Palestine Polytechnic University

College of Engineering & Technology
Electrical & Computer Engineering Department

Graduation Project

Candy Bar Vending Machine

Project Team
Bilal Tamimi
Motaz Tamimi
Nadeem Abu Khalef

Project Supervisor
Dr. Salman Tlahmeh

Hebron-Palestine

June 2005

جامعة بوليتكنيك فلسطين

الخليل – فلسطين

كلية الهندسة والتكنولوجيا

دائرة الهندسة الكهربائية و الحاسوب

اسم المشروع:

أسماء الطلبة

بلال التميمي معتز التميمي

نديم ابو خلف

بناء على نظام كلية الهندسة و التكنولوجيا و إشراف ومتابعة المشرف المباشر على المشروع و موافقة أعضاء اللجنة الممتحنة تم تقديم هذا المشروع إلى دائرة الهندسة الكهربائية و الحاسوب و ذلك للوفاء بمتطلبات درجة البكالوريوس في الهندسة تخصص هندسة أنظمة الحاسوب

توقيع المشرف

د.سلمان التلاحمة .....

توقيع اللجنة الممتحنة

د.سلمان التلاحمة .....

م. .....

توقيع رئيس ا لدائرة

د. غاندي المناصره .....

Dedication

To our parents, to our supervisor, to Palestine Polytechnic University we dedicate this project .Thanks for your encouragement.

Acknowledgment

To our supervisor Dr. Salman Tlahmeh for his guidance, support and encouragement, to every one who provides us with helpful suggestions.

Abstract

The project is a self Independent Candy Bar Vending Machine which works by the coins .It consists of three main parts Coin Recognizer which is the unit than recognizes the coins .Mechanical part includes stepper motor and springs to deliver the Candy to the user, Microcontroller Board which controls the whole system.

This project aims to design a complete practical vending machine which can be implemented in the future.

هذا المشروع عبارة عن ماكنة بيع شوكولا تعمل بالعملة النقدية التي توضع فيها،يتكون المشروع من ثلاث اجزاء رئيسية ،الجزء الأول عبارة عن وحدة تمييز القطع النقدية ،الجزء الثاني عبارة عن وحدة مكانيكية تتكون من زنبركات و ماتورات تعمل هذه الوحدة على ايصال البضاعة المستخدم،والجزء الثالث هو وحدة التحكم في الالتز

يهدف هذا المشروع الى تصميم نظام عملي كامل قابل للتطبيق في المستقبل.

Table of Contents

Subject	Page
CHAPTER INTRODUCTION	
1.1 Introduction	1
1.2 Proposed Project	2
1.3.1 Impact on society	2
1.3.2 Impact on life	2
1.3.3 Impact on Economy	2
1.3.4 Impact on health	2
1.4 Literature review	4
1.4.1 Development of vending Machine	4
1.5 Project Estimated Cost	4
1.6 Time Plan	7
1.7 Report Contents	8
CHAPTER2 THREORITICAL BACKGROUND	
2.1 Introduction	10
2.2 project components	10
2.2.1 Cabinet Architecture	11
2.2.1.1 Cabinet Design	11
2.2.2 Springs	12
2.2.3 Money Depositing	12

2.2.3.1 Photo Diode 12

2.2.3.2 Load Cell 13

2.2.3.3 Size Recognizer 14

2.2.4 Item Delivery System 14

2.2.4.1 Stepper Motor 14

2.2.5 8051 Microcontroller 19

CHAPTER 3 DESIGN CONCEPT

3.1 Introduction 27

3.2 Objectives 27

3.3 Block Diagrams 27

3.4 How the system works 28

CHAPTER 4 HARDWARE DESIGN

4.1 Introduction 32

4.1 Cabinet Design 32

4.2.1 Cabinet Dimension 32

4.2.2 Cabinet Description 32

4.3 Circuits Design 37

4.3.1 Stepper Motors 37

4.3.2 Coin Recognizer 39

4.3.3 Switches 40

CHAPTER 5 SOFTWARE DESIGN

5.1 Overview 42

2.2.3.1 Photo Diode 12

2.2.3.2 Load Cell 13

2.2.3.3 Size Recognizer 14

2.2.4 Item Delivery System 14

2.2.4.1 Stepper Motor 14

2.2.5 8051 Microcontroller 19

CHAPTER 3 DESIGN CONCEPT

3.1 Introduction 27

3.2 Objectives 27

3.3 Block Diagrams 27

3.4 How the system works 28

CHAPTER 4 HARDWARE DESIGN

4.1 Introduction 32

4.1 Cabinet Design 32

4.2.1 Cabinet Dimension 32

4.2.2 Cabinet Description 32

4.3 Circuits Design 37

4.3.1 Stepper Motors 37

4.3.2 Coin Recognizer 39

4.3.3 Switches 40

CHAPTER 5 SOFTWARE DESIGN

5.1 Overview 42

2.2.3.1 Photo Diode 12

2.2.3.2 Load Cell 13

2.2.3.3 Size Recognizer 14

2.2.4 Item Delivery System 14

2.2.4.1 Stepper Motor 14

2.2.5 8051 Microcontroller 19

CHAPTER 3 DESIGN CONCEPT

3.1 Introduction 27

3.2 Objectives 27

3.3 Block Diagrams 27

3.4 How the system works 28

CHAPTER 4 HARDWARE DESIGN

4.1 Introduction 32

4.1 Cabinet Design 32

4.2.1 Cabinet Dimension 32

4.2.2 Cabinet Description 32

4.3 Circuits Design 37

4.3.1 Stepper Motors 37

4.3.2 Coin Recognizer 39

4.3.3 Switches 40

CHAPTER 5 SOFTWARE DESIGN

5.1 Overview 42

5.2 Functional Requirement 42

5.3 Non Functional Requirement 42

5.4 Algorithm 42

5.5 Software Code 48

CHAPTER 6 IMPLEMENTION AND TESTING

6.1 Introduction 55

6.2 Hardware implementation and Testing 55

CHAPTER 7 CONCLSIONS and FUTURE WORK

7.1 Conclusion 59

7.2 Future Work 59

List of Figures

Figure **Pages**

Figure(2.1): System components 10

Figure(2.2): Cabinet Shape 11

Figure(2.3): Strain Gage 14

Figure(2.4): Stepper Motor 15

Figure(2.5): Unipolar Stepper Motor 15

Figure(2.6): Reversal current of Unipolar Motor 16

Figure(2.7): Conceptual Model of Unipolar Stepper Motor 16

Figure(2.8): Bipolar Stepper Motor 18

Figure(2.9): Reversal current of Bipolar Stepper Motor 18

Figure(2.10): Conceptual Model of Bipolar Stepper Motor 19

Figure(2.11): 8051 Microcontroller 20

Figure(3.1): General Block Diagram 27

Figure(3.2): Block Diagram 28

Figure(3.3): System process Model 29

Figure (3.4): Data Flow 30

Figure (4.1): Vending Machine body shape 33

Figure (4.2): Side view of vending machine 34

Figure (4.3): Front view of vending machine 35

Figure(4.4): Front view of vending machine 36

Figure (4.5): Stepper motor interfacing circuit 38

Figure(4.6): coin recognizer circuit 39

Figure (4.7): Switches circuit 40

Figure(5.1): Coin Recognizer Flow Chart 44

Figure(5.2): Switches Flow Chart 45

Figure(5.3): Stepper motor Flow chart 46

Figure (5.4) :System Flow chart 47

Figure (6.1): Fixed Point of the coin points 49

List of Tables

Table **Page**

Table(1.1):Estimated Cost 6

Table(1.2): Time Plan 7

Table(2.1) Table of stepping sequences 16

Table(2.2) Table of stepping sequences of Bipolar Model 19

Chapter One
INTRODUCTION

1.1 Introduction
1.2 Proposed Project
1.3 Project Importance
1.4 Literature Review
1.5 Project Estimated Cost
1.6 Time Plan
1.7 Report Contents

1.1 Introduction

Vending Machine is a self independent machine that is used to sell many kinds of products such as cigarettes, candy, soft drinks ....etc. It becomes very important and, comfort way to sell the products .The project is aimed to design a candy vending machine that has the ability to accept money (coins) from the user and deliver the selected candy to the user if the money is enough otherwise it will give the user the chance to insert more money .In addition for administrator mode which allow the administrator to change the prices and the software of the vending machine.

1.2 Proposed project

This project aims to design a Candy Bar Vending Machine which will sell specific kinds of candy .The project will give a simple practical design that can be implemented in the future in our region .This design of the machine include designing of a cabinet box , coin recognizer unit, mechanical delivery unit.

1.3 Project importance:

The following points explain the impacts of the vending machine on many aspects of life and clarify the importance of the vending machine by giving many examples.

1.3.1 Impact on Society

Vending machines have evolved from dispensing postcards in the early 1880s to dispensing anything we can think of in present day. We can now purchase coffee, canned drinks, cup noodles, chocolate, newspapers, medicine - the list is inexhaustible. They have changed our lives in many ways.

1.3.2 Impact on life

We are now living in a fast-paced, automated world whereby we will like everything to be fast, effective and convenient. We have benefited from the usage of vending

machines around us to an extent that we are not even consciously aware of their importance in our everyday life.

1.3.3 Impact on Economy

Vending machines can be a profitable business, in areas like manufacturing, maintenance, distributing and even companies that specializes in outsourcing locations for vending machines installation. These companies have allowed the creation of many jobs, from the factory worker attaching parts of the vending machines to the technician providing maintenance support to the deliveryman topping up the products in the machines. All these jobs created are sources of income for those employed.

It is noteworthy to point out that in Japan alone, there are more than 5.6 million vending machines in a country of about 126 million people. The sales of products dispensed through vending machines in 2000 exceeded 7.1 trillion yen ($57.6 billion). Given the high sales figures in Japan, there are countless economical opportunities available in the vending industry for anyone to explore.

On an interesting note, there has been a growing emphasis by many countries, on the need for innovation and creativity. The vending industry thrives on such factors because any breakthrough in the way a machine dispense its products can be translated into high profits. For example, the Japanese unit of Coca-Cola has started the service of providing mode phone subscribers the cashless purchase of beverages and to deliver advertisements and other information on a display installed in the Coke's vending machines. Such a complementary service is a win-win situation for both the companies and the consumers. For the consumers, it gives them more variety of choices in the way they purchase drinks. For the companies, such innovation will make them stand out from the rest and in the long run; they can increase their market share as well.

As long as there is a public demand for vending machines, the vending industry will continue to innovate and bring cutting-edge technology to the machines to serve the ever-growing needs of consumers today.

1.3.Impact on human health:

Vending machines have allowed food and beverage items to be dispensed to consumers in a more cleaner and hygienic manner, due to the fact that the items are dispensed from within an unexposed and technically sterile environment, which minimize exposure to pollutants and microbes as compared to if handled by human hands. Since the birth of the world's first commercial vending machines till now, there have no major report of any casualties resulting from consuming products bought from vending machines. In addition, vending machines are maintained and cleaned by maintenance workers on a regular basis, hence ensuring a relatively high standard of cleanliness throughout its operation.

1.4 Literature review:

In Palestine Polytechnic University there is no previous studies about the vending machine, the following section will show discuss about the vending machine including commercial vending machine and how it is started and evolved by the time.

1.4.1 Development of Vending Machines:

Vending machines, otherwise known as "automatic retailing" machines, have a long history. The earliest documented existence of the world's first vending machine dates back to 215 B.C. in Alexandra, ancient Egypt, where the Greek mathematician Hero, invented a device to dispense holy water in temples for ritual cleansing.

Worshippers would insert coins through an opening at the top of the device, which would then fall on and depresses a pan below.

The depression of the pan would in turn lift up a valve and allowed water to flow out. As the pan tilts, the coins would fall off and the reduction in weight to the pan forces the pan back to its original position. This action then causes the valve move in the opposite direction and shut down the channel for the water to flow.

Vending machines were used widely throughout Europe in the late 1800s. The world's first commercial coin-operated machine was introduced in London, England in the 1880s, which dispensed post cards. Around this period, an English book publisher, Richard Carlisle, invented a vending machine that sold books to the public.

However, vending machines only became popular when they were introduced into the United States in the late 1800s and at the turn of the 1900s. In 1888, the Thomas Adams Gum Company introduced the first machines on the elevated subway platforms of New York City, selling the company's Tutti-Frutti brand of chewing gums. In 1897, another gum manufacturing company, Pulvar Manufacturing Company, brought a whole new revolution into the designing of vending machines, by adding animated figures onto its own gum vending machines, in order to attract more patrons. The familiar round spherical gumball vending machines, introduced in 1907, was also a result of the new machine design revolution

The rise in popularity in vending machines in the United States was so overwhelming that people virtually dispensed almost anything from machines. Such items include cigarettes, postcards, stamps, candies, coffee and snacks. In 1902, the Horn and Hardart Baking Company introduced a new concept to the use of vending machines by setting up a completely coin-operated restaurant in Philadelphia. Patrons would purchase their food and beverages entirely from vending machines and consume them on tables and chairs provided, just as they would in normal restaurants. It stayed opened until 1962, where unfortunately, the dawn of the fast food craze in America put them out of business.

1.5 Project Estimated Cost

Table (1.1) shows the estimated cost of vending machine.

Table (1.1): Estimated Cost

	Component	units	Estimated cost ($)
1-	8051 Microcontroller Board (50 I/O lines, 32k SRAM, High speed rates baud) + LCD + Charging fees+Taxes	1	$240
2-	16 Darlington Transistor	16	$16
3-	16 Opt couplers	16	$10
4_	Stepper motors	4	$100
5_	Springs	4	$10
6_	Photo transistors	4	$4
7-	Photo Diode	4	$4
8-	Cabinet	1	$100
9-	ICs Bases	32	$10
10	Boards	3	$10
11	Cables and Connectors		$8
12-	Power Supplies (12V), (26V)	2	$10
	Total		$522

1.6 Time Plan

Table (1.2) shows the time Plan suggested for the project

Table (1.2): Time Plan

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
Data Gathering															
Data Analysis															
System Requirement specification															
System Design															
Implementation															
Testing															
Documentation															

1.7 Report Contents

Report consists of seven chapters; the following is a brief description of the topics that are covered in each chapter.

Chapter 1 presenting general idea about the project and its importance, also the literature review, estimated cost, time plan are shown.

Chapter 2 introduces and illustrates the idea of the project, describes the components used in the project.

Chapter 3 details the design concepts, introduces project objectives, shows the general block diagram of the system and explains how system works.

Chapter 4 outlines formal procedure for design, discuss design options and justify those chosen for the project.

Chapter 5 "Software System Design": shows out the system software flow charts, and subroutine.

Chapter 6 "Implementation and Testing "discusses the steps followed to implement and test our own design.

Chapter 7 "Conclusions and Future Work" provides many suggestions to expand this project and summarizes the project conclusions.

Chapter Two
Theoretical Background

2.1 Introduction

2.2 Project Components

2.1 Introduction

This chapter focuses on the theoretical subjects related to the main idea of the project, and information about the components used in the project and also some shape designs about the available commercial vending machines.

2.2 Project components

This project has many components which are listed in the following section as shown in the fig (2.1), these components are:

- Cabinet Box
- Money Depositing (Coin Recognizer)
- Item Delivery unit (Stepper Motors + springs)
- Microcontroller 8051

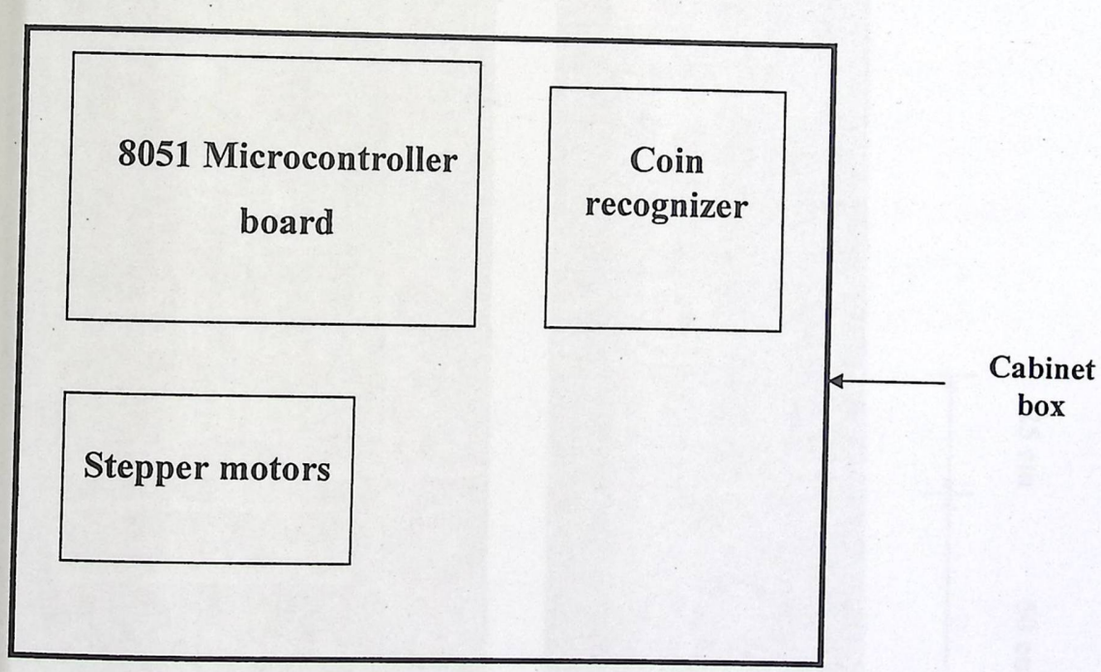


Fig (2.1) System components

2.2.1 Cabinet box

The Cabinet is a box of wood or aluminum fixed with the ground and has a height suitable for the users with front of glass in order to show the products. The cabinet as shown in the figure l has switches in order to select the product.

2.2.1.1 Cabinet box shape

There are many shapes alternatives for the cabinet of vending machine and these shapes are specified in Figure (2.2).

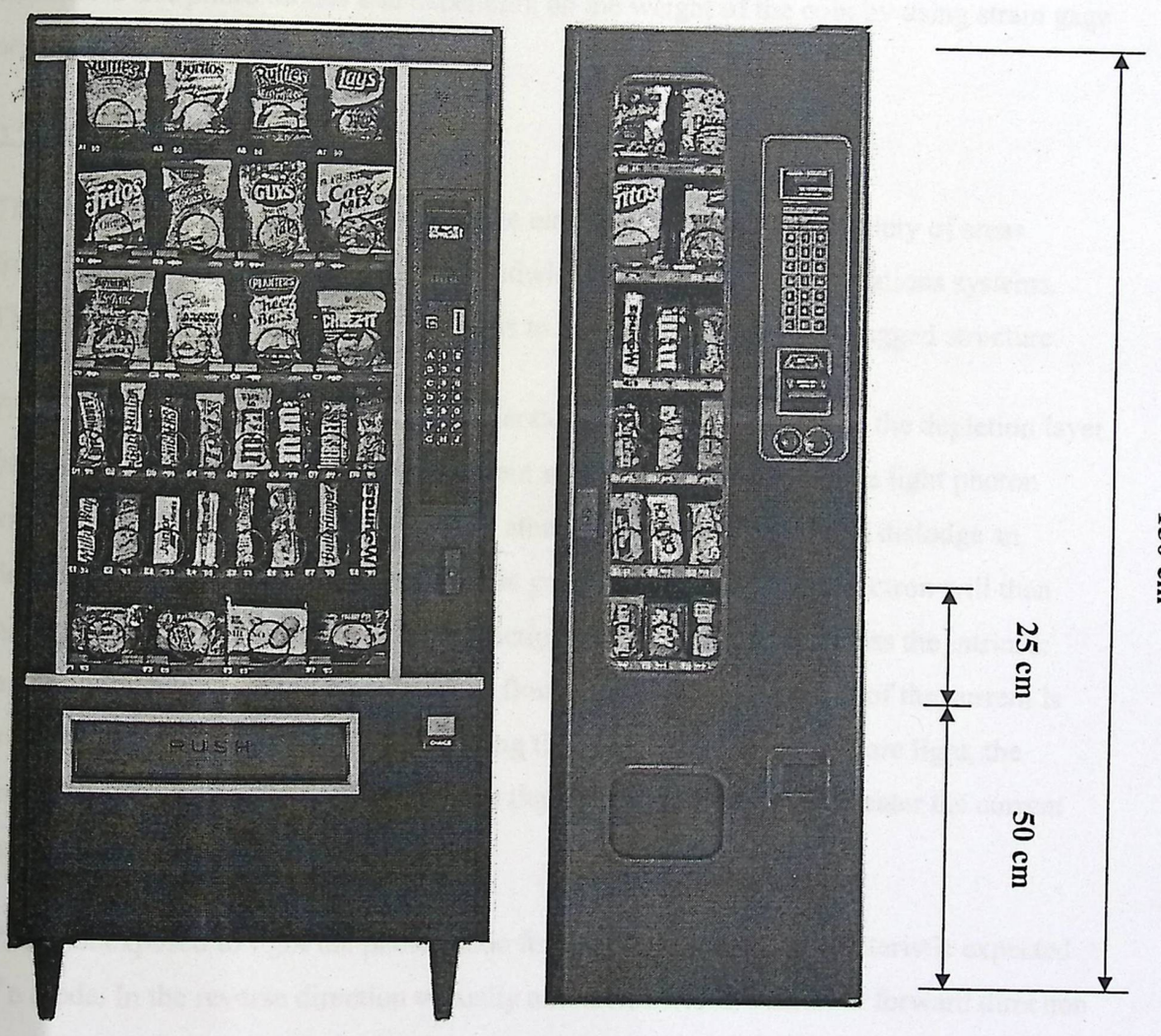


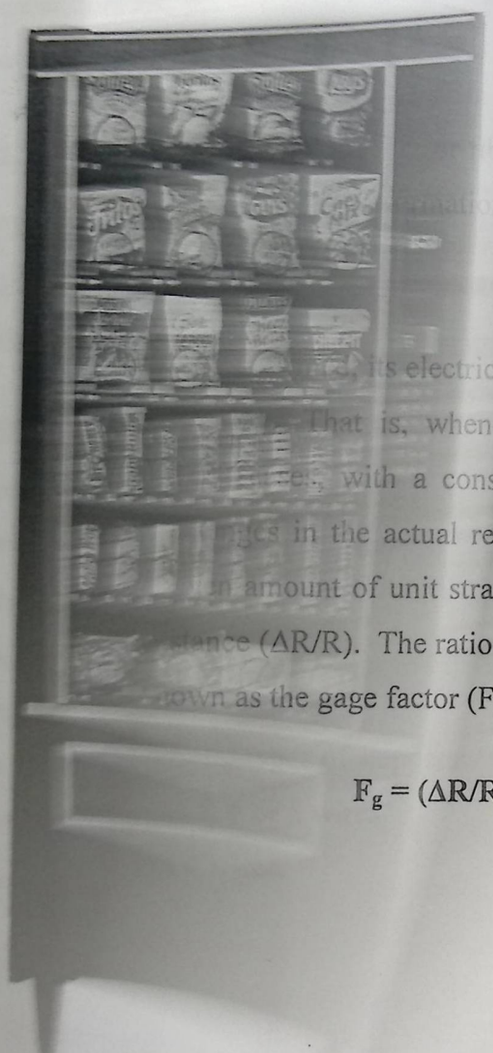
Fig (2.2): Cabinet Shape

2.2.1 Cabinet box

The Cabinet is a box of wood or aluminum fixed with the ground and [illegible] suitable for the users with front of glass in order to show the products, and shown in the figure 1 has switches in order to select the product.

2.2.1.1 Cabinet box shape

There are many shapes alternatives for the cabinet of vending machine [illegible] are specified in Figure (2.2).



[illegible] its electrical resistance changes. This fact is explained partly [illegible] That is, when a conductor is stretched lengthwise, its cross-[illegible], with a consequent increases in resistance. It is also partly [illegible] in the actual receptivity of the gage material when subjected to [illegible] amount of unit strain (ΔL/L), the gage will undergo a corresponding [illegible] (ΔR/R). The ratio of the unit change in resistance to the unit change [illegible] as the gage factor (F_g) of the gage:

$$F_g = (\Delta R/R) / (\Delta L/L)$$

greatest effect is seen in the reverse direction. Here the largest changes are noticed, and the normal forward current does not mask the effects due to the light.

2.2.3.2 Load cell (Strain Gage sensor)

is a sensor which is used to specify the weight of the coin it will be affected by any change of the weight and this change leading to change in the voltage which will be very small (mV), so it need to be amplified by an Amplification circuit using TL081.The resistance strain gage is an electrical sensing device that varies its resistance as a linear function of the strain experienced by the structural surface to which it is bonded. Strain is the deformation of a solid material as the result of applied forces (internal or external), and is normally expressed in units of microinches per inch (or microstrain).

A typical strain gage consists of a conductive grid pattern of etched metallic foil, mounted on a thin base of epoxy or fiberglass. It can then be bonded to a surface in such a way that any subsequent deformation of the surface produces a like deformation of the gages.

When the gage is deformed, its electrical resistance changes. This fact is explained partly by simple geometry. That is, when a conductor is stretched lengthwise, its cross-sectional area decreases, with a consequent increases in resistance. It is also partly explained by changes in the actual receptivity of the gage material when subjected to strain. For a given amount of unit strain ($\Delta L/L$), the gage will undergo a corresponding change in resistance ($\Delta R/R$). The ratio of the unit change in resistance to the unit change in length is known as the gage factor (F_g) of the gage:

$$\mathbf{F_g = (\Delta R/R) / (\Delta L/L)}$$

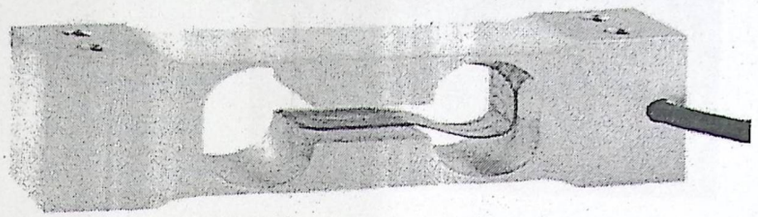


Fig (2.3): Strain Gage Sensor

2.2.3.3 Size and Diameter Recognizer

The Entry or the slot of the coin will be designed to be exactly the same as the coin size and diameter so any attempt to enter a coin larger than the size of the shekel will not work.

2.2.4 Item Delivery Unit

This part of the system is responsible for delivering the products to the user it consists of stepper motors, and their driver circuits and springs.

2.2.4.1 Stepper Motors:

There are two basic types of steppers:

1. Unipolar, has 5, 6 or 8 wires.
2. Bipolar, has 4 wires.

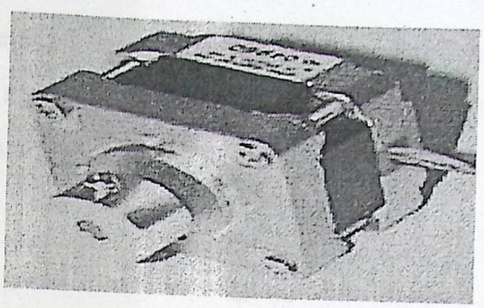


Figure (2.4): stepper motor

1. Unipolar Stepper Motor

The Unipolar Stepper motor has two coils, simple lengths of wound wire. The coils are identical and are not electrically connected. Each coil has a center tap a wire coming out from the coil that is midway in length between its two terminals.

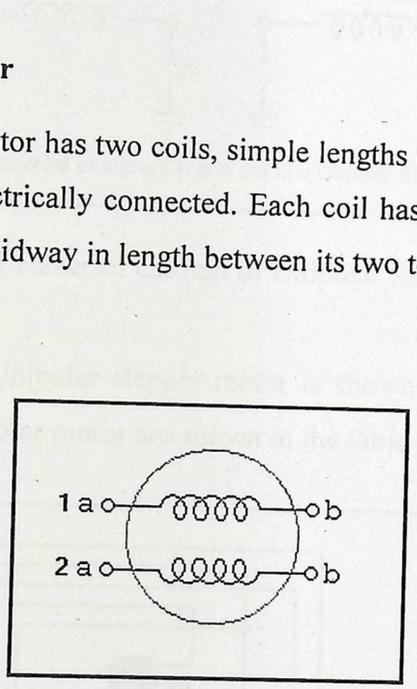


Figure (2.5): Unipolar Stepper Motor.

You can identify the center tap by measuring resistance with a suitable ohmmeter (capable of measuring low resistance <10 ohm), the resistance from a terminal to the center tap is half the resistance from the two terminals of a coil.

When talking about the motor control circuitry, we see that the current flowing through a coil produces a magnet field, which attracts a permanent magnet rotor, which is connected to the shaft of the motor. The basic principle of stepper control is to reverse the direction of current through the two coils of a stepper motor, in sequence, in order to influence the rotor. Since there are two coils and two directions, that give us a possible 4-phase sequence. The internal arrangement of the motor is quite complex. The winding

and core repeated around the perimeter of the motor many times. The rotor is advanced only a small angle, either forward or reverses and the 4-phase sequence is repeated many times before a complete revolution occurs.

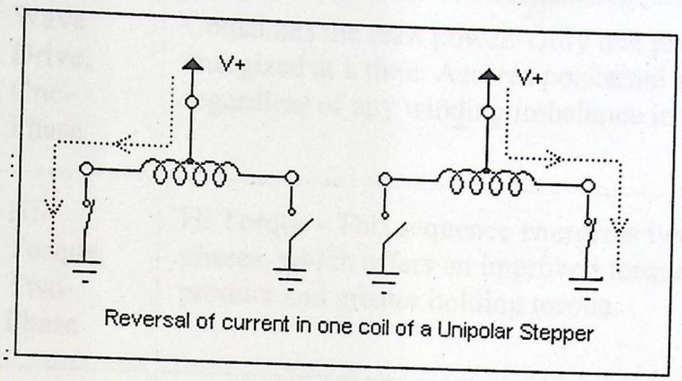


Figure (2.6): Reversal Current of Unipolar Stepper Motor.

The conceptual model of Unipolar stepper motor is shown in the figure (2.7), and the stepping sequences of Unipolar motor are shown in the table (2.1).

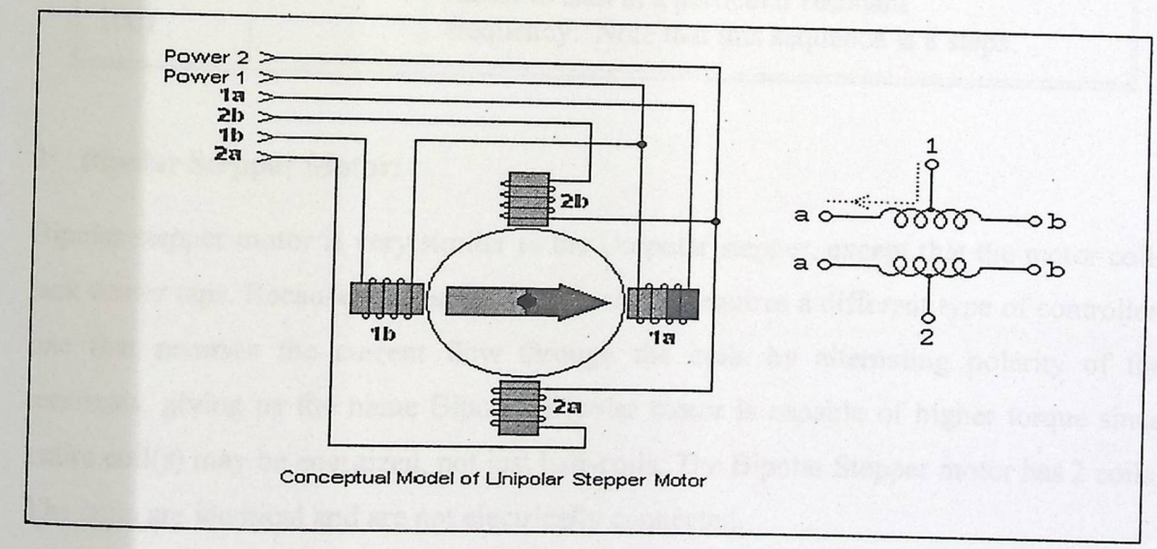


Figure (2.7): Conceptual Model of Unipolar Stepper Motor

Table (2.1): Table of Stepping Sequences

Sequence	Name	Description
	Wave Drive, One-Phase	Consumes the least power. Only one phase is energized at a time. Assures positional accuracy regardless of any winding imbalance in the motor.
0011 0110 1100 1001	Hi-Torque, Two-Phase	Hi Torque - This sequence energizes two adjacent phases, which offers an improved torque-speed product and greater holding torque.
0001 0011 0010 0110 0100 1100 1000 1001	Half-Step	Half Step - Effectively doubles the stepping resolution of the motor, but the torque is not uniform for each step. (Since we are effectively switching between Wave Drive and Hi-Torque with each step, torque alternates each step.) This sequence reduces motor resonance which can sometimes cause a motor to stall at a particular resonant frequency. Note that this sequence is 8 steps.

2 Bipolar Stepper Motor:

Bipolar stepper motor is very similar to the Unipolar stepper, except that the motor coils lack center taps. Because of this, the bipolar motor requires a different type of controller, one that reverses the current flow through the coils by alternating polarity of the terminals, giving us the name Bipolar. Bipolar motor is capable of higher torque since entire coil(s) may be energized, not just half-coils. The Bipolar Stepper motor has 2 coils. The coils are identical and are not electrically connected.

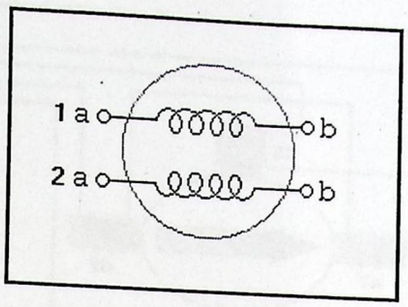


Figure (2.8): Bipolar Stepper Motor.

When we talk about the Motor Control Circuitry we see that the Bipolar controller must be able to reverse the polarity of the voltage across either coil, so current can flow in both directions. And, it must be able to energize these coils in sequence

.

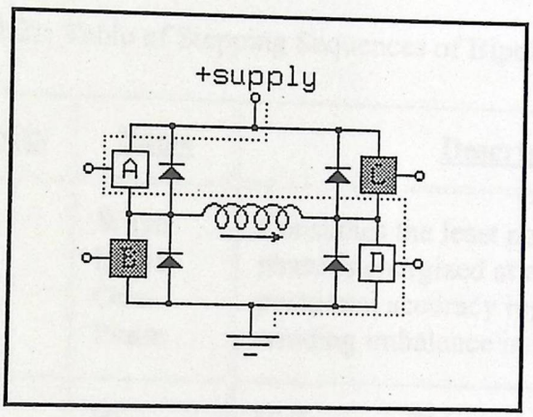


Figure (2.9): Reversal Current of Bipolar Stepper Motor.

This circuit is called an H-Bridge, because it resembles a letter "H". The current can be reversed through the coil by closing the appropriate switches - AD to flow one direction then BC to flow the opposite.

The Conceptual model of Bipolar stepper motor is shown in the figure (2.9), and stepping sequences of the bipolar motor are shown in the table (2.2).

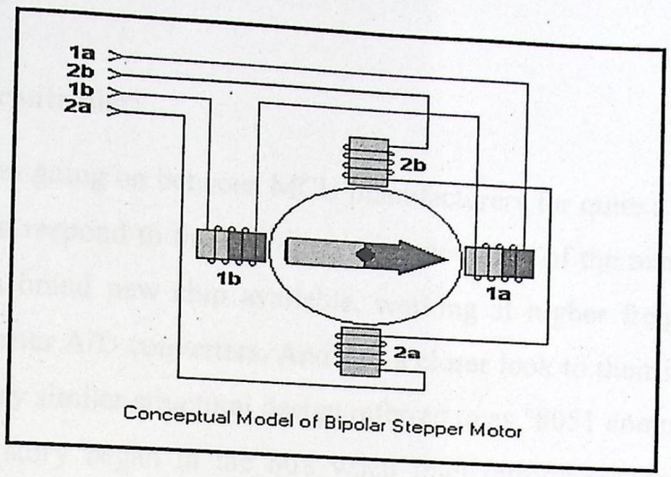


Figure (2.10): Conceptual Model of Bipolar Stepper Motor

Table (2.2): Table of Stepping Sequences of Bipolar Model

Sequence	Polarity	Name	Description
0001 0010 0100 1000	---+ --+- -+-- +---	Wave Drive, One-Phase	Consumes the least power. Only one phase is energized at a time. Assures positional accuracy regardless of any winding imbalance in the motor.
0011 0110 1100 1001	--++ -++- ++-- +--+	Hi-Torque, Two-Phase	Hi Torque - This sequence energizes two adjacent phases, which offers an improved torque-speed product and greater holding torque.
0001 0011 0010 0110 0100 1100 1000 1001	---+ --++ --+- -++- -+-- ++-- +--- +--+	Half-Step	Half Step - Effectively doubles the stepping resolution of the motor, but the torque is not uniform for each step. (Since we are effectively switching between Wave Drive and Hi-Torque with each step, torque alternates each step.) This sequence reduces motor resonance which can sometimes cause a motor to stall at a particular resonant frequency. Note that this sequence is 8 steps.

2.2.5 8051 Microcontroller:

A struggle has been going on between MCU manufacturers for quite a long time, each of them trying to best respond to the ever-increasing demands of the market. Every couple of days there is a brand new chip available, working at higher frequency, with more memory or with better A/D converters. And yet, a closer look to their interior reveals the same or at least very similar structural design referred to as "8051 compatibility". What is it all about? The story began in the 80's when Intel introduced their microcontroller family MCS 8051 to the market. Although this family had quite limited capabilities by today's notions, it quickly captivated the world and became the standard for what is today understood as 'microcontroller'. The most significant cause for such a success can be found in the cleverly chosen configuration which can satisfy a diversity of needs, yet allowing for continuous upgrades (in form of new controllers). In a brief period of time, a decent amount of software has been developed for 8051, making further changes of the hardware core simply uneconomical. Consequently, there is a variety of MCUs available today, basically just the upgraded 8051 models. What exactly makes this microcontroller so special and universal that it is still manufactured by all the major companies, just under a different label?

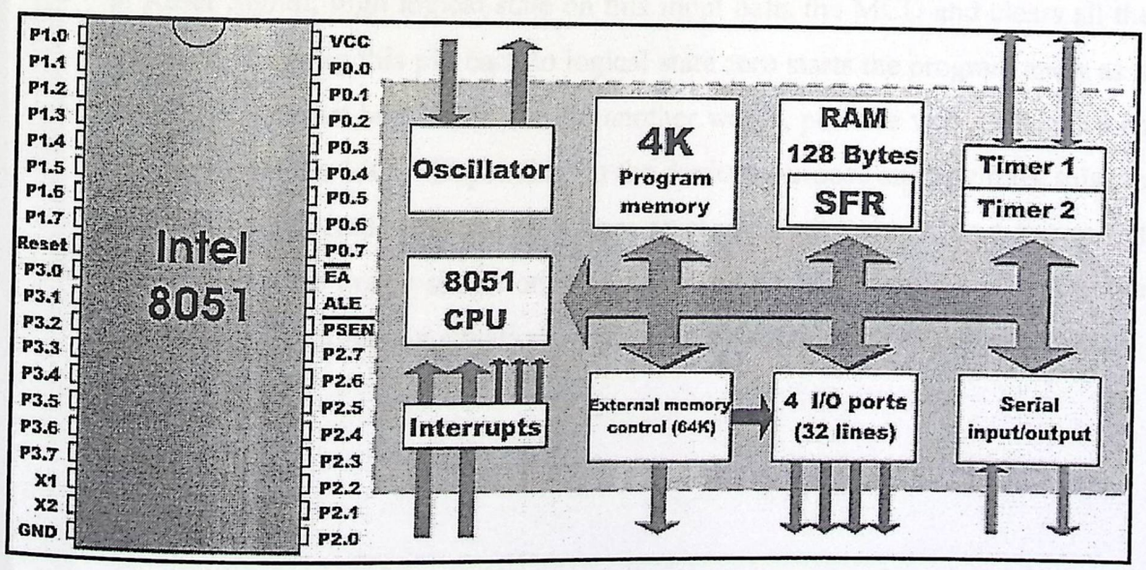


Fig (2.11): 80851 Microcontroller

As can be seen in the Fig (2.10), there is nothing particularly remarkable about MCU 8051:

- 4 kilobytes of ROM is neither too little nor too much.
- 128 bytes of RAM (SFR registers included) can satisfy the basic needs, but is not really astounding.
- 4 ports totaling 32 I/O lines, are usually sufficient for connecting to the environs and are by no means luxury.

Obviously, 8051 configuration is intended to satisfy the needs of programmers developing the controlling devices and instruments. This is one part of its key to success: there is nothing missing, yet there is no lavishness; it is meant for the average user. The other clue can be found in the organization of RAM, Central Processor Unit (CPU), and ports - all of which maximally utilize the available resources and allow further upgrades.

Pins On The Case

- 1–8: Port 1; Each of these pins can be used as either input or output according to your needs. Also, pins 1 and 2 (P1.0 and P1.1) have special functions associated with Timer 2.
- 9: Reset Signal; high logical state on this input halts the MCU and clears all the registers. Bringing this pin back to logical state zero starts the program anew as if the power had just been turned on. In another words, positive voltage impulse on this pin resets the MCU. Depending on the device's purpose and environs, this pin is usually connected to the push-button, reset-upon-start circuit or a brown out reset circuit .The image shows one simple circuit for safe reset upon starting the controller. It is utilized in situations when power fails to reach its optimal voltage.

- **10-17: Port 3**; As with Port 1, each of these pins can be used as universal input or output. However, each pin of Port 3 has an alternative function:
  - Pin 10: RXD - serial input for asynchronous communication or serial output for synchronous communication.
  - Pin 11: TXD - serial output for asynchronous communication or clock output for synchronous communication
  - Pin 12: INT0 - input for interrupt 0
  - Pin 13: INT1 - input for interrupt 1
  - Pin 14: T0 - clock input of counter 0
  - Pin 15: T1 - clock input of counter 1
  - Pin 16: WR - signal for writing to external (add-on) RAM memory
  - Pin 17: RD - signal for reading from external RAM memory
- **18-19: X2 and X1**; Input and output of internal oscillator. Quartz crystal controlling the frequency commonly connects to these pins. Capacitances within the oscillator mechanism (see the fig 2.14) are not critical and are normally about 30pF. Instead of a quartz crystal, miniature ceramic resonators can be used for dictating the pace. In that case, manufacturers recommend using somewhat higher capacitances (about 47 pF). New MCUs work at frequencies from 0Hz to 50MHz+.
- **20: GND**; Ground
- **21- 28: Port 2**; If external memory is not present, pins of Port 2 act as universal input/output. If external memory is present, this is the location of the higher address byte, i.e. addresses A8 – A15. It is important to note that in cases when not all the 8 bits are used for addressing the memory (i.e. memory is smaller than 64kB), the rest of the unused bits are not available as input/output.
- **29: PSEN**; MCU activates this bit (brings to low state) upon each reading of byte (instruction) from program memory. If external ROM is used for storing the program, PSEN is directly connected to its control pins.

- **30: ALE**; Before each reading of the external memory, MCU sends the lower byte of the address register (addresses A0 – A7) to port P0 and activates the output ALE. External register (74HCT373 or 74HCT375 circuits are common), memorizes the state of port P0 upon receiving a signal from ALE pin, and uses it as part of the address for memory chip. During the second part of the mechanical MCU cycle, signal on ALE is off, and port P0 is used as Data Bus. In this way, by adding only one cheap integrated circuit, data from port can be multiplexed and the port simultaneously used for transferring both addresses and data.
- **31: EA**; Bringing this pin to the logical state zero (mass) designates the ports P2 and P3 for transferring addresses regardless of the presence of the internal memory. This means that even if there is a program loaded in the MCU it will not be executed, but the one from the external ROM will be used instead. Conversely, bringing the pin to the high logical state causes the controller to use both memories, first the internal, and then the external (if present).
- **32-39: Port 0**; Similar to Port 2, pins of Port 0 can be used as universal input/output, if external memory is not used. If external memory is used, P0 behaves as address output (A0 – A7) when ALE pin is at high logical level, or as data output (Data Bus) when ALE pin is at low logical level.
- **40: VCC**; Power +5V

Input – Output (I/O) Ports

Every MCU from 8051 family has 4 I/O ports of 8 bits each. This provides the user with 32 I/O lines for connecting MCU to the environs. Unlike the case with other controllers, there is no specific SFR register for designating pins as input or output. Instead, the port itself is in charge: 0=output, 1=input. If particular pin on the case is needed as output, the appropriate bit of I/O port should be cleared. This will generate 0V on the specified controller pin. Similarly, if particular pin on the case is needed as input, the appropriate bit of I/O port should be set. This will designate the pin as input, generating +5V as a side effect (as with every TTL input).

Port 0

Port 0 has two fold role: if external memory is used, it contains the lower address byte (addresses A0-A7), otherwise all bits of the port are either input or output. Another feature of this port comes to play when it has been designated as output. Unlike other ports, Port 0 lacks the "pull up" resistor (resistor with +5V on one end). This seemingly insignificant change has the following consequences:

- When designated as input, pin of Port 0 acts as high impedance offering the infinite input resistance with no "inner" voltage.
- When designated as output, pin acts as "open drain". Clearing a port bit grounds the appropriate pin on the case (0V). Setting a port bit makes the pin act as high impedance. Therefore, to get positive logic (5V) at output, external "pull up" resistor needs to be added for connecting the pin to the positive pole.

Therefore, to get one (5V) on the output, external "pull up" resistor needs to be added for connecting the pin to the positive pole.

Port 1

This is "true" I/O port, devoid of dual function characteristic for Port 0. Having the "pull up" resistor, Port 1 is fully compatible with TTL circuits.

Port 2

When using external memory, this port contains the higher address byte (addresses A8–A15), similar to Port 0. Otherwise, it can be used as universal I/O port.

Port 3

Beside its role as universal I/O port, each pin of Port 3 has an alternate function. In order to use one of these functions, the pin in question has to be designated as input, i.e. the appropriate bit of register P3 needs to be set. From a hardware standpoint, Port 3 is similar to Port 0.

As can be seen from the individual descriptions of the ports, they all share highly similar structure. However, you need to consider which task should be assigned to which port. For example: if utilizing port as output with high level (5V), avoid using Port 0 as its pins cannot produce high logical level without an additional resistor connected to +5V. If using other port to a same end, bear in mind that built-in resistors have relatively high values, producing the currents limited to few hundreds of amperes as pin output.

Chapter Three
Design Concepts

3.1 Introduction

3.2 Objectives

3.3 Block Diagrams

3.4 How System Works

3.1 Introduction

This chapter discusses the design concepts of the project .It includes the project objectives .Additionally, the system block diagrams that helps to understand the system and how it works.

3.2 Objectives

The project objectives:

- Designing a self independent candy bar vending machine.
- Designing a coin recognizer unit which has the ability to recognize the coins
- Making a mechanical delivery unit which deliver the products to the user
- Controlling the system using Microcontroller

3.3 Block Diagrams

The following figure (3.1) show a block diagram for a vending machine it consists of input switches for selecting products ,coin inputs and in the other side (output) there are four kind of products could be delivered for the user .

Fig 3.2 show more detailed block diagram and main parts in the system.

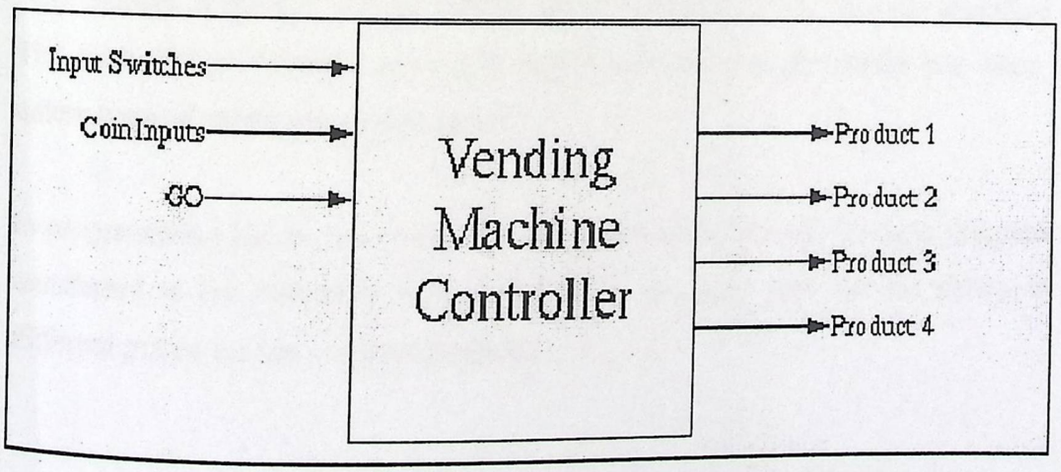


Figure (3.1): General Block Diagram

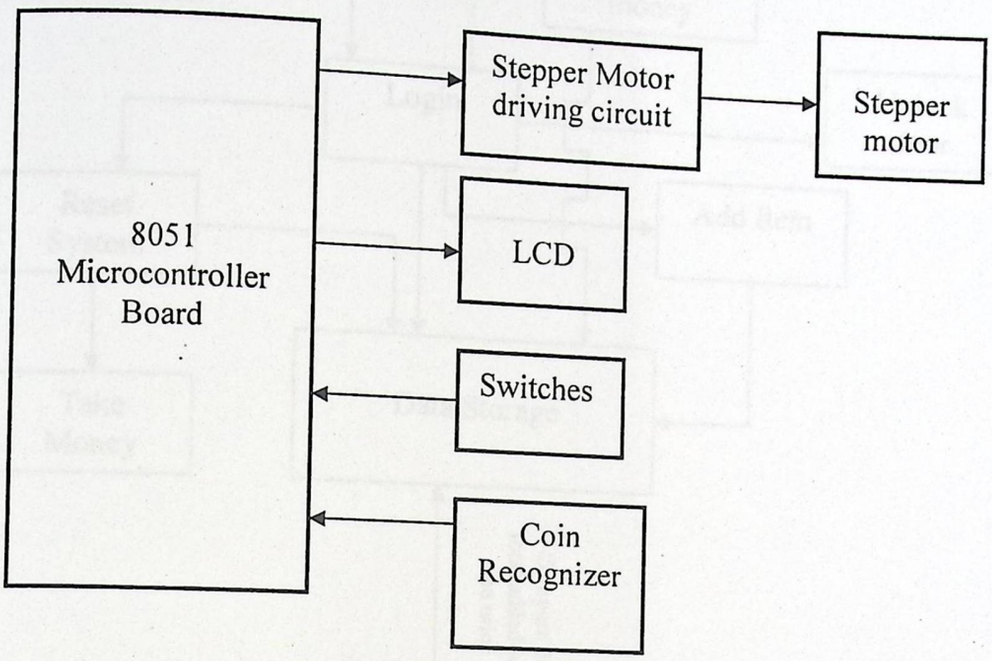


Figure (3.2): Block Diagram

3.4 How the system works

The vending machine proposed in this project operates just like any other vending machine. The controller will keep a running total when coins are deposited. After the money has been deposited, the user will select the product to be vended and push the "GO" button. If there is enough money in the machine, it will vend the specified product. The controller will ignore any vend requests if not enough money had been deposited unless enough money has been added.

In programming mode, the prices for the products are stored. Using a program could be transferred to the controller by a serial cable; the controller has the ability to program different prices for the various products.

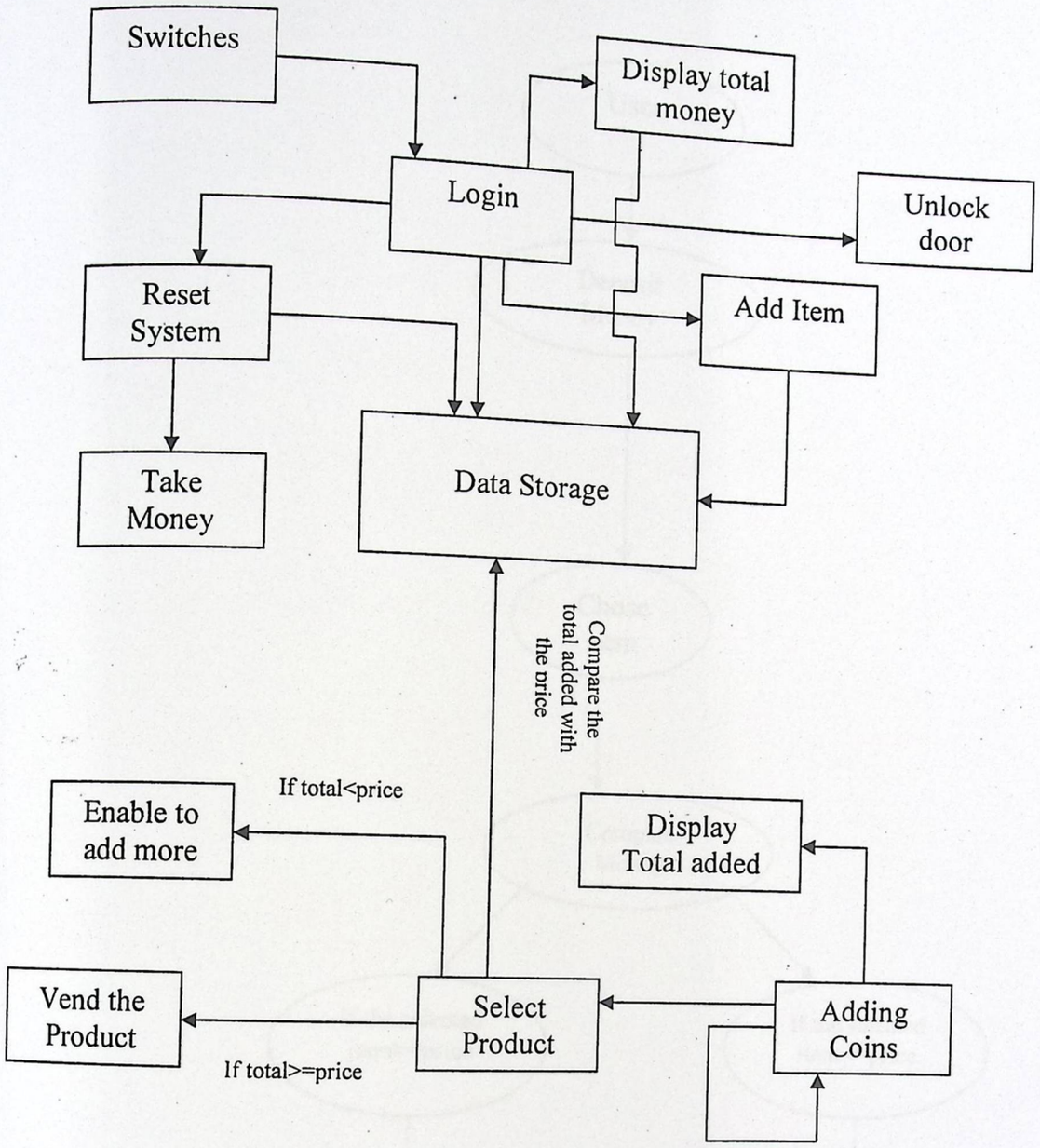
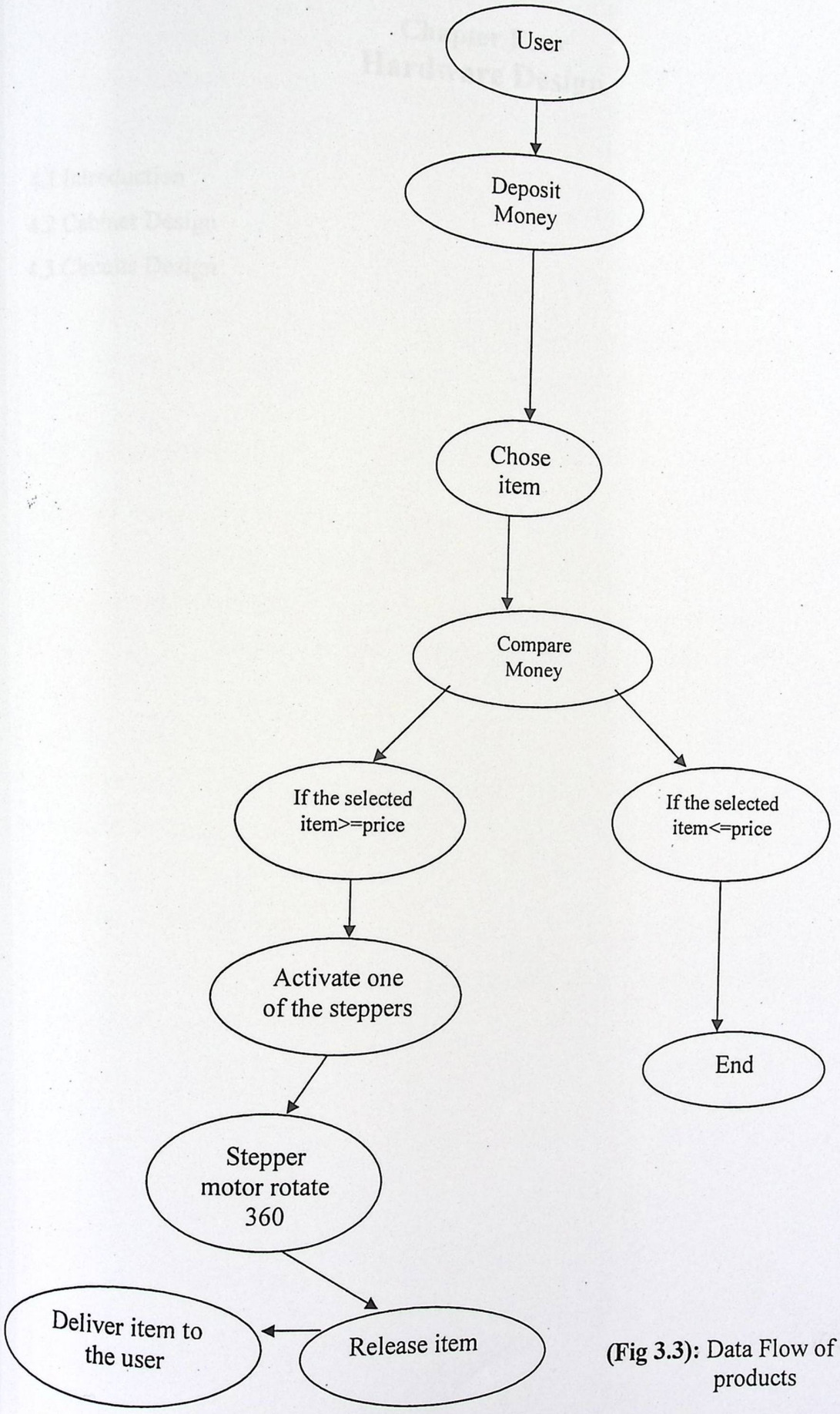


Fig (3.2): System Process model



(Fig 3.3): Data Flow of selling products

Chapter Four
Hardware Design

4.1 Introduction

4.2 Cabinet Design

4.3 Circuits Design

4.1 Introduction:

To build the system we divided the Hardware design into two general parts: the electrical part, which includes the electrical circuits of the system including the driving circuits of the motor, the coin recognizer circuits and the switches circuit, the second part is the cabinet of the system.

4.2 Cabinet Design

The cabinet was made by an aluminum shop, the design was done with aluminum with front of glass with the shape and dimensions given in the following sections

4.2.1 Cabinet Dimension:

-Width: 65 cm divided into two vertical sections

First section 40 cm which include the Delivery Item system.

Second section 25 cm which include the switches and the coin recognizer

-Height: 150 cm Divided into five sections horizontally divided into five sections of 30cm.

-Length: 65 cm.

Fig (4.1) shows the shape of vending machine and figure (4.2) and (4.3) show the planes of the vending machine.

4.2.2 Cabinet Description:

The front of the cabinet will be made of glass to show the products and the bottom part of the cabinet will be made of aluminum

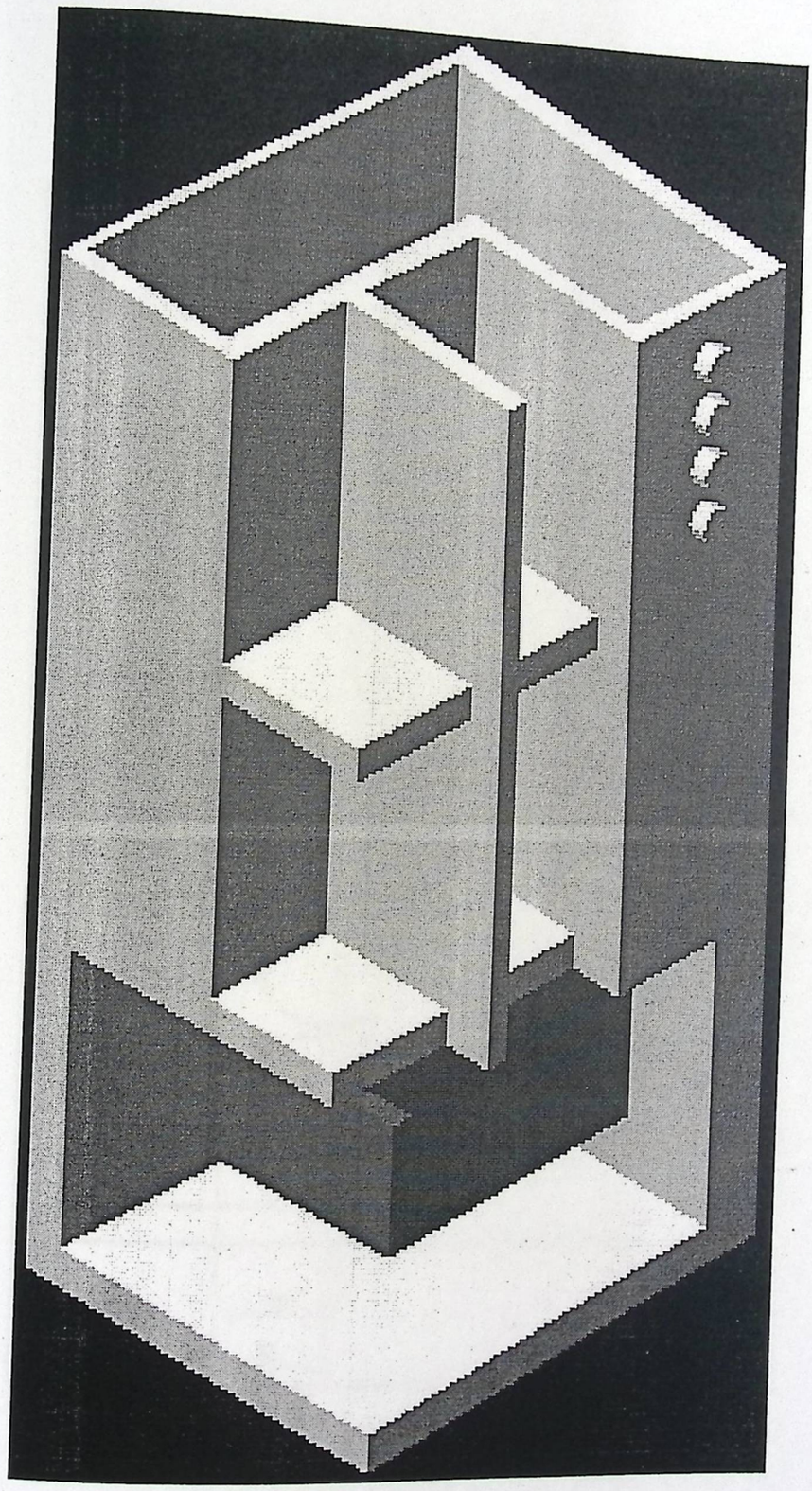


Fig (4.1): Vending Machine Body shape

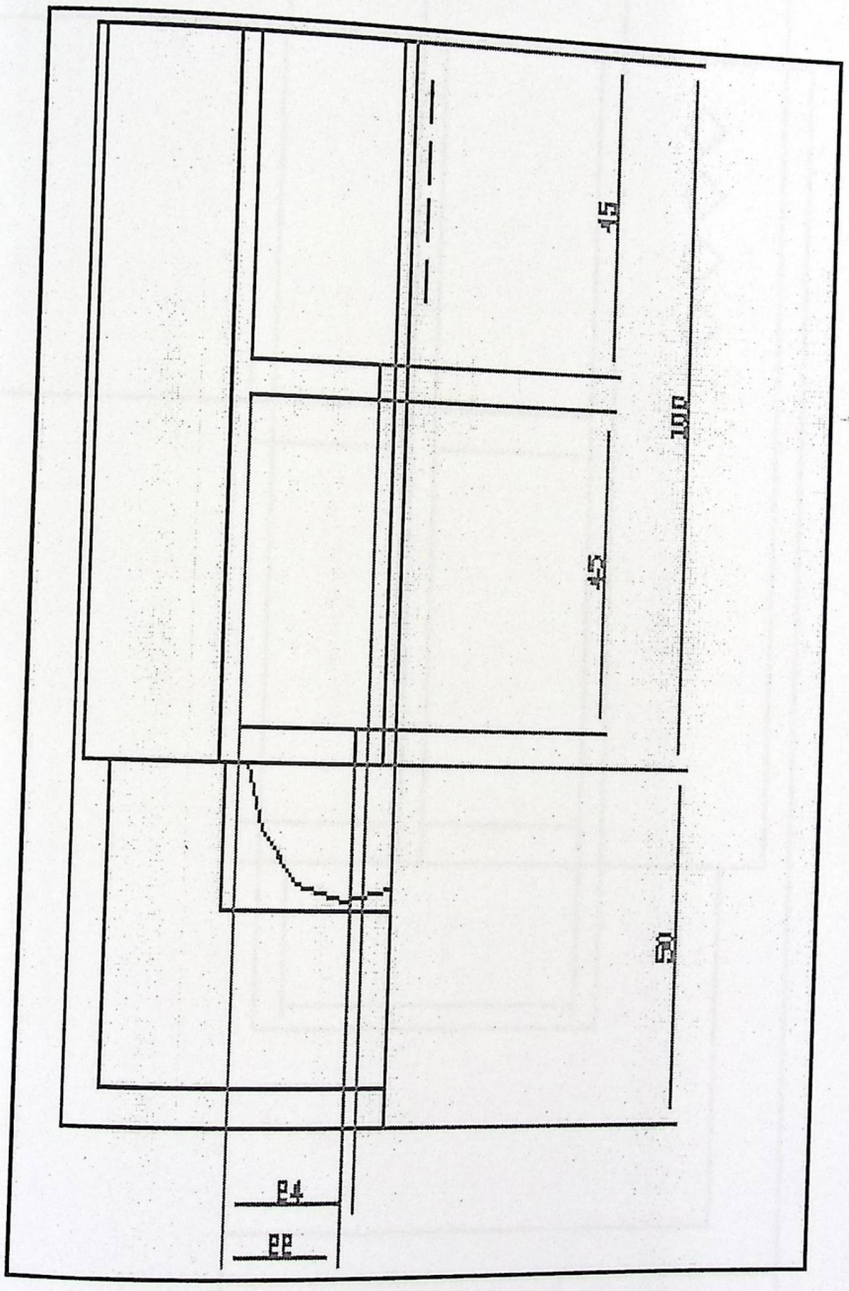


Fig (4.2): side view of Vending Machine

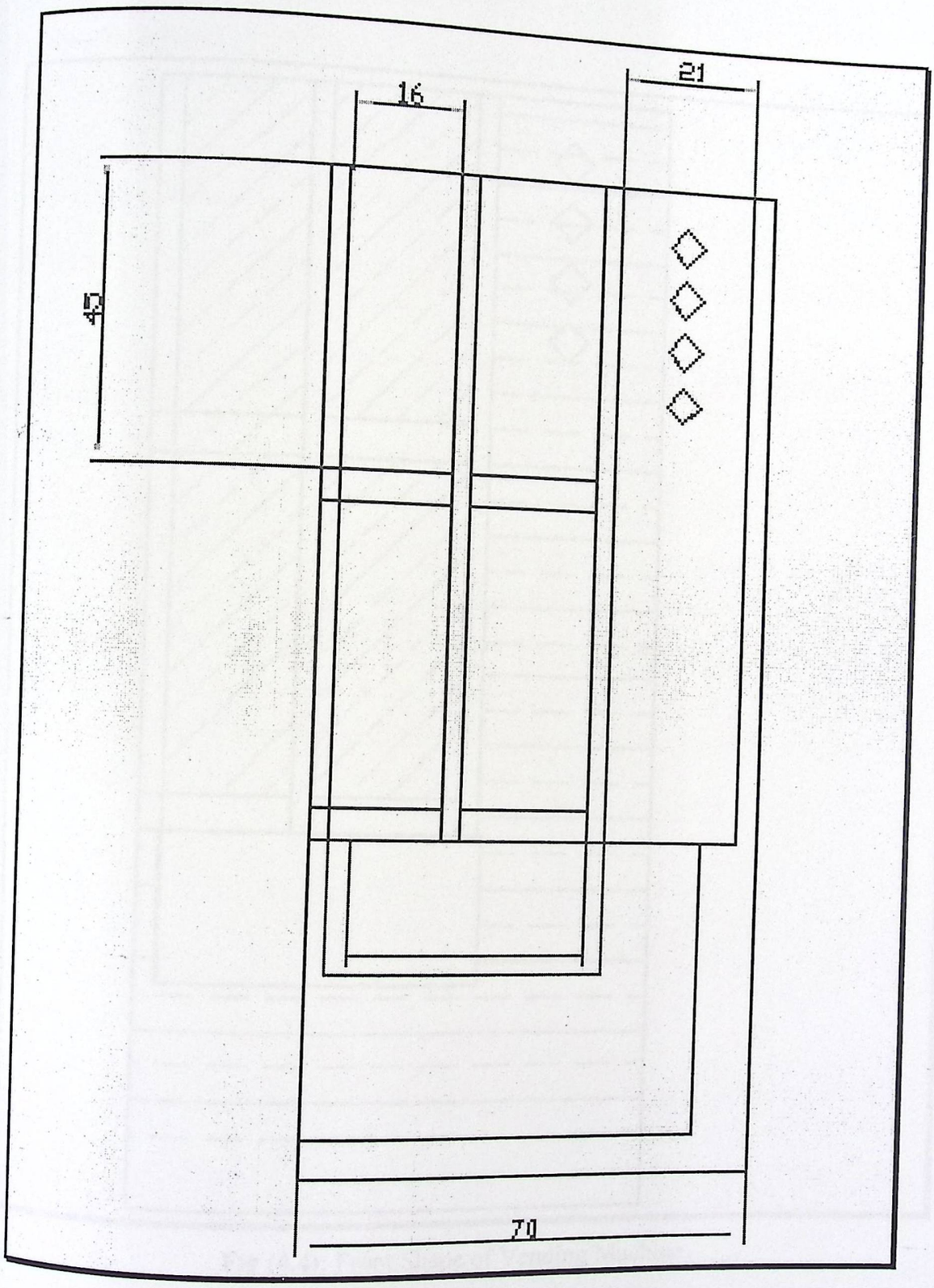


Fig (4.3): Front View of Vending Machine

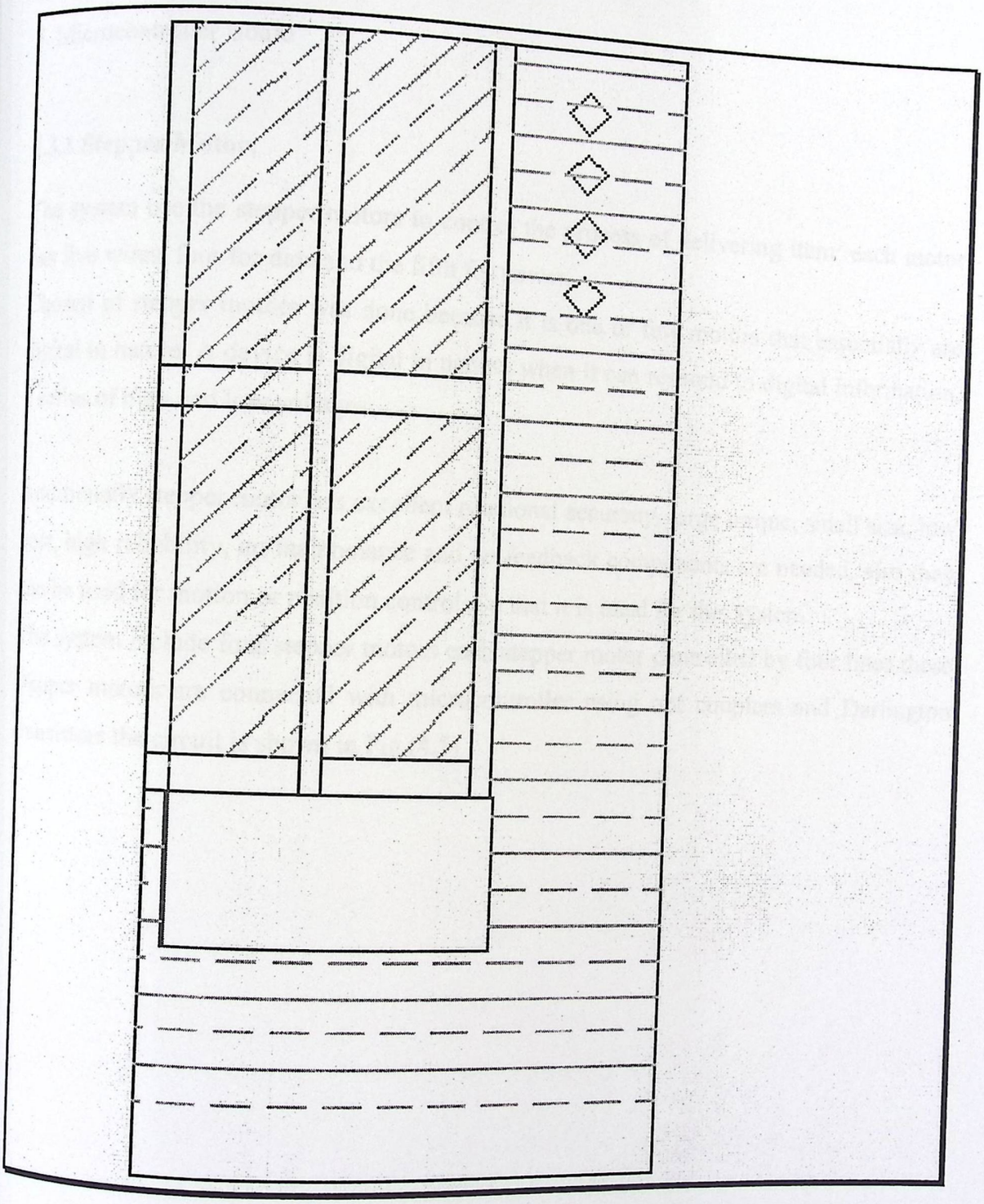


Fig (4.4): Front Shape of Vending Machine

4.3 Circuits Design:

The system consists of three main parts:

1. Stepper motor drive circuit
2. Coin Recognizer.
3. Microcontroller Board

4.3.1 Stepper Motor:

The system use the stepper motors to control the process of delivering item; each motor has five wires, four for data and the fifth for power.
Chosen of stepper motors was done because it is one of few motors that essentially are digital in nature. A device is digital in nature, when it can respond to digital information, a series of high and low voltages.

Additionally stepper motor has excellent rotational accuracy, large torque, small size, low cost, high reliability, no maintenance and no feedback components are needed, also they can be used for motion or position control .So that it is ideal for this system.
The system include four stepper motors each stepper motor controlled by four lines these stepper motors are connected with microcontroller using opt couplers and Darlington transistors the circuit is shown in Fig (4.5)

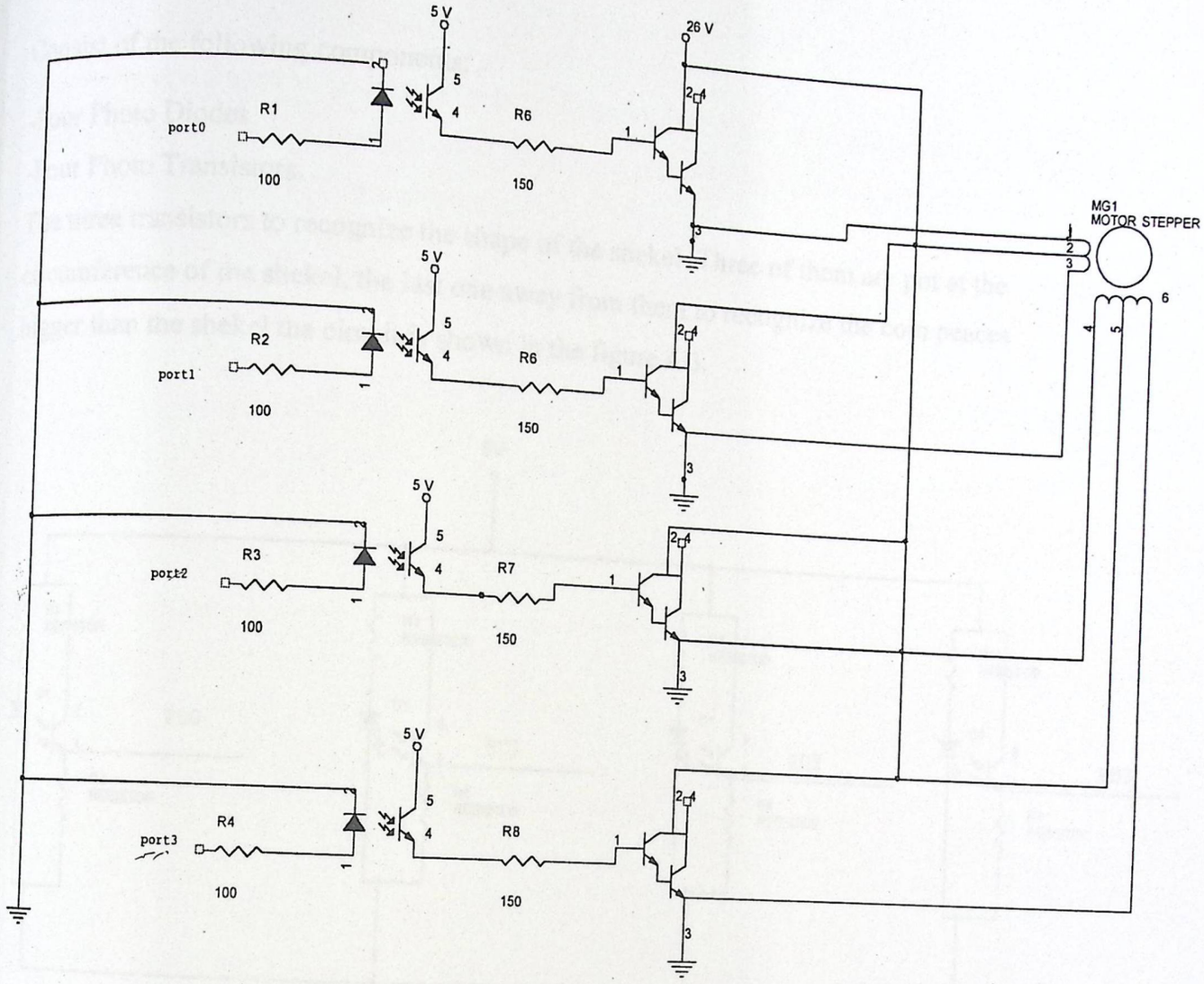


Fig (4.5): Steppers motor interfacing circuit

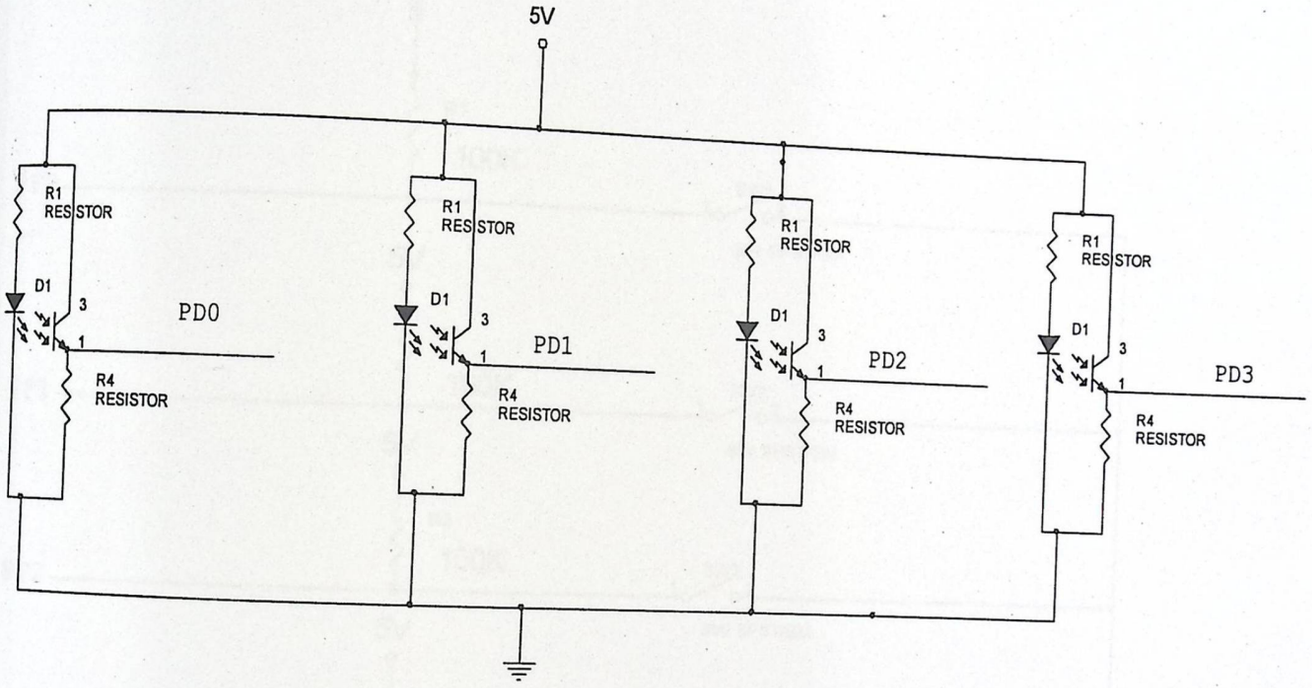
4.3.2 Coin Recognizer:

Consist of the following components:

-Four Photo Diodes.

-Four Photo Transistors.

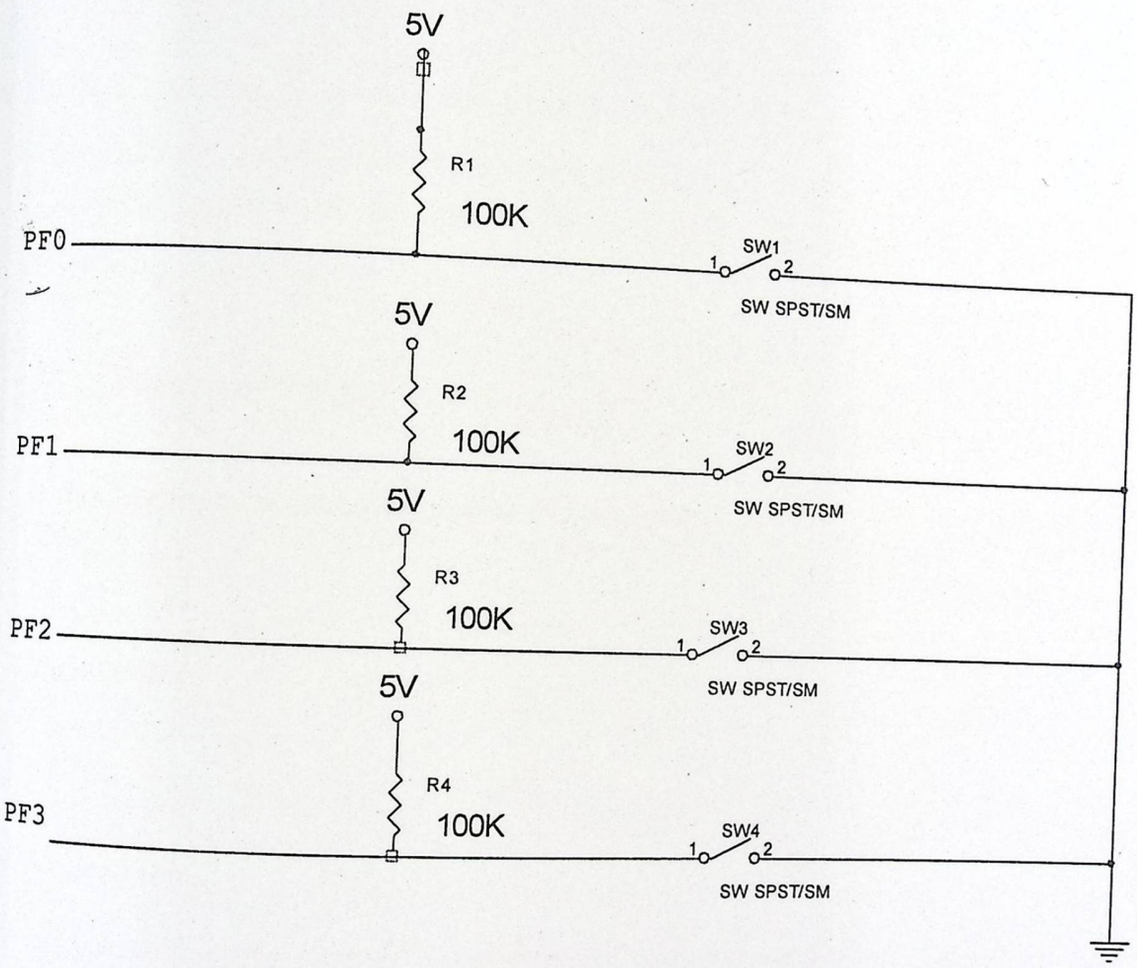
The three transistors to recognize the shape of the shekel .Three of them are put at the circumference of the shekel, the last one away from them to recognize the coin peaces bigger than the shekel the circuit is shown in the figure 4.6.



(Fig 4.6): Coin Recognizer Circuit

4.3.3 Switches:

The switches are used in the project to select between four different kinds of products the input signals for the switches have been taken from port F of the microcontroller Fig 4.7 shows the circuit of the switches.



(Fig 4.7): Switches

Chapter Five
Software Design

5.1 Overview

5.2 Functional Requirement

5.3 Non Functional Requirement

5.4 Algorithms

5.5 Software Code

5.1 Overview

In Chapter Four: Hardware System Design the hardware equipments and their interconnections have been discussed .In this chapter the software subroutines and their functionality are going to be explained in the next sections.
The software functions include writing a program to recognize money, select product and to activate motors.

5.2 Functional Requirement

The functional Requirements of the machine are:
- Recognizing a coin deposited by the user.
- Accepting the Choice of the user.
- Controlling the Motor

5.3 Non Functional Requirement

The non functional requirements of the machine are:
- PC Pentium 3
- Windows XP with HyperTerminal Program
-Serial Cable
- Assembler as31_sdcc

5.4 Algorithms

In this section the main parts of the program are going to be explained. These parts are coin recognizer, switches and motors.

5.4.1 Coin Recognizer

Coin recognizer consist of four photo diodes and four photo transistors ,the check will depends on the shape of the shekel which represent by the code 08H the flowchart begin with enable port d as input ,P0,P1,P2,P3 for coin recognizer and the fourth P4 for end

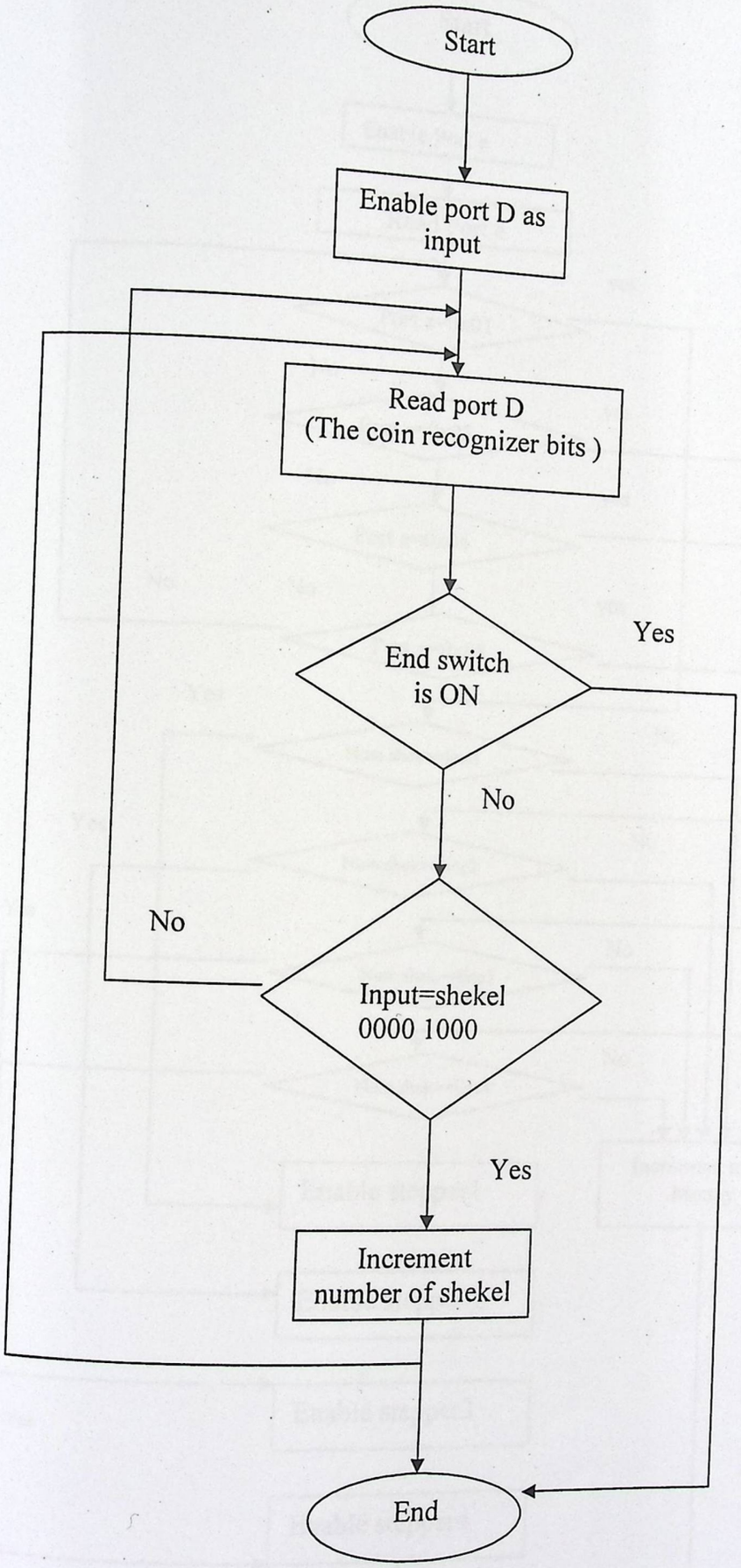
switch .the system will continue checking the status of the end switch and in each loop it will check if the deposited coin is shekel if so it will increment the number of shekels(counter) until the end switch is pressed this is shown in Fig 5.1.

5.4.2 Accepting choice

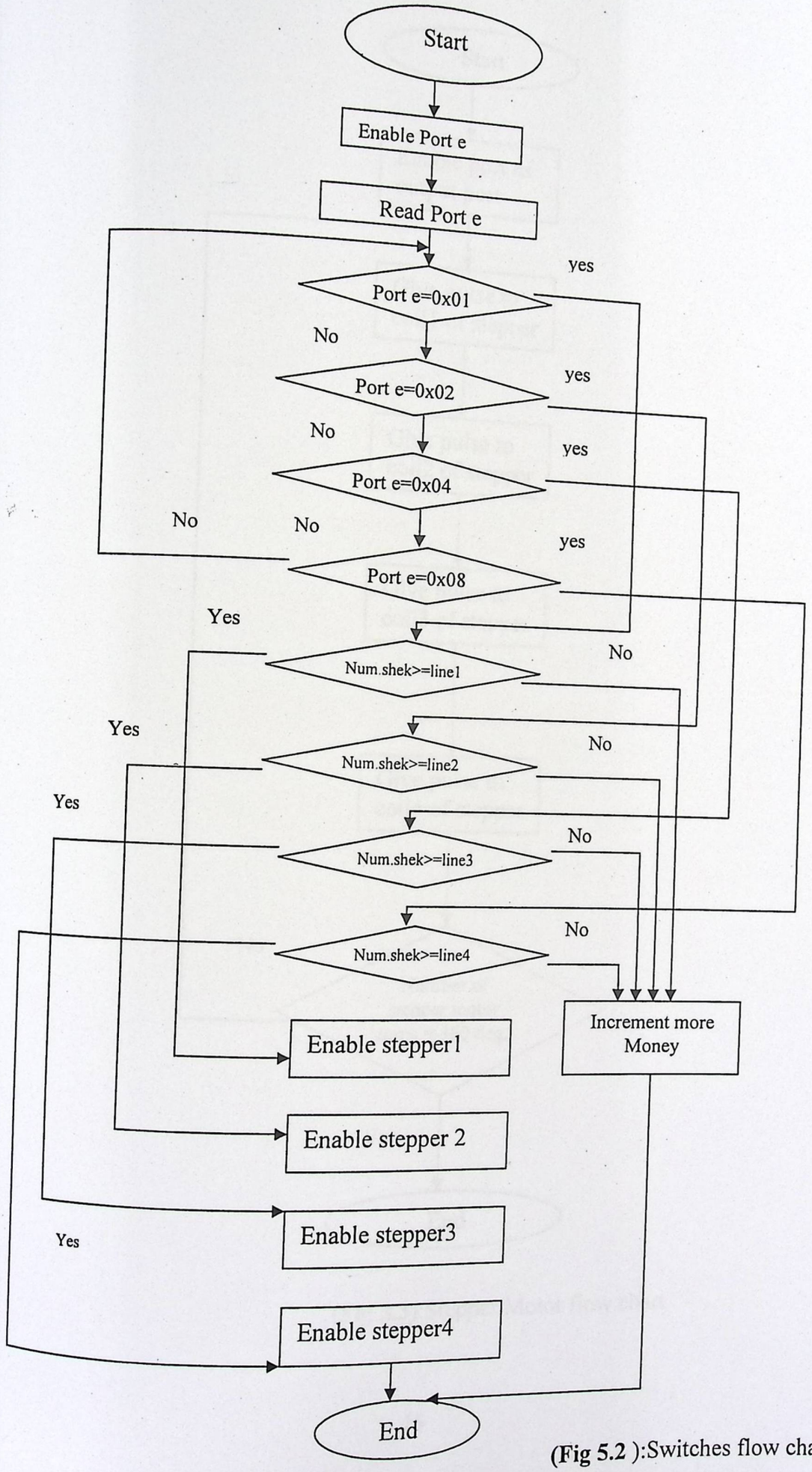
The system will contain four switches, which will be connected to port e, the flowchart will begin with enable port e as input, the system will keep running in a loop until one of the switches is pressed, then it will check if the price which represented by the selected switch if it is greater or equal to the deposited money it will enable the stepper motor which will release the selected product if not it will return to the read process this is shown in the Fig 5.2.

5.4.3 Motor Controlling

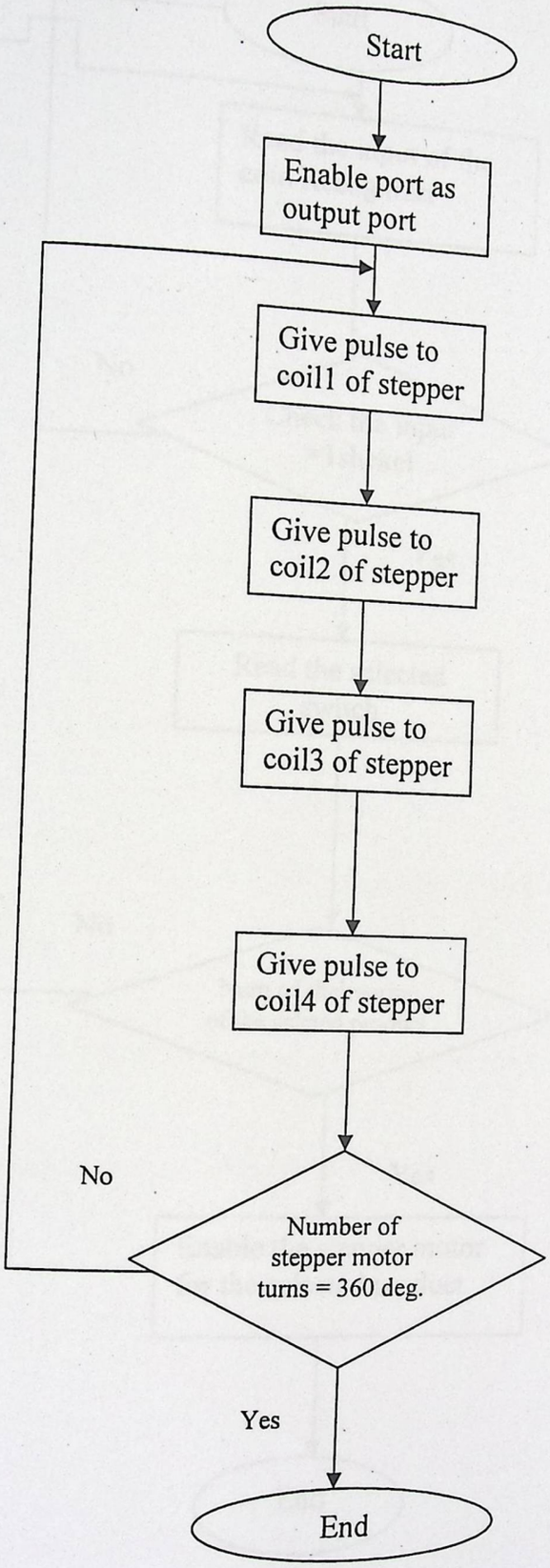
The motors will be connected to the port a and b ,the port a and b will be enabled as output port ,data will be written to the motor in the sequence 0x01,0x02,0x04,0x08 and it will keep running until it will be stopped when the motor rotate 360 degree this is shown in the Fig 5.3.



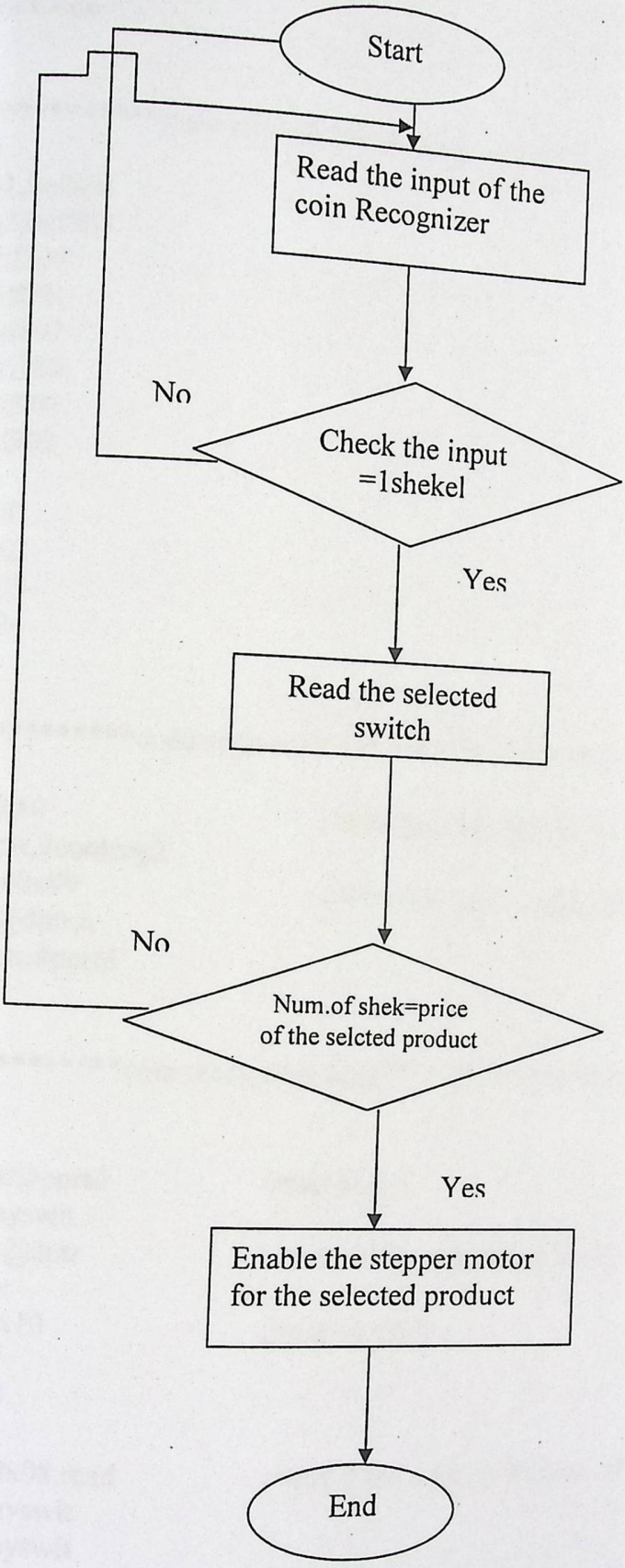
(Fig 5.1): Coin Recognizer flow chart



(Fig 5.2):Switches flow chart



(Fig 5.3):Stepper Motor flow chart



(Fig 5.4): System Flow chart

5.5 Software Code

```
;******************main definitions variabelse **********************
.org 0x8000
.equ contreg1,0xf803
.equ contreg2,0xf903
.equ porta,0xf800
.equ portb,0xf801
.equ portc,0xf802
.equ portd,0xf900
.equ porte,0xf901
.equ portf,0xf902

.equ line1,0x01
.equ line2,0x02
.equ line3,0x03
.equ line4,0x04


;******************code segment**********************

        mov r0,#0                   ; initialize the register r0
        mov dptr,#contreg2
        mov a,#0x99                 ;port D as input, port E output, port f input
        movx @dptr,a
        mov dptr,#portd


;******************coin recognizer read**********************
read:

        mov dptr,#portd             ;read port d
l       call delayswit
        movx a,@dptr                ;check if the end switch is on    0001 0000
        mov r3,a
        anl a,#0x10                 ;jump to exist
        jnz exit1
        mov a,r3

        cjne a,#0x08,read           ;check if the input is shekel    0000 1000
        lcall delayswit
        lcall delayswit

        inc r0                      ;increment the number of shekels
```

```
        mov dptr,#porte
        mov a,r0                ;display on leds the value of the number of shekels
        movx @dptr,a
        ljmp read               ;keep reading


;*******************select the switching**********************
exit1:
        mov r3,#0               ;initialize r3
        mov dptr,#portf
        lcall delayswit         ;read the selected switch from portf
        movx a,@dptr
        lcall delayswit
        mov r3,a
        anl a,#0x01             ;if switch1 is on jmp to swit1
        jnz swit1
        mov a,r3
        anl a,#0x02             ;if switch2 is on jmp to swit2
        jnz swit2
        mov a,r3
        anl a,#0x04             ;if switch3 is on jmp to swit3
        jnz swit3
        mov a,r3
        anl a,#0x08             ;if switch4 is on jmp to swit4
        jnz swit4
        ljmp exit1              ;loop

;*******************item delivery**********************
swit1:
        clr c                   ;clear carry
        mov a,r0                ;move the number of shekels to register a
        subb a,#line1           ;compare the number of shekels with line1 if it is not
        jc read                 ;greater or equal then jmp to read
        mov dptr,#contreg1      ;define port a ,b,c as outputs
        mov a,#128
        movx @dptr,a
        mov dptr,#porta         ; jmp to step1
        lcall step1
        ljmp read
swit2:
        clr c                   ;clear carry
        mov a,r0                ;move the number of shekels to register a
        subb a,#line2           ;compare the number of shekels with line1 if it not
        jc read                 ;greater or equal then jmp to read
```

```
        mov dptr,#porte
        mov a,#0x80
        movx @dptr,a
        mov dptr,#contreg1
        mov a,#128
        movx @dptr,a
        mov dptr,#porta
        lcall step2
        ljmp read

swit3:
        clr c                   ;clear carry
        mov a,r0                ;move the number of shekels to register a
        subb a,#line3           ;compare the number of shekels with line1 if it is not
        jc read                 ;greater or equal then jmp to read
        mov dptr,#contreg1
        mov a,#128
        movx @dptr,a
        mov dptr,#portc
        lcall step1
        ljmp read

read2:
        lcall read

swit4:
        clr c
        mov a,r0
        subb a,#line4
        jc read2
        mov dptr,#contreg1
        mov a,#128
        movx @dptr,a
        mov dptr,#portb
        lcall step
        ljmp read


step1:
        mov r5,#0
        write1:


        mov a,#0x01
        movx @dptr,a
```

```
        lcall delay
        lcall delay


        mov a,#0x02
        movx @dptr,a

        lcall delay
        lcall delay


        mov a,#0x04
        movx @dptr,a


        lcall delay
        lcall delay

        mov a,#0x08
        movx @dptr,a




        lcall delay
        lcall delay
        inc r5

        cjne r5,#50,write1

        mov a,#0x01
        movx @dptr,a
    mov a,#0x00
        movx @dptr,a

        lcall delay
        lcall delay

        ret

step2:
        mov r5,#0
```

```
write:

mov a,#0x10
movx @dptr,a

lcall delay
lcall delay

mov a,#0x20
movx @dptr,a

lcall delay
lcall delay

mov a,#0x40
movx @dptr,a

lcall delay
lcall delay

mov a,#0x80
movx @dptr,a

lcall delay
lcall delay
inc r5

cjne r5,#50,write

mov a,#0x10
movx @dptr,a
mov a,#0x00
movx @dptr,a

lcall delay
lcall delay

ret
```

```
delay:
        dly30: mov    r1, #10
dly31:
        nop


        nop
        djnz    r1, dly31       ;repeat 230 times for 1 ms
        djnz    r2, dly30       ;repeat for specified # of ms
        ret

delayswit:
dly2:   mov     r1, #230
dly3:   nop
        nop
        nop                     ;3 NOPs + DJNZ is 2.155 us
        djnz    r1, dly3        ;repeat 230 times for 1 ms
        djnz    r2, dly2        ;repeat for specified # of ms
        ret
```

Chapter Six
Implementation and Testing

6.1 Introduction

6.2 Hardware Implementation and Testing

6.1 Introduction

This chapter discusses the steps we followed to test out our design, the circuits and the Subroutines we implement during developing this project.

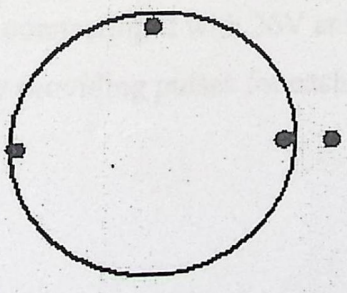
6.2 Hardware Implementation and Testing

The system units is built and tested separately, and in the following next subsections these steps are going to be discussed.

6.2.1 Coin Recognizer:

The coin recognizer depends on the shape of the coins (shekel) in the process of recognizing the coin , it consists of four photo diodes and four photo transistors, each photo diode fixed in a way facing the other photo diode .

The figure 6.1 showhow the diodes and the transistors are fixed. The reading of the photo transistors is 5V when the shekel passes it will cover three photo transistors so the reading will be 0V for three of them the forth will be 5V so on the system the binary code of the shekel is 0001. if the coin is bigger or smaller than shekel the code will be different so the system will not count it. The photo diodes and the photo transistors are fixed on a wood peace which has a path entry suitable for the coin (shekel) .



(Fig 6.1): fixed photo diodes on coins points

6.2.2 LCD

The LCD is used in the system to display sentences or commands for the user how to use the system it represents the interface between the user and the machine, the LCD was tested by writing words on it ,

The final program show on the LCD the command (insert money) after that it shows the number of shekels deposited in the machine (the account).

6.2.3 Switches

Switches are being used to select the kind of product that the user wants to buy; they are 4 ordinary switches like the switches used in the keyboard they have been tested by connecting them to the microcontroller and writing a program and observing the results.

6.2.4 Springs

The springs which is used to deliver the product to the user was used to arrange the candy in stacks, it was made of metal connected to circuital head and to the motor each rotate of the motor 360 degree will release one of the candy in the spring stack .

6.2.5 Stepper Motor

The motors were chosen to implement the vending machine were stepper motors which are very easy to use and control, the stepper motor used, provides enough torque which has the ability to push the stack arranged in the spring stack.

The stepper motor was tested by connecting it with 26V and connecting the coil with the PPI of the microcontroller and by providing pulses for each coil of the motor it will rotate in each pulse 1.8 degree.

6.2.6 The cabinet Box

The cabinet box was implemented with the dimensions and shape specified in the chapter of Hardware Design, it was constructed in aluminum shop with front of glass.

6.2.7 The Microcontroller

The microcontroller board which have been chosen to implement the project is 8051 with has two PPI, the microcontroller could be programmed by using the HyperTerminal program which could be find in any PC and by connecting the microcontroller to the pc by the serial program any program could be transferred to the microcontroller after writing it on any text editor either by using the assembly which was used in our project or it could be programmed using C language.

In the website of the 8051 microcontroller board there is many programs to test the board for example there is a program called LEDs which is being used to test the LEDs of the microcontroller and the PPIs.

Chapter Seven
Conclusions and Future Work

7.1 Conclusions

7.2 Future Work

7.1 Conclusions:

- The main difficulty to implement this project was in the mechanical design especially the design of the springs.
- It was decided to depend on the weight and the size to recognize the coin but because of not getting the load cell we depend only to the size to recognize the coin.
- The 8051 Microcontroller Board is a powerful microcontroller which is very easy to use and could be used for multipurpose applications.
- Vending machine is a full automated machine facilitates the life of the human in different aspects.

7.2 Future work

- Because we haven't found a sensor for sensing small weights the coin recognizer unit has been implemented depending on the size of the coin deposited by using photo diodes and photo transistor which is not so accurate way so we recommend in the future to use both sensors of the size and the weight which will be very accurate way to recognize the coins.
- Adding cashless option for vending product of the vending machine by using the cell phone.

References

http :// Sunpal7.mit.edu/6.111/s99/lab2/lab2s99.html

http://www.8052.com

http://www.straingage/the strain gage.htm

http://www.hanna-vending.com

http://www.ava-vending .org

http://www.allegromico.com

www.datasheetcatalog.com

Appendix A

Data Sheets

Appendices

Appendix A: Datasheets

Appendix B: 8051 Microcontroller Board

PJRC Electronic Projects Components Available Worldwide

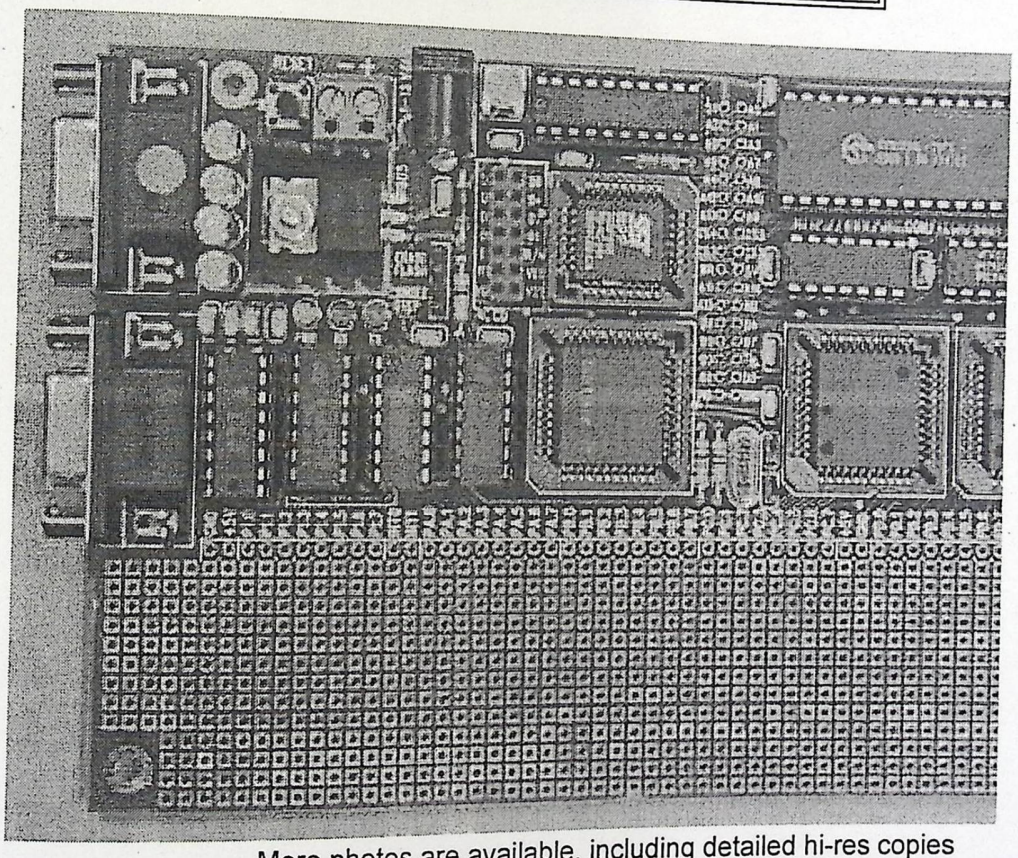
PJRC Store

- Full Product List
- ⊞ MP3 Player
- ▶ 8051 Dev Board
  - ▶ Assembled Board
  - LCD 20x2 Display
  - LCD 16x2 Display
  - Unassembled Kit
- ⊞ Individual Parts
- ⊞ Misc
- Payment Options
- Privacy Policy

Fully Assembled and Teste 8051 Development Board

Add To Order	DEV_PCB_ASSEM	Fully Assembled and Tes 8051 Development Board 5

Inventory Status: **In Stock**
Last physical count: May 27, 2005



More photos are available, including detailed hi-res copies

All boards are the latest Rev 5 version.

The 8051 development board provides an easy and low-cost way tc your 8051 based microcontroller projects, without purchasing any of equipment, such as IC programmers or emulators. The board come with PAULMON2, which provides a simple menu-based system that you to download your own code into the RAM or Flash ROM on the When your project is finished, you can set a few bytes to tell PAULM automatically run your program at start-up, and download it into the chip, so the board will then run your application instead of booting ir PAULMON2 menu system. A jumper is provided, should you need t

Flash ROM to make any changes.

The board features two 82C55 chips, that provies 50 I/O lines and 8
addition to the 10 lines from the 8051's port #1, and the 8051's bus
second serial connector is available, with a simple switching circuit,
easier to develop applications that need the serial port and must no
menus from PAULMON2. The bottom section of the board is a pad-
prototype area, where simple circuits can be added.

The fully assembled board comes ready to use. Just connect to you
port, and you're ready to start using it to develop your project.

Related PJRC Project Pages

- 8051 Development Board
- Schematic and Parts List

PJRC Store

- 8051 Dev Board, $79
- LCD 20x2 Display, $11
- Serial Cable, $5
- 12 Volt Power, $8
- More Components...

8051 Tools

- Main Page
- ⊞ **Software**
- ⊞ **PAULMON Monitor**
- ▶ **Development Board**
  - Features
  - Photos
  - Getting Started
- ⊞ **Example Code**
  - Memory Map
  - Ports & Pinouts
  - Schematic & Parts
  - Circuit Board
  - Construction
- ⊞ **Troubleshooting**
- ▶ **Old Versions**
  - Revision List
  - ▶ **Rev 4 (2002)**
    - Photos & Features
    - ▶ Getting Started
      - Example Code
      - Memory Map

Using The 8051 Development Board For The First Time

This page is intended to walk you through all the steps to use the 8051 development board for the first time. An attempt is made to show every step, however small. If anything here is unclear or significantly different than what happens when you follow these steps, please contact me so that I can update this page.

Connecting The Board

The 8051 development board connects to a PC using a standard serial cable. The cable should be a straight-though type. Do not use a null modem cable. The normal connection to a PC is made with the upper connector, as shown in figure 1. A DC power source between 8 to 15 volts must be connected to the power input.

• Ports & Pinouts
• Schematic & Parts
• Circuit Board
• Construction
• Troubleshooting
⊞ Rev 3 (2001)
⊞ Rev 2 (1997)
⊞ Rev 1 (1992)
⊞ Code Library
⊞ 89C2051 Programmer
⊞ Other Resources

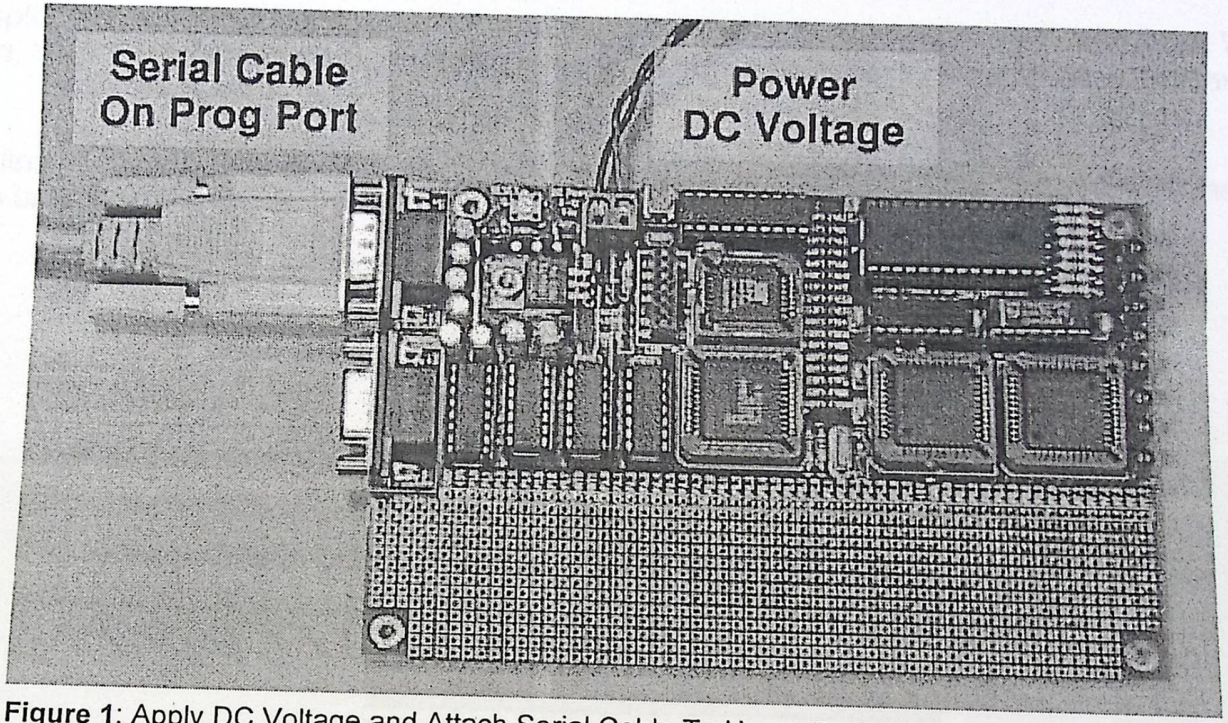


Figure 1: Apply DC Voltage and Attach Serial Cable To Upper Port

It is best to leave the board's power turned off until the terminal emulation program is running and properly configured. If some other software attempts to communicate with a device on your serial port, the board will hear whatever is sent, and may "learn" the baud rate used by the other program. This can happen, for example, when Windows is booted and the plug-n-play configuration attempt to find any modems connected. If the board has detected an incorrect baud rate from a previous communication, just turn off the power for a minute or two.

Running A Terminal Emulation Program

To communicate with the 8051 development board, you must use a terminal emulation program. Nearly any terminal emulation software will work. Hyperterminal is a popular choice, because it is included in the default Microsoft Windows installation. With Linux, minicom is an excellent console mode terminal emulator, and is included with most Linux distributions. Seyon is available for X11, though RedHat hasn't included it since 5.2, so you may have to download and install it.

The example here will focus on Windows Hyperterminal. The dialogs shown are from a fresh Windows 95 installation. Your system may display slightly different dialogs, particularly if you have a modem already installed.

Hyperterminal is usually located in the "Accessories" group, which is available from "Programs" in the "Start" menu. It is usually just a folder, which should appear similar to figure 2.

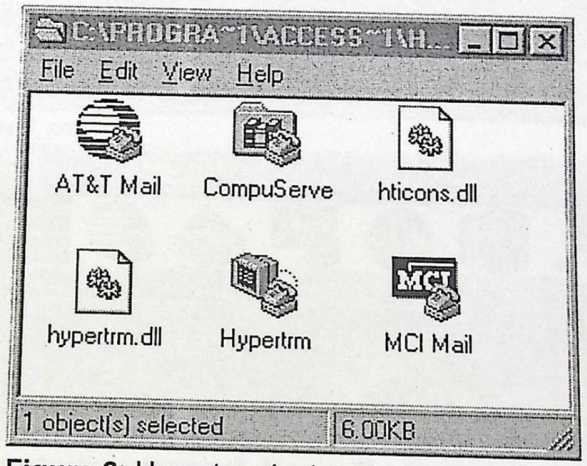


Figure 2: Hyperterminal, Included In Default Windows Installation

The "Hypertrm" icon represents the Hyperterminal program. When Hyperterminal is first run, it will display a dialog suggesting that you need to install a modem, as shown in figure 3. You do not actually need to install a modem, so select **No** from this dialog, if it appears.

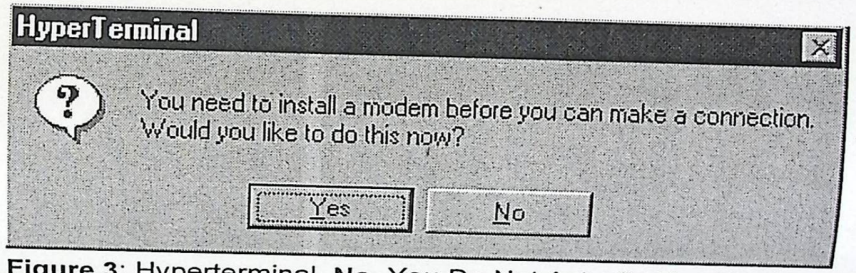


Figure 3: Hyperterminal, **No**, *You Do Not Actually Need To Install* A Modem

Next, Hyperterminal will ask you to provide a name for the new connection, as in figure 4. It does not matter what you name the connection. Hyperterminal may not allow certain names, such as "COM1:".

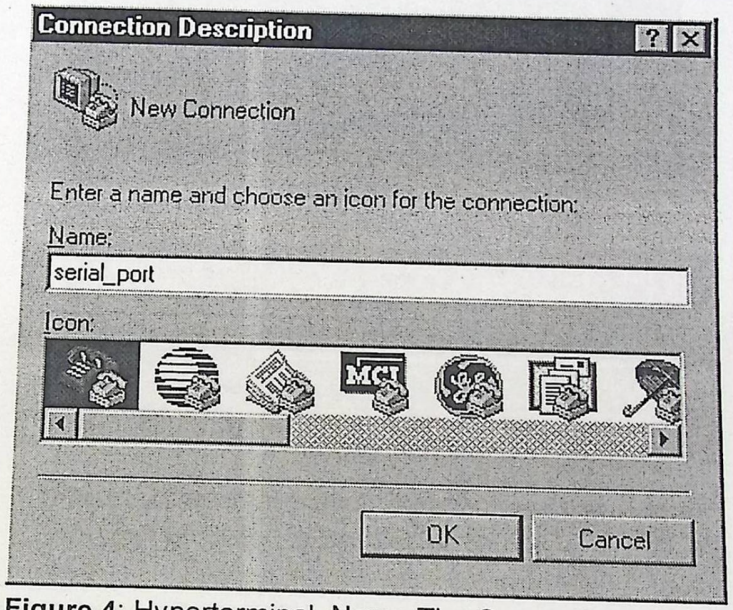


Figure 4: Hyperterminal, Name The Connection (Whatever You Like)

After you've provided a name, you will need to select the port where the 8051 development board is connected. Most newer (ATX) PC computers have two 9-pin serial ports, where the one closer to the top of the computer is COM1 (/dev/ttyS0 in linux), and the lower is COM2 (/dev/ttyS1 in linux). Select the one where you connected the 8051 development board.

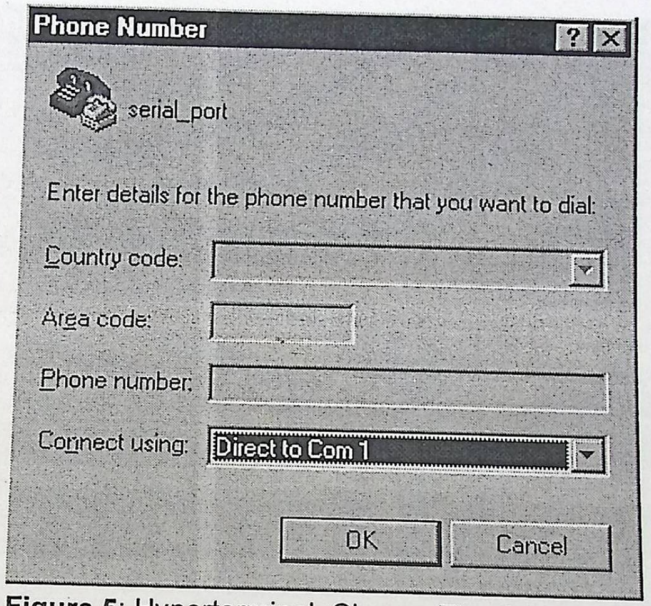


Figure 5: Hyperterminal, Choose The COM Port Where The Board Is Connected

The last, and most important step is to configure the communication settings. You must select "None" for Flow Control. The default setting is "Hardware", and this is likely to cause problems. The 8051 development board can work at all standard baud rates, except 300 baud. Most people choose 115200 baud.

Some older PC computers, particularly those with 16450 (no fifo) UARTs are unable to communicate well at 115200 baud. Nearly all Pentium-class PCs can use 115200, but some older 486 computers can not communicate this fast. With an older computer that lacks a 16550 uart, 9600 baud is probably a better choice.

If you return to this menu and change to a different baud rate, be sure to turn off the board's power. The board detects and remembers your baud rate setting, so it must be shut off (a warm boot will not do) when you change the baud rate.

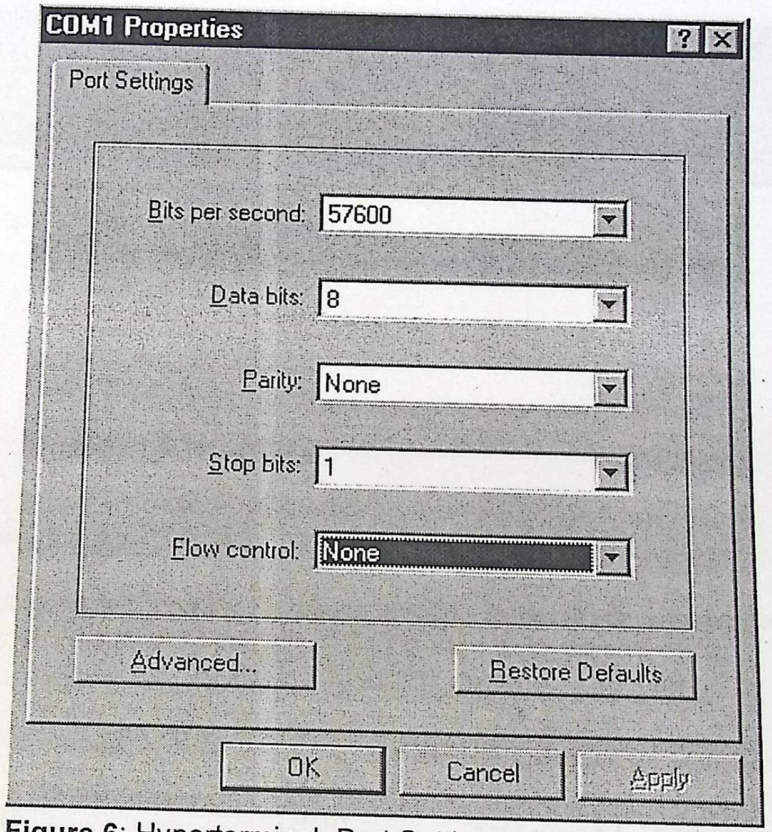


Figure 6: Hyperterminal, Port Settings, 57600 Baud, No Flow Control. Turn The Board's Power Off When Changing Baud Rates

After setting up the communication parameters, Hyperterminal will show it's main window, with nothing in it. Turn on the board's power. If the board was already on, shut it off for several seconds and turn it back on.

The 8051 development board waits for you to press Enter, when it does not "know" what baud rate you have selected. When you press Enter, the welcome message should appear, as in Figure 7.

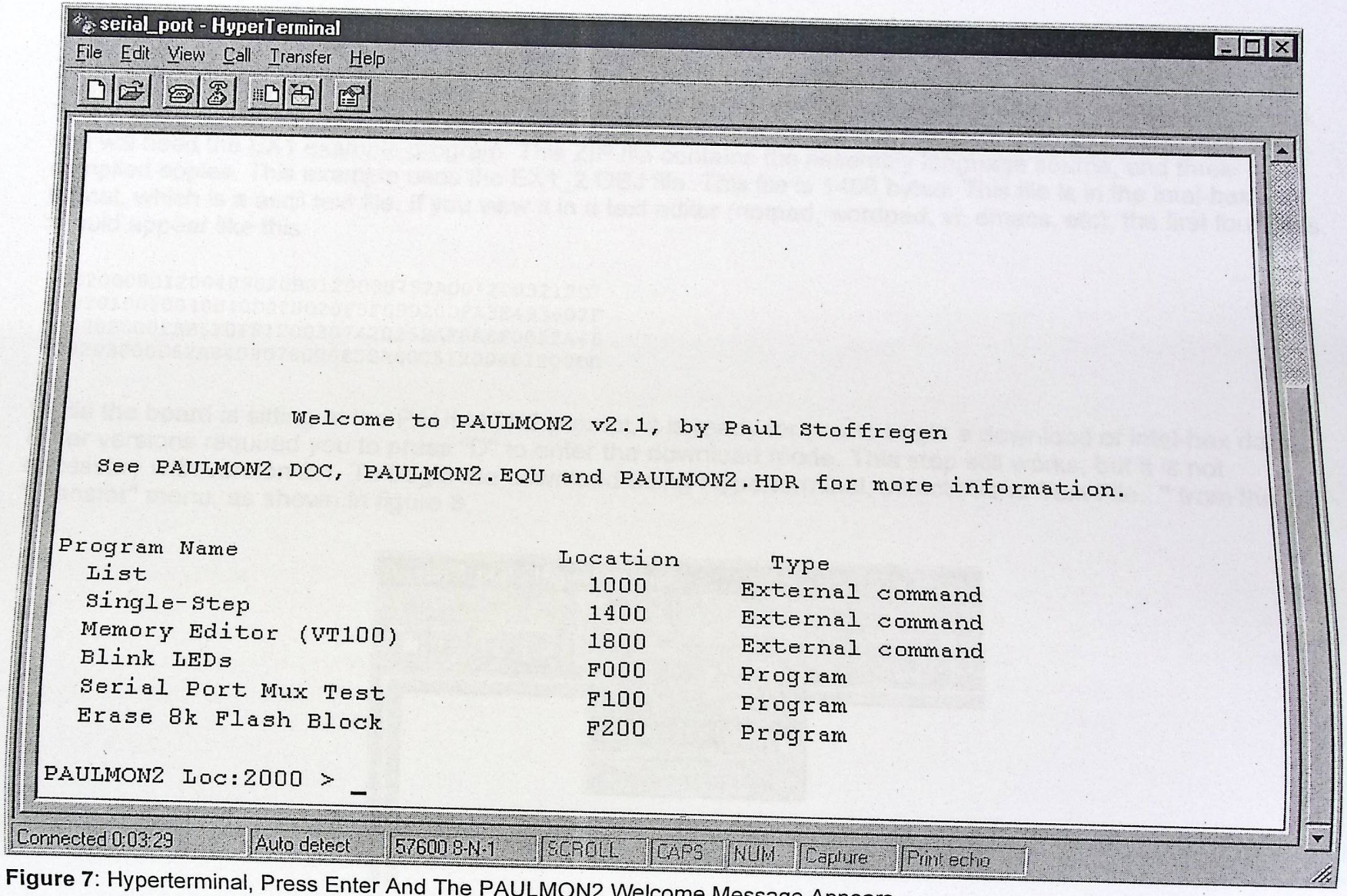


Figure 7: Hyperterminal, Press Enter And The PAULMON2 Welcome Message Appears

Hyperterminal's File menu has a "Save As..." option which will allow you to save the setting to a file. This file can be copied to the desktop, so that you may easily run Hyperterminal again with the settings to use your board.

Downloading And Running A Program

To make use of the 8051 development board, you will need to be able to download and run your own programs. This section demonstrates the steps to download and run programs using Hyperterminal. To follow these steps, you will need the EX1 example program. This ZIP file contains the assembly language source, and three compiled copies. This example uses the EX1_2.OBJ file. This file is 1406 bytes. This file is in the intel-hex format, which is a ascii text file. If you view it in a text editor (notpad, wordpad, vi, emacs, etc), the first four lines should appear like this:

```
:10200000120048902080120038752A0012003212D7
:102010000040B40D028020F5F09020DFA3E493602F
:10202000EBB5F0F81200307420252AF8A6F0052A46
:10203000E52AB409D78000E52A60C51200481200DD
```

While the board is sitting at the PAULMON2 prompt, it is ready for you to begin a download of intel-hex data. Older versions required you to press "D" to enter the download mode. This step still works, but it is not necessary with version 2.1. To begin the download using Hyperterminal, select "Send Text File..." from the "Transfer" menu, as shown in figure 8.

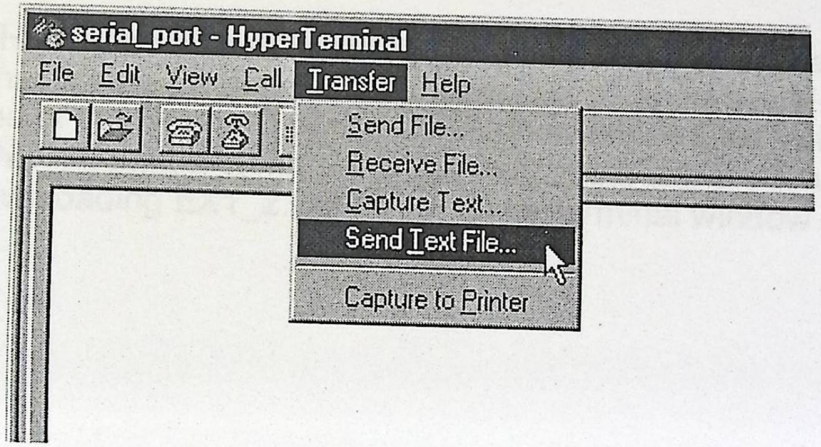


Figure 8: Downloading Example Code, Use "Send Text File..."

The usual Windows file choosing dialog will appear. You will need to change the type to "All files (*.*)", because

the file is EX1_2.OBJ, and the default dialog only displays .TXT files. Of course, navigate to the directory where the EX1_2.OBJ file is stored and select it, as shown in figure 9.

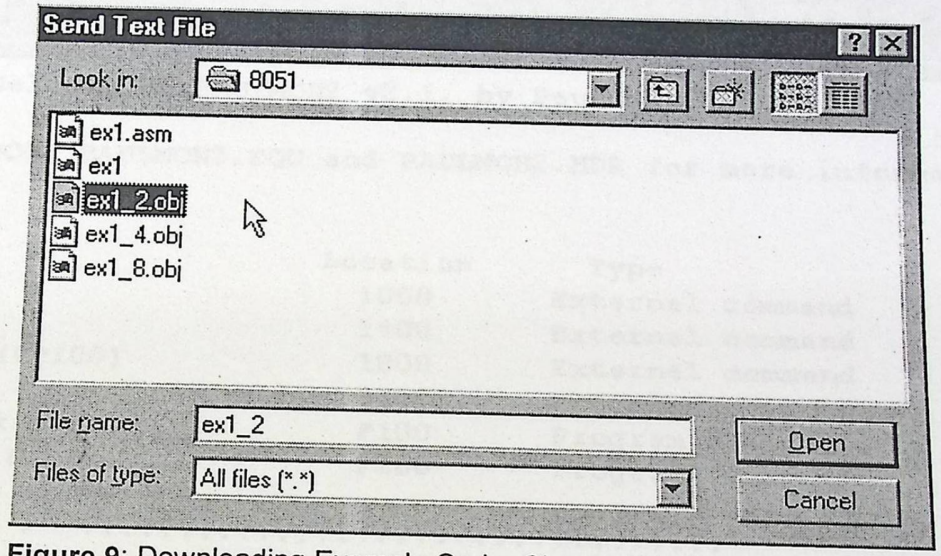


Figure 9: Downloading Example Code, Choose File To Send

Once you've opened the file, Hyperterminal will begin sending it. A carrot "^" character will appear, which is in indication from PAULMON2 that is saw the colon ":" character that begins an intel-hex download. At PAULMON2 receives each line of the file, it will print a dot to show the progress of the download. When the transfer is finished, PAULMON2 will print a summary of the download, and it will alert you to any errors in the reception of the data. After downloading EX1_2.OBJ, your Hyperterminal window should appear similar to figure 10.

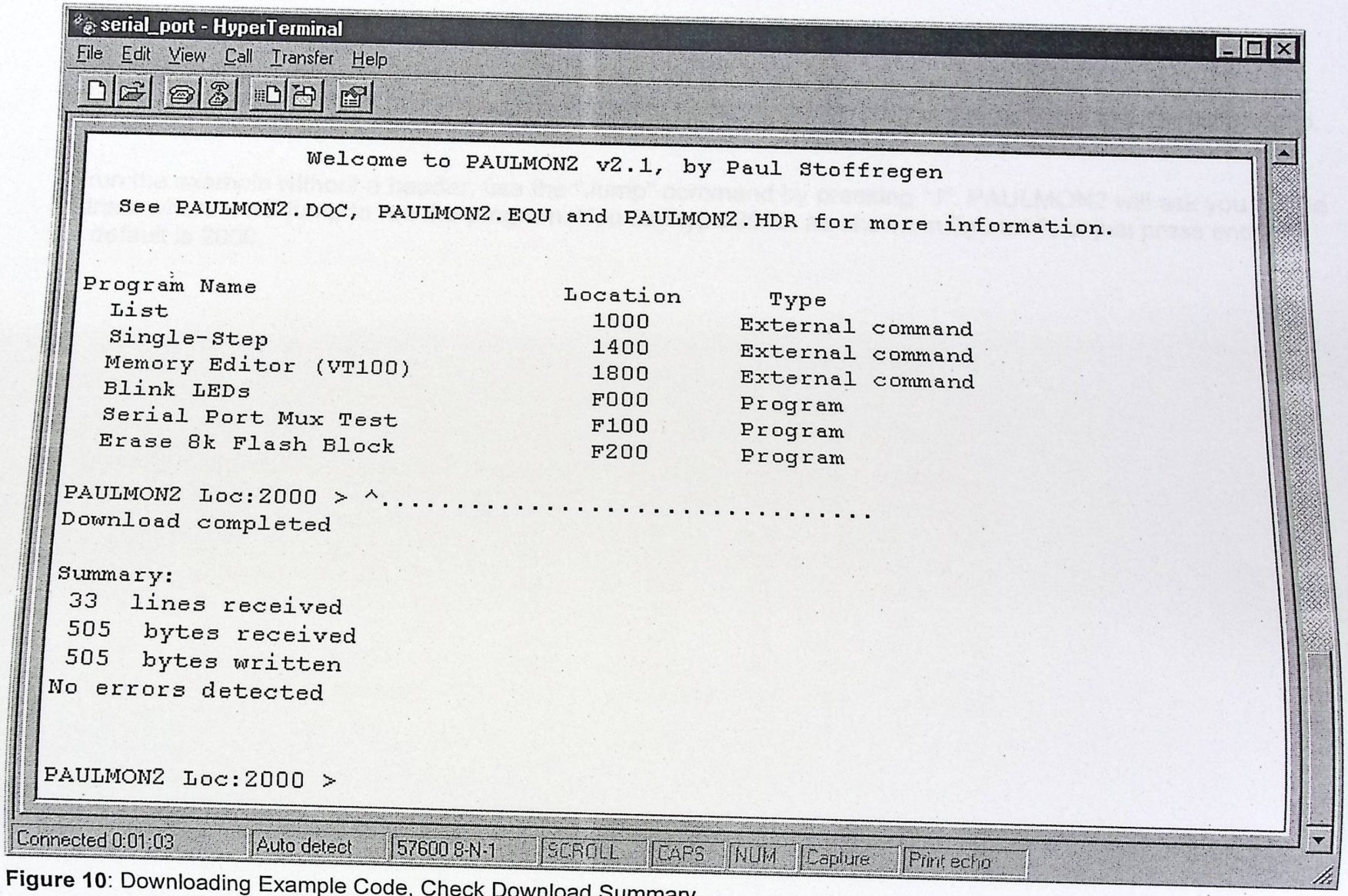


Figure 10: Downloading Example Code, Check Download Summary

The intel hex data format encodes the address where the bytes are to be stored within its data stream. Most PAULMON2 functions operate based in the "Loc:" address shown in the prompt, but the intel-hex download will always place the data at the location specified in the file. Here is a detailed description of the intel-hex data

format.

In this example program, the data happens to begin at location 0x2000. Some programs have a 64 byte header which is recognized by PAULMON2 and allow them to be shown in the menus and run from the "run" command. This example, however, does not have a header.

To run the example without a header, use the "Jump" command by pressing "J". PAULMON2 will ask you for the address where it will jump to run the program. You can type 2000, as shown in figure 11, or just press enter if the default is 2000.

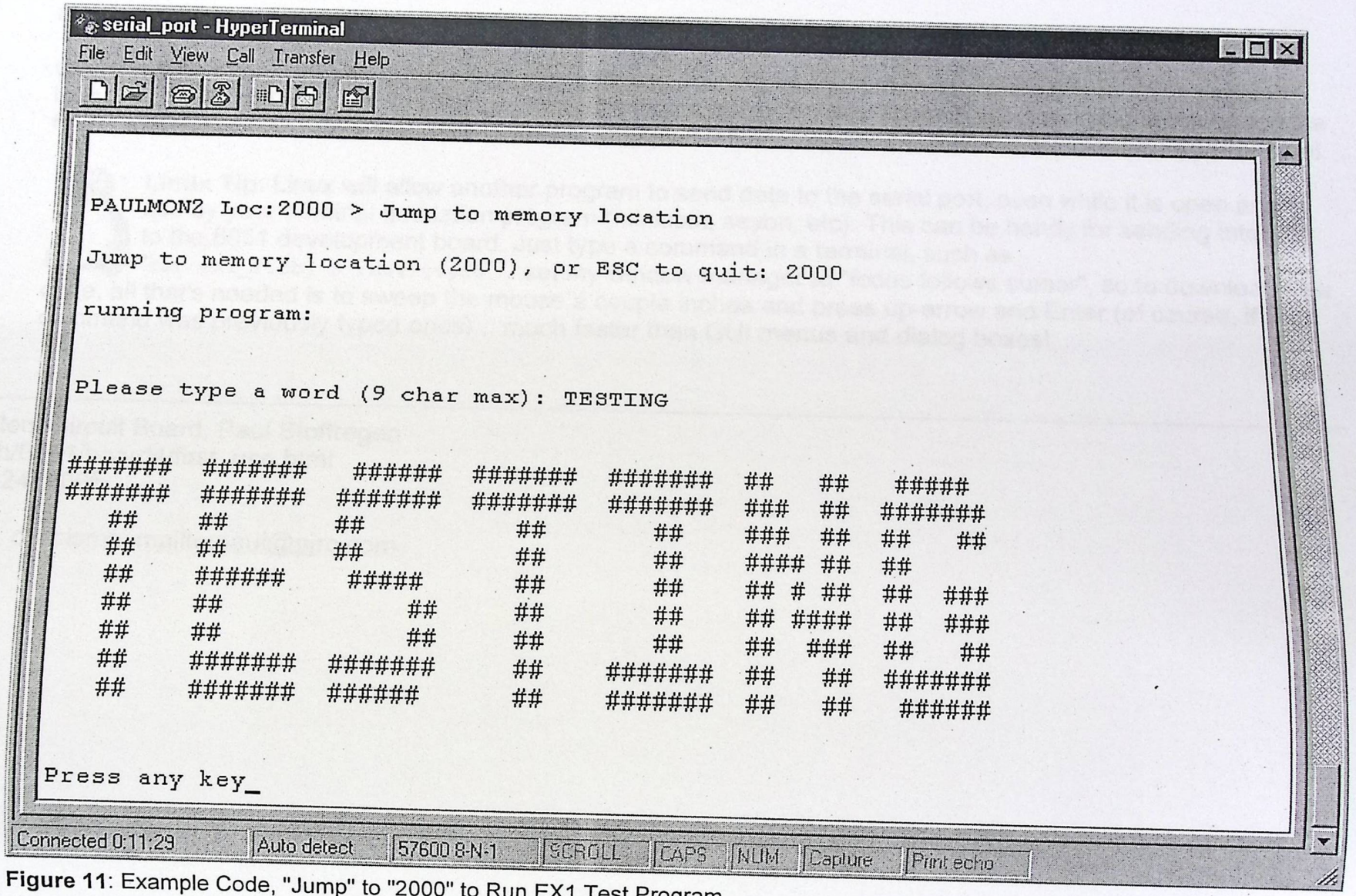
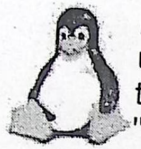


Figure 11: Example Code, "Jump" to "2000" to Run EX1 Test Program

The example program will ask you to type a word, and then it will print it in large block letters. Perhaps this isn't the most exciting program you could imagine, but the point has been to learn to use Hyperterminal to download and run a program. Now you're ready to create something more interesting to download to the 8051

development board.

Windows Tip: Hyperterminal sends data slowly. If you develop a large program and download times become long, you may need to switch to a better terminal emulator program. Many Windows users have recomended the free TeraTerm, which offers fast downloads and many advanced features not found in Microsoft's Hyperterminal.



Linux Tip: Linux will allow another program to send data to the serial port, even while it is open and in use by your terminal emulation program (minicom, seyon, etc). This can be handy for sending intel-hex to the 8051 development board. Just type a command in a terminal, such as "`cat ex1_2.obj > /dev/ttyS0`". I set my window manager to "focus follows cursor", so to download new code, all that's needed is to sweep the mouse a couple inches and press up-arrow and Enter (of course, if the command was previously typed once)... much faster than GUI menus and dialog boxes!

PJRC Electronic Projects Components Available Worldwide

PJRC Store

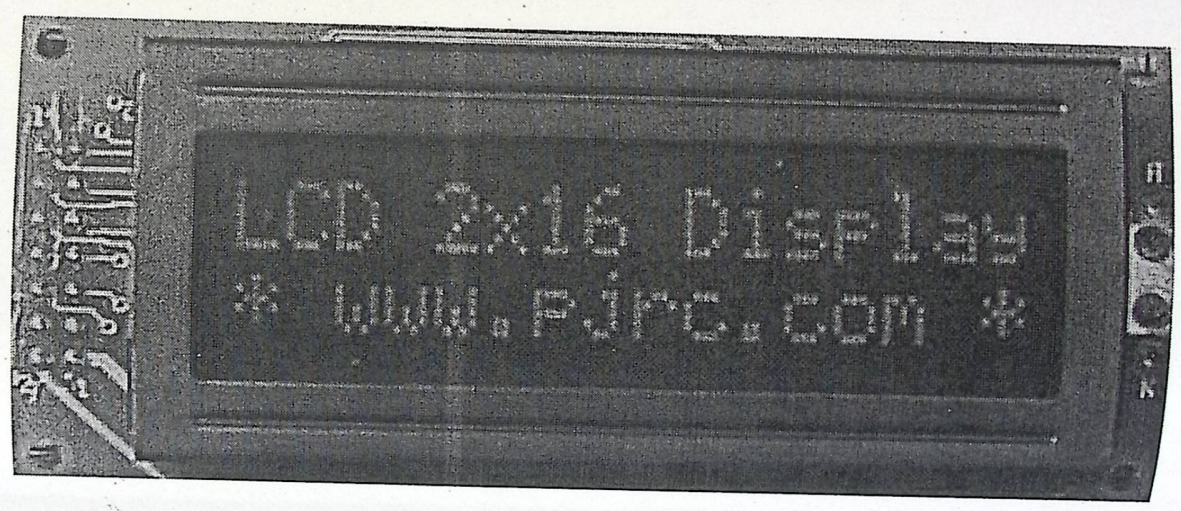
- Full Product List
- ⊞ **MP3 Player**
- ▶ **8051 Dev Board**
  - Assembled Board
  - LCD 20x2 Display
  - ▶ LCD 16x2 Display
  - Unassembled Kit
  - ⊞ **Individual Parts**
- ⊞ **Misc**
- Payment Options
- Privacy Policy

16x2 LCD for use with 8051 Development Board

Update, May 19, 2001: The 16x2 LCD is no longer available. We now have a larger and brighter 20x2 LCD available. This 20x2 LCD is recommended for new projects.

Sold Out	DEV_DISPLAY_16X2	Display For 8051 Development Board, 16x2 characters	$10

This backlight display plugs directly into the 8051 development board (rev 5). It displays 16 characters by 2 lines and uses a green backlight behind a transmissive LCD, to provide highly readable green characters against a dark blue background.



Related PJRC Project Pages

- 8051 development board
- Example code for this LCD

PJRC Store

- 8051 Dev Board, $79
- LCD 20x2 Display, $11
- Serial Cable, $5
- 12 Volt Power, $8
- More Components...

8051 Tools

- Main Page
- ⊞ Software
- ⊞ PAULMON Monitor
- ▶ Development Board
  - Features
  - Photos
  - Getting Started
- ⊞ Example Code
- Memory Map
- Ports & Pinouts
- Schematic & Parts
- Circuit Board
- Construction
- ⊞ Troubleshooting
- ▶ Old Versions
  - Revision List
  - ▶ Rev 4 (2002)
    - Photos & Features
    - Getting Started
    - Example Code
    - ▶ Memory Map

Memory Map

Begin	End	Code	Data	Application
0000	1FFF	PAULMON2	RAM	Program Variables Only
2000	7FFF	RAM		Program Code (development) and Program Variables
8000	F7FF	Flash ROM		Program Code and Data Logging
F800	FEFF	Peripherals		LCD, I/O Pins, Bus Expansion Connector
FF00	FFFF	Unused		Reduces Power (P2 defaults to 0xFF)

With SDCC, memory usage is controlled by command line options given to SDCC when it links the code. These options are typically written in the project Makefile. For example, "--code-loc 0x2000" would cause the linker to place your code at the beginning of the RAM which can be used for downloading code, and "--xram-loc 0x6000" would instruct the linker to place all your "xdata" variables beginning at 0x6000.

With AS31, your program's location is controlled by the ".org" directive within the code. Typically a ".equ" directive defines a constant near the top of the code, and subsequent ".org" directives utilize this constant to allow the code to be "moved" more easily.

Flash ROM must be erased before it is written. When you download code built between 8000 to F7FF, PAULMON2 will automatically program your code into Flash ROM (Flash programming is a procedure different from simply writing to RAM). Flash programming can only turn 1's into 0's. To turn programmed 0's back into 1's, you must erase the flash chip ('Z' Command) or a 4k sector of the flash chip. The erasure causes all the bytes in the erased area to turn back into 0xFF. If you download a new version of your program "on top of" the

- Ports & Pinouts
- Schematic & Parts
- Circuit Board
- Construction
- Troubleshooting

⊞ Rev 3 (2001)
⊞ Rev 2 (1997)
⊞ Rev 1 (1992)
⊞ Code Library
⊞ 89C2051 Programmer
⊞ Other Resources

original, the result is usually incorrect and PAULMON2 will print an "Unable to write" error message. If this happens, simply erase the flash rom and try again (or use the RAM for development until your code is debugged).

Peripheral Mapping

Begin	End	Peripheral	Address	Access	Function
F800	F8FF	82C55 (A, B, C)	F800	RD / WR	Port A
			F801	RD / WR	Port B
			F802	RD / WR	Port C
			F803	WR Only	Config A,B,C
F900	F9FF	82C55 (D, E, F)	F900	RD / WR	Port D
			F901	RD / WR	Port E (LEDs)
			F902	RD / WR	Port F
			F903	WR Only	Config D,E,F
FA00	FAFF	User Expansion CS2 Signal Asserted Low	User Defined		
FB00	FBFF	User Expansion CS3 Signal Asserted Low	User Defined		
FC00	FCFF	User Expansion CS4 Signal Asserted Low	User Defined		
FD00	FDFF	User Expansion CS5 Signal Asserted Low	User Defined		

FE00	FEFF	LCD Port	FE00	WR Only	Command Register
			FE01	RD Only	Status Register
			FE02	WR Only	Display or CGRAM Buffer
			FE03	RD Only	

PJRC Store

- 8051 Dev Board, $79
- LCD 20x2 Display, $11
- Serial Cable, $5
- 12 Volt Power, $8
- More Components...

8051 Tools

- Main Page
- **Software**
- **PAULMON Monitor**
- **Development Board**
  - Features
  - Photos
  - Getting Started
  - **Example Code**
    - LED Blink, C
    - LED Blink, Asm
    - Demo Programs
    - Auto Startup
    - LCD Display
    - Timers
    - 82C55 I/O
    - Keil C51
    - More Code...
  - Memory Map
  - Ports & Pinouts
  - Schematic & Parts
  - Circuit Board
  - Construction
  - **Troubleshooting**

Automatic Start-Up Of Your Application

Eventually, you will want to make your own application run automatically when you boot the board, instead of PAULMON2's menus. To accomplish this, you must do two things:

1. Compile your application for the flash rom.
2. Download the autostart code from this page.

When you reboot the board, the autostart code will run automatically. It will configure the baud rate, set up the interrupt vectors, and then jump to your code. The board will be permanently dedicated to run your application, unless you use the flash erase jumper to restore it to the original, erased condition.

Default Settings: Code at 0x8000, Baud Rate is 115200 Bits/sec

If the default settings are ok, then you can use the autostart example code without any modification. Just download the autostart.hex file (zip format).

This code resides in the memory from 0xF600 to 0xF7FF (the last 512 bytes of the flash), so your application must not use that memory. Of course, the full source code is available if you need that memory, or the default settings are not correct for your application.

Configuring Your Code's Build Location

If you are using SDCC, just change the --code-loc parameter in the Makefile, so that your application is built to reside entirely inside the flash rom, and rebuild the code (often times "make clean" or deleting the .hex file is

⊞ Old Versions
⊞ Code Library
⊞ 89C2051 Programmer
⊞ Other Resources

needed). Usually 0x8000 is the best choice, so that your code begins at the beginning of the flash rom and has the maximum amount of space available. It should look like this:

```
--code-loc 0x8000
```

If you are using AS31, you will need to place a ".org" directive at the beginning of your code. For example:

```
        .org    0x8000
```

Before downloading the autostart.hex file, try using PAULMON2's "J" (Jump) command to jump to your code at *0x8000*. If that works and you are ready to permanently dedicate the board to automatically running your application, then you are ready to download the autostart.hex file.

Erasing The Flash

After you have downloaded this "autostart.hex" file, your board will be permanently dedicated to running your own application code and PAULMON2's interactive menus will no longer be available.

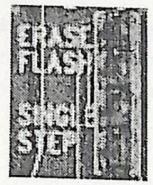


Figure 1: Once you've downloaded AUTOSTART.HEX, this FLASH ERASE jumper is the only way to return to PAULMON2's interactive menus.

If you wish to return to PAULMON2's menus, you must use the flash erase jumper to erase the flash rom. Short the upper two pins together and *keep them shorted while you press the RESET button*. When the board resets, the flash rom will be erased and you will be returned to the PAULMON2 prompt, where you may further develop your application or use the board for a new purpose.

Customizing Baud Rate Setup

When you boot the board, a portion of the autostart code inserts some bytes into memory to configure the baud

rate that PAULMON2 will use to initialize the serial port. Several common baud rates are defined in the code, so you can simply uncomment one if your baud rate in listed:

```
.equ    baud_const, 255         ;115200 baud w/ 22.1182 MHz
;.equ   baud_const, 254         ;57600 baud w/ 22.1182 MHz
;.equ   baud_const, 253         ;38400 baud w/ 22.1182 MHz
;.equ   baud_const, 250         ;19200 baud w/ 22.1182 MHz
;.equ   baud_const, 244         ;9600 baud w/ 22.1182 MHz
```

If you need a different baud rate, you must compute the value for "baud_const", using this formula:

baud_const = 256 - (115200 / baud_rate)

Customizing Application Startup

The default setting is to configured for your application code to begin at 0x8000, which is the beginning of the flash memory. The autostart code is located from 0xF600 to 0xF7FF, the end of the flash memory. However, it is easy to change these by editing constants in the code:

```
; set this to the location where your application code begins.
; With SDCC, this is the value you used for --code-loc
; With AS31, this is the .org locations where your code was built

.equ    application, 0x8000


; set this to where you'd like these two auto-startup routines
; to be placed in the flash.  Normally 0xF600 is a good choice,
; to place this at the end of the flash (where it is least likely
; to conflict with the space for your application).

.equ    location, 0xF600
```

Of course, the complete source code is provided, so you can edit the code to perform any customized startup you need. The normal code simply writes seven LJMPs into RAM in the 0x2000, so that the interrupts will jump to your interrupt code in the flash rom, and then jumps to your program's starting location.

For example, if the additional LJMP interrupt latency is not acceptable, you could modify the code to instead copy your interrupt handler to the RAM. Or perhaps you are not using interrupts and you want to avoid writing to the RAM during a reboot, in which case you could simply delete that section.

Download The Autostart Code

- autostart.hex - Ready to use HEX file with default settings.
- autostart.asm - Source code with comments.
- autostart.zip - ZIP archive with both of these.

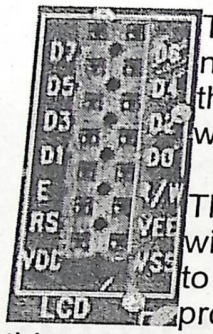
PJRC Store

- 8051 Dev Board, $79
- LCD 20x2 Display, $11
- Serial Cable, $5
- 12 Volt Power, $8
- More Components...

8051 Tools

- Main Page
- **Software**
- **PAULMON Monitor**
- **Development Board**
  - Features
  - Photos
  - Getting Started
- **Example Code**
- Memory Map
- Ports & Pinouts
- Schematic & Parts
- Circuit Board
- Construction
- **Troubleshooting**
- **Old Versions**
  - Revision List
  - **Rev 4 (2002)**
    - Photos & Features
    - Getting Started
    - Example Code
    - Memory Map

LCD Port

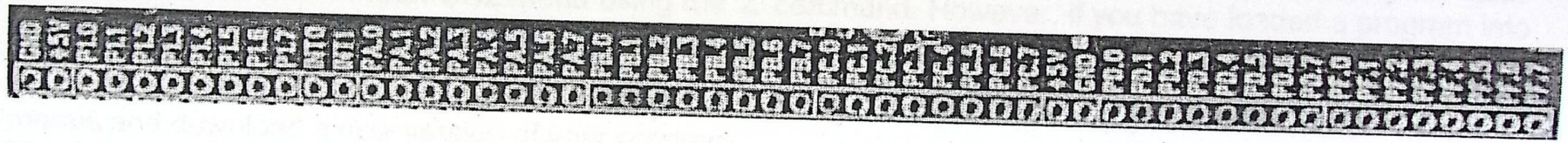


The LCD port provides the 14 standard signals required to interface to nearly all standard alpha-numeric character mode displays from 8 to 80 characters. **VDD** provides +5 volt regulated power to the display (**VSS** is Ground). **VEE** ranges from 0 (maximum intensity) to 2 volts (minimum intensity) with the adjustment of the variable resistor located just above next to the power input.

The **E** signal is an active high enable, which is asserted when the processor makes a memory access within the address range of 0xFE00 to 0xFEFF. **RS** and **R/W** are control lines for the display. In order to meet the timing requirements for all standard LCD displays, these are connected to the processor's address lines so they are asserted and remain stable while **E** is asserted. Because of this, separate locations are used to read and write to the LCD. See the memory map page for details.

I/O Lines

The 8051 development board provides 50 dedicated I/O lines, which are accessible along the top edge of the board's prototype construction area. Together with the 50 I/O lines, 4 pads provide easy access to the regulated +5 volt power from the board's voltage regulator.



The **P1.0** through **P1.7** are connected directly to the 87C52's port #1. These pins are the easiest to use as single bits. In assembly, they are written using "CLR P1.4" or "SETB P1.4", and they are read using "MOV C, P1.4" (moves the bit value into the carry bit). In C (using SDCC, with #include <8051.h>), they are accessed using names such as "P1_4". For example: `if (!P1_3) printf("pin P1.3 is low");` The 8051 port pins are quasi-bidirectional, which essentially means that you must write a 1 (which is the default) to the pin to cause it to

- Ports & Pinouts
- Schematic & Parts
- Circuit Board
- Construction
- Troubleshooting

⊞ Rev 3 (2001)
⊞ Rev 2 (1997)
⊞ Rev 1 (1992)
⊞ Code Library
⊞ 89C2051 Programmer
⊞ Other Resources

act as an input.

Signals **INT0** and **INT1** connect to the 87C52's two interrupt pins. The 87C52 can be configured to execute and interrupt routine associated with each pin when it is low or when a falling edge occurs. In the low level sensitive setting, the interrupt service code usually takes some action which causes the hardware to stop driving the pin low, so that another interrupt does not immediately occur when the interrupt service code returns to the main program. If interrupt are not enabled, these pins can be accessed as ordinary P3.2 and P3.3 port pins. INT1 is also connected to the SINGLE STEP jumper and will be shorted to ground if that jumper is installed.

PA.0 to **PA.7**, **PB.0** to **PB.7**, and **PC.0** to **PC.7** are connected to a 82C55 chip mapped at 0xF800. **PD.0** to **PD.7** and **PF.0** to **PF.7** are connected to a second 82C55 chip mapped at 0xF900, and they correspond to ports A and C on that second chip, but that second chips ports are labeled D, E, and F to avoid confusion with the ports from the first 82C55 chip (port E connects to the 8 LEDs). Each 82C55 chip is controlled with 4 memory mapped locations, one to read or write each 8-bit port, and a 4th register to configure the 82C55 chip. See the memory map page for details. The 82C55 must first be configured by writing a byte to its config register, and then the three locations which access the ports may be used.

Jumpers

A 4-pin header may be used for two optional jumpers. The upper two pins may be shorted to erase the flash rom, and the lower two pins may be shorted to enable the single-step feature.

The **FLASH ERASE** jumper causes T1 (also P3.5, pin 17 on the 82C52) to be shorted to ground. During normal operation, this does not erase the flash. When PAULMON2 boots, it reads this pin and if it remains shorted for 256 consecutive reads, the PAULMON2 will erase the flash rom chip. Normally the flash rom is erased from the PAULMON2 menu using the 'Z' command. However, if you have loaded a program into the flash rom and used and "auto-start" header on it, PAULMON2 will jump to your code instead of presenting the normal menus on the serial port. If your code does not return back to the monitor, then you will be unable to get to the normal PAULMON2 menus and this pin will allow you to erase the chip you can can return to the menus and download a new version of your program.

The **SINGLE STEP** jumper shorts the INT1 interrupt pin to ground. This is required to use PAULMON2's single-step feature. The single-step operates by enabling interrupt #1 and using the 8051's feature where 1 instruction is always executed after and interrupt. This can be a nice way to "see" your code run, particularly if you are learning assembly. Due to the interrupt usage, it is rarely useful for debugging sophisticated applications.

Ports & Pinouts
- Schematic & Parts
- Circuit Board
- Construction
- Troubleshooting

Rev 3 (2001)
Rev 2 (1997)
Rev 1 (1992)
Code Library
89C2051 Programmer
Other Resources

act as an input.

Signals **INT0** and **INT1** connect to the 87C52's two interrupt pins. The 87C52 can be configured to execute and interrupt routine associated with each pin when it is low or when a falling edge occurs. In the low level sensitive setting, the interrupt service code usually takes some action which causes the hardware to stop driving the pin low, so that another interrupt does not immediately occur when the interrupt service code returns to the main program. If interrupt are not enabled, these pins can be accessed as ordinary P3.2 and P3.3 port pins. INT1 is also connected to the SINGLE STEP jumper and will be shorted to ground if that jumper is installed.

PA.0 to **PA.7**, **PB.0** to **PB.7**, and **PC.0** to **PC.7** are connected to a 82C55 chip mapped at 0xF800. **PD.0** to **PD.7** and **PF.0** to **PF.7** are connected to a second 82C55 chip mapped at 0xF900, and they correspond to ports A and C on that second chip, but that second chips ports are labeled D, E, and F to avoid confusion with the ports from the first 82C55 chip (port E connects to the 8 LEDs). Each 82C55 chip is controlled with 4 memory mapped locations, one to read or write each 8-bit port, and a 4th register to configure the 82C55 chip. See the memory map page for details. The 82C55 must first be configured by writing a byte to its config register, and then the three locations which access the ports may be used.

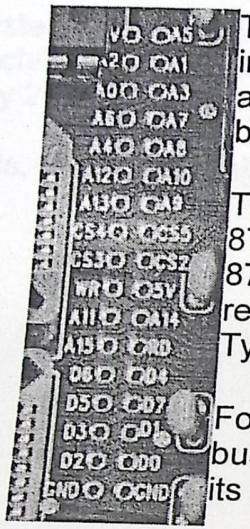
Jumpers

A 4-pin header may be used for two optional jumpers. The upper two pins may be shorted to erase the flash rom, and the lower two pins may be shorted to enable the single-step feature.

The **FLASH ERASE** jumper causes T1 (also P3.5, pin 17 on the 82C52) to be shorted to ground. During normal operation, this does not erase the flash. When PAULMON2 boots, it reads this pin and if it remains shorted for 256 consecutive reads, the PAULMON2 will erase the flash rom chip. Normally the flash rom is erased from the PAULMON2 menu using the 'Z' command. However, if you have loaded a program into the flash rom and used and "auto-start" header on it, PAULMON2 will jump to your code instead of presenting the normal menus on the serial port. If your code does not return back to the monitor, then you will be unable to get to the normal PAULMON2 menus and this pin will allow you to erase the chip you can can return to the menus and download a new version of your program.

The **SINGLE STEP** jumper shorts the INT1 interrupt pin to ground. This is required to use PAULMON2's single-step feature. The single-step operates by enabling interrupt #1 and using the 8051's feature where 1 instruction is always executed after and interrupt. This can be a nice way to "see" your code run, particularly if you are learning assembly. Due to the interrupt usage, it is rarely useful for debugging sophisticated applications.

Bus Expansion Signals

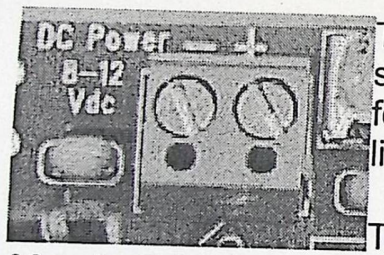


The 8051 bus signals are available on 34 pins in the center of the board. These signals are intended to be used with bus-style peripherals, such as UARTs and A/D converters. All 16 address and 8 data signals are provided. Two **GND** and two **5V** pins provide 5 volt regulated power from the board's voltage regulator.

The **WR** and **RD** signals are active low strobes for write and read. **WR** is connected directly to the 87C52's WR pin, but **RD** is connected to the 74AC08 AND gate. **RD** is asserted low with either the 87C52's PSEN or RD signal is asserted. This means that either MOVX or MOVC may be used to read your connected peripheral chips. Some peripheral chips call their read pin OE (output enable). Typically, the **RD** pin can connect directly to the peripheral's OE pin.

Four chip select signals, **CS2**, **CS3**, **CS4**, and **CS5** are provided to allow easy connection of most bus-style peripheral chips. Each of these signals is asserted low when and access is made within its 256 byte range. See the memory map page for details.

Power Input



The board accepts unregulated DC voltage, between 8 to 12 volts. A terminal block with screws allows a wide range of wires sizes to easily attach to the board without the need for a connector. A 1N5819 diode protects against reverse polarity, and a standard 7805 linear voltage regulator creates the regulated 5 volts needed by the board's circuitry.

Though the printed maximum voltage is only 12 volts, the board can actually accept up to 30 volts DC. Higher voltages will cause the 7805 voltage regulator to become hot. The 7805 includes automatic thermal shutdown, but it can become very hot before this upper limit is reached, so caution should be observed if a higher input voltage is used.

The board requires approximately 50 mA when executing code from the flash rom and communicating with a PC on the serial port. Each LED adds about 4 mA. If a LCD with a backlight is used, the backlight will consume considerable current. The 16x2 LCD from PJRC uses approximately 250 mA for its backlight. Additional current also causes the 7805 to heat up, so the board should not be run with more than 12 volts if a LCD backlight is

used.

8051 Development System Circuit Board, Paul Stoffregen
http://www.pjrc.com/tech/8051/board4/pinouts.html
Last updated: February 24, 2005
Status: finished
Suggestions, comments, criticisms: mailto:paul@pjrc.com

PJRC Store

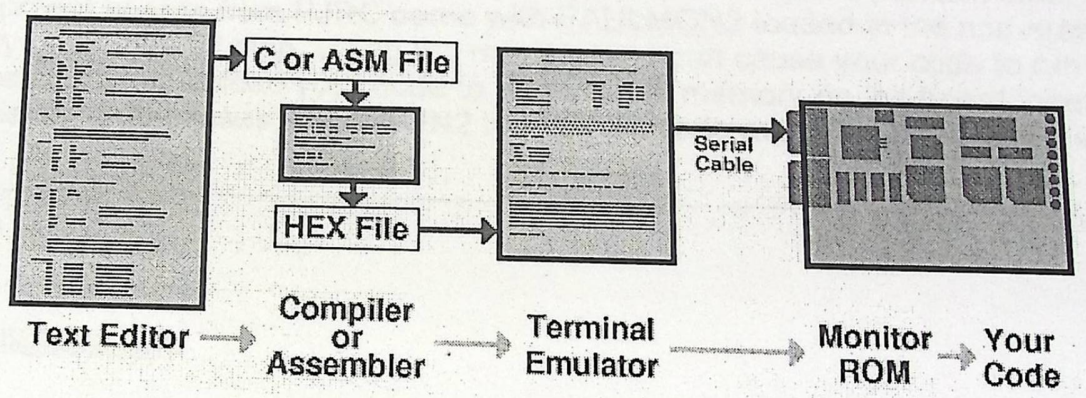
- 8051 Dev Board, $79
- LCD 20x2 Display, $11
- Serial Cable, $5
- 12 Volt Power, $8
- More Components...

8051 Tools

- Main Page
- ▶ **Software**
  - ▶ Overview
  - Linux AS31/SDCC
  - Windows AS31/SDCC
  - AS31 Manual
  - AS31 Inst. List
  - Old AS31
- ⊞ PAULMON Monitor
- ⊞ Development Board
- ⊞ Code Library
- ⊞ 89C2051 Programmer
- ⊞ Other Resources

8051 Software Tools Overview

To develop your 8051-based project, you will need three PC-based programs, and a development board with a monitor ROM. Here is the typical data flow using these tools:



For each tool, there you may choose on of several options. The basic tools are:

1. **Text Editor**
   This is where you will compose the code that ultimately will run on your 8051 board and make you project function. All PC operating systems include a text editor, and there are many free text editors available on the internet. PJRC does not provide a text editor. All Microsoft Windows systems have **NOTEPAD**, which is a very simple editor. Linux systems usually have ***VI*** and **EMACS** installed, as well as several others.
2. **Compiler or Assembler**
   Your source code will be turned into a .HEX file by either an Assembler or Compiler, depending on your choice of programming language. PJRC provides free downloads of the AS31 assembler

and SDCC C Compiler. These free tools are available for Linux-based Systems and Microsoft Windows. It is also possible to use other compilers, such as the Keil C compiler.

3. **Terminal Emulator**

   To communicate with your 8051 board, you must run a terminal emulator program. You will be able to transmit your .HEX file to the board and run its code, observe its results and information it may send to the serial port, and you can also use the terminal emulator to examine and manipulate memory with the Monitor ROM. Microsoft provides **HyperTerminal** with windows (often it must be installed from the windows cdrom using "add/remove programs"). Linux distributions usually provide **minicom**.

4. **Monitor ROM**

   The Monitor ROM is 8051 code that runs when your board boots. It provides interactive menus that allow you to download code, run it, manipulate memory and perform other functions. All 8051 development boards from PJRC come with PAULMON2 loaded in the non-erasable internal memory of the 87C52 chip. Using the monitor, you can cause your code to run on the board. It is also possible to download your code to non-volatile memory on the board together with a "auto-start header" that causes PAULMON2 to run your code automatically when the board boots.

PJRC Store

- 8051 Dev Board, $79
- LCD 20x2 Display, $11
- Serial Cable, $5
- 12 Volt Power, $8
- More Components...

8051 Tools

- Main Page
- ⊞ Software
- ⊞ PAULMON Monitor
- ▶ Development Board
  - Features
  - Photos
  - Getting Started
  - ▶ Example Code
    - LED Blink, C
    - LED Blink, Asm
    - Demo Programs
    - Auto Startup
    - ▶ LCD Display
    - Timers
    - 82C55 I/O
    - Keil C51
    - More Code...
  - Memory Map
  - Ports & Pinouts
  - Schematic & Parts
  - Circuit Board
  - Construction
  - ⊞ Troubleshooting

Example Code To Use The LCD Display

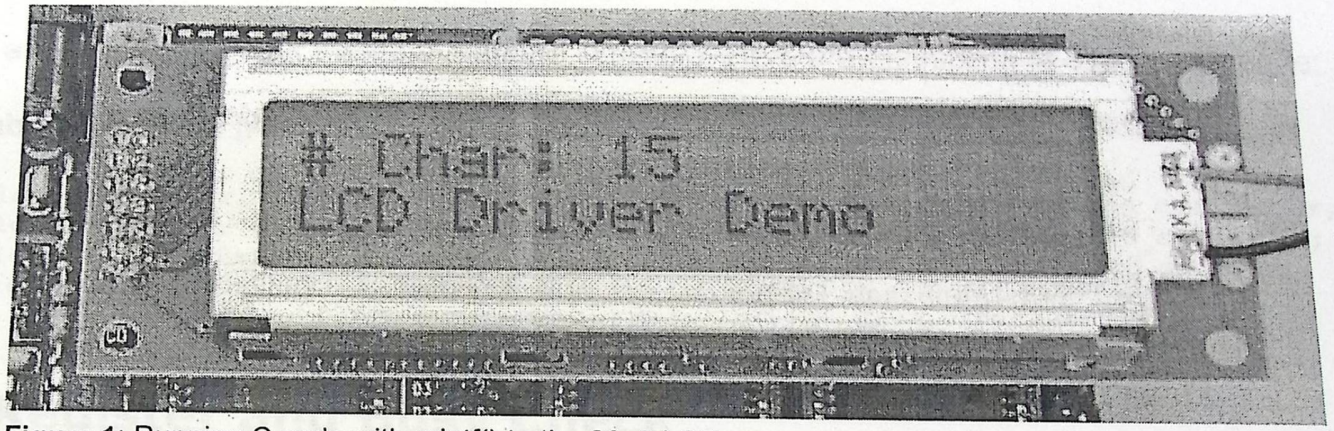


Figure 1: Running C code with printf() to the 20x2 LCD

Here are several examples of how to use the LCD port with any standard character LCD.

- C Code Example, Including PRINTF to the LCD. This SDCC-based LCD demo code includes a collection of "driver" functions to access the LCD, and an example of using a custom putchar() to route printf output to either the LCD or serial port so you can use printf's nice number and string formatting features on the LCD. The included "lcd_driver.c" and "lcd_driver.h" files are intended to easily "drop into" your project, and both C-based (easy to understand) and optimized assembly (faster) versions of the driver are provided.
- Assembly Language Example with Simple Routines (also as a ZIP File) - *A simple program that echos* whatever you type onto the display. Includes assembly routines *to print a character, print a string, move* cursor to X/Y position, clear screen, etc.
- Free C Code From Craig Hadady - *C code to copy whatever you type to the display. Includes support for*

⊞ Old Versions
⊞ Code Library
⊞ 89C2051 Programmer
⊞ Other Resources

scrolling.

How To Access The LCD Registers

This section describes how to directly access the LCD registers. It is not difficult, but if you only want to print normal text to the LCD, using the example code above is the easiest approach. However, if you have special needs, or you want to understand how the example code works, or you want to write your own driver for the LCD, then this section is for you.

The LCD is accessed using four memory mapped registers. Here are definitions to use in your code:

Assembly Language (AS31)

```
.equ    lcd_command_wr, 0xFE00
.equ    lcd_status_rd, 0xFE01
.equ    lcd_data_wr, 0xFE02
.equ    lcd_data_rd, 0xFE03
```

C Language (SDCC)

```
volatile xdata at 0xFE00 unsigned char lcd_command_wr;
volatile xdata at 0xFE01 unsigned char lcd_status_rd;
volatile xdata at 0xFE02 unsigned char lcd_data_wr;
volatile xdata at 0xFE03 unsigned char lcd_data_rd;
```

Status Register and Busy Check

Before writing to the LCD, you must wait for it to be available to receive data. Usually, it will not be busy, but there are times when it is performing internal operations and you must wait.

To check if the LCD is busy, simply read the lcd_status_rd register and check the most significant bit. If this bit is 1, the LCD is busy and you must wait to write. If it is a 0, then the LCD is ready.

Assembly Language (AS31)

```
        mov     dprt, #lcd_status_rd
lcd_wt: movx    a, @dptr
        jb      acc.7, lcd_wt
```

C Language (SDCC)

```
while (lcd_status_rd & 0x80)
    ; /* do nothing */
```

The lower 7 bits of the status register return the current address pointer. Normally, your code would set that address and "know" what it is by how many characters have been written, so the address read feature is rarely needed or used.

Commands For The LCD

Once the LCD is not busy, you can give it a command by simply writing a byte to the command register. There are many commands available. All commands are a single byte requiring only one write operation.

Command	Function
0x01	Clear display
0x02	Move cursor to home position
0x04 - 0x07	Set entry mode bit 0: 1 = Display is shifted, 0 = Not shifted bit 1: 1 = Cursor movement increases, 0 = Cursor movement decreases
0x08 - 0x0F	Set Display, Cursor, Blink On/Off bit 0: 1 = Blinking On, 0 = Blinking Off bit 1: 1 = Cursor On, 0 = Cursor Off bit 2: 1 = Display On, 0 = Display Off
0x10 - 0x1F	Set Shift mode bit 0: (unused) bit 1: (unused) bit 2: 1 = Right shift, 0 = Left shift bit 3: 1 = Display shift, 0 = Cursor move
0x20 - 0x3F	Configure Display and Interface bit 0: (unused) bit 1: (unused) bit 2: 1 = 5x10 pixel chars, 0 = 5x7 pixel chars bit 3: 1 = 2 line display, 0 = 1 line display bit 4: 1 = 8 bit interface, 0 = 4 bit interface
0x40 - 0x7F	Set Character Generator RAM address bits 0-6: address in character generator RAM
0x80 - 0xFF	Set Display Buffer RAM address bits 0-7: address in display buffer RAM

When first initializing the LCD, you would write the 0x38 command first. This configures the interface for 8 bits, enables both lines, and sets the font to 5x7 pixels which is the correct size for most LCDs (if you have a rare 5x10 pixel LCD, you would change this to 0x3C.

Normally, the command 0x0C is written to turn on the LCD and disable showing a cursor. However, if a solid or blinking cursor is desired, one of the two lower bits may be set.

Then, the shift mode command is sent, usually to 0x10, to disable shifting the display and instead simply moving the cursor. This makes the LCD operate in a manner similar to a computer screen, where the characters stay where you put them and the cursor moves forward. However, you can turn on the shift mode to make the LCD act like a reader board, where your characters always appear at the same place and the whole display shifts every time you write a character.

Before writing to the display memory, the command to set the address in the memory is usually sent. On most 2 line LCDs, address 0x00 (command 0x80) set the cursor to the beginning of the first line, and address 0x40 (command 0xC0) sets the address to the beginning of the second line.

Writing to the Display

Once the LCD is initialized, you make it display text by simply writing to the lcd_data_wr register. The position where the character will appear depends on the current address, which you set with Set Display Buffer RAM address command. Remember that there are 128 possible addresses, but only 40 of them will correspond to on-screen display on a 20x2 LCD.

Writing bytes 0x20 to 0x7D will display ASCII characters. Many LCDs render arrows for 0x7E and 0x7F, and a variety of other symbols and characters from 0x80 to 0xFF. Bytes 0x00 to 0x0F will render user defined characters.

Each time you write to lcd_data_wr, the display address will be incremented or decremented, depending on the shift mode.

It is also possible to read the contents of the display buffer RAM, though this is rarely done.

Creating Your Own Custom Characters

By sending the Set Character Generator RAM address, you can access a small RAM used to load custom characters. All writes to lcd_data_wr will write into the custom character RAM rather than the display buffer RAM, so you will need to issue the Set Display Buffer RAM address after you have finished loading your custom characters.

In 7x5 pixel mode, eight custom characters can be stored. Each takes 8 bytes of memory, where only the lower 5 bits are used and the 8th byte is unused.

After loading your custom characters, you can cause them to display by writing bytes 0x00 to 0x07.

More Information

The Hitachi HD44780 Datasheet is a complete reference for the operation of the controller chip on most character mode LCDs.

LCDs with this controller chip, or compatible chips from other manufacturers, are very common. Here are links to other excellent pages with information about using the controller chip:

- Dincer Aydin's JavaScript LCD Simulator
- Ian Harries HD44780-Based LCD Modules
- Peter Ouwehand's How to control a HD44780-based Character-LCD

If you know of other sites that should be on this list, please contact me.

PJRC Store

- 8051 Dev Board, $79
- LCD 20x2 Display, $11
- Serial Cable, $5
- 12 Volt Power, $8
- More Components...

8051 Tools

- Main Page
- ⊞ **Software**
- ⊞ **PAULMON Monitor**
- ▶ **Development Board**
  - Features
  - Photos
  - Getting Started
  - ▶ **Example Code**
    - LED Blink, C
    - LED Blink, Asm
    - Demo Programs
    - Auto Startup
    - LCD Display
    - Timers
    - ▶ 82C55 I/O
    - Keil C51
    - More Code...
  - Memory Map
  - Ports & Pinouts
  - Schematic & Parts
  - Circuit Board
  - Construction
  - ⊞ **Troubleshooting**

Using the 82C55 I/O Chips

Most of the I/O lines on the 8051 development board are provided by two 82C55 programmable peripherial I/O chips. This page provides detailed instructions and examples for using these I/O chips in your application. Before reading these examples, the simple LED blink examples should be read.

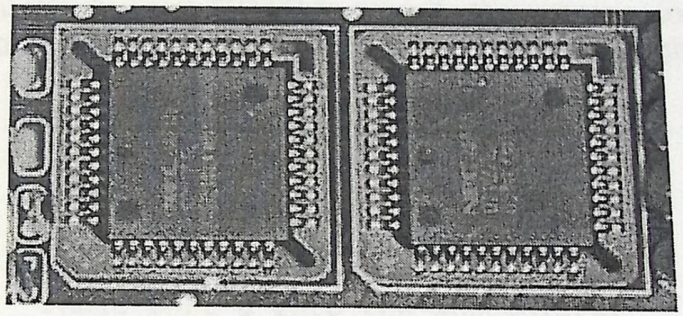


Figure 1: Two 82C55 Chips Provide 40 I/O Lines and 8 LEDs

The chips shown in figure 1 are NEC UPD71055L. Despite the strange part number, these are NEC's version of the 82C55 chip. They are labeled D71055L on the top of each chip.

Memory Mapped Peripheral Access

Both 82C55 chips are external memory mapped peripherals. This means you must access them using certain memory locations using the MOVX instruction. Each chip has byte-wide registers that you access. These definitions allow your code to access the 82C55 chips.

⊞ Old Versions
⊞ Code Library
⊞ 89C2051 Programmer
⊞ Other Resources

Assembly Language (AS31)

```
.equ  port_a, 0xF800
.equ  port_b, 0xF801
.equ  port_c, 0xF802
.equ  port_abc_pgm, 0xF803
.equ  port_d, 0xF900
.equ  port_e, 0xF901
.equ  port_f, 0xF902
.equ  port_def_pgm, 0xF903
```

C Language (SDCC)

```
volatile xdata at 0xF800 unsigned char p82c55_port_a;
volatile xdata at 0xF801 unsigned char p82c55_port_b;
volatile xdata at 0xF802 unsigned char p82c55_port_c;
volatile xdata at 0xF803 unsigned char p82c55_abc_config;
volatile xdata at 0xF900 unsigned char p82c55_port_d;
volatile xdata at 0xF901 unsigned char p82c55_port_e;
volatile xdata at 0xF902 unsigned char p82c55_port_f;
volatile xdata at 0xF903 unsigned char p82c55_def_config;
```

Since the 82C55 chips are connected to the exteral memory bus, they must be accessed using the MOVX instruction. The MOVX instruction always uses DPTR to specify the memory location, and data is always transfered to/from the accumulator. Here is an example of how the 82C55 registers are accessed using assembly language:

```
do_some_io:
        mov     dptr, #port_a
        mov     a, #0xA2
        movx    @dptr, a        ;Write 0xA2 to Port A
        mov     dptr, #port_b
        movx    a, @dptr        ;Read Port B into the accumulator
        ret
```

When programming in C, the compiler takes care of handling the registers, so all you need to do is read the 82C55 registers as if they were ordinary global variables. Here is a simple example, equivilant to the assembly code above (the compile may generate slightly different assembly code):

```
unsigned char do_some_io(void)
{
        p82c55_port_a = 0xA2;   /* Write 0xA2 to Port A */
        return(p82c55_port_b);  /* Read Port B and return it */
}
```

Using SDCC, the variable declaration gives the compiler all the information it needed to "know" how to access the 82C55 chip. The "volatile" keyword tells the compiler's optimizer to always access the chip as shown. The

"xdata" keyword tells the compiler that the access is the external data bus. The special syntax "at 0xF800" tells the compiler that the variable is located in memory at 0xF800 (by default, the linker would choose a location). And, of course, the variable is an unsigned char, so it will be accessed as a single byte. The 8 definitions above will simply work with the 82C55 chips, but understanding the definitions is necessary if you need to create them for additional custom hardware.

Simple Input/Output (Mode 0) Configuration

Before you can access the 6 registers corresponding to the I/O pins, you must configure the 82C55 chips. Luckily, all you need to do to configure each chip is write a single byte to its configuration register. Here is a table with all of the commonly used configuration bytes:

Configuration Byte		Port A	Port B	Port C	Port C
Decimal	Hex	Bits 7:0	Bits 7:0	Bits 7:4	Bits 3:0
128	0x80	Output	Output	Output	Output
129	0x81	Output	Output	Output	Input
136	0x88	Output	Output	Input	Output
137	0x89	Output	Output	Input	Input
130	0x82	Output	Input	Output	Output
131	0x83	Output	Input	Output	Input
138	0x8A	Output	Input	Input	Output
139	0x8B	Output	Input	Input	Input
144	0x90	Input	Output	Output	Output

145	0x91	Input	Output	Output	Input
152	0x98	Input	Output	Input	Output
153	0x99	Input	Output	Input	Input
146	0x92	Input	Input	Output	Output
147	0x93	Input	Input	Output	Input
154	0x9A	Input	Input	Input	Output
155	0x9B	Input	Input	Input	Input

Each chip requires its own configuration byte, so the process for configuring ports D, E (leds) and F is the same.

When the system is reset, the 82C55 chips default to all inputs (configuration byte 0x9B). When configuring the 82C55 chips immediately following a system reset, a delay should be added. The reset signal is generated by a resistor and capacitor, and as the voltage falls, the 8051 will often begin executing code before the 82C55 reset pin detects the voltage change (it tends to "see" the high to low transition at a lower voltage). So a simple software delay before configuring the 82C55 chip should be used to allow time for the 82C55 to begin operating. The 82C55 will ignore the write to its configuration register if it is still in reset mode.

Once you have written the configuration byte, the three ports may be accessed simply by reading or writing their registers. Port C (or port E on the second 82C55 chip) may be configured where half of the pins are input and the other half output. In that case, 82C55 chip ignores the unneeded 4 bytes when you write to port C, and when you read port C, the undefined 4 bits should be ignored.

You may change the 82C55 configuration at any time by writing a new configuration byte. Doing this effectively resets the 82C55 chip. All pins defined as outputs by the new configuration byte are cleared to zero. This should be considered when designing hardware that interfaces to 82C55 pins in an applicaton where the configuration will be changed, rather than simply programmed once at startup. The output pins default to low immediately after the configuration byte is written to the 82C55.

TODO: explain the complex modes

TODO: single bit writes to port C

TODO: example code (any suggestions ??)

Using the 82C55 I/O Chips, Paul Stoffregen
http://www.pjrc.com/tech/8051/board5/82c55.html
Last updated: February 24, 2005
Status: finished
Suggestions, comments, criticisms: mailto:paul@pjrc.com

PJRC Store

- 8051 Dev Board, $79
- LCD 20x2 Display, $11
- Serial Cable, $5
- 12 Volt Power, $8
- More Components...

8051 Tools

- Main Page
- **Software**
  - Overview
  - Linux AS31/SDCC
  - Windows AS31/SDCC
  - AS31 Manual
  - AS31 Inst. List
  - Old AS31
- **PAULMON Monitor**
- **Development Board**
- **Code Library**
- **89C2051 Programmer**
- **Other Resources**

AS31, SDCC and GNU Make For Windows Systems

- Download SDCC C Compiler and AS31 Assembler and GNU Make Win32 Binaries (1.1M download)

Step 1: Old Version of Windows 95

Windows 98, Me, NT, 2000, and Windows 95 OSR 2 and later versions come with the required libraries. Skip this step if you are using any of these versions of Windows.

Very old versions of Windows 95, such as the one Microsoft actually shipped in 1995, lack the required C runtime libraries to run this program. If you have this old version, in all likelihood some other software you've installed has already updated your libraries. If not, you will need to install the Windows Library Update service pack (link to Microsoft's download page... scroll down to the "Service Packs" section).

These programs will not run on 16 bit platforms (Windows 3.11, MSDOS 6.x, etc).

Step 2: Unpack The ZIP File

Use your favorite ZIP extraction program (probably WinZip) to uncompress the ZIP file. The ZIP file contains several nested subdirectories, which must remain in the proper relative order for SDCC to function properly.

AS31 and Make do not require any support files, but they are included in SDCC's "BIN" subdirectory, so that they will be able to run with the PATH command in step 4.

Step 3: Move To C:\SDCC

It is recommended to move the SDCC directory created by the ZIP extraction to C:\SDCC. This step is needed because SDCC will search for its C library include files in \SDCC\SHARE\SDCC\INCLUDE and library object code in \SDCC\SHARE\SDCC\LIB. It is possible to pass command line options to SDCC to tell it where to find its library files, but installing it in C:\SDCC is much simpler.

This image shows the SDCC directory moved to the recommended location.

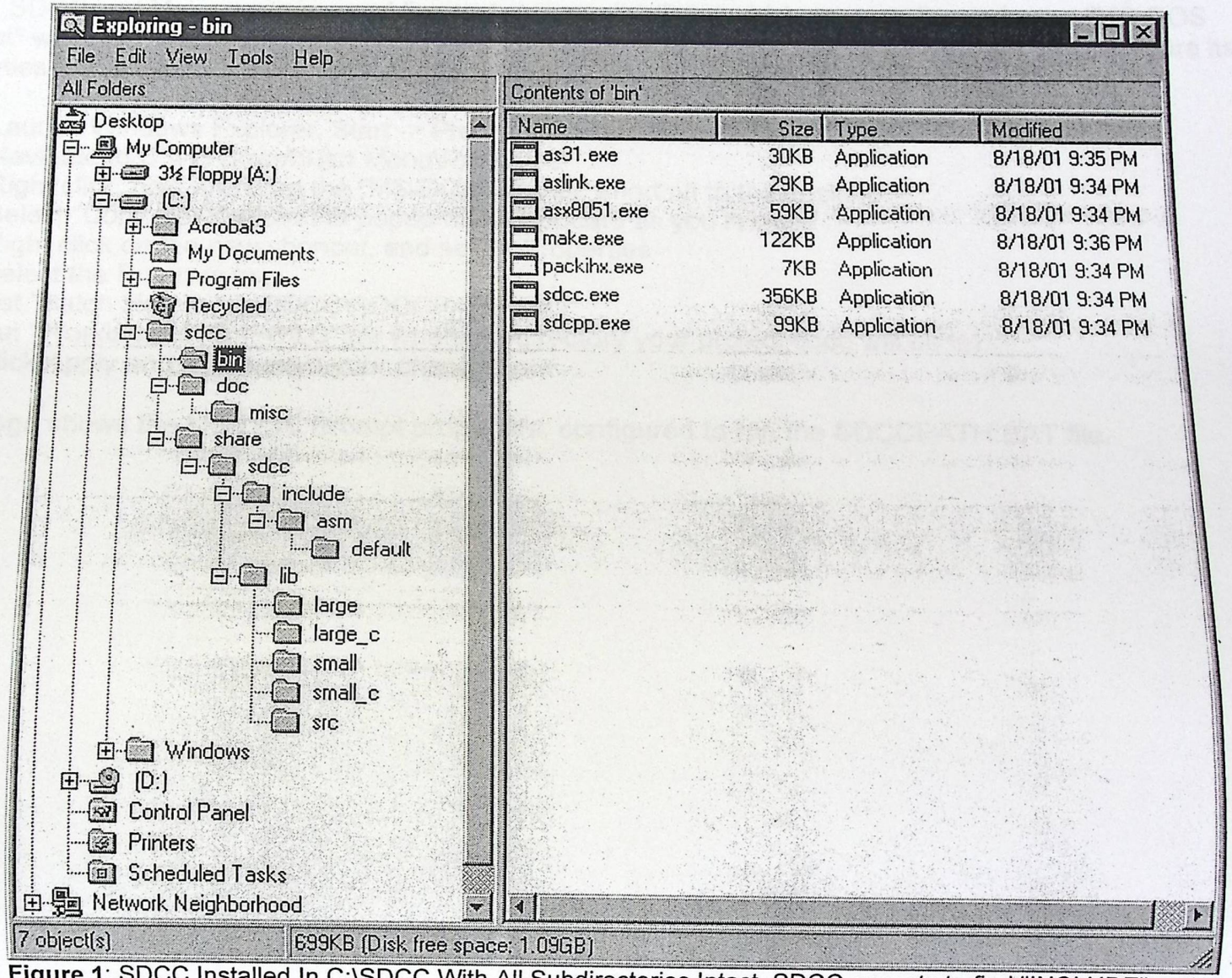


Figure 1: SDCC Installed In C:\SDCC With All Subdirectories Intact. *SDCC expects to find "INCLUDE" and* "LIB" in these locations.

Step 4: Set up MS-DOS Prompt Shortcut with correct PATH

AS31, SDCC and Make are command line text-only applications, which are usually run from a "MS-DOS Prompt" window. To run these, you should create a copy of the MS-DOS Prompt shortcut and configure its properties for SDCC.

1. Launch Windows Explorer, Start -> Programs -> Windows Explorer
2. Navigate to C:\Windows\Start Menu\Programs
3. Right click, hold and drag the "MS-DOS Prompt" shortcut to the desktop
4. Select "Copy Here" when the popup menu appears as you release
5. Right click on the new shortcut, and select Properties
6. Select the Program tag
7. Set "Batch file:" to "C:\SDCC\SDCCPATH.BAT"
8. Set "Working:" to "C:\SDCC" (or whatever directory your project code will be in).
9. Click Apply and OK to save your changes.

This image shows the MS-DOS Prompt properties, configured to run the SDCCPATH.BAT file.

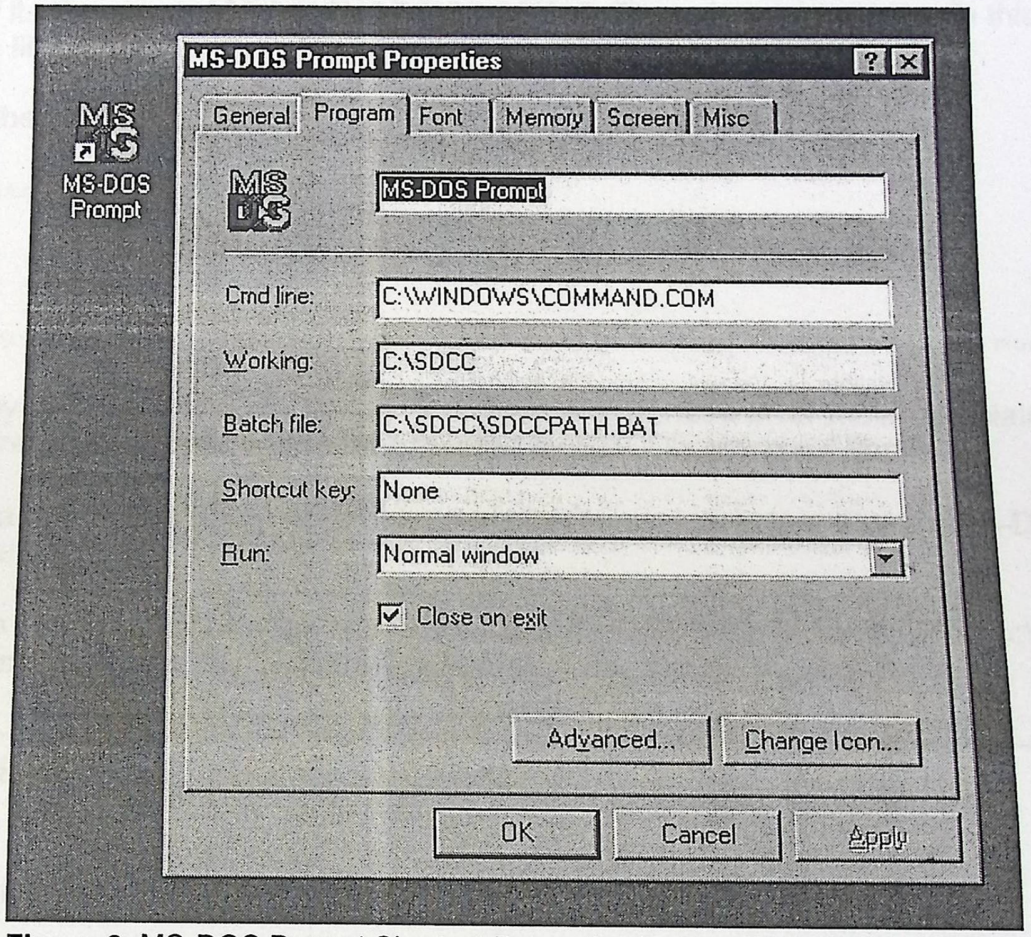


Figure 2: MS-DOS Prompt Shortcut Properties To Run SDCCPATH.BAT

Windows XP: Use the start menu, click "All Programs", navigate to Accessories and then click and hold with the right mouse button on "Command Prompt" and drag it to your desktop, and select "Copy Here" in the popup menu. Microsoft removed the ability to run a batch file from the properties dialog... you can run the batch file manually, or manually type a command to add SDCC to your path. *TODO: is there a way to do this* automatically in XP ???

Step 5: Your C-Based Project Should Be On C: Drive

SDCC will search for its libraries in \SDCC\SHARE\SDCC\LIB. There is no drive letter in this pathname, so SDCC will look for its library files on the same drive you're using.

If you get errors like these:

```
test.c:1:20: stdio.h: No such file or directory
```

or

```
?ASlink-Warning-Undefined Global '__sdcc_external_startup' referenced by module 'test'
```

then SDCC is probably not able to find its library or include files. These error message claim that SDCC can not find items which are clearly supposed to be provided by SDCC's standard library.

Check the directory placements (step 3) and make sure the current directory in your MS-DOS Prompt shows the same drive as where SDCC is installed.

The **putchar()** function required for printf is **NOT** provided in SDCC's libraries (this may change someday). You must write a putchar function. See the LED Blink Example for a simple putchar function.

Appendix B

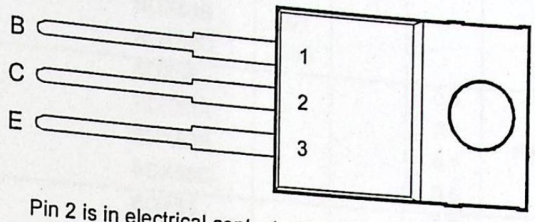
80851 Microcontroller Board

BDX53, BDX53A, BDX53B, BDX53C
NPN SILICON POWER DARLINGTONS

MAY 1989 - REVISED MARCH 1997

- **Designed for Complementary Use with BDX54, BDX54A, BDX54B and BDX54C**
- **60 W at 25°C Case Temperature**
- **8 A Continuous Collector Current**
- **Minimum h_{FE} of 750 at 3 V, 3 A**

TO-220 PACKAGE
(TOP VIEW)



Pin 2 is in electrical contact with the mounting base.

MDTRACA

Absolute maximum ratings at 25°C case temperature (unless otherwise noted)

RATING		SYMBOL	VALUE	UNIT
Collector-base voltage (I_E = 0)	BDX53	V_{CBO}	45	V
	BDX53A		60	
	BDX53B		80	
	BDX53C		100	
Collector-emitter voltage (I_B = 0)	BDX53	V_{CEO}	45	V
	BDX53A		60	
	BDX53B		80	
	BDX53C		100	
Emitter-base voltage		V_{EBO}	5	V
Continuous collector current		I_C	8	A
Continuous base current		I_B	0.2	A
Continuous device dissipation at (or below) 25°C case temperature (see Note 1)		P_{tot}	60	W
Continuous device dissipation at (or below) 25°C free air temperature (see Note 2)		P_{tot}	2	W
Operating junction temperature range		T_j	-65 to +150	°C
Operating temperature range		T_{stg}	-65 to +150	°C
Operating free-air temperature range		T_A	-65 to +150	°C

NOTES: 1. Derate linearly to 150°C case temperature at the rate of 0.48 W/°C.
2. Derate linearly to 150°C free air temperature at the rate of 16 mW/°C.

electrical characteristics at 25°C case temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS				MIN	TYP	MAX	UNIT
$V_{(BR)CEO}$	Collector-emitter breakdown voltage	I_C = 100 mA	I_B = 0	(see Note 3)	BDX53	45			V
					BDX53A	60			
					BDX53B	80			
					BDX53C	100			
I_{CEO}	Collector-emitter cut-off current	V_{CE} = 30 V	I_B = 0		BDX53			0.5	mA
		V_{CE} = 30 V	I_B = 0		BDX53A			0.5	
		V_{CE} = 40 V	I_B = 0		BDX53B			0.5	
		V_{CE} = 50 V	I_B = 0		BDX53C			0.5	
I_{CBO}	Collector cut-off current	V_{CB} = 45 V	I_E = 0		BDX53			0.2	mA
		V_{CB} = 60 V	I_E = 0		BDX53A			0.2	
		V_{CB} = 80 V	I_E = 0		BDX53B			0.2	
		V_{CB} = 100 V	I_E = 0		BDX53C			0.2	
I_{EBO}	Emitter cut-off current	V_{EB} = 5 V	I_C = 0					2	mA
h_{FE}	Forward current transfer ratio	V_{CE} = 3 V	I_C = 3 A	(see Notes 3 and 4)		750			
$V_{BE(sat)}$	Base-emitter saturation voltage	I_B = 12 mA	I_C = 3 A	(see Notes 3 and 4)				2.5	V
$V_{CE(sat)}$	Collector-emitter saturation voltage	I_B = 12 mA	I_C = 3 A	(see Notes 3 and 4)				2	V
V_{EC}	Parallel diode forward voltage	I_E = 3 A	I_B = 0					2.5	V

NOTES: 3. These parameters must be measured using pulse techniques, t_p = 300 µs, duty cycle ≤ 2%.
4. These parameters must be measured using voltage-sensing contacts, separate from the current carrying contacts.

thermal characteristics

PARAMETER		MIN	TYP	MAX	UNIT
$R_{\theta JC}$	Junction to case thermal resistance			2.08	°C/W
$R_{\theta JA}$	Junction to free air thermal resistance			62.5	°C/W

resistive-load-switching characteristics at 25°C case temperature

PARAMETER		TEST CONDITIONS †			MIN	TYP	MAX	UNIT
t_{on}	Turn-on time	I_C = 3 A	$I_{B(on)}$ = 12 mA	$I_{B(off)}$ = -12 mA		1		µs
t_{off}	Turn-off time	$V_{BE(off)}$ = -4.5 V	R_L = 10 Ω	t_p = 20 µs, dc ≤ 2%		5		µs

† Voltage and current values shown are nominal; exact values vary slightly with transistor parameters.

TYPICAL CHARACTERISTICS

TYPICAL DC CURRENT GAIN vs COLLECTOR CURRENT

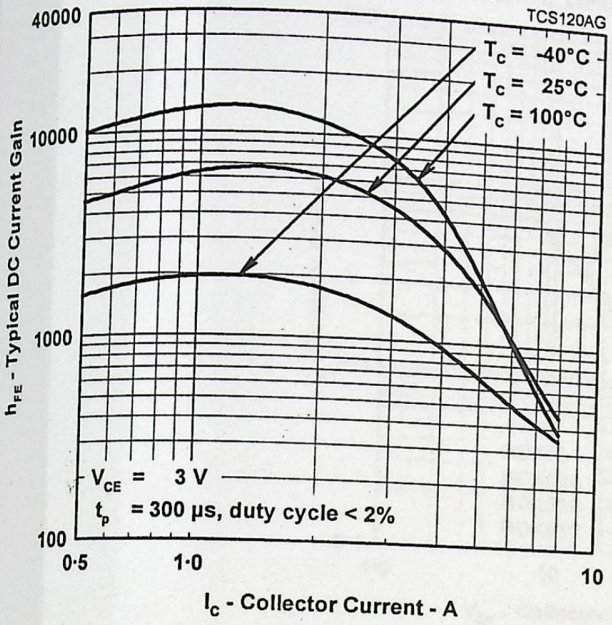


Figure 1.

COLLECTOR-EMITTER SATURATION VOLTAGE vs COLLECTOR CURRENT

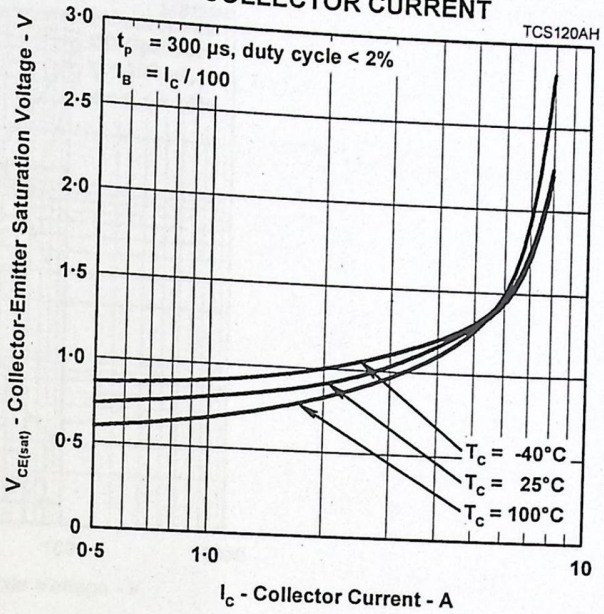


Figure 2.

BASE-EMITTER SATURATION VOLTAGE vs COLLECTOR CURRENT

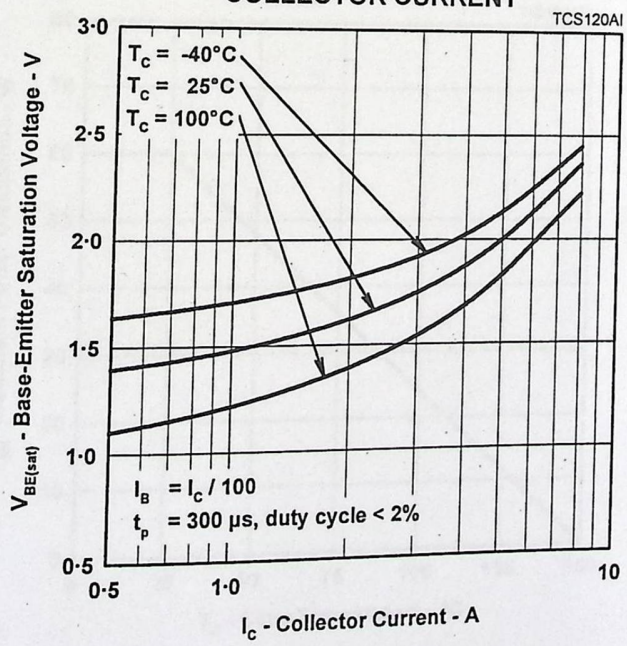


Figure 3.

MAXIMUM SAFE OPERATING REGIONS

MAXIMUM FORWARD-BIAS
SAFE OPERATING AREA

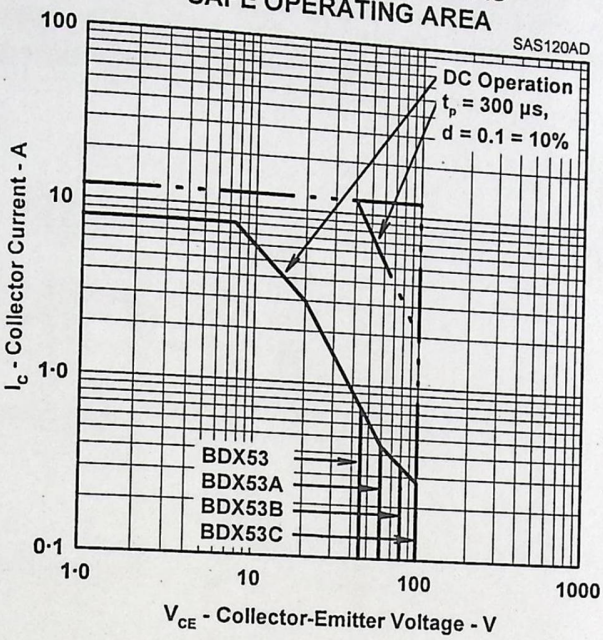


Figure 4.

THERMAL INFORMATION

MAXIMUM POWER DISSIPATION
vs
CASE TEMPERATURE

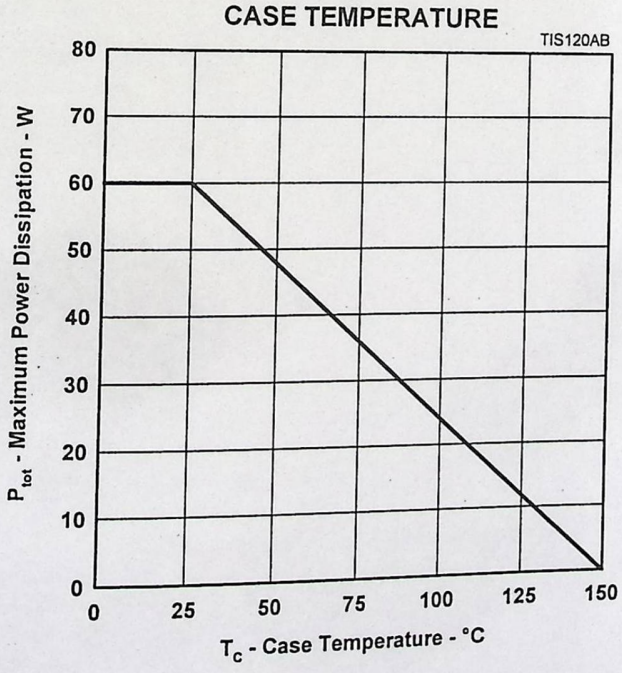


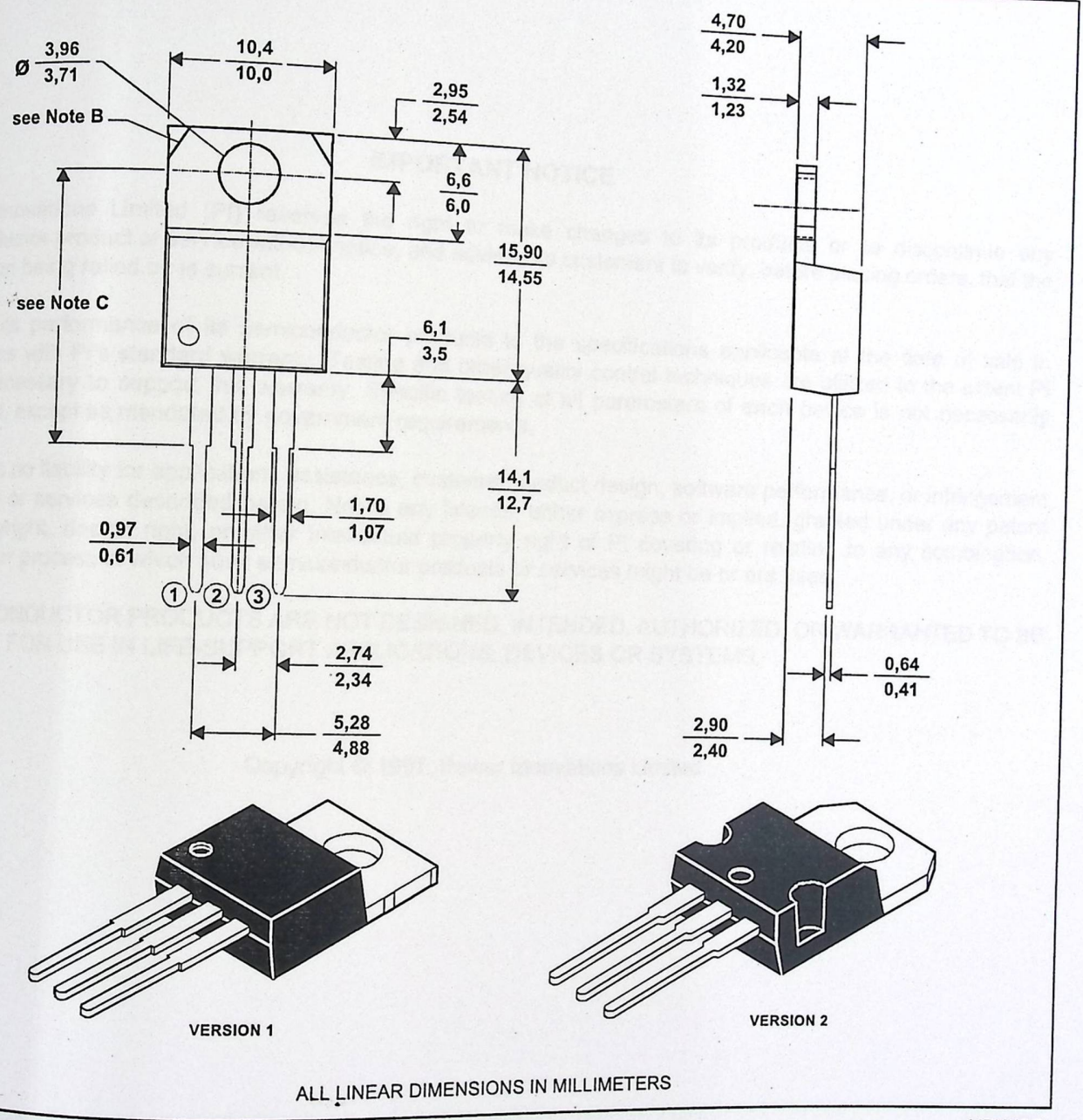
Figure 5.

MECHANICAL DATA

TO-220
3-pin plastic flange-mount package

This single-in-line package consists of a circuit mounted on a lead frame and encapsulated within a plastic compound. The compound will withstand soldering temperature with no deformation, and circuit performance characteristics will remain stable when operated in high humidity conditions. Leads require no additional cleaning or processing when used in soldered assembly.

TO220



ALL LINEAR DIMENSIONS IN MILLIMETERS

MDXXBE

NOTES: A. The centre pin is in electrical contact with the mounting tab.
B. Mounting tab corner profile according to package version.
C. Typical fixing hole centre stand off height according to package version.
Version 1, 18.0 mm. Version 2, 17.6 mm.

IMPORTANT NOTICE

Power Innovations Limited (PI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to verify, before placing orders, that the information being relied on is current.

PI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with PI's standard warranty. Testing and other quality control techniques are utilized to the extent PI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except as mandated by government requirements.

PI accepts no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor is any license, either express or implied, granted under any patent right, copyright, design right, or other intellectual property right of PI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

PI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS.