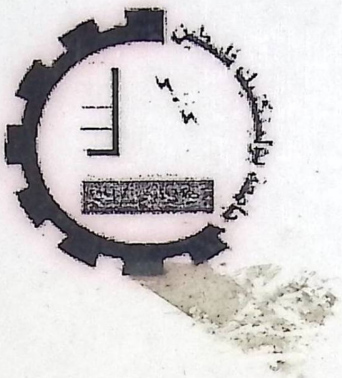
Palestine Polytechnic University



College of Engineering & Technology
Electrical & Computer Department

Graduation Project

Stand Alone Audio CD Player

Project Team
Ruba Sultan
Ruba Al-Hirbawe
Manar Al-Hammoury

Project Supervisor
Eng. Elyan Abu Gharbia

Hebron-Palestine

June 2005

ABSTRACT

Stand Alone CD Player

Project Team:

Manar AL-Hammoury
Ruba Al-Herbawi
Ruba Sultan

Palestine Polytechnic University - 2005

Supervisor:
Eng. Elayan Abu-Gharbyeh

Most CD-ROM drivers come with convenient features to allow users to play and listen to audio CD's, but staying along with the computer to listen.

For portability needs and the ability of using an old and damaged computer CD driver, instead of throwing it away, this project will be constructed to allow users playing music without computer and controlling the running track also to display tracks information on LCD and may play stored tracks without CD.

A CD-drive will be interfaced to LCD and switches through a PIC microcontroller I/O ports. The switches will provide the ability of controlling the drive operation such as open/close door, next track, play/stop, previous track, and pause on/off. The LCD will display tracks info and drive status. An amplifier will be supported to allow the user to listen to the CD without using headphones.

قارئ الأقراص المضغوطة المستقل هو نظام يمنح المستخدم إمكانية تشغيل الأقراص و التحكم بالمسارات المراد الاستماع إليها من خلال الأزرار و أيضا استعراض بعض المعلومات عن العمليات التي يقوم بها النظام من خلال شاشة عرض مصغرة .

سوف يتم عمل ربط بين قارئ أقراص و شاشة العرض و أزرار التحكم من خلال عنصر التحكم الدقيق الذي يدعم منافذ الإدخال / الإخراج الخمسة المستخدمة لنقل البيانات و الإشارات . وسوف يتم التحكم بالأزرار من خلال رصد تغير الجهد الناتج عن كل زر و تحويله داخل عنصر التحكم الدقيق . و سيتم وصل مكبر صوت على قارئ الأقراص إلى سماعة .

Dedication

To our families for their patience

To our brothers and sisters

To our colleagues for their support and encouragement

To our martyrs for their courage

To our Supervisor Eng. Elayan Abu-Gharbyeh for his ideas, supports and advice

ACKNOWLEDGMENT

They said, "Who did not thank people, will not thank God".

Project Team would like to thank the supervisor Eng. Elayan Abu-Gharbyeh for his helpful supervising and directions.

Special thanks to teacher Rasmi Sayyed -Ahmed for his supportive ideas.

With our pleasure, we would like to thank our university, College of Engineering and Technology and the Electrical and Computer Department for their support and help.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter One

Introduction

Chapter One
Introduction

1.1 Overview

The world inclination today is to simplify the way of device usage and controlling without staying at a fixed place. This idea stems from the fact that many devices require a mean of any facility that saves time and labor either in usage or control.

The System implements the idea of device operating and controlling. The system focuses on making a stand a lone audio CD-Player. This System is useful to allow users to operate it away from computer. Basically, the system will build a CD-Player with its control button that allows you to play tracks and control them; also will provide an LCD display that gives the user information about the current playing media.

The implementation uses a smart electronic chip called Microcontroller that has specified pins that allow an IDE/ATA parallel bus to be attached to it. Eventually, the system has to connect the CD-Drive to the microcontroller, and attach some control buttons, display system and other electronic components to allow controlling the playing track. Some programming language will be used to program the microcontroller to allow connecting the control keys and display system.

This system is expected to fully implement the required connectivity between microcontroller and the CD-Drive and control keys and display system and adding some electronics to enable listen to the sound without headphones. On the other hand, it must guarantee the correct access to the functions already supported by the CD-Drive.

1.2 Literature Review

This project is the first project of its kind in our university, but there are some CD-Drive projects in the web that match our project.

We can categorize the Literature Survey into two main groups:

PPU Projects:

As we had mentioned it's the first one for the project idea.

Other Projects:

Through searching the Web, we had seen various and different projects and suggestions tackling the idea of this project, some of them are tightly related to it, such as:

1. "Mp3 CD-Player project"

Project Description:

This project is able to play CDs containing mp3-files, Audio CDs and optionally it has a build-in hard drive. About 12 hours of continuous music fit on a standard CD-R, the hard drive used (20 GB), CPU, RAM, sound cards, floppy …[1]

Relationship to our project:

This project is related to our project idea, isolating the CD-Drive from computer, also:

1. Using amplifiers and display screen.
2. Using buttons for control.

But there still general schemas that may use new ways, such as:

1. This project will not read mp3 files.
2. This project will not use hard disk.
3. This project, in general uses fewer component and smaller software, and makes fewer functions.

2. "An ATA/ATAPI CDROM application using a 8051 based Micro controller"

Project Description:

This project aimed to design a cheap CD player for phone systems- a device that plays CD's with recordings of music and people. The signals from this device would be fed into a telephone system. Whenever somebody would ring up the business and get "put on hold", they would hear the CD. [2]

Relationship to our project:

This project is related to our project because it connects the CD-Drive with the microcontroller, also using the CD-Drive as a stand alone system from the computer, also controlling the playing track.

But there still general differences from our project such as connecting telephony system.

1.4 Estimated Cost

This section lists the overall cost that is considered in implementing the system. The costs are divided into: hardware costs and software costs.

Hardware Estimated Cost

Table 1- 1: Hardware Cost

Component	Cost $
PIC Microcontroller	10
PIC Programmer	200
LCD Display	10
CD-Drive	20
Power supply	5
Buttons	5
Amplifier	5
Speaker	5
Cables	5
Miscellaneous	20

Software Estimated Cost

Table 1- 2: Software Cost

Software	Cost
Software Implementation	200$

1.5 Time Schedule

The following table illustrates how the time will be managed to reach the goal. This time management aims to clarify each step of the project and guarantees the full completion of the overall system.

Table 1- 3: Time Schedule

Weeks / Tasks	1 2	3 4	5 6	7 8	9 10	11 12	13 14	15 16	17 18	19 20	21 22	23 24	25 26	27 28	29 30	31 32
System definition collecting data																
Planning																
Requirement analysis																
System design																
Software design																
Implementation																
Testing & Operate project																
Documentation																

1.6 Report Contents

The documentation for this project is organized into seven chapters. Each chapter concerns with a logical or physical part of the system. It seems to have a hierarchical schema for simulating the system. However, the followings summarize briefly what each chapter will explain:

- **Chapter One: Introduction**

This chapter demonstrates an overview about the system, system design options, a literature review, estimated cost and time planning.

- **Chapter Two: Theoretical Background**

This chapter focuses on theories and materials that are related to our system operation and behavior. It mainly talks about CD-Drive, Microcontroller, LCD Display.

- **Chapter Three: Design Concepts**

This chapter describes the system in its abstract formula. It describes the project objectives, a general block diagram and how the system works.

- **Chapter Four: Hardware System Design**

This chapter discusses system design options and justifies those that are chosen in the project. A detailed description about the different project parts also is included.

- **Chapter Five: Software System Design**

This chapter handles the software related to our system, depicts flow charts about system operation and illustrates different algorithms and techniques that will be considered in writing the software.

- **Chapter Six: System Implementation and Testing**

This chapter manifests the implementation procedures to be acted so as to integrate the project. Then, a sequence of procedural testing will be listed. The testing comprises both software and hardware testing.

- **Chapter Seven: Conclusions and Future Work**

This chapter lists the problems faced us in accomplishing the system and how did they resolved. Notes and Conclusions will help readers are also included. A future work is also proposed.

Chapter Two

Theoretical Background

Chapter Two

Theoretical Background

Chapter Two
Theoretical Background

This Chapter talks about the hardware theoretical background that will be used in this project.

2.1 The CD-Drive:

2.1.1 General Description

CD-ROM (Compact Disk Read Only Memory) is a drive which reads aluminum-coated round plastic discs. The CD-ROM standard was officially introduced in 1982.
CD-ROM drives connect to an IDE port on the motherboard or hard drive interface card. IDE CD-ROM drives use standard ATAPI command set for controlling. IDE is commonly used for CD-ROM's in standard computers today.

Nowadays there are two commonly used CDROM interfaces used in PCs: IDE and SCSI. IDE CD-ROMs are the most common in normal PCs. SCSI CD-ROMs are commonly used in high-end PCs and workstations. Specifications for SCSI and AT Attachment Packet Interface (ATAPI) CD drives have long been established, but little testing has been done to ensure that drives conform to these standards. Both AT Attachment Packet Interface (ATAPI) and SCSI drive command set for reading audio CDs (for more information sees "ATA Packet Interface for CD-ROMs" and "SCSI-3 Multimedia Commands"

Audio data can be read from a CD-ROM drive by sending it a Read CD command and the address of a sector known to contain digital audio. The drive then returns a block of 2352 bytes of raw audio data (normal data reads return 2048 bytes). These data reads reflect an extra layer of error correction not available to sectors containing digital audio. Data tracks also contain "formatted Q-sub code data" that confirm to the drive exactly which sector is currently being read. Without this extra information, accurate positioning of the read head and error-free transmission of audio data is difficult. [3]

2.1.2 CD-ROM Media Organization

The formats written on the CD-ROM and CD-DA (Digital Audio) media require special interfacing considerations.

Discs may contain audio, data or a mixture of the two.

The Sector:

The physical format defined by the CD-ROM media standards provides 2352 bytes per sector. For usual computer data applications, 2048 bytes are used for user data, 12 bytes for a synchronization field, 4 bytes for a sector address tag field and 288 bytes.

A CD-ROM physical sector size is 2048, 2052, 2056, 2324, 2332, 2336, 2340 or 2352 bytes per sector. These values correspond to the user data plus various configurations of header, sub header and EDC/ECC.

The Track:

A track may be viewed as a partition of the CD-ROM address space. A CD-ROM media contains from one to ninety-nine tracks. All information sectors of a track are required to be of the same type (audio or data) and mode. Each change in the type of information on the disc requires a change in track number. A disc containing both audio and data would have at least two tracks, one for audio and one for data.

The tracks of a CD media are numbered consecutively with values between 1 and 99. However, the first information track may have a number greater than 1. Tracks have a minimum length of 300 sectors including any transition area that is part of a track.

The CD-ROM media standards require transition areas between tracks encoded with different types of information. In addition, transition areas may be used at the beginning or end of any track.

Areas inside the Media:

CD-ROM media have lead-in and lead-out areas. These areas are outside of the user-accessible area. The lead-in area of the media is designated track zero. The lead-out area is designated track 0AAh. The sub-channel Q in the lead-in track contains a table of contents (TOC) of the disc.

The Table of Contents:

The table of contents gives the absolute MSF location of the first information sector of each track. Control information (audio/data, method of audio encoding, etc.)

The MSF Address is the physical address written on CD-ROM discs. It is expressed as a sector count relative to either the beginning of the medium (absolute) or to the beginning of the current track (relative).

The MSF locations of the beginning of data tracks in the TOC are required to be accurate; however, the TOC values for audio tracks have a tolerance of plus or minus 75 sectors. Information from the TOC can be used to reply to a READ CD-ROM CAPACITY command. When this is done, the drive implementer shall consider the possible tolerances and return a value that allows access to all information sectors.

The Index:

An index is a partition of a track. Pre-gap areas are encoded with an index value of zero. Pause areas at the beginning of audio tracks are also encoded with an index value of zero. The first information sector of a track has an index value of one. Consecutive values up to 99 are permitted. Index information is not contained in the TOC. Not all sectors are encoded with the index value in the Q-sub-channel data (the requirement is 9 out of 10). A sector without an index value is presumed to have the same index as the preceding sector.

2.1.3 CD-ROM Physical Data Format

The physical format of CD-ROM and CD-DA media uses a smaller unit of synchronization than the more familiar magnetic or optical recording systems. The basic data stream synchronization unit is a small frame. This is not the same large frame (sector) as referred to in the MSF unit. Each small frame consists of 588 bits. A sector on CD-ROM media consists of 98 small frames.

A CD-ROM small frame consists of:

1. 1 synchronization pattern (24+3 bits)
2. 1 byte of sub-channel data (14+3 bits)
3. 24 bytes of data (24 x (14+3) bits)

4. 8 bytes of CIRC code (8 x (14+3) bits) Total: 588 bits.

Data, sub-channel and CIRC bytes are encoded with an 8-bit to 14-bit code; then three merging bits are added. The merging bits are chosen to provide minimum low frequency signal content and optimize phase lock loop performance.

2.1.3.1 Frame Format for Audio

Each small frame of an audio track on a two-channel CD-DA or CD-ROM media consists of six digitized 16-bit samples of each audio channel. These 24 bytes of data are combined with a synchronization pattern, CIRC bytes and a sub-channel byte to make a frame. Each frame takes approximately 136.05s to play. This gives a sampling rate of 44.1 kHz for each channel. The sub-channel information creates the higher level sector grouping for audio tracks.

2.1.3.2 Sector Format for Data

The data bytes of 98 small frames comprise the physical unit of data referred to as a sector. (98 small frames times 24 bytes per small frame equals 2352 bytes of data per sector.)

A sector that contains CD-ROM Data Mode 1 data has the following format:

1. 12-byte synchronization field
2. 4-byte CD-ROM data header

Absolute M field

Absolute S field

Absolute F field

CD-ROM data mode field

3. 2048-byte user data field
4. 4-byte error detection code
5. 8 bytes zero
6. 276-byte layered error correction code

A sector that contains CD-ROM Data Mode 2 data has the following format:

1. 12-byte synchronization field
2. 4-byte CD-ROM data header

Absolute M field

4. 8 bytes of CIRC code (8 x (14+3) bits) Total: 588 bits.

Data, sub-channel and CIRC bytes are encoded with an 8-bit to 14-bit code; then three merging bits are added. The merging bits are chosen to provide minimum low frequency signal content and optimize phase lock loop performance.

2.1.3.1 Frame Format for Audio

Each small frame of an audio track on a two-channel CD-DA or CD-ROM media consists of six digitized 16-bit samples of each audio channel. These 24 bytes of data are combined with a synchronization pattern, CIRC bytes and a sub-channel byte to make a frame. Each frame takes approximately 136.05s to play. This gives a sampling rate of 44.1 kHz for each channel. The sub-channel information creates the higher level sector grouping for audio tracks.

2.1.3.2 Sector Format for Data

The data bytes of 98 small frames comprise the physical unit of data referred to as a sector. (98 small frames times 24 bytes per small frame equals 2352 bytes of data per sector.)

A sector that contains CD-ROM Data Mode 1 data has the following format:

1. 12-byte synchronization field
2. 4-byte CD-ROM data header

Absolute M field

Absolute S field

Absolute F field

CD-ROM data mode field

3. 2048-byte user data field
4. 4-byte error detection code
5. 8 bytes zero
6. 276-byte layered error correction code

A sector that contains CD-ROM Data Mode 2 data has the following format:

1. 12-byte synchronization field
2. 4-byte CD-ROM data header

Absolute M field

Absolute S field

Absolute F field

CD-ROM data mode field

3. 2336-byte user data field (2048 bytes of mode 1 data plus 288 bytes of auxiliary data)

2.1.3.3 Sub-channel Information Formats

The sub-channel byte of each frame is assigned one bit for each of the 8 sub-channels, designated P, Q, R, S, T, U, V, W.

Sub-channel P is a simple flag bit that may be used for audio muting control and track boundary determination.

Sub-channel Q has a higher level of structure. All the sub-channel Q bits of a sector define the sub-channel Q information block. (For audio tracks, decoding the Q sub-channel is the only way to distinguish sector boundaries.)

The sub-channel Q block consists of 98 bits, one bit from each small frame in a sector. Three formats are defined for the sub-channel Q information block. The first format provides location information and is defined as follows:

1. 2-bit sub-channel synchronization field
2. 4-bit ADR field (defines the format)
3. 4-bit control field (defines the type of information in this sector)
4. 8-bit track number
5. 8-bit index number
6. 24-bit track relative MSF address
7. 8 bits Reserved (0)
8. 24-bit Absolute MSF address
9. 16-bit CRC error detection code

This format is required to exist in at least nine out of ten consecutive sectors.

2.1.4 Physical Interface

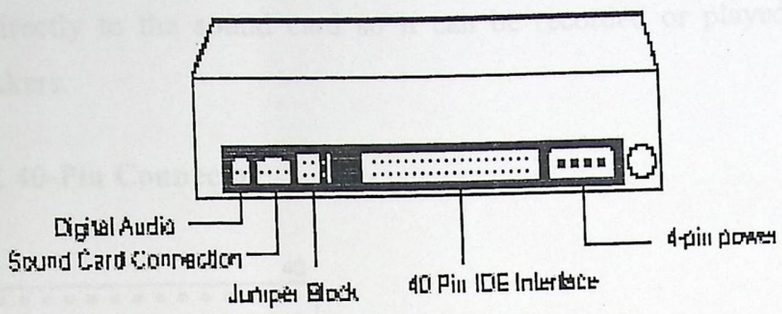


Figure 2-1: CD Drive Interface

Every CD-ROM drive contains a logic board. As with the hard disk drive, the job of this board is to both control the drive and interface to the PC either using IDE/ATAPI or SCSI, the connectors and jumpers, and the signals of the cable will be described in details.

2.1.4.1 Connectors and Jumpers

The connectors and jumpers on a CD-ROM are similar in most ways to what you will find on a hard disk drive. Mercifully, CD-ROM drive manufacturers have done a much better job of being at least somewhat standardized in the use of jumpers and connectors, and even in where they are located on the drive. All CD-ROMs that we have seen have their jumpers and connectors located at the back of the drive.

Standard 4-pin power connector on the back of a regular internal CD-ROM drive, the same kind that is used for hard disk drives and most other internal devices. This is pretty universal and is found on most every drive. The other connections and jumpers depend on the interface that the drive is using; an IDE/ATAPI drive will use different ones than a SCSI drive for example. For ATAPI, you will find the standard 40-pin data connector, along with jumpers to select the drive as a master or slave device. For SCSI, you will find a 50-pin connector and jumpers to set the device ID and termination. But in our project we will use the IDE/ATAPI.

One connector that is found on a CD-ROM and not on a hard disk drive is the audio connector that goes to the sound card. This three- or four-wire cable is used to send CD audio output directly to the sound card so it can be recorded or played back on the computer's speakers.

CDROM-IDE 40-Pin Connector

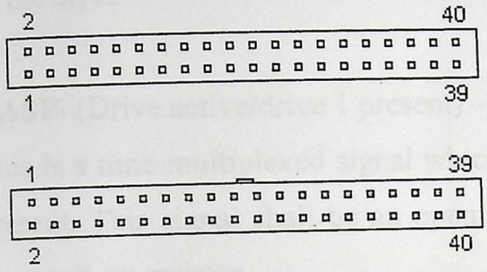


Figure 2-2: IDE Connector

Standard IDE/ATA hard disks and ATAPI devices use two different connectors. The first is the data connector, to which the IDE/ATA cable attaches. The second is the power connector, which comes from the power supply, and of course, provides power to the drive.

There are 40 wires in a regular IDE/ATA cable, so it's no surprise that there are 40 corresponding signals.

2.1.4.2 Signal descriptions

The interface signals and pins are described in some details:

CS10- (drive chip select 0) – 37

This is the chip select signal decoded from the host address bus used to select the Command Block Registers.

One connector that is found on a CD-ROM and not on a hard disk drive is the audio connector that goes to the sound card. This three- or four-wire cable is used to send CD audio output directly to the sound card so it can be recorded or played back on the computer's speakers.

CDROM-IDE 40-Pin Connector

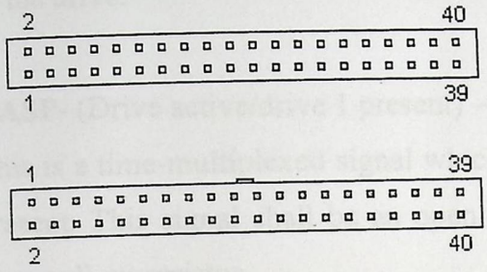


Figure 2-2: IDE Connector

Standard IDE/ATA hard disks and ATAPI devices use two different connectors. The first is the data connector, to which the IDE/ATA cable attaches. The second is the power connector, which comes from the power supply, and of course, provides power to the drive.

There are 40 wires in a regular IDE/ATA cable, so it's no surprise that there are 40 corresponding signals.

2.1.4.2 Signal descriptions

The interface signals and pins are described in some details:

CS10- (drive chip select 0) – 37

This is the chip select signal decoded from the host address bus used to select the Command Block Registers.

CS31- (drive chip select 1) – 38

This is the chip select signal decoded from the host address bus used to select the Control Block Registers.

DA0-2 (Drive address bus) - 35, 33, 36

This is the 3-bit binary coded address asserted by the host to access a register or data port in the drive.

DASP- (Drive active/drive 1 present) – 39

This is a time-multiplexed signal which indicates that a drive is active, or that Drive 1 is present. This signal shall be an open collector output and each drive shall have a 10K ohm pull-up resistor.

During power on initialization or after RESET- is negated, DASP- shall be asserted by Drive 1 within 400 m sec to indicate that Drive 1 is present. Drive 0 shall allow up to 450 m sec for Drive 1 to assert DASP-. If Drive 1 is not present, Drive 0 may assert DASP- to drive an activity LED.

DASP- shall be negated following acceptance of the first valid command by Drive 1 or after 31 seconds, whichever comes first.

Any time after negation of DASP-, either drive may assert DASP- to indicate that a drive is active.

DD0-DD15 (Drive data bus) – [3-18]

This is an 8- or 16-bit bidirectional data bus between the host and the drive. The lower 8 bits are used for 8-bit transfers e.g., register ECC bytes and if the drive supports the Features Register capability to enable 8-bit-only data transfers.

DIOR- (Drive I/O read) – 25

This is the Read strobe signal. The falling edge of DIOR- enables data from a register or the data port of the drive onto the host data bus, DD0-DD7 or DD0-DD15. The rising edge of DIOR- latches data at the host.

DIOW- (Drive I/O write) – 23

This is the Write strobe signal. The rising edge of DIOW- clocks data from the host data bus, DD0-DD7 or DD0-DD15, into a register or the data port of the drive.

DMACK- (DMA acknowledge) – 29

This signal shall be used by the host in response to DMARQ to either acknowledge that data has been accepted, or that data is available.

DMARQ (DMA request) – 21

This signal, used for DMA data transfers between host and drive, shall be asserted by the drive when it is ready to transfer data to or from the host.

The direction of data transfer is controlled by DIOR- and DIOW-. This signal is used in a handshake manner with DMACK- i.e., the drive shall wait until the host asserts DMACK- before negating DMARQ, and re-asserting DMARQ if there is more data to transfer.

When a DMA operation is enabled, IOCS16-, CS1FX-, and CS3FX- shall not be asserted and transfers shall be 16-bits wide.

INTRQ (Drive interrupt) – 31

This signal is used to interrupt the host system. INTRQ is asserted only when the drive has a pending interrupt, the drive is selected, and the host has cleared nIEN in the Device Control Register. If nIEN=1, or the drive is not selected, this output is in a high impedance state, regardless of the presence or absence of a pending interrupt.

INTRQ shall be negated by:

- assertion of RESET- or;
- the setting of SRST of the Device Control Register, or;
- the host writing the Command Register or;
- the host reading the Status Register.

IORDY (I/O channel ready) – 27

This signal is negated to extend the host transfer cycle of any host register access (Read or Write) when the drive is not ready to respond to a data transfer request. When IORDY is not negated, IORDY shall be in a high impedance state.

PDIAG- (Passed diagnostics) – 34

This signal shall be asserted by Drive 1 to indicate to Drive 0 that it has completed diagnostics

RESET- (Drive reset) – 1

This signal from the host system shall be asserted for at least 25 u sec after voltage levels have stabilized during power on and negated thereafter unless some event requires that the drive(s) be reset following power on.

CSEL (Cable select) – 28

The drive is configured as either Drive 0 or Drive 1 depending upon the value of CSEL:

- If CSEL is grounded then the drive address is 0
- If CSEL is open then the drive address is 1

Special cabling can be used by the system manufacturer to selectively ground CSEL. [3]

2.1.4.3 DC Cable

The drive receives DC power through a 4-pin or a low-power application 3-pin connector.

A drive designed for 3,3V applications may be plugged into a receptacle designed to accept a drive designed for 5V applications, with 12V lines for additional power. It is not required that the drive operate, but it is recommended that precautions be taken to prevent damage to the drive. A drive designed for 5V applications may be plugged into a receptacle designed to accept a drive designed for 3,3V applications, with 5V lines for additional power. It is not anticipated that damage could occur to the drive, but it is likely to fail in an undetermined manner. [7]

2.1.5 Logical Interface:

I/O registers inside CD-Drive descriptions

Communication to or from the drive is through an I/O Register that routes the input or output data to or from registers.

These registers can be categorized into two types:

- The Control Block Registers are used for drive control and to post alternate status.

These registers include:

1. Device Control register
2. Alternate Status register

- The Command Block Registers are used for sending commands to the drive or

Posting status from the drive.

These registers include:

1. Status register
2. Command register
3. Cylinder high register
4. Cylinder low register
5. Data register
6. Drive address register
7. Drive/head register
8. Error register
9. Features register
10. Sector count register
11. Sector number register

- **Device control register**

Direction

This register is writing only. If this address is read by the host, the Alternate Status register is read.

Access restrictions

This register shall only be written when DMACK- is not asserted.

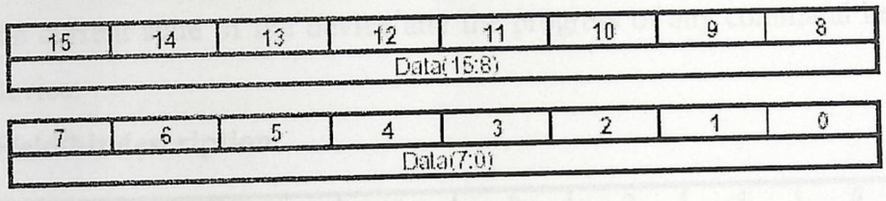
Effectiveness

The content of this register shall take effect when written.

Functional description

This register allows a host to software reset attached devices and to enable or disable the assertion of the INTRQ signal by a selected device. When the Device Control register is written, both devices respond to the write regardless of which device is selected. When the SRST bit is set to one, both devices shall perform the Software reset protocol. The device shall respond to the SRST bit when in the SLEEP mode.

Field/bit description



- **Alternate status register**

Direction

This register is read only. If this address is written to by the host, the Device Control register is written.

Access restrictions

When the BSY bit is set to one, the other bits in this register shall not be used. The entire contents of this register are not valid while the device is in Sleep mode.

Effect

Reading this register shall not clear a pending interrupt.

Functional description

This register contains the same information as the Status register in the command block.

- **Status Register**

Direction

This register is read only. If this address is written to by the host, the Command register is written.

Access restrictions

The contents of this register, except for BSY, shall be ignored when BSY is set to one. BSY is valid at all times. The contents of this register are not valid while a device is in the Sleep mode.

Effect

Reading this register when an interrupt is pending causes the interrupt pending to be cleared. The host should not read the Status register when an interrupt is expected as this may clear the interrupt pending before the INTRQ can be recognized by the host.

Functional description

This register contains the device status. The contents of this register are updated to reflect the current state of the device and the progress of any command being executed by the device.

Field/bit description

7	6	5	4	3	2	1	0
BSY	DRDY	#	#	DRQ	Obsolet e	Obsolet e	ERR

- **Command register**

Direction

This register is writing only. If this address is read by the host, the Status register is read.

Access restrictions

For all commands except DEVICE RESET, this register shall only be written when BSY and DRQ are both

cleared to zero and DMACK- is not asserted. If written when BSY or DRQ is set to one, the results of writing

the Command register are indeterminate except for the DEVICE RESET command. For a device in the Sleep mode, writing of the Command register shall be ignored except for

writing of the DEVICE RESET command to a device that implements the PACKET Command feature set.

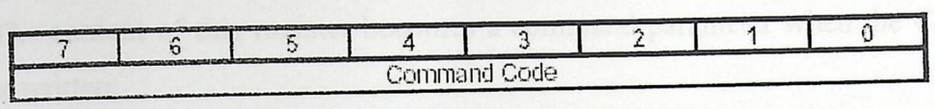
Effect

Command processing begins when this register is written. The content of the Command Block registers become parameters of the command when this register is written. Writing this register clears any pending interrupt condition.

Functional description

This register contains the command code being sent to the device. Command execution begins immediately after this register is written.

Field/bit description



- **Cylinder high register**

Direction

This register is read/write.

Access restrictions

This register shall be written only when both BSY and DRQ are cleared to zero and DMACK- is not asserted.

The contents of this register are valid only when BSY is cleared to zero. If this register is written when BSY or DRQ is set to one, the result is indeterminate. The contents of this register are not valid while a device is in the Sleep mode.

Effect

The content of this register becomes a command parameter when the Command register is written.

Functional description

The content of this register is command dependent.

- **Cylinder low register**

Direction

This register is read/write.

Access restrictions

This register shall be written only when both BSY and DRQ are cleared to zero and DMACK- is not asserted.

The contents of this register are valid only when BSY is cleared to zero. If this register is written when BSY or DRQ is set to one, the result is indeterminate. The contents of this register are not valid while a device is in the Sleep mode.

Effect

The content of this register becomes a command parameter when the Command register is written.

Functional description

The content of this register is command dependent.

- **Data register**

Direction

This register is read/write.

Access restrictions

This register shall be written only when both BSY and DRQ are cleared to zero and DMACK- is not asserted.

The contents of this register are valid only when BSY is cleared to zero. If this register is written when BSY or DRQ is set to one, the result is indeterminate. The contents of this register are not valid while a device is in the Sleep mode.

Effect

The content of this register becomes a command parameter when the Command register is written.

Functional description

The content of this register is command dependent.

Field/bit description

15	14	13	12	11	10	9	8
Data(15:8)							

7	6	5	4	3	2	1	0
Data(7:0)							

- **Drive address register**

This register contains the inverted drive select and head select addresses of the currently selected drive

- **Drive/head register**

Direction

This register is read/write.

Access restrictions

This register shall be written only when both BSY and DRQ are cleared to zero and DMACK- is not asserted.

Effect

The DEV bit becomes effective when this register is written by the host or the signature is set by the device. All other bits in this register become a command parameter when the Command register is written.

Functional description

Bit 4, DEV, in this register selects the device. Other bits in this register are command dependent.

Field/bit description

7	6	5	4	3	2	1	0
Obsolete	#	Obsolete	DEV	#	#	#	#

- Error register

Direction

This register is read only. If this address is written to by the host, the Features register is written.

Access restrictions

The contents of this register shall be valid when BSY and DRQ equal zero and ERR equals one. The contents of this register shall be valid upon completion of power-on, or after a hardware or software reset, or after command completion of an EXECUTE DEVICE DIAGNOSTICS or DEVICE RESET command. The contents of this register are not valid while a device is in the Sleep mode.

Effect

None.

Functional description

This register contains status for the current command.

At command completion of any command except EXECUTE DEVICE DIAGNOSTIC the contents of this register are valid when the ERR bit is set to one in the Status register.

Field/bit description

7	6	5	4	3	2	1	0
#	#	#	#	#	ABRT	#	#

- Features register

Direction

This register is write only. If this address is read by the host, the Error register is read.

Access restrictions

This register shall be written only when BSY and DRQ equal zero and DMACK- is not asserted. If this register is written when BSY or DRQ is set to one, the result is indeterminate.

Effect

The content of this register becomes a command parameter when the Command register is written.

Functional description

The content of this register is command dependent.

- **Sector count register**

Direction

This register is read/write.

Access restrictions

This register shall be written only when both BSY and DRQ are zero and DMACK- is not asserted. The contents of this register are valid only when both BSY and DRQ are zero. If this register is written when BSY or DRQ is set to one, the result is indeterminate. The contents of this register are not valid while a device is in the Sleep mode.

Effect

The content of this register becomes a command parameter when the Command register is written.

Functional description

The content of this register is command dependent

- **Sector number register**

Direction

This register is read/write.

Access restrictions

This register shall be written only when both BSY and DRQ are zero and DMACK- is not asserted. The contents of this register are valid only when both BSY and DRQ are zero. If this register is written when BSY or DRQ is set to one, the result is indeterminate. The contents of this register are not valid while a device is in the Sleep mode.

Effect

The content of this register becomes a command parameter when the Command register is written.

Functional description

The content of this register is command dependent. [3]

2.1.6 ATAPI Command Packet

An ATAPI command is communicated by sending a Command Packet to the ATAPI CD-ROM Drive. For several commands, the Command Packet is accompanied by a list of parameters sent upon receiving an interrupt following the Command Packet being sent. See the specific commands for detailed information. The Command Packet always has an operation code as its first byte.

For all commands, if there is an invalid parameter in the Command Packet, then the ATAPI Device shall abort the command without altering the medium.

Typical Command Packet for Most Commands

Byte \ Bit	7	6	5	4	3	2	1	0
0	Operation Code							
1	Reserved			Reserved				
2	(MSB) Logical Block Address (if required)							
3								
4								(LSB)
5	Reserved							
6	Transfer Length (if required) or Parameter List Length (if required) or Allocation Length (if required)(LSB)							
7 - 8	(MSB)							
9	Reserved							
10	Reserved							
11	Reserved							

Operation Code

The operation code of the Command Packet has a group code field and a command code field. The three-bit group code field provides for eight groups of command codes. The five-bit command code field provides for thirty-two command codes in each group. Thus, a total of 256 possible operation codes exist. Operation codes are defined in the subsequent Transfer Length

The Transfer Length Field specifies the amount of data to be transferred, usually the number of blocks. For several commands the transfer length indicates the requested number of bytes to be sent as defined in the command description. For these commands the Transfer Length Field may be identified by a different name.

In commands that use multiple bytes for the transfer length, a transfer length of zero indicates that no data transfer shall take place. A value of one or greater indicates the number of blocks that shall be transferred.

Parameter List Length

The Parameter List Length is used to specify the number of bytes to be sent to the Drive. This field is typically used in Command Packets for parameters that are sent to a Drive (e.g. mode parameters, diagnostic parameters, etc.). A parameter length of zero indicates that no data shall be transferred.

Allocation Length

The Allocation Length Field specifies the maximum number of bytes that a Host Computer has allocated for returned data. An allocation length of zero indicates that no data shall be transferred. The Drive shall terminate the data transfer when allocation length bytes have been transferred or when all available data have been transferred to the Host Computer, whichever is less. The allocation length is used to limit the maximum amount of data (e.g. sense data, mode data, etc.) returned to a Host Computer.

Status

A Status byte shall be sent from the Drive to the Host Computer at the completion of each command unless the command is terminated by one of the following events:

1. A hard reset condition.
2. An unexpected event.

2.1.7 ATAPI Packet Commands for CD-ROM Devices

Command Description	Opcode	Type
INQUIRY	12h	M
LOAD/UNLOAD CD	A6h	O***
MECHANISM STATUS	BDh	M
MODE SELECT (10)	55h	M
MODE SENSE (10)	5Ah	M
PAUSE/RESUME	4Bh	O*
PLAY AUDIO (10)	45h	O*
PLAY AUDIO MSF	47h	O*
PLAY CD	BCh	O
PREVENT/ALLOW MEDIUM REMOVAL	1Eh	M
READ (10)	28h	M
READ (12)	A8h	M
READ CD-ROM CAPACITY	25h	M
READ CD	BEh	M
READ CD MSF	B9h	M
READ HEADER	44h	M
READ SUB-CHANNEL	42h	M
READ TOC	43h	M
REQUEST SENSE	03h	M
SCAN	BAh	O
SEEK	2Bh	M
SET CD SPEED	BBh	O
STOP PLAY / SCAN	4Eh	M
START STOP UNIT	1Bh	M
TEST UNIT READY	00h	M
Reserved for future use	BFh	

2.1.7.1 INQUIRY Command

The INQUIRY command requests that information regarding parameters of the ATAPI CD-ROM Drive be sent to the Host Computer. An option allows the Host Computer to request additional information about the ATAPI CD-ROM Drive.

Bit Byte	7	6	5	4	3	2	1	0
0	Operation code (12h)							
1	Reserved							Reserved
2	Reserved							
3	Reserved							
4	Allocation Length							
5	Reserved							
6	Reserved							
7	Reserved							
8	Reserved							
9	Reserved							
10	Reserved							
11	Reserved							

The INQUIRY command shall return CHECK CONDITION status only when the ATAPI CD-ROM Drive cannot return the requested INQUIRY data. The INQUIRY data should be returned even though the peripheral device may not be ready for other commands

2.1.7.2 PAUSE/RESUME Command

The PAUSE/RESUME command requests that the device stop or start an audio play operation. This command is used with PLAY AUDIO commands that are currently executing.

Bit Byte	7	6	5	4	3	2	1	0
0	Operation Code (4Bh)							
1	Reserved							
2	Reserved							
3	Reserved							
4	Reserved							
5	Reserved							
6	Reserved							
7	Reserved							Resume
8	Reserved							
9	Reserved							
10	Reserved							
11								

2.1.7.1 INQUIRY Command

The INQUIRY command requests that information regarding parameters of the ATAPI CD-ROM Drive be sent to the Host Computer. An option allows the Host Computer to request additional information about the ATAPI CD-ROM Drive.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (12h)							
1	Reserved							Reserved
2	Reserved							
3	Reserved							
4	Allocation Length							
5	Reserved							
6	Reserved							
7	Reserved							
8	Reserved							
9	Reserved							
10	Reserved							
11	Reserved							

The INQUIRY command shall return CHECK CONDITION status only when the ATAPI CD-ROM Drive cannot return the requested INQUIRY data. The INQUIRY data should be returned even though the peripheral device may not be ready for other commands

2.1.7.2 PAUSE/RESUME Command

The PAUSE/RESUME command requests that the device stop or start an audio play operation. This command is used with PLAY AUDIO commands that are currently executing.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (4Bh)							
1	Reserved							
2	Reserved							
3	Reserved							
4	Reserved							
5	Reserved							
6	Reserved							
7	Reserved							Resume
8	Reserved							
9	Reserved							
10	Reserved							
11								

A Resume bit of zero causes the drive to enter the hold track state with the audio output muted after the current block is played. A Resume bit of one causes the drive to release the pause/scan and begin play at the block following the last block played/scanned.
If an audio play operation cannot be resumed and the resume bit is one, the command is terminated with CHECK CONDITION status. If the resume bit is zero and an audio play operation cannot be paused, (no audio play operation has been requested, or the requested audio play operation has been completed), the command is terminated with CHECK CONDITION status. Stop Play/Play Audio/Audio Scan/Pause/Resume Sequencing. It shall not be considered an error to request a PAUSE when a pause is already in effect or to request a RESUME when a play operation is in progress.

2.1.7.3 *PLAY AUDIO MSF Command*

The PLAY AUDIO MSF command requests that the ATAPI CD-ROM Drive begin an audio playback operation. The command function and the output of audio signals shall be as specified by the settings of the mode parameters including the SOTC bit.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation Code (47h)							
1	Reserved							
2	Reserved							
3	Starting M Field							
4	Starting S Field							
5	Starting F Field							
6	Ending M Field							
7	Ending S Field							
8	Ending F Field							
9	Reserved							
10	Reserved							
11	Reserved							

This command responds with immediate status, allowing overlapped commands. This command shall set the DSC bit upon command completion
The Starting M field, the Starting S field, and the Starting F field specify the absolute MSF address at which the audio play operation shall begin. The Ending M field, the Ending S field, and the Ending F field specify the absolute MSF address where the audio play operation shall end. All contiguous audio sectors between the starting and the ending

MSF address shall be played. If the Starting Minutes, Seconds and Frame Fields are set to FFh, the Starting address is taken from the Current Optical Head location. This allows the Audio Ending address to be changed without interrupting the current playback operation. A Starting MSF address equal to an ending MSF address causes no audio play operation to occur. This shall not be considered an error. If the Starting MSF address is greater than the Ending MSF address, the command shall be terminated with CHECK CONDITION status.

2.1.7.4 START/STOP UNIT Command

The START/STOP UNIT command requests that the ATAPI CD-ROM Drive enable or disable media access operations.

Bit Byte	7	6	5	4	3	2	1	0
0	Operation code (1Bh)							
1	Reserved							Immed
2	Reserved							
3	Reserved							
4	Reserved						LoEj	Start
5								
6	Reserved							
7	Reserved							
8	Reserved							
9	Reserved							
10	Reserved							
11	Reserved							

An immediate (Immed) bit of one indicates that status shall be returned as soon as the Command Packet has been validated. An Immed bit of zero indicates that status shall be returned after the operation is completed.

A start bit of one request the Device is made ready for use. A start bit of zero requests that the Device be stopped (media cannot be accessed by the Host Computer).

Any attempt to Eject or Load a Disc when the Drive does not support that capability shall result in an error condition being reported to the Host (Sense key 05 ILLEGAL REQUEST, Sense Code 24 INVALID FIELD IN COMMAND PACKET.)

A load eject (LoEj) bit of zero requests that no action be taken regarding loading or ejecting the medium. A LoEj bit of one requests that the medium be unloaded if the start bit is zero. A LoEj bit of one request that the medium be loaded if the start bit is one.

2.1.7.5 READ TOC Command

The READ TOC command requests that the ATAPI CD-ROM Drive transfer data from the table of contents to the Host Computer. Some drives will cache the TOC data and will be able to return it during a Play command. Drives that do not cache the data will generate an error and not complete the command.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (43h)							
1	Reserved						MSF (Mandatory)	Reserved
2	Reserved					Format		
3	Reserved							
4	Reserved							
5	Reserved							
6	Starting Track / Session Number							
7	MSB Allocation Length							
8								LSB
9	Format	Reserved						
10	Reserved							
11	Reserved							

2.1.7.6 READ SUB-CHANNEL Command

The READ SUB-CHANNEL command requests that the ATAPI CD-ROM Drive return the requested sub-channel data plus the state of play operations.

Bit / Byte	7	6	5	4	3	2	1	0
0	Operation code (42h)							
1	Reserved						MSF (Mandatory)	Reserved
2	Reserved	SubQ (Mandatory)	Reserved					
3	Sub-channel Data Format							
4	Reserved							
5	Reserved							
6	Track Number							
7	MSB Allocation Length							
8								LSB
9	Reserved							
10	Reserved							
11	Reserved							

Sub-channel data returned by this command may be from the last appropriate sector encountered by a current or previous media accessing operation. When there is no current play operation, the ATAPI CD-ROM Drive may access the media to read the sub-channel data. The ATAPI CD-ROM Drive is responsible for ensuring that the data returned are current and consistent. [4]

2.2 PIC Microcontroller:

Stand alone CD drive needs a microcontroller with large number of IO ports, internal memory, converters, timers and communication units. So, this project is supported by the PIC (Peripheral Interface Controller) as an optimal choice.

2.2.1 Device Features

In general, the PIC has the following features:

- Five IO Ports, port A,B,C,D,E
- 10-bit Analog-to-Digital Module, 8 input channels
- Data Memory (368 bytes)
- EEPROM Data Memory (256 bytes)
- Flash Program Memory (8k)
- Serial Communications (MSSP,USART)
- Parallel Communication (PSP)
- Three Timers
- Instruction Set (35 instructions)
- Interrupts (15)
- Analog Comparator (2)

Peripheral Features:

• Timer0: 8-bit timer/counter with 8-bit prescaler

• Timer1: 16-bit timer/counter with prescaler, can be incremented during Sleep via external crystal/clock

• Timer2: 8-bit timer/counter with 8-bit period register, prescaler and postscaler

• Universal Synchronous Asynchronous Receiver Transmitter (USART/SCI) with 9-bit address detection

• Parallel Slave Port (PSP) – 8 bits wide with external RD, WR and CS controls (40/44-pin only)

• Brown-out detection circuitry for Brown-out Reset (BOR)

Analog Features:

- 10-bit, up to 8-channel Analog-to-Digital Converter (A/D)
- Brown-out Reset (BOR)
- Analog Comparator module with:
- Two analog comparators
- Programmable on-chip voltage reference (VREF) module
- Programmable input multiplexing from device inputs and internal voltage reference
- Comparator outputs are externally accessible

High-Performance RISC CPU:

- Only 35 single-word instructions to learn
- All single-cycle instructions except for program branches, which are two-cycle
- Operating speed: DC – 20 MHz clock input DC – 200 ns instruction cycle
- Up to 8K x 14 words of Flash Program Memory, Up to 368 x 8 bytes of Data Memory (RAM), Up to 256 x 8 bytes of EEPROM Data Memory

Special Microcontroller Features:

- 100,000 erase/write cycle Enhanced Flash program memory typical
- 1,000,000 erase/write cycle Data EEPROM memory typical
- Data EEPROM Retention > 40 years
- Self-reprogrammable under software control
- In-Circuit Serial Programming™ (ICSP™) via two pins
- Single-supply 5V In-Circuit Serial Programming
- Watchdog Timer (WDT) with its own on-chip RC oscillator for reliable operation
- Programmable code protection
- Power saving Sleep mode
- Selectable oscillator options
- In-Circuit Debug (ICD) via two pins

CMOS Technology:

- Low-power, high-speed Flash/EEPROM technology
- Fully static design
- Wide operating voltage range (2.0V to 5.5V)
- Commercial and Industrial temperature ranges
- Low-power consumption

2.2.2 Memory Organization

There are three memory blocks; the program memory and data memory have separate buses so that concurrent access can occur. The third memory is EEPROM data memory block

Program Memory Organization

The PIC16F877 devices have a 13-bit program counter capable of addressing an 8K word x 14 bit program memory space. The PIC16F877 devices have 8K words x 14 bits of Flash program memory. Accessing a location above the physically implemented address will cause a wraparound.

Data Memory Organization

The data memory is partitioned into multiple banks which contain the General Purpose Registers and the Special Function Registers. Bits RP1 (Status<6>) and RP0 (Status<5>) are the bank select bits. Each bank extends up to 7Fh (128 bytes). The lower locations of each bank are reserved for the Special Function Registers. Above the Special Function Registers are General Purpose Registers, implemented as static RAM. All implemented banks contain Special Function Registers. Some frequently used Special Function Registers from one bank may be mirrored in another bank for code reduction and quicker access.

EEPROM and Flash Program Memory

The data EEPROM and Flash program memory is readable and writable during normal operation. This memory is not directly mapped in the register file space. Instead, it is indirectly addressed through the Special Function Registers.

The EEPROM data memory allows single-byte read and write. The Flash program memory allows single-word reads and four-word block writes. Program memory write operations automatically perform an erase-before write on blocks of four words. A byte write in data EEPROM memory automatically erases the location and writes the new data (erase-before-write). The write time is controlled by an on-chip timer. The write/erase voltages are generated by an on-chip charge pump, rated to operate over the voltage range of the device for byte or word operations.

When the device is code-protected, the CPU may continue to read and write the data EEPROM memory. Depending on the settings of the write-protect bits, the device may or may not be able to write certain blocks of the program memory; however, reads of the program memory are allowed. When code-protected, the device programmer can no longer access data or program memory; this does NOT inhibit internal reads or writes. [6]

For reading and writing from and to operations flash program and EPROM memory, see appendix.

2.2.3 Special Function Registers

The Special Function Registers are registers used by the CPU and peripheral modules for controlling the desired operation of the device. These registers are implemented as static RAM. The Special Function Registers can be classified into two sets: core (CPU) and peripheral.

- **Status Register**

The Status register contains the arithmetic status of the ALU, the Reset status and the bank select bits for data memory.

The Status register can be the destination for any instruction, as with any other register. If the Status register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled.

These bits are set or cleared according to the device logic. Furthermore, the TO and PD bits are not writable, therefore, the result of an instruction with the Status register as destination may be different than intended.

- **OPTION_REG Register**

The OPTION_REG Register is a readable and writable register, which contains various control bits to configure the TMR0 prescaler/WDT postscaler (single assignable register known also as the prescaler), the external INT interrupt, TMR0 and the weak pull-ups on PORTB.

- **INTCON Register**

The INTCON register is a readable and writable register, which contains various enable and flag bits for the TMR0 register overflow, RB port change and external RB0/INT pin interrupts.

- **PIE1 Register**

The PIE1 register contains the individual enable bits for the peripheral interrupts.

- **PIR1 Register**

The PIR1 register contains the individual flag bits for the peripheral interrupts.

- **PIE2 Register**

The PIE2 register contains the individual enable bits for the CCP2 peripheral interrupt, the SSP bus collision interrupt, EEPROM write operation interrupt and the comparator interrupt.

- **PIR2 Register**

The PIR2 register contains the flag bits for the CCP2 interrupt, the SSP bus collision interrupt, EEPROM write operation interrupt and the comparator interrupt.

- **PCON Register**

The Power Control (PCON) register contains flag bits to allow differentiation between a Power-on Reset (POR), a Brown-out Reset (BOR), a Watchdog Reset (WDT) and an external MCLR Reset.

- **PCL and PCLATH**

The Program Counter (PC) is 13 bits wide. The low byte comes from the PCL register which is a readable and writable register. The upper bits (PC<12:8>) are not readable, but are indirectly writable through the PCLATH register. On any Reset, the upper bits of the PC will be cleared.

- **Indirect Addressing, INDF and FSR Registers**

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing. Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself, indirectly (FSR = 0) will read 00h. Writing to the INDF register indirectly results in a no operation (although status bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (Status<7>).

2.2.4 I/O PORTS

Some pins for these I/O ports are multiplexed with an alternate function for the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

- **PORTA and the TRISA Register**

PORTA is a 6-bit wide, bidirectional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISA bit (=

- **PCON Register**

The Power Control (PCON) register contains flag bits to allow differentiation between a Power-on Reset (POR), a Brown-out Reset (BOR), a Watchdog Reset (WDT) and an external MCLR Reset.

- **PCL and PCLATH**

The Program Counter (PC) is 13 bits wide. The low byte comes from the PCL register which is a readable and writable register. The upper bits (PC<12:8>) are not readable, but are indirectly writable through the PCLATH register. On any Reset, the upper bits of the PC will be cleared.

- **Indirect Addressing, INDF and FSR Registers**

The INDF register is not a physical register. Addressing the INDF register will cause indirect addressing. Indirect addressing is possible by using the INDF register. Any instruction using the INDF register actually accesses the register pointed to by the File Select Register, FSR. Reading the INDF register itself, indirectly (FSR = 0) will read 00h. Writing to the INDF register indirectly results in a no operation (although status bits may be affected). An effective 9-bit address is obtained by concatenating the 8-bit FSR register and the IRP bit (Status<7>).

2.2.4 I/O PORTS

Some pins for these I/O ports are multiplexed with an alternate function for the peripheral features on the device. In general, when a peripheral is enabled, that pin may not be used as a general purpose I/O pin.

- **PORTA and the TRISA Register**

PORTA is a 6-bit wide, bidirectional port. The corresponding data direction register is TRISA. Setting a TRISA bit (= 1) will make the corresponding PORTA pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISA bit (=

0) will make the corresponding PORTA pin an output (i.e., put the contents of the output latch on the selected pin).

Reading the PORTA register reads the status of the pins, whereas writing to it will write to the port latch. All write operations are read-modify-write operations. Therefore, a write to a port implies that the port pins are read; the value is modified and then written to the port data latch.

- **PORTB and the TRISB Register**

PORTB is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISB. Setting a TRISB bit (= 1) will make the corresponding PORTB pin an input (i.e., put the corresponding output driver in a High-Impedance mode). Clearing a TRISB bit (= 0) will make the corresponding PORTB pin an output (i.e., put the contents of the output latch on the selected pin).

Three pins of PORTB are multiplexed with the In-Circuit Debugger and Low-Voltage Programming function: RB3/PGM, RB6/PGC and RB7/PGD. Each of the PORTB pins has a weak internal pull-up. A single control bit can turn on all the pull-ups. This is performed by clearing bit RBPU (OPTION_REG<7>). The weak pull-up is automatically turned off when the port pin is configured as an output. The pull-ups are disabled on a Power-on Reset.

RB0/INT is an external interrupt input pin and is configured using the INTEDG bit (OPTION_REG<6>).

Four of the PORTB pins, RB7:RB4, have an interrupt-on-change feature. Only pins configured as inputs can cause this interrupt to occur (i.e., any RB7:RB4 pin configured as an output is excluded from the interrupt-on-change comparison). The input pins (of RB7:RB4) are compared with the old value latched on the last read of PORTB.

The "mismatch" outputs of RB7:RB4 are OR'ed together to generate the RB port change interrupt with flag bit RBIF (INTCON<0>).

- **PORTC and the TRISC Register**

PORTC is an 8-bit wide, bidirectional port. The corresponding data direction register is TRISC. Setting a TRISC bit (= 1) will make the corresponding PORTC pin an input (i.e.,

put the corresponding output driver in a High-Impedance mode). Clearing a TRISC bit (= 0) will make the corresponding PORTC pin an output (i.e., put the contents of the output latch on the selected pin).

- **PORTD and TRISD Registers**

PORTD is an 8-bit port with Schmitt Trigger input buffers. Each pin is individually configurable as an input or output. PORTD can be configured as an 8-bit wide microprocessor port (Parallel Slave Port) by setting control bit, PSPMODE (TRISE<4>). In this mode, the input buffers are TTL.

- **PORTE and TRISE Register**

PORTE has three pins (RE0/RD/AN5, RE1/WR/AN6 and RE2/CS/AN7) which are individually configurable as inputs or outputs. These pins have Schmitt Trigger input buffers.

The PORTE pins become the I/O control inputs for the microprocessor port when bit PSPMODE (TRISE<4>) is set. In this mode, the user must make certain that the TRISE<2:0> bits are set and that the pins are configured as digital inputs. Also, ensure that ADCON1 is configured for digital I/O. In this mode, the input buffers are TTL.

PORTE pins are multiplexed with analog inputs. When selected for analog input, these pins will read as '0's. TRISE controls the direction of the RE pins, even when they are being used as analog inputs. The user must make sure to keep the pins configured as inputs when using them as analog inputs.

2.2.5 Analog-to-Digital Converter (A/D) Module

The Analog-to-Digital (A/D) Converter module has eight inputs for the 40/44-pin devices.

The conversion of an analog input signal results in a corresponding 10-bit digital number. The A/D module has high and low-voltage reference input that is software selectable to some combination of VDD, VSS, RA2 or RA3.

The A/D converter has a unique feature of being able to operate while the device is in Sleep mode. To operate in Sleep, the A/D clock must be derived from the A/D's internal RC oscillator.

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 Register 0 (ADCON0)
- A/D Control Register 1 (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 (RA3 can also be the voltage reference) or as digital I/O.

The ADRESH: ADRESL registers contain the 10-bit result of the A/D conversion. When the A/D conversion is complete, the result is loaded into this A/D Result register pair, the GO/DONE bit (ADCON0<2>) is cleared and the A/D interrupt flag bit ADIF is set.

After the A/D module has been configured as desired, the selected channel must be acquired before the conversion is started. The analog input channels must have their corresponding TRIS bits selected as inputs.

To do an A/D Conversion, the following steps are taken into account:

1. Configure the A/D module:
   - Configure analog pins/voltage reference and digital I/O (ADCON1)
   - Select A/D input channel (ADCON0)
   - Select A/D conversion clock (ADCON0)
   - Turn on A/D module (ADCON0)
2. Configure A/D interrupts (if desired):
   - Clear ADIF bit
   - Set ADIE bit
   - Set PEIE bit
   - Set GIE bit
3. Wait the required acquisition time.

4. Start conversion:

• Set GO/DONE bit (ADCON0)

5. Wait for A/D conversion to complete by either:

• Polling for the GO/DONE bit to be cleared (interrupts disabled); OR

• Waiting for the A/D interrupt

6. Read A/D Result register pair (ADRESH: ADRESL), clear bit ADIF if required.

7. For the next conversion, go to step 1 or step 2 as required. The A/D conversion time per bit is defined as TAD.

2.2.6 Timers

- **Timer0 Module :**

The Timer0 module timer/counter has the following features:

• 8-bit timer/counter

• Readable and writable

• 8-bit software programmable prescaler

• Internal or external clock select

• Interrupt on overflow from FFh to 00h

• Edge select for external clock

Timer mode is selected by clearing bit T0CS (OPTION_REG<5>). In Timer mode, the Timer0 module will increment every instruction cycle (without prescaler). If the TMR0 register is written, the increment is inhibited for the following two instruction cycles. The user can work around this by writing an adjusted value to the TMR0 register.

Counter mode is selected by setting bit T0CS (OPTION_REG<5>). In Counter mode, Timer0 will increment either on every rising or falling edge of pin RA4/T0CKI. The incrementing edge is determined by the Timer0 Source Edge Select bit, T0SE (OPTION_REG<4>). Clearing bit T0SE selects the rising edge.

- **Timer1 Module:**

The Timer1 module is a 16-bit timer/counter consisting of two 8-bit registers (TMR1H and TMR1L) which are readable and writable. The TMR1 register pair (TMR1H:TMR1L) increments from 0000h to FFFFh and rolls over to 0000h. The TMR1 interrupt, if enabled,

is generated on overflow which is latched in interrupt flag bit, TMR1IF (PIR1<0>). This interrupt can be enabled/disabled by setting/clearing TMR1 interrupt enable bit, TMR1IE (PIE1<0>).

Timer1 can operate in one of two modes:

- As a Timer
- As a Counter

The operating mode is determined by the clock select bit, TMR1CS (T1CON<1>).

In Timer mode, Timer1 increments every instruction cycle.

In Counter mode, it increments on every rising edge of the external clock input.

Timer1 can be enabled/disabled by setting/clearing control bit, TMR1ON (T1CON<0>).

- **Timer2 Module:**

Timer2 is an 8-bit timer with a prescaler and a postscaler. It can be used as the PWM time base for the PWM mode of the CCP module(s). The TMR2 register is readable and writable and is cleared on any device Reset.

The input clock (FOSC/4) has a prescale option of 1:1, 1:4 or 1:16, selected by control bits

T2CKPS1:T2CKPS0 (T2CON<1:0>).

The Timer2 module has an 8-bit period register, PR2.

Timer2 increments from 00h until it matches PR2 and then resets to 00h on the next increment cycle. PR2 is a readable and writable register. The PR2 register is

Initialized to FFh upon Reset.

The match output of TMR2 goes through a 4-bit postscaler (which gives a 1:1 to 1:16 scaling inclusive) to generate a TMR2 interrupt (latched in flag bit, TMR2IF (PIR1<1>)).

Timer2 can be shut-off by clearing control bit, TMR2ON (T2CON<2>), to minimize power consumption.

2.2.7 Instruction Set Summary

The PIC instruction set is highly orthogonal and is comprised of three basic categories:

- **Byte-oriented** operations
- **Bit-oriented** operations
- **Literal and control** operations

Each instruction is a 14-bit word divided into an **opcode** which specifies the instruction type and one or more **operands** which further specify the operation of the instruction. The format for each category is presented in Figure 2.3.

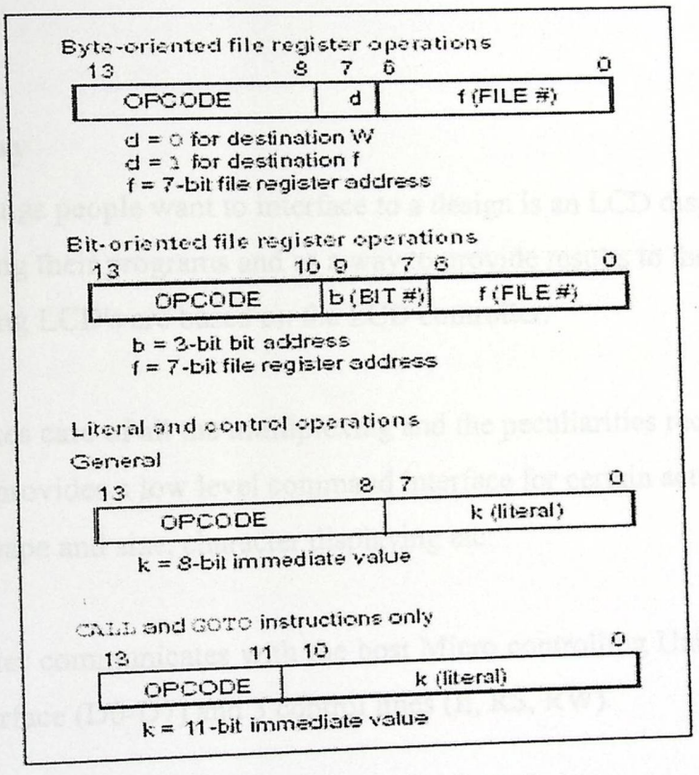


Figure 2-3: Instruction formats

For **byte-oriented** instructions, 'f' represents a file register designator and'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction. The destination designator specifies where the result of the operation is to be placed. If'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For **bit-oriented** instructions, 'b' represents a bit field designator which selects the bit affected by the operation, while 'f' represents the address of the file in which the bit is located.

For **literal and control** operations, 'k' represents an eight or eleven-bit constant or literal value.

One instruction cycle consists of four oscillator periods; for an oscillator frequency of 4 MHz, this gives a normal instruction execution time of 1us. All instructions are executed within a single instruction cycle, unless a conditional test is true, or the program counter is changed as a result of an instruction. When this occurs, the execution takes two instruction cycles with the second cycle executed as a NOP. [5]

2.3 LCD Display

One of the first things people want to interface to a design is an LCD display, both to help with debugging their programs and as a way to provide results to the outside world. Most text displaying LCD's are based on the LCD controller.

This controller takes care of all the multiplexing and the peculiarities required by the LCD display and provides a low level command interface for certain actions like screen clearing, cursor shape and size, character displaying etc.

The LCD controller communicates with the host Micro controlling Unit through an 8 bit bi-directional interface (D0-D7) and 3 control lines (E, RS, RW).

LCD's are slow devices when compared to microcontrollers. Special attention must be paid so the commands and data are not send too quickly from the Micro controlling Unit, and the designer must control the communication speed and timing using latching devices to ensure that the slow LCD and the fast Micro controlling Unit stay synchronized.

The LCD module contains 3 different functional blocks: The Character Generator ROM (CG-ROM), the character generator RAM (CG-RAM) and then Display Data RAM (DD-RAM).

The ROM character generator is factory programmed and contains the 208 characters to be displayed.

The CG-RAM allows the user to define up to 8 special characters that are not included in the CG-ROM.

The DD-RAM holds the characters that are actually displayed in the LCD
Both the CG-RAM and DD-RAM can be read and written (through commands) by the Micro controlling Unit.

The most important of the function blocks is the DD-RAM. This has to do with the way that is implemented and sometimes is different from LCD to LCD depending on how many lines by how many rows the LCD can display the DD-RAM address changes. [6]

Chapter Three

Design Concepts

3.1 Project Objectives

3.2 General Block Diagram

3.3 How System Works

Chapter Three

Design Concepts

3.1 Project Objectives

The system is considered to have many objectives, these objectives are:-

- Driving a stand alone CD-Drive using microcontroller.
- Controlling of a CD-Drive using control buttons. This controlling is described by the following aspects:
    1. Play and stop track.
    2. Increase and decrease the volume.
    3. Selection between different playing tracks.
    4. Retrieving and forwarding track.
    5. Pause on and off.
- Develop an LCD display system connected to the CD-Drive.
- Amplifying the volume out from CD-Drive .

3.2 General Block Diagram

The following block diagram defines the general system components:

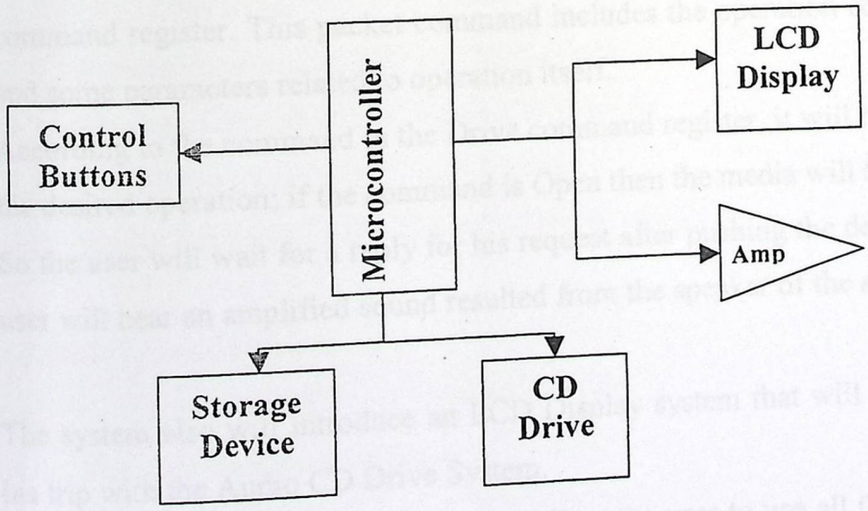


Figure 3-1: General Block Diagram

Chapter Three

Design Concepts

3.1 Project Objectives

The system is considered to have many objectives, these objectives are:-

- Driving a stand alone CD-Drive using microcontroller.
- Controlling of a CD-Drive using control buttons. This controlling is described by the following aspects:
    1. Play and stop track.
    2. Increase and decrease the volume.
    3. Selection between different playing tracks.
    4. Retrieving and forwarding track.
    5. Pause on and off.
- Develop an LCD display system connected to the CD-Drive.
- Amplifying the volume out from CD-Drive .

3.2 General Block Diagram

The following block diagram defines the general system components:

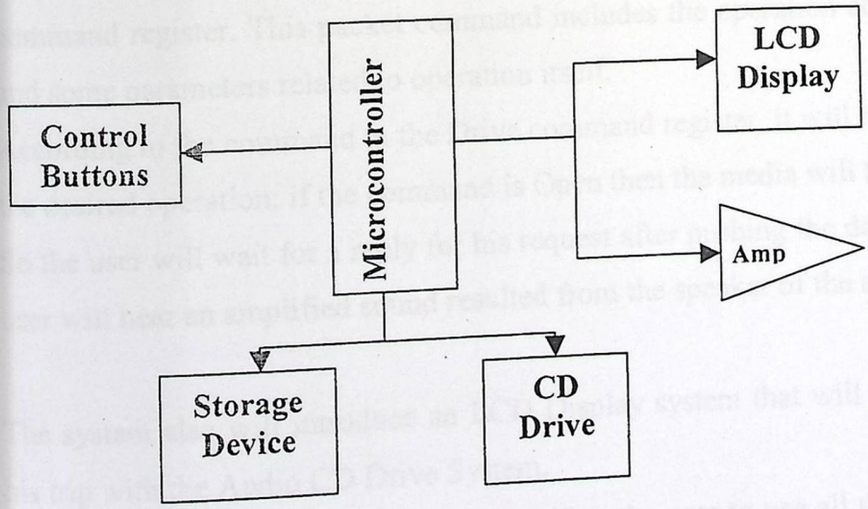


Figure 3-1: General Block Diagram

3.3 How System Works

The CD-Drive is the basic component in our system, that it will be driven as an audio CD without the computer in order to act as a strand alone system.
The system will use the microcontroller for this purpose and because the CD-Drive needs to be connected to a display system and switches. So all the components that will be used is connected to this microcontroller.

The user initiate the system using the power button, the PIC microcontroller will be initialized so that the ports will be configured, the registers will be zeroed and the analog to digital converter will be set.
Then the PIC microcontroller will initialize the CD drive by sending the 12 byte command packet, and make a reset to the CD.
The PIC microcontroller will poll the state of the port which is connected to the push buttons, in order to determine which function should be done.
This will be performed by comparing the result of the analog to digital converter "which is connected to voltage divider circuit" by a stored value that indicate a specific operation.

When the PIC microcontroller determines which Key is pressed, it will execute the subroutine that corresponds to that operation.
Each subroutine will be executed by sending a special packet command to the CD Drive command register. This packet command includes the operation code for the command and some parameters related to operation itself.
According to the command in the Drive command register, it will reply for, and perform the desired operation; if the command is Open then the media will be ejected, and so on.
So the user will wait for a reply for his request after pushing the desired button. And the user will hear an amplified sound resulted from the speaker of the amplifier circuit.

The system also will introduce an LCD Display system that will guide the user during his trip with the Audio CD Drive System.
In addition to that, the system will also allow the user to use all the existing features in the CD-Drive such as, using headphones...

Chapter Four

Hardware System Design

Chapter Four
Hardware System Design

This chapter describes the main hardware components used during building the system. It represents how these components are integrated to each other. A block diagrams and general schematic diagrams will describe this system.

The main hardware components in the system are:

1. CDROM-IDE 40-Pin Connector
2. PIC Microcontroller
3. LCD Display
4. Amplifier
5. Input Switches
6. Cables and Connections

4.1 Design Options

4.1.1 The microcontroller

There are many microcontrollers that have the required construction and operations to fulfill the system needs.

1. PIC microcontroller
2. 8051 Chip

PIC Microcontroller

The option of choosing the PIC Microcontroller refers to its existence in our city. Also, because PIC Microcontroller owns the required IO ports and some internal chips that the system needs, without any external circuits to add.

To view the features of the microcontroller see page 34.

8051 Microcontroller

It can support this project, but there is a need to add additional chips, such as analog to digital converter, also mapping is needed …

The component of typical 8051

- CPU with Boolean processor
- 5 or 6 interrupts: 2 are external, 2 priority levels
- 2 or 3 16-bit timer/counters
- Programmable full-duplex serial port (baud rate provided by one of the timers)
- 32 I/O lines (four 8-bit ports)
- RAM
- ROM/EPROM in some models

Stand alone CD drive need a microcontroller with large number of IO ports, in order to connect the IDE connector, LCD and switches to control the CD drive.

Also internal memory is needed, and other common microcontroller features.

PIC microcontroller is chosen because it provides much more features that do not exist in 8051 microcontroller, the most important one is the large number of IO ports, and this will allow connecting the LCD display with one of the input ports.

In addition, the PIC microcontroller provide analog to digital converter that is important in this project, and the feature of low power consuming, encourage any designer to choose it.

4.1.2 The Programmer

In order to burn our software program into the PIC microcontroller, there is a need of a hardware programmer that will install the hex file program into the PIC.
The programmer options are:

4.1.2.1 Companies Programmers product

The microchip company" The producer of the PIC" offers many suitable programmers Such as Microchip PIC-Start which runs about $200.
Or programmers from Australian company "Newfoundland" that makes a compatible programmer that does everything the Microchip programmer does and then some for $100. The advantage to this sort of programmer is that you can use it directly with MPLAB.

4.1.2.2 In-Circuit Serial Programming

When this circuit has been built, the programmer is needed once to set up the PIC with a small booter called boot loader, and all other programming will be via this circuit.
A serial port communication will be used with the PC, and small software that will download the hex file program to the PIC.

The circuit uses a MAX232 5V to RS232 converter chip, this converts the 0-5V TTL levels at the PIC pins to the +12V/-12V levels used in RS232 links. As is common with these devices it inverts the data during the conversion, the PIC USART hardware is designed to take account of this - but for software serial communications you need to make sure that you invert both the incoming and outgoing data bits.

The two closed links on the RC7 and RC6 lines are for connection to the 16F876 board (the 16F876 uses RC6 and RC7 for its USART connection), and are the two top wire links shown on the top view of the board below. The two open links on the RC1 and RC2 lines are for the 16F628 board (the 16F628 uses RB1 and RB2 for it's USART connection.

The used software is called downloader that downloads in few seconds the desired program.

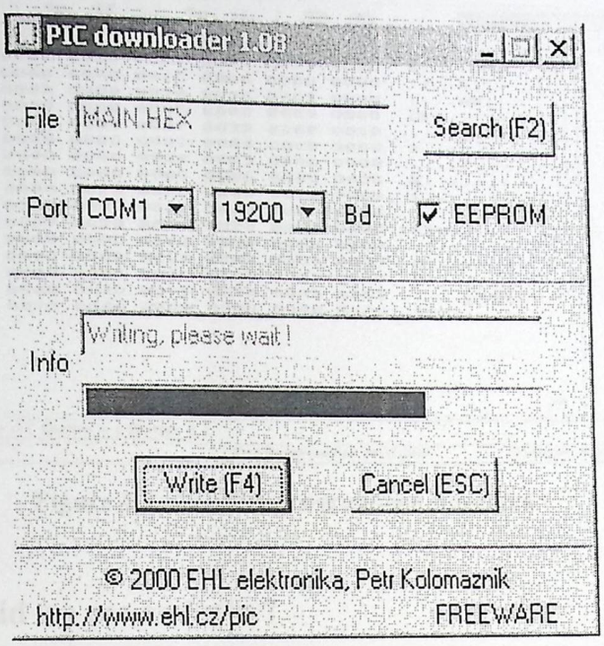


Figure 4-1: Serial programmer Interface

4.1.1.3 Parallel Port Programmer

This programmer can be easily built with a few components, and can be used to program the PIC without the need of the programmer at all!!

Special software is needed only; this software is Windows based software to control a development programmer for PIC microcontrollers. In order for this software to operate we have to attach a programmer to the computer and set up the hardware & software appropriately.

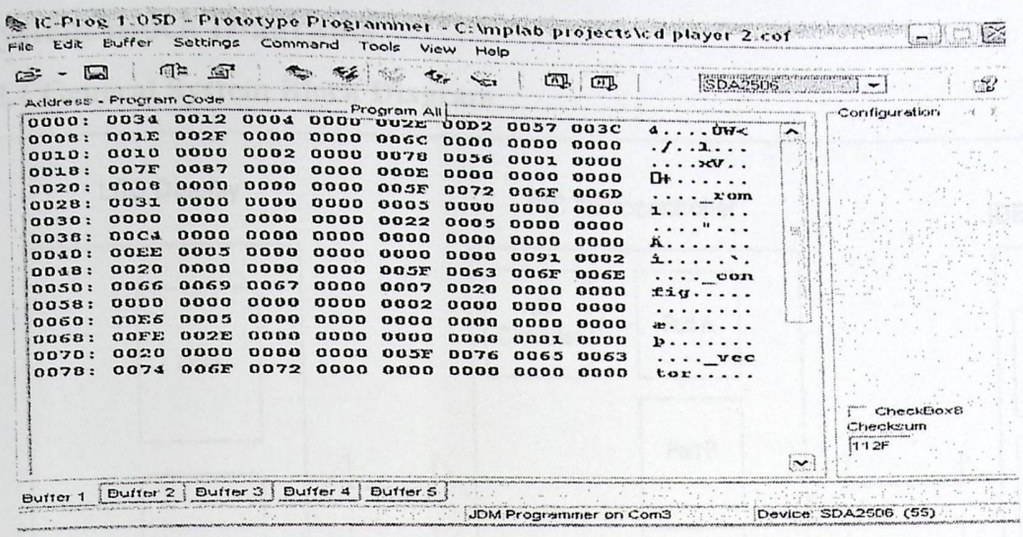


Figure 4-2: Parallel programmer Interface

Which programmer did we use and why?

1. We use parallel port programmer, because:

- ✱ Easy to build
- ✱ Do not need the programmer at all
- ✱ Has free software
- ✱ Support many version of PIC microcontroller
- ✱ Allow to read, program, verify and erase the PIC microcontroller

2. We try to use the In Circuit Serial Programming, but some problems faces us because:

- ✱ It needs a programmer once, to burn the boot loader into the PIC
- ✱ The boot loader does not burned at the desired address

3. The choice of company produced programmer is abandoned because it is not available in our city.

4.2 System Block Diagrams

4.2.1 General System Block Diagram

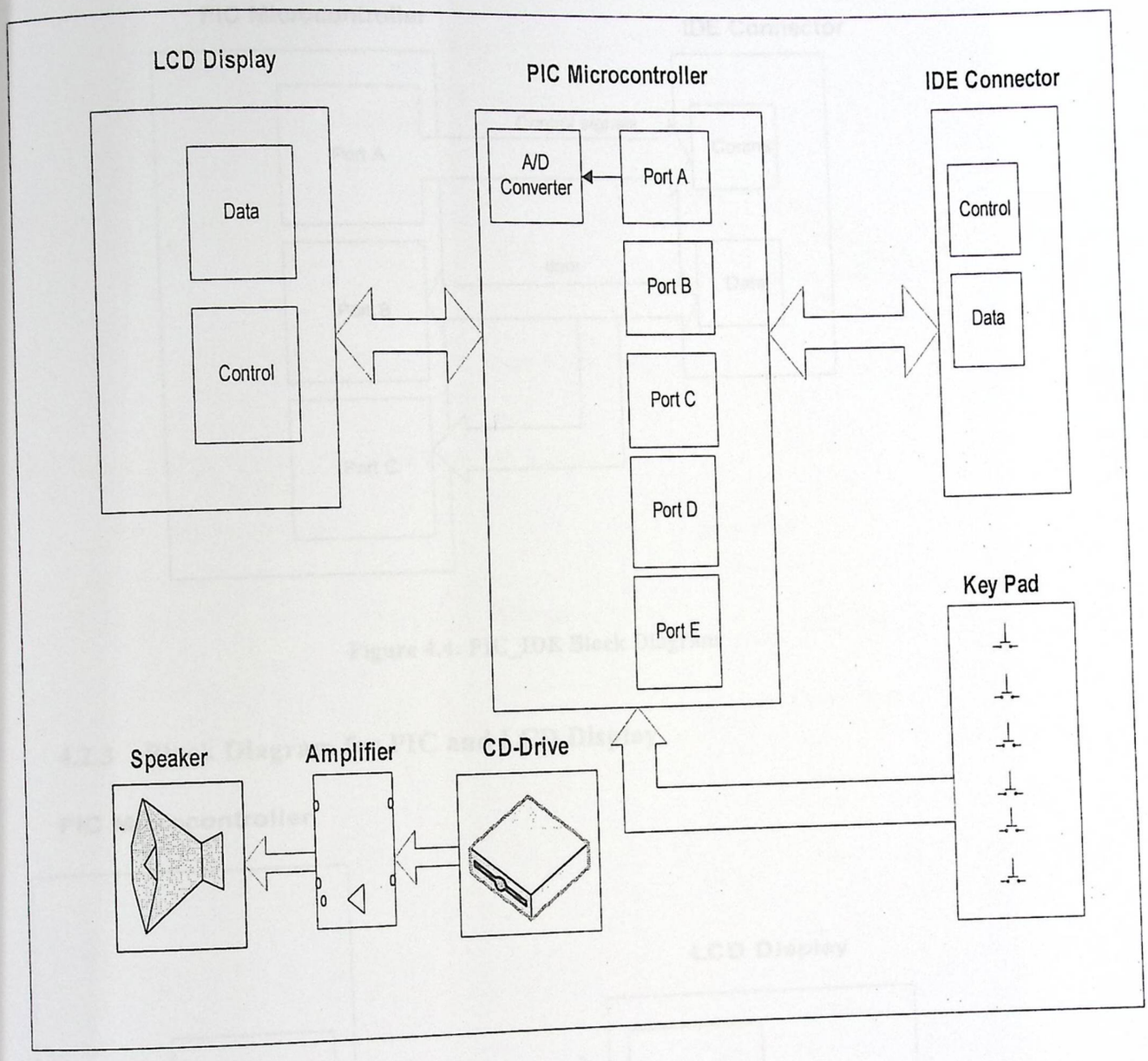


Figure 4.3: System Block Diagram

4.2.2 Block Diagram for PIC and IDE Connector

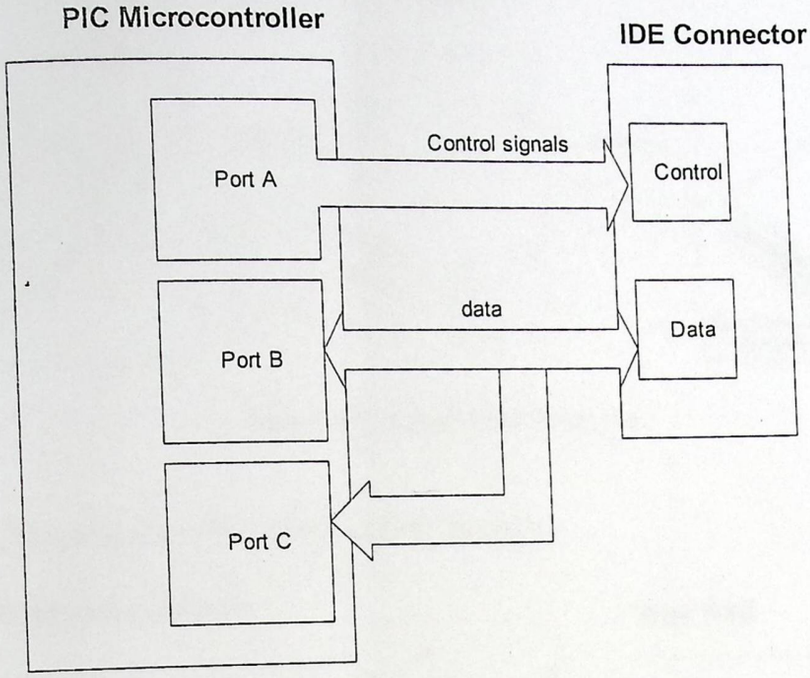


Figure 4.4: PIC_IDE Block Diagram

4.2.3 Block Diagram for PIC and LCD Display

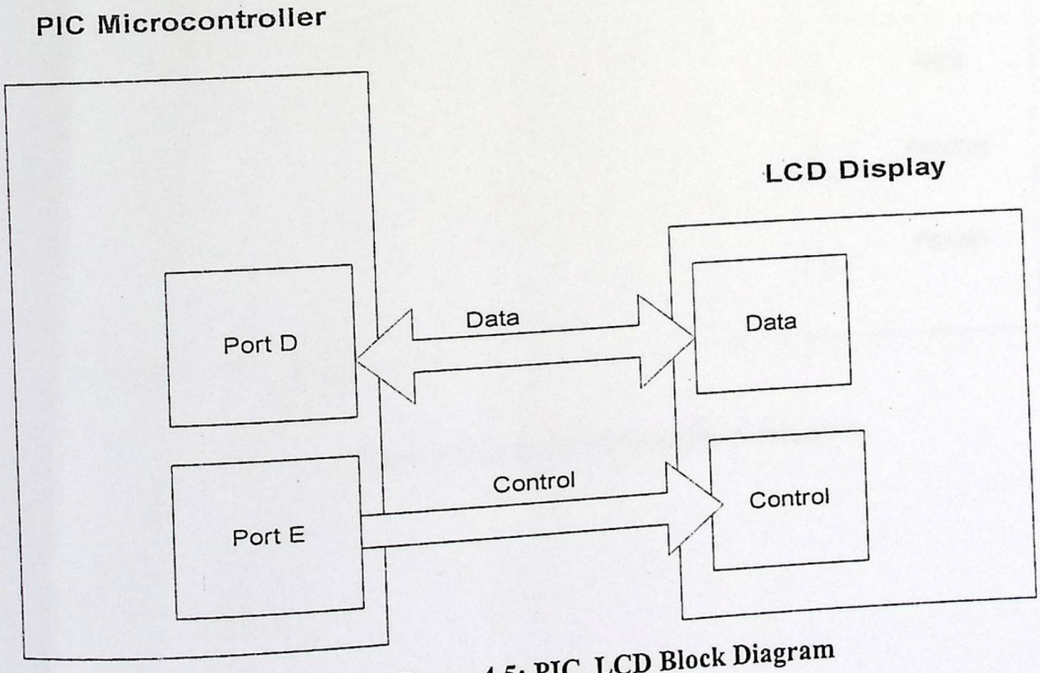


Figure 4.5: PIC_LCD Block Diagram

4.2.4 Block Diagram for Audio Connection

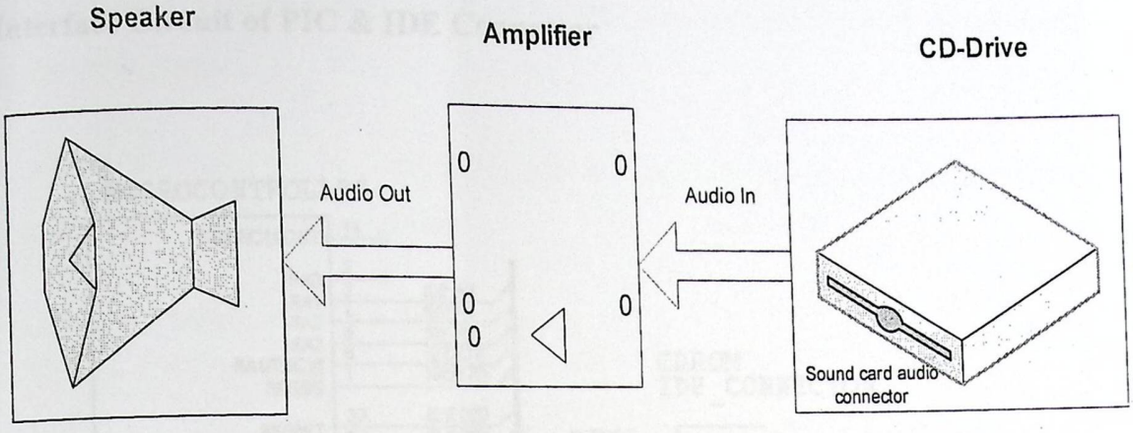


Figure 4.6: Audio Block Diagram

4.2.5 Block Diagram for PIC and Control Buttons

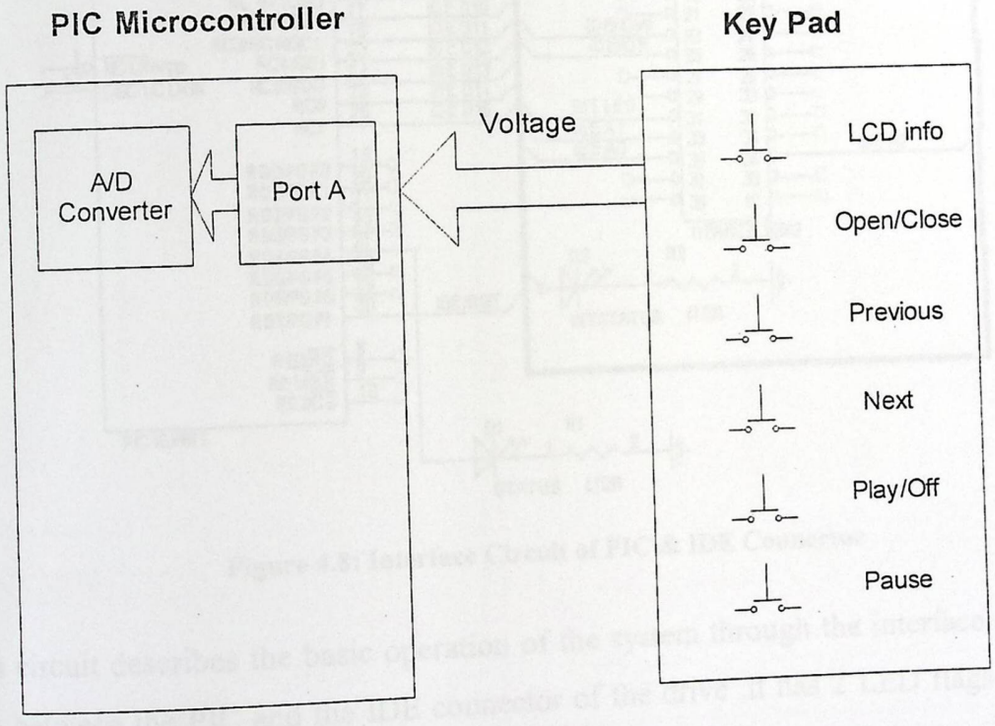


Figure 4.7: Control Buttons Block Diagram

4.3 System Circuits

4.3.1 Interface Circuit of PIC & IDE Connector

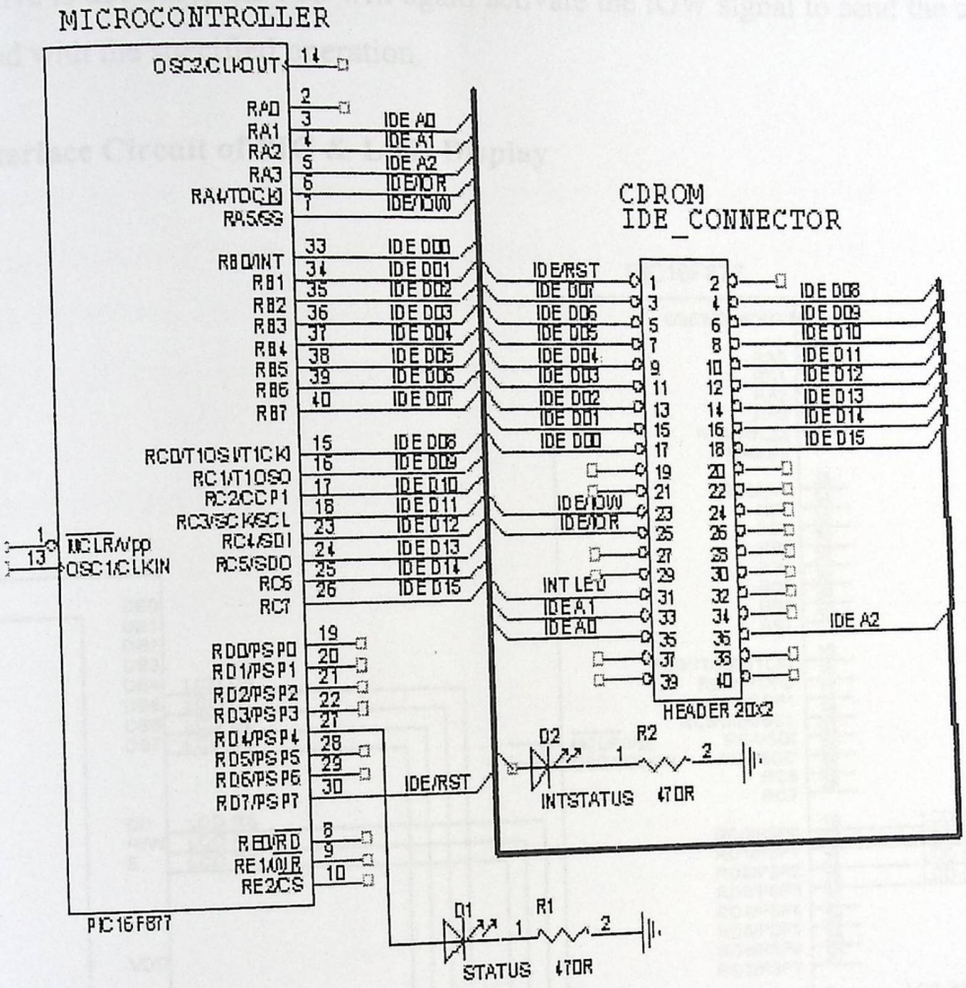


Figure 4.8: Interface Circuit of PIC & IDE Connector

This circuit describes the basic operation of the system through the interface signals and data between the PIC and the IDE connector of the drive .It has 2 LED flags that tell us status of the PIC and the drive.

As its clear, port A of PIC is used for addresses and read or write signals transferred to the drive. Port B and port C are used for data transfer.

The microcontroller will activate the IOW signal and send a CHECK STATUS command packet via the data bus, and then the PIC will activate the IOR signal in order to read the drive status.

If the drive is not busy, the PIC will again activate the IOW signal to send the command associated with the specified operation.

4.3.2 Interface Circuit of PIC & LCD Display

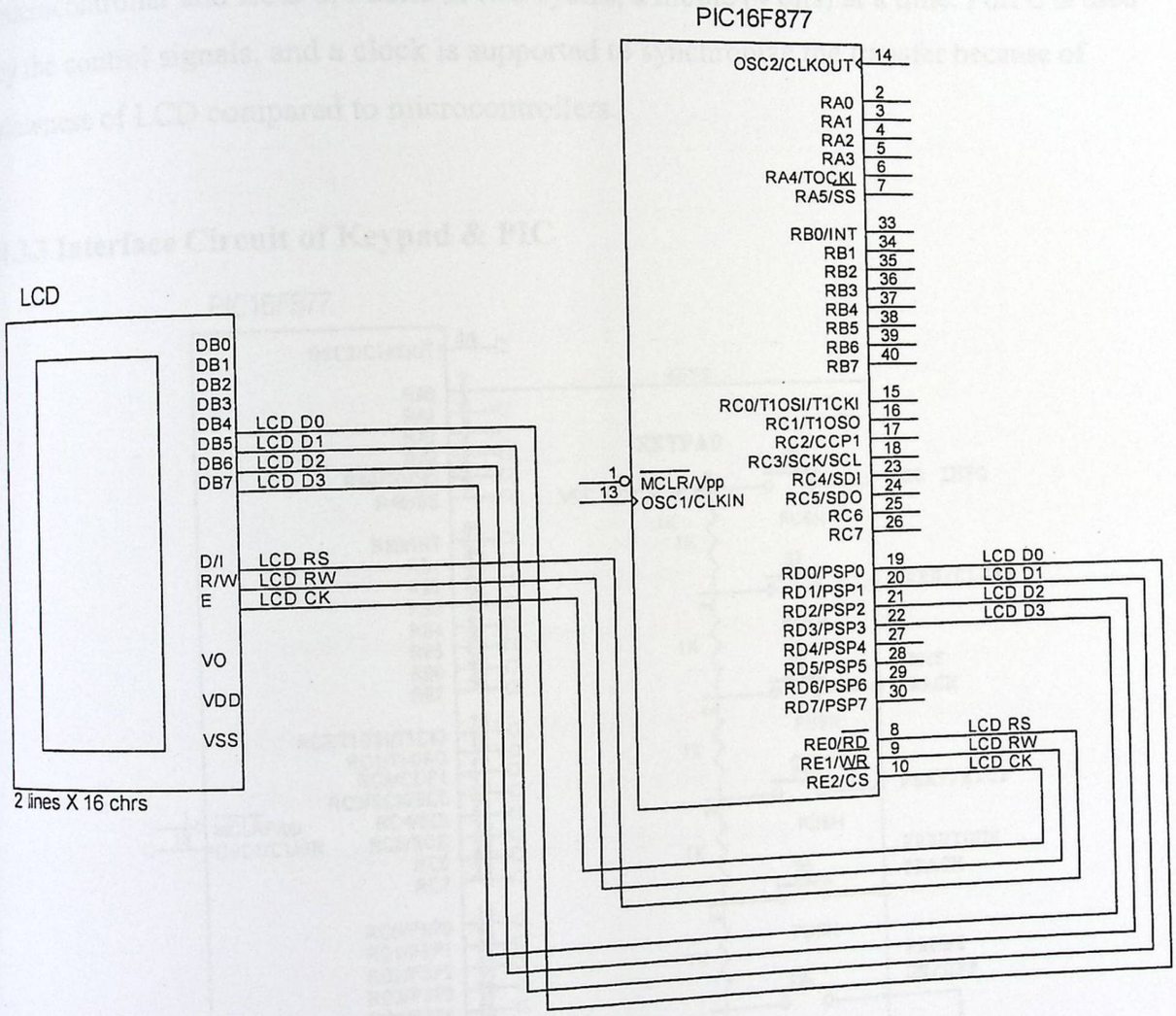


Figure 4.9: Interface Circuit of PIC & LCD Display

The microcontroller will activate the IOW signal and send a CHECK STATUS command packet via the data bus, and then the PIC will activate the IOR signal in order to read the drive status.

If the drive is not busy, the PIC will again activate the IOW signal to send the command associated with the specified operation.

4.3.2 Interface Circuit of PIC & LCD Display

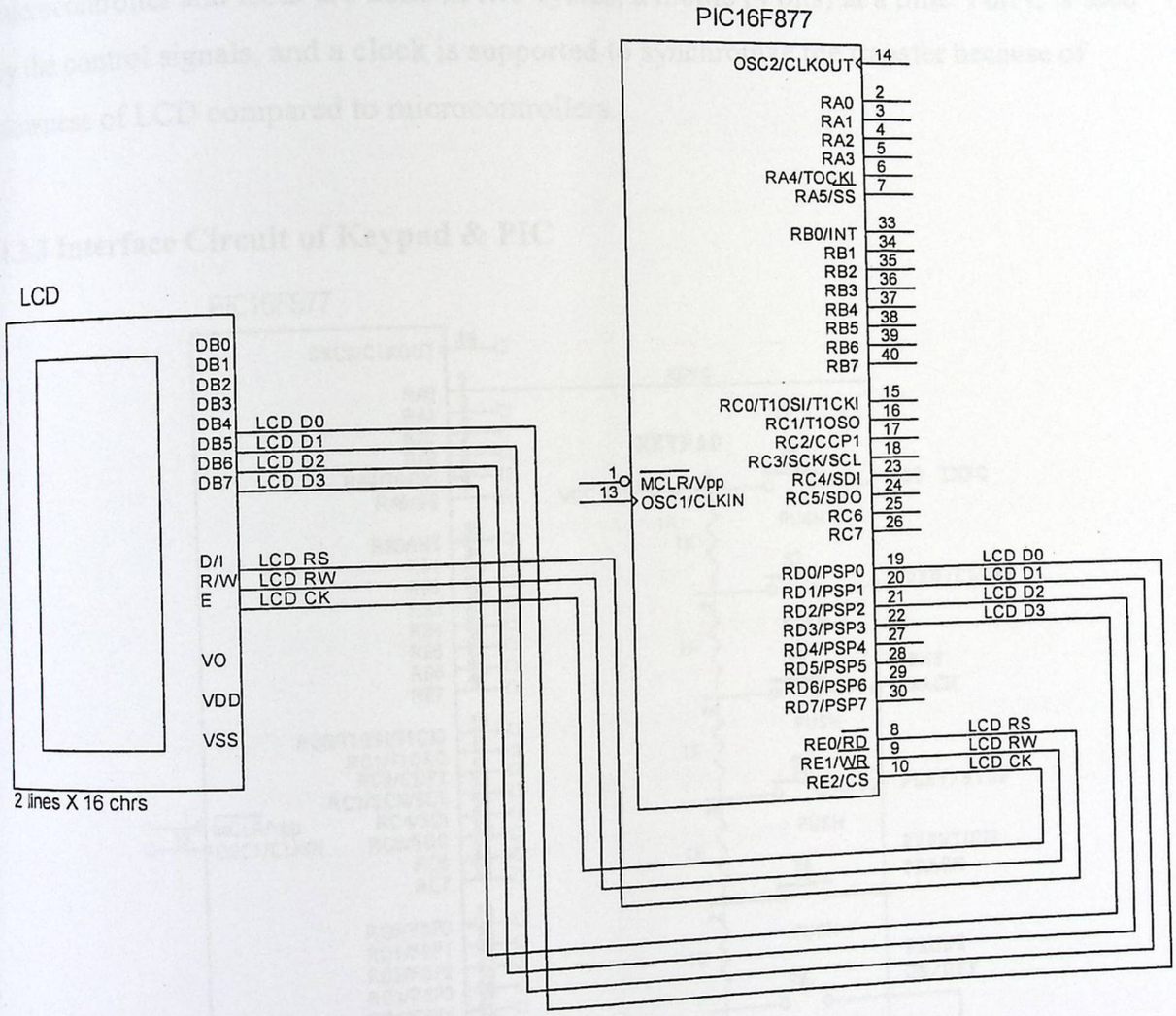


Figure 4.9: Interface Circuit of PIC & LCD Display

Here is the interface between the microcontroller and the display unit which is LCD. The tracks information that is read from the drive, are passed by PIC to be displayed on the LCD.

The LCD communicates with the microcontroller through an 4 bit bi-directional interface (D4-D7) ,taking into account that the system designer is "low" on I/O lines and 3 control lines (E, RS, RW).

Data lines use the low order lines of port D and data transfers between the microcontroller and LCD are done in two cycles, a nibble (4 bits) at a time. Port E is used by the control signals, and a clock is supported to synchronize the transfer because of slowness of LCD compared to microcontrollers.

4.3.3 Interface Circuit of Keypad & PIC

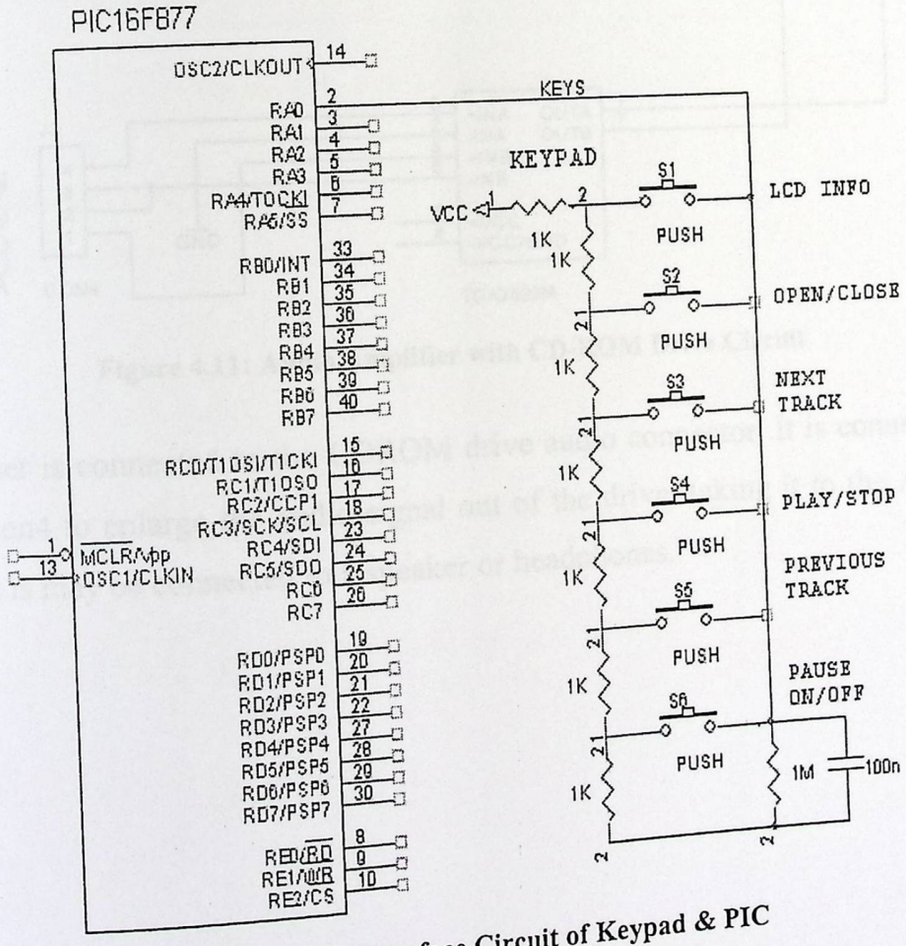


Figure 4.10: Interface Circuit of Keypad & PIC

The keypad is represented by a voltage divider structure, where each control button represents a specific value of voltage to define its task to the microcontroller.

As this voltage is analog, the A/D converter in the PIC translates the voltage to the desired operation.

The provided control buttons are to start and stop the play of the CD, pause, and display info. And also to view next track and previous track.

4.3.4 Audio Amplifier with CD-ROM Drive Circuit

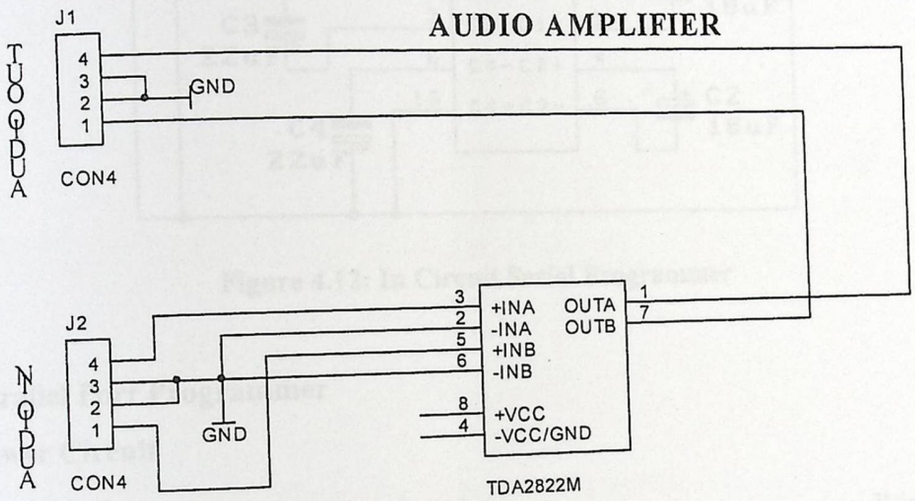


Figure 4.11: Audio Amplifier with CD-ROM Drive Circuit

The amplifier is connected to the CDROM drive audio connector .It is connected to the Audio IN con4 to enlarge the audio signal out of the drive, taking it to the Audio OUT con4 which is may be connected to a speaker or headphones.

4.4 Programmer Circuits

4.4.1 In Circuit Serial Programming Circuit

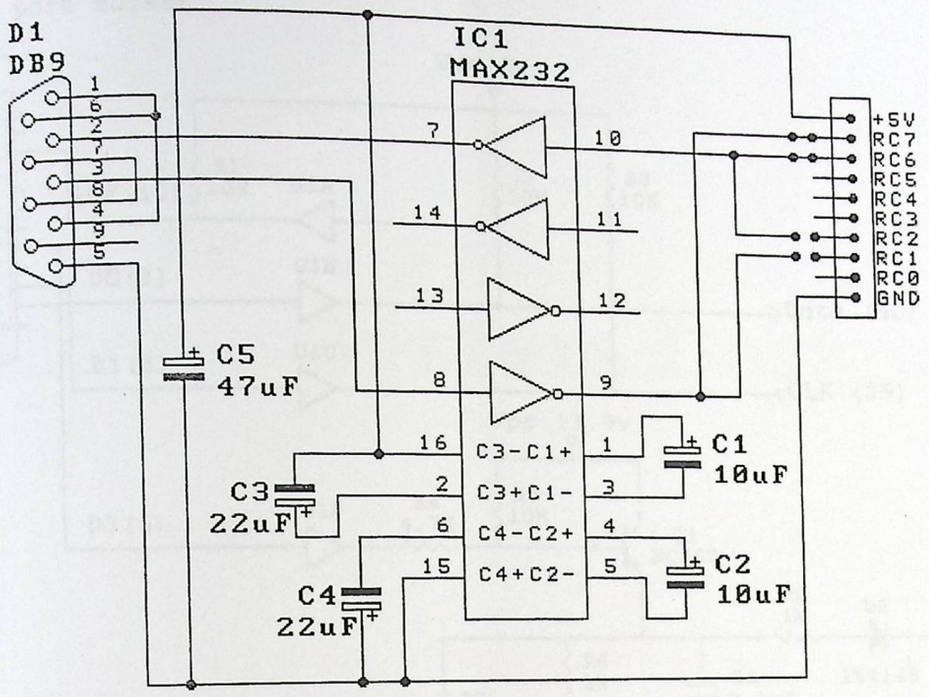


Figure 4.12: In Circuit Serial Programmer

4.4.2 Parallel Port Programmer

- Power Circuit

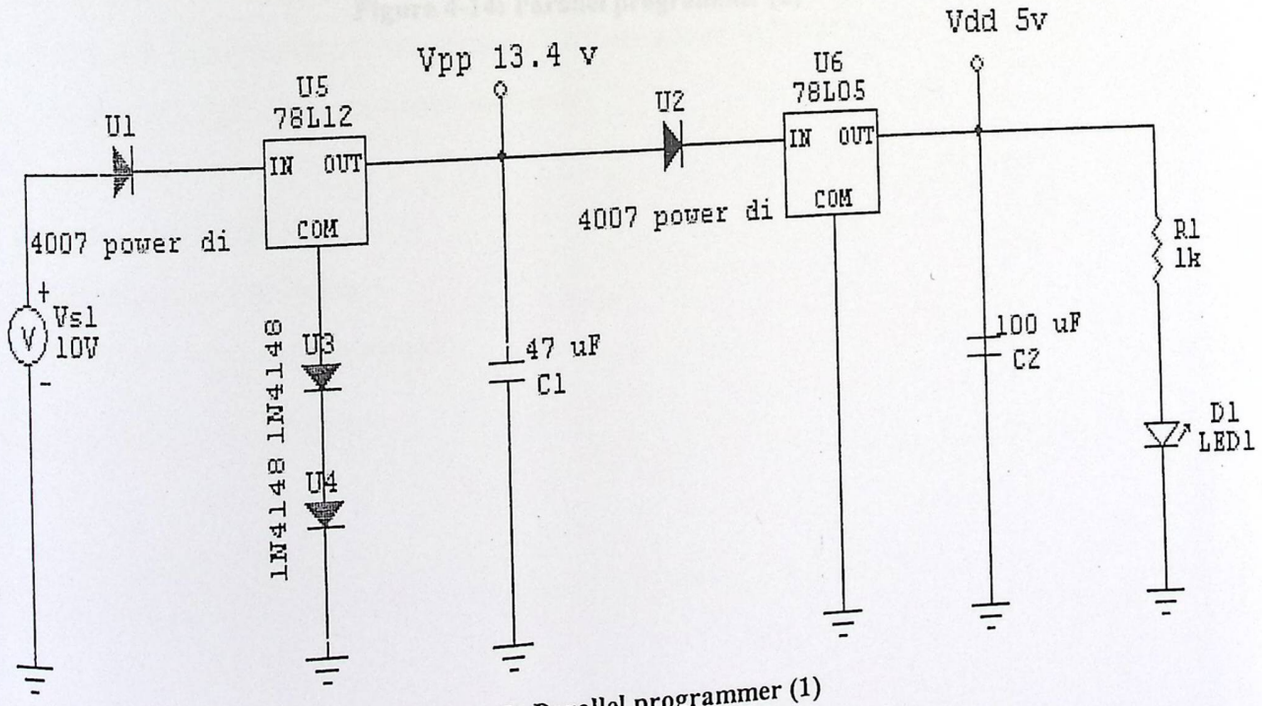


Figure 4-13: Parallel programmer (1)

- Programmer Circuit

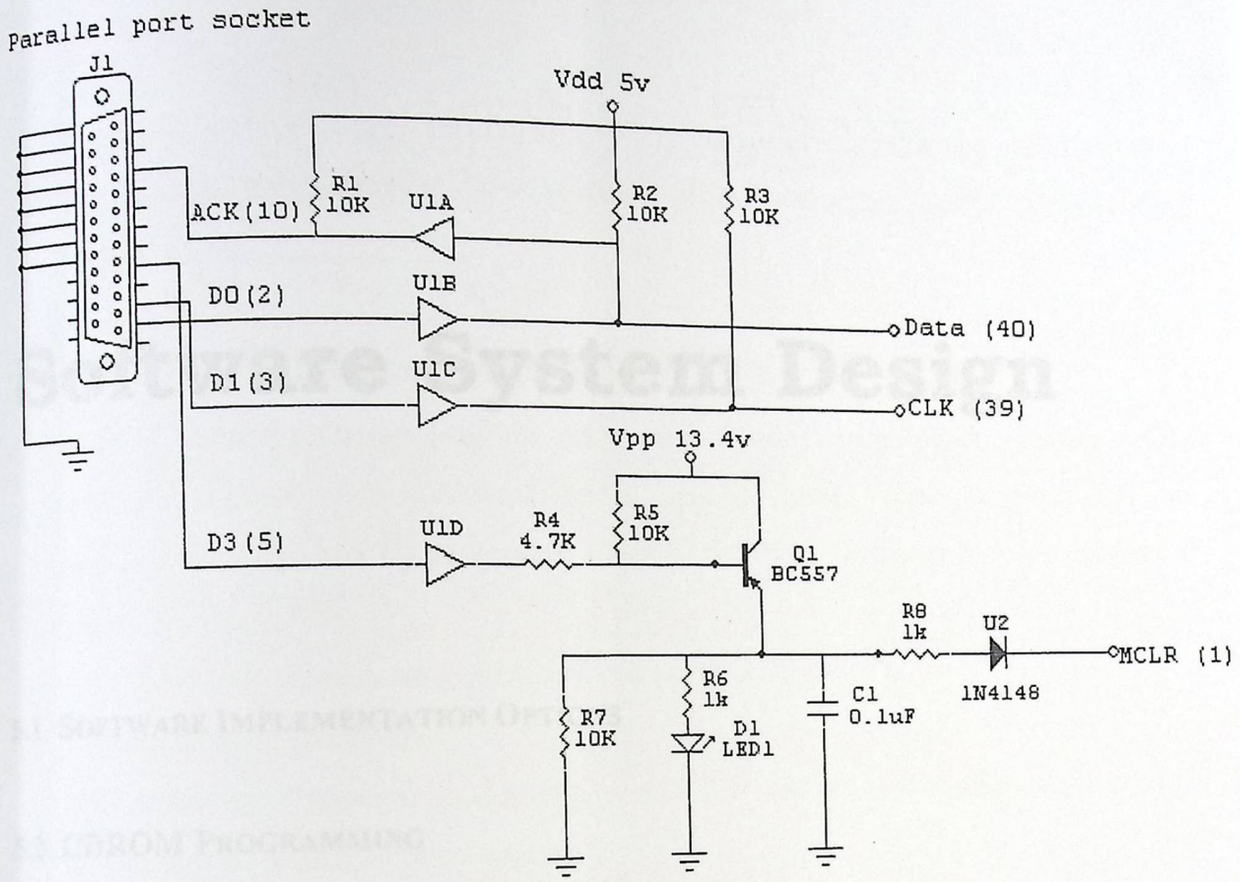


Figure 4-14: Parallel programmer (2)

Chapter Five

Software System Design

5.1 Software Implementation Options

5.2 CDROM Programming

5.3 Software Tools

5.3.1 MPLAB Integrated Development Environment Software

5.3.2 IC-PROG PIC Programmer Software

5.4 General System Diagram

5.4.1 Key Update Flowchart

5.4.2 CDROM Update Flowchart

Chapter Five
Software System Design

This chapter includes implementing the system from the programming point of view. Software System Implementation depends on CD-ROM programming, PIC programming and software tools.

5.1 Software Implementation Options

The general software implementation may seem limited at a glance. Rather, when scrutinize the available ways, some different options appear. There is more than one software approach that can be adopted in order to fulfill the software implementation of the system.

There are two programming languages that can support the system:

1. C programming Language
2. PIC Assembly Programming Language, which is used in our application because it is easy to deal with CD Drive registers.

5.2 CDROM Programming

ATAPI (ATA Packet Interface) is an extension to ATA which essentially allows SCSI commands (commands used to control SCSI devices) to be sent to ATA devices. ATAPI is used specifically for CD-ROM drives, which, when they first started appearing for computers, were almost universally SCSI. Because SCSI controllers were expensive, the SCSI command set was eventually adopted for IDE, and typical CD-ROM drives today use ATAPI. ATAPI basically uses "packets" (similar to the packet concept of computer networking as it applies to TCP/IP, for example) to send and receive data and commands. Properly speaking, ATAPI is part of the EIDE (Enhanced IDE) standard.

A packet sent to an ATAPI device which contains a command is called a **command packet**. These command packets are written to the data register via the ATA interface, and that's how ATAPI devices receive their commands. The command packet has a 12-byte standard format, and the first byte of the command packet contains the actual operation code. (The remaining 11 bytes supply parameter info for the command.) Note that although the command packet is 12 bytes long, the packet is sent to the ATAPI device through word writes, not byte writes. A "word" in PC assembly language is 2 bytes, so you'll actually send the 12-byte command packet in only 6 write operations.

The "operation code" value you place in the ATAPI command packet is actually a SCSI command code. You do not use ATA commands with ATAPI devices; ATAPI devices use SCSI commands. For example, the SCSI command to eject a CD-ROM drive tray is the "START/STOP UNIT" command, which is SCSI command 1Bh. Similarly, to get an ATAPI CD-ROM drive to eject, you'd send it a command packet with 1Bh for an operation code.

ATAPI contains several commands, but the most fundamental of these is the PACKET command, which has an ATAPI opcode of A0h. The first step in sending a command to an ATAPI device is to send it the PACKET command over the regular ATA command register, just as the program above sends the IDENTIFY DEVICE command. Once this PACKET command is sent, the ATAPI interface goes into a condition called **HPD0: Check_Status_A State**, which means that the ATA controller is to wait for 400 nanoseconds, and then poll the status register until the BSY bit is zero. If the BSY bit is one, the host is supposed to keep polling the status register until the BSY bit clears.

Once the BSY bit does clear (and DRQ is set to one), the ATAPI interface changes to **HPD1: Send_Packet State**. In this state, the host is supposed to send the command packet to the ATA controller's data register, one byte at a time. When all the bytes of the command packet have been sent, the host is to either transition to HPD2: Check_Status_B State (if nIEN is set to one), or to HPD3: INTRQ_Wait State (if nIEN is set to zero). Note that nIEN is the second-rightmost bit of the ATA Device Control

Register. You will note that this register is write-only, so you should write to this register to set nIEN before you begin sending the ATAPI packet.

If the host does transition to HPD3: INTRQ_Wait State, all it's supposed to do is wait for INTRQ to be asserted. When INTRQ is asserted, then the host shall transition to HPD2: Check_Status_B State.

The HPD2: Check_Status_B State is where things get a little hairy. This is where you check the status register, but there are a lot of condition bits you're supposed to check. First of all, the ATAPI spec specifies that "When entering this state from the HP1 ... state, the host shall wait one PIO transfer cycle time before reading the Status register. The wait may be accomplished by reading the Alternate Status register and ignoring the result."

Once that's done, start checking the status register. First of all, if BUSY is set to 1, the host is not to leave the HPD2 state. The host is supposed to remain in HPD2 until BUSY clears to zero.

Once BUSY is zero, check DRQ. If DRQ is set to one, then the host shall transition to yet another state, called the HPD4: Transfer_Data State. However, the only time you'd need to enter HPD4 is if DRQ is 1 during the HPD2 State. If DRQ is zero now, you can skip HPD4 altogether.

If both BUSY and DRQ are zero, then, the command is probably complete. Technically, there are a few other things you're supposed to check, but we won't worry about those now. [7]

5.3 Software Tools

5.3.1 MPLAB Integrated Development Environment Software

The MPLAB IDE software brings an ease of software development previously unseen in the 8/16-bit microcontroller market. The MPLAB IDE is a Windows® based application that contains:

• An interface to debugging tools

- Simulator

- Programmer (sold separately)

- Emulator (sold separately)

- In-circuit debugger (sold separately)

• A full-featured editor with color coded context

• A multiple project manager

• Customizable data windows with direct edit of contents

• High level source code debugging

• Extensive on-line help

The MPLAB IDE allows you to:

• Edit your source files (either assembly or C)

• One touch assemble (or compile) and download to PICmicro emulator and simulator tools

• Debug using:

- Source files (assembly or C)

- Absolute listing file (mixed assembly and C)

- Machine code

5.3.2 IC-PROG PIC Programmer Software

IC-Prog is Windows based software to control a development programmer for PIC microcontrollers. To operate this software, a basic knowledge about electronics and Windows is necessary.

In order for this software to operate the programmer must be attached to the computer and set up the hardware & software appropriately.

IC-Prog requires Windows 95, 98, ME, NT, or 2000 and an internal or external math coprocessor to operate. IC-Prog has been designed as a universal programming application for all programmers.

5.4 General System Diagram

The following diagram describes the sequence in which system processes are executed The system is initialized by configuring the PIC I/O ports , ADC registers , CDROM read/write signals , zeroing the variables that keep triggered events and states by control buttons and initializing the CDROM to read the first track . The CDROM is reset, a standard 12byte block of data which is a command packet is written and the CDROM registers are initialized. The analog input port of the microcontroller is scanned and compared with specified values in order to determine which key is pressed and execute the corresponding subroutine.

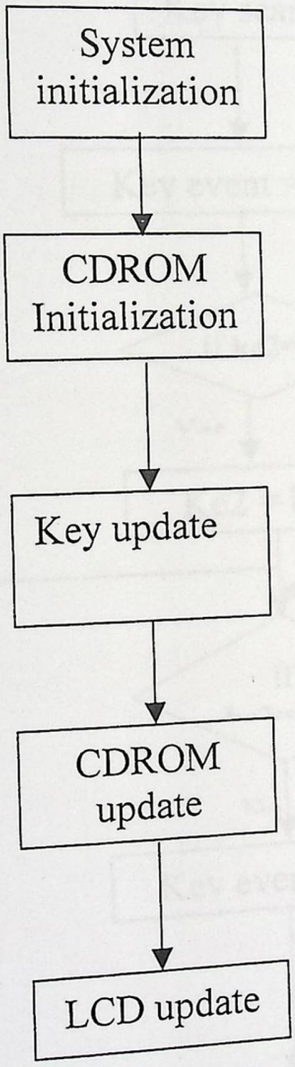


Figure 5-1: System flowchart

5.4.1 Key Update Flowchart

When a key is pushed (updated), a key scan process is done to specify the next event by checking the voltage on the port and beyond it, the event corresponding to the key voltage is stored in a variable (ke1). To perform the specified event , the new event - stored in ke1- is compared with the last one –stored in ke2- so that its obvious what is the next step(change or not) .

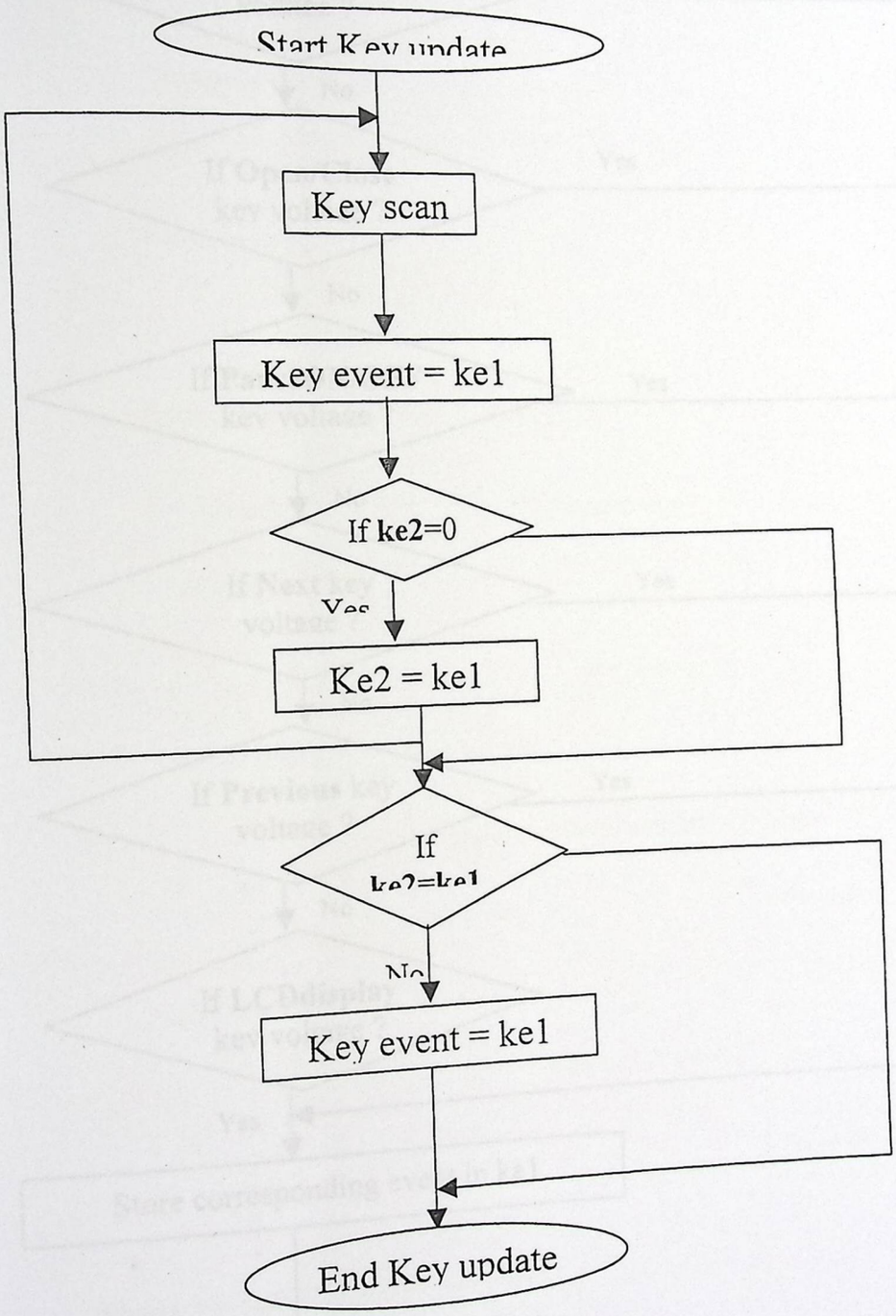


Figure 5-2: Key update flowchart

Key Scan Flowchart:

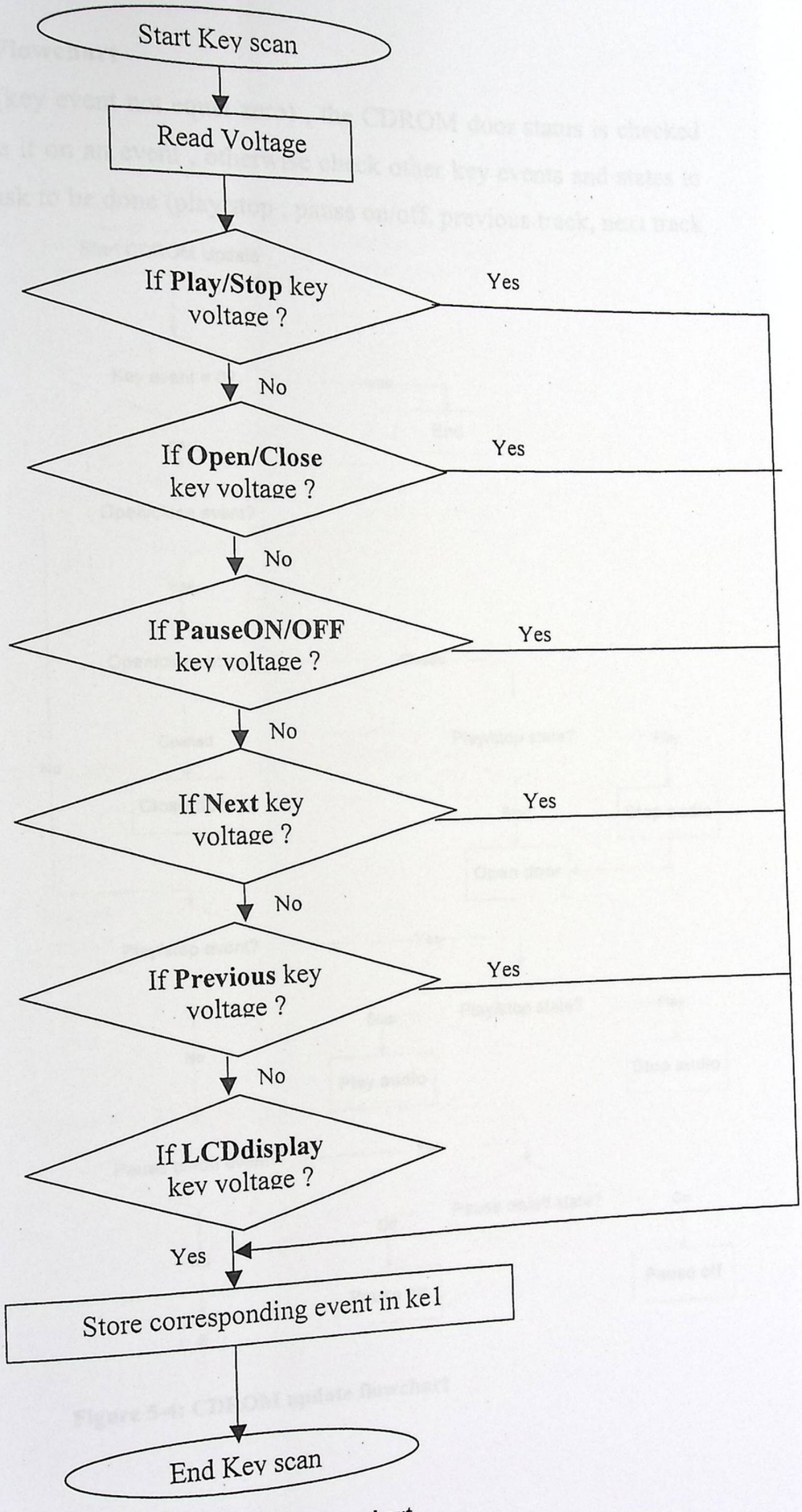


Figure 5-3: Key scan flowchart

5.4.2 CDROM Update Flowchart

If an event is occurred (key event not equal zero) , the CDROM door status is checked whether its open to close it on an event , otherwise check other key events and states to determine the required task to be done (play/stop , pause on/off, previous track, next track or display LCD info.) .

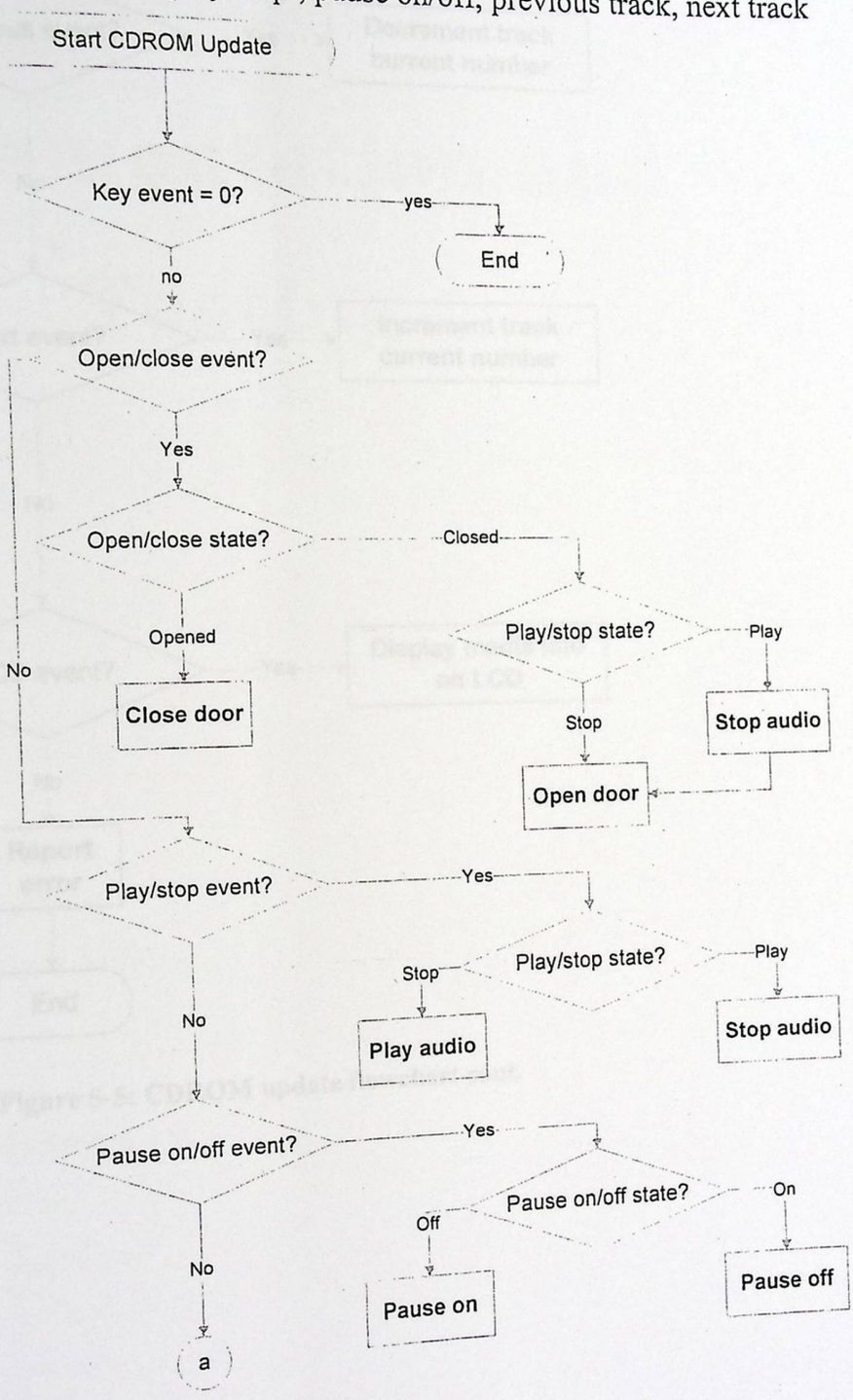


Figure 5-4: CDROM update flowchart

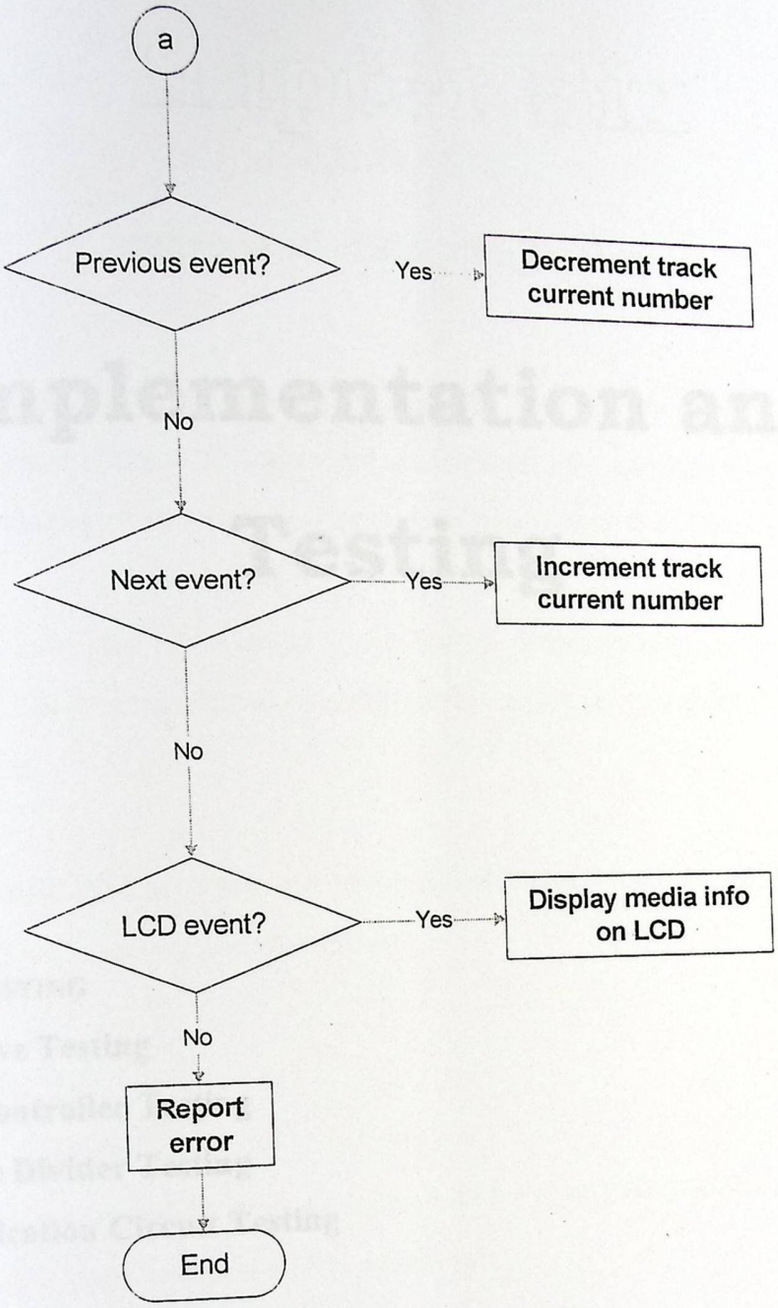


Figure 5-5: CDROM update flowchart cont.

Chapter Six

Implementation and Testing

Chapter Six

Implementation and Testing

This chapter demonstrates the procedures used to test and examine the basic system subroutines. System testing is an important and crucial step in implementing a system. It senses the effectiveness of that system just before introducing it to its users.

This system has more than one issue to be tested. Some testing parts reflect software, hardware. Also, testing procedures concentrate on a single device independent from the over whole system.

Here are the testing issues. They are not ordered in any manner; rather they represent some way of system integrity and operation:

6.1 Initial Testing

6.1.1 CD Drive Testing

The CD Drive is examined to play an audio CD by connecting a DC12V power supply, and connecting the audio out port to the speakers.

6.1.2 PIC Microcontroller Testing

Some experiments are performed to verify the microcontroller operation, such as:

- **Flashing LED Experiment**

A 4MHZ crystal is connected between pin 13 &14, and a led anode is connected to the port pin RC4, the led cathode with 1K resistor and to the ground.

Here is the experiment code:

```
;       list P = 16F877
;       include "P16f877.inc"  ; use definition file for 16F877
```

```
      USER RAM DEFINITIONS
;     CBLOCK 0x20    ; RAM starts at address 20h

NaHi
NaLo
NbHi
NbLo
      ENDC

;     org 0x0000       ; start address = 0000h

; INITIALISE PORTS
; binary used to see individual pin level

      movlw b'00000000'        ; all port pins = low
      movwf PORTA
      movlw b'00000000'
      movwf PORTB
      movlw b'00000000'
      movwf PORTC
      movlw b'00000000'
      movwf PORTD
      movlw b'00000000'
      movwf PORTE

      bsf STATUS,RP0   ; set RAM Page 1 for TRIS registers

; INITIALISE PORTS
; binary used to see individual pin IO status

      movlw b'00000000'        ; all IO pins = outputs
      movwf TRISA
      movlw b'00000000'
      movwf TRISB
      movlw b'00000000'
      movwf TRISC
      movlw b'00000000'
      movwf TRISD
      movlw b'00000000'
      movwf TRISE

      movlw b'00000110'        ; all analog pins = digital
      movwf ADCON1

      bcf STATUS,RP0   ; back to RAM page 0

; LED FLASH LOOP

Loop  bsf PORTC,4      ; RC4 = high = led on
      call Delay

      bcf PORTC,4      ; RC4 = low = led off
      call Delay
      goto Loop

; 1/2 SEC DELAY SUBROUTINE WITH 4MHz CLOCK

Delay movlw 01h
      movwf NbHi
      movlw 03h
      movwf NbLo
      movlw 8Ah
      movwf NaHi
      movlw 5Bh
```

```
        movwf NaLo

DeLoop0 decfsz NaLo,F
        goto DeLoop0
        decfsz NaHi,F
        goto DeLoop0
        decfsz NbLo,F
        goto DeLoop0
        decfsz NbHi,F
        goto DeLoop0
        ;
        return

        end
```

- **PIC Analog to Digital Converter Module Experiment**

This circuit is built by connecting pin 1 = 5VDC and pin 2 = 0 - 5 VDC analog input from potentiometer (center tap on 20k variable resistor).

A crystal of 4MHz is connected between pin 13 and 14

The output will be seen on Port C, by connecting 330 ohm resistor in series with LED to the ground.

The testing program code:

```
        list p=16f877
        include "p16f877.inc"

        ; Start at the reset vector
        org     0x000
        goto    Start
        org     0x004
Interrupt
        retfie
Start
        bsf     STATUS,RP0      ;bank 1
        bcf     STATUS,RP1
        movlw   H'00'           ;portc [7-0] outputs
        movwf   TRISC           ;left justified, all inputs a/d
        clrf    ADCON1          ;bank 0
        bcf     STATUS,RP0      ;Fosc/8 [7-6], A/D ch0 [5-3], a/d on [0]
        movlw   B'01000001'
        movwf   ADCON0
Main
        call    ad_portc
        goto    Main
ad_portc                        ;wait for acquision time (20uS)
                                ;(non-critical for this test)

        bsf     ADCON0,GO       ;Start A/D conversion

Wait                            ;Wait for conversion to complete
        btfsc   ADCON0,GO
        goto    Wait
        movf    ADRESH,W        ;Write A/D result to PORTC
```

```
movwf   PORTC           ;LEDs
return

end
```

PIC-LCD Display Experiment

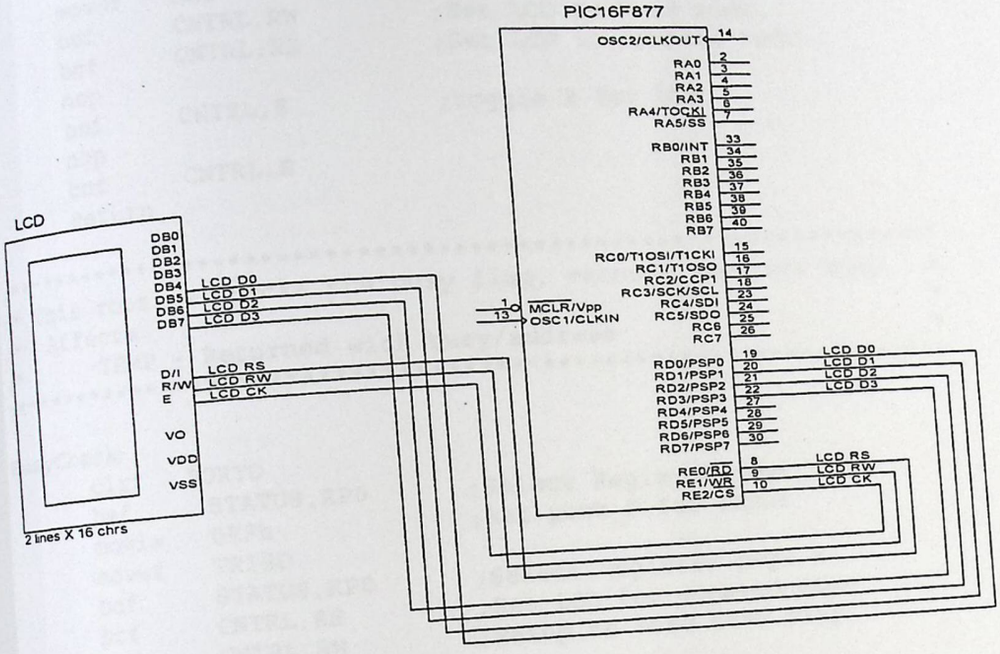


Figure 6-1: PIC-LCD Display Circuit

The Testing Code:

```
TEMP   equ    20h             ;Temporary storage location
CHAR    EQU     21h            ;Character storage location
DAT     EQU     PORTD                  ;LCD data port
CNTRL   EQU     PORTE                  ;LCD control port
E       EQU     3              ;LCD enable signal
RW      EQU     2              ;LCD R/W signal
RS      EQU     1              ;LCD register select

;*****************************************************************
;*  SendChar - Sends character contained in register W to LCD    *
;*****************************************************************

SendChar
        movwf   CHAR            ;Character to be sent is in W
        call    BusyCheck       ;Wait for LCD to be ready
        movf    CHAR,w
        movwf   DAT             ;Send data to LCD
        bcf     CNTRL,RW         ;Set LCD in read mode
        bsf     CNTRL,RS         ;Set LCD in data mode
        nop
        bsf     CNTRL,E          ;toggle E for LCD
        nop
        bcf     CNTRL,E
        return
```

```
;*************************************************************
;*  SendCmd - Sends command contained in register W to LCD    *
;*************************************************************

SendCmd
      movwf    CHAR              ;Command to be sent is in W
      call     BusyCheck         ;Wait for LCD to be ready
      movf     CHAR,w
      movwf    DAT               ;Send data to LCD
      bcf      CNTRL,RW           ;Set LCD in read mode
      bcf      CNTRL,RS           ;Set LCD in command mode
      nop
      bsf      CNTRL,E            ;toggle E for LCD
      nop
      bcf      CNTRL,E
      return

;*************************************************************
;* This routine checks the busy flag, returns when not busy   *
;*                                                            *
;*  Affects:                                                  *
;*      TEMP - Returned with busy/address                     *
;*************************************************************

BusyCheck
      clrf   PORTD
      bsf      STATUS,RP0        ;Select Register page 1
      movlw    0FFh              ;Set port_D for input
      movwf    TRISD
      bcf      STATUS,RP0        ;Select Register page 0
      bcf      CNTRL,RS          ;Set LCD for command mode
      bsf      CNTRL,RW          ;Setup to read busy flag
      nop
      bsf      CNTRL,E            ;Set E high
      nop
      nop                        ;Read busy flag, DDram address
      movf     DAT,w              ;Set E low
      bcf      CNTRL,E
      movwf  TEMP                 ;Check busy flag, high=busy
      btfsc    TEMP,7
      goto     BusyCheck
      bcf      CNTRL,RW           ;Select Register page 1
      bsf      STATUS,RP0
      movlw    000h              ;Set port_D for output
      movwf    TRISD              ;Select Register page 0
      bcf      STATUS,RP0
      return

;*************************************************************
;* This routine initializes the LCD module                    *
;*  Affects:                                                  *
;*      TEMP - Returned with busy/address                     *
;*************************************************************

LCDInit
      bcf    STATUS,RP0
      clrf   PORTE
      clrf   PORTD                ;Select Register page 1
      bsf      STATUS,5          ;Set port_D as outputs
      movlw    B'00000000'
      movwf    TRISD             ;Set port_e as outputs
      movlw    B'00000000'
      movwf    TRISE               ;Select Register page 0
      bcf      STATUS,5
```

```
clrf    PORTE           ;

        movlw   B'00111000'     ;Set LCD to 8 bit interface
        movwf   DAT
        nop
        bsf     CNTRL,E         ;toggle E for LCD
        nop
        bcf     CNTRL,E

        movlw   0x0             ;Setup call to SetupDelay
        movwf   TEMP
        call    SetupDelay      ;Each call to delay is
        call    SetupDelay      ;0.771ms, six makes 4.6ms
        call    SetupDelay      ;This wait is necessary because
        call    SetupDelay      ;the busy flag is not valid yet.
        call    SetupDelay      ;
        call    SetupDelay      ;

        movlw   B'00111000'     ;Function set to 2 lines
        movwf   DAT             ;of 5x7 bit chars
        nop
        bsf     CNTRL,E         ;toggle E for LCD
        nop
        bcf     CNTRL,E
        call    SetupDelay

;Busy flag should be valid after this point
        movlw   B'00001110'     ;Display on, cursor on
        call    SendCmd

        movlw   B'00000001'     ;Clear display
        call    SendCmd

        movlw   B'00000110'     ;Set entry mode inc, no shift
        call    SendCmd

        movlw   B'10000000'     ;Address DDRam upper left
        call    SendCmd

        return
;*****************************************************************
;*  This routine is a software delay.                            *
;*  At 4Mhz clock, the loop takes 3uS, so initialize TEMP with   *
;*  a value of 3 to give 9uS, plus the move etc should result in *
;*  a total time of > 10uS.                                      *
;*****************************************************************

SetupDelay
        nop
        decfsz  TEMP, F
        goto    SetupDelay
        return

        end
```

1.3 Voltage Divider Circuit Testing

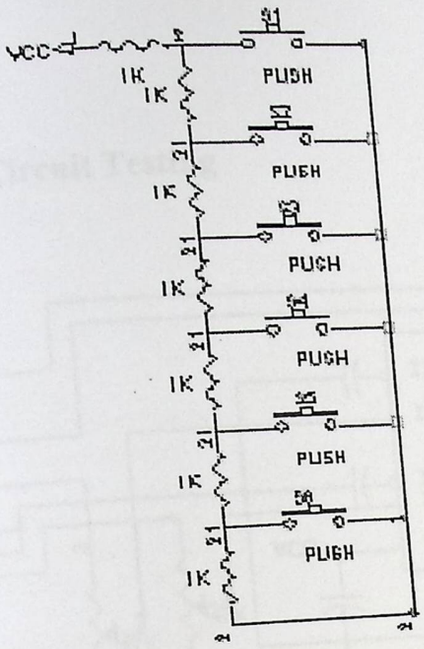


Figure 6-2: Voltage Divider Circuit

The voltage generated by pressing a switch is measured and found to be equal to the expected voltages depending on the following equation:

Vcc = 5v , Rtotal = 7 KOhm, I = 5 / 7K = 0 .71 mA

Vout (S1) = 5 – I (1K) = 4.29v

Measured Vout = 4.29 v

Vout (S2) = 5 – I (1K+1K) = 3.58v

Measured Vout =3.57 v

Vout (S3) = 5 – I (1K+1K+1K) = 2.87 v

Measured Vout = 2.86 v

Vout (S4) = 5 – I (1K+1K+1K+1K) = 2.16 v

Measured Vout =2.14 v

Vout (S5) = 5 – I (1K+1K+1K+1K+1K) = 1.45 v

sured Vout =1.43 v

t (S6) = 5 – I (1K+1K+1K+1K+1K+1K) = 0.74 v

easured Vout =0.71 v

.1.4 Amplification Circuit Testing

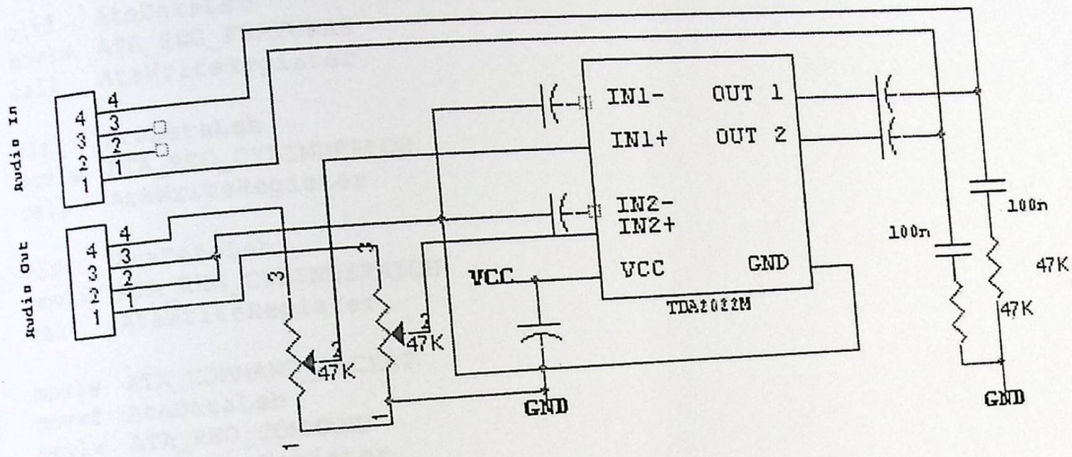


Figure 6-3 : Amplification Circuit

The audio out of the CD Drive is an input to the amplifier chip , the voice can be enlarged and controlled via the potentiometers and out to the speaker .

6.2 System Testing

6.2.1 Open CD Drive Operation

Dc voltage equal to 3.57V is applied to port A pin 2 of the PIC microcontroller, this value Corresponds to open\close operation.

The Testing Code:

```
AtaCmdOpenDoor:
    movlw  0x1b
    movwf  AtaPacket
    movlw  0x02
    movwf  AtaPacket+4
    call   AtaWritePacket
    call   AtaReadPacketEmpty
    btfsc  status,z
    return
```

Measured Vout =1.43 v

Vout (S6) = 5 – I (1K+1K+1K+1K+1K+1K) = 0.74 v

Measured Vout =0.71 v

6.1.4 Amplification Circuit Testing

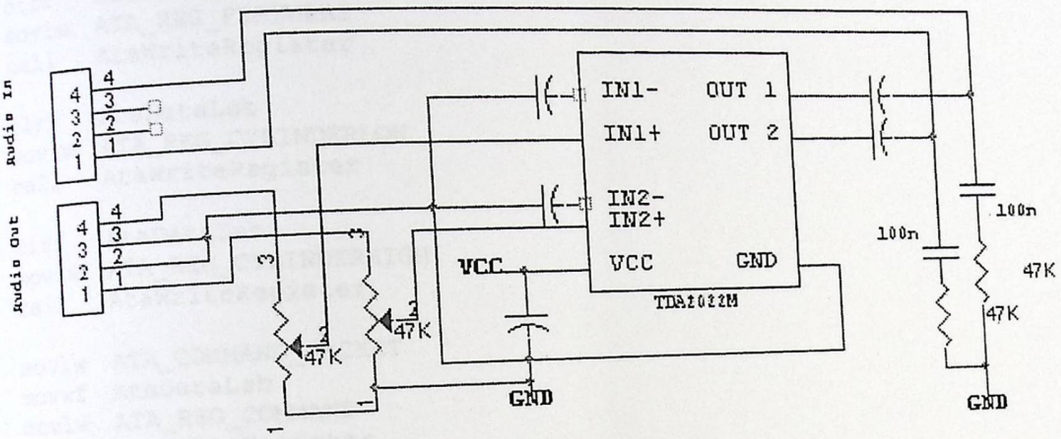


Figure 6-3 : Amplification Circuit

The audio out of the CD Drive is an input to the amplifier chip , the voice can be enlarged and controlled via the potentiometers and out to the speaker .

6.2 System Testing

6.2.1 Open CD Drive Operation

Dc voltage equal to 3.57V is applied to port A pin 2 of the PIC microcontroller, this value Corresponds to open\close operation.

The Testing Code:

```
AtaCmdOpenDoor:
    movlw  0x1b
    movwf  AtaPacket
    movlw  0x02
    movwf  AtaPacket+4
    call   AtaWritePacket
    call   AtaReadPacketEmpty
    btfsc  status,z
    return
```

```
    decfsz AtaErrorRetries,f
    goto   AtaCmdOpenDoor
    call   AtaReset
    return
AtaWritePacket:
    movlw  0x0a
    movwf  AtaDataLsb
    movlw  ATA_REG_DRIVEHEAD
    call   AtaWriteRegister

    clrf   AtaDataLsb
    movlw  ATA_REG_FEATURES
    call   AtaWriteRegister

    clrf   AtaDataLsb
    movlw  ATA_REG_CYLINDERLOW
    call   AtaWriteRegister

    clrf   AtaDataLsb
    movlw  ATA_REG_CYLINDERHIGH
    call   AtaWriteRegister

    movlw  ATA_COMMAND_PACKET
    movwf  AtaDataLsb
    movlw  ATA_REG_COMMAND
    call   AtaWriteRegister


    call   Delay10ms

AtaWriteRegister:
    movwf  AtaRegister
    bcf          status,rp1
    bsf          status,rp0
    movlw  PORTB_CONFIG2
    movwf  trisb
    movlw  PORTC_CONFIG2
    movwf  trisc
    bcf          status,rp1
    bcf          status,rp0
    rlf          AtaRegister,w
    iorlw  0x30
    movwf  porta
    call   Delay1ms
    movf   AtaDataLsb,w
    movwf  portb
    clrf   portc
    call   Delay1ms
    bcf          porta,ATA_IOW
    call   Delay1ms
    bsf          porta,ATA_IOW
    call   Delay1ms
    return

AtaReadPacketEmpty:

AtaRe1:
    movlw  ATA_REG_SECTORCOUNT
    call   AtaReadRegister
    movf   AtaDataLsb,w
    andlw  0x03
```

```
    xorlw  0x03
    btfss  status,z
    goto   AtaRe1

AtaReadNachamble:
    call   Delay200ms

AtaRe2:
    movlw  ATA_REG_STATUS
    call   AtaReadRegister
    movf   AtaDataLsb,w
    andlw  0xc8
    xorlw  0x40
    btfss  status,z
    goto   AtaRe2

    movf   AtaDataLsb,w
    xorlw  0x50
    return

AtaReadRegister:
    movwf  AtaRegister
    bcf          status,rp1
    bsf          status,rp0
    movlw  PORTB_CONFIG1
    movwf  trisb
    movlw  PORTC_CONFIG1
    movwf  trisc
    bcf          status,rp1
    bcf          status,rp0
    rlf          AtaRegister,w
    iorlw  0x30
    movwf  porta
    call   Delay1ms
    bcf           porta,ATA_IOR
    call   Delay1ms
    movf   portb,w
    movwf  AtaDataLsb
    movf   portc,w
    movwf  AtaDataMsb
    call   Delay1ms
    bsf           porta,ATA_IOR
    call   Delay1ms
    return
```

6.2.2 Play Audio operation

Dc voltage equal to 2.14V is applied to port A pin 2 of the PIC microcontroller, this value Corresponds to Play\Stop operation.

The Testing Code:

```
AtaCmdPlayAudio:
    movlw  0x47
    movwf  AtaPacket
    movf   TrackLeadOutMSF_M,w
    movwf  AtaPacket+6
    movf   TrackLeadOutMSF_S,w
```

```
movwf  AtaPacket+7
movf   TrackLeadOutMSF_F,w
movwf  AtaPacket+8
call   AtaWritePacket
call   AtaReadPacketEmpty
btfsc  status,z
return
decfsz AtaErrorRetries,f
goto   AtaCmdPlayAudio
call   AtaReset
return
```

6.2.3 Pause On Audio Operation

Dc voltage equal to 0.71V is applied to port A pin 2 of the PIC microcontroller, this value

Corresponds to Pause On operation.

The Testing Code:

```
AtaCmdPauseOff:
movlw  0x4b
movwf  AtaPacket
movlw  0x01
movwf  AtaPacket+8
call   AtaWritePacket
call   AtaReadPacketEmpty
btfsc  status,z
return
decfsz AtaErrorRetries,f
goto   AtaCmdPauseOff
call   AtaReset
return

 AtaReset:
bcf          portd,ATA_RST
call   Delay200ms
bsf          portd,ATA_RST
call   Delay200ms
return
```

Chapter Seven

Conclusions and Future Works

7.1 CONCLUSION

7.2 PROBLEMS

7.3 FUTURE WORK

Chapter Seven

Conclusions and Future Works

This chapter introduces some significant points in the way of continuing do more and more in the field of the system concepts or tools. Also, it represents the conclusions extracted during designing.

7.1 Conclusions

Many experiences were added to the team cognitive knowledge through studying and designing this project. Many conclusions can be stated here, but only significant and important ones are described here:

1. CD-Drive is programmed by writing command packets into the command register.

2. PIC microcontroller has a lot of features that provide this project and it is programmed to control the CD-Drive but it needs a special programmer.

3. The CD-Drive is constructed as audio player and controlled using buttons; but not all are well-operated.

4. The sound out of the CD-ROM is enlarged using an amplification circuit.

5. The LCD display unit can be controlled using the PIC commands and signals.

6. Programming the PIC using the In-circuit serial programming requires the universal programmer once at least.

7. PIC is programmed by constructing a parallel port programmer.

7.2 Problems

System completion in regard to its objectives is an implementation dependent issue, so studying and designing cannot discover all the problems, but the problems mentioned here are that faced us during studying and design. Skipping these problems is a success. No degradation affects the system if problems appear. Here are problems faced the project team during the system studying:

1. It takes much time for the PIC microcontroller to be available.

2. Implementing an application using a PIC microcontroller is innovative in the university.

3. It was thought that the PIC could be programmed without a programmer, so an In-circuit serial programmer is built but programmer is still needed. Consequently, a parallel port is used.

4. Dealing with CD-Drive commands and signals, and accessing its registers is somehow complicated since it needs a critical timing.

7.3 Future Work

The following ideas are recommended to be considered and implemented as a development of this application:

1. Developing the display unit for better system operation.
2. Operating the buttons that don't work well and complete the entire system.
3. Developing the system to read and control MP3 files.
4. Interfacing the system with a remote control unit.

References

References

[1] http://www.quisquose.com/mp3-cd-player.htm

[2] http://www.pirc.com/tech/mp3/

[3] ATA/ATAPI documentation.

[4] ATA Packet Interface for CD-ROMs documentation.

[5] PIC16F87XA Data Sheet.

[6] http://www.hardwarebook.net/connector

[7] http://www.geocities.com/SiliconValley/2072/atapi.htm

References

[1] http://www.quisquose.com/mp3-cd-player.htm

[2] http://www.pjrc.com/tech/mp3/

[3] ATA/ATAPI documentation.

[4] ATA Packet Interface for CD-ROMs documentation.

[5] PIC16F87XA Data Sheet.

[6] http://www.hardwarebook.net/connector

[7] http://www.geocities.com/SiliconValley/2072/atapi.htm

Appendices

Appendix A: PIC Microcontroller

Appendix B: LCD Display

Appendix C: System Circuit and Code

Appendix A

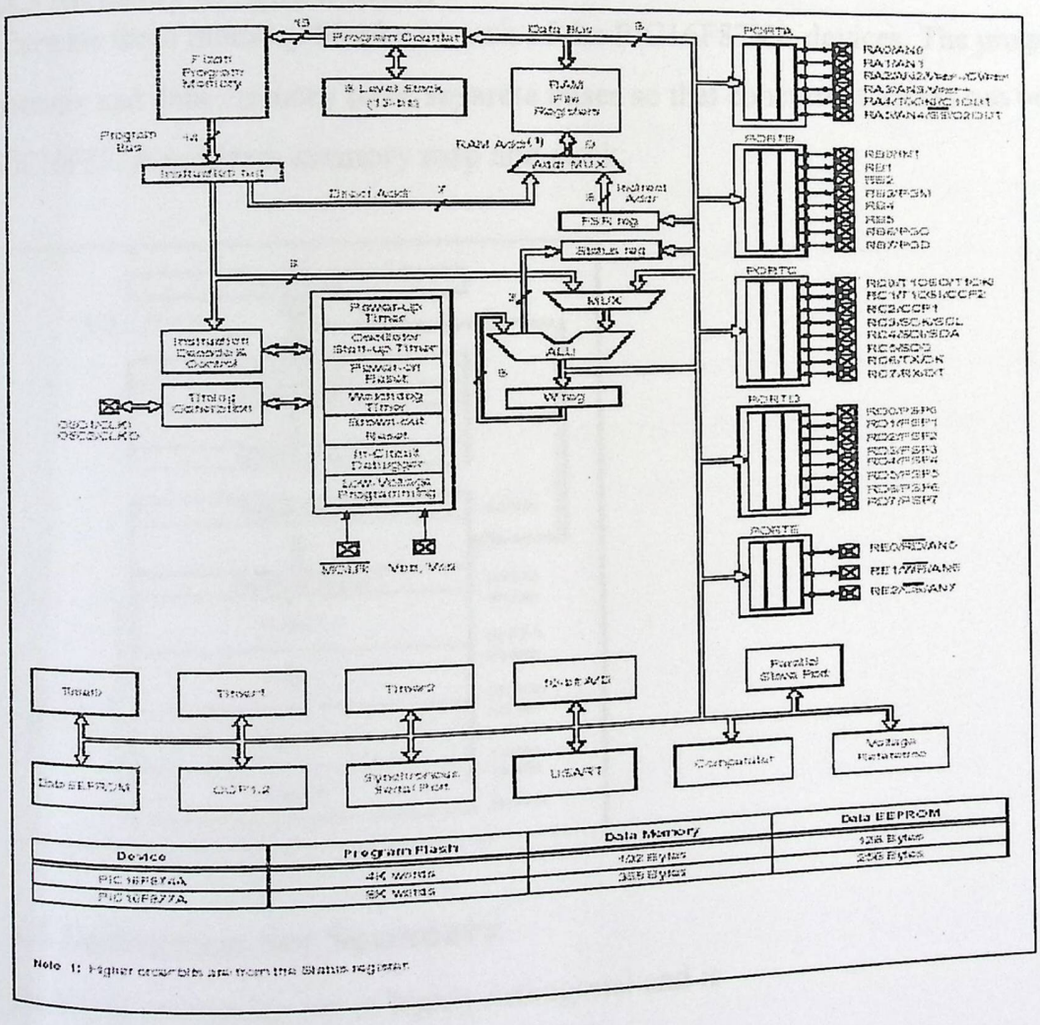
PIC Microcontroller

Appendix A

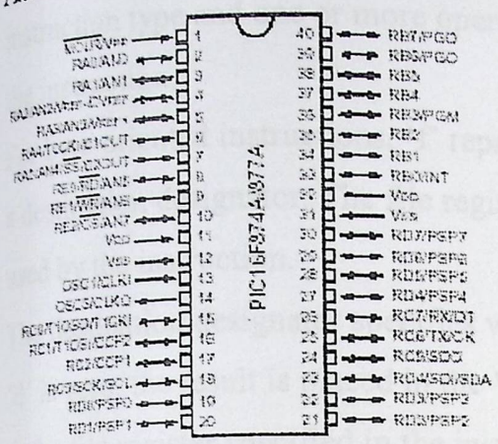
PIC16F877A Microcontroller

The PIC is a microcontroller that contains Five IO Ports (A,B,C,D,E),10-bit Analog-to-Digital Module(8 input channels), 368 bytes data Memory ,256 bytes EEPROM Data Memory,8K Flash Program Memory ,Serial Communications (MSSP,USART), Parallel Communication (PSP), Three Timers, Instruction Set (35 instructions), 15 Interrupts and two Analog Comparator.

A.1 Block Diagram



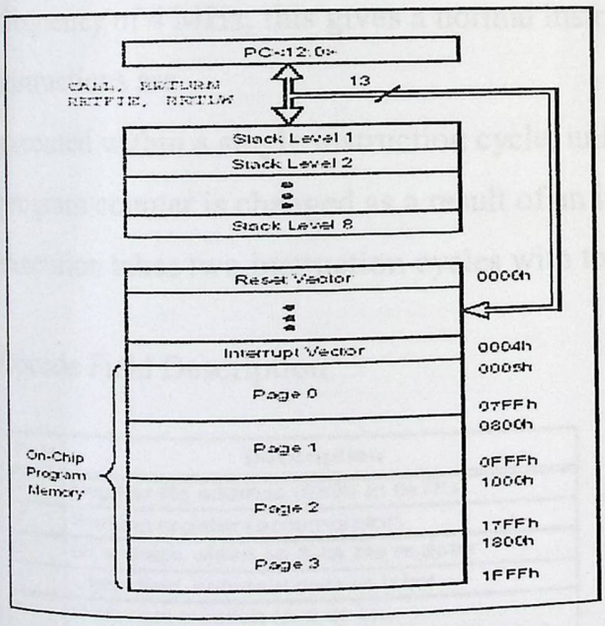
A.2 Pin Diagram



A.3 Memory Organization

There are three memory blocks in each of the PIC16F87XA devices. The program memory and data memory have separate buses so that concurrent access can occur.

PIC16F877A program memory map and stack:



A.6 Instruction Set Summary

The PIC16 instruction set is highly orthogonal and is comprised of three basic categories:

- Byte-oriented operations
- Bit-oriented operations
- Literal and control operations

Each PIC16 instruction is a 14-bit word divided into an opcode which specifies the instruction type and one or more operands which further specify the operation of the instruction.

For byte-oriented instructions, 'f' represents a file register designator and 'd' represents a destination designator. The file register designator specifies which file register is to be used by the instruction.

The destination designator specifies where the result of the operation is to be placed. If 'd' is zero, the result is placed in the W register. If 'd' is one, the result is placed in the file register specified in the instruction.

For bit-oriented instructions, 'b' represents a bit field designator which selects the bit affected by the operation, while 'f' represents the address of the file in which the bit is located.

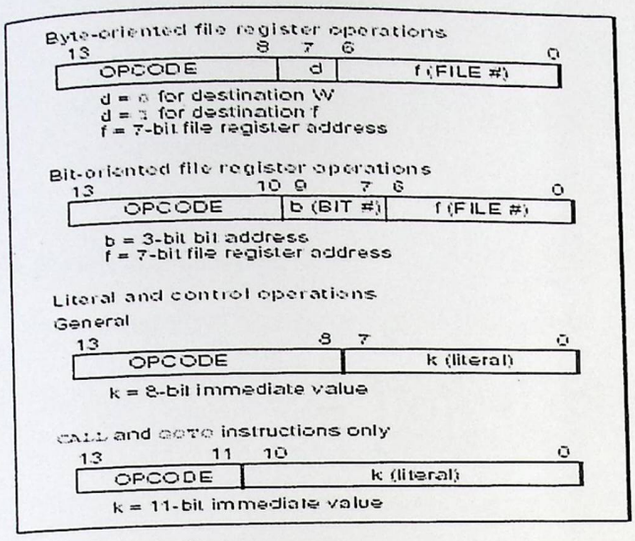
For literal and control operations, 'k' represents an eight or eleven-bit constant or literal value One instruction cycle consists of four oscillator periods; for an oscillator frequency of 4 MHz, this gives a normal instruction execution time of 1 µs. All instructions are

executed within a single instruction cycle, unless a conditional test is true, or the program counter is changed as a result of an instruction. When this occurs, the execution takes two instruction cycles with the second cycle executed as a NOP.

Opcode Field Description

Field	Description
f	Register file address (0x00 to 0x7F)
W	Working register (accumulator)
b	Bit address within an 8-bit file register
k	Literal field, constant data or label
x	Don't care location (= 0 or 1). The assembler will generate code with x = 0. It is the recommended form of use for compatibility with all Microchip software tools.
d	Destination select; d = 0: store result in W. d = 1: store result in file register f. Default is d = 1.
PC	Program Counter
TO	Time-out bit
PD	Power-down bit

General Format for Instructions



Instruction Set

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb	LSb	Status Affected	Notes
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 lfff ffff		Z	2
CLRW	-	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 lfff 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			
CLRWDT	-	Clear Watchdog Timer	1	00 0000 0110 0100		$\overline{TO}$,$\overline{PD}$	
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	00 0000 0000 1000			
RETURN	-	Return from Subroutine	2	00 0000 0110 0011		$\overline{TO}$,$\overline{PD}$	
SLEEP	-	Go into Standby mode	1	00 0000 0110 0011		C,DC,Z	
SUBLW	k	Subtract W from Literal	1	11 110x kkkk kkkk		Z	
XORLW	k	Exclusive OR Literal with W	1	11 1010 kkkk kkkk			

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

LCD Display

Appendix B

LCD Display

B.1 Introduction

One of the first things people want to interface to a design is an LCD display, both to help with debugging their programs and as a way to provide results to the outside world. Most text displaying LCD's are based on the Hitachi HD44780A LCD controller. This controller takes care of all the multiplexing and the peculiarities required by the LCD display and provides a low level command interface for certain actions like screen clearing, cursor shape and size, character displaying etc.

The HD44780A communicates with the host mcu through an 8 bit bi-directional interface (D0-D7) and 3 control lines (E, RS, RW).

LCD's are slow devices when compared to microcontrollers. Special attention must be paid so the commands and data are not send too quickly from the mcu, and the designer must control the communication speed and timing to ensure that the slow LCD and the fast mcu stay synchronized.

The LCD module contains 3 different functional blocks.

The Character Generator ROM (CG-ROM) , the character generator RAM (CG-RAM) and the Display Data RAM (DD-RAM) .

The ROM character generator is factory programmed and contains the 208 characters to be displayed.

The CG-RAM allows the user to define up to 8 special character that are not included in the CG-ROM.

The DD-RAM holds the characters that are actually displayed in the LCD

Both the CG-RAM and DD-RAM can be read and written (through commands) by the mcu.

The most important of the function blocks is the DD-RAM. This has to do with the way that is implemented and sometimes is different from LCD to LCD depending on how many lines by how many rows the LCD can display the DD-RAM address changes.

B.3 Commands and initialization sequence

The HD44780A provides the user with some commands that control the behavior of the LCD.

Instruction	Code										Description	Execution time
	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear display	0	0	0	0	0	0	0	0	0	1	Clears display and returns cursor to the home position .	1.64mS
Cursor home	0	0	0	0	0	0	0	0	1	*	Returns cursor to home position DDRAM contents remains unchanged.	1.64mS
Entry mode set	0	0	0	0	0	0	0	1	I/D	S	Sets cursor move direction (I/D), specifies to shift the display (S).	40uS
Display On/Off control	0	0	0	0	0	0	1	D	C	B	Sets On/Off of all display (D), cursor On/Off (C) and blink of cursor position character (B).	40uS
Cursor/display shift	0	0	0	0	0	1	S/C	R/L	*	*	Sets cursor-move or display-shift (S/C), shift direction (R/L). DDRAM contents remains unchanged.	40uS
Function set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display line (N) and character font(F).	40uS
Set CGRAM address	0	0	0	1	CGRAM address						Sets the CGRAM address.	40uS
Set DDRAM address	0	0	1	DDRAM address							Sets the DDRAM address.	40uS
Read busy-flag and address counter	0	1	BF	DDRAM address							Reads Busy-flag (BF) indicating internal operation is being performed and reads address counter contents.	0uS
Write to CGRAM or DDRAM	1	0	write data								Writes data to CGRAM or DDRAM.	40uS
Read from CGRAM or DDRAM	1	1	read data								Reads data from CGRAM or DDRAM.	40uS

B.4 LCD Commands

The LCD accepts the following commands

Function	Code	ASCII
Cursor Home	Ctrl A	1
Hide Cursor	Ctrl D	4
Show Underline Cursor	Ctrl E	5
Show Blinking Block Cursor	Ctrl F	6
Backspace	Ctrl H or Backspace key	8
Vertical Tab	Ctrl K	11
Clear Screen	Ctrl L	12
Carriage Return	Ctrl M	13

B.5 LCD Software

A number of functions and macros are implemented in order to facilitate the re-use of the code in case we do not need a dedicated controller for the LCD, but we want to be able to correctly initialize and use one.

These functions are:

LCD_Init - Initialize the LCD and set cursor at the beginning of first line

LCD_Command - Send a byte command to the LCD

LCD_Write - Write a char at current position of the cursor

LCD_GotoXY - Move cursor to a specific position

LCD_EPulse - Toggle the E line High and then Low

Delay40us - Delay 40 us

Delay100us - Delay 100 us

Delay4ms - Delay 4ms

A number of macros is also implemented

LCD_CLS - Clear LCD move cursor at beginning of first line

LCD_Home - Move cursor to left edge of the display

LCD_ON - Turn display on

LCD_OFF - Turn display off

LCD_CursorOn - Cursor will be displayed

LCD_CursorOFF - No cursor will be displayed

LCD_BlinkON - The character at the cursor position will blink

LCD_BlinkOFF - The character at the cursor position will not blink

LCD_ShiftCursorL - Shift cursor one position to the left

LCD_ShiftCursorR - Shift cursor one position to the right

LCD_ShiftDisplayL - Shift BOTH LINES 1 char to the Left

LCD_ShiftDisplayR - Shift BOTH LINES 1 char to the Right

Fist we make sure that the stack of the 2313 is initialized properly before we start calling any functions.

PortB and PortD as set according to their usage and then we initialize the LCD, displaying a "splash screem" which is stored in the EEPROM..

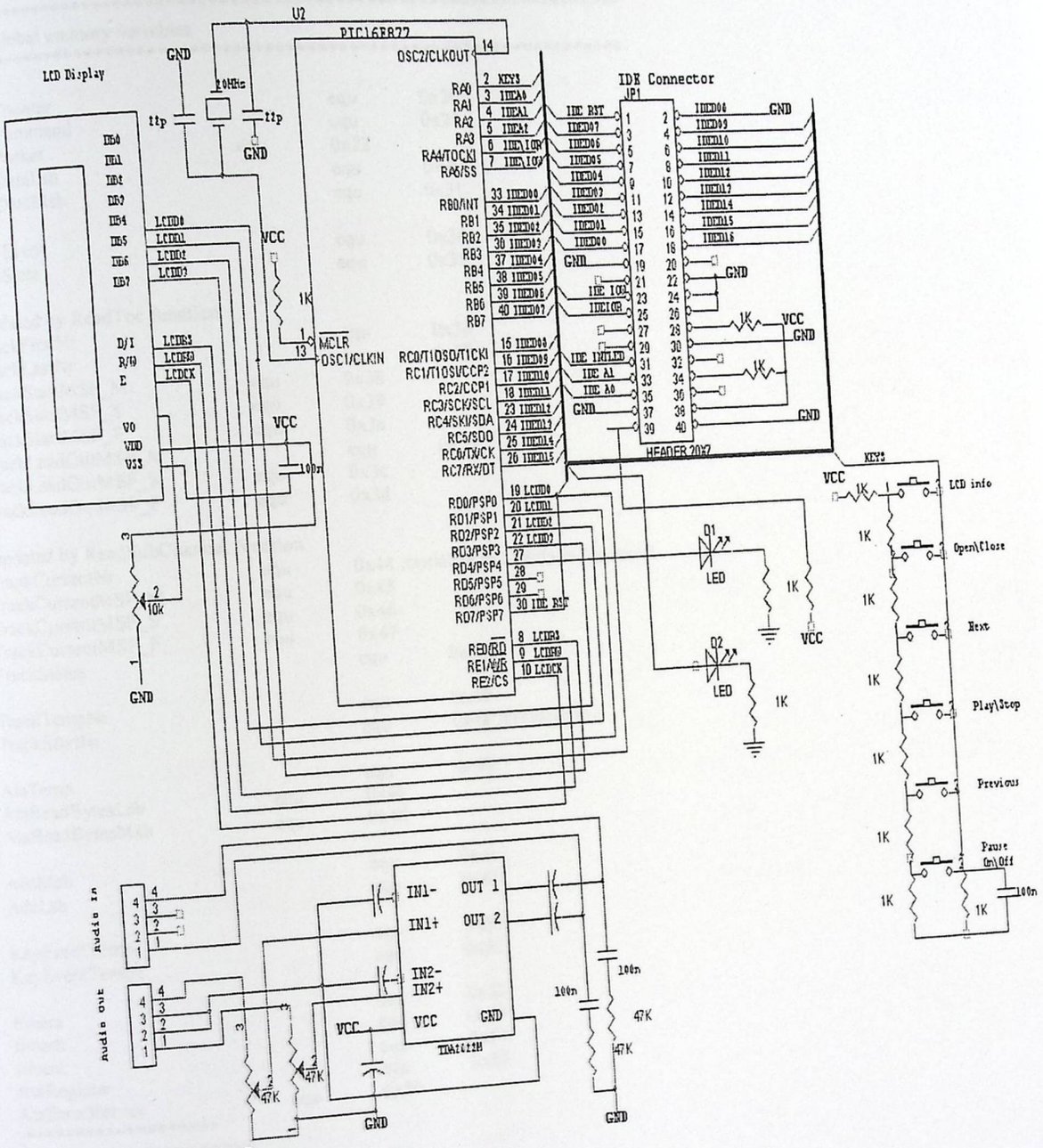
Then we read the EEPROM to get the speed that the UART is going to use and proceed with its initialization.

After that we do a constant loop that waits for an incoming character, parses the character to see iff it is a control code and finally displays it on the LCD.

Appendix C

SYSTEM SCHEMATIC DIAGRAM & CODE

1. System Circuit:



2. Software Code:

```
;*************************************************************
;All global memory variables
;*************************************************************

AtaCounter                      equ     0x20
AtaCommand                      equ     0x21
AtaPacket               equ     0x22
AtaDataLsb                      equ     0x30
AtaDataMsb                      equ     0x31

KeyEvent                        equ     0x34
KeyState                        equ     0x35

;updated by ReadToc function
TrackFirstNr                    equ     0x36
TrackLastNr                     equ     0x37
TrackStartMSF_M         equ     0x38
TrackStartMSF_S         equ     0x39
TrackStartMSF_F         equ     0x3a
TrackLeadOutMSF_M               equ     0x3b
TrackLeadOutMSF_S       equ     0x3c
TrackLeadOutMSF_F       equ     0x3d

;updated by ReadSubChannel function
TrackCurrentNr          equ     0x44 ;initialise to 1 (default 1st track)
TrackCurrentMSF_M       equ     0x45
TrackCurrentMSF_S       equ     0x46
TrackCurrentMSF_F       equ     0x47
TrackStatus                     equ     0x48

TrackTempNr                     equ     0x49
TrackStartNr                    equ     0x4a ;initialise to 1

AtaTemp                         equ     0x4b
AtaReadBytesLsb         equ     0x4c
AtaReadBytesMsb         equ     0x4d

AdcMsb                          equ     0x4e
AdcLsb                          equ     0x4f

KeyEventTemp1                   equ     0x50
KeyEventTemp2                   equ     0x51

timera                          equ     0x52
timerb                          equ     0x53
timerc                          equ     0x54
AtaRegister                     equ     0x55
AtaErrorRetries         equ     0x56
;*******************
;PORT CONFIGURATIONS
;*******************
;analog key input, IDE control signals
PORTA_CONFIG1           equ     0x01    ;bit5: out,out,out,out,out,in :bit0
ATA_IOW                         equ     5
ATA_IOR                 equ     4
ATA_A2                  equ     3
ATA_A1                  equ     2
ATA_A0                  equ     1
KEYS                            equ     0

;used as a bidirectional data bus for IDE_D00/D15
PORTB_CONFIG1           equ     0xff    ;bit7: in,in,in,in,in,in,in,in :bit0
PORTC_CONFIG1           equ     0xff    ;bit7: in,in,in,in,in,in,in,in :bit0

PORTB_CONFIG2           equ     0x00    ;bit7: out,out,out,out,out,out,out,out :bit0
PORTC_CONFIG2           equ     0x00    ;bit7: out,out,out,out,out,out,out,out :bit0

;lcd data bus D0/D4, IDE reset, status leds
```

```
PORTD_CONFIG1           equ     0x00    ;bit7: out,in,out,out,out,out,out,out :bit0
ATA_RST                 equ     7
LED_3                   equ     6
LED_2                   equ     5
LED_STATUS              equ     4

;lcd control signals
PORTE_CONFIG1           equ     0x00    ;bit2: out,out,out :bit0
LCD_CLK                 equ     2
LCD_RW                  equ     1
LCD_RS                  equ     0

;keypad events
KEY_EVENT_PLAY_STOP     equ     0
KEY_EVENT_NEXT          equ     1
KEY_EVENT_PREVIOUS      equ     2
KEY_EVENT_OPEN_CLOSE    equ     3
KEY_EVENT_PAUSE_ONOFF   equ     4
KEY_EVENT_LCD_INFO      equ     5

;key states (bit in variable KeyState)
KEY_STATE_PLAY_STOP             equ     0
KEY_STATE_PAUSE_ONOFF           equ     1
KEY_STATE_OPEN_CLOSE            equ     2


;keypad inputs voltages
;DigValue=(Vin*1023)/5                  equ     73      ;msb=0,lsb=73       (0x49)      ;<0,35V
;KEYPAD_NO_KEY                          equ     74      ;msb=0,lsb=74(0x4a) ;0,36V-0,71V-1,06V
;KEYPAD_PAUSE_ONOFF                     equ     219     ;msb=0,lsb=219(0xdb)            ;1,07V-
;KEYPAD_PREVIOUS                                        equ     365     ;msb=1,lsb=109(0x6d)
1,43V-1,78V                             equ     366     ;msb=1,lsb=110(0x6e)            ;1,79V-
;
;KEYPAD_PLAY_STOP                       equ     512     ;msb=2,lsb=0
2,14V-2,49V
;KEYPAD_NEXT
        ;2,50V-2,86V-3,20V               equ     657     ;msb=2,lsb=145(0x91)            ;3,21V-3,57V-
;KEYPAD_OPEN_CLOSE                                       equ     803     ;msb=3,lsb=35(0x23)
3,92V                                   equ     804     ;msb=3,lsb=36(0x24) ;3,93V-4,29V-
;
;KEYPAD_LCD_INFO                        equ     951     ;msb=3,lsb=183(0xb7)            ;>4,65V
4,64V
;KEYPAD_MORE_THAN_ONE_KEY
;ATA registers                  equ     0x00
ATA_REG_DATA                    equ     0x01
ATA_REG_ERROR                   equ     0x01
ATA_REG_FEATURES                0x02
ATA_REG_SECTORCOUNT     equ     0x03
ATA_REG_SECTORNUMBER    equ     0x04
ATA_REG_CYLINDERLOW     equ     0x05
ATA_REG_CYLINDERHIGH    equ     equ     0x06
ATA_REG_DRIVEHEAD               0x07
ATA_REG_STATUS          equ     equ     0x07
ATA_REG_COMMAND

;ATA commands                   equ     0xa1
ATA_COMMAND_IDENTIFY_DRIVE      equ     0xa0
ATA_COMMAND_PACKET


;*******************
;Main Function
;*******************


list    p=16f877,r=hex

include "pic16f877.inc"
include "main.inc"
```

```
        global  Delay1ms
        global  Delay10ms
        global  Delay200ms
        global  Delay5s


        extern  AtaUpdate
        extern  AtaInit
#ifdef LCD
        extern  LcdUpdate
#endif


_config code
        dw
        FOSC_HS|WDTE_OFF|PWRTE_ON|CP_OFF|BODEN_OFF|LVP_OFF|CPD_OFF|WRT_ON|ICD_OFF|CP2_OFF




_vector code
        goto    Start


_rom1 code
Start:
        call    SysInit
        call    AtaInit
Mainloop:
        call    KeyUpdate
        call    AtaUpdate
#ifdef LCD
        call    LcdUpdate
#endif
        goto    Mainloop


SysInit:

        bcf             status,rp1
        bsf             status,rp0

        movlw   PORTA_CONFIG1
        movwf   trisa
        movlw   PORTB_CONFIG1
        movwf   trisb
        movlw   PORTC_CONFIG1
        movwf   trisc
        movlw   PORTD_CONFIG1
        movwf   trisd
```

```
        movlw   PORTE_CONFIG1
        movwf   trise

        movlw   B'10001110'
        movwf   adcon1

        bcf             status,rp1
        bcf             status,rp0

        movlw   B'10000001'
        movwf   adcon0

        bsf             porta,ATA_IOW
        bsf             porta,ATA_IOR

        bsf             portd,ATA_RST
        bcf             portd,LED_STATUS

        clrf    KeyEvent
        clrf    KeyState

        movlw   0x01
        movwf   TrackCurrentNr
        movwf   TrackStartNr

        return

KeyUpdate:
        clrf    KeyEvent
        call    KeyScan
        movf    KeyEventTemp1,w
        xorlw   0x00
        btfsc   status,z
        goto    Kuend
        movf    KeyEventTemp1,w
        movwf   KeyEventTemp2
        call    Delay10ms
        call    KeyScan
        movf    KeyEventTemp1,w
        xorwf   KeyEventTemp2,w
        btfss   status,z
        goto    Kuend
        movf    KeyEventTemp2,w
        movwf   KeyEvent
Kuend:
        return

KeyScan:        bsf             adcon0,godone
Kuw:    btfsc   adcon0,godone
        goto    Kuw

        movf    adresh,w
        movwf   AdcMsb
        bcf             status,rp1
```

```
        bsf     status,rp0
        movf    adresl,w
        bcf     status,rp1
        bcf     status,rp0
        movwf   AdcLsb

        clrf    KeyEventTemp1
        movf    AdcMsb,w
        sublw   0x03
        btfss   status,c
        goto    Kend
        movf    AdcMsb,w
        sublw   0x03
        btfss   status,z
        goto    Ku1
        movf    AdcLsb,w
        sublw   D'183'
        btfss   status,c
        goto    Kend
        movf    AdcLsb,w
        sublw   D'36'
        btfsc   status,c
        goto    Ku0
        bsf     KeyEventTemp1   ,KEY_EVENT_LCD_INFO
        goto    Kend
Ku0:    bsf     KeyEventTemp1   ,KEY_EVENT_OPEN_CLOSE
        goto    Kend
Ku1:    movf    AdcMsb,w
        sublw   0x02
        btfss   status,z
        goto    Ku3
        movf    AdcLsb,w
        sublw   D'145'
        btfsc   status,c
        goto    Ku2
        bsf     KeyEventTemp1   ,KEY_EVENT_OPEN_CLOSE
        goto    Kend
Ku2:    bsf     KeyEventTemp1   ,KEY_EVENT_NEXT
        goto    Kend
Ku3:    movf    AdcMsb,w
        sublw   0x01
        btfss   status,z
        goto    Ku5
        movf    AdcLsb,w
        sublw   D'110'
        btfsc   status,c
        goto    Ku4
        bsf     KeyEventTemp1   ,KEY_EVENT_PLAY_STOP
        goto    Kend
Ku4:    bsf     KeyEventTemp1   ,KEY_EVENT_PREVIOUS
        goto    Kend
Ku5:    movf    AdcLsb,w
        sublw   D'219'
        btfsc   status,c
        goto    Ku6
        bsf     KeyEventTemp1   ,KEY_EVENT_PREVIOUS
        goto    Kend
Ku6:    movf    AdcLsb,w
        sublw   D'74'
        btfsc   status,c
        goto    Kend
        bsf     KeyEventTemp1   ,KEY_EVENT_PAUSE_ONOFF
Kend:
        return



Delay1ms:
        movlw   0x07
```

```
        movwf   timerb
L11:    movlw   0xed
        movwf   timera
L12:    decfsz  timera,f
        goto    L12
        decfsz  timerb,f
        goto    L11
        return


Delay10ms:
        movlw   0x41
        movwf   timerb
L21:    movlw   0xff
        movwf   timera
L22:    decfsz  timera,f
        goto    L22
        decfsz  timerb,f
        goto    L21
        return


Delay200ms:
        movlw   0x06
        movwf   timerc
L31:movlw       0xd9
        movwf   timerb
L32:    movlw   0xff
        movwf   timera
L33:    decfsz  timera,f
        goto    L33
        decfsz  timerb,f
        goto    L32
        decfsz  timerc,f
        goto    L31
        return


Delay5s:
        movlw   0x7f
        movwf   timerc
L41:movlw       0xff
        movwf   timerb
L42:    movlw   0xff
        movwf   timera
L43:    decfsz  timera,f
        goto    L43
        decfsz  timerb,f
        goto    L42
        decfsz  timerc,f
        goto    L41
        return



LedTest:
        bsf             portd,LED_STATUS
        call    Delay200ms
        call    Delay200ms
        bcf             portd,LED_STATUS
        call    Delay200ms
        call    Delay200ms

        return

        END
```

```
;*******************
;ATA Function
;*******************

list        p=16f877,r=hex

            include     "pic16f877.inc"
            include     "main.inc"
            include "ata.inc"



            global      AtaUpdate
            global      AtaInit


            extern      Delay1ms
            extern      Delay10ms
            extern      Delay200ms
            extern      Delay5s

_rom1 code



AtaUpdate:
            movf        KeyEvent,w
            xorlw       0x00
            btfsc       status,z
            return
            bsf                 portd,LED_STATUS
            movlw       D'10'
            movwf       AtaErrorRetries


            btfsc       KeyState,KEY_STATE_OPEN_CLOSE
            goto        Iu2
            btfss       KeyEvent,KEY_EVENT_PLAY_STOP
            goto        Iu2
            btfsc       KeyState,KEY_STATE_PLAY_STOP
            goto        Kst
Kpl:        call        AtaCmdReadToc
            movf        TrackStartMSF_M,w
            movwf       AtaPacket+3
            movf        TrackStartMSF_S,w
            movwf       AtaPacket+4
            movf        TrackStartMSF_F,w
            movwf       AtaPacket+5
            call        AtaCmdPlayAudio
            bsf                 KeyState,KEY_STATE_PLAY_STOP
            bcf                 KeyState,KEY_STATE_PAUSE_ONOFF
            goto        IuEnd
Kst:        call        AtaCmdStopAudio
            bcf                 KeyState,KEY_STATE_PLAY_STOP
            goto        IuEnd


Iu2:        btfsc       KeyState,KEY_STATE_OPEN_CLOSE
            goto        Iu3
            btfsc       KeyState,KEY_STATE_PAUSE_ONOFF
            goto        Iu3
            btfss       KeyEvent,KEY_EVENT_NEXT
            goto        Iu3
```

```
        incf    TrackStartNr,f
        btfss   KeyState,KEY_STATE_PLAY_STOP
        goto    luEnd
        call    AtaCmdReadSubChannel
        incf    TrackCurrentNr,w
        movwf   TrackStartNr
        call    AtaCmdReadToc
        movf    TrackStartMSF_M,w
        movwf   AtaPacket+3
        movf    TrackStartMSF_S,w
        movwf   AtaPacket+4
        movf    TrackStartMSF_F,w
        movwf   AtaPacket+5
        call    AtaCmdPlayAudio
        goto    luEnd


lu3:    btfsc   KeyState,KEY_STATE_OPEN_CLOSE
        goto    lu4
        btfsc   KeyState,KEY_STATE_PAUSE_ONOFF
        goto    lu4
        btfss   KeyEvent,KEY_EVENT_PREVIOUS
        goto    lu4
        decf    TrackStartNr,f
        btfss   KeyState,KEY_STATE_PLAY_STOP
        goto    luEnd
        call    AtaCmdReadSubChannel
        decf    TrackCurrentNr,w
        movwf   TrackStartNr
        call    AtaCmdReadToc
        movf    TrackStartMSF_M,w
        movwf   AtaPacket+3
        movf    TrackStartMSF_S,w
        movwf   AtaPacket+4
        movf    TrackStartMSF_F,w
        movwf   AtaPacket+5
        call    AtaCmdPlayAudio
        goto    luEnd


lu4:    btfss   KeyEvent,KEY_EVENT_OPEN_CLOSE
        goto    lu5
        btfsc   KeyState,KEY_STATE_OPEN_CLOSE
        goto    Kcl
Kop:    btfsc   KeyState,KEY_EVENT_PLAY_STOP
        call    AtaCmdStopAudio
        call    AtaCmdOpenDoor
        bsf             KeyState,KEY_STATE_OPEN_CLOSE
        goto    luEnd
Kcl:    call    AtaCmdCloseDoor
        call    AtaCmdStopAudio
        bcf             KeyState,KEY_STATE_PLAY_STOP
        bcf             KeyState,KEY_STATE_PAUSE_ONOFF
        bcf             KeyState,KEY_STATE_OPEN_CLOSE
        movlw   0x01
        movwf   TrackStartNr
        goto    luEnd


lu5:    btfsc   KeyState,KEY_STATE_OPEN_CLOSE
        goto    luEnd
        btfss   KeyEvent,KEY_EVENT_PAUSE_ONOFF
        goto    luEnd
        btfss   KeyState,KEY_STATE_PLAY_STOP
        goto    luEnd
        btfsc   KeyState,KEY_STATE_PAUSE_ONOFF
```

```
        goto    Kpf
Kpo:    call    AtaCmdPauseOn
        bsf             KeyState,KEY_STATE_PAUSE_ONOFF
        goto    luEnd
Kpf:    call    AtaCmdPauseOff
        bcf             KeyState,KEY_STATE_PAUSE_ONOFF
        goto    luEnd
luEnd:
        bcf             portd,LED_STATUS
        return



AtaInit:
        bsf             portd,LED_STATUS
        call    AtaReset
        call    Delay5s
        call    Delay5s

        movlw   D'12'
        movwf   AtaCounter
        movlw   AtaPacket
        movwf   fsr
Ai1:    clrf    indf
        incf    fsr,f
        decfsz  AtaCounter,f
        goto    Ai1
        call    AtaCmdStopAudio
#ifdef LCD
        call    AtaCmdInquiry
#endif
        bcf             portd,LED_STATUS
        return



AtaReset:
        bcf             portd,ATA_RST
        call    Delay200ms
        bsf             portd,ATA_RST
        call    Delay200ms
        return



AtaCmdStopAudio:
        movlw   0x4e
        movwf   AtaPacket
        call    AtaWritePacket
        call    AtaReadPacketEmpty
        xorlw   0x01
        btfss   status,z
        return
        decfsz  AtaErrorRetries,f
        goto    AtaCmdStopAudio
        call    AtaReset
        return



AtaCmdPauseOn:
```

```
        movlw   0x4b
        movwf   AtaPacket
        call    AtaWritePacket
        call    AtaReadPacketEmpty
        btfsc   status,z
        return
        decfsz  AtaErrorRetries,f
        goto    AtaCmdPauseOn
        call    AtaReset
        return


AtaCmdPauseOff:
        movlw   0x4b
        movwf   AtaPacket
        movlw   0x01
        movwf   AtaPacket+8
        call    AtaWritePacket
        call    AtaReadPacketEmpty
        btfsc   status,z
        return
        decfsz  AtaErrorRetries,f
        goto    AtaCmdPauseOff
        call    AtaReset
        return


AtaCmdPlayAudio:
        movlw   0x47
        movwf   AtaPacket
        movf    TrackLeadOutMSF_M,w
        movwf   AtaPacket+6
        movf    TrackLeadOutMSF_S,w
        movwf   AtaPacket+7
        movf    TrackLeadOutMSF_F,w
        movwf   AtaPacket+8
        call    AtaWritePacket
        call    AtaReadPacketEmpty
        btfsc   status,z
        return
        decfsz  AtaErrorRetries,f
        goto    AtaCmdPlayAudio
        call    AtaReset
        return


AtaCmdCloseDoor:
        movlw   0x1b
        movwf   AtaPacket
        movlw   0x03
        movwf   AtaPacket+4
        call    AtaWritePacket
        call    AtaReadPacketEmpty
        btfsc   status,z
        return
        decfsz  AtaErrorRetries,f
        goto    AtaCmdCloseDoor
        call    AtaReset
        return
```

```
AtaCmdOpenDoor:
        movlw   0x1b
        movwf   AtaPacket
        movlw   0x02
        movwf   AtaPacket+4
        call    AtaWritePacket
        call    AtaReadPacketEmpty
        btfsc   status,z
        return
        decfsz  AtaErrorRetries,f
        goto    AtaCmdOpenDoor
        call    AtaReset
        return


#ifdef LCD
AtaCmdInquiry:
        movlw   0x12
        movwf   AtaPacket
        movlw   0xff
        movwf   AtaPacket+4
        call    AtaWritePacket
        call    AtaReadPacketInquiry
        btfsc   status,z
        return
        decfsz  AtaErrorRetries,f
        goto    AtaCmdInquiry
        call    AtaReset
        return
#endif


AtaCmdReadToc:
        movlw   0x43
        movwf   AtaPacket
        movlw   0x02
        movwf   AtaPacket+1
        movlw   0xff
        movwf   AtaPacket+7
        movlw   0xff
        movwf   AtaPacket+8
        call    AtaWritePacket
        call    AtaReadPacketToc
        btfsc   status,z
        return
        decfsz  AtaErrorRetries,f
        goto    AtaCmdReadToc
        call    AtaReset
        return


AtaCmdReadSubChannel:
        movlw   0x42
        movwf   AtaPacket
        movlw   0x02
        movwf   AtaPacket+1
        movlw   0x40
        movwf   AtaPacket+2
        movlw   0x01
        movwf   AtaPacket+3
        movlw   0xff
        movwf   AtaPacket+7
```

```
        movlw   0xff
        movwf   AtaPacket+8
        call    AtaWritePacket
        call    AtaReadPacketSubChannel
        btfsc   status,z
        return
        decfsz  AtaErrorRetries,f
        goto    AtaCmdReadSubChannel
        call    AtaReset
        return



AtaWritePacket:
        movlw   0x0a
        movwf   AtaDataLsb
        movlw   ATA_REG_DRIVEHEAD
        call    AtaWriteRegister

        clrf    AtaDataLsb
        movlw   ATA_REG_FEATURES
        call    AtaWriteRegister

        clrf    AtaDataLsb
        movlw   ATA_REG_CYLINDERLOW
        call    AtaWriteRegister

        clrf    AtaDataLsb
        movlw   ATA_REG_CYLINDERHIGH
        call    AtaWriteRegister

        movlw   ATA_COMMAND_PACKET
        movwf   AtaDataLsb
        movlw   ATA_REG_COMMAND
        call    AtaWriteRegister


        call    Delay10ms


AtaWc2:
        movlw   ATA_REG_STATUS
        call    AtaReadRegister
        movf    AtaDataLsb,w
        andlw   0x08
        btfsc   status,z
        goto    AtaWc2

        bcf             status,rp1
        bsf             status,rp0
        movlw   PORTB_CONFIG2
        movwf   trisb
        movlw   PORTC_CONFIG2
        movwf   trisc
        bcf             status,rp1
        bcf             status,rp0
        bcf             porta,ATA_A0
        bcf             porta,ATA_A1
        bcf             porta,ATA_A2
        movlw   0x06
        movwf   AtaCounter
        movlw   AtaPacket
        movwf   fsr
AtaWP1:
        movf    indf,w
        movwf   portb
        incf    fsr,f
```

```
        movf    indf,w
        movwf   portc
        call    Delay10ms
        incf    fsr,f
        bcf             porta,ATA_IOW
        call    Delay10ms
        bsf             porta,ATA_IOW
        call    Delay10ms
        decfsz  AtaCounter,f
        goto    AtaWP1


        movlw   D'12'
        movwf   AtaCounter
        movlw   AtaPacket
        movwf   fsr
Au1:    clrf    indf
        incf    fsr,f
        decfsz  AtaCounter,f
        goto    Au1
        return



AtaWriteRegister:
        movwf   AtaRegister
        bcf             status,rp1
        bsf             status,rp0
        movlw   PORTB_CONFIG2
        movwf   trisb
        movlw   PORTC_CONFIG2
        movwf   trisc
        bcf             status,rp1
        bcf             status,rp0
        rlf             AtaRegister,w
        iorlw   0x30
        movwf   porta
        call    Delay1ms
        movf    AtaDataLsb,w
        movwf   portb
        clrf    portc
        call    Delay1ms
        bcf             porta,ATA_IOW
        call    Delay1ms
        bsf             porta,ATA_IOW
        call    Delay1ms
        return



AtaReadRegister:
        movwf   AtaRegister
        bcf             status,rp1
        bsf             status,rp0
        movlw   PORTB_CONFIG1
        movwf   trisb
        movlw   PORTC_CONFIG1
        movwf   trisc
        bcf             status,rp1
        bcf             status,rp0
        rlf             AtaRegister,w
        iorlw   0x30
        movwf   porta
        call    Delay1ms
```

```
        bcf             porta,ATA_IOR
        call    Delay1ms
        movf    portb,w
        movwf   AtaDataLsb
        movf    portc,w
        movwf   AtaDataMsb
        call    Delay1ms
        bsf             porta,ATA_IOR
        call    Delay1ms
        return



AtaReadPacketEmpty:

AtaRe1:
        movlw   ATA_REG_SECTORCOUNT
        call    AtaReadRegister
        movf    AtaDataLsb,w
        andlw   0x03
        xorlw   0x03
        btfss   status,z
        goto    AtaRe1

AtaReadNachamble:
        call    Delay200ms

AtaRe2:
        movlw   ATA_REG_STATUS
        call    AtaReadRegister
        movf    AtaDataLsb,w
        andlw   0xc8
        xorlw   0x40
        btfss   status,z
        goto    AtaRe2

        movf    AtaDataLsb,w
        xorlw   0x50
        return


#ifdef LCD
AtaReadIdentifyDrive:

        return
#endif



#ifdef LCD
AtaReadPacketInquiry:

        return
#endif
```

```
AtaReadPacketSubChannel:
        call    AtaReadPreamble

        bcf             porta,ATA_IOR
        movf    portc,w
        movwf   TrackStatus
        call    Delay10ms
        bsf             porta,ATA_IOR
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
        bsf             porta,ATA_IOR
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
        bsf             porta,ATA_IOR
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
        movf    portb,w
        movwf   TrackCurrentNr
        bsf             porta,ATA_IOR
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
        movf    portc,w
        movwf   TrackCurrentMSF_M
        bsf             porta,ATA_IOR
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
        movf    portb,w
        movwf   TrackCurrentMSF_S
        movf    portc,w
        movwf   TrackCurrentMSF_F
        bsf             porta,ATA_IOR
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
        bsf             porta,ATA_IOR
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
        bsf             porta,ATA_IOR
        call    Delay10ms
        call    AtaReadNachamble
        return



AtaReadPacketToc:
        call    AtaReadPreamble

        bcf             porta,ATA_IOR
        call    Delay10ms
        bsf             porta,ATA_IOR
        call    Delay10ms
```

```
        bcf             porta,ATA_IOR
        call    Delay10ms
        movf    portb,w
        movwf   TrackFirstNr
        movf    portc,w
        movwf   TrackLastNr
        bsf             porta,ATA_IOR

AtaRt1:
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
        bsf             porta,ATA_IOR
        call    Delay10ms


        bcf             porta,ATA_IOR
        call    Delay10ms
        movf    portb,w
        movwf   TrackTempNr
        bsf             porta,ATA_IOR
        call    Delay10ms

        movf    TrackTempNr,w
        xorwf   TrackStartNr,w
        btfsc   status,z
        goto    AtaRSave1
        movf    TrackTempNr,w
        xorlw   0xaa
        btfsc   status,z
        goto    AtaRSave2

        bcf             porta,ATA_IOR
        call    Delay10ms
        bsf             porta,ATA_IOR
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
        bsf             porta,ATA_IOR
        goto    AtaRt1

AtaRSave1:
        bcf             porta,ATA_IOR
        call    Delay10ms
        movf    portc,w
        movwf   TrackStartMSF_M
        bsf             porta,ATA_IOR
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
        movf    portb,w
        movwf   TrackStartMSF_S
        movf    portc,w
        movwf   TrackStartMSF_F
        bsf             porta,ATA_IOR
        call    Delay10ms
        goto    AtaRt1

AtaRSave2:
        bcf             porta,ATA_IOR
        call    Delay10ms
        movf    portc,w
        movwf   TrackLeadOutMSF_M
        bsf             porta,ATA_IOR
        call    Delay10ms

        bcf             porta,ATA_IOR
        call    Delay10ms
```

```
        movf    portb,w
        movwf   TrackLeadOutMSF_S
        movf    portc,w
        movwf   TrackLeadOutMSF_F
        bsf             porta,ATA_IOR
        call    Delay10ms
        call    AtaReadNachamble
        return



AtaReadPreamble:

AtaRp1:
        movlw   ATA_REG_SECTORCOUNT
        call    AtaReadRegister
        movf    AtaDataLsb,w
        andlw   0x03
        xorlw   0x02
        btfss   status,z
        goto    AtaRp1

        bcf             status,rp1
        bsf             status,rp0
        movlw   PORTB_CONFIG1
        movwf   trisb
        movlw   PORTC_CONFIG1
        movwf   trisc
        bcf             status,rp1
        bcf             status,rp0

        bcf             porta,ATA_A0
        bcf             porta,ATA_A1
        bcf             porta,ATA_A2

        END
```