORD
hvz1           VAR hvz.BYTE1
hvz0           VAR hvz.BYTE0
cv             VAR WORD
cv0            VAR cv.BYTE0
cv1            VAR cv.BYTE1
acc            VAR WORD
acc0           VAR acc.BYTE0
acc1           VAR acc.BYTE1
Yacc0          VAR accY.BYTE0
Yacc1          VAR accY.BYTE1
Xacc0          VAR accX.BYTE0
Xacc1          VAR accX.BYTE1
accY           VAR WORD
accX           VAR WORD
Zacc0          VAR accz.BYTE0
Zacc1          VAR accz.BYTE1
accz           VAR WORD
acc=1
cv=7
i              VAR BYTE
x              VAR BYTE
usb_loop       VAR BYTE
rest           VAR BYTE
X=2
j=1
c=0
sector0=0
sector1=0
addr_H = $0000
'acceleration constant
'speed constant
'counter var
'control var

```

```

addr_L = $0200
rdaddr_H = $0000
rdaddr_L = $0000
rd_addr=0
limitaddr=504
*****MMC PART OF THE FIRMWARE*****
'////////////////////////////////////
'// 11-MMC INITIALIZATION in SPI mode //
'// 12-READ THE PREVIOUSLY SAVED DATA and Index //
'// 14-WRITE MMC //
'// 13-goto LM629 //
'////////////////////////////////////
MMC:
frstop=0
readusb=1
wr_addr=0
last_addr=65500
ctl=$A0
setd=0
IF datain=1 Then MMC 'check ftu inpu buffer if there is data or not
GoSub USBREAD
*****
***** MMC INITIALIZATION in SPI mode *****
*****
IF in_array[0]=$11 Then
ss= 1
dat=$ff
cntr= 10
GoSub shiftout_dat
ss= 0
Pause 50
cmd= $40 'MMC Command 0
crc= $95
dat= cmd
ShiftOut SI, SCK, MSBFIRST,[dat]
dat= 0
cntr= 4
GoSub shiftout_dat
dat= crc
ShiftOut SI, SCK, MSBFIRST,[dat]
chk= 1
GoSub chk_res
ss= 1
Pause 50
ss= 0:
While res <> 0
ss= 1
dat=$ff
ShiftOut SI, SCK, MSBFIRST,[dat]
GoSub shiftin_res
ss= 0
dat= $41 'MMC Command 1
ShiftOut SI, SCK, MSBFIRST,[dat]

```

```

dat = 0
cntr = 4
GoSub shiftout_dat
dat = $ff
GoSub shiftout2
GoSub shiftin_res
Wend
dat = $ff

'MMC now successfully initialized in SPI mode

addr_H = $0000'mset_blok length:
addr_L = $0200
cmd = $50      'Cmd16
crc = $ff
GoSub send_cmd
bcount = 2
GoSub BLINK
GoTo MMC
Else
'*****
'*****  Read MMC and index  *****
'*****
IF in_array[0]=$12 Then
I2CRead sda,scl,ctl,last_addr,[sector11,sector10,sector01,sector00,load_ee1,load_ee0]
TRISD=0
PORTD=sector11
wrusb=0
wrusb=1
PORTD=sector10
wrusb=0
wrusb=1
PORTD=sector01
wrusb=0
wrusb=1
PORTD=sector00
wrusb=0
wrusb=1
PORTD=load_ee1
wrusb=0
wrusb=1
PORTD=load_ee0
wrusb=0
wrusb=1
READMMC:
ctl=$A2
ss = 1
arg4 = $ff
GoSub shiftout1
GoSub shiftin_res
ss = 0
arg0 = $51
arg1 = addr_H.BYTE1
arg2 = addr_H.BYTE0
arg3 = addr_L.BYTE1

```



```

arg4 = addr_L.BYTE0
GoSub shiftout5
arg4 = $ff
GoSub shiftout1
GoSub shiftin_res
chk = 0
GoSub chk_res
chk = $FE
GoSub chk_res

' Read cmd successful
' Read the MMC data and save in eeprom
'read 512 bytes can be any number

For wr_addr = 0 TO 511
ShiftIn SO, SCK, MSBPRES, [dat]
'We now have a byte do something with it!
I2CWrite sda,scl,ctl,wr_addr,[dat]
Pause 10'Required after a MMRead
Next wr_addr

GoSub shiftin_res
GoSub shiftin_res
GoSub mstatus
'Get read status
rd_addr=0
GoSub blink
IF read_cntrl=1 Then
read_cntrl=0
GoTo loop3
ENDIF
Else
*****
*****  MMC WRITE  *****
*****
IF in_array[0]=$14 Then
readusb=2
GoSub USBREAD
load_ee0=in_array[1]
load_ee1=in_array[0]
idx=load_ee/64
readusb=64
rest_data=load_ee//64

' how many byte will be loaded to eeprom

IF idx=0 Then
idx=1
readusb=load_ee
rest=0
Else
rest=1
ENDIF
i = 0
While i < idx
i=i+1
GoSub USBREAD
I2CWrite sda,scl,ctl,wr_addr,[STR in_array\64]
Pause 10

```

```

wr_addr = wr_addr + 64
Wend
IF rest = 1 AND rest_data <> 0 Then
readusb=rest_data
GoSub USBREAD
I2CWrite sda,scl,ctl,wr_addr,[STR in_array\64]
Pause 10
wr_addr = wr_addr + rest_data
EndIF
PORTD=wr_addr.byte0
wrusb=0
wrusb=1
'Write into I2C EEPROM then write to MMC
IF c=0 Then
addr_H = $0000
addr_L = $0200
c=c+1
EndIF
ss = 1
arg4 = $ff
GoSub shiftout1
GoSub shiftin_res
ss = 0
arg0 = $58
arg1 = addr_H.BYTE1
arg2 = addr_H.BYTE0
arg3 = addr_L.BYTE1
arg4 = addr_L.BYTE0
GoSub shiftout5
arg4 = $ff
GoSub shiftout1
GoSub shiftin_res
chk = 0
GoSub chk_res

'Write command was a success
'File token tells the MMC data is to follow

arg4 = $FE
GoSub shiftout1
'MMC now ready to write data in blocks of 512 bytes assumes eeprom address in addr
For addr = 0 TO 511
I2CRead sda,scl,ctl,addr,[dat] 'Read from I2C ext eeprom
ShiftOut SI, SCK, MSBFIRST,[dat]
Next addr
dat=$ff
cntr = 2
GoSub shiftout_dat
GoSub shiftin_res
y=res & $0f
GoSub shiftin_res
While res = 0 'MMC is busy writing until response is not = 0
ShiftIn SO, SCK, MSBPRES, [res]
Wend
GoSub mstatus 'Get write status

```

```

I2CWrite
sda,scl,ctl,last_addr,[addr_H.byte1,addr_H.byte0,addr_L.byte1,addr_L.byte0,load_ee1,load_ee0]
Pause 10
addr_L = addr_L + $0200
IF carry_flag = 1 Then
addr_H = addr_H + 1
EndIF
GoTo MMC
Else
'*****
'*****  GOTO LM629 PART  *****
'*****
IF in_array[0]=$13 Then
GoTo loop2
Else
GoTo MMC
EndIF
EndIF
EndIF
EndIF
'*****LM629 PART OF THE FIRMWARE*****
'////////////////////////////////////
'// 21-RESET LM629 //
'// 22-STOP MODULE //
'// 23-LOAD POSITIONS MANUALLY //
'// 24-UPDATE PID PARAMETERS //
'// 25-UPDATE VELOCITY AND ACCELARATION CONSTANTS //
'// 26-START READING MMC AND TRAJECTORY //
'// 27-GOTO STOP POINT //
'// 28-GOTO MMC PART //
'// 29-MOVE Z AXES //
'////////////////////////////////////
loop2:
IF setd=0 Then
TRISD=0
PORTD=255
GoSub CLOCK
setd=1
EndIF
LM629:
readusb=1
IF DATAIN=1 Then LM629
LOOP3:
'*****Trajectory complete module*****
IF datain=0 Then INIT ' check if a control command is sent or not
Pause 1
IF HI= 0 Then loop3 ' check the trajectory is complete or not
IF DATAIN=0 Then
INIT:
readusb=1
GoSub USBREAD

```

```

'*****
'****      RESET LM629      *****
'*****

IF in_array[0]=$21 Then
addr_H = $0000
addr_L = $0200
j=1
ctl= $A2
Xcurr = 0
Ycurr = 0
Zcurr = 0
signxcurr=0
signycurr=0
signzcurr=0
rd_addr=0
IF j=1 Then
CSX=0
CSY=1
CSZ=1
GoSub CLOCK
j= 2
Else
RESETY:
IF j=2 Then
j=3
CSX=1
CSY=0
CSZ=1
GoSub CLOCK
Else
RESETZ:
j=1
CSX=1
CSY=1
CSZ=0
GoSub CLOCK
EndIF
EndIF
GoSub HWRESET
X=1
GoSub CHECK
IF stat =196 OR stat =132 Then
GoTo GORST
Else
GoSub ERROR
EndIF
GORST:
command=29
GoSub WRCOMM
in_array[0] = 0
in_array[1] = 0
GoSub WWRD
X=1

```

'RSTI

```

GoSub CHECK
IF stat = 192 OR stat= 128 Then
command = 28                                'MSKI
GoSub WRCOMM
in_array[0]=0
in_array[1]=4
GoSub WWRD
Else
GoSub ERROR
EndIF
IF j=2 Then RESETY
IF j=3 Then RESETZ
bcount=3
GoSub BLINK
' Load PID parameters to all lm629/
CSY = 0
CSX = 0
CSZ = 0
GoSub CLOCK
command = 30                                'LFIL
GoSub WRCOMM
in_array[0]=0
in_array[1]=15                             ' set all PID parameters bit which indicate all of them will be loaded
GoSub WWRD
in_array[0]=1                               'KP var
in_array[1]=0
GoSub WWRD
in_array[0]=0                               'KI var
in_array[1]=0
GoSub WWRD
in_array[0]=0                               'KD var
in_array[1]=0
GoSub WWRD
in_array[0]=0                               'IL var
in_array[1]=0
GoSub WWRD
COMMAND= 4                                  'UDF
GoSub WRCOMM
GoTo LM629
Else
*****
*****          STOP MODULE          *****
*****
IF in_array[0]=$22 Then
CSX=0
CSY=0
CSZ=0
GoSub CLOCK
COMMAND= 31                                'LTRJ
GoSub WRCOMM
in_array[0]=4                               'set smoothly stop bit of trajectory complete word
in_array[1]=0
GoSub WWRD

```

```

COMMAND= 1          'STT
GoSub WRCOMM
CHK:
Pause 1
IF HI= 0 Then CHK

                                'read positions

CSX=0
CSY=1
CSZ=1
GoSub CLOCK
GoSub RDRP
IF RDRP3=255 Then          ' check the position of motors if it is on negative axis
signxcurr=1
Xcurr =65535-RDRPW
Else
signxcurr=0
Xcurr = RDRPW
EndIF
CSX=1
CSY=0
CSZ=1
GoSub CLOCK
GoSub RDRP
IF RDRP3=255 Then
signycurr=1
ycurr=65535-rdrpw
Else
signycurr=0
Ycurr= RDRPW
EndIF
CSX=1
CSY=1
CSZ=0
GoSub CLOCK
GoSub RDRP
IF RDRP3=255 Then
signzcurr=1
zcurr=65535- RDRPW
Else
signzcurr=0
zcurr= RDRPW
EndIF
CSX=0
CSY=0
CSZ=0
GoSub CLOCK
frstop=1                ' set stop flag
rd_addr=rd_addr-9       ' take read address of I2C eeprom back
GoTo LM629
Else

```

```

*****
*****  LOAD POSITION MANUALLY  *****
*****

IF in_array[0]=$23 Then
frusb=1
GoSub BLINK
readusb=9
GoSub USBREAD
signx= in_array[0]           ' read 9 byte position
xread1=in_array[1]
xread0=in_array[2]
signy= in_array[3]
yread1=in_array[4]
yread0=in_array[5]
signz= in_array[6]
zread1=in_array[7]
zread0=in_array[8]
GoTo LTRJ2
Else
*****
*****  UPDATE PID  *****
*****

IF in_array[0]=$24 Then
GoSub USBREAD
IF in_array[0]=$0 Then           'enable all
CSY= 0
CSX= 0
CSZ= 0
Else
IF in_array[0]=$1 Then           'enable x
CSY= 1
CSX= 0
CSZ= 1
Else
IF in_array[0]=$2 Then           'enable y
CSY= 0
CSX= 1
CSZ= 1
Else
IF in_array[0]=$3 Then           'enable z
CSY= 1
CSX= 1
CSZ= 0
ENDIF
ENDIF
ENDIF
ENDIF
GoSub CLOCK
command = 30
GoSub WRCOMM                     'LFIL
in_array[0]=0
in_array[1]=15
GoSub WWRD

```

```

PauseUs 150
readusb=2
GoSub USBREAD
GoSub WWRD                                'KP
GoSub USBREAD
GoSub WWRD                                'KI
GoSub USBREAD
GoSub WWRD                                'KD
GoSub USBREAD
GoSub WWRD                                'IL
GoSub USBREAD
GoSub WRCOMM                              'UDF
GoSub BLINK
GoTo LM629
Else
*****
***** UPDATE VELOCITY AND ACC *****
*****
IF in_array[0]=$25 Then
readusb=2
GoSub USBREAD
cv1=in_array[0]
cv0=in_array[1]
GoSub USBREAD
acc1=in_array[0]
acc0=in_array[1]
GoSub BLINK
GoTo LM629
Else
*****
***** START FROM MMC *****
*****
IF in_array[0]=$26 Then
frusb=0
GoTo LTRJ
Else
*****
***** GO TO MMC part *****
*****
IF in_array[0]=$28 Then
GoSub BLINK
GoTo MMC
Else
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
ENDIF
GoTo LM629
ENDIF

```



```

***** DATA PROCESSING PART *****
'////////////////////////////////////
'// READ POSITIONS FROM I2C EPROM //
'// LOAD TRAJECTORY PARAMETERS TO LM629 //
'////////////////////////////////////
LTRJ:
I2CRead sda,scl,ctl,rd_addr,[signx,xread1,xread0,signy,yread1,yread0,signz,zread1,zread0]
rd_addr=rd_addr + 9
LTRJ2:
GoSub POSITION
IF signx = 0 Then
hxread1=0
hxread0=0
Else
hxread1=255
hxread0=255
xread=65535-xread
EndIF

IF signy=0 Then
hyread1=0
hyread0=0
Else
hyread1=255
hyread0=255
yread=65535-yread
EndIF

IF signz = 0 Then
hzread1=0
hzread0=0
Else
hzread1=255
hzread0=255
zread=65535-zread
EndIF

'calculation of velocity and accelaration according to absolute distance
lvx=cv*posx
hvx=(cv**posx) + (hposx*cv)
accx=acc*posx
lvy=cv*posy
hvy=(cv**posy) + (hposy*cv)
accy=acc*posy
lvz=cv*posz
hvz=(cv**posz) + (hposz*cv)
accz=acc*posz
CSX=0 ' load x servo controller
CSY=1
CSZ=1
GoSub CLOCK
command = 31
GoSub WRCOMM
in_array[0] = 0

```

```

in_array[1]= 42
GoSub WWRD          ' ACCELERATION
in_array[0]=0
in_array[1]=0
GoSub WWRD
in_array[0]=Xacc1
in_array[1]=Xacc0
GoSub WWRD          ' VELOCITY
in_array[0]=hvX1
in_array[1]=hvX0
GoSub WWRD
in_array[0]=lvx1
in_array[1]=lvx0
GoSub WWRD          ' POSITION
in_array[0]=Hxread1
in_array[1]=Hxread0
GoSub WWRD
in_array[0]=xread1
in_array[1]=xread0
GoSub WWRD
CSX=1
CSY=0               ' load y servo controller
CSZ=1
GoSub CLOCK
command = 31
GoSub WRCOMM
in_array[0] = 0
in_array[1] = 42
GoSub WWRD
in_array[0]=0
in_array[1]=0
GoSub WWRD          ' ACCELERATION
in_array[0]=Yacc1
in_array[1]=Yacc0
GoSub WWRD
in_array[0]=hvY1
in_array[1]=hvY0
GoSub WWRD          ' VELOCITY
in_array[0]=lvy1
in_array[1]=lvy0
GoSub WWRD
in_array[0]=Hyread1
in_array[1]=Hyread0
GoSub WWRD          ' POSITION
in_array[0]=yread1
in_array[1]=yread0
GoSub WWRD
movez:
CSX=1
CSY=1
CSZ=0               ' load z servo controller
GoSub CLOCK
command = 31

```

```

GoSub WRCOMM
in_array[0]= 0
in_array[1]= 42
GoSub WWRD
in_array[0]=0
in_array[1]=0
GoSub WWRD          ' ACCELERATION
in_array[0]=Zacc1
in_array[1]=Zacc0
GoSub WWRD
in_array[0]=hvZ1
in_array[1]=hvZ0
GoSub WWRD          ' VELOCITY
in_array[0]=lvZ1
in_array[1]=lvZ0
GoSub WWRD
in_array[0]=Hzread1
in_array[1]=Hzread0
GoSub WWRD          ' POSITION
in_array[0]=zread1
in_array[1]=zread0
GoSub WWRD
BREAK = 0
' give start command to all servo controllers to provide synchronization
CSX=0
CSY=0
CSZ=0
GoSub CLOCK
command= 1
PauseUs 10
GoSub WRCOMM          'STT
command=29
PauseUs 10
GoSub WRCOMM          'RSTI
in_array[0] = 0
in_array[1] = 0
GoSub WWRD
PauseUs 10
command=28
PauseUs 10
GoSub WRCOMM          'MSKI
in_array[0] = 0
in_array[1] = 4
GoSub WWRD
IF frusb=1 Then
frusb=0
GoTo lm629
EndIF
' check the updated address
IF addr_H >= sector1 AND addr_L >= sector0 Then
limitaddr= load_ee
IF rd_addr >= limitaddr Then
PORTD=$cc

```

```

wrusb=0
wrusb=1
GoTo MMC
EndIF
Else
IF rd_addr >= limitaddr Then
addr_L = addr_L + $0200
IF carry_flag = 1 Then
addr_H = addr_H + 1
EndIF
rd_addr=0
read_cntrl=1
GoTo READMMC
EndIF
EndIF
IF frstop <> 0 Then
frstop=0
GoTo LM629
EndIF
GoTo loop3
'*****
'*****  SUBPROCEDURES  *****
'*****

USBREAD :
TRISD= 255
For usb_loop=1 TO readusb
wfd:
IF datain = 1 Then wfd  ' wait for data
rdusb=0
rdusb=1
in_array[usb_loop-1] = PORTD
PauseUs 3
Next
TRISD=0
Return
' ***** MMC subroutines *****

'Send a command to the mmc assumes CMD, addr_H and addr_L

SEND_CMD:
ss = 1
arg4 = $ff
GoSub shiftout1
GoSub shiftin_res
ss = 0
arg0 = cmd
arg1 = addr_H.BYTE1
arg2 = addr_H.BYTE0
arg3 = addr_L.BYTE1
arg4 = addr_L.BYTE0
GoSub shiftout5
Dat = $FF
cntr = 2
GoSub shiftout_dat
GoSub shiftin_res

```

```

IF erase_flag = 1 Then Return
chk = 0
GoSub chk_res
Return
shiftout_dat:
For y = 1 TO cntr
ShiftOut SI, SCK, MSBFIRST,[dat]
Next y
Return

'Shift out 5 (ARG0-ARG4)
shiftout5:
ShiftOut SI, SCK, MSBFIRST,[arg0]
shiftout4:
ShiftOut SI, SCK, MSBFIRST,[arg1]
shiftout3:
ShiftOut SI, SCK, MSBFIRST,[arg2]
shiftout2:
ShiftOut SI, SCK, MSBFIRST,[arg3]
shiftout1:
ShiftOut SI, SCK, MSBFIRST,[arg4]
Return

shiftin_res:
ShiftIn SO, SCK, MSBPRESS, [res]
Return

chk_res:
While res <> chk
ShiftIn SO, SCK, MSBPRESS, [res]
Wend
Return
'MMC status

mstatus:
ss = 1
dat = $ff
ShiftOut SI, SCK, MSBFIRST,[dat]
GoSub shiftin_res

ss = 0
arg4 = $4D
GoSub shiftout1
dat = 0
cntr = 4
GoSub shiftout_dat
arg4 = $ff
GoSub shiftout1
ShiftIn SO, SCK, MSBPRESS, [res2\16]
mmc_status = res2
Return

```

' ***** LM629 subroutines *****

CHECK:

```
TRISC = 255
direct = 0
dclock=0
dclock=1
PauseUs 300
ps = 0
rd = 0
PauseUs 3
stat = PORTC
rd =1
ps =1
IF x=1 Then
x=2
Return
EndIF
IF stat.0 = 1 Then CHECK
Return
```

HWRESET :

```
direct = 0
rst = 0
GoSub CLOCK
rst = 1
dclock=0
dclock=1
PauseUs 1500
Return
```

WRCOMM:

```
TRISC = 0
direct = 1
GoSub CLOCK
ps = 0
wr = 0
PORTC =command
PauseUs 25
wr = 1
ps = 1
PauseUs 15
GoSub CHECK
Return
```

WWRD :

```
direct =1
GoSub CLOCK
ps=0
TRISC = 0
PauseUs 5
ps = 1
PORTC = in_array[0]
wr = 0
```

```

PauseUs 5
wr= 1
ps= 0
PauseUs 3
ps=1
PORTC = in_array[1]
wr =0
PauseUs 5
wr= 1
ps= 0
PauseUs 15
GoSub CHECK
Return

```

POSITION:

```

IF signx=0 AND signxcurr=0 Then
hposx=0
IF xcurr > xread Then
posx = xcurr - xread
Else
IF Xcurr < xread Then
posx = xread - xcurr
Else
posx=1
xread=xread+1
EndIF
EndIF
EndIF

```

```

IF signx=1 AND signxcurr=1 Then
hposx=0
IF xcurr > xread Then
posx = xcurr - xread
Else
IF Xcurr < xread Then
posx = xread - Xcurr
Else
posx=1
xread=xread+1
EndIF
EndIF
EndIF

```

```

IF signx <> signxcurr Then
posx=xread+xcurr
IF Xcurr < posx Then
hposx =0
Else
hposx=1
EndIF
EndIF

```

```

IF signy=0 AND signycurr=0 Then

```

```

hposy=0
IF ycurr > yread Then
posy = ycurr - yread
Else
IF Ycurr < yread Then
posy = yread - ycurr
Else
posy=1
yread=yread+1
EndIF
EndIF
EndIF

```

```

IF signy=1 AND signycurr=1 Then
hposy=0
IF ycurr > yread Then
posy = ycurr - yread
Else
IF Ycurr < yread Then
posy = yread - Ycurr
Else
posy=1
yread=yread+1
EndIF
EndIF
EndIF

```

```

IF signy <> signycurr Then
posy=yread+Ycurr
IF ycurr < posy Then
hposy =0
Else
hposy=1
EndIF
EndIF

```

```

IF signz=0 AND signzcurr=0 Then
hposz=0
IF zcurr > zread Then
posz = zcurr - zread
Else
IF zcurr < zread Then
posz = zread - zcurr
Else
posz=1
zread=zread+1
EndIF
EndIF
EndIF

```

```

IF signz=1 AND signzcurr=1 Then
hposz=0
IF zcurr > zread Then

```



```

posz = zcurr - zread
Else
IF zcurr < zread Then
posz = zread - zcurr
Else
posz=1
zread=zread+1
EndIF
EndIF
EndIF

IF signz <> signzcurr Then
posz=zread+zcurr
IF zcurr < posz Then
hposz =0
Else
hposz=1
EndIF
EndIF

signxcurr=signx
signycurr=signy
signzcurr=signz
Xcurr=xread
Ycurr=yread
zcurr=zread
Return

```

ERROR:

```

IF DATAIN=1 Then
inform=0
GoSub CLOCK
Pause 220
inform=1
GoSub CLOCK
Pause 220
GoTo ERROR
Else
GoTo INIT
EndIF
GoTo ERROR
Return

```

BLINK:

```

For i=1 TO bcount
inform=0
GoSub CLOCK
Pause 480
inform=1
GoSub CLOCK
Pause 480
Next i

```

```
bcount=1  
Return
```

RDRP:

```
command = 10  
GoSub WRCOMM  
direct = 0  
GoSub CLOCK  
TRISC = 255  
PauseUs 10  
ps = 0  
PauseUs 3  
ps = 1  
rd = 0  
PauseUs 15  
'RDRP4 = PORTC  
rd = 1  
ps = 0  
PauseUs 10  
ps = 1  
rd = 0  
PauseUs 25  
RDRP3 = PORTC  
rd = 1  
ps = 0  
PauseUs 5  
ps = 1  
rd = 0  
PauseUs 25  
RDRP2 = PORTC  
rd = 1  
ps = 0  
PauseUs 5  
ps = 1  
rd = 0  
PauseUs 25  
RDRP1 = PORTC  
rd = 1  
ps = 0  
Return
```

CLOCK:

```
dclock=0  
dclock=1  
PauseUs 100  
Return  
End
```

APPENDIX H: SPI Command Set

Table H-1: Command Classes in SPI Mode

	Class	Supported Commands																											
CCC	Descrip.	0	1	9	10	12	13	16	17	18	23	24	25	27	28	29	30	35	36	38	42	55	56	58	59				
0	Basic	+	+	+	+		+																		+	+			
1	NS																												
2	Block read					+		+	+	+	+																		
3	NS																												
4	Block write							+			+	+	+	+															
5	Erase																	+	+	+									
6	Write-protect															+	+	+											
7	Lock card																				+								
8	App-specific																					+	+						
9	NS																												
10-11	R																												

Key: CCC = Card CMD Class
 NS = Not supported in SPI mode.
 R = Reserved

Command Description

Table H-2 provides a detailed description of the SPI Mode commands. . A “yes” in the SPI Mode column indicates that the command is supported in SPI Mode.

With these restrictions, the command class description in the CSD is still valid. If a command does not require an argument, the value of this field should be set to zero. Reserved SPI commands are also reserved in MultiMediaCard mode. The binary code of a command is defined by the mnemonic symbol. As an example, the content of the CMD Index field for CMD0 is (binary) “000000” and for CMD39 is (binary) “100111”.

Table H-2: SPI Bus command description

CMD Index	SPI Mode	Argument	Resp	Abbreviation	Description
CMD0	Yes	None	R1	GO_IDLE_STATE	Resets MultiMediaCard or RS-MultiMediaCard.
CMD1	Yes	None	R1	SEND_OP_COND	Activates the card's initialization process.
CMD2	No	---	---	---	---
CMD3	No	---	---	---	---
CMD4	No	---	---	---	---
CMD5	Reserved				
CMD6	Reserved				
CMD7	No	---	---	---	---
CMD8	Reserved				
CMD9	Yes	None	R1	SEND_CSD	Asks the selected card to send its specific data (CSD).
CMD10	Yes	None	R1	SEND_CID	Asks the selected card to send its identification (CID).
CMD11	No	---	---	---	---
CMD12	Yes	None	R1	STOP_TRANSMISSION	Forces card to stop transmission during a multiple block read operation.
CMD13	Yes	None	R2	SEND_STATUS	Asks the selected card to send its status register.
CMD14	Reserved				
CMD15	No	---	---	---	---
CMD16	Yes	[31:0] block length	R1	SET_BLOCKLEN	Selects a block length (in bytes) for all following block commands (read & write).
CMD17	Yes	[31:0] data address	R1	READ_SINGLE_BLOCK	Reads a block of the size selected by the SET_BLOCKLEN command.
CMD18	Yes	[31:0] data address	R1	READ_MULTIPLE_BLOCK	Continuously transfers data blocks from card to host until interrupted by a STOP_TRANSMISSION command or the requested number of data blocks transmitted.

Table H-2: SPI Bus command description (continued)

CMD Index	SPI Mode	Argument	Resp	Abbreviation	Description
CMD19	Reserved				
CMD20	No	---	---	---	---
CMD21 ... CMD23	Reserved				
CMD24	Yes	[31:0] data address	R1	WRITE_BLOCK	Writes a block of the size selected by the SET_BLOCKLEN command.
CMD25	Yes	[31:0] data address	R1	WRITE_MULTIPLE_BLOCK	Continuously writes blocks of data until a stop transmission token is sent or the requested number of blocks is received.
CMD26	No	---	---	---	---
CMD27	Yes	None	R1	PROGRAM_CSD	Programming of the programmable bits of the CSD.
CMD28	Yes	[31:0] data address	R1b	SET_WRITE_PROT	If the card has write protection features, this command sets the write protection bit of the addressed group. Write protection properties are coded in the card-specific data (WP_GRP_SIZE).
CMD29	Yes	[31:0] data address	R1b	CLR_WRITE_PROT	If the card has write protection features, this command clears the write protection bit of the addressed group.
CMD30	Yes	[31:0] write protect data address	R1	SEND_WRITE_PROT	If the card has write protection features, this command asks the card to send the status of the write protection bits.
CMD31	Reserved				
CMD32	Yes	[31:0] data address	R1	TAG_SECTOR_START	Sets the address of the first sector of the erase group.

Table H-2: SPI Bus command description (continued)

CMD Index	SPI Mode	Argument	Resp	Abbreviation	Description
CMD33	Yes	[31:0] data address	R1	TAG_SECTOR_END	Sets the address of the last sector in a continuous range within the selected erase group, or the address of a single sector to be selected for erase.
CMD34	Yes	[31:0] data address	R1	UNTAG_SECTOR	Removes one previously selected sector from the erase selection.
CMD35	Yes	[31:0] data address	R1	TAG_ERASE_GROUP_START	Sets the address of the first erase group within a range to be selected for erase.
CMD36	Yes	[31:0] data address	R1	TAG_ERASE_GROUP_END	Sets the address of the last erase group within a continuous range to be selected for erase.
CMD37	Yes	[31:0] data address	R1	UNTAG_ERASE_GROUP	Removes one previously selected erase group from the erase selection.
CMD38	Yes	[31:0] stuff bits	R1b	ERASE	Erases all previously selected sectors.
CMD39	No	---	---	---	---
CMD40	No	---	---	---	---
CMD41	Reserved				
CMD42	Yes	[31:0] stuff bits	R1b	LOCK_UNLOCK	Set/resets the password or lock/unlock the card. The size of the Data Block is defined by the SET_BLOCK_LEN command.
CMD43 ... CMD54	Reserved				
CMD55	Yes	This optional MMCA command is not supported in the SanDisk MultiMediaCard/RS-MultiMediaCard.			
CMD56	Yes	This optional MMCA command is not supported in the SanDisk MultiMediaCard/RS-MultiMediaCard.			
CMD57	Reserved				
CMD58	Yes	None	R3	READ_OCR	Reads the OCR Register of a card.
CMD59	Yes	[31:1] stuff bits, [0:0] CRC option	R1	CRC_ON_OFF	Turns the CRC option on or off. A "1" in the CRC option bit will turn the option on; a "0" will turn it off.
CMD60- CMD63	No	---	---	---	---