Palestine Polytechnic University

College of Engineering & Technology
Electrical and Computer Engineering Department

Graduation Project

Wood Carving System
Computer Driving Software

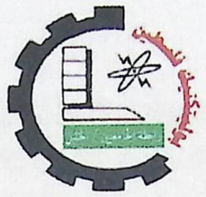
Project Team
Omar Al Kababji
Mohammad Abu Ajamieh

Project Supervisor
Eng. Ayman Wazwaz

Hebron – Palestine

May, 2007

Palestine Polytechnic University



College of Engineering & Technology
Electrical and Computer Department

Graduation Project

Wood Carving System
Computer Driving Software

Project Team
Omar Al Kababji
Mohammad Abu Ajamieh

Project Supervisor
Eng. Ayman Wazwaz

Hebron – Palestine

May, 2007

Wood Carving System
Computer Driving Software

Project Team
Omar Al Kababji
Mohammad Abu Ajamieh

Project Supervisor

Eng. Ayman Wazwaz

This Under-Graduate Project Report submitted to Computer and Electrical Engineering "Department in College of Engineering and Technology

Palestine Polytechnic University

For accomplishment the requirements of the bachelor degree in Engineering field at Computer System Engineering

Palestine Polytechnic University
Hebron – Palestine

May - 2007

جامعة بوليتكنك فلسطين
الخليل – فلسطين
كلية الهندسة والتكنولوجيا
دائرة الهندسة الكهربائية والحاسوب

اسم المشروع

Wood Carving System
Computer Driving Software

اسماء الطلبة

عمر "محمـد جميل" الكبيجـــي محمد عبد الرحمن أبو عجميه

بناء على نظام كلية الهندسة والتكنولوجيا واشراف ومتابعه المشرف المباشر على المشروع وموافقة اعضاء اللجنة الممتحنة تم تقديم هذا المشروع الى دائرة الهندسة الكهربائية والحاسوب وذلك للوفاء بمتطلبات درجة البكالوريوس في الهندسة تخصص هندسة أنظمة الحاسوب.

توقيع المشرف

.....

توقيع اللجنة الممتحنة

....

توقيع رئيس الدائرة

.....

DEDICATION

To our parents who made it all possible

To our brothers and sisters for their encouragement

To all the people who encouraged us
during the preparation of this project

To all the people who like to know
and look for the knowledge

For your mentorship, friendship, and tireless devotion
to insisting that we "get it right" and helping us to so

Omar Al Kababji
Mohammad Abu Ajamieh

ACKNOWLEDGMENT

We would like to thank Engineer Ayman Wazwaz who read our numerous revisions, helped us to make some sense of the confusion, and his valuable time.

We must not forget to thank all the instructors in the Electrical and Computer Engineering Department for their great impact in our education.

Special thanks to Dr. Nabeel Arman for all his care and valuable advices since the first day we joined Palestine Polytechnic University.

Special thanks to Dr. Alaa Al Halawani for his precious help in image processing, and Eng. Majdi Zaloum for his help in Matlab.

We acknowledge Palestine Polytechnic University for giving us the possibility to show some of what we have learned from it

Finally we can't forget to acknowledge our great parents who sacrificed themselves for educating us and facilitate our life, and for all their *coffee* and tolerance.

Omar Al Kababji
Mohammad Abu Ajamieh

ABSTRACT

Carving decorations manually on pieces of wood demands a lot of time, needs talented people to perform this operation, without neglecting the high probability of making mistakes while carving, which may completely ruin the piece of wood.

The carving operation would be more efficient by constructing a carving machine that moves in the three directions X, Y, Z and implementing a software driving program that creates and reads images (wood decorations), translates them, and then generates control signals that automates the carving operation by controlling the carving machine, and also monitoring the whole operation on the computer screen indicating the exact point in the carving operation and informing the user how much time is still needed.

By this approach the carving operation will be faster than before, more accurate, complex decorations will be feasible, identical copies are also possible, and wood carving talented people will be no more needed.

Computerized wood carving certainly removed many barriers faced when this process was done manually, and allowed many other new capabilities that were impossible before.

ان الحفر اليدوي على الخشب (الارابيسك) يتطلب الكثير من الوقت والجهد والمال، ويحتاج الى عمال مهره وموهوبون للقيام بهذه العمليه، بالاضافه الى امكانيه الخطأ في الحفر والذي قد يسبب تلف قطعه الخشب.

ان حوسبه اي نظام تسهل وتزيد فعاليه هذا النظام، وهذا هو الهدف الرئيسي من هذا المشروع، حيث سنعمل على تصميم ماكنه حفر في ثلاث اتجاهات المحور السيني (العرض) والمحور الصادي (الطول)، والمحور الاخير هو العمق، وسنعمل ايضا على تصميم برنامج حاسوب كامل لمسح وتصميم صور زخارف الاربيسك، حيث يقوم البرنامج بعدها بتحليل هذه الصور وترجمتها وتوليد الاشارات المناسبه للتحكم في محركات الابعاد الثلاث، هذا بالاضافه الى تصميم وحده مراقبه للنظام والتي ستتولى مسوؤليه متابعه عمليه الحفر خطوة بخطوه والتاكد من سير العمليه بنجاح وبدون اخطاء، مع اعلام المستخدم اين وصلت عمليه الحفر، وموقع راس الحفر.

ABSTRACT

Carving decorations manually on pieces of wood demands a lot of time, needs talented people to perform this operation, without neglecting the high probability of making mistakes while carving, which may completely ruin the piece of wood.

The carving operation would be more efficient by constructing a carving machine that moves in the three directions X, Y, Z and implementing a software driving program that creates and reads images (wood decorations), translates them, and then generates control signals that automates the carving operation by controlling the carving machine, and also monitoring the whole operation on the computer screen indicating the exact point in the carving operation and informing the user how much time is still needed.

By this approach the carving operation will be faster than before, more accurate, complex decorations will be feasible, identical copies are also possible, and wood carving talented people will be no more needed.

Computerized wood carving certainly removed many barriers faced when this was done manually, and allowed many other new capabilities that were not before.

يحتاج الى عمال
والجهد والمال،
الخشب.

من اهداف حوسبه هذا النظام هو تسريع عمليه الحفر وزياده الدقه وامكانيه حفر زخارف معقده، وعمل نسخ من نفس الشكل على اكثر من لوح خشب، والاستغناء عن العمال موهوبون للقيام او حتى الاشراف على هذه العمليه.

Table of Contents

First Pages

List of Figures

CHAPTER 6 : SUBSYSTEM TESTING AND SIMULATION

List of Tables

List of Tables

CHAPTER ONE

INTRODUCTION

1.1 Problem Overview

In the past wood cutting, modeling and decoration was done manually, this process had many drawbacks such as the process required time, loss of accuracy, and necessity of talent. Today many products are created by drawing the desired decoration by computer, adjusting it as needed, and then the carving operation is accomplished by computer. Computerizing this process; decreases carving time, increases job accuracy, and removes demand for a carving expert. Only a graphics talented user is needed.

This project targets the computerization of wood carving and decoration, by implementing a driving program that controls a carving machine, translating user drawn decorations to control signals that will automate the machine hardware components. In other words the user supplies the program with the desired image to be carved, – on the piece of wood – then the program will process the image, interpreting it to appropriate control signals; these signals serves as an automating language for the carving machine.

From the above explanation the project consists of two major subprojects, the construction of the carving machine, and the software application that analyzes images and controls the machine. A team of electrical and automation engineering students (Team A) will be working on the design and construction of the carving machine, while the computer engineering students (Team B) will be working on the design and implementation of the driving program which will process, understand, and translate computer drawings to control signals for the machine components. Of course interfacing circuits between the computer and the machine needs comprehensive understanding of the software so their design will lay upon Team B responsibilities. Table 1.1 clarifies each team majors and responsibilities.

Table 1.1: Project Teams

	Team A	Team B
Major	Electrical Engineering	Computer Systems Engineering
Responsibilities	Carving machine design and construction	Implementation of the machine driving application, interfacing circuits, and associated drawing program

Figure 1.1 shows a block diagram of the project; where the driving program which resides at the software application supplies the second block (Machine hardware components) with the needed signals through the interfacing circuits.

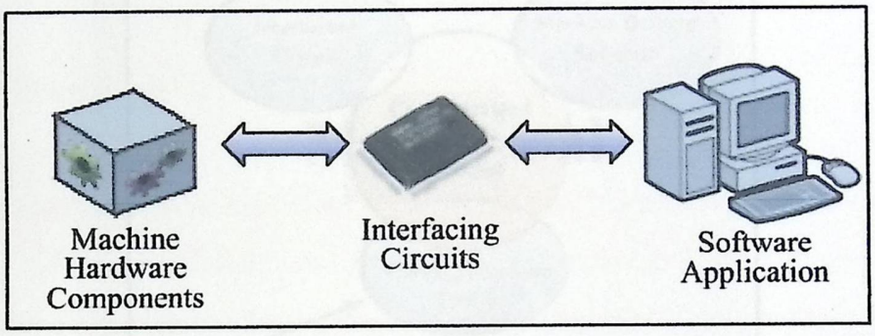


Figure 1.1: System Block Diagram

1.2 Aim of the project

The project whole idea concentrates on constructing a computerized wood carving machine where images are supplied to the computer, translated and then carved by the machine.

In more abstraction transferring an image from the computer screen into a carved shape on a piece of wood, to achieve this, a group of sub goals must be accomplished starting from the construction of the carving machine, translation of an image

residing on the computer to the interfacing circuit which in turn generates the motors controlling signals.

Since the two sub-projects complete each other and have many dependencies, to achieve the project main goal both teams should accomplish their tasks in time and with a high degree of professionalism. The figure below shows each team sub goals, along with the project general goal.

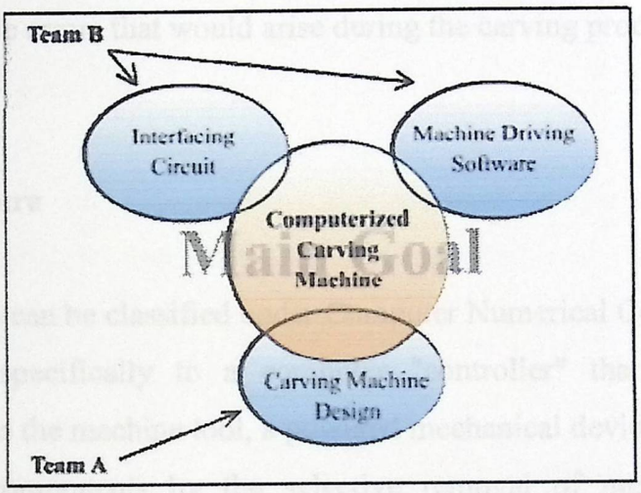


Figure 1.2: Project goals

The rest of this documentation will focus on Team B goals, taking in consideration that Team A tasks shall be completed.

Team B – computer systems engineering team – sub goals of the project will be:

- Software application implementation.
    - Make the software as flexible as possible.
    - Processing and understanding images; i.e. carve the piece of wood depending on certain information stored in the image.
    - Monitoring machine progress and reporting any encountered errors.

- Design of a drawing program tailored for the creation of decorations and images that will be carved on the wood.

- Interfacing circuits design and implementation.
  - Adjust the received signals from the PC and send them to the hardware components.
  - Send the machine state to the monitoring module, along with any possible errors that would arise during the carving process.

1.3 Review of literature

The whole system can be classified under Computer Numerical Control machines (CNC), and refers specifically to a computer "controller" that reads G-code instructions and drives the machine tool, a powered mechanical device typically used to fabricate metal components by the selective removal of metal. CNC does numerically directed interpolation of a cutting tool in the work envelope of a machine.

G-code is a common name for the programming language that drives NC and CNC machine tools. It was developed by EIA in the early 1960s, a final revision was approved in February 1980 as RS274D.

Due to the lack of further development, the sheer variety of machine tool configurations, and little demand for interoperability, few machine tool controllers (CNCs) adhere to this standard. Extensions and variations have been added to it independently by manufacturers, meaning that operators have to know the dialects and quirks of the particular machines they use, and CAM systems have had to limit themselves to the lowest common denominator of all the tools that they support.

A group of projects was developed in the same field of our project such as "Building a mill from a drill" created Jan 31st, 2003 by V.Chan, this project concerns on controlling a changing a drill machine to a milling machine using a group of stepper motors. A documentation of this project can be found by visiting the following link http://www.pathcom.com/~vhchan/cnc/cnc.html

1.4 Project estimated cost

This project consists of both hardware and software components in the first section we will describe the needed hardware components and their costs, while software components are considered in section 1.4.2.

1.4.1 Hardware equipments

Hardware equipments are both found in the carving machine, and the interfacing circuit, here we will consider only the cost of the needed equipments of the interfacing circuit, the cost of the carving machine equipments can be found in the documentation of the second team.

Taking a look at figure 3.5 in chapter 3 – system design – shows the needed hardware components, more details about these parts can be found in chapter 4 – detailed hardware design –.

Majority of the interfacing circuit components are low cost IC's, resistors, transistors, LED's, push buttons, an ON\OFF switch, wires, and a PIC microcontroller.

To program the PIC microcontroller a dedicated programmer must be used, but we will use the university programmers so there will be no need to buy a programmer.

1.4.2 Software Utilities

Different software utilities will be used in the design and implementation of this system, but the majority of them are open source programs meaning that they will be used for free with no need to pay for using them; the list of software we will use is as follows:

- Microsoft Windows XP Professional: selected for its ease of use, and its compatibility with the rest of the software utilities used.

- Netbeans IDE: best integrated development environment for implementing and designing Java applications in addition that it is open source software. It is used because the driving program will be implemented in Java.

- JDK 1.5: the java development kit which consist of the Java utility classes and the virtual machine. This software is also free.

- Open office: another open source software used to write and edit word documents, its functionality is the same of Microsoft office but its free.

- PsPice: an electrical program used to draw schematic diagrams of electrical circuits, only the free student version of it will be used.

1.4.3 Cost and Delivery Schedule

Table 1.2 shows the cost of each component needed in this project, and the delivery period for each component.

Table 1.2 Costs and Delivery Schedule

Resource Name	Type	Cost	Delivery
One Pentium4 PC	HW	350$	One Day
PIC 16C84	HW	6.95$	One Day
4 MHz crystal	HW	2.50$	One Day
PIC Programmer	HW	Use the university Programmer	Several Hours
Push Buttons	HW	6$	Several Hours
IC's, Transistors, Resistors, and Wires	HW	10$	Several Hours
10 LED's	HW	1$	Several Hours
Microsoft Windows XP professional	SW	included with PC	Several Hours
NetBeans IDE	SW	Internet. Connection	Several Hours
JDK 1.5	SW	Internet Connection	Several Hours
Open Office	SW	Internet Connection	Several Hours
PsPice student version	SW	Internet Connection	Several Hours

The internet connection is the total cost of using the internet to download the software. The totally estimated cost for this project is 376.45$. But assuming that the PC already exist as in our case then the total cost is only 26.45$ thanks for using open source software.

Again the cost of the equipments needed for constructing the carving machine are not included in this calculation, they will be considered in the cost estimation of the other team.

1.5 Assumptions and Dependencies

The following assumptions are considered in this project

- The physical machine will be constructed with the ability to accomplish its work precisely. This means that the machine must have an acceptable carving accuracy, smooth movements, and other properties needed to carve precisely on the wood.
- The wood pieces that will be carved have a flat surface.
- Wood pieces will have maximum dimensions of (250 * 200 * 30) cm corresponding to length, width, and height, respectively.
- Users of the machine have a good knowledge in computers and CAD.

Accomplishing our goals depends on the success of the other team (Team A) in building the machine with all of its required components working properly. For example we will be able to determine the required control signals after the other team supplies us with information about the type of motors used and how to control their motion. Dependencies can be summarized in the following points:

- Types of motors used to control the motion of the machine and their controlling signals.
- Accuracy of the machine constructed, will be useful to know how much accurately an image can be carved.
- How errors are reported from the machine, and their types.
- How much fast the machine can operate. This is important in maximizing its operation, which is an efficiency concern.
- The minimum size of the carving drill (its diameter) which will be used in the carving operation, and any others sizes used.

1.6 Development process

1.6.1 Software Engineering

This system will be developed using the evolutionary development process, which develops an initial implementation and exposing this to user comment and refining this through many versions until an adequate system has been developed, rather than having separate specification, development and validation activities, these are carried out concurrently with rapid feedback across these activities.

Evolutionary development model is choused rather than other software process models such as waterfall since it's more effective in producing systems that meet the immediate needs of customers and the specification can be developed incrementally. As users develop a better understanding of their problem, this can be reflected in the software system. Figure 1.3 shows the evolutionary approach.

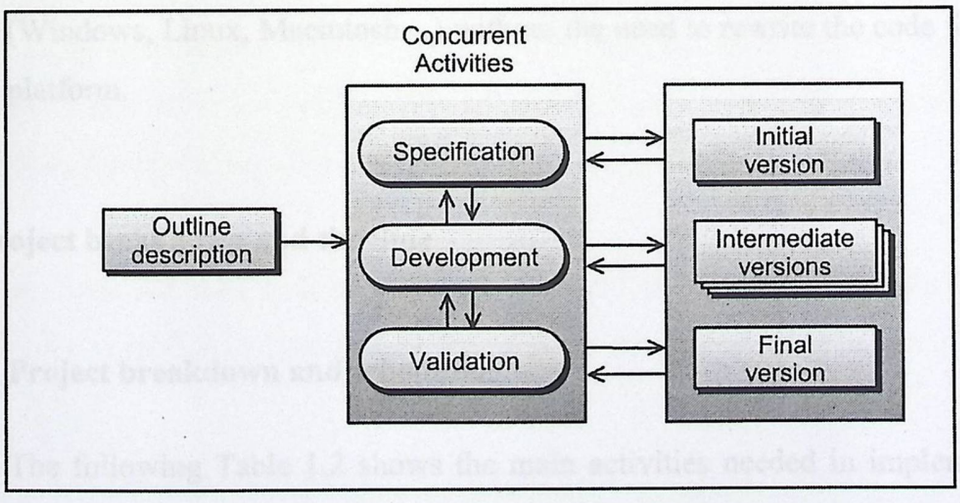


Figure 1.3: Evolutionary approach

1.6.2 Language of choice

The machine driving software will be implemented in Java programming language for the following reasons:

1. Java utility classes: by these classes we will reduce the implementation and development time. Using C++ for example instead of Java would waste a lot of time implementing these utilities that will effect the overall time of the project. Meanwhile not forgetting the excellent performance reached in Java applications with its last virtual machine release JDK 5.
2. Our good experience in the Java language.
3. Stability of Java programs, and its good exception handling mechanism which will be useful in preventing unexpected program behavior.
4. Java graphics support: the Java Graphics2D API has many useful libraries for image processing and manipulating, with very sleek look and feel interface which will be helpful and easy for the user to deal with.
5. Write once run anywhere: by Java you can run your program on any platform (Windows, Linux, Macintosh…) without the need to rewrite the code for each platform.

1.7 Project breakdown and timeline

1.7.1 Project breakdown and scheduling

The following Table 1.2 shows the main activities needed in implementing the project the time (in weeks) needed to finish each task, the implementer, and each activity dependencies. Note that the project will require 13 weeks, approximately 91 days.

Table 1.3: Project tasks

	Task	Duration	Dependencies	Implementer
T1	Requirements gathering and project understanding	1	M1	Team
	PART 1: Background study			
T2	Study of Java 2D API's	2		Mohd
T3	Study of PIC (Peripheral Interface Controller)	2		Omar
T4	PC Interfacing	1		Omar
T5	Motor Controlling Methodologies	1	M2	Mohd
	PART 2: System Development			
T6	Designing Interfacing Circuits and PC connection	½	T4, T5	Omar
T7	Programming PIC microcontroller needed programs	½	T3	Mohd
T8	Algorithms Design	2	T1, T2	Team
T9	Defining software roadmap	2	T8	Team
	PART 3: Documenting			
T10	Writing the documentation	-	M4	Team
T11	Seminar Preparation	1	M5	Team
		Total = 13		

Requirements gathering and project understanding: here we contact the project supplier (Eng. Mohammad Shobaki), and collect the functional and non-functional requirements needed in the system then will analyze and classify them in categories, other requirements will be collected by brainstorming process.

Study of Java 2D API's: this phase is important since the core of the project depends mainly on image processing which needs a good knowledge in graphics.

Study of PIC (Peripheral Interface Controller): studying different PIC micro-controllers' features and choosing the one that best fits the system needs.

PC Interfacing: studying the interfacing ports in the PC mainly the parallel and serial ports and how to read and write data bits to these ports, needed for connecting software with the hardware.

Motor Controlling Methodologies: how to control the motors motion and speed, and how to determine their position, we will focus mainly on stepper and servo motors.

Designing Interfacing Circuits and PC connection: after finishing PART1, we will design and draw the circuits needed to interface the PC with the hardware. Some circuit drawing programs will be used as a simulation.

Programming PIC microcontroller needed programs: the programs needed to translate data (coordinates) from the PC and generation of motor controlling signals will be written, loaded to the microcontroller, and tested.

Algorithms Design: designing the algorithms needed for translating and understanding the images drawn by the user and converting it to an intermediate language understood by the microcontroller.

Defining software roadmap: a high level view of the software, its interfaces, and majority of the needed classes since we will use object oriented approach in the development. UML and data models (i.e. dataflow) will be designed in this phase.

Machine process simulation: a 3d simulation of the machine will be made, showing the main hardware parts, with the interfacing circuit, this simulation will be designed using 3d Studio Max.

Writing the documentation: documenting the final report will be done from the beginning of the project in parallel with the other tasks.

Seminar Preparation: a summarizing presentation will be made in the final week to be shown in the seminar discussion.

Milestones

M1: delivery of all the project requirements.

M2: acquisition of needed knowledge to accomplish the project.

M3: completion of the theoretical part of the system.

M4: documentation of the full project will be ready.

M5: Seminar presentation completeness.

1.7.2 Project Timeline

The following Figure 1.4 shows the timeline of the project in days, which extends from Saturday, 7-October-2006 to Thursday, 28-December-2006

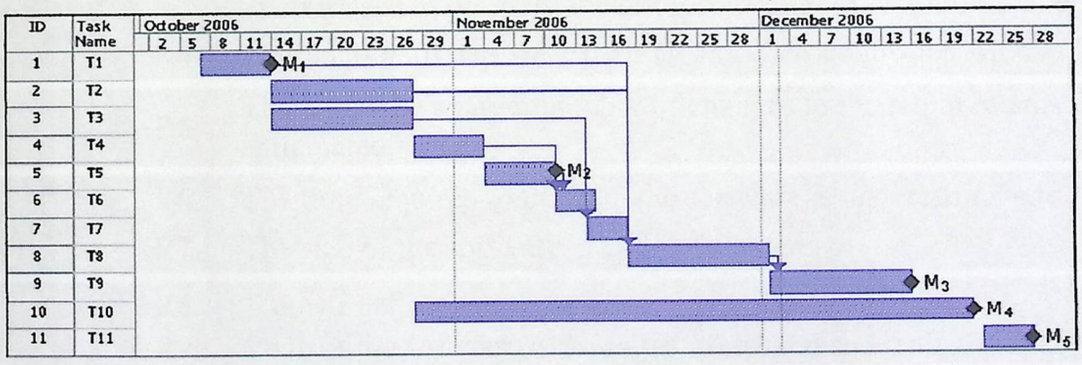


Figure 1.4: Project timeline

1.8 Risk identification and planning

Risks always exist in any project, anticipating risks that might affect the project schedule or the quality of the software being developed and to take action to avoid these risks is very important for the whole success of the project.

Table 1.3 lists the types of risk which may impact the project development process, showing the risk, its type – Technology, people, organizational, tools, requirements, and estimation – and a short description about it.

Table 1.4: risks and risk types

Risk type	Possible Risks	
Technology	R1	Hardware which is essential for the project may not be delivered on schedule.
	R2	The machine used to carve on wood may no be constructed by the other team.
	R3	The kit used to program the microcontroller may not be available.
	R4	Malfunction of hardware parts (IC's, Microcontroller, Motors)
Requirement	R5	There may be a larger number of changes to the requirements than anticipated.
Estimation	R6	The time required to develop the software is underestimated
	R7	The size of the software is underestimated.
Organizational Tools	R8	Time of delivery of the project changed.
	R9	Code of the software is damaged or deleted suddenly.
People	R10	Illness of one or more

Table 1.4 shows the probability of each risk and a seriousness of it, while table 1.5 considers each of the key risks which have been identified and identifies strategies to manage the risk.

Table 1.5: risk analysis

Risk ID	Risk	Probability	Effects
R1	Hardware which is essential for the project will not be delivered on schedule.	moderate	catastrophic
R2	The machine used to carve on wood will no be constructed by the other team.	low	catastrophic
R4	Malfunction of hardware parts (IC's, Microcontroller, Motors)	moderate	serious
R5	There will be a larger number of changes to the requirements than anticipated.	low	serious
R9	Code of the software is damaged or deleted suddenly.	Very low	catastrophic
R3	The kit used to program the microcontroller will not be available.	low	serious
R7	The size of the software is underestimated.	Low	tolerable
R6	The time required to develop the software is underestimated	Low	tolerable
R8	Time of delivery of the project changed.	Very low	Tolerable
R10	Key staffs are ill and unavailable at critical times.	Low	Insignificant

Table 1.6: Risk Management Strategies

Risk ID	Strategy
R1	Identify the needed hardware as fast as possible and order them.
R2	Continue the project development and simulate the machine operation.
R3	Chose a microcontroller that can be programmed using the programming kits available in the university labs.
R4	Order spare hardware parts.
R5	Derive traceability information to assess requirements change impact, and maximize information hiding in the design.
R6	Understand exactly what is needed in the software.
R7	Understand exactly what is needed in the software.
R8	No strategy
R9	Always save back up copies of the software (on flash, CD's, hard disks)
R10	Reorganize team so that there is more overlap in the work

1.9 Report overview

The following chapters will be included in this report:

***Chapter One*:** an introduction to the project, describing the problem, project idea, and its aims and the software engineering methodology followed.

Chapter Two: a theoretical background study covering the Java2D API, motors, PIC microcontroller, and the PC parallel port.

Chapter Three: system design, dissecting the complete system into a group of subsystems, describing the functionality of these subsystem, interfaces between them, and finally giving a complete overview of how the system works starting from supplying the program with an image and getting the image carved on the piece of wood.

Chapter Four: detailed hardware design, describing in more detail the hardware components of the carving machine focusing on the interfacing circuit components, showing their functions, the components type numbers, and schematic diagrams of each component.

Chapter Five: detailed software design and implementation, where the software is explained as a group of modules, using code snaps, algorithms, flowcharts, dataflow, and UML.

Chapter Six: system testing, shows the methodologies used to test the system different modules, contains images showing results of testing operations.

Chapter Seven: Conclusion and Future Work, conclusions obtained from the system design, and suggestions and improvements to the system which could be used in future work.

CHAPTER TWO

THEORETICAL BACKGROUND

2.1 Java Graphics 2d API

2.2 Motors

2.3 PIC microcontroller

2.4 PC Parallel port

2.5 Requirements

2.1 Java Graphics 2d API

Java is one of the most modern programming languages with lot of precompiled classes, API's, and reusable components. Since image understanding and interpretation covers a major need in the project software, then this application will highly depend on image processing, which is one of the main API's used in Java 2D.

But why not using MATLAB for image processing? Even if MATLAB have a good and powerful image tool box which could help in image processing, the software needs an appealing graphical user interface and an associated drawing program to be used for the creation of decorations and images. So Java is more adequate than MATLAB.

Java 2D API enhances the graphics, text, and imaging capabilities for the Java application development, these API's contain many graphics related libraries, with some advanced graphics libraries such as images special effects and transformations, as well as the creation of read and write filters for both image and graphic files.

The Java 2D API provides a uniform rendering model across different types of devices. At the application level, the rendering process is the same whether the target rendering device is a screen or a printer. When a component needs to be displayed, its "`paint`" or "`update`" method is automatically invoked with an appropriate "`Graphics`" context.

2.1.1 Coordinate Systems

Two coordinate systems are used in Java 2D API:

- User space: which is device-independent: it's a logical coordinate system, this system is used exclusively inside the Java program, and all geometries are specified in this space.
- Device space: is a device-dependent, which means it varies from device to another, according to the device itself, i.e. one computer monitor can contain multi configurations like 640*480*16 colors, 640*480*256 color, and 800*600*256 colors, or a printer.

User space coordinates are automatically transferred into appropriate device space when a graphic object is rendered, which is Java2D API responsibility to do the needed transformation with no user intervention.

Device space will be very useful in this system as we can represent the carving machine as a "GraphicsDevice" suited for the piece of wood dimensions, so we can take advantage of the Java2D capabilities for transforming between the two spaces.

2.1.2 Images

Images are collections of pixels organized spatially, each pixel defines the properties of the appearance of the image at one single point, and images are represented by a two-dimensional array of pixels called raster.

Images are divided into two main parts depending on the type of pixel definition used:

1- Directed colors: used in images that contain more than 256 colors usually, pixels in directed mode define the color, alpha, transparency, and other display characteristics, storing these images for sure consumes more memory space than other image types.

2- Indexed colors: usually used in images that contain 16 or less colors, in this mode all the colors are indexed in a colors array, where each pixel is given an index number, instead of representing each pixel as an object with different attributes as in Directed color mode – which requires more storage space –, here each pixel is represented by one byte only, Figure 2.1 shows an array of colors used in indexed colors mode.

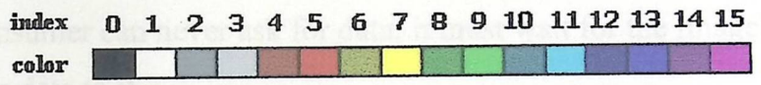


Figure 2.1: Array of 16 colors, used in Indexed colors images

Indexed colors mode, advantages and disadvantages:

- **Advantages**
  1- Smaller to store in memory.
  2- Processing the colored image is done easily with simple algorithms and requires less time.
  3- Flexibility in defining the colors array, where the user can change the colors in the array easily.
- **Disadvantages**
  1- Limited number of colors: not a drawback in our software application since there in no need for using a large number of colors, only certain colors to distinguish between different carving depth levels.
  2- Inability to define other display characteristics: also not a drawback since the images will be aliased; where we only need the color of the pixel and its position (X, Y) in the image.

For the previous advantages, indexed colors mode will be used for dealing with images in the software application.

2.1.3 Imaging Models

The Java 2D API supports three imaging models:

1- Producer/consumer: also called (push) model, is a simple filter model of image producers and consumers for image processing. An Image object is an abstraction that is not manipulated directly. Called (push) model because an ImageConsumer can never ask for data; it must wait for the ImageProducer to "push" the data to it.

2- Pipeline model (pull): also called Java Advanced Imaging model (JAI), this model is used to add more powerful and advanced imaging features, image sources such as file or network participate in the "pull" imaging model by responding to requests for arbitrary areas.

3- Immediate model: this model provides techniques for dealing with pixel mapped images whose data resides in memory. This model supports accessing image data in a variety of storage formats and manipulating image data through several types of filtering operations.

Translating the images to control signals for the motors will need lot of processing, for colors extraction, pixel manipulation, area specification and many other processes. The most appropriate model is immediate model as it deals with images stored in memory fast and efficiently, which will significantly decrease processing time, "BufferedImage" is one of the main classes used in the immediate model, which makes an instance of the image in the memory, with its other related information.

2.2 Motors

The hardware part of the machine will contain mainly a group of motors with different functions, these motors will be used to enable accurate positioning of the carving head in the three space directions (X, Y, Z), and drill rotational motion. So we need three motors for head accurate positioning (Servo motors), and one for rotational motion (DC motor). In this section we discuss the motors interface and control operation with no interest about their specific information (i.e. why servos not steppers, are they suitable for carving wood or not …etc) since chousing the motor types was the responsibility of Team A and of course they have their justifications, at the end of the road for us there is a servo motor that needs to be controlled which is Team B responsibility.

Sub-section 2.1.1 covers DC motors while sub-section 2.1.2 covers the needed information about controlling servo motors.

2.2.1 DC Motors

A simple electric DC motor has six parts, as shown in Figure 2.2 these parts are as follows:

- Armature or rotor
- Commutator
- Brushes
- Axle
- Field magnet
- DC power supply of some sort

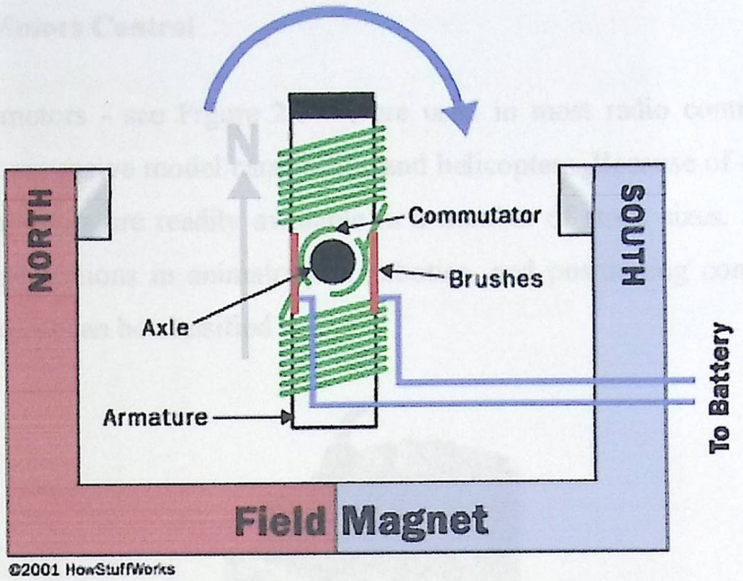


Figure 2.2: Parts of an electrical DC motor

An electric motor is all about magnets and magnetism: A motor uses magnets to create motion. If you have ever played with magnets you know about the fundamental law of all magnets: Opposites attract and likes repel. So if you have two bar magnets with their ends marked "north" and "south," then the north end of one magnet will attract the south end of the other. On the other hand, the north end of one magnet will repel the north end of the other (and similarly, south will repel south). Inside an electric motor, these attracting and repelling forces create rotational motion.

In the above diagram, you can see two magnets in the motor: The armature (or rotor) is an electromagnet, while the field magnet is a permanent magnet (the field magnet could be an electromagnet as well, but in most small motors it isn't in order to save power).

2.2.2 Servo Motors Control

Servomotors - see Figure 2.3 - , are used in most radio controlled model airplanes and expensive model cars, boats, and helicopters. Because of this hobbyist market, servomotors are readily available in a number of stock sizes. Servomotors have many applications in animatronics, robotics, and positioning control systems where this project can be classified under it.

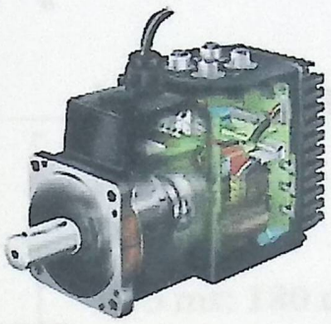


Figure 2.3 A Servo motor

Fundamentally, servomotors are geared dc motors with a positional feedback control that allows the rotor to be positioned accurately. The specifications state that the shaft can be positioned through a minimum of 90° (± 45). In reality, we can extend this range closer to 180° (±90°) by adjusting the positional control signal. For more information see appendix A at the end of the documentation.

There are three wire leads to a servomotor. Two leads are for power, Vcc and GND. The third lead feeds a position control signal to the motor. The position control signal is a single variable-width pulse. The pulse can be varied from 1 to 2 ms. The width of the pulse controls the position of the servomotor shaft.

A 1-ms pulse rotates the shaft to the extreme counterclockwise position (-45°). A 1.5-ms pulse places the shaft in a neutral midpoint position (0°). A 2-ms pulse rotates

the shaft to the extreme clockwise position (+45°). The pulse width is sent to the servomotor approximately 50 times a second (50Hz). Figure 2.4 illustrates the relationship of pulse width to servomotor position.

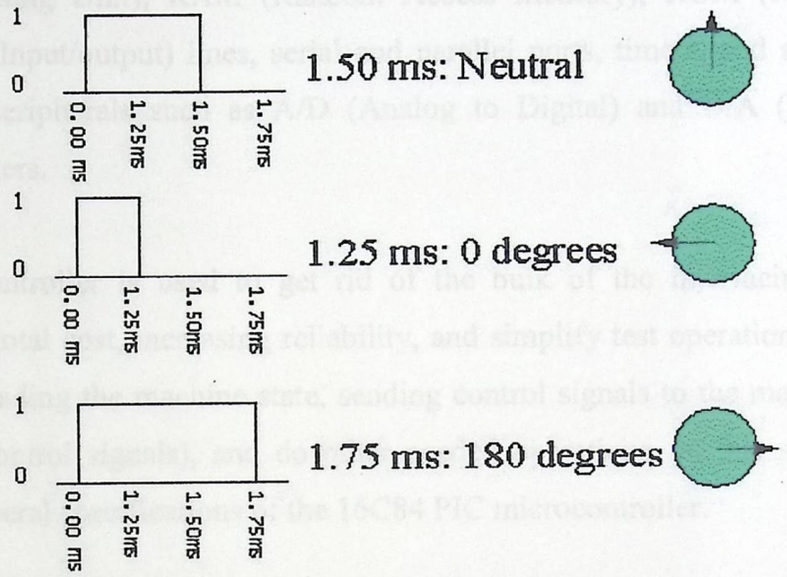


Figure 2.4 Servomotor positions with respect to pulse width

In the system design three servo motors will be used, one for each axis. Control signals for each motor will be generated from a PIC microcontroller or directly from the PC Parallel Port if it will be reliable to generate accurate signals; which will remove the complexity of the interfacing circuit. Each motor will have a dedicated control line.

2.3 Microcontroller

A microcontroller is an inexpensive single-chip computer. Single-chip computer means that the entire computer system lies within the confines of the

integrated circuit chip. The microcontroller on the encapsulated sliver of silicon has features similar to those of our standard personal computer. The microcontroller is capable of storing and running a program. The microcontroller contains a CPU (Central Processing Unit), RAM (Random Access Memory), ROM (Read Only Memory), I/O (Input/output) lines, serial and parallel ports, timers, and sometimes other built-in peripherals such as A/D (Analog to Digital) and D/A (Digital to Analog) converters.

A PIC microcontroller is used to get rid of the bulk of the interfacing circuit, minimizing its total cost, increasing reliability, and simplify test operation. Its main functions are reading the machine state, sending control signals to the machine (i.e. servo motors control signals), and do other needed operations. In this section we describe the general specifications of the 16C84 PIC microcontroller.

The PIC16F94 belongs to the mid-range family of the PICmicro© microcontroller devices (see Figure 2.5). The program memory contains 1K words, which translates to 1024 instructions. Since each 14-bit program memory word is the same width as each device instruction. The data memory (RAM) contains 68 bytes. Data EEPROM is 64 bytes too.

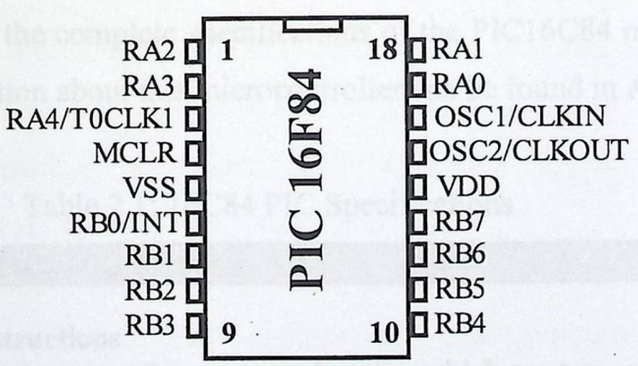


Figure 2.5: 16C84 PIC microcontroller pinout diagram

There are 13 I/O pins divided into two ports, port A with 5 I/O lines and port B with 8 I/O lines. Each port has registers associated with it, the TRIS (Tri State) register

integrated circuit chip. The microcontroller on the encapsulated sliver of silicon has features similar to those of our standard personal computer. The microcontroller is capable of storing and running a program. The microcontroller contains a CPU (Central Processing Unit), RAM (Random Access Memory), ROM (Read Only Memory), I/O (Input/output) lines, serial and parallel ports, timers, and sometimes other built-in peripherals such as A/D (Analog to Digital) and D/A (Digital to Analog) converters.

A PIC microcontroller is used to get rid of the bulk of the interfacing circuit, minimizing its total cost, increasing reliability, and simplify test operation. Its main functions are reading the machine state, sending control signals to the machine (these are the program memory and servo motors control signals), and do other needed processing so that access to each block can occur describe the general specifications of the 16C84 PIC microcontroller. The data memory can further be broken down into

The PIC16F94 belongs to the mid-The data memory also contains the data 64 byte devices (see Figure 2.5). The program memory is not directly mapped into the data memory, but is to 1024 instructions. Since each instruction is 14 bits wide, an indirect address pointer specifies the address of the each device instruction to read/write.
is 64 bytes too.

Table 2.1 summarizes the complete specifications of the PIC16C84 microcontroller. More detailed information about this microcontroller can be found in Appendix A.

Table 2.1: 16C84 PIC Specifications

High performance RISC CPU features
Only 35 single word instructions All instruction single-cycle except for program braches which are two-cycle Operating speed: DC - 20 MHz clock input DC – 200 ns instruction cycle 1024 words of program memory 368 bytes of Data RAM 64 bytes of Data EEPROM 14-bit wide instruction words

and the port register address itself. The TRIS register controls whether a particular pin on a port is configured as an input line or an output line. Once the ports are configured the user may then read or write information to the port using the port register address.

Some of the I/O pins are multiplexed with other functions, these functions include:

- External interrupt.
- Change on Port B interrupt
- Timer 0 clock input

There are two memory blocks in the PIC16C84. These are the program memory and the data memory. Each block has its own bus, so that access to each block can occur during the same oscillator cycle. The data memory can further be broken down into the general purpose RAM and the Special Function Registers (SFRs), which are used to control the peripheral modules. The data memory also contains the data 64 byte EEPROM memory. This memory is not directly mapped into the data memory, but is indirectly mapped. That is, an indirect address pointer specifies the address of the data EEPROM memory to read/write.

Table 2.1 summarizes the complete specifications of the PIC16C84 microcontroller. More detailed information about this microcontroller can be found in Appendix A.

Table 2.1: 16C84 PIC Specifications

High performance RISC CPU features
Only 35 single word instructions **All instruction single-cycle except for program braches which are two-cycle** **Operating speed: DC - 20 MHz clock input** **DC – 200 ns instruction cycle** **1024 words of program memory** **68 bytes of Data RAM** **68 bytes of Data EEPROM** **14-bit wide instruction words**

8-bit wide data bytes **15 special function hardware registers** **Eight=level deep hardware stack** **Direct, indirect and relative addressing modes** **Four interrupt sources:** - **External RB0/INT pin** - **TMR0 timer overflow** - **Port B <7:4> interrupt on change** - **Data EEPROM write complete**
Peripheral Features
13 I/O pins with individual direction control **High current sink/source for direct LED drive** - **25 mA sink max. per pin** - **25 mA source max. per pin** **TMR0: 8-bit timer/counter with 8-bit programmable prescaler**
Special Microcontroller Features
10,000 erase/write cycles Enhanced FLASH program memory typical **10,000,000 typical erase/write cycles EEPROM Data memory typical** **EEPROM Data Retention> 40 years** **In-Circuit serial programming™ (ICSP™) – via two pins** **Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)** **Watchdog Timer (WDT) with its own On-Chip RC Oscillator for reliable operation** **Code protection** **Power saving SLEEP mode** **Selectable oscillator options**

The MPASM cross assembler will be used to write and develop PIC programs, it will take our source code file which would have been created using any text editor and assemble it into the machine code which will be executed by the PIC. The file produced is in a format suitable for being sent to the PIC.

The final stage is to download the machine code file generated by the cross assembler to the PIC16C84, a special programmer software will be used to send – write– the machine code (.hex file) to the PIC 16C84 microcontroller embedded in the development kit.

2.4 PC Parallel Port

The Parallel Port known as the printer port is the most commonly used port for interfacing computers with home made projects. This port allows the input of up to 9 bits or the output of 12 bits at any given time, thus requiring minimal external circuitry to implement many simpler tasks.

The parallel port, as implemented on the PC, consists of a connector with 17 signal lines and 8 ground lines. The signal lines are divided into three groups: 4 Control lines,
5 Status lines, and 8 Data lines.
As originally designed, the Control lines are used as interface control and handshaking signals from the PC to the printer. The Status lines are used for handshake signals and as status indicators for such things as paper empty, busy indication and interface or peripheral errors. The data lines are used to provide data from the PC to the printer, in that direction only. Later implementations of the parallel port allowed for data to be driven from the peripheral to the PC. Figure 2.6 shows the parallel port pinout.

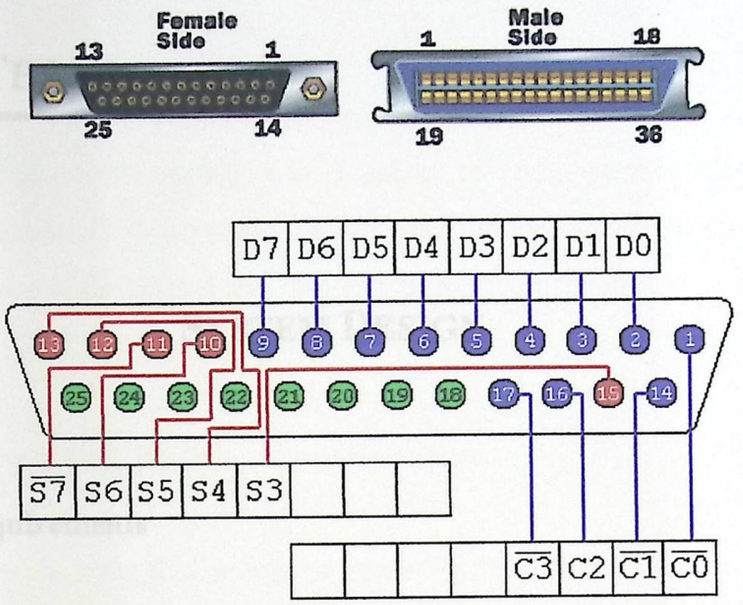


Figure 2.6: Parallel Port Pins

Accessing the parallel port differs between programming languages for example in Ansi C, there is some built in functions which are used for reading from and writing to the port. In Java there are not such as functions so we will need to use JNI (Java Native Interface) and use native methods to communicate with the port. The native methods will load a dynamic link library file named *jnpout32reg.dll*, which have function calls to access the parallel port.

The software will use the following native methods to read and write data to the parallel port.

public native void Out32 (short portAddress, short Data)

public native short Int32 (short portAddress)

CHAPTER THREE

SYSTEM DESIGN

3.1 System Requirements

3.2 Objectives

3.3 System Parts

3.4 System modeling

3.5 Interfacing circuit

3.6 How the system works

3.1 System Requirements

As any project this project has a group of requirements that should be considered and satisfied during the design and implementation process, these requirements consist of functional and non-functional requirements.

3.1.1 Functional and Non-Functional requirements

The functional requirements that should be considered are as follows:

- The machine shall be able to carve on a piece of wood.
- The carving head shall have the ability to be positioned precisely.
- The machine shall report any encountered errors during the carving operation.
- The machine shall carve in the three directions X, Y, Z.
- The carving head can be changed to other diameter sizes.
- The driving program shall understand images drawn with some format and convert them to control signals to the machine hardware components.
- The software shall allow the user to draw and edit images and decorations.
- The driving software shall show progress of the carving process.
- If the carving process stops suddenly for any reason, the machine shall continue the carving process from the point it stopped.
- The software shall allow the user to define the dimensions of the piece of wood.

The non-functional requirements are:

- The carving must be of a high quality and precision.
- The machine should terminate the carving operation as fast as possible.
- The use of the software driving program and drawing program must be as easy as possible.
- The software must be reliable and robust as much as possible.

After defining the functional and non-functional requirements user and system requirements will be considered.

3.1.2 User and System requirements

The user requirements are specified on the demand of the machine owner, and can be summarized in the following:

- Carve a predrawn image on wood
- Give the user the ability to define the piece of wood dimensions in different metrics (cm, inch ...) and margins.
- Scaling the image to fit on the wood size
- Carve the wood at different depth levels, these depths should be specified on the original image using different colors.
- Move the carving head manually, using keyboard arrows or mouse.
- Monitor the carving process and indicate the progress on the image.
- Save the progress, this is needed to enable the machine to continue its process in the case of a fault preventing the machine to continue carving is encountered (i.e. power turns off).
- Possibility to change the drill heads with different diameters.
- Control the speed of motors from the software application depending on the carving area if possible.
- Associate the driving program with a drawing program tailored for drawing decorations and manage them.
- Control the machine manually with no need of a computer.

System requirements:

- Process and analyze input image then translate them to appropriate control

After defining the functional and non-functional requirements user and system requirements will be considered.

3.1.2 User and System requirements

The user requirements are specified on the demand of the machine owner, and can be summarized in the following:

- Carve a predrawn image on wood
- Give the user the ability to define the piece of wood dimensions in different metrics (cm, inch ...) and margins.
- Scaling the image to fit on the wood size
- Carve the wood at different depth levels, these depths should be specified on the original image using different colors.
- Move the carving head manually, using keyboard arrows or mouse.
- Monitor the carving process and indicate the progress on the image.
- Save the progress, this is needed to enable the machine to continue its process in the case of a fault preventing the machine to continue carving is encountered (i.e. power turns off).
- Possibility to change the drill heads with different diameters.
- Control the speed of motors from the software application depending on the carving area if possible.
- Associate the driving program with a drawing program tailored for drawing decorations and manage them.
- Control the machine manually with no need of a computer.

System requirements:

- Process and analyze input image then translate them to appropriate control

signals.

- Define device coordinates, user coordinates and transform and scale images between these two modes.
- Define multiple colors in the image where each color specifies the depth that the Z-motor will move, i.e. black color means a 5 mm depth level, and green color means 7 mm.
- Ability to control the motors manually by mouse or keyboard by defining appropriate signals related to each key, or mouse direction of move.
- Read feedback signals from sensors connected to the machine to report encountered errors during the carving operation.
- Ability to pause the carving process or even stop it and postpone the work for later, in case of emergencies or any other user requirements.
- Generate signals with different frequencies, so the motors speed is alternated depending on these signals.

Fault tolerance by providing sensors on edges of the wood to stop the motors if they overpass the allowed area.

3.2 Objectives

The objectives of this project can be divided into two main parts, the machine driving software objectives and the carving machine objectives.

The machine driving software objectives are:

- Ability to process a predrawn image, analyze it, and generate suitable signals for the machine hardware to carve it.
- Provide a dedicated drawing program to the user, to create drawings and decorations.
- Allow the user to see and modify the texture (on screen), before carving it on wood.

- Speed up the carving operation, and carving quality through multiple carving passes.
- Supply the user with a monitoring system, used to show the progress of the carving operation.
- Report errors encountered during the carving process.

The machine objectives are:

- Carve the piece of wood based on the signals sent from the software application.
- Increase carving quality and accuracy by moving the carving head smoothly and precisely.
- Send feedback signals from sensors to the software application, about the state of the machine and encountered errors.

3.3 System Parts

To satisfy user and system requirements – listed in section 3.1 – the machine should consist of the following hardware and software parts.

The hardware parts:

- X direction servo motor, used to move the carving head in the X direction.
- Y direction servo motor, used to move the carving head in the Y direction.
- Z direction servo motor, used to move the carving head in the Z direction.
- Drill, a drill dedicated for carving on wood.
- Drill Motor, used to rotate the drill in order to carve on the piece of wood.
- Wood holder: a place to mount and fix the piece of wood on it
- Edge detection sensors used to signal a 5v signal when the carving head exceeds the allowable displacement in both X and Y directions.

- Wood detection sensor which is used to signal a 5v signal when the piece of wood is mounted on the machine and 0v when there is no wood on the machine.
- Manual control mechanism to control the machine manually.
- An interfacing circuit, described later in this chapter.

The software parts:

- Image loader and analyzer, used to open a user defined image, scale it to fit the device coordinates, and then analyze it.
- Control Signals generator, generates signals to move the motors in different directions.
- Error detector, a dedicated module for error detection and sensing.
- Monitoring system, used to display progress of carving operation, different machine parts and their states, and report detected errors.
- Drawing program, used to enable the user to draw images, decorations and edit existing ones. Also it should be responsible to manage and store images.

3.4 System Modeling

The system can be modeled as a set of subsystems and relationships between them, this decomposition is based on the functionality of the subsystems with no distinction between hardware and software components (i.e. a subsystem may consist of software and hardware components). In section 3.4 hardware components will be grouped in one subsystem called the interfacing circuit.

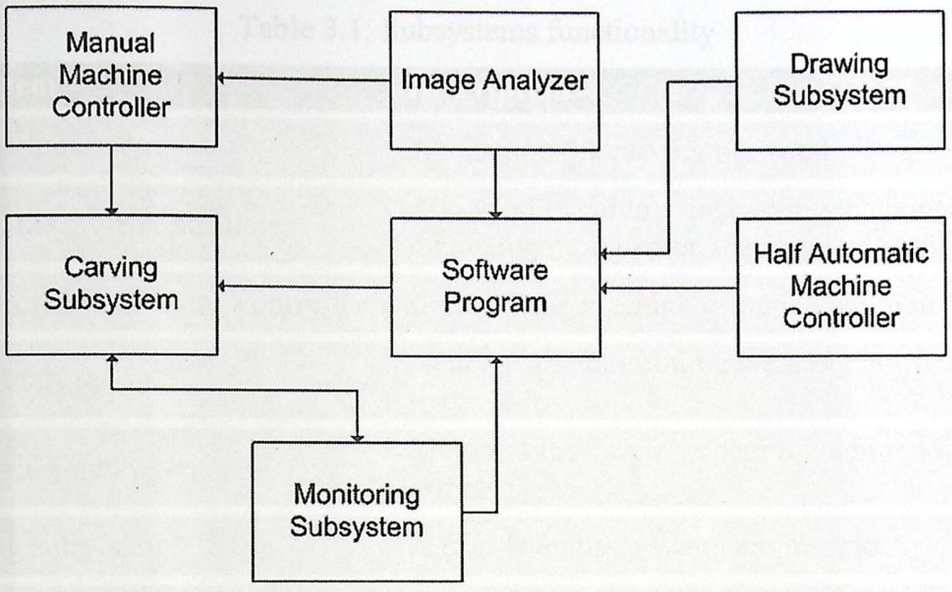


Figure 3.1: Subsystems of the carving machine

The system is decomposed of a group of subsystems in order to satisfy user and system requirements mentioned in section 2.5, the subsystems can be either hardware or software or a combination of them, and are specified as follows:

- Carving subsystem.
- Manual Machine Controller.
- Half Automatic Machine Controller.
- Image Analyzer.
- Monitoring subsystem.
- Drawing subsystem.
- Software program.

Table 3.1 shows a brief description of each subsystem

Table 3.1: Subsystems functionality

Subsystem	Description
Carving subsystem	A machine for carving on wood
Manual machine controllers	A keypad to control the carving machine manually, mounted on the interfacing circuit.
Half automatic machine controller	Controls the machine using the computer
Image analyzer	Analyze an image and generates control signals to carve it
Monitoring subsystem	Monitors the carving operation and reports errors
Drawing subsystem	A user friendly drawing environment
Software program	Contains all subsystems software in one program

The software program is not a stand alone subsystem; it serves as a software container for all subsystems. As it can be noticed it has relations with the Image analyzer, drawing subsystem, half automatic machine controller, and the monitoring subsystem. It will contain all the software of these subsystems.

The following subsections will explain these subsystems in more details concerning on the hardware components and circuit block diagrams, detailed information about the hardware circuits and schematics are provided in Chapter 4 (Detailed Hardware Design), while detailed information about the software is presented in Chapter 5 (Implementation).

3.4.1 Carving subsystem

The carving subsystem consists of a group of motors that make the machine capable of carving on different pieces of wood, three motors are used to place the

carving head in a precise location in the three directions X, Y, Z, and another motor is used to rotate the carving head.

The positioning motors are servo motors controlled by a position control signal. The position control signal is a square wave signal; the motor will rotate in specific degrees by depending on the number of signals it receives. The fourth motor which is a DC type is used to rotate a drill mounted on it, its control operation is only on/off.

The complete construction of this sub system is under the responsibility of the other team – Team A –.

3.4.2 Manual Machine Control

This subsystem allows the user to control the machine manually – with no need for a computer –; it contains a keypad with different function buttons where each button generates one of the various machine orders.

The user will be able to generate the following commands using this subsystem:

1. Move the carving head in the X direction.
2. Move the carving head in the Y direction.
3. Move the carving head in the Z direction, downward and upward.
4. Spin the carving head.

The figure below shows a block diagram of the manual machine control.

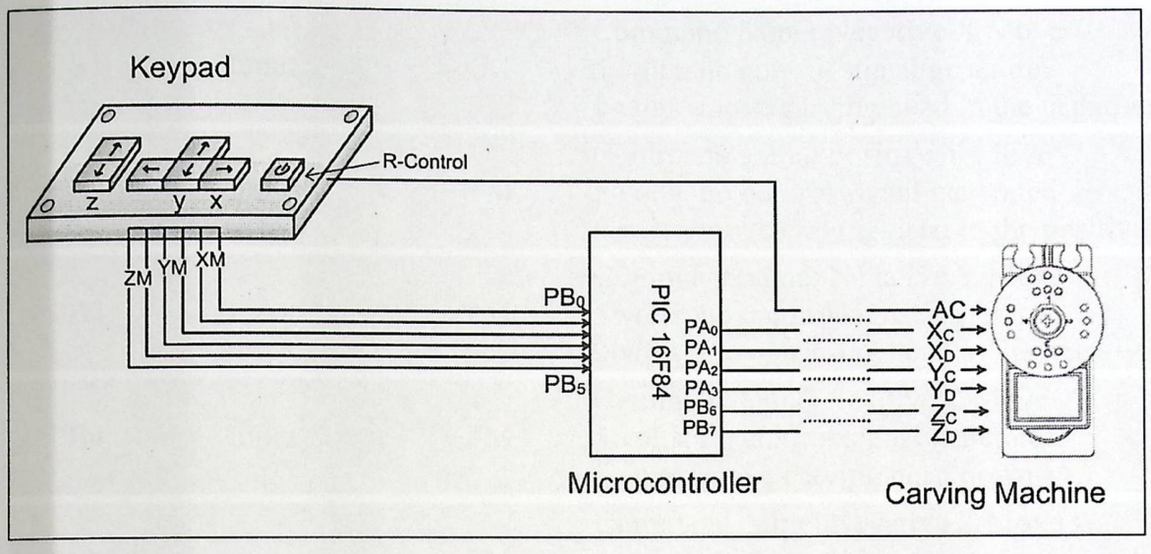


Figure 3.2: Manual machine control diagram

The circuit consists of a 16F84 PIC microcontroller, with the following configuration:

- Port B (PB0- PB5) configured as input lines, (PB6, PB7) configured as output lines.
- Port A (PA0 – PA3) configured as output lines.

The buttons for moving the carving head in the three directions are push buttons, so that the command will be executed as long as the button is pushed, if the button is released the command terminates. The last button is used for spinning the drill and it's an ON/OFF switch. The table shown below describes each of the signals generated by the buttons with a brief description.

Table 3.2: Manual machine control buttons

Pin Name	Input/Output	Command	Command Description
PB0	Input	+XM	Command name: Positive X Move 5 volts: no control signal generated 0 volts: move carving head in the positive X direction

PB1	Input	-XM	Command name: Negative X Move 5 volts: no control signal generated 0 volts: move carving head in the negative X direction
PB2	Input	+YM	Command name: Positive Y Move 5 volts: no control signal generated 0 volts: move carving head in the positive Y direction
PB3	Input	-YM	Command name: Negative Y Move 5 volts: no control signal generated 0 volts: move carving head in the negative Y direction
PB4	Input	+ZM	Command name: Positive Z Move 5 volts: no control signal generated 0 volts: move carving head down
PB5	Input	-ZM	Command name: Negative Z Move 5 volts: no control signal generated 0 volts: move carving head up
PA0	Output	X-pulse	A clock pulse signal to control the X servo motor
PA1	Output	X-Direction	Direction of the X motor CW or CCW 5 volts: CCW translated to backward motion 0 volts: CW translated to forward motion
PA2	Output	Y-pulse	A clock pulse signal to control the Y servo motor
PA3	Output	Y-Direction	Direction of the Y motor CW or CCW 5 volts: CCW translated to backward motion 0 volts: CW translated to forward motion
PB6	Output	Z-pulse	A clock pulse signal to control the Z servo motor
PB7	Output	Z-Direction	Direction of the Z motor CW or CCW 5 volts: CCW translated to backward motion 0 volts: CW translated to forward motion
*	-	R-control	Command name: Rotation Control 0 volts: stop the rotation of the carving head 5 volts: start the rotation of the carving head

*: R-control signal is directly connected – from the keypad – to the DC signal without passing through the microcontroller. The rest of the keypad buttons are connected to Port B of the microcontroller and depending on these values the PIC will generate output signals for each motor.

PIC microcontroller will be programmed to control manual operation; it will read Port B values and then generate control signals and output them on PA0 – PA3, PB6 and PB7. Figure 3.3 shows a flow chart diagram of the program operation.

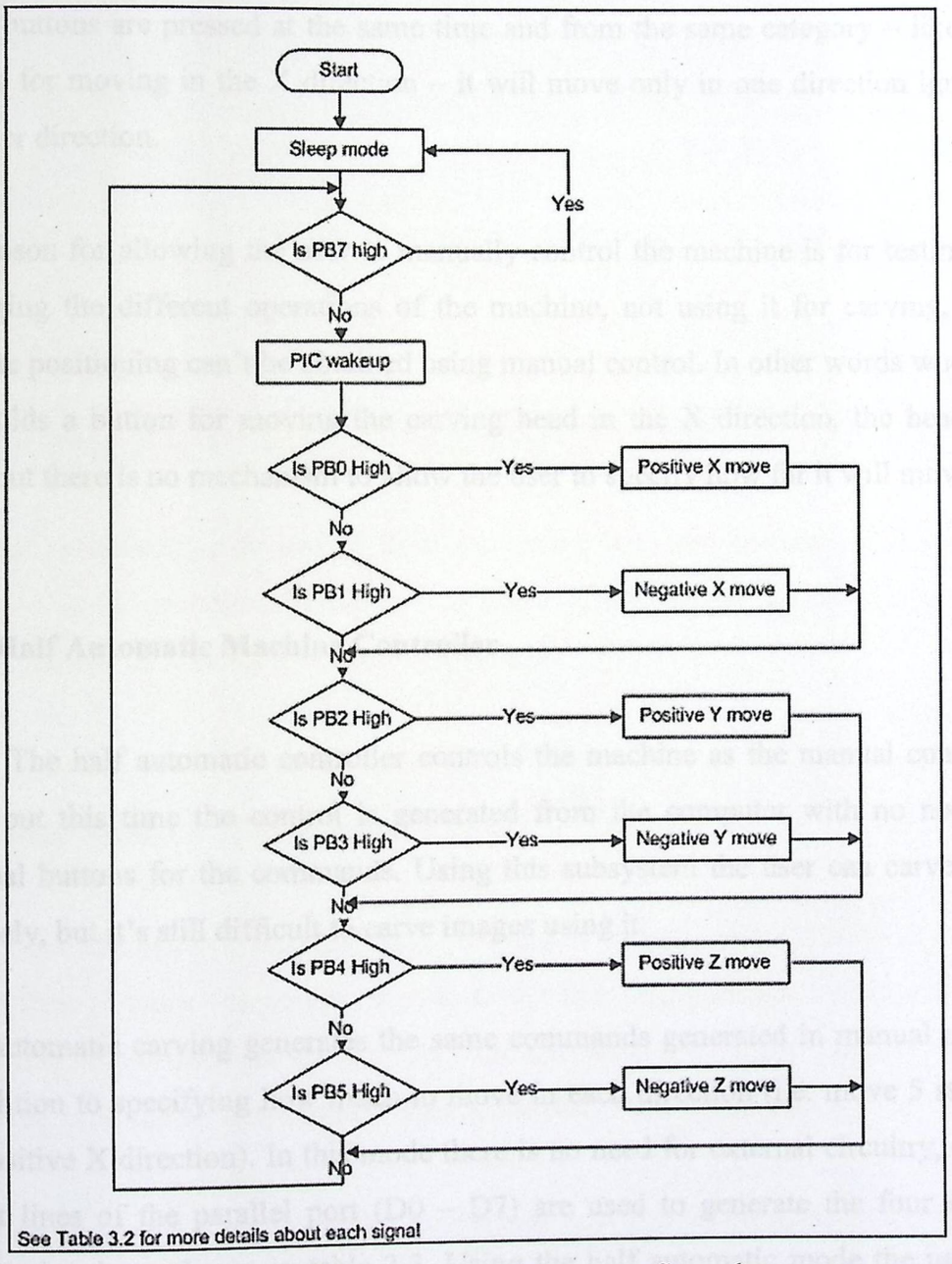


Figure 3.3: Manual control program flow chart

After starting the program it will check which buttons are hold generating the corresponding control signals. And then go back to the beginning of the program and check the buttons again.

If two buttons are pressed at the same time and from the same category – like both buttons for moving in the X direction – it will move only in one direction ignoring the other direction.

The reason for allowing the user to manually control the machine is for testing and debugging the different operations of the machine, not using it for carving, since accurate positioning can't be obtained using manual control. In other words when the user holds a button for moving the carving head in the X direction, the head will move but there is no mechanism to allow the user to specify how far it will move.

3.4.3 Half Automatic Machine Controller

The half automatic controller controls the machine as the manual controller does, but this time the control is generated from the computer with no need for external buttons for the commands. Using this subsystem the user can carve more precisely, but it's still difficult to carve images using it.

Half automatic carving generates the same commands generated in manual control in addition to specifying how much to move in each direction (i.e. move 5 steps in the positive X direction). In this mode there is no need for external circuitry, only 8 output lines of the parallel port (D0 – D7) are used to generate the four motors control signals as shown in table 3.3. Using the half automatic mode the user can enable and disable the HPE (hardware protection enable) line through the parallel port D7 line.

Table 3.3: Parallel port output lines and their corresponding motor control signals

Parallel Port Line	Corresponding Motor
D0	X motor control pulse
D1	Y motor control pulse
D2	Z motor control pulse
D3	Controlling the on/off operation of the DC rotation motor
D4	For controlling the rotation direction of the X servo motor, whether it's Clock Wise or Counter Clock Wise.
D5	For controlling the rotation direction of the Y servo motor, whether it's Clock Wise or Counter Clock Wise.
D6	For controlling the rotation direction of the Z servo motor, whether it's Clock Wise or Counter Clock Wise.
D7	HPE, hardware protection enable. Used to enable or disable the hardware protection if an error is encountered.

The half automatic mode can be used for testing both the hardware and software operation of the machine, in addition to carve simple shapes which can be easily understood by the user such as squares and lines.

3.4.4 Image Analyzer (Full automatic mode)

This subsystem is responsible for reading an image; scale it if needed, analyze it, and then generate the control signals to the interfacing circuit which will cause the motion of the different motors. Only the hardware needed for this subsystem is explained here, the software implementation is described in Chapter 5.

This subsystem is full automatic where the user provides an image to be carved, and then the subsystem will analyze it using a group of algorithms – described in chapter 5 –. After analyzing the image it will generate 7 control signals through the parallel port lines D0-D6 for controlling the motors as shown in table 3.3.

3.4.5 Monitoring subsystem

Responsible for monitoring the whole operation of the machine, showing carving progress on the computer screen, reporting and notifying the user about any errors encountered, and providing a hardwired protection mechanism in the machine.

The monitoring system is a hybrid of software and hardware, its software part shows the carving operation on the computer by drawing the image pixel by pixel and shading the completed areas of the image, another part of this subsystem is the notification of errors, more details about the software implementation of the monitoring system are described in chapter 5.

The hardware part of the monitoring subsystem consists of a circuit for reporting errors to the software part, a hardware solution used to stop the machine if an error exists, and a LED's panel that indicates which line is active during the carving process. Figure 3.4 shows the circuit used in this subsystem.

Another part of the monitoring subsystem is responsible for sending feedback information from the three servo motors notifying the software program that the motors has reached the destination point, these signals are considered as acknowledgments.

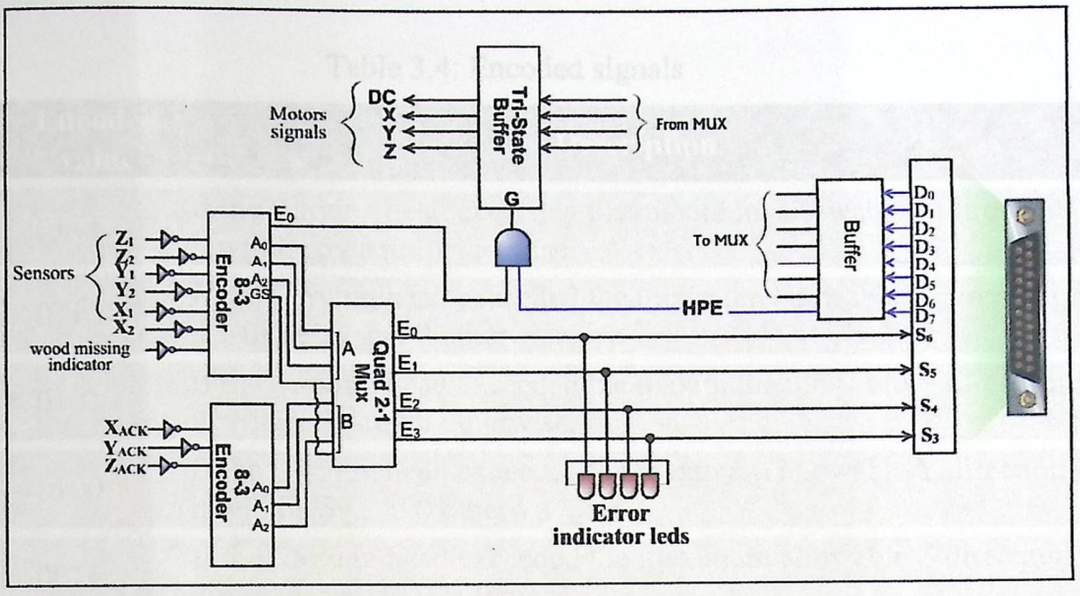


Figure 3.4: Monitoring System Circuit

As shown the state of the machine is signaled to the monitoring system through 4 input lines, sent to the computer through the parallel port input lines (S3 – S6). However the number of inputs from the machine is 10 but the parallel port has only 5 input lines, so an encoding circuit is used to encode the 10 different lines to 4 bits of data sent to the computer. Using 4 bits give the opportunity to up to 16 distinct inputs that are enough for 10 input lines.

Table 3.4: Encoded signals

Input name	Encoded value	Description
EX1	0111	If the Carving head exceeded the maximum allowable X direction it will be 5v and 0 otherwise.
EX2	0110	If the Carving head exceeded the minimum allowable X direction it will be 5v and 0 otherwise.
EY1	0101	If the Carving head exceeded the maximum allowable Y direction it will be 5v and 0 otherwise.
EY2	0100	If the Carving head exceeded the maximum allowable Y direction it will be 5v and 0 otherwise.
EZ1	0011	If the Carving head exceeded the maximum allowable Z direction it will be 5v and 0 otherwise.
EZ2	0010	If the Carving head exceeded the maximum allowable Z direction it will be 5v and 0 otherwise.
NW	0001	5 volt signal when there is no piece of wood mounted in the machine
x-ack	1110	X motor acknowledgement line
y-ack	1101	Y motor acknowledgement line
z-ack	1100	Z motor acknowledgement line

A group of LED's are used as indicators for the user; they will notify the user about the values sent during the carving operation these LED's are specified as follows:

- L1: X servo motor indicator; it will be on when the carving head is moving in the X direction. And turn off otherwise.
- L2: Y servo motor indicator; it will be on when the carving head is moving in the Y direction. And turn off otherwise.
- L3: Z servo motor indicator; it will be on when the carving head is down (i.e. in carving process), and turn off when the head is not carving.
- L4: Drill Motor indicator; it will be on when the drill of the machine is rotating, and off when it's stopped.

- L5: no piece of wood indicator; it will turn on when there is no piece of wood mounted on the machine.
- L6: direction of the X motor.
- L7: direction of the Y motor.
- L8: direction of the Z motor.
- L9: X direction error indicator; it will turn on to notify the user that the carving head is outside the allowed X position.
- L10: Y direction error indicator; it will turn on to notify the user that the carving head is outside the allowed Y position.
- L11: Z direction error indicator; it will turn on to notify the user that the carving head has exceeded the allowable depth.
- L12: X acknowledgment.
- L13: Y acknowledgment.
- L14: Z acknowledgment.
- L15: power on.
- L16: operation mode, notifies whether the control is from the computer or from the keypad.

3.4.6 Drawing subsystem

A utility program that allows the user to open, edit, and draw images to be carved later on the computer, this subsystem will be tailored for creating images and decorations to be carved on wood.

No hardware is used in this subsystem, so it will be described in more detail in chapter 5.

3.5 Interfacing Circuit

The interfacing circuit interfaces and connects the different subsystems, it can be considered as an interpreter between the software part of the system and the hardware part; it will read signals from the sensors of the carving machine and send them to the computer, and output control signals from the computer, adjust them to the needed voltage value, and send them to the motors.

The interfacing circuit depends mainly on the PC parallel port (printer port) for all the Input/Output operations, this decreases the needed hardware components in the circuit minimizing the total cost and simplifying the control operation. The circuit in figure 3.5 shows the whole interfacing circuit.

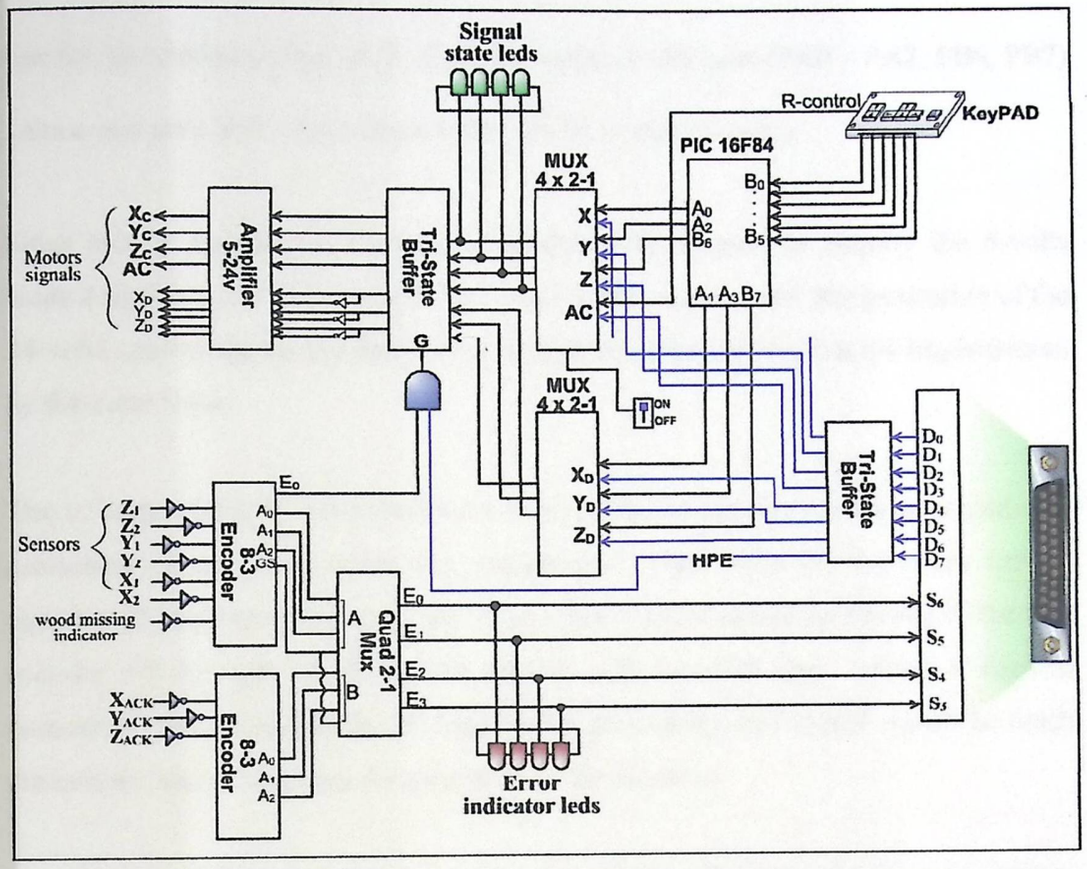


Figure 3.5: Interfacing circuit block diagram

The interfacing circuit contains hardware parts of the subsystems which are the manual control, monitoring system, image analyzer, and half automatic. So the circuit parts related to these subsystems are not described in this section.

Control signals passed to the motors are generated using the PIC in the machine manual control mode and carried on lines PA0 – PA3, PB6 PB7, or by the PC parallel port and carried on the lines D0 – D6 in the half automatic and automatic modes. But it's not allowed that the control signals are generated by both parts at the same time – PIC and parallel port –. So the circuit must be able to determine if the motor control signals are on (PA0 – PA2, PB6, PB7) or (D0 – D6) lines, and then pass them to amplification stage. This is done using a multiplexer with $\overline{MCE}$,

line for its selection when $\overline{MCE}$ is low (0-volts) it will pass (PA0 – PA2, PB6, PB7) values, and pass (D0 – D6) values when $\overline{MCE}$ is high (5-volts).

Since motors operate on 24 volts an amplifier IC is used to amplify the 5-volts control signals to 24-volts and sending them to the motors, after the generation of the 24-volts control signals the data will propagate in other circuits that are implemented by the other team.

The tri-state buffer IC (74LS368A) is used in the interfacing circuit as a hardware protection circuit when errors are encountered, when there are no errors control signals will pass through it normally. If an error is encountered the E0 line of the first encoder will be high which will be ANDed with the HPE line – which is high in common – and so disable the tri-state buffer preventing the control signals to reach the motors, and so stopping the operation of the machine.

Note that the HPE line is connected controlled by line D7 of the parallel port, if D7 is high the hardware protection circuit is enabled, but if D7 is low the hardware protection circuit is disabled, allowing control signals to be sent to the machine even if there are errors. This is important to allow the machine to recover from errors, and receive new control signals to position the carving head to the starting point.

As scenario suppose that while the machine is in a carving process, the carving head had exceeded the maximum allowable x distance, at this point the hardware protection enable circuit will buffer the control signals, and prevent them from reaching the carving machine, at the same time the user will be notified about that certain error. At this point the user has to reinitialize the machine, but to send signals he has to disable the HPE line and then send the initialization signals.

3.6 How the system works

In chapter one – introduction – we have described how the system should work at a high level of abstraction where an image is provided for the driving program on the computer side, and then control signals are sent to the motors to carve this image on the piece of wood after analyzing the image.

In more details to carve the image into the piece of wood the system should pass through the following steps:

- Get the image that the user desires to carve on the piece of wood.
- Specify the Dimensions of the piece of wood we wish to carve the image on it.
- Scale the image by a factor where each pixel in the image will correspond to a hole carved in the wood by the drill of the machine. for example if the image is of size 400 * 300 pixels and the piece of wood is 80 * 60 cm and the drill diameter is 3mm. then the image should be scaled by a factor of 2/3 so that each pixel in the image will correspond to a circle of 3mm diameter on the piece of wood. Figure 3.6 shows this operation.

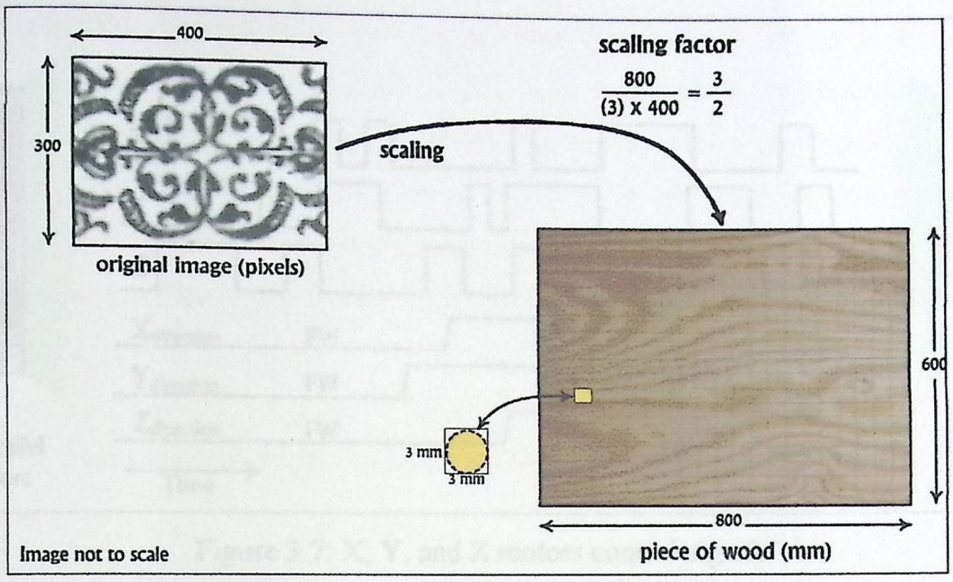


Figure 3.6: Scaling an image to device coordinates

- Analyze the scaled image, first it will analyze the outline of the shape and carve it, then it will dissect the interior of the shape it into a group of horizontal and vertical lines and store them in a data structure. The machine will first carve the horizontal lines and then carve the vertical lines. Note that horizontal lines would be enough and complete but adding vertical lines will result with a better carving quality.
- Carving the lines should be done in a systematic way, with a big concern for the order in which lines are to be carved; in order to decrease the total time needed to accomplish the carving process. So after carving one line the head will move to the nearest line to its current location and saving time by doing so. And then store the sequence of lines to be carved in a file.
- Translate the sequence from the file to three ON/OFF on the parameters of the line segments, each signal will be used to control the motion of the X, Y, Z servo motors, and store them in a file. Figure 3.7 shows an example for these signals.

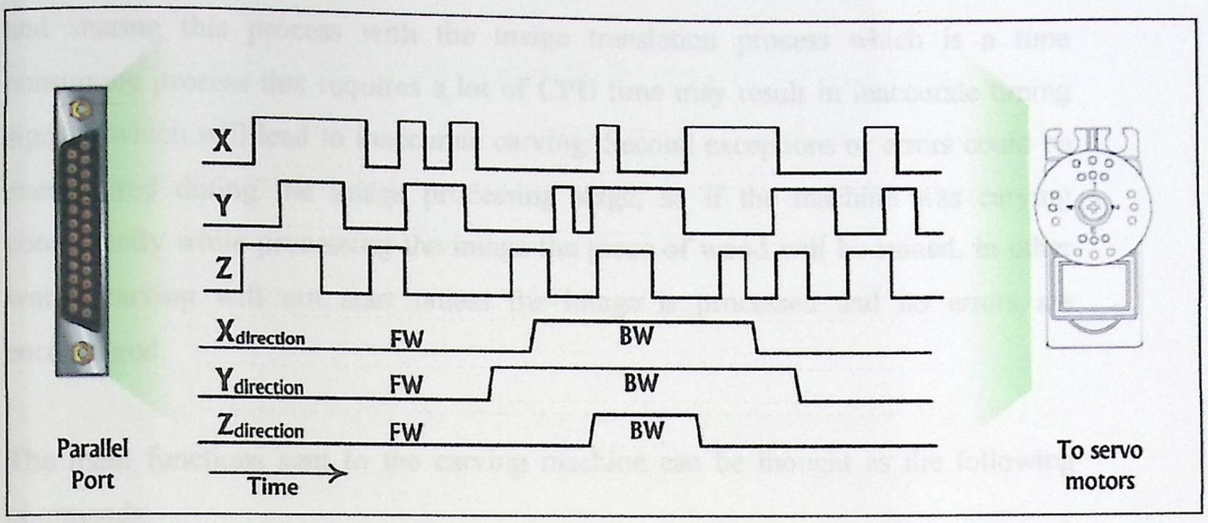


Figure 3.7: X, Y, and Z motors control signals

- Initialize the machine to start a new job moving the carving head to location (0, 0), and turn on the DC motor which will rotate the carving drill.
- Read the needed control signals generated in point 6 and output them on the parallel port data lines D0-D6.
- Amplify the 5v signals generated by the computer to 24v using an appropriate amplifier circuit and send the values to the motors.
- The motors will then start their operation and carve the image on the piece of wood.
- While the carving process is working, the progress of the operation is stored in a file so that if electricity turns off, the user will be able to continue the operation from the point where the machine had stopped.

From the points mentioned above it can be noticed that the machine will not start the carving operation, unless the translation and conversion of the image is finished, and the needed control signals for the motors are known and stored in a file. After finishing this stage the computer starts sending the control signals to the interfacing circuit by simply reading from the stored file. There are two reasons for doing this, first because the control signals must be very accurate -of microsecond precision-

and sharing this process with the image translation process which is a time consuming process that requires a lot of CPU time may result in inaccurate timing signals; which will lead to inaccurate carving. Second exceptions or errors could be encountered during the image processing stage; so if the machine was carving concurrently while processing the image the piece of wood will be ruined, in other words carving will not start unless the image is processed and no errors are encountered.

The main functions sent to the carving machine can be thought as the following commands:

1) Move to location (x, y).
2) Carve to location (x, y).
3) Move the carving head down by a depth value of "d" mm where "d" is a certain number.
4) Move the carving head up by a value of "d" mm.
5) Initialize the machine.
6) Rotate the carving drill.
7) Stop the rotation of the carving drill.

A data flow diagram describing the software application stages of the machine driver is shown in Figure 3.8

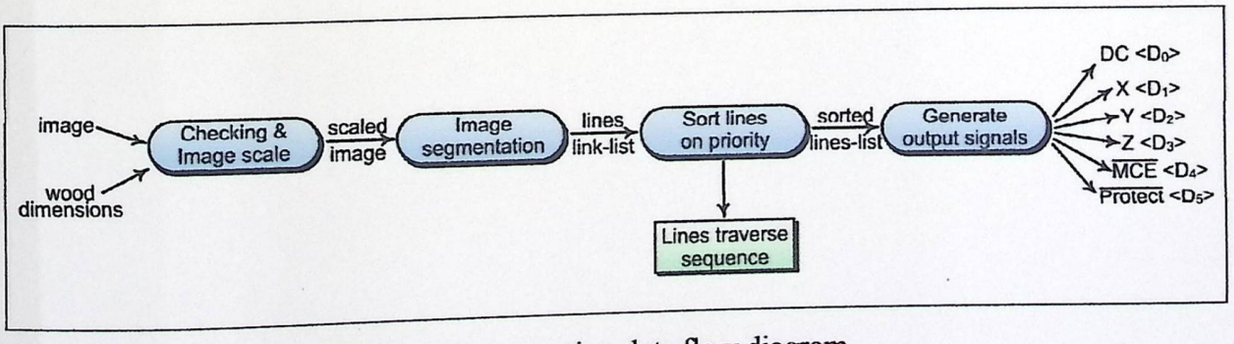


Figure 3.8: Machine operation data flow diagram.

Another important part of the machine driving program is the monitoring system which will monitor the progress of the carving operation with an appealing GUI interface showing the current location of the carving head, how much carving did it finished, how much time does it still need to complete the carving, which parts of the machine are currently moving, depth level, and the signals outputted to the machine.

An error controller process will be running on the computer while the carving operation is in progress. Its main function is to detect and report errors that are encountered during the carve operation and receive feedback information from the machine, these errors are fired when the head exceeds the allowed locations in both X, Y, and Z directions, or when there is no piece of wood mounted on the machine. The monitoring controller will report any encountered errors and notify the user about them and abort the carving operation if needed. Feedback information consists of three acknowledgment signals sent back to the computer when the motor reaches the destination location.

Figure 4.2 describes the hardware architectural model showing the operation of each component of the interfacing circuit and how control signals propagates to the motors and how errors are reported from the machine sensors.

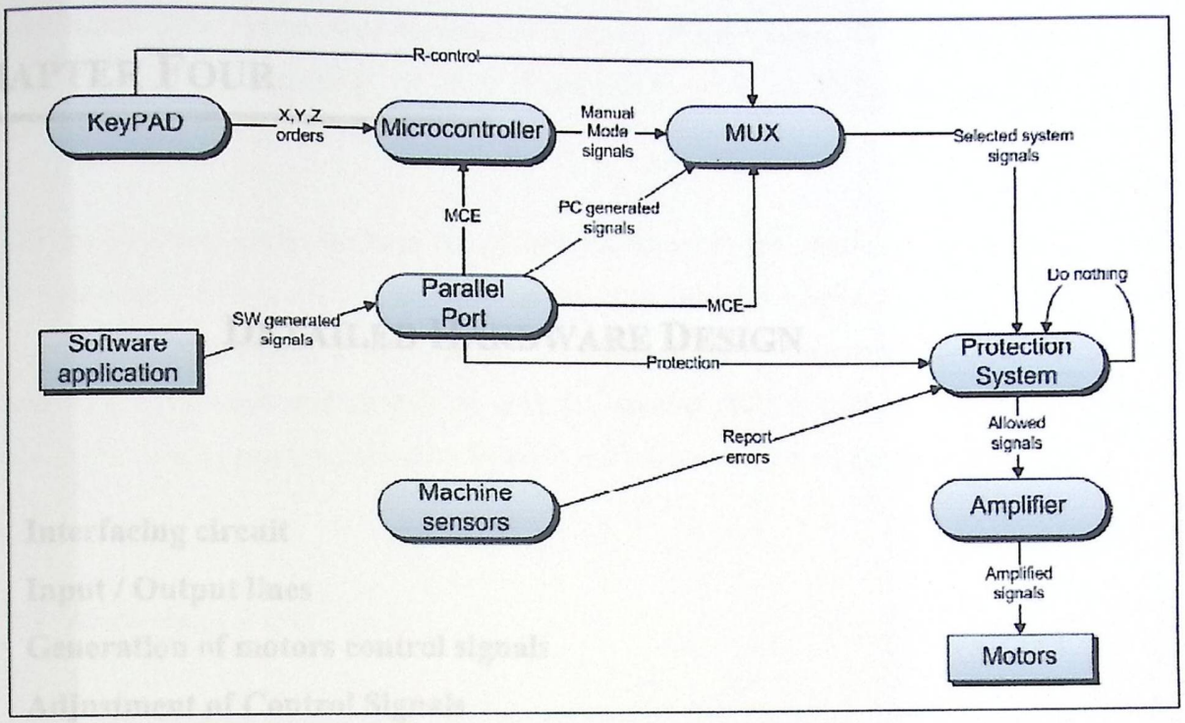


Figure 3.9: Architectural model of the interfacing circuit

CHAPTER FOUR

DETAILED HARDWARE DESIGN

4.1 Interfacing circuit

4.2 Input / Output lines

4.3 Generation of motors control signals

4.4 Adjustment of Control Signals

4.5 Hardware Protection Circuit

4.6 LED's Panel

4.7 Microcontroller connection and manual control

In Chapter 3 the system was modeled as a group of subsystems, these subsystems where classified depending on their functionality with no distinction between software and hardware components.

All the hardware components in the system are found in the interfacing circuit and the carving machine. In this chapter we will consider the interfacing circuit, and describe its functionality in more details, showing schematic diagrams of the circuits and how it operates and specifying each IC number. While detailed information about the carving machine could be found in the documentation of the other team.

4.1 Interfacing Circuit

The interfacing circuit shown in Figure 4.1, consists of a group of components, these components are used to generate and control different functions in the machine.

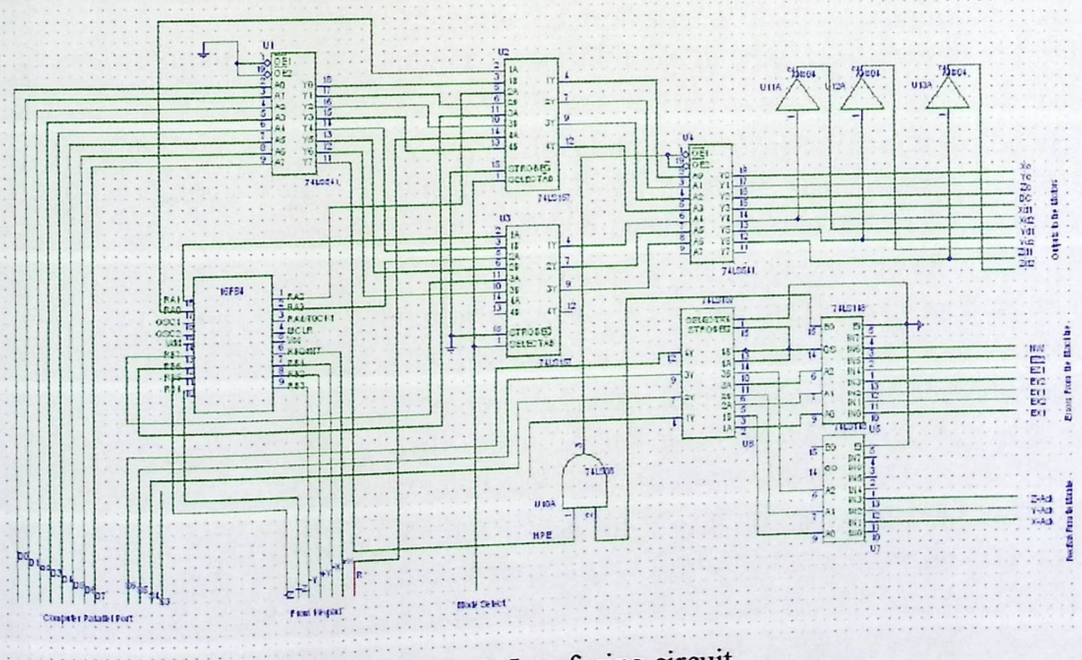


Figure 4.1: Interfacing circuit

The following sections will describe each component in the interfacing circuit in more details specifying how it works, its inputs and outputs.

4.2 Input / Output Lines

As stated before the interfacing circuit stands between the computer and the carving machine, so it is used to send control signals to the machine, and send data and information about the machine to the computer. The main output lines in the interfacing circuit are Xc, Xd, Yc, Yd, Zc, Zd, DC, and HPE, while the input lines are NW, EX1, EX2, EY1, EY2, EZ1, EZ2, X-ack, Y-ack, Z-ack. Table 4.1 describes each of these lines and the type of information they carry.

Table 4.1: Interfacing circuit signals

Signal name	Input / Output	Description
Xc	Output	X motor control signal, used to control the servo motor responsible for moving the carving head in the X direction.
Xd	Output	X motor direction control signal, if 0 then the motor will rotate CW (X position will increase) else it will rotate CCW (X position will decrease).
Yc	Output	Y motor control signal, used to control the servo motor responsible for moving the carving head in the Y direction.
Yd	Output	Y motor direction control signal, if 0 then the motor will rotate CW (Y position will increase) else it will rotate CCW (Y position will decrease).
Zc	Output	Z motor control signal, used to control the servo motor responsible for moving the carving head in the Z direction.
Zd	Output	Z motor direction control signal, if 0 then the motor will rotate CW (Z position will increase) else it will rotate CCW (Z position will decrease).
DC	Output	An on/off signal which is used to control the DC motor used to spin the drill.

HPE	Output/Status	Hardware Protection Enable, an active low signal used to notify the interfacing circuit whether hardware protection is enabled or not.
NW	Input	No Wood, an input signal that notifies the computer when there is no piece of wood mounted on the machine.
EX1	Input	X Position Error, it will be high (5-volts) when the carving head exceeds the maximum allowable X
EX2	Input	X Position Error, it will be high (5-volts) when the carving head exceeds the minimum allowable X
EY1	Input	Y Position Error, it will be high (5-volts) when the carving head exceeds the maximum allowable Y
EY2	Input	Y Position Error, it will be high (5-volts) when the carving head exceeds the minimum allowable Y
EZ1	Input	Z Position Error, it will be high (5-volts) when the carving head exceeds the maximum allowable Z
EZ2	Input	Z Position Error, it will be high (5-volts) when the carving head exceeds the minimum allowable Z
X-ack	Input	Feedback information from the X servo motor, considered as acknowledgment sent back to the computer notifying that the servo has reached the destination point.
Y-ack	Input	Same as Xack but its used for the Y motor.
Z-ack	Input	Same as Xack but its used for the Y motor.

4.3 Generation of Motors Control Signals

The control signals Xc, Xd, Yc, Yd, Zc, Zd, DC are generated from two different parts; from the PC parallel port when operating in the half or full automatic modes or from the PIC microcontroller when operating in the manual mode.

The control signals generated from the PC parallel port are carried through its data lines D0 – D6 for Xc, Xd, Yc, Yd, Zc, Zd, DC respectively, while the control signals in the manual mode are carried through the microcontroller lines A0 – A3 B6 B7 for

Xc, Xd, Yc, Yd, Zc, Zd, respectively and the keypad button R-Control for the DC motor.

These two sources of – motors control signals – are multiplexed and only one source is used to control the motors at the same time. The decision is based on the machine operation mode (manual, half-automatic, automatic) which is signaled using the status line $\overline{\text{MCE}}$. And here comes the reason of using the multiplexers – see Figure 4.1 – which consists of two quad 2-lines to 1-line multiplexer with a common selection line connected to $\overline{\text{MCE}}$. The inputs of the multiplexers are shown in Table 4.2 where each two signals with the same function are the inputs for each Multiplexer (i.e. D3 and R-Control, D1 and PA0). The schematic diagram of the 74157 multiplexers is shown in Figure 4.2.

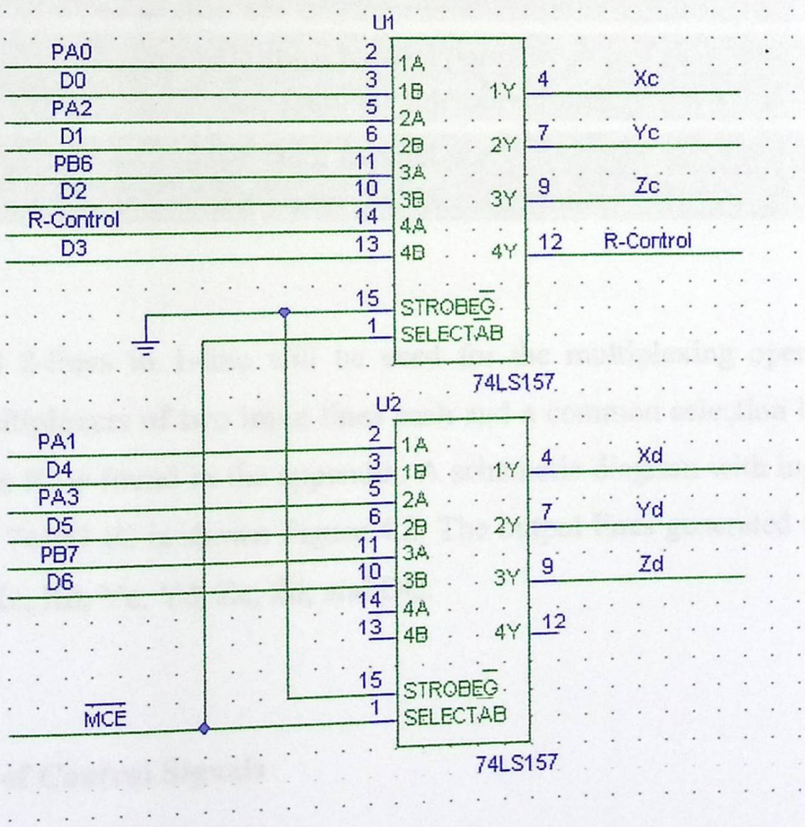


Figure 4.2: 74157 Multiplexer schematic

Table 4.2: Multiplexer input lines

Multiplexer number	Input line
Mux #1	D3 from parallel port R-Rotate from keypad
Mux #2	D0 from parallel port PA0 from the microcontroller
Mux #3	D1 from parallel port PA2 from the microcontroller
Mux #4	D2 from parallel port PB6 from the microcontroller
Mux #5	D4 from parallel port PA1 from the microcontroller
Mux #6	D5 from parallel port PA3 from the microcontroller
Mux #7	D6 from parallel port PB7 from the microcontroller

The 74157 quad 2-lines to 1-line will be used for the multiplexing operation; it contains four multiplexers of two input lines each and a common selection line. The data sheet for this IC is found in the appendix. A schematic diagram with input lines connected to the 74157 IC is shown Figure 4.2. The output lines generated from the multiplexer are Xc, Xd, Yc, Yd, Zc, Zd, and DC.

4.4 Adjustment of Control Signals

The control signals outputted from the 74157 multiplexer IC are of 5 volts value, which is insufficient for controlling the motors since they need 24 volts signals to operate.

An amplification circuit is constructed to amplify the 5-volts signals to 24-volts signals and then pass new amplified signals to the motors; the circuit consists of 4 transistors configured to work as switches. A schematic of the circuit is shown in Figure 4.3.

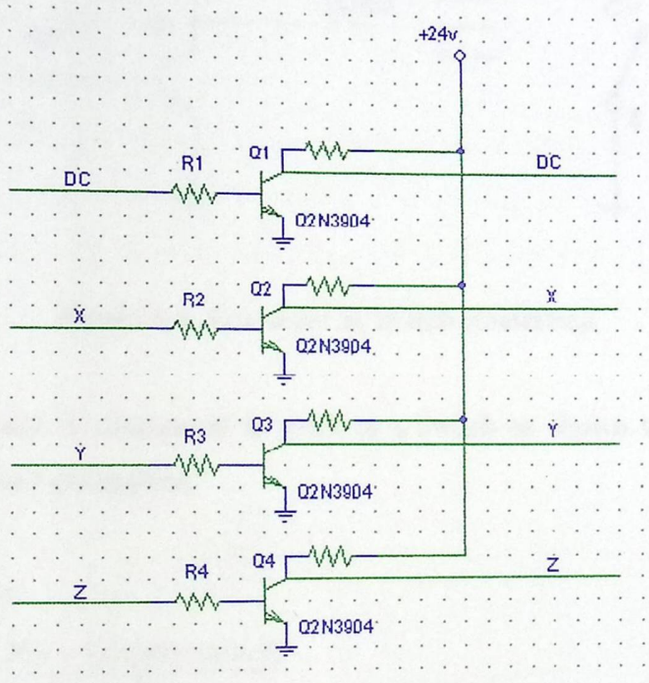


Figure 4.3: Amplification circuit schematic

Each line of 5-volts value is connected to the base of the transistor. When the value on the base of the transistor is of 5-volts the 24-volts value is passed to the corresponding output line, when the base has a 0-volts value the output value will be of 0 volts.

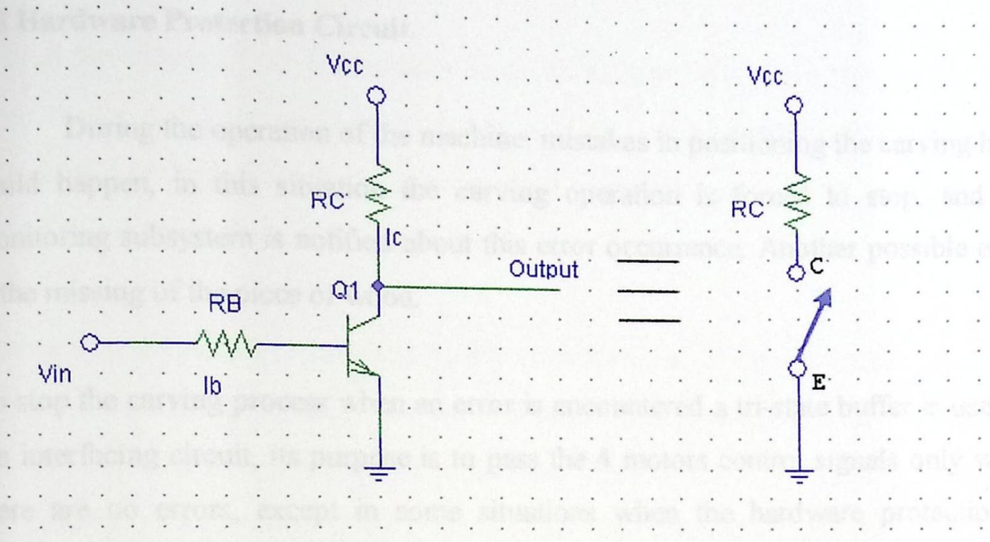


Figure 4.4: Transistor as switch connection

When the transistor is configured to work as a switch as shown in Figure 4.4 the following equations are applied.

$$Vce(cutoff) = Vcc$$

$$Ic(saturation) = \frac{Vcc - Vce(saturation)}{Rc}$$

$$Ib(\min) = \frac{Ic(saturation)}{Bdc}$$

4.5 Hardware Protection Circuit

During the operation of the machine, mistakes in positioning the carving head could happen, in this situation the carving operation is forced to stop, and the monitoring subsystem is notified about this error occurrence. Another possible error is the missing of the piece of wood.

To stop the carving process when an error is encountered a tri-state buffer is used in the interfacing circuit, its purpose is to pass the 4 motors control signals only when there are no errors, except in some situations when the hardware protection is disabled – HPE signal is low –. The schematic of the hardware protection circuit is shown in Figure 4.5.

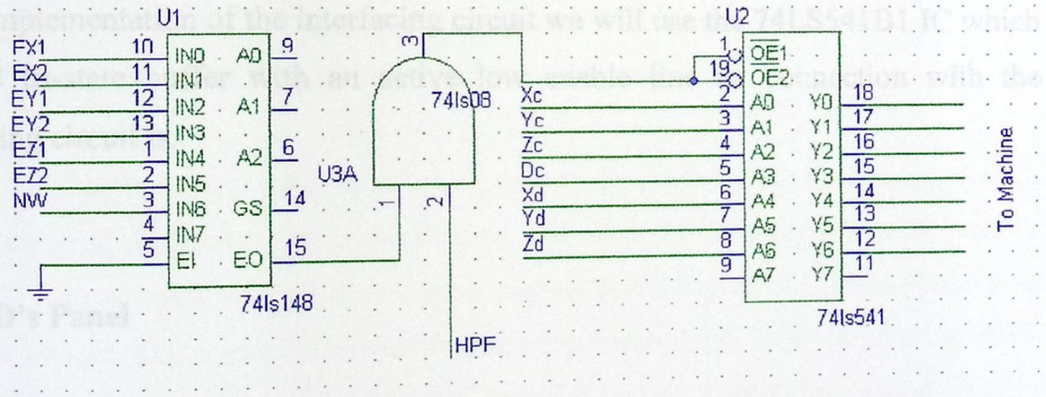


Figure 4.5: Hardware protection circuit

In this circuit there are seven tri-state buffers, the seven possible errors are connected to an encoder (74148) with its E0 line ANDed with the HPE line and then connected to the enable line of the tri-state buffer.

As default the HPE line is high, which enables the hardware protection circuit, supposing that line EX1 goes high signaling that an error in the X position is

encountered the output of the E0 encoder line will be high, and since HPE is high too the output of the AND gate will be high and so TSB1 will be disabled and all of its output lines will be high impedance preventing the control signals to propagate to the motors and so stopping the carving operation.

At the same time this error is encoded by the encoder to an error of number 0111 and sent to the software program using the parallel port status lines S3-S6, in this case the program will reset HPE to logic zero and so the output of the AND gate will be zero and so disabling the protection circuit, then it will send control signals to position the carving head to the start point (0, 0), after the machine is reset to the start location HPE signal will go high enabling the hardware protection circuit again.

In the implementation of the interfacing circuit we will use the 74LS541B1 IC which is an 8 tri-state buffer with an active low enable line its connection with the interfacing circuit is.

4.6 LED's Panel

The LED's panel is used to allow the user to know what is the current data sent to the machine, it consists of 16 LED's for indicating the state of the seven control signals Xc, Xd, Yc, Yd, Zc, Zd, DC and the seven possible error lines NW, EX1, EX2, EY1, EY2, EZ1, EZ2 and the status signals $\overline{\text{\quad\quad}}$,MCE and HPE.

The LED's are connected to all the mentioned lines as shown in Figure 4.6.

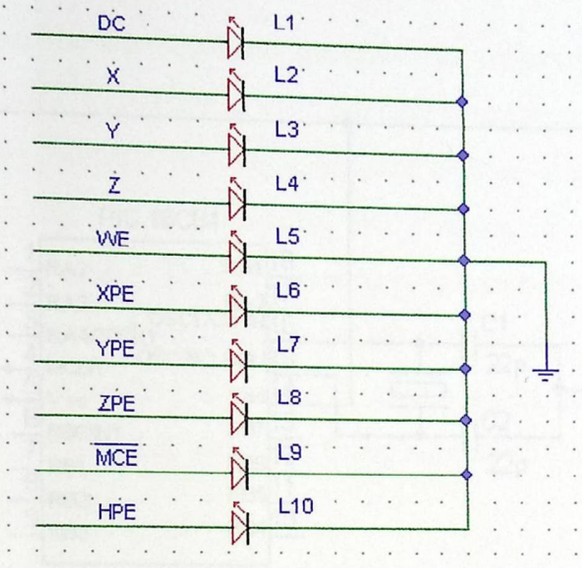


Figure 4.6: LED's panel connection

4.7 Microcontroller Connection and Manual Control

The microcontroller is used to generate the motors control signals when the machine is operating in the manual mode, it is connected to a keypad which consists of 7 buttons and it will generate control signals depending on the pressed buttons on the keypad. The operation needed by the microcontroller is quite simple; capability of input, output, an internal timer, and enough memory to store the needed program inside it, after analyzing these requirements the PIC 16F84 is enough for this circuit.

Only few components are needed to get the microcontroller up and running – see Figure 4.8 –. Primarily a pull-up resistor on Pin 4 (MCLR), a 4 MHz crystal with two 22-pF capacitors, and a 5-V power supply.

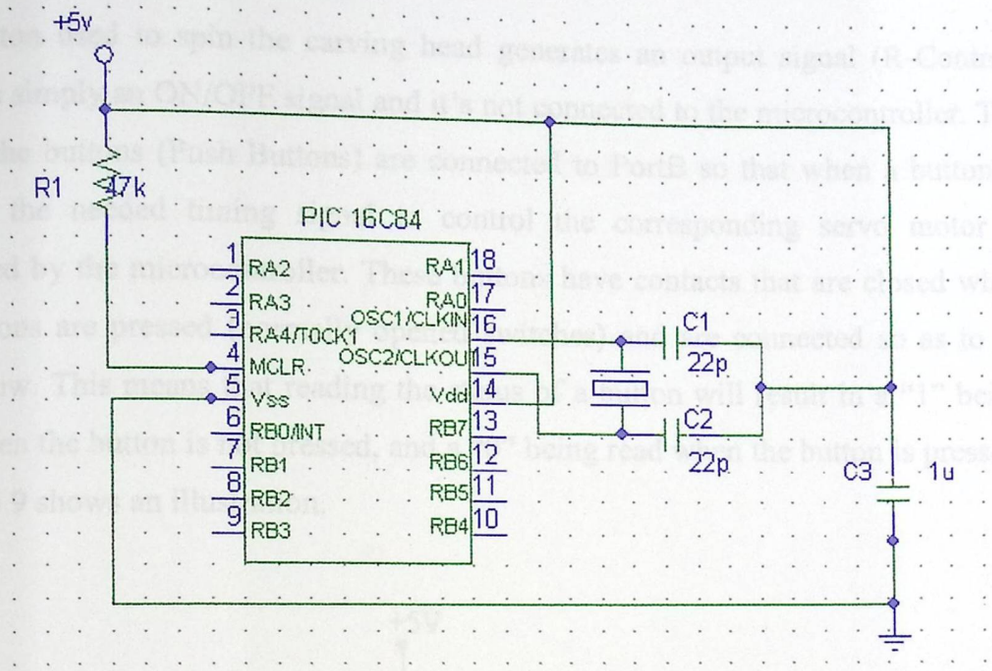


Figure 4.7: Basic 16C84 PIC connection

After connecting the microcontroller the buttons of the keypad must be connected to its ports as shown in Figure 4.8. The outputs generated from the microcontroller are sent on lines PA0 – PA3, PB6, PB7. The functions of the buttons are shown in Table 3.2.

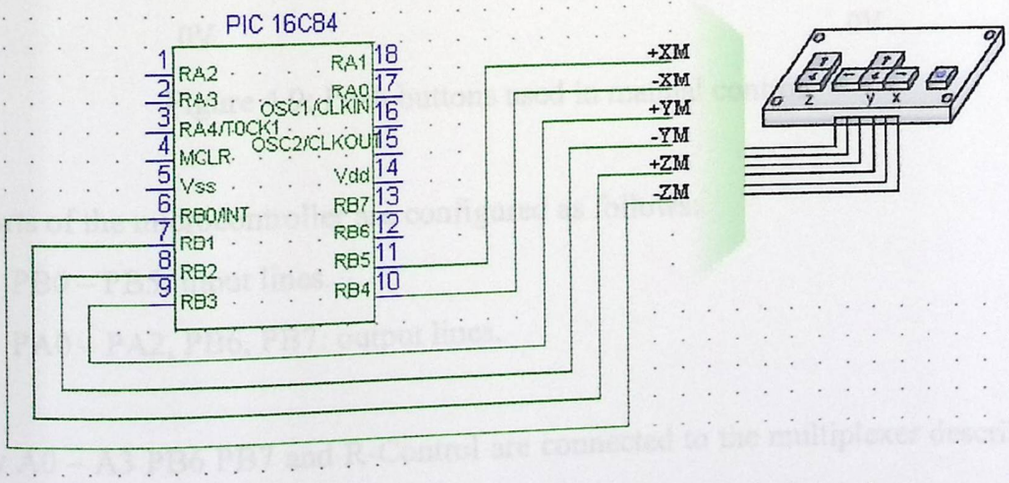


Figure 4.8: Microcontroller connection with the keypad

The button used to spin the carving head generates an output signal (R-Control) which is simply an ON/OFF signal and it's not connected to the microcontroller. The rest of the buttons (Push Buttons) are connected to PortB so that when a button is pressed the needed timing signal to control the corresponding servo motor is generated by the microcontroller. These buttons have contacts that are closed when the buttons are pressed (normally opened switches) and are connected so as to be active low. This means that reading the status of a button will result in a "1" being read when the button is not pressed, and a "0" being read when the button is pressed. Figure 4.9 shows an illustration.

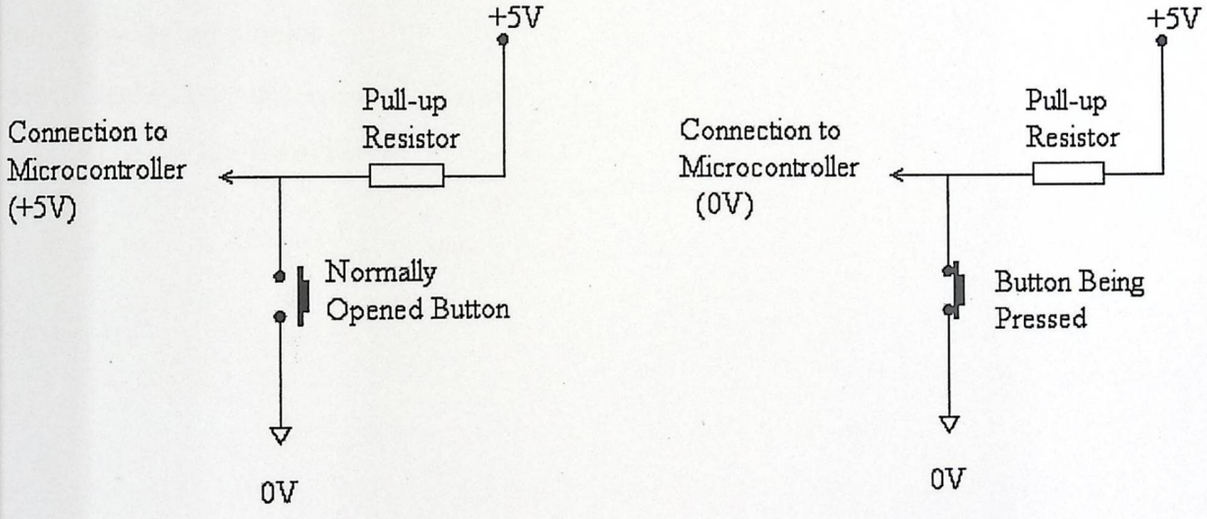


Figure 4.9: Push buttons used in manual control

The ports of the microcontroller are configured as follows:

- PB0 – PB5: input lines.
- PA0 – PA2, PB6, PB7: output lines.

Finally A0 – A3 PB6 PB7 and R-Control are connected to the multiplexer described in section 4.3.

Before the microcontroller is mounted in its socket in the interfacing circuit, it will be programmed using a specialized programming kit; where the program which will run on the microcontroller will be assembled and then loaded into the PIC memory.

CHAPTER FIVE

IMPLEMENTATION

In the previous chapter a complete hardware design was considered, focusing on the interfacing circuit design, in addition to schematics of the various parts of the hardware components of the machine.

While chapter 4 talked about hardware components, in this chapter we will introduce the software implementation of the Wood Carve System, the software mainly consists of three parts; Image analyzer, Monitoring program, and the Drawing program, detailed explanation of each subsystem will be provided in this chapter.

5.1 Image Analyzer

One of the main goals of the project is to be able to process a pre-drawn image, analyze it, and generate suitable control signals sent to the machine hardware to carve it. To achieve this goal an image analyzing subsystem will be developed and implemented.

5.1.1 Image Analyzer Objectives

Image Analyzer should achieve the following objectives:

- Analyze a wood carving system image (WCS) or bitmap image (BMP) to a list of colored images, where each image contains only one color that represents a certain carving depth.
- Process the –one color– images separating the shape fill and outline, and then save each one in a separate file (image).
- Find the coordinates of the lines of the shape outline.

- Break shapes fill into an array of horizontal lines, and another array of vertical lines.
- Produce a control signals file that can be handled by the motors signal generator.

5.1.2 Image Analyzer Classes

Image Analyzer subsystem consists of multiple classes. The Unified Modeling Language (UML) of these classes and the rest of the classes used in the whole system are shown in the appendix.

An overview of the functionality of this subsystem is listed as follows:

1) Image import: responsible for reading a user specified image, this class will mainly deal with WCS format images, which are images drawn using the associated Drawing subsystem. When a WCS image is opened it will be displayed on the drawing program. This class is also responsible for reading .bmp images, which will trigger a dedicated wizard for carving .bmp images.

2) Image processing and analyzing: this class is the core of the application where most of the work is done, the image processor work will be mainly affected by the type of the image, whether .wcs or .bmp, and here is the processing stages for each of them:

   A) WCS format:

   In case the read image was in .wcs format, the image analyzer will first look for its information (i.e. width, height, colors, and shapes list), the

image will then be sent to a *carveOperation* object where each shape is flattened to approximate its curves into lines depending on a specified flatness parameter, which specifies the maximum distance that any point on the unflattened curve can deviate from the returned flattened path segments figure 5.1 clarifies the flattening process.

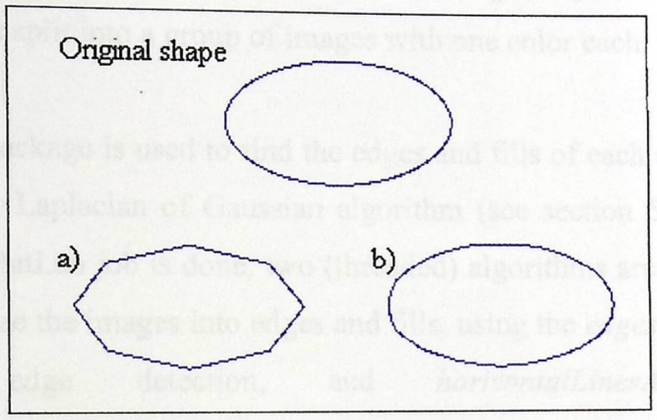


Figure 5.1: Flattening a Shape. a) Flatness =8 b) Flatness =0.5

After flattening the image is analyzed to edges (path of the shape outline), and fill lines, then a control signals file containing the data of the lines is generated, this file is written in plain text with a .csf extension, which stands for (Control Signals File), figure 5.2 shows a small part of the contents of the .CSF file.

```
R
0 82 43
+5
1 82 94
1 83 92
+7
1 236 46
1 239 46
1 241 47
S
```

Figure 5.2: Part of the Signals file (CSF file)

B) BMP format:

In case the read image was in .bmp format, it will pass through multiple stages before its carved, first the image enters the *imageColorCounter* which counts the number of colors in the image and store them in a colors array. Next the image is sent to a *extractColoredImages* object where the original image is split into a group of images with one color each.

Later MatLab package is used to find the edges and fills of each shape in the image using Laplacian of Gaussian algorithm (see section 5.1.4 for details), when MatLab job is done, two (threaded) algorithms are fired in parallel to analyze the images into edges and fills, using the *edgesToLines* class for edge detection, and *horizontalLinesAnalyzer, verticalLinesAnalyzer* classes for finding the fills.

At this point the .bmp image is dealt exactly as .wcs images, where the previous edges and fills are sent to *a signalsFileGenerator* object which writes the suitable signals file, which is the same as the file shown in figure 5.2.

5.1.3 How BMP Image Analyzing Works

The user predefined image should have certain standards so that the program will understand what it should carve and how to carve it. The main constraints on the image are the image type and carving depth determination method.

When a .bmp image is opened the program initiates a specific wizard for the preparation of the carving process, this wizard contains three cascaded screens, as following:

B) BMP format:

In case the read image was in .bmp format, it will pass through multiple stages before its carved, first the image enters the *imageColorCounter* which counts the number of colors in the image and store them in a colors array. Next the image is sent to a *extractColoredImages* object where the original image is split into a group of images with one color each.

Later MatLab package is used to find the edges and fills of each shape in the image using Laplacian of Gaussian algorithm (see section 5.1.4 for details), when MatLab job is done, two (threaded) algorithms are fired in parallel to analyze the images into edges and fills, using the *edgesToLines* class for edge detection, and *horizontalLinesAnalyzer, verticalLinesAnalyzer* classes for finding the fills.

At this point the .bmp image is dealt exactly as .wcs images, where the previous edges and fills are sent to *a signalsFileGenerator* object which writes the suitable signals file, which is the same as the file shown in figure 5.2.

5.1.3 How BMP Image Analyzing Works

The user predefined image should have certain standards so that the program will understand what it should carve and how to carve it. The main constraints on the image are the image type and carving depth determination method.

When a .bmp image is opened the program initiates a specific wizard for the preparation of the carving process, this wizard contains three cascaded screens, as following:

a) First screen of the wizard (Image info), where the program shows the filename, width, height, and find the number of colors in the image as shown in figure 5.3.

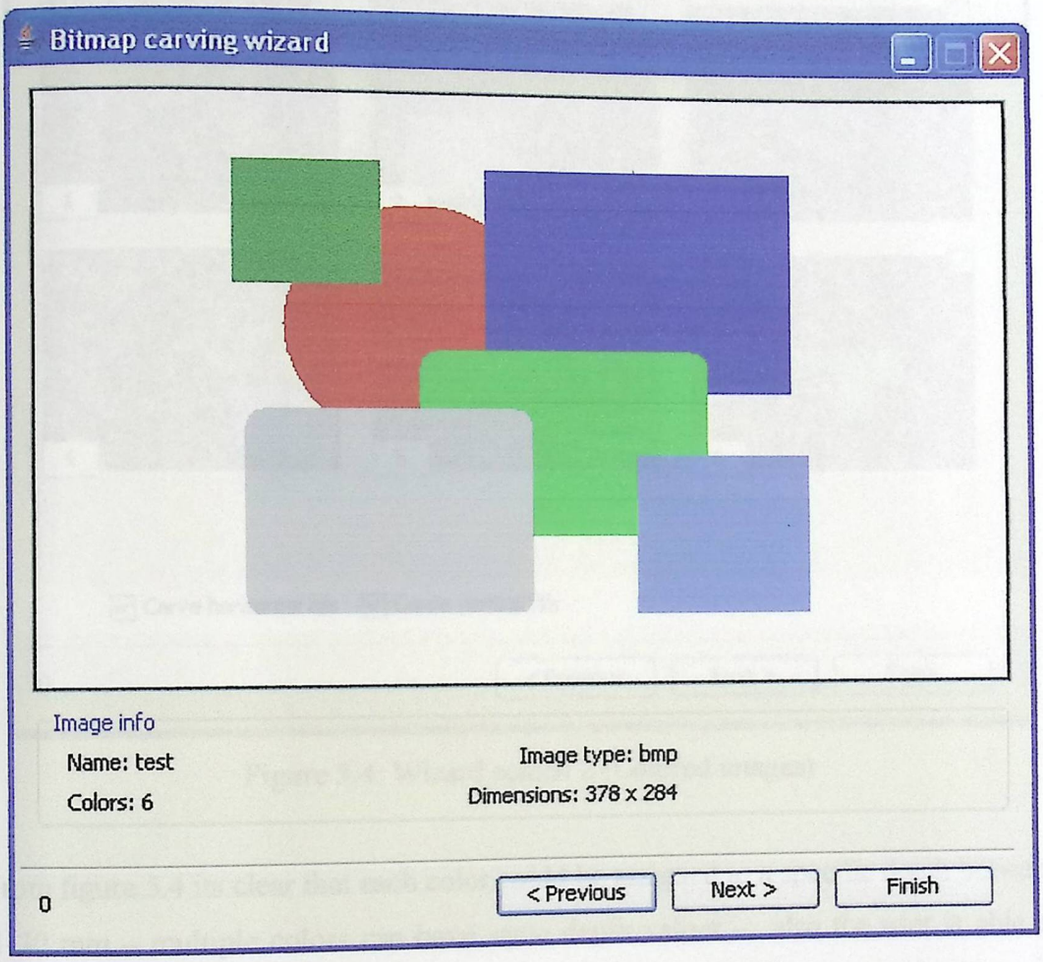


Figure 5.3: Wizard screen 1 (Image info)

b) Second screen of the wizard (Colored images), where the image is split into multiple images, each image contains only one color, that represents a specific carving depth, figure 5.4 shows this screen.

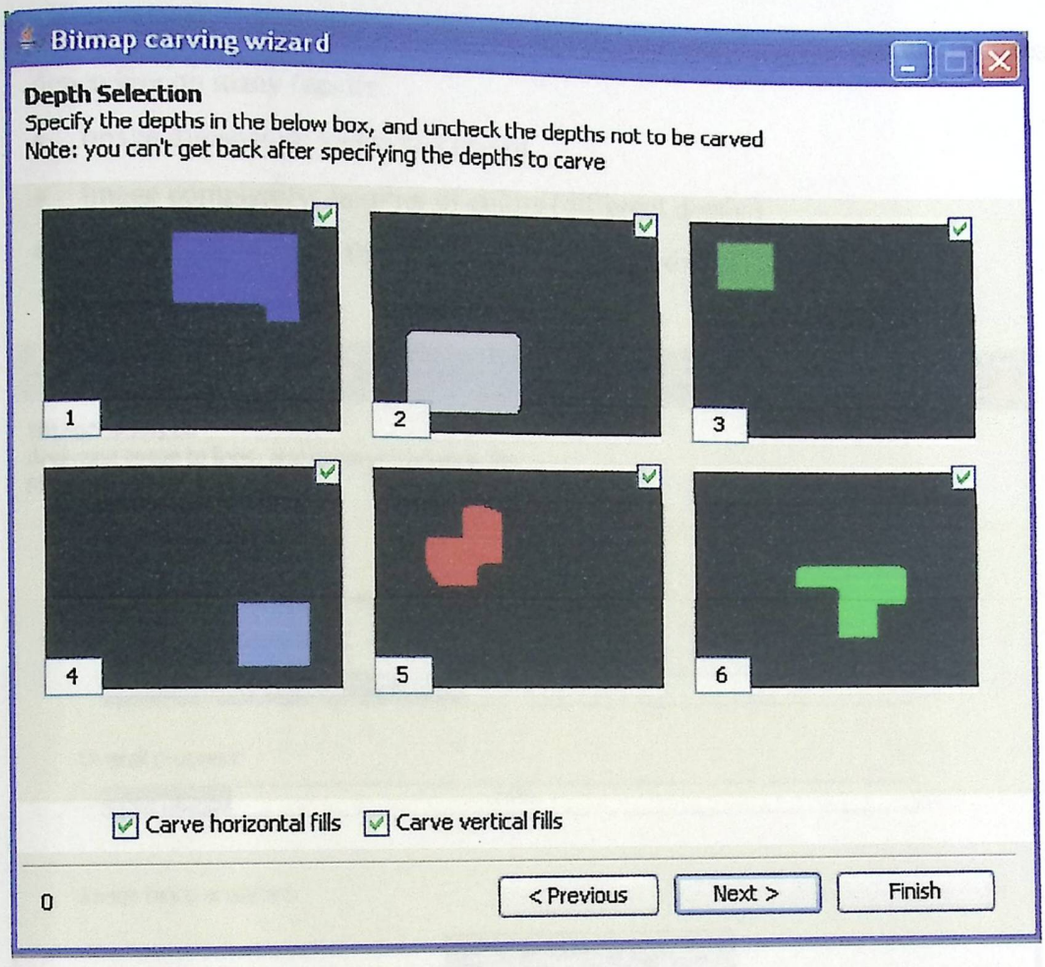


Figure 5.4: Wizard screen 2 (Colored images)

from figure 5.4 its clear that each color could be assigned to a specific depth between 1-30 mm – multiple colors can have same depth values – also the user is able to select whether the specific color will be carved or not, and if the machine will carve horizontal or vertical fills or both.

c) Third screen: which is image analyze screen, figure 5.5 shows this screen, here edges of each image are detected, and then its fill lines, as seen in the figure two progress bars are used, the first shows the progress of finding the edges and filling lines for the current image, and the second shows the overall progress for

all the images, analyzing the images in this part may require several minutes depending on many factors:

- Image dimension: width and height
- Image complexity: number of colors (different depths)
- Image inner shapes: filled or just strokes, and size of the shape

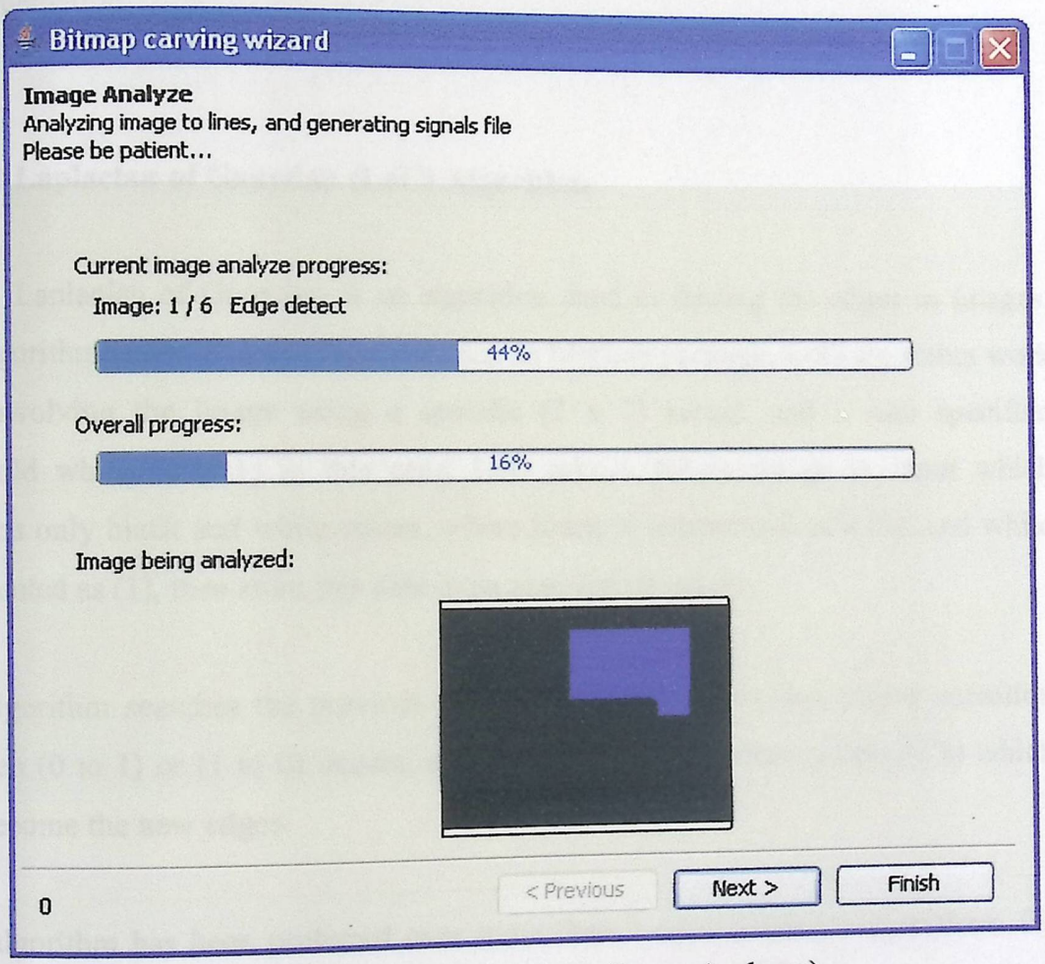


Figure 5.5: Wizard screen 3 (Image Analyzer)

5.1.4 Image Analyzer Algorithms

Multiple algorithms have been used through the software program of the Wood Carving System, only one of these algorithms is an outsider, which is from MatLab package specifically, while the rest of algorithms are our own design and implementation.

5.1.4.1 Laplacian of Gaussian (LoG) Algorithm

Laplacian of Gaussian is an algorithm used in finding the edges in images; this algorithm is part of the Image ToolBox in MatLab package, LoG algorithm work by convolving the image using a specific (7 x 7) kernel, and a user specified threshold which is (0.1) in this case, LoG take a binary image as input which contains only black and white colors, where black is represented as a (0), and white represented as (1), then store this data in an appropriate matrix.

The algorithm searches the previous matrix to find the areas that where transition between (0 to 1) or (1 to 0) occurs, and then stores these areas as new (1's) which will become the new edges.

LoG algorithm has been preferred over more than 7 edge detection algorithms, for many reasons which are:

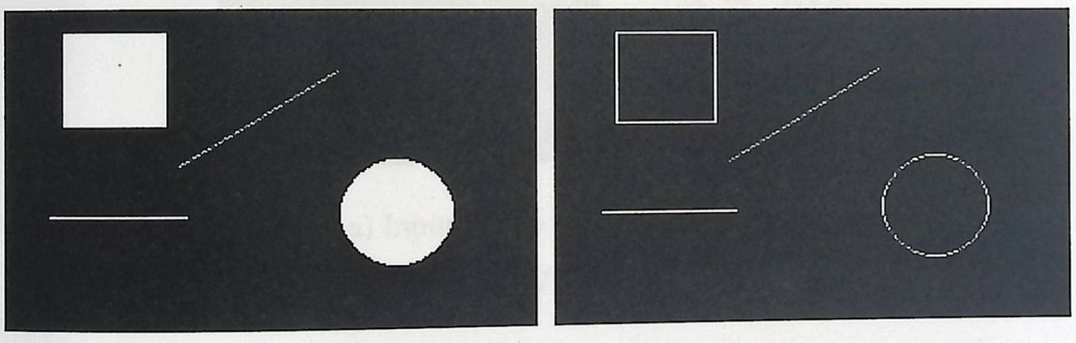
1- Find edges around filled areas with no errors
2- Leave lines as they were in the original image
3- Find the edges exactly, where some algorithms don't find all the edges
4- Works on binary images, which is easily produced by the *ImageAnalyze* subsystem.

5- Very fast edge detection, less than 1 ms to find the edges in an image of size (2000*1000 pixels).

A comparison between many algorithms have been made, like Roberts-Cross, Canny, Rothwell, Sobel, Zero-Crossing and others, using MatLabs Image Toolbox module.

In the result neither of the previous algorithms found the edges properly where each of them had its own defects.

Figure 5.6 shows an image before and after LoG algorithm is applied, its clear in that LoG algorithm is very useful in finding edges around filled areas, while leaving the stroked areas unaffected.



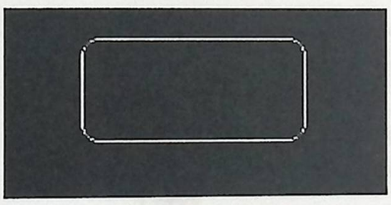
a) b)

Figure 5.6 LoG algorithm in action, a) before LoG b) after LoG

5.1.4.2 Line fitting algorithm (edges to lines)

In order to generate an appropriate control signals file, an array of lines should be generated first, where this array contains the lines coordinates, and if these coordinates are stored as MOVE_TO, LINE_TO commands, see section ***** for more details about these commands.

Using MatLab in finding the edges is very useful, and the Laplacian of Gaussian algorithm gives the expected results, but after finding the edges the image is still a binary image, and the lines coordinates are unknown too, so a new algorithm has been designed, known as *edgesToLines* algorithm, figure 5.7 shows the main idea of this new algorithm.



```
MOVE_TO  54, 40
LINE_TO  147,40
LINE_TO  150,42
LINE_TO  153,46
LINE_TO  153,82
.........
LINE_TO  53,41
```

a) Input image b) Output array

Figure 5.7: *edgesToLines* Algorithm

From the figure it's clear that the algorithm takes an image as input and produces an array of drawing commands.

The idea of the algorithm is to find a starting point, the starting point is the first point from the upper left corner in the image that is not BLACK (background color), then starting from this point the algorithm looks in the 8 surrounding points, based on a specific priority, figure 5.8 shows the 8 surrounding points and the priority of each point, where lower points mean higher priorities. From the figure notice that the algorithm works on a clock wise manner where the right point and bottom points have the highest priorities.

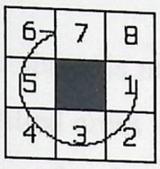


Figure 5.8: 8 Areas Around the Point

When finding an adjacent point the algorithm looks at the previous stored point, if both points are at the same horizontal, vertical or diagonal lines then look for the next point without storing the current point, else if the next point found is not on the same line, then store the current point as the end of the line, and the next point as the starting point of a new line, figure 5.9 shows the pseudo-code.

```
edgesToLines  (input as Image)
    start
        currentPoint=findFirstPoint
        allPoints=countAllPoints
        Loop
                Loop while (area1 is white)
                    else  endPoint=currentPoint
                Loop while (area2 is white)
                    else  endPoint=currentPoint
                Loop while (area3 is white)
                    else  endPoint=currentPoint
                ....
                ....
                allPoints=allPoints-1
                if allPoints equal 0 then done
    end
```

Figure 5.9: *edgesToLines* algorithm Pseudo-code

The result of applying this algorithm is the same input image, but this time represented as an array of lines, which is understandable for the *signalGenerator* algorithm.

5.1.4.3 Fills finding algorithms (*horizontalLinesAnalyzer* + *verticalLinesAnalyzer*)

Finding the edges around the shape is not enough, since some of the shapes could be filled, in order to find the filled areas a small process must be done, figure 5.10 shows how to find the filled areas after finding the edges.

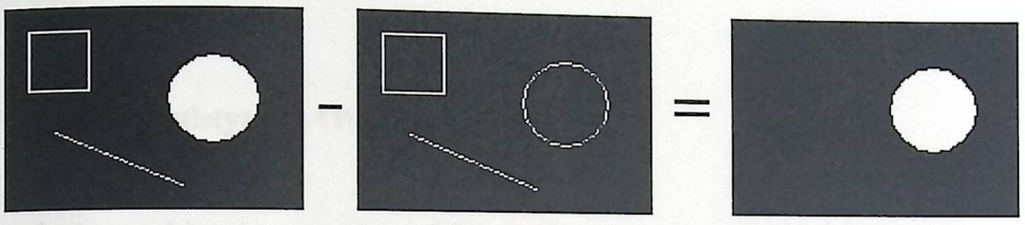


Figure 5.10: Finding filled Areas

The previous figure show the small operation for finding the image filled areas, the idea is simply to subtract the (edges image) from the (original image), which will result in a new image that contains the filled areas.

Finding the filled areas image is just the beginning for processing filled images, as mentioned in edge detection algorithm the result is still an image, where the *signalGenarator* algorithm needs lines array to generate the signals.

The algorithm starts from the upper left corner looking for the first WHITE pixel (white pixels mean foreground points), then search horizontally until a BLACK point is found which means the current line is done, if the image boundaries are reached before a BLACK point is found then the last point of the line will be the same as image width, next just increment the Y coordinate by one and search again, but this time from right, and that's to simplify the work on the machine motors.

5.2 Drawing Subsystem

A utility program that allows the user to open, draw, edit and save images to be carved later on the wood, this program is tailored for drawing images that will be carved based on the user requirements and specifications.

5.2.1 Drawing Subsystem Objectives

In the design and implementation of the drawing program the following points where taken in consideration:

- Simple, user familiar, consistent Graphical User Interface (GUI).
- User manual that describes how different operations could be done using the program.
- Enable the user to do the following:
  - Create simple and complex two dimensional shapes.
  - Open/save images in a special format.
  - Painting image areas, specifying the depth of each area.
  - Customizing the GUI to the meet the user preferences.
  - An easy method for choosing the color depth.

- Use the new drawing technologies, such as layers where the image consists of multiple layers with different Z values.
- Abilities to apply transformations on shapes, rotation, translation, scaling … etc.
- Apply combinations on shapes, such as adding two shapes, subtracting them, intersection and exclusive or
- Placing shapes and sizing them accurately, this is done by supplying rulers and indicators that notify the user the current mouse location.

5.2.2 Drawing Program Implementation

5.2.2.1 Layers Technology

In the drawing program each shape is drawn on a separate layer, and the final image is the result of combining all the layers together. This approach enables the user to do the following:

- Work with each layer individually, where all layers are independent and changes on one layer don't affect other layers.
- Change the layers Z index simply, by doing this it is easy to specify the order of the shapes.
- The user will be able to create new layers from existing layers, for example the user can create a new shape by applying a (subtraction operation) from two layers, or creating a new layer from an existing one (layer via copy).
- With layers the user will be able to draw images and work with them easier since he can hide and show the layers he wants. For example if there is a layer above another layer – so that the back layer can't be see – the user can hide the top layer and then edit the back layer.

From all of the above it's obvious that dealing with shapes as layers in the drawing program will facilitate image manipulation, and thus accomplish one of the main user requirements which are *ease of use*.

Figure 5.11 shows how the layers technology works. You can notice that there is a different shape in each layer, and that the complete image is created by combining all

the layers. Another important note is that changing the Z-index of the layers will result in different images.

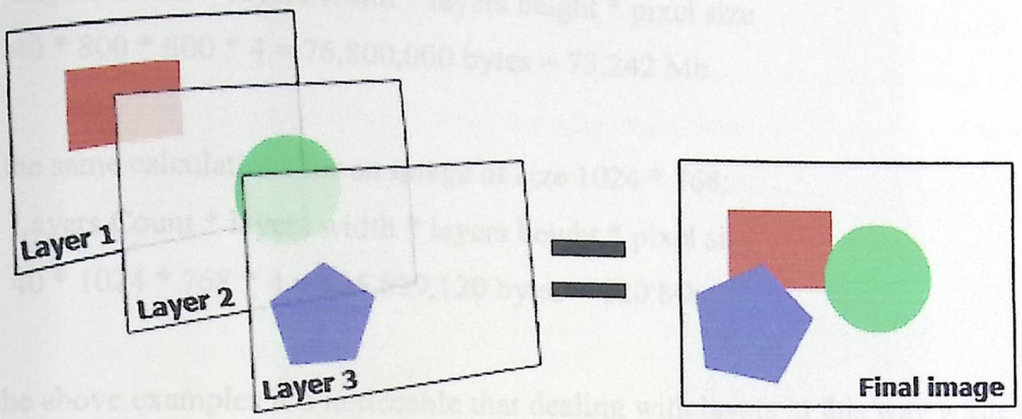


Figure 5.11: How Layers Technology Work

In the first design each layer was implemented as an Image – *BufferedImage* in Java terminology – where each pixel of the image had four attributes Red, Green, Blue and an Alpha value. The size of the layer was set the same as the size of the image. For example if the user had created a new image and specified that its size is (800 * 600) pixels then each layer had a size equal to (800 * 600) pixels.

Using this approach dealing with layers was easy but it leaded to a memory leakage problem as the number of layers increased. The reason behind this is that even if we draw a square of 10*10 pixels size we still have to allocate memory for a *BufferedImage* of size (800*600) pixels. In other words if the image had 40 layers and its size is (800* 600) pixels it would demand:

Pixel consists of: Red (1 byte)
Green (1 bye)
Blue (1 byte)
Alpha (1 byte)

So each pixel requires 4 bytes then the total allocated size for the 20 layers is:

Layers Count * layers width * layers height * pixel size

= 40 * 800 * 600 * 4 = 76,800,000 bytes = 73.242 Mb

Doing the same calculations for an image of size 1024 * 768:

Layers Count * layers width * layers height * pixel size

= 40 * 1024 * 768 * 4 = 125,829,120 bytes = 120 Mb

From the above examples it's noticeable that dealing with layers in this way wastes a lot of memory. So a new approach was used to store layers but this time instead of representing layers as a BufferedImage of a same size equal to the image size, the actual shape will be dealt with so that we store the information of the shape itself instead of a two dimensional array of pixels (Raster).

5.2.2.2 Representing Layers as Shapes

As mentioned in the previous section representing layers as BufferedImages had crucial memory drawbacks, a solution to this problem is representing each layer as a shape, where the shape is recognized by groups of commands that describes its outline – border – and fill. Using this approach it has the following advantages:

- Memory leakage problem is resolved; since each layer is stored by its mathematical representation – named GeneralPath described in the next section – as a substitute of storing all of its pixels.
- Shapes are represented in mathematical representation instead of raster of pixels, this will first save space and second it will be easier to analyze images in this way, for a reason mentioned later.

- Transformations (rotation, scaling, and translation) can be applied easier and faster to shapes than *BufferedImages*; because the top left point of the shape is know in the shape, while in the BufferedImage the top left point is unknown, as shown in figure 5.12, so before applying a transformation on a buffered image its top left point must be found which will require additional processing time, increasing the transformation needed time.

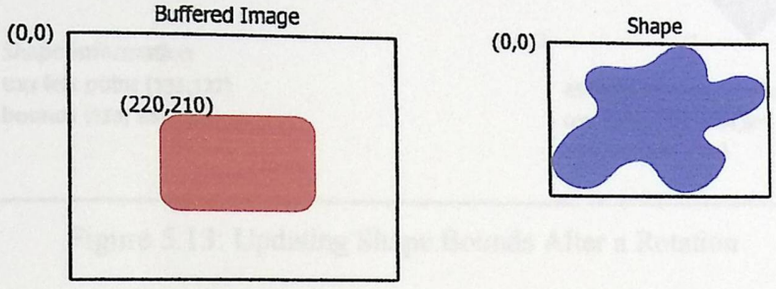


Figure 5.12: Top Left Point in Shapes and in BufferedImages

The only disadvantage of this method is that it's more difficult to implement, because the shape bounds may change (i.e. when rotating the shape its bounds will change). So after applying transformations on shapes its new bounds must be updated as shown in figure 5.13.

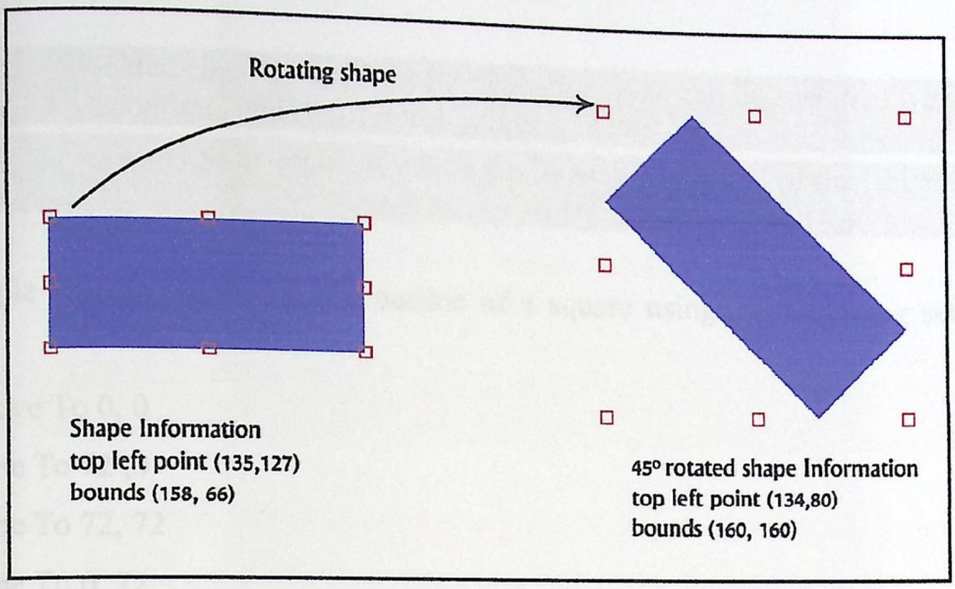


Figure 5.13: Updating Shape Bounds After a Rotation

5.2.2.3 General Path

Each shape is represented by an object of type *GeneralPath*. The general path has a *PathIterator* object used to describe the border of the shape, the border is represented using a collection of segments (instructions), and each segment can be one of the segments shown in table 5.1.

Table 5.1 *PathIterator* Segments

Segment name	Description
MOVE TO	This segment type is used to update the location of the path without drawing anything.
LINE TO	This segment type is a straight line, drawn from the last point in the path.
QUAD TO	This segment type is a curved line that is represented by a quadratic (second-order) equation. The segment is fully described by two endpoints and a control point, which determines the curve's tangents at its endpoints.
QUBIC TO	This segment type represents a Bezier cubic curve. Basically a quadratic curve with an additional control point; is mathematics this

	is described by a third-order equation. And can be specified with two endpoints and two control points.
CLOSE	This type of segment draws a line back to the end of the last MOVE TO

For example you can represent the outline of a square using the following set of segments

1. Move To 0, 0
2. Line To 72 ,0
3. Line To 72, 72
4. Line To 0, 72
5. Close

The *PathIterator* will be very helpful in analyzing the image since its segments will be used to describe how the carving motors should move in order to carve the shape. In other words if we want to carve the square defined above we will follow the segments of its *PathIterator* and move the carving head accordingly.

5.2.2.4 Layer Representation

Each layer used in the drawing program is represented by a class named *layer*; this class contains the members shown in table 5.2.

Table 5.2: Members of *layer* Class

Member Name	Description
Point2D point	The top left point of the shape with respect to the complete image.
String name	Layer name, used to identify the layer and give meaningful names for each layer.
int zIndex	The z index of the layer, used for ordering the layer on top of each others.

in our case only bg1 is repainted while the user is dragging and bg2 which contains the 20 layers remains the same, and so saving a lot of processing time. Note that only the final shape will be drawn on bg2 which occurs when the user releases the mouse button.

For more explanation about how the two backgrounds works, the next section describes a step by step approach of how the rectangle tool operates, starting from a mouse click and ending when the user releases the mouse button.

3. While the mouse is dragged the *releasePoint* and the *pressPoint* are used to draw the rectangle on bg1 where the rectangle can be drawn in four different methods. This is done by sending both points to the method *setShapeParameters*() which will return an instance of the a rectangle shape.

4. After obtaining the rectangle the program should know whether to draw only the outline of the rectangle or to fill the rectangle. This is done by checking a Boolean variable named *drawOptions*.

5. The final shape of the rectangle is obtained only when the user releases the mouse button – drag operation is terminated – which will create a new instance of the *layer* class with the following information

    a. Selected shape: which is the rectangle drawn by the user.
    b. Shape Location: a rectangle specifying the upper left point of the rectangle.
    c. The Z index of the shape which is used also as a counter for the shape.
    d. Shape name: a temporary name for the shape which could be used to identify the shape later. The user can change the name after the shape is drawn.
    e. Type: used to specify the type of the shape whether it's a rectangle, ellipse, general path…etc.
    f. Fill: a Boolean value specifying whether it's an outlined or a filled shape.
    g. Paint: the color of the shape.
    h. Stroke: information related to the shape outline.

6. After obtaining the final shape of the rectangle, it is drawn on *bg2*, and stored in the shapes linked list, and the whole panel is repainted, figure 5.15 shows the flow chart of this process.

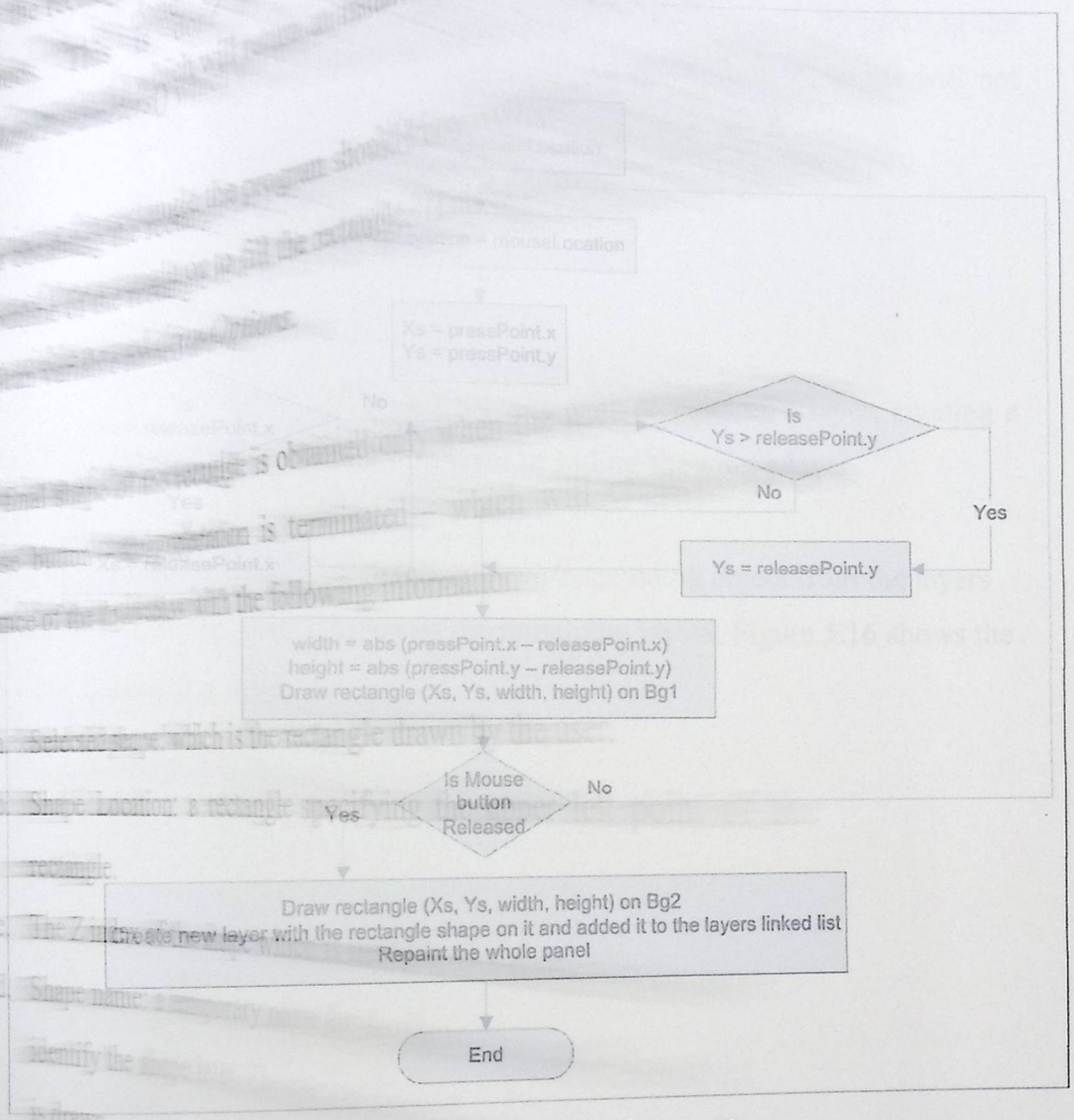


Figure 5.15: Draw Rectangle Flow Chart

From the above steps it's noticeable that bg2 was updated only when the final rectangle shape was obtained, and all the visual simulation was done on bg1, again this was done for saving time and not repainting all layers during the whole dragging operation.

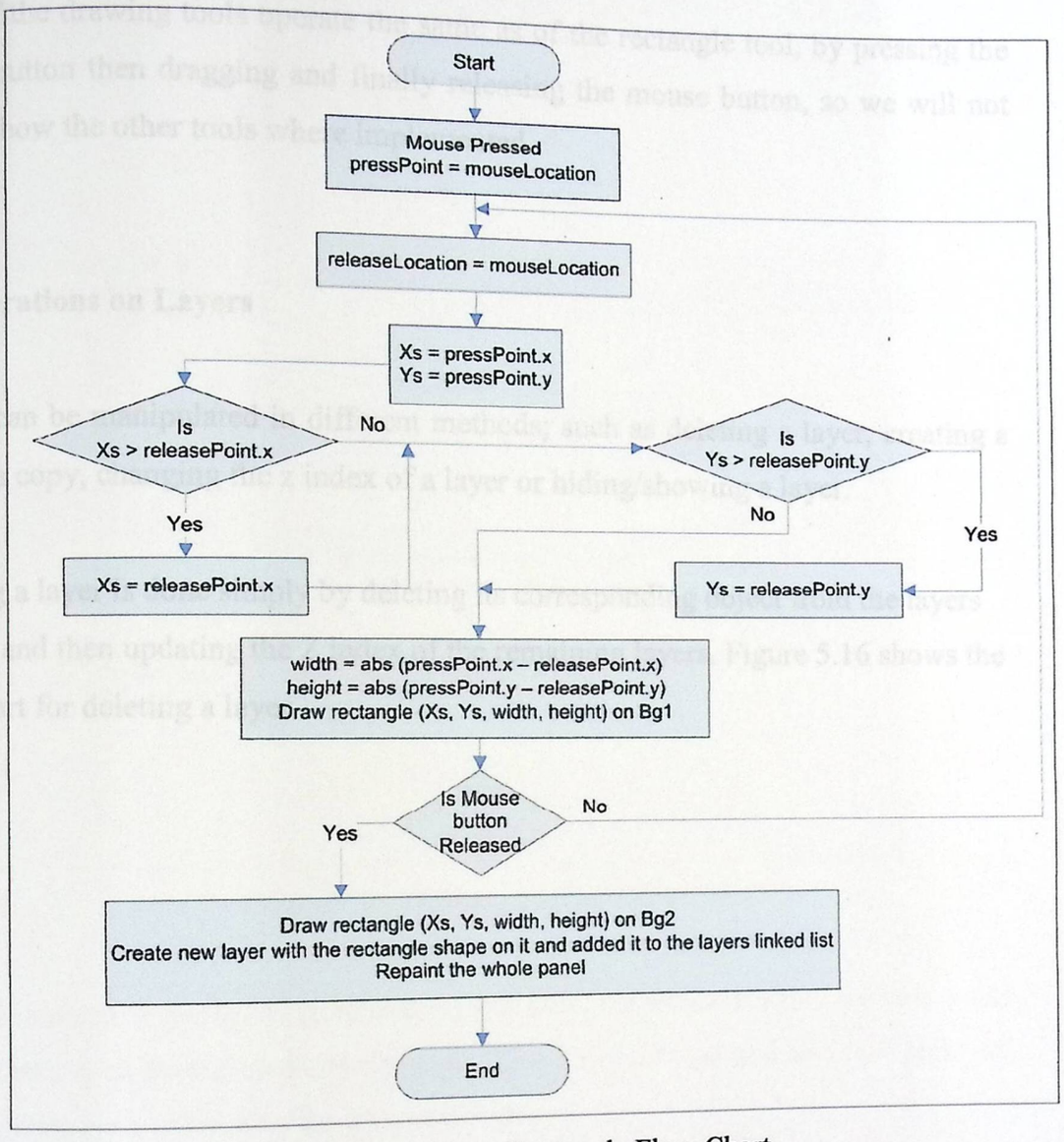


Figure 5.15: Draw Rectangle Flow Chart

From the above steps it's noticeable that bg2 was updated only when the final rectangle shape was obtained, and all the visual simulation was done on bg1, again this was done for saving time and not repainting all layers during the whole dragging operation.

Most of the drawing tools operate the same as of the rectangle tool, by pressing the mouse button then dragging and finally releasing the mouse button, so we will not include how the other tools where implemented.

5.3 Operations on Layers

Layers can be manipulated in different methods; such as deleting a layer, creating a layer via copy, changing the z index of a layer or hiding/showing a layer.

Deleting a layer is done simply by deleting its corresponding object from the layers link list and then updating the Z index of the remaining layers. Figure 5.16 shows the flow chart for deleting a layer.

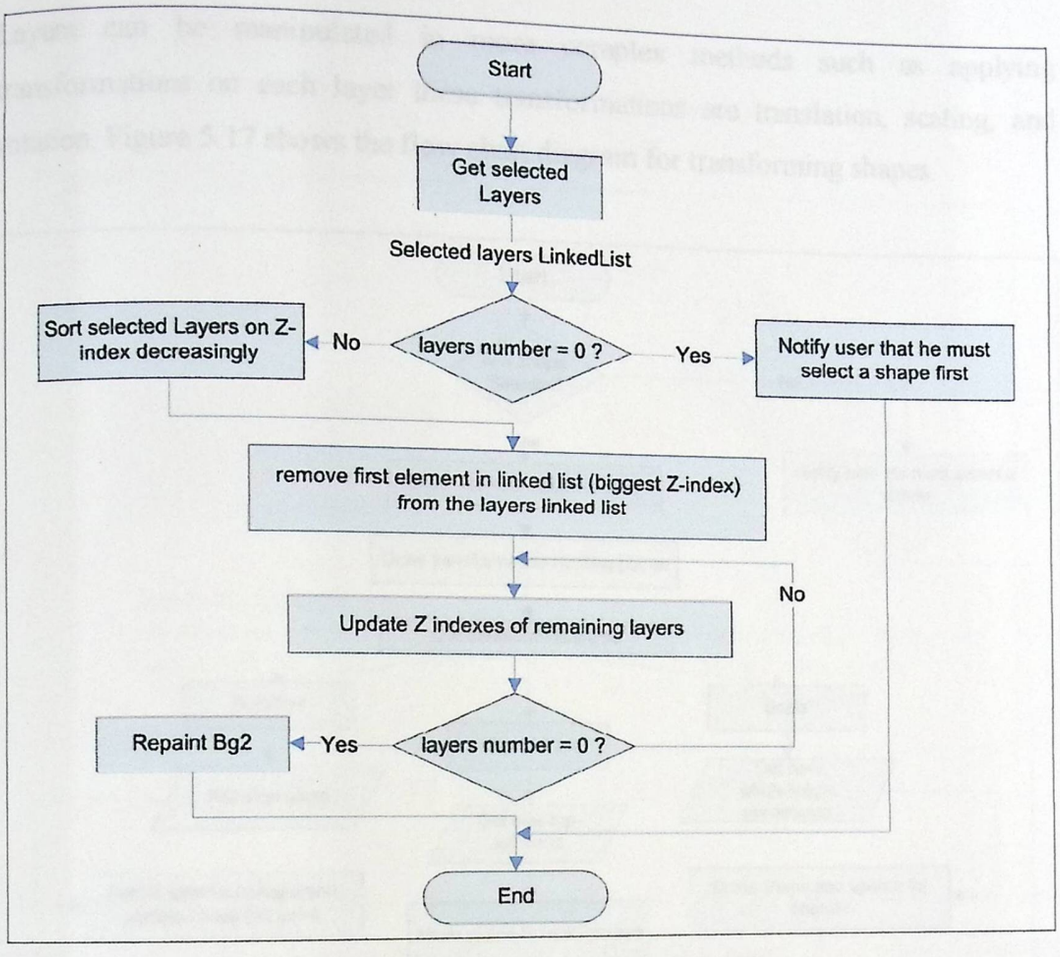


Figure 5.16: Deleting a Layer Flow Chart

When deleting multiple layers at the same time, the selected layers are first sorted depending on their Z index decreasingly, then a loop is used to delete each layer and updating the z-index after the layer is deleted.

When a new layer is created a *layer* object is initialized with all of its parameters, then it is added to the layers linked list, finally Bg2 is repainted, the same thing is done when a new layer is created from an existing layer where a new layer object is created with its parameters equal to the selected layer and then added to the layers linked list.

Layers can be manipulated in more complex methods such as applying transformations on each layer these transformations are translation, scaling, and rotation. Figure 5.17 shows the flow chart diagram for transforming shapes.

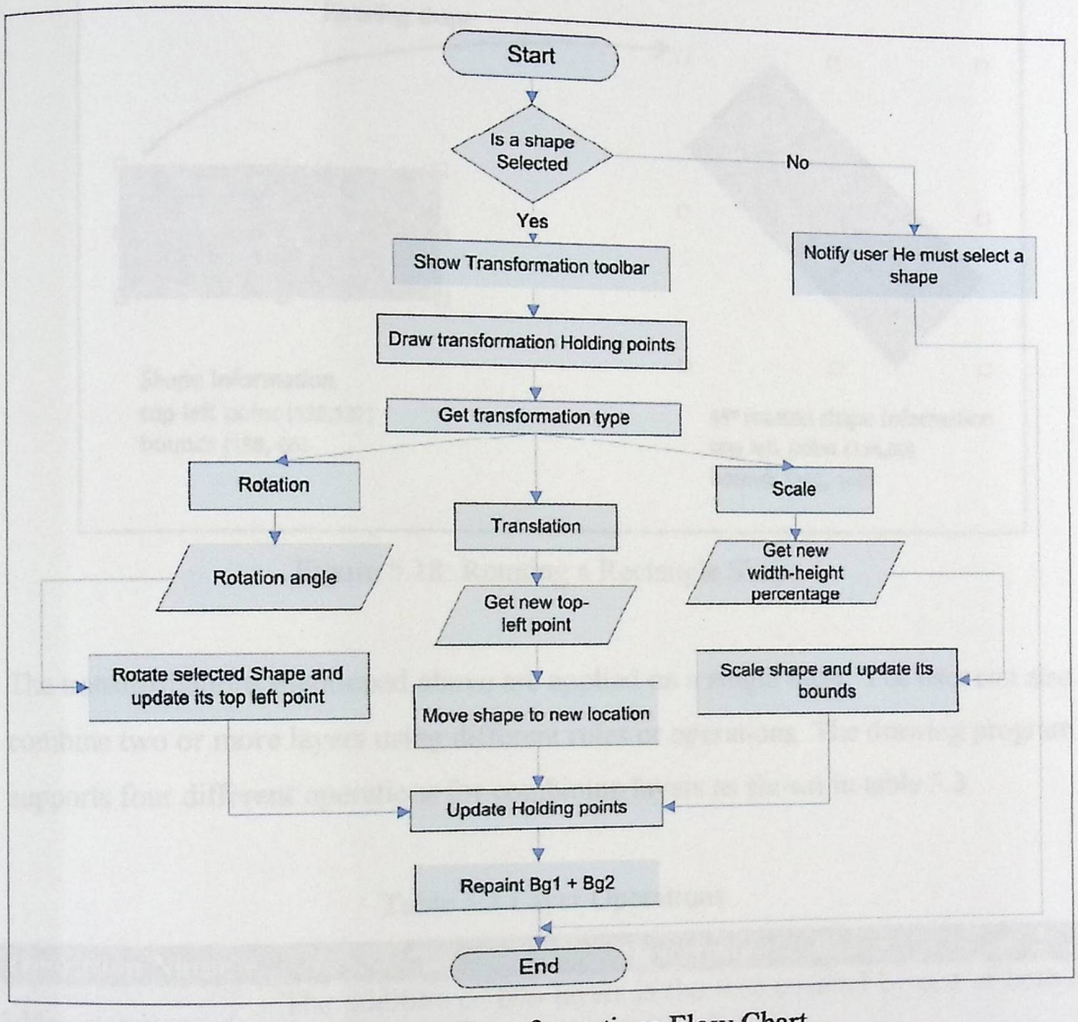


Figure 5.17: Transformations Flow Chart

At the beginning the program identifies which layer is selected, and retrieves the *shape* then applies the desired transformation on its *GeneralPath*, then the bounds of the layer (i.e. the top left point and size) are updated depending on the transformation to fit the new shape, finally the old *GeneralPath* is replaced by the new transformed

one and bg2 is repainted. Figure 5.18 shows how a rotation transformation is applied on a rectangle and how the top left point is updated if needed.

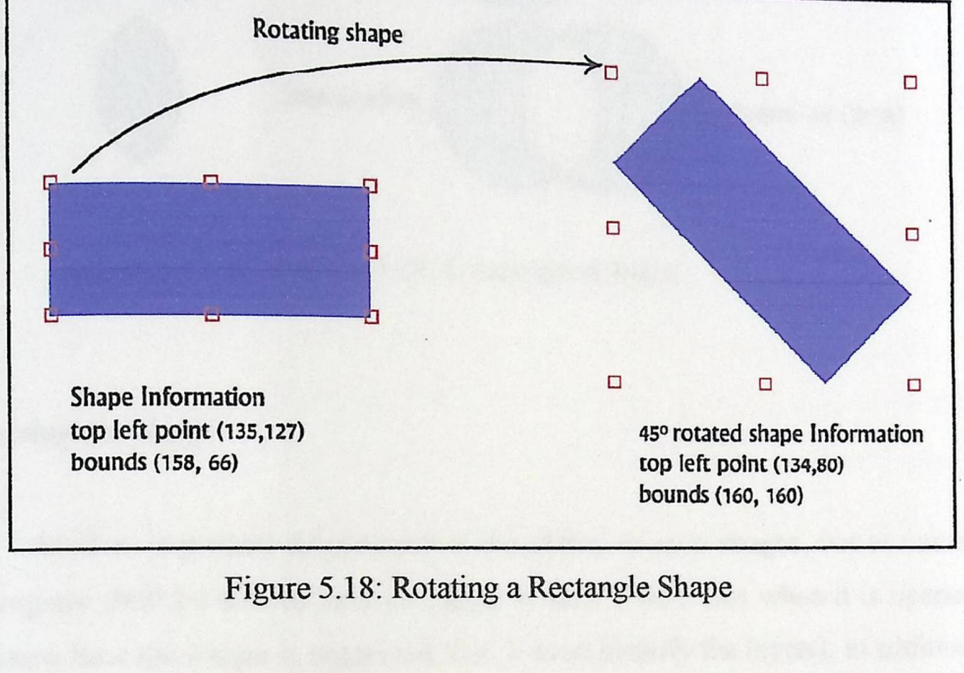


Figure 5.18: Rotating a Rectangle Shape

The transformations mentioned above are applied on a single layer. The user can also combine two or more layers using different rules or operations. The drawing program supports four different operations for combining layers as shown in table 5.3.

Table 5.3 Layer Operations

Operation Name	Description
Addition (union)	The addition of two layers is the area covered by one or both of the layers.
Intersection	The intersection of two layers is the area that is covered by both layers simultaneously.
Subtraction	The result of subtracting one layer from another is the area covered by one that is not covered by the other.
Exclusive or	The exclusive or operator is the inverse

Figure 5.19 shows how the four combination rules are applied on two shapes

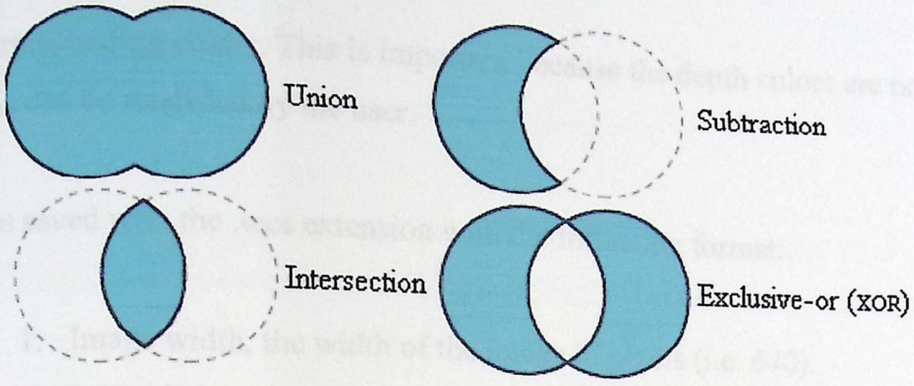


Figure 5.19: Composition Rules

5.4 Saving an Image

Another important requirement is the ability to save images, but in our case the program shall be able to save an image in such a way, that when it is opened it will know how the image is organized, (i.e. it must identify the layers), in addition to the environment which is the depth of each color in the image.

For example if the user had specified that the RED color represents a 1mm depth, and saved the image. When reopening the image the program must still recognize that the RED color is bound to the depth of 1mm. for saving an image the following points must be considered:

- The program must remember all the layers and their order.
- The program must remember the state of the layers, whether they are visible or not.
- The program must remember the user color depth mapping configuration, so that when re-opening the image it will still remember the depths and their

corresponding colors. This is important because the depth colors are not fixed and can be modified by the user.

Images are saved with the .wcs extension with the following format:

1. Image width, the width of the image in pixels (i.e. 640).
2. Image height, the height of the image in pixels (i.e. 480).
3. Number of colors used in the image, maximum 30 colors.
4. In each line each color represented by its RGB value and its corresponding depth in mm's (i.e. 255 0 0 1), which represents the R = 255, G = 0, B = 0, and a depth of 1mm. note that if the number of colors used is 30 then there must be 30 lines defining the colors and their depths.
5. Layer definition, where each layer is stored in a separate line. Where each line consists of all the layer members shown in table 5.2.

5.4.1 Saving Layer Members

Storing a layer needs storing all the information related to the *layer* class mentioned in table 5.2; all these members are stored in one line separated by comas in the following order.

1. Top left point x value.
2. Top left point y value.
3. Z index of the layer.
4. Fill member where true = filled shape, and false = outlined only.
5. Name of the layer.
6. Red value of the Color of the shape (R value).

7. Green value of the Color of the shape (G value).
8. Blue value of the Color of the shape (B value).

After these attributes we need to store the information related to the stroke of the shape (the *BasicStroke* member) which is represented by the following information:

1. Border width.
2. Type of the end cap of the stroke. There are three different styles for the end cap of the line (Butt, Round, and Square) as shown in image 5.20.

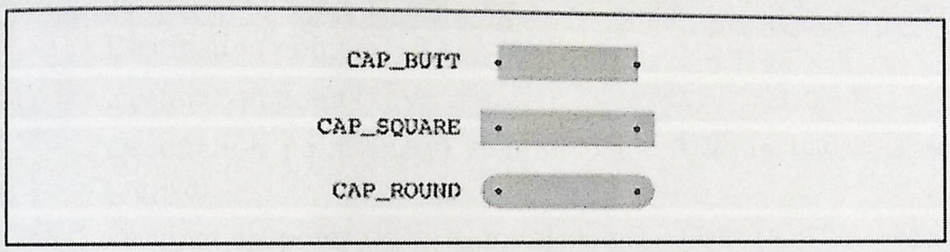


Figure 5.20: End Styles (Butt, Round, and Square)

3. Type of how the lines should be joined. There are three different styles for joining the shape lines (Bevel, Miter, and Round) as shown in figure 5.21.

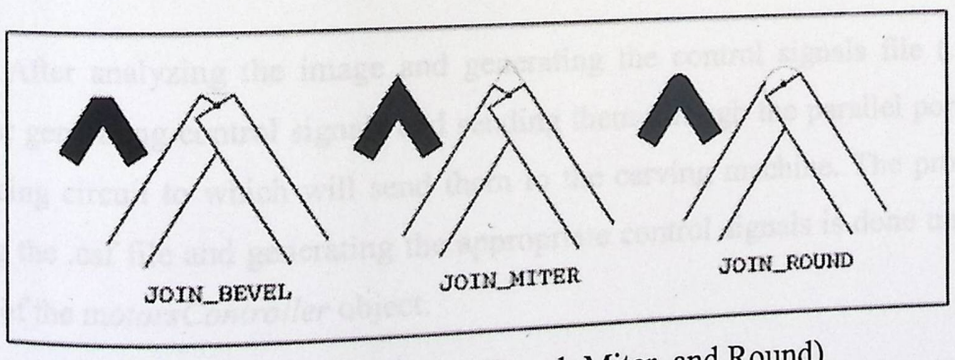


Figure 5.21: Join Styles (Bevel, Miter, and Round)

4. If the stroke is a dashed line then we store the length of the dashes in the stroke or zero if it's not a dashed outline.

5. The distance between dashes if the outline is dashed or 0 if it's not a dashed outline.

After storing the stroke of the layer, we store the GeneralPath of the layer depending on its PathIterator that describes its outline. Information is stored by iterating through all the segments and storing the segment type and its needed information as shown in table 5.4.

Table 5.4: Segment Needed Information

Segment Type	Needed information	Example
Move To	Destination point (x, y)	Move to, 10, 200
Line To	Destination point (x, y)	Line To, 100, 50
Quad To	Destination point (x, y) Control point (xc, yc).	Quad To, 100, 12, 13, 14
Cubic To	Destination point (x, y) Control point1 (xc1, yc1), control point2 (xc2, yc2)	Cubic To, 100, 12, 50, 45, 70, 80
Close	No needed information	Close

5.5 Generating the Control Signals

After analyzing the image and generating the control signals file (.csf) its time for generating control signals and sending them through the parallel port to the interfacing circuit to which will send them to the carving machine. The process of reading the .csf file and generating the appropriate control signals is done under the behalf of the m*otorsController* object.

5.5.1 Motors Controller Overview

The m*otorsController* class is responsible for reading .csf files and generating the needed control signals. This process runs in a separate thread of execution in order to increase the responsiveness of the GUI program while processing the carving operation.

The *motorsController* process demands a lot of processing time; since the control signals must be send at high frequencies, so if it doesn't run in a separate thread all the program will be blocked, and stop responding to user interaction. But by running it in its own thread a user can for example edit other images while a carving operation is being processed.

Figure 5.22 shows the flow chart for this thread execution, it starts by reading the first command from the .csf file which is one of the commands shown in table 5.5, decodes it, executes it, and then reads the next command from the file and so on until the end of file is reached.

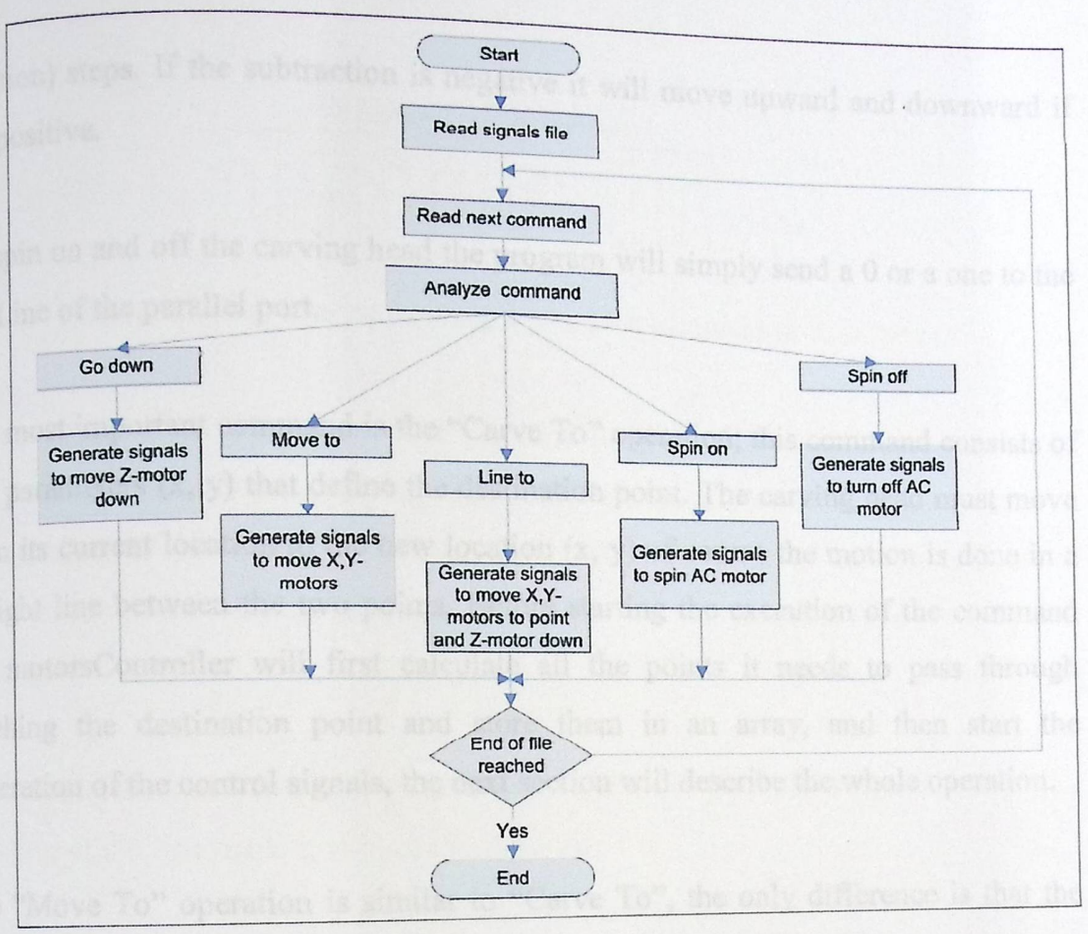


Figure 5.22: motorsController Flow Chart Diagram

Table 5.5: Control Signal File Commands and Their Execution.

Command type	Syntax	Execution
Move to depth	+depth	Move the Z motor to the corresponding depth value.
Move To	0 x y	Initialize the Z motor, and then move to the specified location by (x, y)
Carve To	1 x y	Move to the specified location by (x, y)
Turn On Head Motor	R	Rotate the carving head
Turn off Head Motor	S	Stop the Rotation of the carving head

The execution of the +depth command is quiet simple; it will calculate the current location of the Z motor which is stored in a variable and then move (depth – Z

location) steps. If the subtraction is negative it will move upward and downward if it's positive.

To spin on and off the carving head the program will simply send a 0 or a one to the D3 Line of the parallel port.

The most important command is the "Carve To" operation; this command consists of two parameters (x, y) that define the destination point. The carving head must move from its current location to the new location (x, y) of course the motion is done in a straight line between the two points. Before starting the execution of the command the motorsController will first calculate all the points it needs to pass through reaching the destination point and store them in an array, and then start the generation of the control signals, the next section will describe the whole operation.

The "Move To" operation is similar to "Carve To", the only difference is that the carving head must be raised up completely – so that the machine will not carve – and then move it to the destination point. This command is used to move from one point to another without carving.

5.5.2 Controlling the Carving Head Motion

As mentioned in chapter three the carving head position is controlled by three servo motors, and the minimum resolution of the machine is 1mm – the minimum displacement possible in the machine – for example moving the carving head 1 mm in the X direction requires ten pulses sent to the X servo motor driver as shown in figure 5.23.

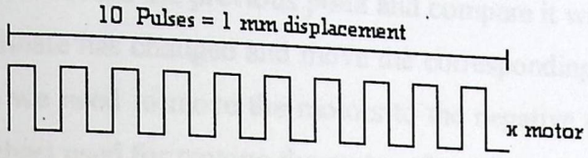


Figure 5.23 Signals for Moving 1 mm Displacement in X Direction

As mentioned in the previous section the control signals file contains commands and instructions for moving the carving head in straight lines – Move To and Carve To –. For example if the current command is (1 255 141) and the carving head is at location (0, 0). The carving head will have to carve from the point (0, 0) to the point (255, 141).

For moving the head through two points (x1, y1) and (x2, y2) we need to find the equation of the line passing through both points.

$$y - y1 = m * (x - x1) \ldots\ldots\ldots \text{ where m is the slope } (\Delta x / \Delta y)$$

Using the line equation we will find the path between the two points and store all the points in an array. While moving through the line and supposing that the slope is positive and $x_1 < x_2$, the next point after approximation can be one of the following:

- $(x_1 , y_1 + 1)$
- $(x_1 + 1 , y_1)$
- $(x_1 + 1 , y_1 + 1)$

From those points the possible movements of the x, y motors are as follows:

- Move y motor one step to the positive direction.
- Move x motor one step to the positive direction.
- Move x, y motors one step to the positive direction.

So the software will record the previous point and compare it with the new point, and find which coordinate has changed and move the corresponding motors. The same is done in the case we need to move the motors to the negative direction. Figure 5.24 shows the flow chart used for moving the motors through a line.

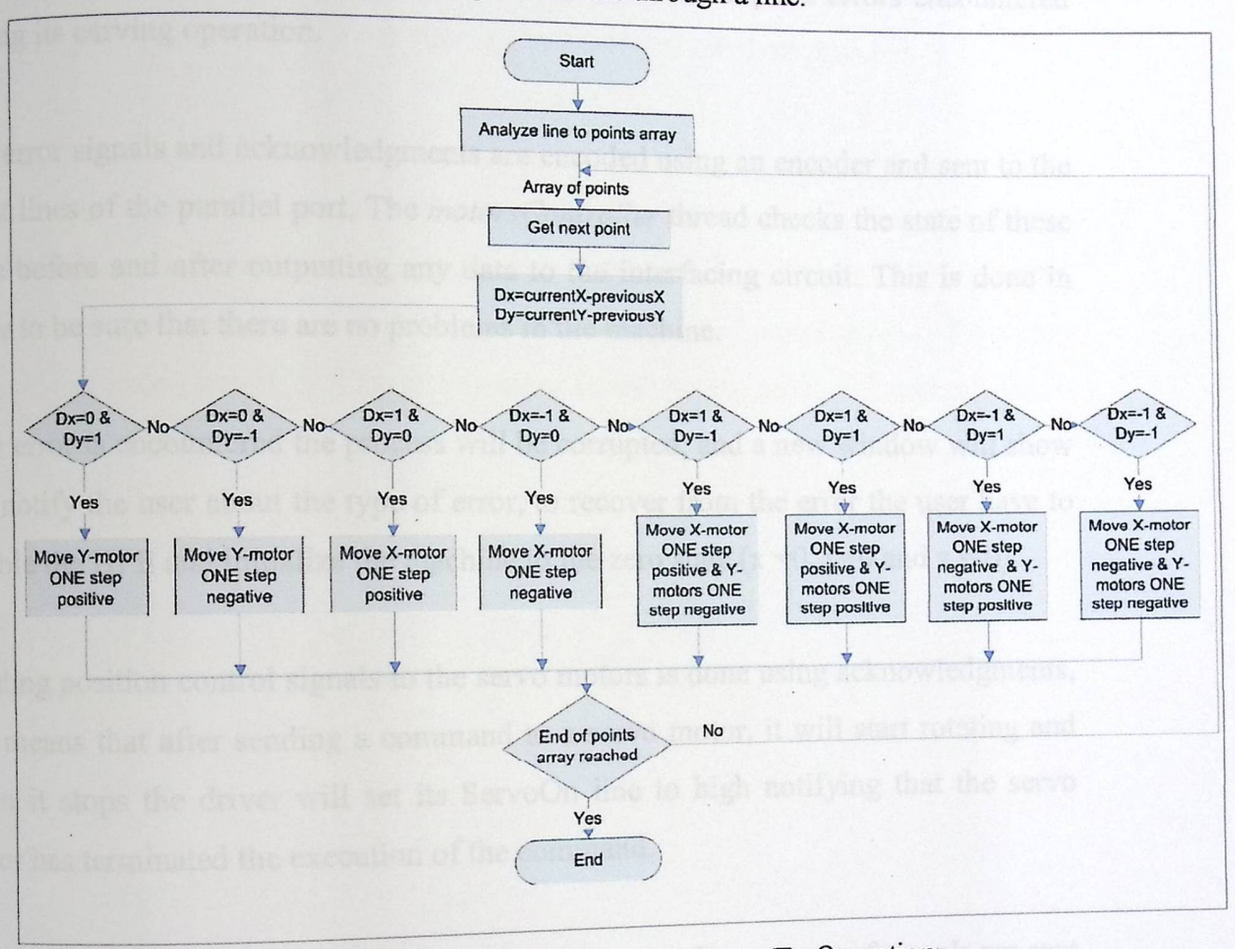


Figure 5.24: Flow Chart For Executing a Carve To Operation

5.5.3 Feedback Signals

As mentioned in chapter 4. The machine sends back to the computer a group of signals notifying the current state of the machine and reports errors encountered during its carving operation.

The error signals and acknowledgments are encoded using an encoder and sent to the input lines of the parallel port. The *motorsController* thread checks the state of these lines before and after outputting any data to the interfacing circuit. This is done in order to be sure that there are no problems in the machine.

If an error is encountered the process will be corrupted, and a new window will show and notify the user about the type of error, to recover from the error the user have to disable the HPE and initialize the machine to the zero state (x =0, y=0 and z = 0).

Sending position control signals to the servo motors is done using acknowledgments, this means that after sending a command to a servo motor, it will start rotating and when it stops the driver will set its ServoOn line to high notifying that the servo motor has terminated the execution of the command.

These signals are sent as a feedback to the computer. After a group of signals are sent to the servo the motorsController will suspend its execution and wait for an acknowledgment notifying it that the servo had executed the command.

Of course the motorsController will suspend its execution for a limited time, and if no acknowledgments are received during this time, it will notify the user that an external error occurred.

5.5.4 Increasing Performance

Since control signals are sent from the computer to the carving machine at high frequencies, performance of the algorithms used in analyzing the commands must be considered and efficiency must be increased as much as possible.

Finding the path of the line is the most time consuming operation, an increase in performance could be achieved using Bresenham's line algorithm instead of using the direct line algorithm. The algorithm flow chart is shown in figure 5.25, table 5.6 shows a comparison between the time needed to calculate the path points of arbitrary lines of maximum length of 2000 pixels using Bresenham's and the direct method.

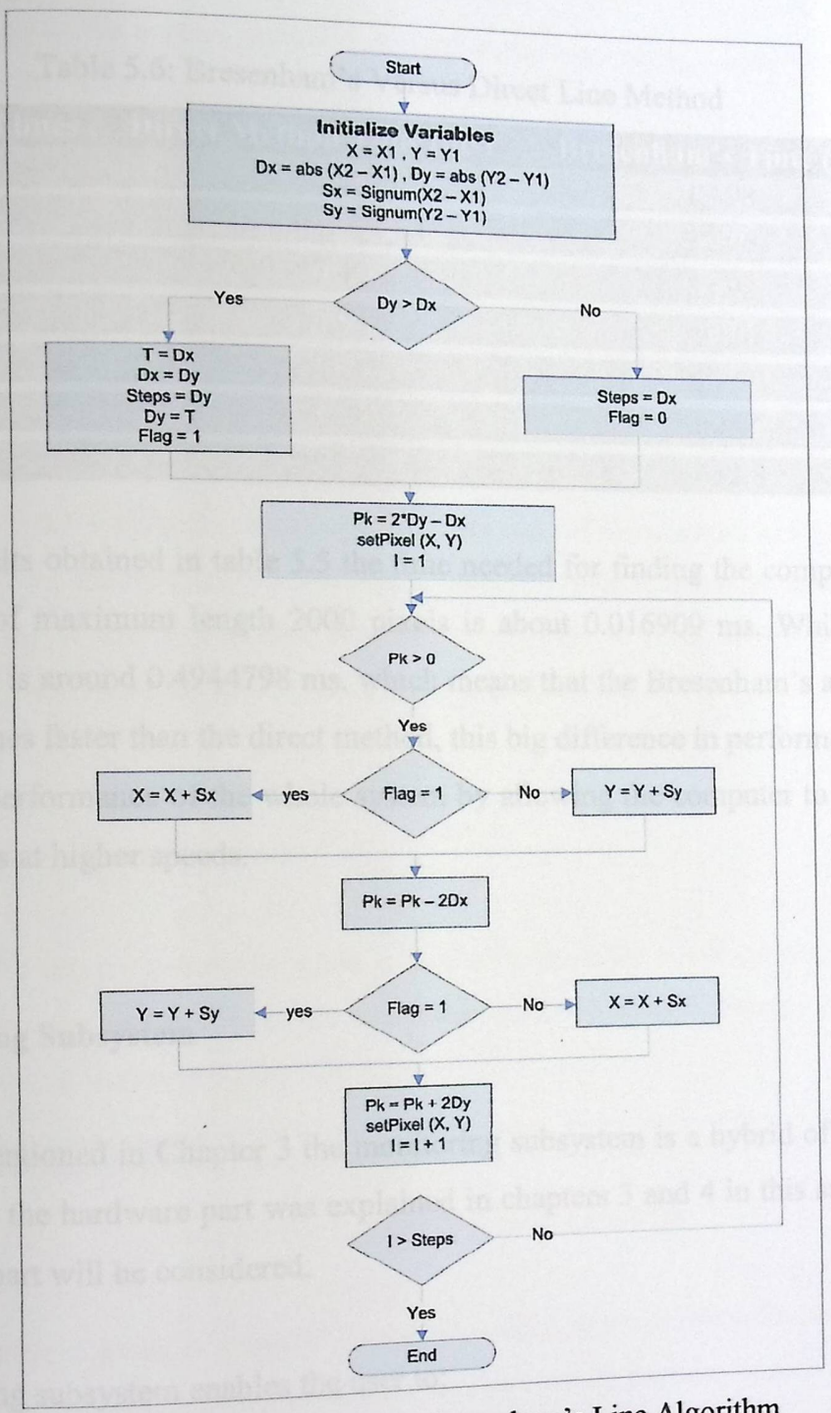


Figure 5.25: Flow Chart of Bresenham's Line Algorithm

Table 5.6: Bresenham's Versus Direct Line Method

Number of lines	Direct Method Time (ms)	Bresenham's Time (ms)
1000	292.65	17.98
2000	886.23	32.96
5000	2387.49	83.92
10000	5311.74	169.03
20000	11175.37	327.79
30000	19957.91	509.97
100000	49288.9	1682.8

From the results obtained in table 5.5 the time needed for finding the complete path for one line of maximum length 2000 pixels is about 0.016909 ms. While in the direct method is around 0.4944798 ms. which means that the Bresenham's algorithm is 29.2436 times faster than the direct method, this big difference in performance will increase the performance of the whole system by allowing the computer to generate control signals at higher speeds.

5.6 Monitoring Subsystem

As mentioned in Chapter 3 the monitoring subsystem is a hybrid of software and hardware the hardware part was explained in chapters 3 and 4 in this subsection the software part will be considered.

The monitoring subsystem enables the user to:

1. Watch the carving process progress.
2. Alert the user if any errors are encountered during the carving operation, showing type and position of the error.
3. Track the current location of the carving head, showing the X, Y, and Z coordinates.

4. Display the control signals outputted to the interfacing circuit through the parallel port.
5. Indicate the current carving depth and its corresponding color.
6. Grant the user control over the carving process, by starting, pausing, and stopping the carving process.
7. Save the carving progress so that if any errors are encountered the process could be resumed from the moment the interrupt happen.

The monitoring subsystem works along with the *motorsController* thread – explained in section 5.4.1 – reflecting every change in the state of the machine to the user. For example when the motorsController thread outputs signals to move the carving head by one step in the X direction, this will change the x value in the monitoring subsystem, and move the carving head in the simulation too.

5.6.1 Carving Process Progress

The carving process progress is used to indicate the whole carving progress, which gives the user an idea of the estimated remaining progress using a percentage progress bar as shown in figure 5.26.

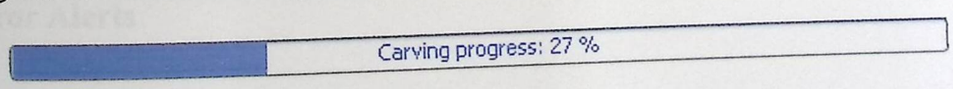


Figure 5.26: Carving Process Progress Bar

The progress is saved in a special format file which stores all the information about the state of the program in case that an interrupt would happen, in case of sudden interrupts like electricity shortage or computer error – caused from outside the carving software environment – the progress file will be used to reset the status of the machine to the same before interrupt, and so the carving process could continue from the interruption point.

The progress is estimated by counting the number of instructions in the .csf file and so setting the progress bar max value to that value, then after the execution of each instruction the progress bar value is updated.

```
BufferedReader n=new BufferedReader(new FileReader(f));
String s;
int lineCount=0;

while((s=n.readLine())!=null)
   lineCount++;

progressBarMax=lineCount;
mainFrame.carvingProgress.setMaximum(lineCount);
```

The update to the progress bar values is done using the following code

```
carvingProgress.setValue((int)(lineCount++ % progressBarMax));
carvingProgress.setString("Carving progress: " + (int)((lineCount*1.0) /
                          progressBarMax *100) +" %");
```

5.6.2 Error Alerts

As errors may be encountered during the carving operation, the monitoring system should be able to report these errors, alert the user about them, specifying the error type, and its location in the machine if possible, all possible errors reported from the machine are listed in table 5.7

Table 5.7: Hardware Machine Reported Errors

Error ID	Error name	Error description
0	EX1	Triggered when the X-motor exceeds the maximum allowed position on its axis
1	EX2	Triggered when the X-motor exceeds the minimum allowed position on its axis

2	EY1	Triggered when the Y-motor exceeds the maximum allowed position on its axis
3	EY2	Triggered when the Y-motor exceeds the minimum allowed position on its axis
4	EZ1	Triggered when the Z-motor exceeds the maximum height allowed on its axis
5	EZ2	Triggered when the Z-motor exceeds the minimum height allowed on its axis
6	NW	Triggered when there is no piece of wood on the machine board

As seen in the previous table, errors are mainly classified into two categories, the motors position errors, and the wood missing error, in case of motors position error, the following procedure will be executed:

1- Indicate the error type, and position.
2- Disable the hardware protection enable (HPE=0).
3- Initialize the machine; reset all motors to zero position.
4- Enable the hardware protection circuit by setting (HPE=1).

Figure 5.27 shows an example of the window shown on the computer when the EX1 error is encountered.

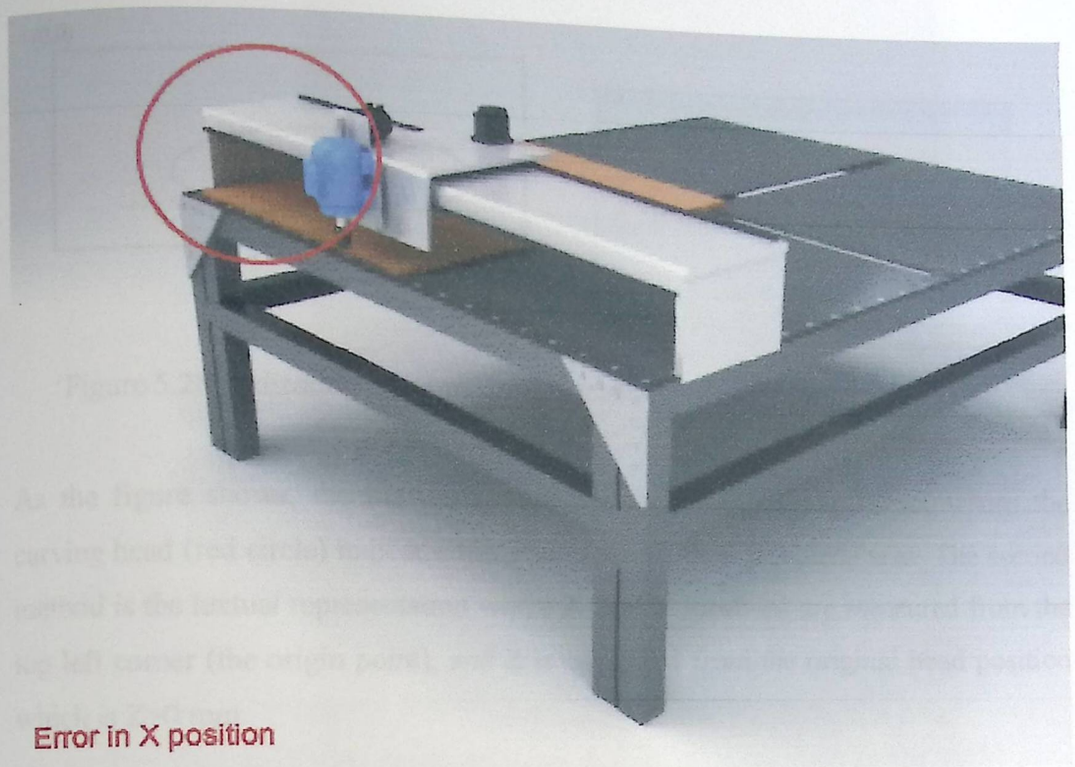


Figure 5.27: EX1 Error Reported on Monitoring Program

5.6.3 Carving Head Tracking

One of the main functionalities of the monitoring subsystem is to show the carving head position on the three axes all the time, knowing the head position depends on two parameters:

- Signals going out to the machine through the parallel port
- Acknowledgments received from the motors.

Indicating the carving head position is done by two different representations, figure 5.28 show both methods

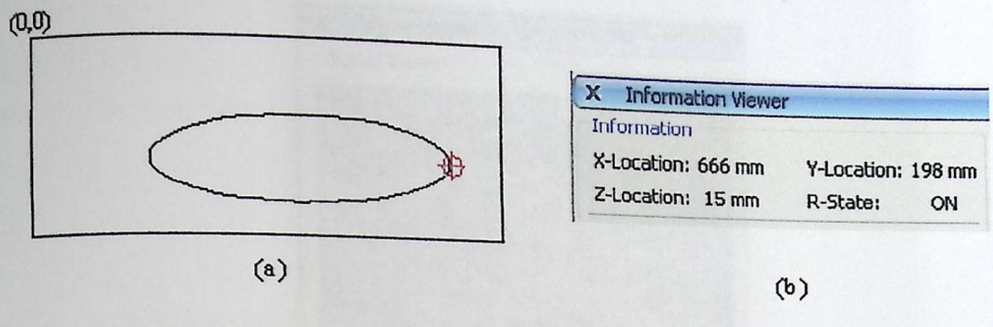


Figure 5.28: Indicating Carving Head Location, a) Graphically b) Textually

As the figure shows, the first method is the graphical representation, where the carving head (red circle) indicates the position over the wood piece area. The second method is the textual representation where X, and Y locations are measured from the top left corner (the origin point), and Z is measured from the original head position which is Z=0 mm.

In addition to the X, Y and Z location the monitoring subsystem will indicate the R-motor state, whether it's rotating or not.

5.6.4 Displaying Control Signals

Controlling the motors is achieved after a long and complicated process, which ends by outputting the appropriate signals on the parallel port, as kind of check on the system work, the monitoring subsystem displays all the signals going out on each of the parallel port pins, where each pin is connected to one motor, figure 5.29 shows these signals

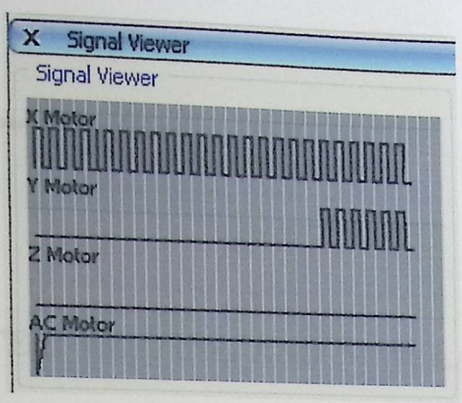


Figure 5.29: Graphical Control Signals

5.6.5 Depth Indicator

As user requirements specify, multiple carving depths should be possible in the system, then indicating these different depths must be done in the monitoring subsystem, where a depth indicator is used to visualize the current depth of the Z-motor, figure 5.30 shows the depth indicator, the indicator will map each color to its corresponding depth, highlighting the currently used depth.

Figure 5.30: Depth Indicator

CHAPTER SIX

SUBSYSTEM TESTING AND SIMULATION

6.1 Parallel Port Testing

6.2 Signals Generation and Timing Test

6.3 Manual Mode Test

6.4 Half Automatic Mode Test

6.5 Full Automatic Mode Test

6.6 Error Encoding and Feedback Test

Before constructing the machine physically and controlling it by computer we will simulate the operation of the driving program and the interfacing circuit using a group of simple circuits. By this simulation we will be sure of the operation of each subsystem and help in detecting possible errors and modifying them.

The simulation process will start by constructing simple circuits and programs for each subsystem if needed, for example a simple program for controlling the parallel port data lines with an associated circuit of a group of LED's and switches. Then after being sure that each separate subsystem is working properly they will be combined together and tested all together. The following subsystems will be tested:

- Parallel port testing module.
- Signal generation and timing testing module.
- Manual mode.
- Half automatic mode.
- Full automatic mode.
- Error encoding and feedback data.

Each testing module of the above will be described in more details in the following sub sections.

6.1 Parallel Port Testing

This testing module is used to test the main functionality of the parallel port, in other words the input and output operations in the parallel port, where a simple program will be implemented for input and output operations.

The program will be able to do the following:

- Output specific values (0 or 5 volts) to each pin of the parallel data lines. These pins will be connected to a group of LED's to test the output.
- Read the value on each of the parallel port input lines (status lines). A DIP switch will be used to send a 0 or 5 volts value to the input lines.

Figure 6.1 shows the program used to control the operation of the parallel port:

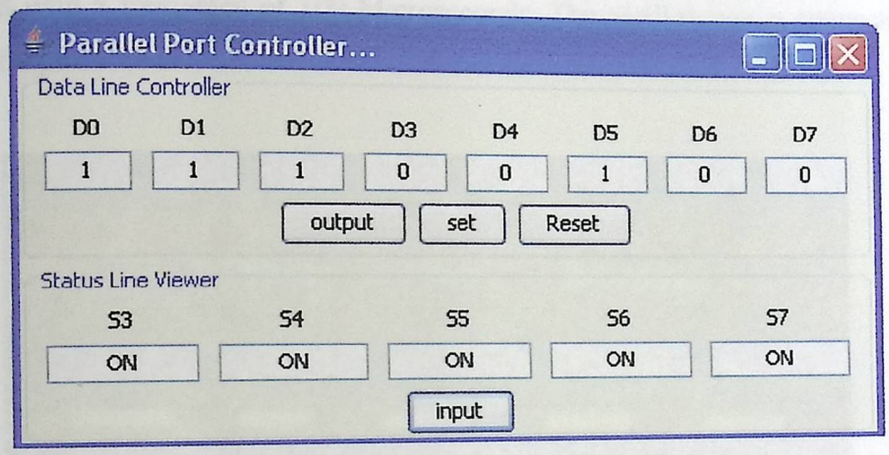


Figure 6.1: Parallel Port Testing Module

6.2 Signals Generation and Timing Test

The aim of this module is to test the accuracy of the timing signals generated by the parallel port. The program will output an ON-OFF signal on the parallel port line D0 with a user predefined delay (in Microseconds). To test this signal accuracy the output will be connected to an oscilloscope.

Accuracy in the generated timing signals is very important in servo motors motion control operation, where precision in positioning the carving head requires accurate timing signals. Figure 6.3 shows the program used to generate the signals.

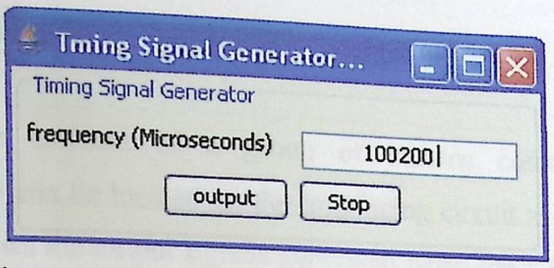


Figure 6.2: Timing Generation Testing Module.

Figure 6.4 shows the signal displayed on the oscilloscope screen for a signal generated with a frequency of 100 Microseconds. The oscilloscope is connected to the parallel port line D0.

Figure 6.3: Generated Signal Viewed on an Oscilloscope

6.3 Manual Mode Testing

The manual mode consists of a group of buttons connected to a 16F84 microcontroller, and can be located on the interfacing circuit as shown in figure 6.4 while figure 6.5 shows the output signals generated when the +X and –X buttons are pressed, where the first signal is the Xc signal and the second signal is the Xd1 signal which is high when moving in the forward direction.

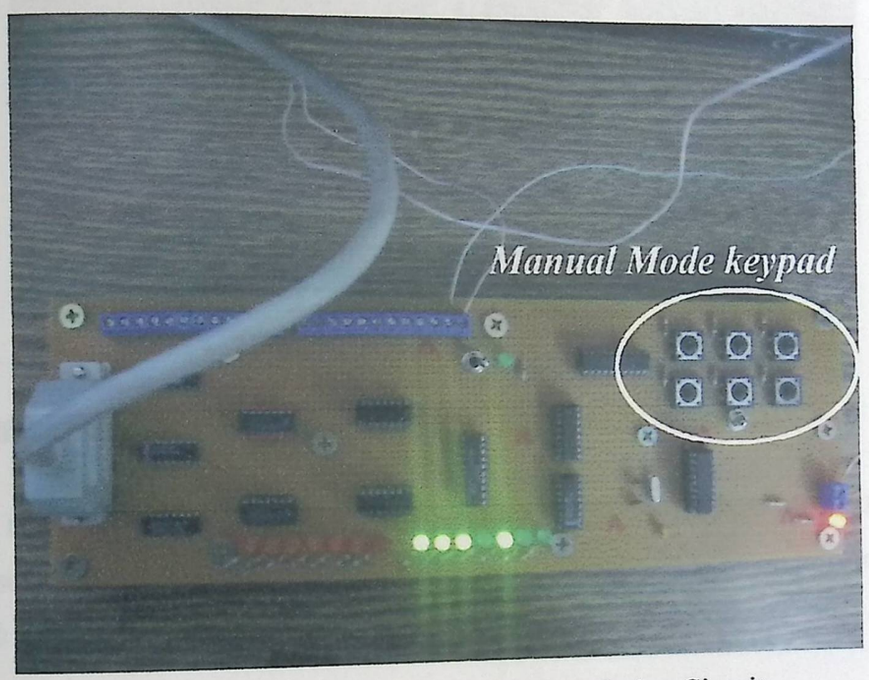


Figure 6.4 Manual Mode on The Interfacing Circuit

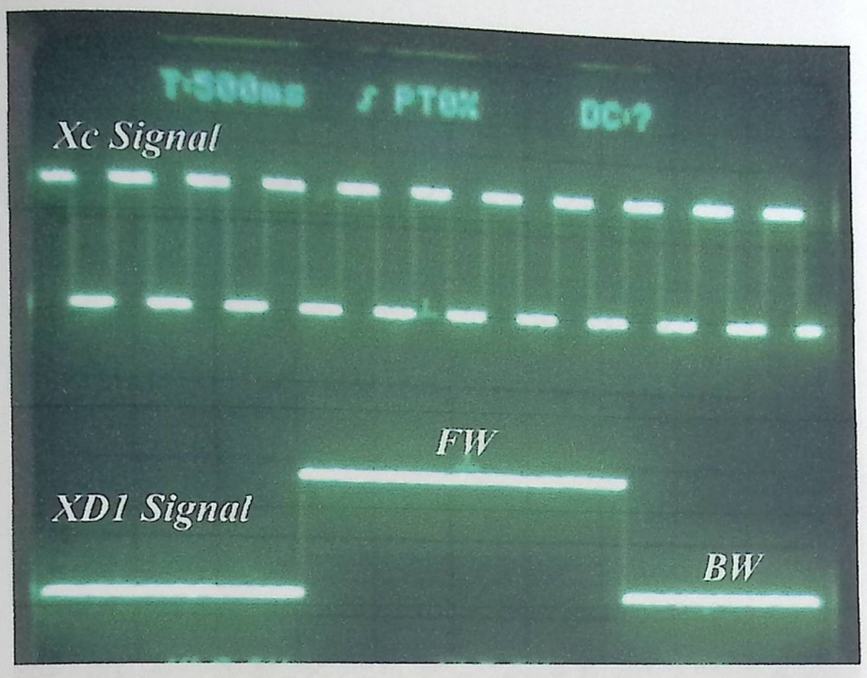


Figure 6.5 Output Signals From The Interfacing Circuit.

6.4 Half Automatic Mode Testing

The manual control subsystem is used to control the machine manually, using it the user can send the basic instructions to the machine such as moving from current location to another location, increasing the X direction and so on.

Figure 6.6 shows the main panel of the half automatic control, while figure 6.7 shows the generated signals on the oscilloscope when Carve To 100, 200 command is executed.

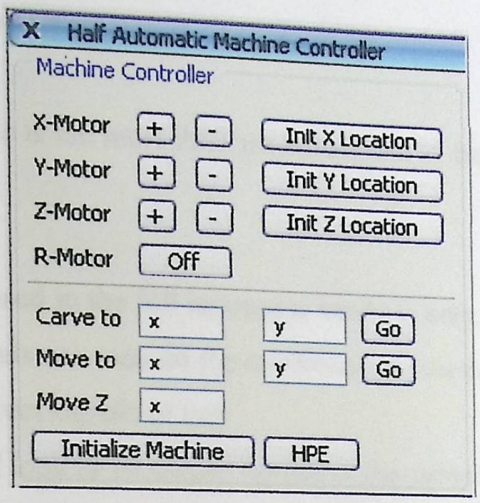


Figure 6.6 Half Automatic Control

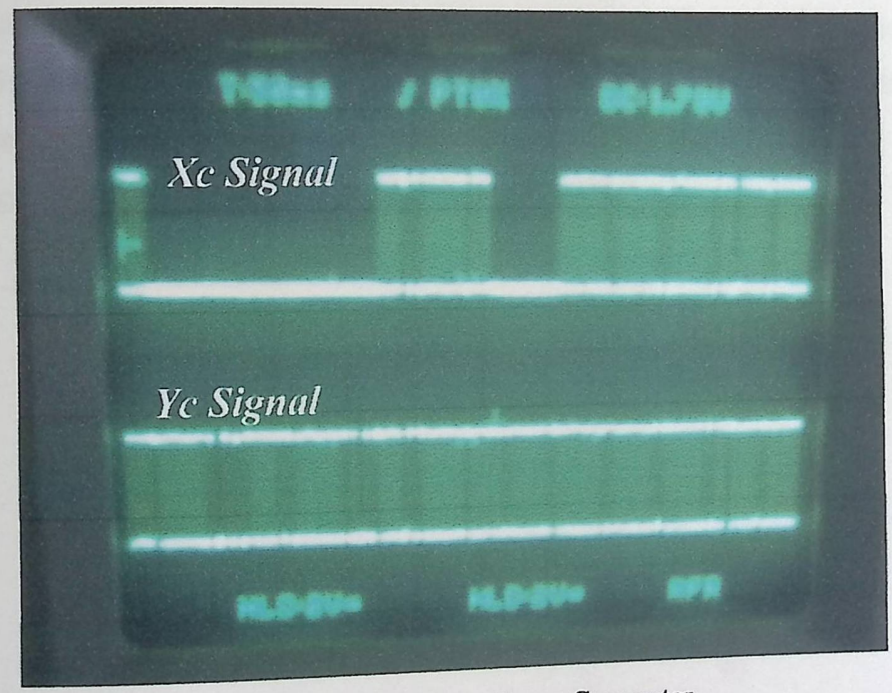


Figure 6.7: Outputs From Computer

6.5 Full Automatic Mode Test

The full automatic mode is the most difficult subsystem to test; this is because of the followings points:

1. The frequency used in the full automatic mode is very high.
2. Even if the signals are seen on the oscilloscope you can't understand whether they are the correct signals or not.
3. The user should look at 10 output signals at the same time which are three for each servo motor, and the on/off signal of the AC motor.

Because of the previously mentioned difficulties in testing the full automatic mode, simulation software was designed and integrated in the monitoring subsystem, this software was used to translate the signals outputted through the parallel port to drawings drawn on the computer as shown if figure 6.8

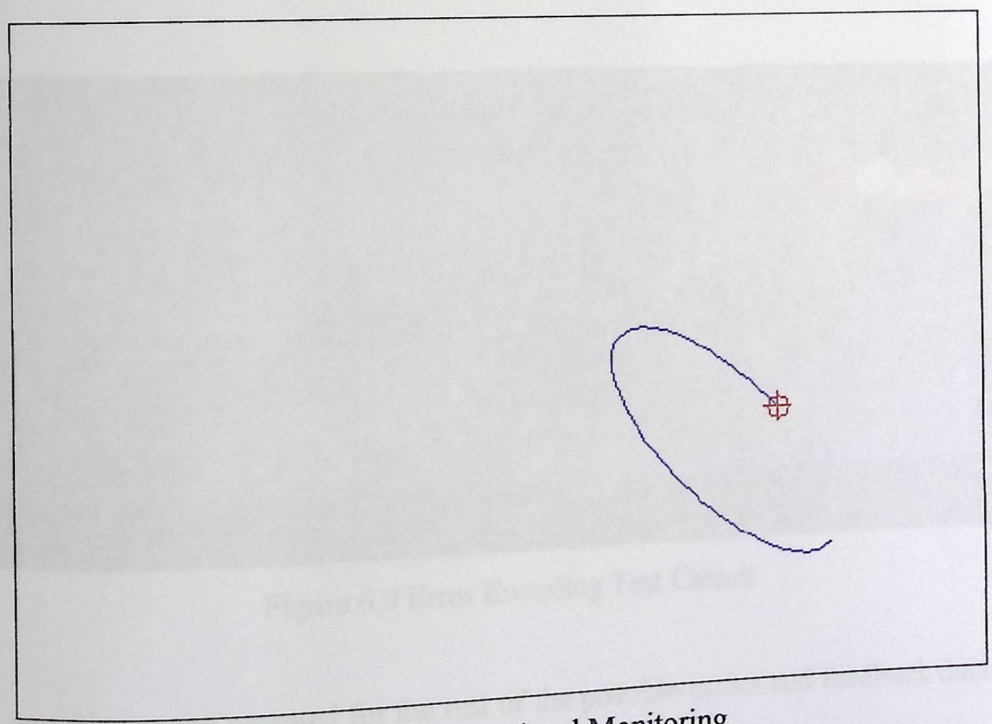


Figure 6.8: Simulated Monitoring

To be sure that the full automatic mode was working properly, an image was drawn using the drawing program and then carving signals where sent through the computer, when the carving process was terminated we compared the original image and the image drawn on the monitoring subsystem and they where identical and so insuring that the outputted signals where correct.

6.6 Error Encoding and Feedback Test

Errors returned to the computer are encoded in a 4-bit value inputted by the parallel port lines S3 – S6; the 4-bit encoding is generated using an encoder circuit. To test the correct operation of this module we connected the output of the encoder circuit to the parallel port S3 – S6 and signaled an error of type EX1 and then we read the input data on the computer using the simple parallel program and got the value of (0111) which is the corresponding code for the EX1 error as shown if figure 6.9

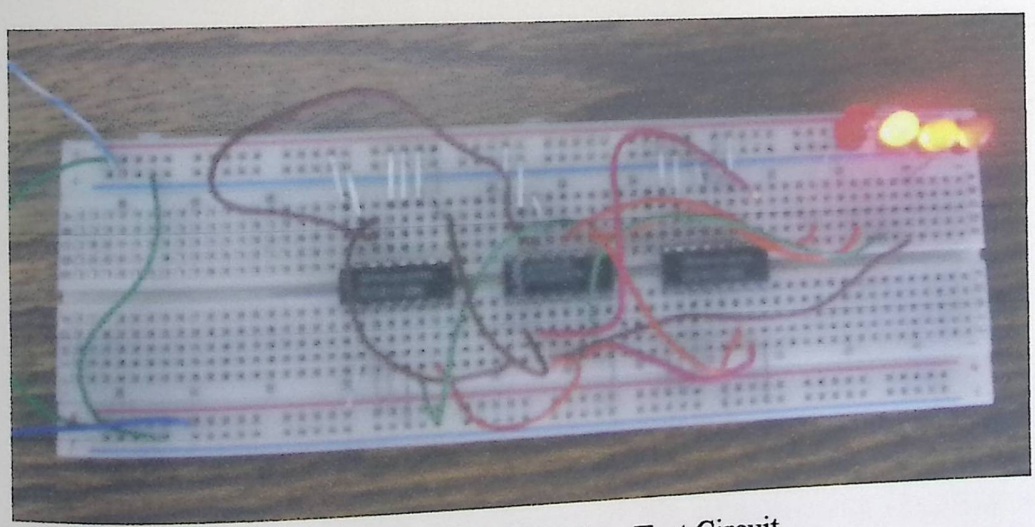


Figure 6.9 Error Encoding Test Circuit

All operations were repeated for the rest of the possible errors and feedback data and got the right 4-bit codes.

CHAPTER SEVEN

CONCLUSION AND FUTURE WORK

7.1 Conclusions

7.2 Suggestions and Future Work

7.1 Conclusions

During the design and implementation of the wood carving system, which consisted of the development of the interfacing circuit, and the implementation of the software driving program we have concluded the following:

General Conclusions:

- If more than one team with different backgrounds work on a common project, coordination and synchronization between all teams is the main key for whole project successfulness.
- New enchantments could be achieved in industrial machines if their operation is computerized, where computers allow adding new features that are difficult or even impossible to implement in hardware.
- Good knowledge of industrial machine hardware and user requirements is compulsory for software engineers in order to computerize that industrial system.

Software Implementation Conclusions:

- Choosing a programming language must be based on system requirements, with a big concern on the capabilities of the programming language and system direct needs.
- When dealing with big software programs, organizing the debugging process must be based on a systematic way.
- Object oriented programming simplifies the organization the hierarchy of the software, where different classes are classified in different packages.

- It's easier to split code implementation on various developers when using object oriented programming, since each developer implements his own code, and then all code is merged.
- Code reusability decreases the needed development time.
- Drawing programs face many memory leakage problems, and well organized algorithms must be used to save memory space and execution time.
- Multithreading increases programs responsiveness, especially when the system needs to execute time consuming processes such as image processing, and at the same time still accept user interactions.
- GUI programs must be implemented in such a way, where the user can configure them as he prefers; by giving him the ability to show and hide panels, change configurations, and colors… etc.
- When optimizing code performance, loops and repetitive code blocks must be considered, in addition to object allocation and deallocation.
- Secondary storage devices can be beneficial when huge amounts of data need to be manipulated during program execution, while performance is not a big concern. This is done by encoding data in special formats that could be retrieved later.
- Software developers should develop fast and less memory consuming code, even if it would lead to more difficult and bigger code implementation.
- Using appropriate data types could save memory, for example in some situations byte and short integer types could be used instead of integer types.
- Dynamic data structures – Linked lists instead of Arrays – leads to less memory consumption, where the data structure allocates memory only if needed.

Hardware Design Conclusions:

- In hardware design process, availability of hardware parts and components must be taken in consideration. For example if a microcontroller is needed the availability of its programmer should be considered.
- If more than one hardware component is used in the system, synchronization and timing between these parts is vital. Where fast components must wait for slower ones.
- Simulation and full study of circuits should be done before building the final system, since even small requirements may cause significant changes in the hardware design.
- When some hardware components are not available, it's possible to assemble these components from smaller components, for example a 16-4 encoder could be implemented by two 8-3 encoders.

7.2 Suggestions and Future Work

During the planning of the wood carving system, a number of ideas were thought of, some of them where implemented, while others where not because of time leakage.

- Use of multiple carving heads when carving a single image, where big heads are used in big areas and smaller ones are used when carving sensitive parts.
- Capability of carving scanned images; where an image is entered using a scanner, then processed to extract its information.
- Draw 3D simulation in the monitoring subsystem instead of drawing 2D shapes, where the Z depth is specified by a color instead of a third coordinate.

- Use of artificial intelligence in the system, especially in the drawing program, for example: showing guide lines, shape snapping … etc. or in the motors controller where the carving head could move at different speeds.
- Encode the file formats using Extendable markup language (XML) instead of using plain text.
- Add a zoom tool in the drawing program, where images can be zoomed in and out.
- Output the control signals using the USB port instead of the printer parallel port, since its faster and new computers come with no parallel port plug-in.
- Control the operation of the machine using internet; this would be useful in order to monitor its progress using remote computers. For example the machine would be working in a factory and the user could control and monitor its progress from any where using internet.
- Develop a plug-in program that can be embedded in large drawing applications, like Adobe Photoshop, CorelDraw, or Autodesk AutoCAD.
- Enhance the drawing program capabilities, by adding new tools to the toolbox.
- Add a shapes and decorations library; where the user can create new shapes from existing shapes.

REFERENCES

Iovine, J. 2000
PIC Microcontroller Project Book, Tab books, 1st Edition

Magill, R. 2003
Motor Learning: concepts and Applications, McGraw-Hill, 7 Edition

Millman, J. 1982
Integrated Electronics: Analog and Digital Circuits, Mcgraw-Hill, 1st Edition

Ian, D. 2001
Java Cookbook, O'Reilly, 1st Edition

Knudsen, J. 1999
Java 2d Graphics, O'Reilly, 2nd Edition

Garcia, A. 1999
Programmer's Guide to the Java 2D API, Sun Microsystems, 2nd Edition

Vincent J. Hardy, 1999
Java 2D API Graphics, Prentice Hall, 2nd Edition

Sommerville, Ian, 2001
Software Engineering, Addison-Wisley. 6th Edition

Ramesh S. Goankar, 2000
Microprocessor Architecture Programming with 8085, Prentice Hall, 5th Edition

Walter A.Triebel, 2003
The 8088 and 8086 Microprocessors, Prentice Hall, 4th Edition

Russel Stuart, 2003
Artificial Intelligence A Modern Approach, Prentice Hall, 2nd Edition

Scott Oaks, 2004
Java Threads, O'Reilly, 2nd Edition

Leen Ammeraal, 2006
Computer Graphics for Java Programmers, Wiley, 1st Edition

Y. Daniel Liang, 2006
Computer Graphics Using Java 2D and 3D, Prentice Hall, 1st Edition

JDK™ 5.0 Documentation
http://java.sun.com/j2se/1.5.0/docs/

Toggle an LED using interrupts on a Microchip PIC16F84
http://www.electronic-eng.com/pic/picporg003.html

PIC microcontroller
http://en.wikipedia.org/wiki/PIC_microcontroller

The Basics of Programming and Using a PIC Microcontroller
http://oak.cats.ohiou.edu/~db283101/pichowto.html

What's a Servo
http://www.seattlerobotics.org/guide/servos.html

Microchip PIC 16F84 Microcontroller
http://www.mstracey.btinternet.co.uk/pictutorial/picmain.htm

The SV203 RC Servo Motor Controller
http://www.pontech.com/prducts/sv200/index.htm

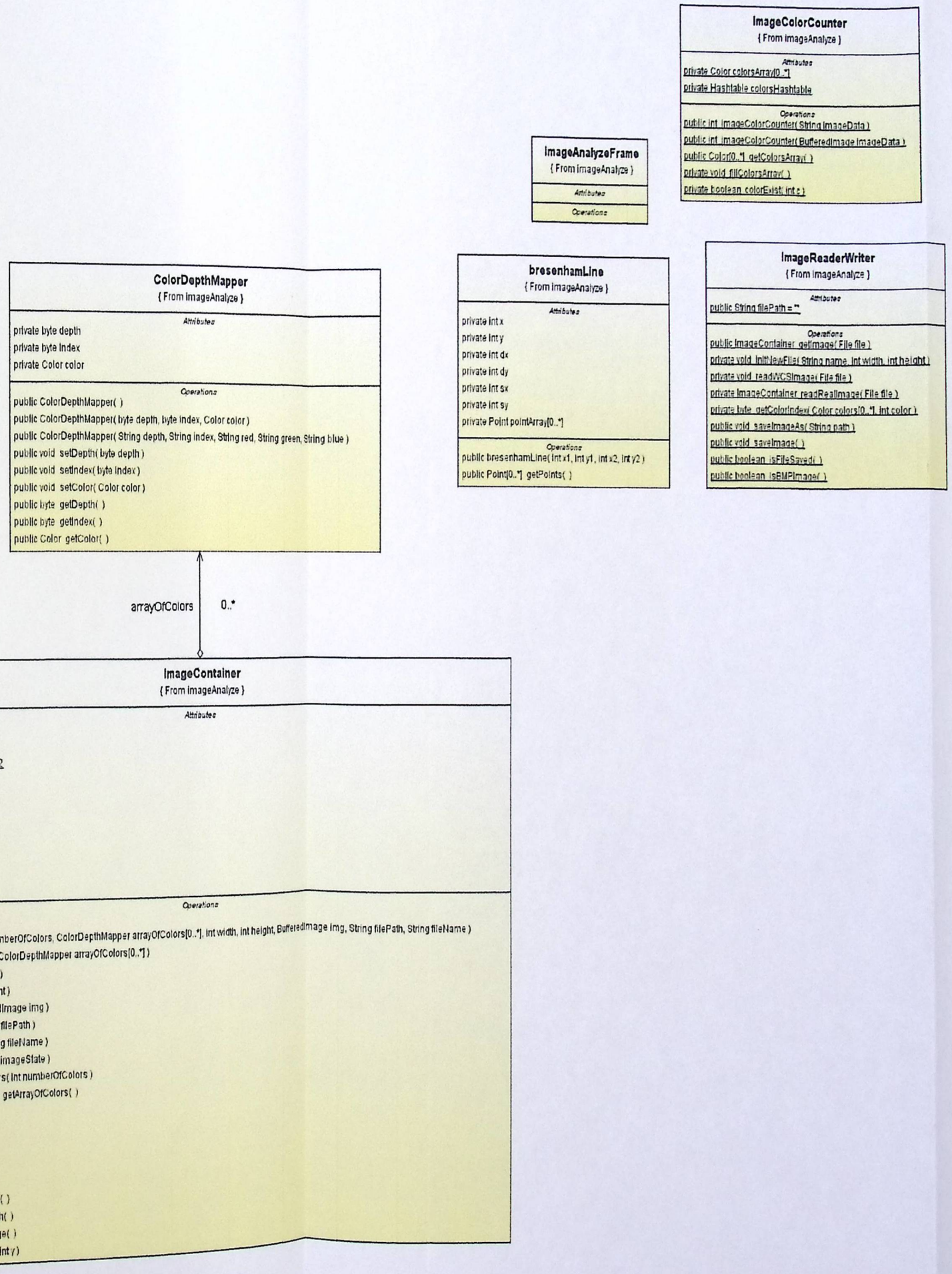
Computer Numerical Control
http://en.wikipedia.org/wiki/CNC

G-code programming language
http://en.wikipedia.org/wiki/G-code

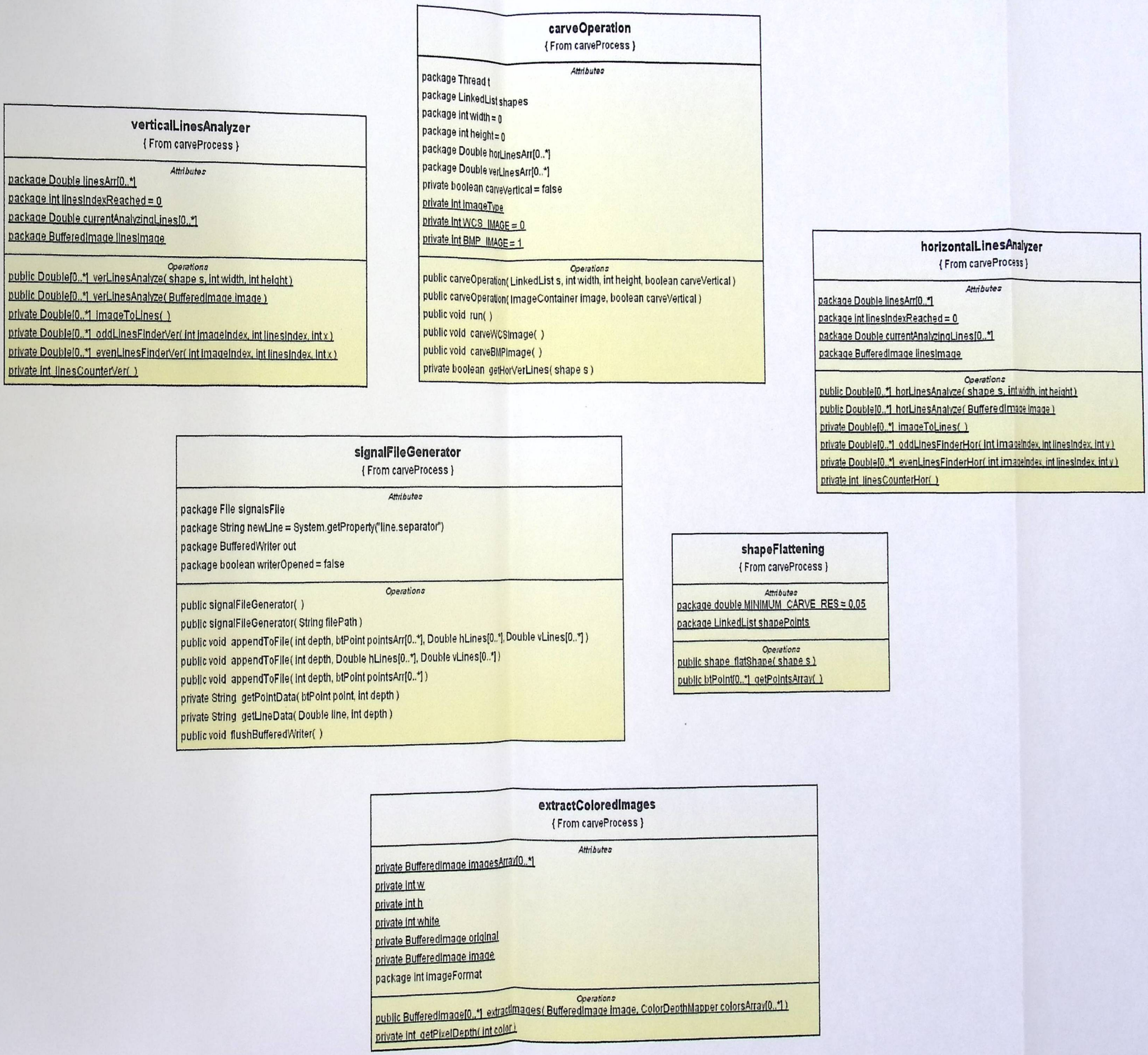
APPENDIX A: UML DIAGRAMS

1. Image Analyzer package
2. Carve Process package
3. Parallel Port package
4. Machine Simulation package
5. Shared Classes package

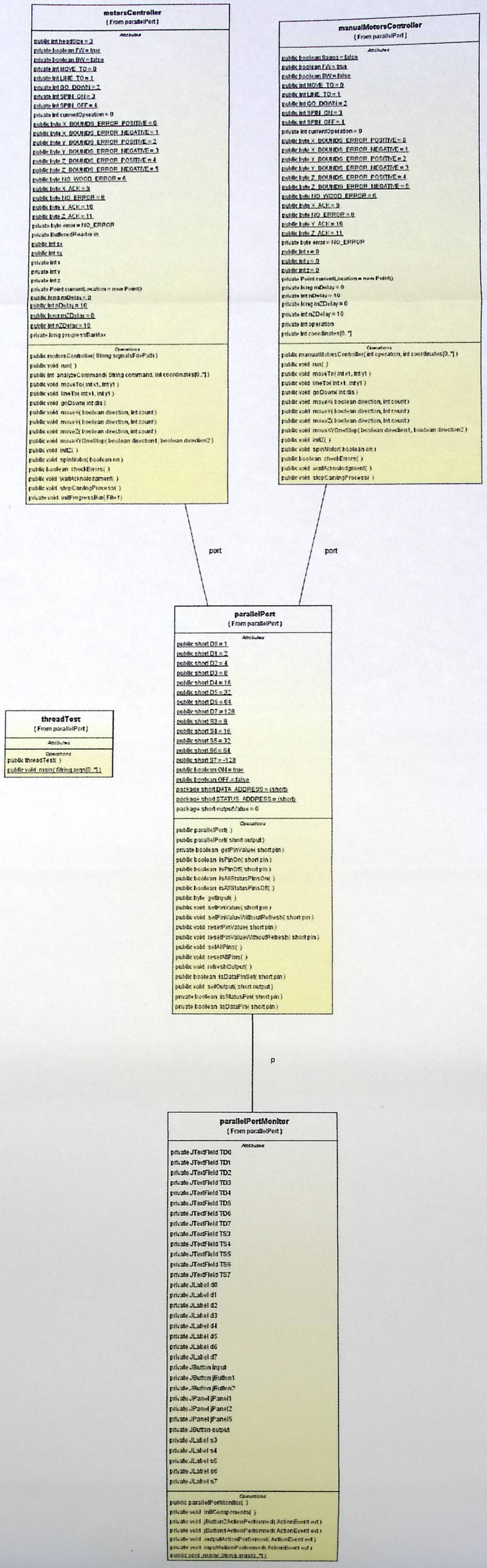
Image Analyzer package UML diagram



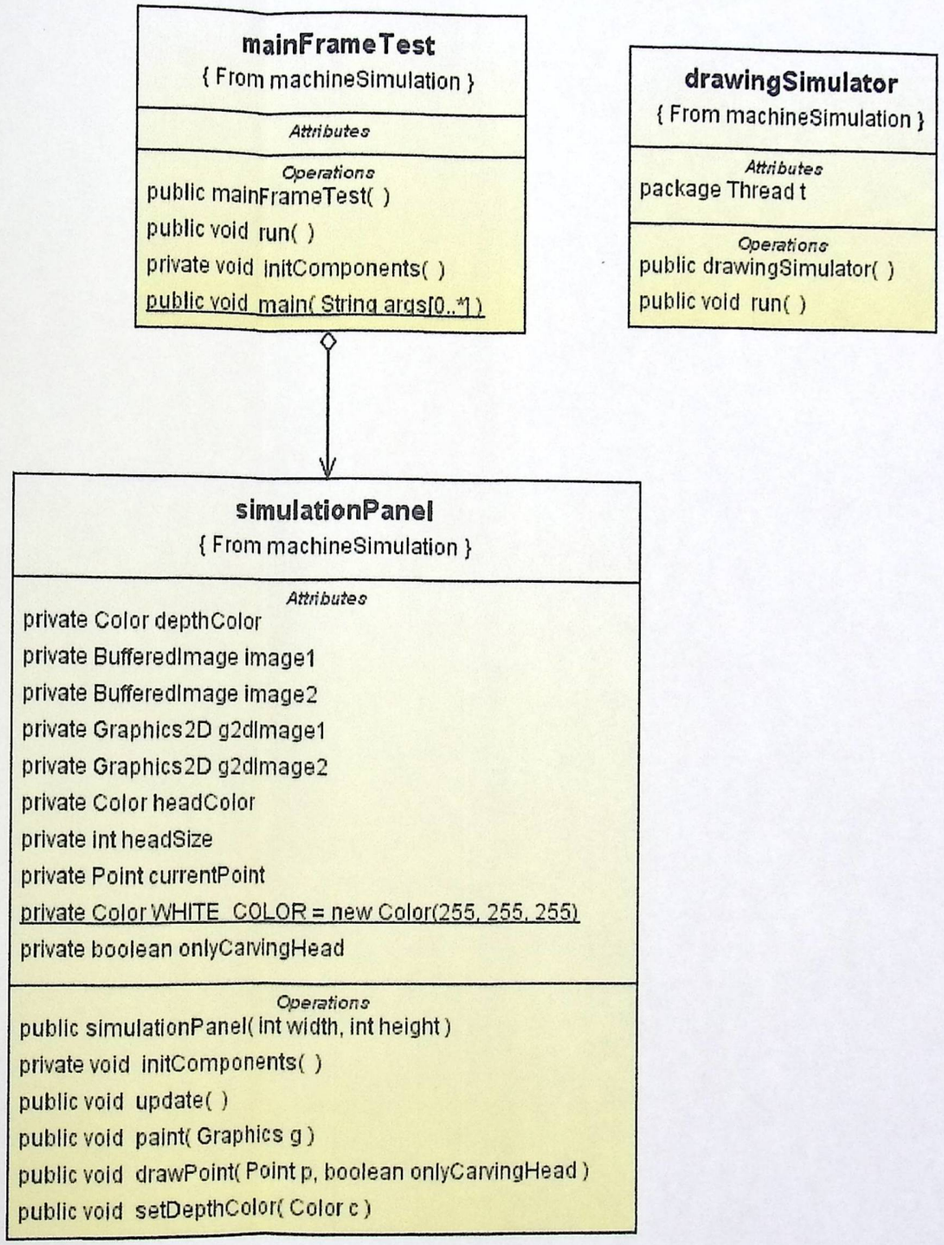
Carve Process package UML diagram



Parallel Port package UML diagram



Machine Simulation package UML Diagram



Shared Classes package UML Diagram

CentredBackgroundBorder
{ From sharedClasses }

Attributes
- private BufferedImage image

Operations
- public CentredBackgroundBorder(BufferedImage image)
- public void paintBorder(Component c, Graphics g, int x, int y, int width, int height)
- public Insets getBorderInsets(Component c)
- public boolean isBorderOpaque( )

bitmapsInfo
{ From sharedClasses }

Attributes
- package LinkedList allBitmaps

Operations
- public bitmapsInfo( )
- public LinkedList getList( )
- public void addImage(bitmapInfo img)
- public BufferedImage getImage(int ID)
- public int getDepth(int ID)
- public void setDepth(int ID, int newDepth)
- public Color getColor(int ID)
- public boolean getCarveState(int ID)
- public void setCarveState(int ID, boolean newState)
- public int getSize( )

imageProperties
{ From sharedClasses }

Attributes
- public int REAL_FORMAT = 0
- public int WCS_FORMAT = 1

Operations

style
{ From sharedClasses }

Attributes
- public boolean showGrid = true
- public int gridSize = 10
- public Color gridColor = new Color(204, 204, 204)
- public boolean transparentBG = false
- public int transGridSize = 8
- public Color transGridColor = new Color(204, 204, 204)
- public Color bgColor = new Color(255, 255, 255)
- public Paint paint = Color.BLACK
- public BasicStroke stroke = new BasicStroke(1..., BasicStroke.CAP_BUTT..., BasicStroke.JOIN_ROUND, 1, new float[], {1, 0}, 0)
- public Color transformColor = Color.RED
- public int transformRectLength = 6
- public Color selectedLayerBg = new Color(49, 106, 197)
- public Stroke selectionStroke = new BasicStroke(2..., BasicStroke.CAP_BUTT..., BasicStroke.JOIN_ROUND, 1, new float[], {10, 10}, 0)
- public Paint selectionPaint = Color.RED
- public int roundVal = 0
- public Color paintColor = Color.BLACK
- public double startAngle = 0
- public double endAngle = 0
- public int arcCloser = 0
- public BasicStroke arcRectangleStroke = new BasicStroke(2..., BasicStroke.CAP_BUTT..., BasicStroke.JOIN_ROUND, 1, new float[], {1, 1}, 0)
- public Color arcRectangleColor = Color.BLACK
- public BasicStroke selectStroke = new BasicStroke(1..., BasicStroke.CAP_BUTT..., BasicStroke.JOIN_ROUND, 1, new float[], {4, 1}, 0)
- public Color selectColor = Color.gray
- public boolean selectionMode = shapes.CONTAINS
- public BufferedImage bmpBG
- public boolean showBMP = false
- public int bmpTx = 0
- public int bmpTy = 0
- public int eraserShape = shapes.RECTANGLE
- public int eraserWidth = 10
- public int eraserHeight = 10
- public Color eraserColor = Color.GRAY
- public Image blankCursorImage
- public Cursor blankCursor
- public Image bucketCursorImage
- public Cursor bucketCursor
- public Image eyeDropperCursorImage
- public Cursor eyeDropperCursor
- public Image handCursorImage
- public Cursor handCursor

Operations
- public style( )

Errors
{ From sharedClasses }

Attributes
- private String error1 = "Depth must be any number between 0-30 only"
- private String error2 = "File type not applicable"
- private String error3 = "File must be in BMP or TXT formats"
- private String error4 = "Unable to import file, only BMP files allowed"
- private String error5 = "File not found"
- public byte DEPTH_ERROR = 1
- public byte FILE_NOT_APPLICAPLE_ERROR = 2
- public byte FILE_TYPE_ERROR = 3
- public byte UNABLE_TO_IMPORT_ERROR = 4
- public byte FILE_NOT_FOUND = 5

Operations
- public void displayError(byte index)
- public void displayError(Component c, byte index)

bitmapInfo
{ From sharedClasses }

Attributes
- private BufferedImage thumbnailImage
- private int depth
- private boolean carveState
- private int imageID
- private Color imageColor

Operations
- public bitmapInfo( )
- public bitmapInfo(int imageID, BufferedImage thumb, int depth, boolean carveState, Color color)
- public BufferedImage getThumbnailImage( )
- public void setThumbnailImage(BufferedImage thumbnailImage)
- public int getDepth( )
- public void setDepth(int depth)
- public boolean getCarveState( )
- public void setCarveState(boolean carveState)
- public int getImageID( )
- public void setImageID(int imageID)
- public Color getImageColor( )
- public void setImageColor(Color imageColor)

btPoint
{ From sharedClasses }

Attributes
- private int pointType
- public int MOVE_TO_POINT = 0
- public int LINE_TO_POINT = 1

Operations
- public btPoint( )
- public btPoint(int x, int y, int pointType)
- public int getPointType( )
- public void setPointType(int pointType)
- public Point getPoint( )
- public double getX( )
- public double getY( )
- public String toString( )
- public boolean equal(btPoint p)

progressFrame
{ From sharedClasses }

Attributes
- public JLabel processName
- public JProgressBar progressBar

Operations
- public progressFrame( )
- private void initComponents( )
- public void main(String args[0..*])
- public void run( )
- public void setValue(int v)
- public void setProcessName(String pName)

measurementsCalc
{ From sharedClasses }

Attributes
- public double PIXEL = 1
- public double CM = 72 / 2.54
- public double MM = 72 / 25.4
- public double INCH = 72
- public double converter = 1

Operations
- public measurementsCalc( )
- public double toPixel(String value, String metric)
- public double metricConvert(String value, int m)

shapeObject
{ From sharedClasses }

Attributes
- private String shapeName = ""
- private int X = 0
- private int Y = 0

Operations
- public shapeObject( )
- public shapeObject(String name, int x, int y)
- public String getShapeName( )
- public int getX( )
- public int getY( )
- public void setShapeName(String name)
- public void setX(int x)
- public void setY(int y)

shapeRect
{ From sharedClasses }

Attributes
- private int width = 0
- private int height = 0

Operations
- public shapeRect( )
- public shapeRect(int x, int y, int width, int height)
- public int getWidth( )
- public int getHeight( )
- public void setWidth(int width)
- public void setHeight(int height)

shapeLine
{ From sharedClasses }

Attributes
- private int x2 = 0
- private int y2 = 0

Operations
- public shapeLine( )
- public shapeLine(int x1, int y1, int x2, int y2)
- public int getX2( )
- public int getY2( )
- public void setX2(int x2)
- public void setY2(int y2)

shapeCircle
{ From sharedClasses }

Attributes
- private int width = 0
- private int height = 0

Operations
- public shapeCircle( )
- public shapeCircle(int x, int y, int width, int height)
- public int getWidth( )
- public int getHeight( )
- public void setWidth(int width)
- public void setHeight(int height)

APPENDIX B: DATASHEETS

1. PIC16F84 Datasheet
2. 74LS157 IC – Quad 2-line to 1-line data selectors – multiplexers datasheet
3. 74LS125A IC – Quad 3 state buffers datasheet
4. 74LS148 IC – 8 to 3 lines Priority Inverting Encoder

MICROCHIP

PIC16F84A

18-pin *Enhanced* FLASH/EEPROM 8-Bit Microcontroller

High Performance RISC CPU Features:

- Only 35 single word instructions to learn
- All instructions single-cycle except for program branches which are two-cycle
- Operating speed: DC - 20 MHz clock input
  DC - 200 ns instruction cycle
- 1024 words of program memory
- 68 bytes of Data RAM
- 64 bytes of Data EEPROM
- 14-bit wide instruction words
- 8-bit wide data bytes
- 15 Special Function Hardware registers
- Eight-level deep hardware stack
- Direct, indirect and relative addressing modes
- Four interrupt sources:
  - External RB0/INT pin
  - TMR0 timer overflow
  - PORTB<7:4> interrupt-on-change
  - Data EEPROM write complete

Peripheral Features:

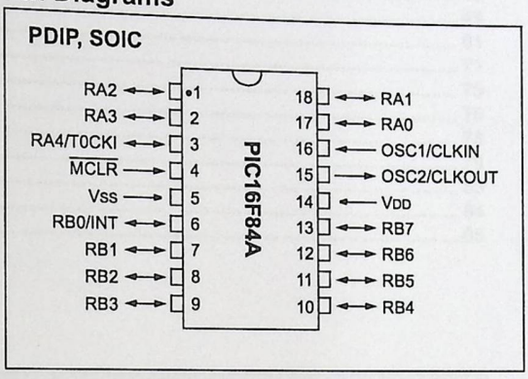
- 13 I/O pins with individual direction control
- High current sink/source for direct LED drive
  - 25 mA sink max. per pin
  - 25 mA source max. per pin
- TMR0: 8-bit timer/counter with 8-bit programmable prescaler

Special Microcontroller Features:

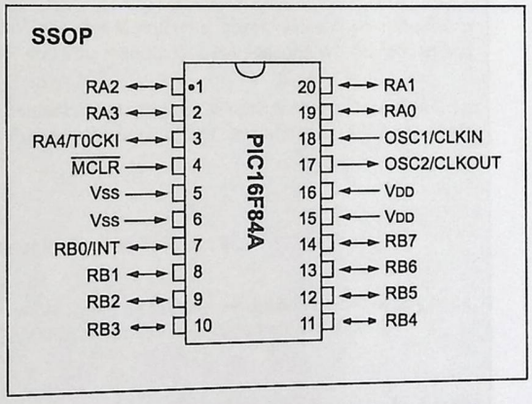
- 10,000 erase/write cycles *Enhanced* FLASH Program memory typical
- 10,000,000 typical erase/write cycles EEPROM Data memory typical
- EEPROM Data Retention > 40 years
- In-Circuit Serial Programming™ (ICSP™) - via two pins
- Power-on Reset (POR), Power-up Timer (PWRT), Oscillator Start-up Timer (OST)
- Watchdog Timer (WDT) with its own On-Chip RC Oscillator for reliable operation
- Code protection
- Power saving SLEEP mode
- Selectable oscillator options

Pin Diagrams

PDIP, SOIC



SSOP



CMOS Enhanced FLASH/EEPROM Technology:

- Low power, high speed technology
- Fully static design
- Wide operating voltage range:
  - Commercial: 2.0V to 5.5V
  - Industrial: 2.0V to 5.5V
- Low power consumption:
  - < 2 mA typical @ 5V, 4 MHz
  - 15 µA typical @ 2V, 32 kHz
  - < 0.5 µA typical standby current @ 2V

Table of Contents

TO OUR VALUED CUSTOMERS

It is our intention to provide our valued customers with the best documentation possible to ensure successful use of your Microchip products. To this end, we will continue to improve our publications to better suit your needs. Our publications will be refined and enhanced as new volumes and updates are introduced.

If you have any questions or comments regarding this publication, please contact the Marketing Communications Department via E-mail at **docerrors@mail.microchip.com** or fax the **Reader Response Form** in the back of this data sheet to (480) 792-4150. We welcome your feedback.

Most Current Data Sheet

To obtain the most up-to-date version of this data sheet, please register at our Worldwide Web site at:

http://www.microchip.com

You can determine the version of a data sheet by examining its literature number found on the bottom outside corner of any page. The last character of the literature number is the version number, (e.g., DS30000A is version A of document DS30000).

Errata

An errata sheet, describing minor operational differences from the data sheet and recommended workarounds, may exist for current devices. As device/documentation issues become known to us, we will publish an errata sheet. The errata will specify the revision of silicon and revision of document to which it applies.

To determine if an errata sheet exists for a particular device, please check with one of the following:

- Microchip's Worldwide Web site; http://www.microchip.com
- Your local Microchip sales office (see last page)
- The Microchip Corporate Literature Center; U.S. FAX: (480) 792-7277

When contacting a sales office or the literature center, please specify which device, revision of silicon and data sheet (include literature number) you are using.

Customer Notification System

Register on our web site at **www.microchip.com/cn** to receive the most current information on all of our products.

1.0 DEVICE OVERVIEW

This document contains device specific information for the operation of the PIC16F84A device. Additional information may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023), which may be downloaded from the Microchip website. The Reference Manual should be considered a complementary document to this data sheet, and is highly recommended reading for a better understanding of the device architecture and operation of the peripheral modules.

The PIC16F84A belongs to the mid-range family of the PICmicro® microcontroller devices. A block diagram of the device is shown in Figure 1-1.

The program memory contains 1K words, which translates to 1024 instructions, since each 14-bit program memory word is the same width as each device instruction. The data memory (RAM) contains 68 bytes. Data EEPROM is 64 bytes.

There are also 13 I/O pins that are user-configured on a pin-to-pin basis. Some pins are multiplexed with other device functions. These functions include:

- External interrupt
- Change on PORTB interrupt
- Timer0 clock input

Table 1-1 details the pinout of the device with descriptions and details for each pin.

FIGURE 1-1: PIC16F84A BLOCK DIAGRAM

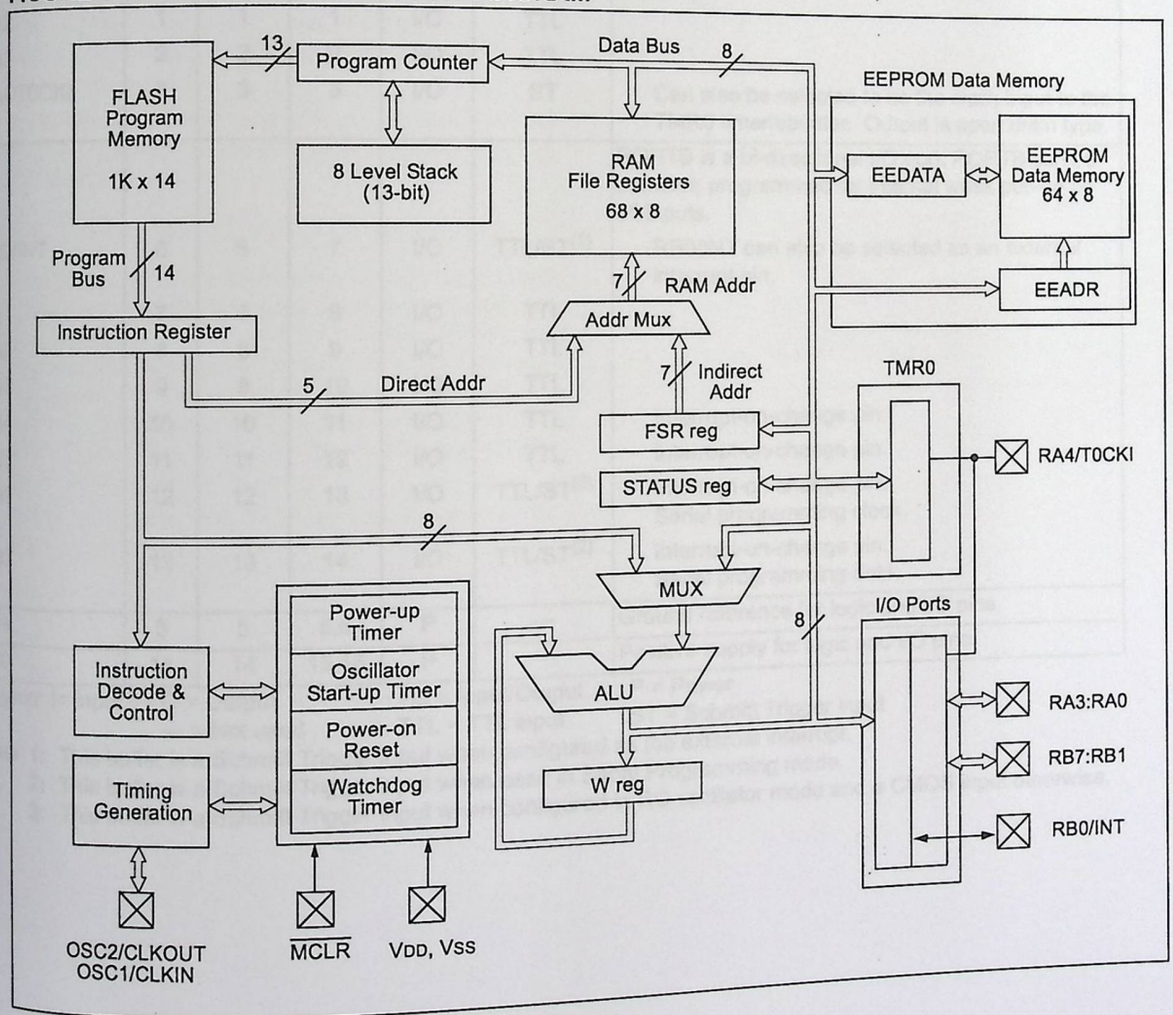


TABLE 1-1: PIC16F84A PINOUT DESCRIPTION

Pin Name	PDIP No.	SOIC No.	SSOP No.	I/O/P Type	Buffer Type	Description
OSC1/CLKIN	16	16	18	I	ST/CMOS(3)	Oscillator crystal input/external clock source input.
OSC2/CLKOUT	15	15	19	O	—	Oscillator crystal output. Connects to crystal or resonator in Crystal Oscillator mode. In RC mode, OSC2 pin outputs CLKOUT, which has 1/4 the frequency of OSC1 and denotes the instruction cycle rate.
$\overline{MCLR}$	4	4	4	I/P	ST	Master Clear (Reset) input/programming voltage input. This pin is an active low RESET to the device.
						PORTA is a bi-directional I/O port.
RA0	17	17	19	I/O	TTL	
RA1	18	18	20	I/O	TTL	
RA2	1	1	1	I/O	TTL	
RA3	2	2	2	I/O	TTL	
RA4/T0CKI	3	3	3	I/O	ST	Can also be selected to be the clock input to the TMR0 timer/counter. Output is open drain type.
						PORTB is a bi-directional I/O port. PORTB can be software programmed for internal weak pull-up on all inputs.
RB0/INT	6	6	7	I/O	TTL/ST(1)	RB0/INT can also be selected as an external interrupt pin.
RB1	7	7	8	I/O	TTL	
RB2	8	8	9	I/O	TTL	
RB3	9	9	10	I/O	TTL	
RB4	10	10	11	I/O	TTL	Interrupt-on-change pin.
RB5	11	11	12	I/O	TTL	Interrupt-on-change pin.
RB6	12	12	13	I/O	TTL/ST(2)	Interrupt-on-change pin. Serial programming clock.
RB7	13	13	14	I/O	TTL/ST(2)	Interrupt-on-change pin. Serial programming data.
VSS	5	5	5,6	P	—	Ground reference for logic and I/O pins.
VDD	14	14	15,16	P	—	Positive supply for logic and I/O pins.

Legend: I= input O = Output I/O = Input/Output P = Power
— = Not used TTL = TTL input ST = Schmitt Trigger input

Note 1: This buffer is a Schmitt Trigger input when configured as the external interrupt.
2: This buffer is a Schmitt Trigger input when used in Serial Programming mode.
3: This buffer is a Schmitt Trigger input when configured in RC oscillator mode and a CMOS input otherwise.

2.0 MEMORY ORGANIZATION

There are two memory blocks in the PIC16F84A. These are the program memory and the data memory. Each block has its own bus, so that access to each block can occur during the same oscillator cycle.

The data memory can further be broken down into the general purpose RAM and the Special Function Registers (SFRs). The operation of the SFRs that control the "core" are described here. The SFRs used to control the peripheral modules are described in the section discussing each individual peripheral module.

The data memory area also contains the data EEPROM memory. This memory is not directly mapped into the data memory, but is indirectly mapped. That is, an indirect address pointer specifies the address of the data EEPROM memory to read/write. The 64 bytes of data EEPROM memory have the address range 0h-3Fh. More details on the EEPROM memory can be found in Section 3.0.

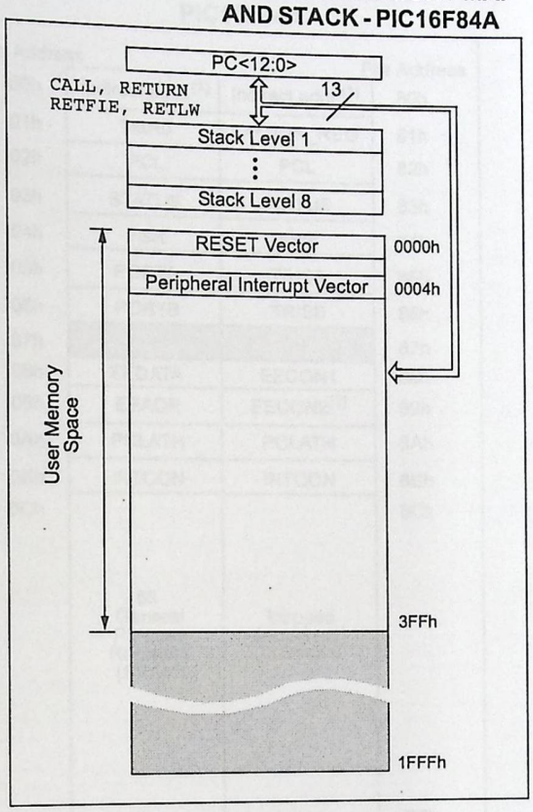
Additional information on device memory may be found in the PICmicro™ Mid-Range Reference Manual, (DS33023).

2.1 Program Memory Organization

The PIC16FXX has a 13-bit program counter capable of addressing an 8K x 14 program memory space. For the PIC16F84A, the first 1K x 14 (0000h-03FFh) are physically implemented (Figure 2-1). Accessing a location above the physically implemented address will cause a wraparound. For example, for locations 20h, 420h, 820h, C20h, 1020h, 1420h, 1820h, and 1C20h, the instruction will be the same.

The RESET vector is at 0000h and the interrupt vector is at 0004h.

FIGURE 2-1: PROGRAM MEMORY MAP AND STACK - PIC16F84A



2.2 Data Memory Organization

The data memory is partitioned into two areas. The first is the Special Function Registers (SFR) area, while the second is the General Purpose Registers (GPR) area. The SFRs control the operation of the device.

Portions of data memory are banked. This is for both the SFR area and the GPR area. The GPR area is banked to allow greater than 116 bytes of general purpose RAM. The banked areas of the SFR are for the registers that control the peripheral functions. Banking requires the use of control bits for bank selection. These control bits are located in the STATUS Register. Figure 2-2 shows the data memory map organization.

Instructions `MOVWF` and `MOVF` can move values from the W register to any location in the register file ("F"), and vice-versa.

The entire data memory can be accessed either directly using the absolute address of each register file or indirectly through the File Select Register (FSR) (Section 2.5). Indirect addressing uses the present value of the RP0 bit for access into the banked areas of data memory.

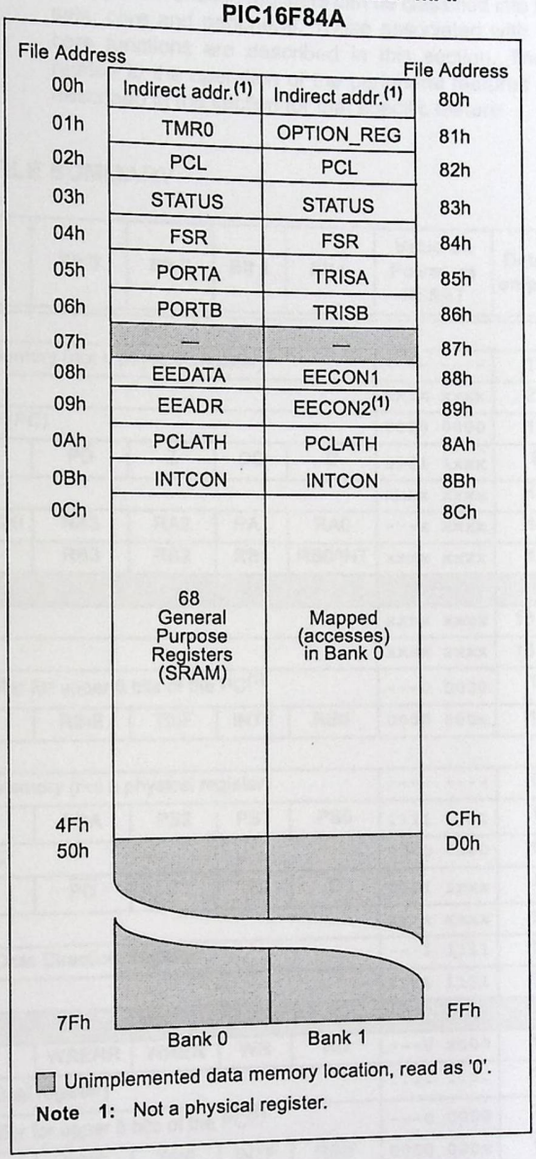
Data memory is partitioned into two banks which contain the general purpose registers and the special function registers. Bank 0 is selected by clearing the RP0 bit (STATUS<5>). Setting the RP0 bit selects Bank 1. Each Bank extends up to 7Fh (128 bytes). The first twelve locations of each Bank are reserved for the Special Function Registers. The remainder are General Purpose Registers, implemented as static RAM.

2.2.1 GENERAL PURPOSE REGISTER FILE

Each General Purpose Register (GPR) is 8-bits wide and is accessed either directly or indirectly through the FSR (Section 2.5).

The GPR addresses in Bank 1 are mapped to addresses in Bank 0. As an example, addressing location 0Ch or 8Ch will access the same GPR.

FIGURE 2-2: REGISTER FILE MAP - PIC16F84A



2.3 Special Function Registers

The Special Function Registers (Figure 2-2 and Table 2-1) are used by the CPU and Peripheral functions to control the device operation. These registers are static RAM.

The special function registers can be classified into two sets, core and peripheral. Those associated with the core functions are described in this section. Those related to the operation of the peripheral features are described in the section for that specific feature.

TABLE 2-1: SPECIAL FUNCTION REGISTER FILE SUMMARY

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on RESET	Details on page
Bank 0											
00h	INDF	Uses contents of FSR to address Data Memory (not a physical register)								---- ----	11
01h	TMR0	8-bit Real-Time Clock/Counter								xxxx xxxx	20
02h	PCL	Low Order 8 bits of the Program Counter (PC)								0000 0000	11
03h	STATUS(2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
04h	FSR	Indirect Data Memory Address Pointer 0								xxxx xxxx	11
05h	PORTA(4)	—	—	—	RA4/T0CKI	RA3	RA2	RA1	RA0	---x xxxx	16
06h	PORTB(5)	RB7	RB6	RB5	RB4	RB3	RB2	RB1	RB0/INT	xxxx xxxx	18
07h	—	Unimplemented location, read as '0'								—	—
08h	EEDATA	EEPROM Data Register								xxxx xxxx	13,14
09h	EEADR	EEPROM Address Register								xxxx xxxx	13,14
0Ah	PCLATH	—	—	—	Write Buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10
Bank 1											
80h	INDF	Uses Contents of FSR to address Data Memory (not a physical register)								---- ----	11
81h	OPTION_REG	RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0	1111 1111	9
82h	PCL	Low order 8 bits of Program Counter (PC)								0000 0000	11
83h	STATUS (2)	IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C	0001 1xxx	8
84h	FSR	Indirect data memory address pointer 0								xxxx xxxx	11
85h	TRISA	—	—	—	PORTA Data Direction Register					---1 1111	16
86h	TRISB	PORTB Data Direction Register								1111 1111	18
87h	—	Unimplemented location, read as '0'								—	—
88h	EECON1	—	—	—	EEIF	WRERR	WREN	WR	RD	---0 x000	13
89h	EECON2	EEPROM Control Register 2 (not a physical register)								---- ----	14
0Ah	PCLATH	—	—	—	Write buffer for upper 5 bits of the PC(1)					---0 0000	11
0Bh	INTCON	GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF	0000 000x	10

Legend: x = unknown, u = unchanged, - = unimplemented, read as '0', q = value depends on condition

Note 1: The upper byte of the program counter is not directly accessible. PCLATH is a slave register for PC<12:8>. The contents of PCLATH can be transferred to the upper byte of the program counter, but the contents of PC<12:8> are never transferred to PCLATH.

2: The $\overline{TO}$ and $\overline{PD}$ status bits in the STATUS register are not affected by a $\overline{MCLR}$ Reset.

3: Other (non power-up) RESETS include: external RESET through $\overline{MCLR}$ and the Watchdog Timer Reset.

4: On any device RESET, these pins are configured as inputs.

5: This is the value that will be in the port output latch.

2.3.1 STATUS REGISTER

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bit for data memory.

As with any register, the STATUS register can be the destination for any instruction. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to device logic. Furthermore, the $\overline{TO}$ and $\overline{PD}$ bits are not writable. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper three bits and set the Z bit. This leaves the STATUS register as `000u u1uu` (where `u` = unchanged).

Only the `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions should be used to alter the STATUS register (Table 7-2), because these instructions do not affect any status bit.

Note 1: The IRP and RP1 bits (STATUS<7:6>) are not used by the PIC16F84A and should be programmed as cleared. Use of these bits as general purpose R/W bits is NOT recommended, since this may affect upward compatibility with future products.

2: The C and DC bits operate as a $\overline{borrow}$ and digit $\overline{borrow}$ out bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

3: When the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. The specified bit(s) will be updated according to device logic

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7-6 **Unimplemented:** Maintain as '0'

bit 5 **RP0:** Register Bank Select bits (used for direct addressing)
`01` = Bank 1 (80h - FFh)
`00` = Bank 0 (00h - 7Fh)

bit 4 **$\overline{TO}$:** Time-out bit
`1` = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
`0` = A WDT time-out occurred

bit 3 **$\overline{PD}$:** Power-down bit
`1` = After power-up or by the `CLRWDT` instruction
`0` = By execution of the `SLEEP` instruction

bit 2 **Z:** Zero bit
`1` = The result of an arithmetic or logic operation is zero
`0` = The result of an arithmetic or logic operation is not zero

bit 1 **DC:** Digit carry/$\overline{borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for $\overline{borrow}$, the polarity is reversed)
`1` = A carry-out from the 4th low order bit of the result occurred
`0` = No carry-out from the 4th low order bit of the result

bit 0 **C:** Carry/$\overline{borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for $\overline{borrow}$, the polarity is reversed)
`1` = A carry-out from the Most Significant bit of the result occurred
`0` = No carry-out from the Most Significant bit of the result occurred

Note: A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

2.3.1 STATUS REGISTER

The STATUS register contains the arithmetic status of the ALU, the RESET status and the bank select bit for data memory.

As with any register, the STATUS register can be the destination for any instruction. If the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. These bits are set or cleared according to device logic. Furthermore, the $\overline{TO}$ and $\overline{PD}$ bits are not writable. Therefore, the result of an instruction with the STATUS register as destination may be different than intended.

For example, `CLRF STATUS` will clear the upper three bits and set the Z bit. This leaves the STATUS register as `000u u1uu` (where `u` = unchanged).

Only the `BCF`, `BSF`, `SWAPF` and `MOVWF` instructions should be used to alter the STATUS register (Table 7-2), because these instructions do not affect any status bit.

Note 1: The IRP and RP1 bits (STATUS<7:6>) are not used by the PIC16F84A and should be programmed as cleared. Use of these bits as general purpose R/W bits is NOT recommended, since this may affect upward compatibility with future products.

2: The C and DC bits operate as a $\overline{borrow}$ and digit borrow out bit, respectively, in subtraction. See the `SUBLW` and `SUBWF` instructions for examples.

3: When the STATUS register is the destination for an instruction that affects the Z, DC or C bits, then the write to these three bits is disabled. The specified bit(s) will be updated according to device logic

REGISTER 2-1: STATUS REGISTER (ADDRESS 03h, 83h)

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7-6 **Unimplemented:** Maintain as '0'

bit 5 **RP0**: Register Bank Select bits (used for direct addressing)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)

bit 4 **$\overline{TO}$**: Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 3 **$\overline{PD}$**: Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

bit 2 **Z**: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 **DC**: Digit carry/$\overline{borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 **C**: Carry/$\overline{borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions) (for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high or low order bit of the source register.

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
- n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

National Semiconductor

June 1989

54157/DM54157/DM74157
Quad 2-Line to 1-Line Data Selectors/Multiplexers

General Description

These data selectors/multiplexers contain inverters and drivers to supply full on-chip data selection to the four output gates. A separate strobe input is provided. A 4-bit word is selected from one of two sources and is routed to the four outputs.

Features

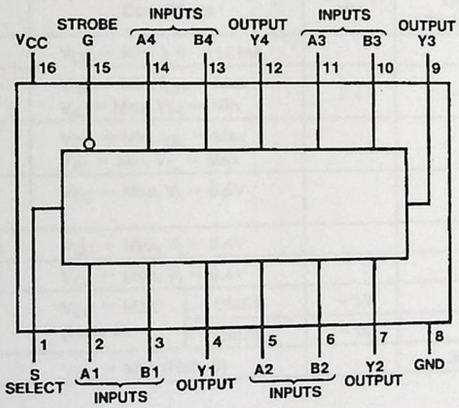
- Buffered inputs and outputs
- Typical propagation time 9 ns
- Typical power dissipation 150 mW
- Alternate Military/Aerospace device (54157) is available. Contact a National Semiconductor Sales Office/ Distributor for specifications.

Applications

- Expand any data input point
- Multiplex dual data buses
- Generate four functions of two variables (one variable is common)
- Source programmable counters

Connection Diagram

Dual-In-Line Package



TL/F/6550–1

Order Number 54157DMQB, 54157FMQB, DM54157J, DM54157W or DM74157N
See NS Package Number J16A, N16E or W16A

Function Table

Inputs				Output Y
Strobe	Select	A	B	
H	X	X	X	L
L	L	L	X	L
L	L	H	X	H
L	H	X	L	L
L	H	X	H	H

H = High Level, L = Low Level, X = Don't Care

RRD-B30M105/Printed in U. S. A.

Absolute Maximum Ratings (Note)

If Military/Aerospace specified devices are required, please contact the National Semiconductor Sales Office/Distributors for availability and specifications.

Supply Voltage	7V
Input Voltage	5.5V
Operating Free Air Temperature Range	
DM54 and 54	−55°C to +125°C
DM74	0°C to +70°C
Storage Temperature Range	−65°C to +150°C

Note: *The "Absolute Maximum Ratings" are those values beyond which the safety of the device cannot be guaranteed. The device should not be operated at these limits. The parametric values defined in the "Electrical Characteristics" table are not guaranteed at the absolute maximum ratings. The "Recommended Operating Conditions" table will define the conditions for actual device operation.*

Recommended Operating Conditions

Symbol	Parameter	DM54157			DM74157			Units
		Min	Nom	Max	Min	Nom	Max	
V_{CC}	Supply Voltage	4.5	5	5.5	4.75	5	5.25	V
V_{IH}	High Level Input Voltage	2			2			V
V_{IL}	Low Level Input Voltage			0.8			0.8	V
I_{OH}	High Level Output Current			−0.8			−0.8	mA
I_{OL}	Low Level Output Current			16			16	mA
T_A	Free Air Operating Temperature	−55		125	0		70	°C

Electrical Characteristics over recommended operating free air temperature range (unless otherwise noted)

Symbol	Parameter	Conditions		Min	Typ (Note 1)	Max	Units
V_I	Input Clamp Voltage	V_{CC} = Min, I_I = −12 mA				−1.5	V
V_{OH}	High Level Output Voltage	V_{CC} = Min, I_{OH} = Max V_{IL} = Max, V_{IH} = Min		2.4	3.4		V
V_{OL}	Low Level Output Voltage	V_{CC} = Min, I_{OL} = Max V_{IH} = Min, V_{IL} = Max				0.4	V
I_I	Input Current @ Max Input Voltage	V_{CC} = Max, V_I = 5.5V				1	mA
I_{IH}	High Level Input Current	V_{CC} = Max, V_I = 2.4V				40	μA
I_{IL}	Low Level Input Current	V_{CC} = Max, V_I = 0.4V				−1.6	mA
I_{OS}	Short Circuit Output Current	V_{CC} = Max (Note 2)	DM54	−20		−55	mA
			DM74	−18		−55	
I_{CC}	Supply Current	V_{CC} = Max (Note 3)			30	48	mA

Note 1: All typicals are at V_{CC} = 5V, T_A = 25°C.

Note 2: Not more than one output should be shorted at a time.

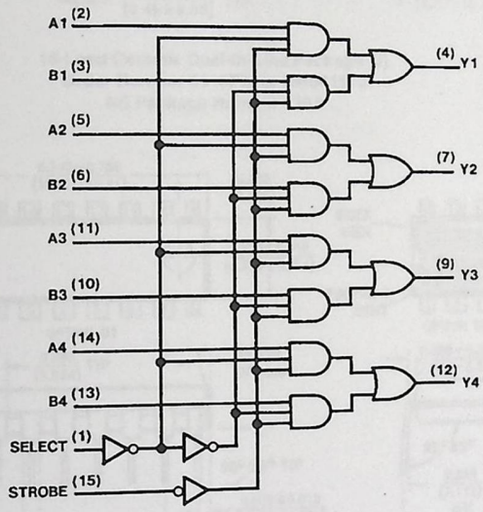
Note 3: I_{CC} is measured with 4.5V applied to all inputs and all outputs open.

Switching Characteristics at $V_{CC} = 5V$ and $T_A = 25°C$ (See Section 1 for Test Waveforms and Output Load)

Symbol	Parameter	From (Input) To (Output)	$R_L = 400\Omega$, $C_L = 15$ pF		Units
			Min	Max	
t_{PLH}	Propagation Delay Time Low to High Level Output	Data to Y		14	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	Data to Y		14	ns
t_{PLH}	Propagation Delay Time Low to High Level Output	Strobe to Y		20	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	Strobe to Y		21	ns
t_{PLH}	Propagation Delay Time Low to High Level Output	Select to Y		23	ns
t_{PHL}	Propagation Delay Time High to Low Level Output	Select to Y		27	ns

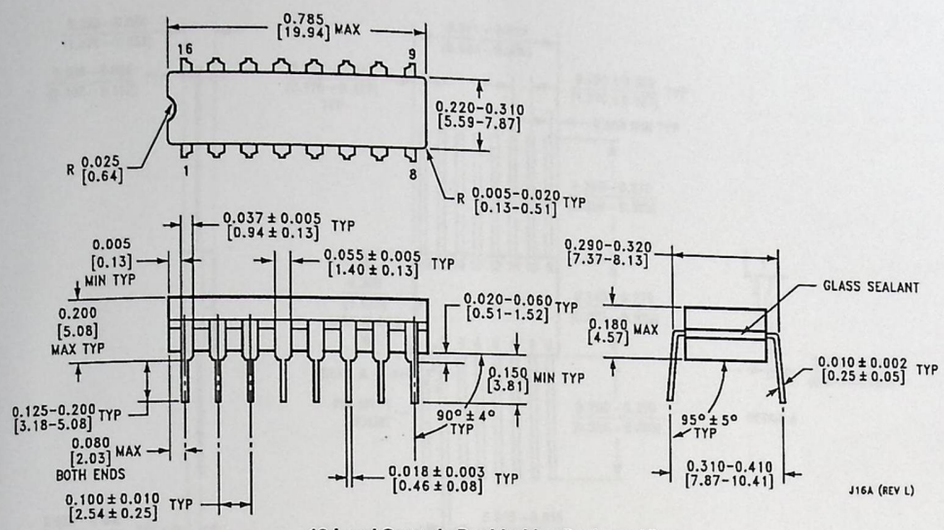
Logic Diagram

157

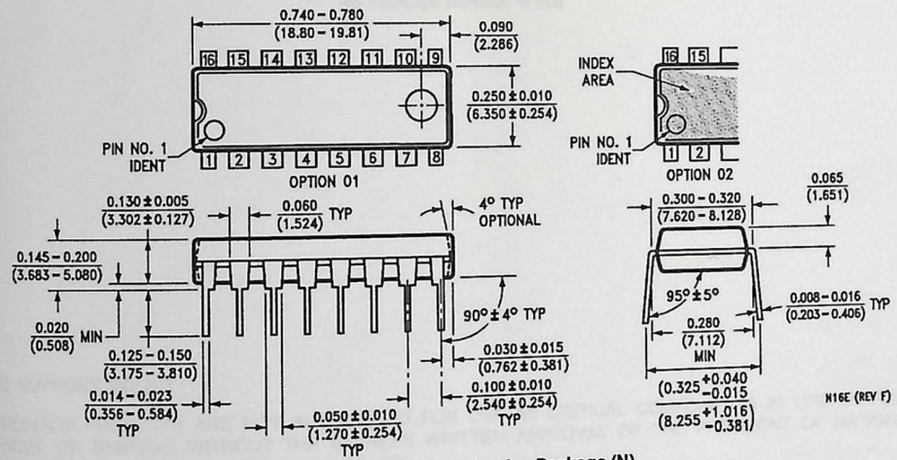


TL/F/6550–2

Physical Dimensions inches (millimeters)

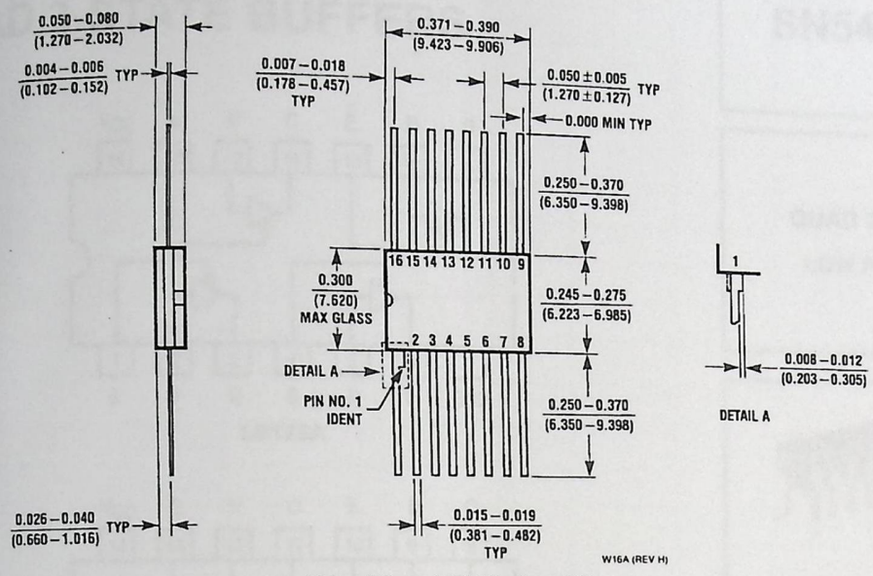


16-Lead Ceramic Dual-In-Line Package (J)
Order Number 54157W or DM54157J
NS Package Number J16A



16-Lead Molded Dual-In-Line Package (N)
Order Number DM74157N
NS Package Number N16E

Physical Dimensions inches (millimeters) (Continued)



16-Lead Ceramic Flat Package (W)
Order Number 54157FMQB or DM54157W
NS Package Number W16A

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

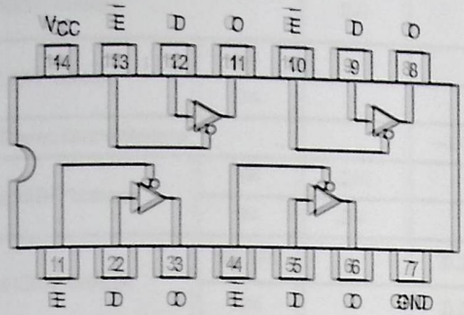
National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-8960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408

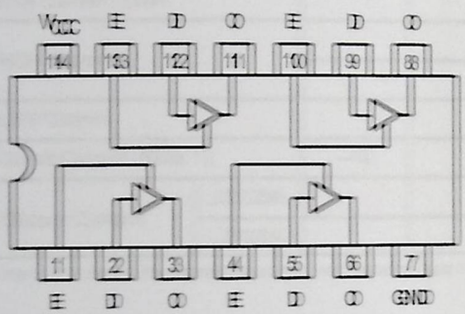
MOTOROLA

QUAD 3-STATE BUFFERS

SN54/74LS125A
SN54/74LS126A



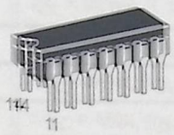
LS125A



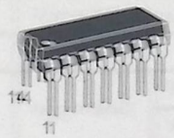
LS126A

QUAD 3-STATE BUFFERS

LOW POWER SCHOTTKY



J SUFFIX
CERAMIC
CASE 632-08



N SUFFIX
PLASTIC
CASE 646-06

D SUFFIX
SOIC
CASE 751A-02

ORDERING INFORMATION

SN54LSXXXJ Ceramic
SN74LSXXXN Plastic
SN74LSXXXD SOIC

TRUTH TABLES

LS125A

INPUTS		OUTPUT
E	D	
L	L	L
L	H	H
H	X	(Z)

LS126A

INPUTS		OUTPUT
E	D	
H	L	L
H	H	H
L	X	(Z)

L = LOW Voltage Level
H = HIGH Voltage Level
X = Don't Care
(Z) = High Impedance (off)

GUARANTEED OPERATING RANGES

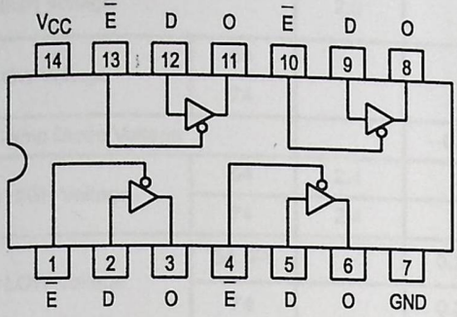
Symbol	Parameter		Min	Typ	Max	Unit
V_{CC}	Supply Voltage	54 74	4.5 4.75	5.0 5.0	5.5 5.25	V
T_A	Operating Ambient Temperature Range	54 74	−55 0	25 25	125 70	°C
I_{OH}	Output Current — High	54 74			−1.0 −2.6	mA
I_{OL}	Output Current — Low	54 74			12 24	mA

MOTOROLA

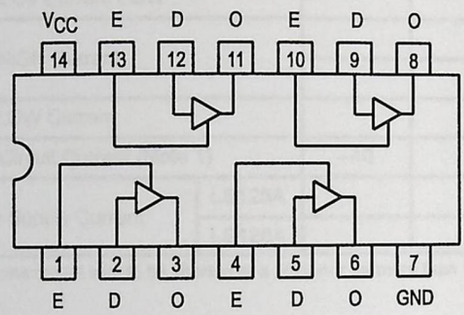
QUAD 3-STATE BUFFERS

SN54/74LS125A
SN54/74LS126A

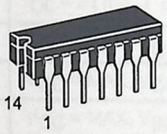
QUAD 3-STATE BUFFERS
LOW POWER SCHOTTKY



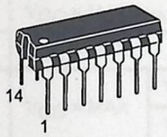
LS125A



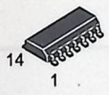
LS126A



J SUFFIX
CERAMIC
CASE 632-08



N SUFFIX
PLASTIC
CASE 646-06



D SUFFIX
SOIC
CASE 751A-02

TRUTH TABLES

LS125A

INPUTS		OUTPUT
$\overline{E}$	D	
L	L	L
L	H	H
H	X	(Z)

LS126A

INPUTS		OUTPUT
E	D	
H	L	L
H	H	H
L	X	(Z)

L = LOW Voltage Level
H = HIGH Voltage Level
X = Don't Care
(Z) = High Impedance (off)

ORDERING INFORMATION

SN54LSXXXJ	Ceramic
SN74LSXXXN	Plastic
SN74LSXXXD	SOIC

GUARANTEED OPERATING RANGES

Symbol	Parameter		Min	Typ	Max	Unit
V_{CC}	Supply Voltage	54 74	4.5 4.75	5.0 5.0	5.5 5.25	V
T_A	Operating Ambient Temperature Range	54 74	−55 0	25 25	125 70	°C
I_{OH}	Output Current — High	54 74			−1.0 −2.6	mA
I_{OL}	Output Current — Low	54 74			12 24	mA

DC CHARACTERISTICS OVER OPERATING TEMPERATURE RANGE (unless otherwise specified)

Symbol	Parameter		Limits Min	Limits Typ	Limits Max	Unit	Test Conditions	
V_{IH}	Input HIGH Voltage		2.0			V	Guaranteed Input HIGH Voltage for All Inputs	
V_{IL}	Input LOW Voltage	54			0.7	V	Guaranteed Input LOW Voltage for All Inputs	
		74			0.8			
V_{IK}	Input Clamp Diode Voltage			−0.65	−1.5	V	V_{CC} = MIN, I_{IN} = −18 mA	
V_{OH}	Output HIGH Voltage	54	2.4			V	V_{CC} = MIN, I_{OH} = MAX, V_{IN} = V_{IH} or V_{IL} per Truth Table	
		74	2.4			V		
V_{OL}	Output LOW Voltage	54, 74		0.25	0.4	V	I_{OL} = 12 mA	V_{CC} = V_{CC} MIN, V_{IN} = V_{IL} or V_{IH} per Truth Table
		74		0.35	0.5	V	I_{OL} = 24 mA	
I_{OZH}	Output Off Current HIGH				20	μA	V_{CC} = MAX, V_{OUT} = 2.4 V	
I_{OZL}	Output Off Current LOW				−20	μA	V_{CC} = MAX, V_{OUT} = 0.4 V	
I_{IH}	Input HIGH Current				20	μA	V_{CC} = MAX, V_{IN} = 2.7 V	
					0.1	mA	V_{CC} = MAX, V_{IN} = 7.0 V	
I_{IL}	Input LOW Current				−0.4	mA	V_{CC} = MAX, V_{IN} = 0.4 V	
I_{OS}	Short Circuit Current (Note 1)		−40		−225	mA	V_{CC} = MAX	
I_{CC}	Power Supply Current	LS125A			20	mA	V_{CC} = MAX	V_{IN} = 0 V, V_E = 4.5 V
		LS126A			22			V_{IN} = 0 V, V_E = 0 V

Note 1: Not more than one output should be shorted at a time, nor for more than 1 second.

AC CHARACTERISTICS (T_A = 25°C)

Symbol	Parameter		Limits Min	Limits Typ	Limits Max	Unit	Test Conditions	
t_{PLH}	Propagation Delay, Data to Output	LS125A		9.0	15	ns	Figure 2	V_{CC} = 5.0 V, C_L = 45 pF, R_L = 667 Ω
t_{PLH}		LS126A		9.0	15			
t_{PHL}		LS125A		7.0	18			
t_{PHL}		LS126A		8.0	18			
t_{PZH}	Output Enable Time to HIGH Level	LS125A		12	20	ns	Figures 4, 5	
		LS126A		16	25			
t_{PZL}	Output Enable Time to LOW Level	LS125A		15	25	ns	Figures 3, 5	
		LS126A		21	35			
t_{PHZ}	Output Disable Time from HIGH Level	LS125A			20	ns	Figures 4, 5	V_{CC} = 5.0 V, C_L = 5.0 pF, R_L = 667 Ω
		LS126A			25			
t_{PLZ}	Output Disable Time from LOW Level	LS125A			20	ns	Figures 3, 5	
		LS126A			25			

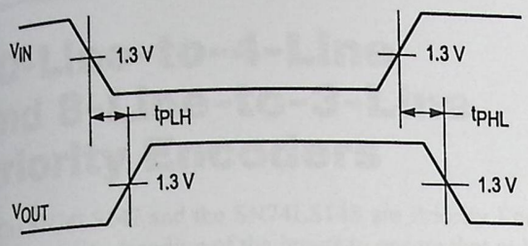


Figure 1

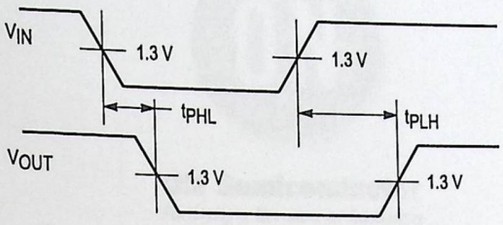


Figure 2

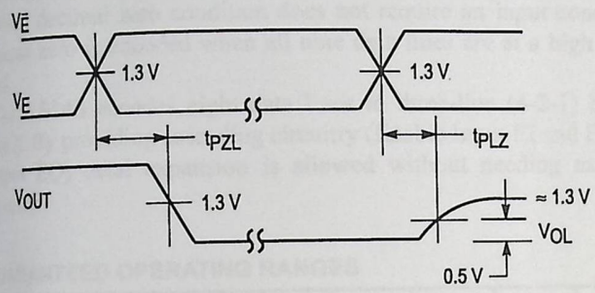


Figure 3

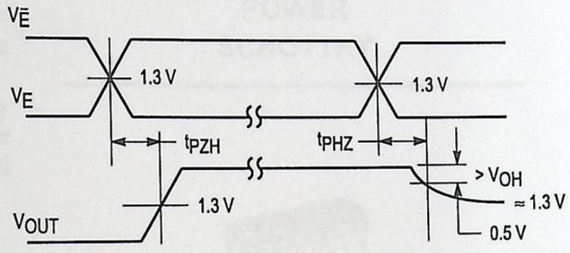


Figure 4

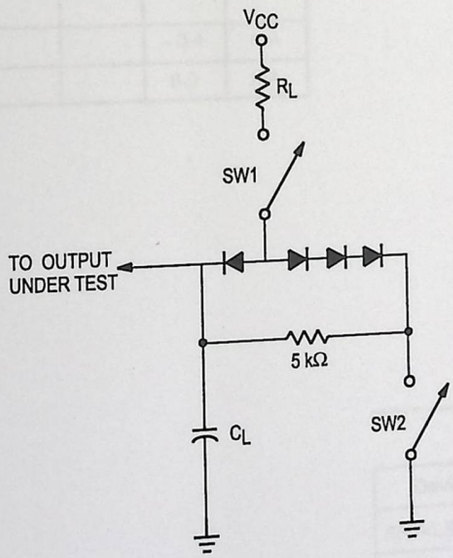


Figure 5

SWITCH POSITIONS

SYMBOL	SW1	SW2
t_{PZH}	Open	Closed
t_{PZL}	Closed	Open
t_{PLZ}	Closed	Closed
t_{PHZ}	Closed	Closed

SN74LS147 SN74LS148

10-Line-to-4-Line and 8-Line-to-3-Line Priority Encoders

The SN74LS147 and the SN74LS148 are Priority Encoders. They provide priority decoding of the inputs to ensure that only the highest order data line is encoded. Both devices have data inputs and outputs which are active at the low logic level.

The LS147 encodes nine data lines to four-line (8-4-2-1) BCD. The implied decimal zero condition does not require an input condition because zero is encoded when all nine data lines are at a high logic level.

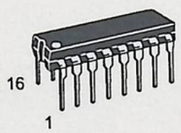
The LS148 encodes eight data lines to three-line (4-2-1) binary (octal). By providing cascading circuitry (Enable Input EI and Enable Output EO) octal expansion is allowed without needing external circuitry.

GUARANTEED OPERATING RANGES

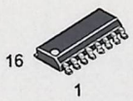
Symbol	Parameter	Min	Typ	Max	Unit
V_{CC}	Supply Voltage	4.75	5.0	5.25	V
T_A	Operating Ambient Temperature Range	0	25	70	°C
I_{OH}	Output Current – High			–0.4	mA
I_{OL}	Output Current – Low			8.0	mA

ON Semiconductor
Formerly a Division of Motorola
http://onsemi.com

LOW POWER SCHOTTKY



PLASTIC
N SUFFIX
CASE 648



SOIC
D SUFFIX
CASE 751B

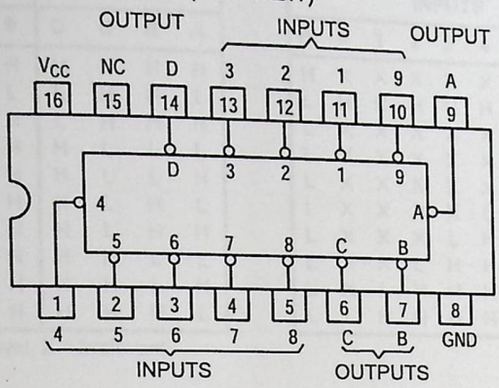
ORDERING INFORMATION

Device	Package	Shipping
SN74LS147N	16 Pin DIP	2000 Units/Box
SN74LS147D	16 Pin	2500/Tape & Reel
SN74LS148N	16 Pin DIP	2000 Units/Box
SN74LS148D	16 Pin	2500/Tape & Reel

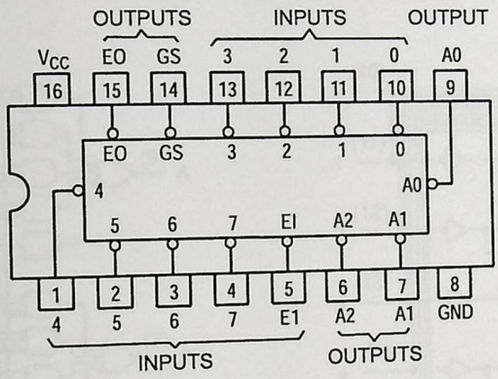
Publication Order Number:
SN74LS147/D

December, 1999 – Rev. 6

SN74LS147
(TOP VIEW)



SN74LS148
(TOP VIEW)



SN74LS147
FUNCTION TABLE

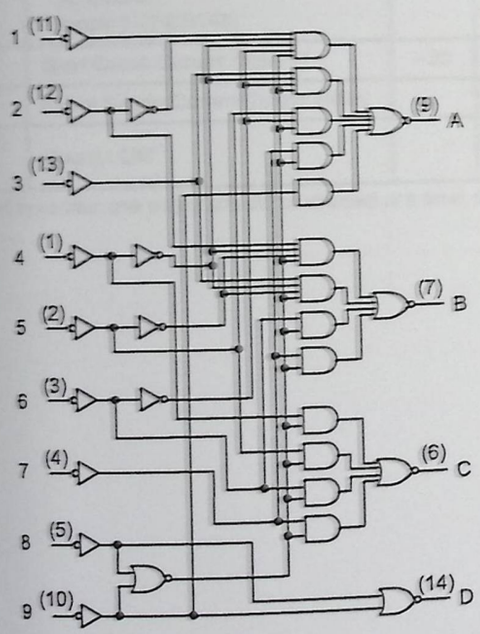
INPUTS									OUTPUTS			
1	2	3	4	5	6	7	8	9	D	C	B	A
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

H = HIGH Logic Level, L = LOW Logic Level, X = Irrelevant

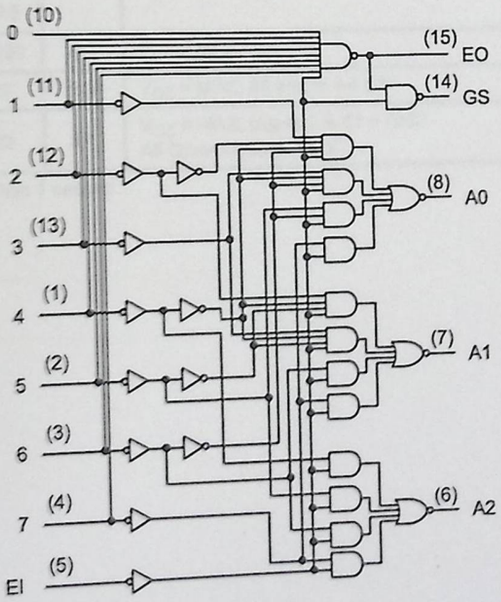
SN74LS148
FUNCTION TABLE

INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

FUNCTIONAL BLOCK DIAGRAMS



SN74LS147



SN74LS148

SN74LS147
FUNCTION TABLE

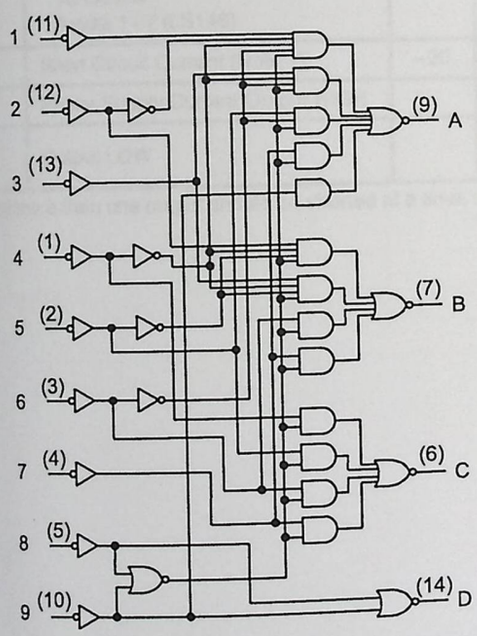
INPUTS									OUTPUTS			
1	2	3	4	5	6	7	8	9	D	C	B	A
H	H	H	H	H	H	H	H	H	H	H	H	H
X	X	X	X	X	X	X	X	L	L	H	H	L
X	X	X	X	X	X	X	L	H	L	H	H	H
X	X	X	X	X	X	L	H	H	H	L	L	L
X	X	X	X	X	L	H	H	H	H	L	L	H
X	X	X	X	L	H	H	H	H	H	L	H	L
X	X	X	L	H	H	H	H	H	H	L	H	H
X	X	L	H	H	H	H	H	H	H	H	L	L
X	L	H	H	H	H	H	H	H	H	H	L	H
L	H	H	H	H	H	H	H	H	H	H	H	L

H = HIGH Logic Level, L = LOW Logic Level, X = Irrelevant

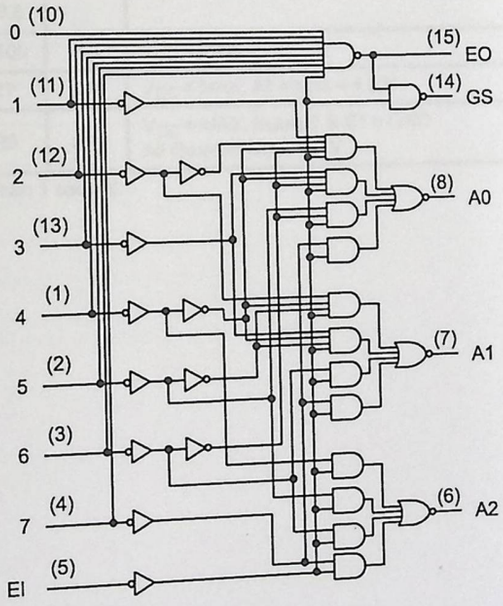
SN74LS148
FUNCTION TABLE

INPUTS									OUTPUTS				
EI	0	1	2	3	4	5	6	7	A2	A1	A0	GS	EO
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

FUNCTIONAL BLOCK DIAGRAMS



SN74LS147



SN74LS148

DC CHARACTERISTICS OVER OPERATING TEMPERATURE RANGE (unless otherwise specified)

Symbol	Parameter	Limits			Unit	Test Conditions	
		Min	Typ	Max			
V_{IH}	Input HIGH Voltage	2.0			V	Guaranteed Input HIGH Voltage for All Inputs	
V_{IL}	Input LOW Voltage			0.8	V	Guaranteed Input LOW Voltage for All Inputs	
V_{IK}	Input Clamp Diode Voltage		−0.65	−1.5	V	V_{CC} = MIN, I_{IN} = −18 mA	
V_{OH}	Output HIGH Voltage	2.7	3.5		V	V_{CC} = MIN, I_{OH} = MAX, V_{IN} = V_{IH} or V_{IL} per Truth Table	
V_{OL}	Output LOW Voltage		0.25	0.4	V	I_{OL} = 4.0 mA	V_{CC} = V_{CC} MIN, V_{IN} = V_{IL} or V_{IH} per Truth Table
			0.35	0.5	V	I_{OL} = 8.0 mA	
I_{IH}	Input HIGH Current All Others Inputs 1−7 (LS148)			20 40	µA	V_{CC} = MAX, V_{IN} = 2.7 V	
	All Others Inputs 1−7 (LS148)			0.1 0.2	mA	V_{CC} = MAX, V_{IN} = 7.0 V	
I_{IL}	Input LOW Current All Others Inputs 1−7 (LS148)			−0.4 −0.8	mA	V_{CC} = MAX, V_{IN} = 0.4 V	
I_{OS}	Short Circuit Current (Note 1)	−20		−100	mA	V_{CC} = MAX	
I_{CCH}	Power Supply Current Output HIGH			17	mA	V_{CC} = MAX, All Inputs = 4.5 V	
I_{CCL}	Output LOW			20	mA	V_{CC} = MAX, Inputs 7 & E1 = GND All Other Inputs = 4.5 V	

Note 1: Not more than one output should be shorted at a time, nor for more than 1 second.

AC CHARACTERISTICS (V_{CC} = 5.0 V, T_A = 25°C)

SN74LS147

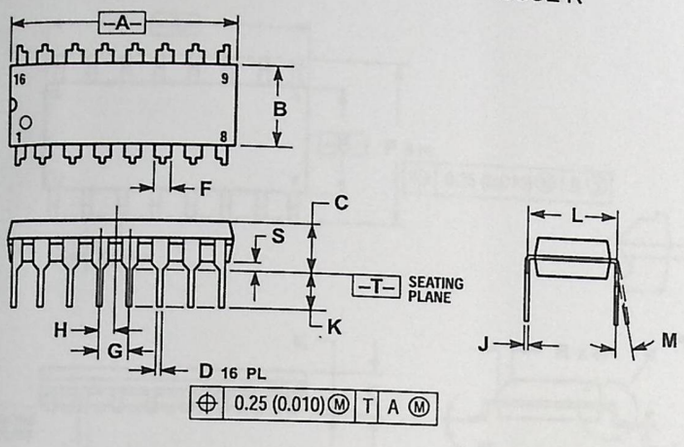
Symbol	From (Input)	To (Output)	Waveform	Limits			Unit	Test Conditions
				Min	Typ	Max		
t_{PLH}	Any	Any	In-phase output		12	18	ns	C_L = 15 pF, R_L = 2.0 kΩ
t_{PHL}					12	18		
t_{PLH}	Any	Any	Out-of-phase output		21	33	ns	
t_{PHL}					15	23		

SN74LS148

Symbol	From (Input)	To (Output)	Waveform	Limits			Unit	Test Conditions
				Min	Typ	Max		
t_{PLH}	1 thru 7	A0, A1, or A2	In-phase output		14	18	ns	C_L = 15 pF, R_L = 2.0 kΩ
t_{PHL}					15	25		
t_{PLH}	1 thru 7	A0, A1, or A2	Out-of-phase output		20	36	ns	
t_{PHL}					16	29		
t_{PLH}	0 thru 7	EO	Out-of-phase output		7.0	18	ns	
t_{PHL}					25	40		
t_{PLH}	0 thru 7	GS	In-phase output		35	55	ns	
t_{PHL}					9.0	21		
t_{PLH}	EI	A0, A1, or A2	In-phase output		16	25	ns	
t_{PHL}					12	25		
t_{PLH}	EI	GS	In-phase output		12	17	ns	
t_{PHL}					14	36		
t_{PLH}	EI	EO	In-phase output		12	21	ns	(LS148)
t_{PHL}					28 30	40 45		

PACKAGE DIMENSIONS

N SUFFIX
PLASTIC PACKAGE
CASE 648–08
ISSUE R

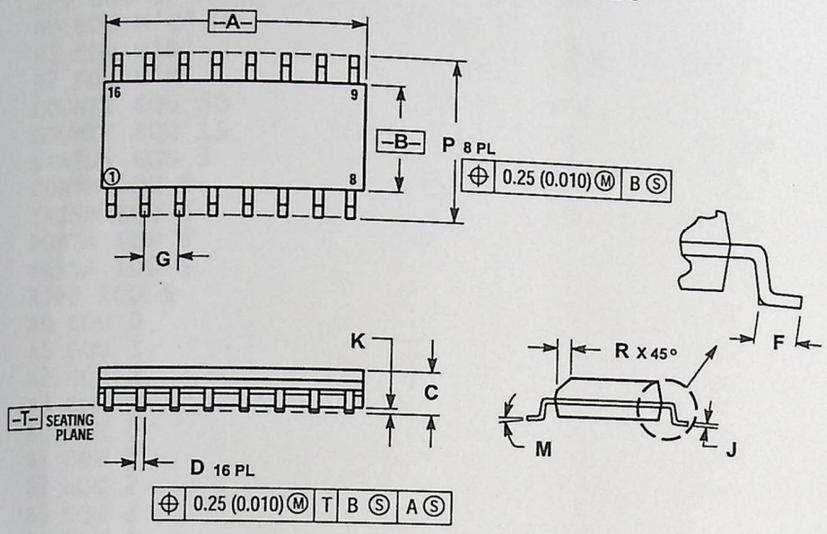


NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: INCH.
3. DIMENSION L TO CENTER OF LEADS WHEN FORMED PARALLEL.
4. DIMENSION B DOES NOT INCLUDE MOLD FLASH.
5. ROUNDED CORNERS OPTIONAL.

	INCHES		MILLIMETERS	
DIM	MIN	MAX	MIN	MAX
A	0.740	0.770	18.80	19.55
B	0.250	0.270	6.35	6.85
C	0.145	0.175	3.69	4.44
D	0.015	0.021	0.39	0.53
F	0.040	0.70	1.02	1.77
G	0.100 BSC		2.54 BSC	
H	0.050 BSC		1.27 BSC	
J	0.008	0.015	0.21	0.38
K	0.110	0.130	2.80	3.30
L	0.295	0.305	7.50	7.74
M	0°	10°	0°	10°
S	0.020	0.040	0.51	1.01

PACKAGE DIMENSIONS

D SUFFIX
PLASTIC SOIC PACKAGE
CASE 751B–05
ISSUE J



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION.
4. MAXIMUM MOLD PROTRUSION 0.15 (0.006) PER SIDE.
5. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.127 (0.005) TOTAL IN EXCESS OF THE D DIMENSION AT MAXIMUM MATERIAL CONDITION.

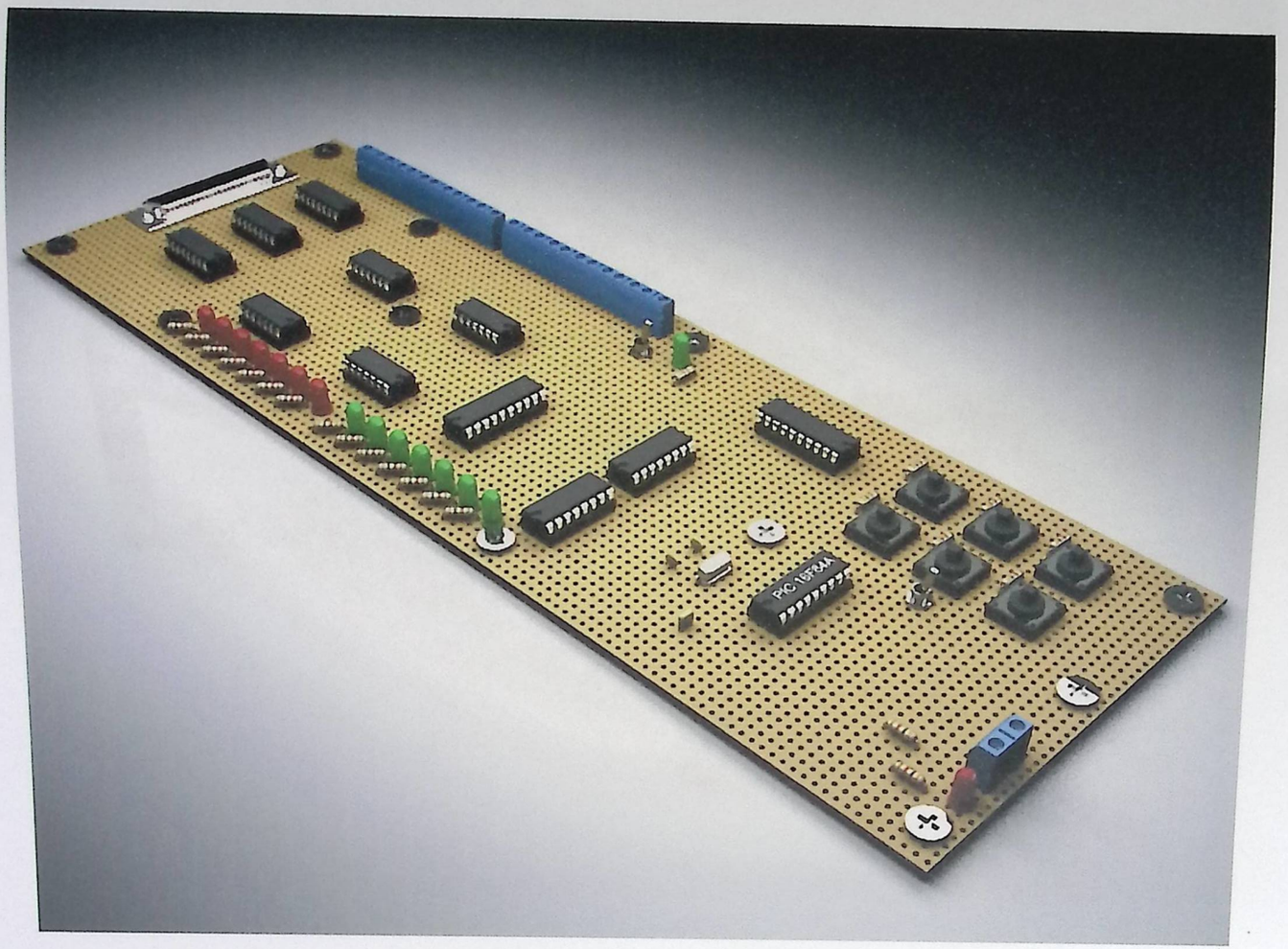
	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	9.80	10.00	0.386	0.393
B	3.80	4.00	0.150	0.157
C	1.35	1.75	0.054	0.068
D	0.35	0.49	0.014	0.019
F	0.40	1.25	0.016	0.049
G	1.27 BSC		0.050 BSC	
J	0.19	0.25	0.008	0.009
K	0.10	0.25	0.004	0.009
M	0°	7°	0°	7°
P	5.80	6.20	0.229	0.244
R	0.25	0.50	0.010	0.019

APPENDIX C: MICROCONTROLLER PROGRAM CODE

```
                LIST C=80,N=60,P=16F84,R=DEC
                TITLE "WCS.ASM"

RP0 EQU 5
R0 EQU H'C'
R1 EQU H'D'
R2 EQU H'E'
COUNT1 EQU 50
COUNT2 EQU 15
STATUS EQU 3
PORTB EQU 6
TRISB EQU 6
PORTA EQU 5
TRISA EQU 5
RJP0 EQU 5
A0 EQU 0
A1 EQU 1
A2 EQU 2
A3 EQU 3
B0 EQU 0
B1 EQU 1
B2 EQU 2
B3 EQU 3
B4 EQU 4
B5 EQU 5
B6 EQU 6
B7 EQU 7

                ORG 0
                BSF STATUS, RP0
                MOVLW B'11000000'
                MOVWF TRISB
                MOVLW B'11111111'
                MOVWF TRISA
                BCF STATUS, RP0
LOOP:

                                ;CHECK IF A0 IS PRESSED OR NOT.....
                BTFSC PORTA, A0 ;SKIP NEXT INSTRUCTION IF A0 IS PRESSED
                GOTO CHECKA1    ;SET THE VALUE OF B0 = 0
                BTFSC PORTB, B0 ;COMPLEMENT THE VALUE OF B0
                GOTO SET_ZERO1
                GOTO SET_ONE1
SET_ZERO1:      BCF PORTB, B0
                GOTO AFTER1
SET_ONE1:       BSF PORTB, B0
AFTER1:         BCF PORTB,B1        ;SET B1 = 0 FOR POSITIVE DIERCTION

                                ;CHECK IF A1 IS PRESSED OR NOT
CHECKA1:        BTFSC PORTA, A1     ;SKIP NEXT INSTURCTION IF A1 IS PRESSED
                GOTO CHECKA2        ;SET B0 = 0 BUTTON IS NOT PRESSED
                BTFSC PORTB, B0     ;COMPLEMENT THE VALUE OF B0
                GOTO SET_ZERO2
                GOTO SET_ONE2
SET_ZERO2:      BCF PORTB, B0
                GOTO AFTER2
SET_ONE2:       BSF PORTB, B0
AFTER2:         BSF PORTB, B1       ;FOR MOVING NEGATIVE DIRECTION

                                    ;CHECK IF A2 IS PRESSED OR NOT
```

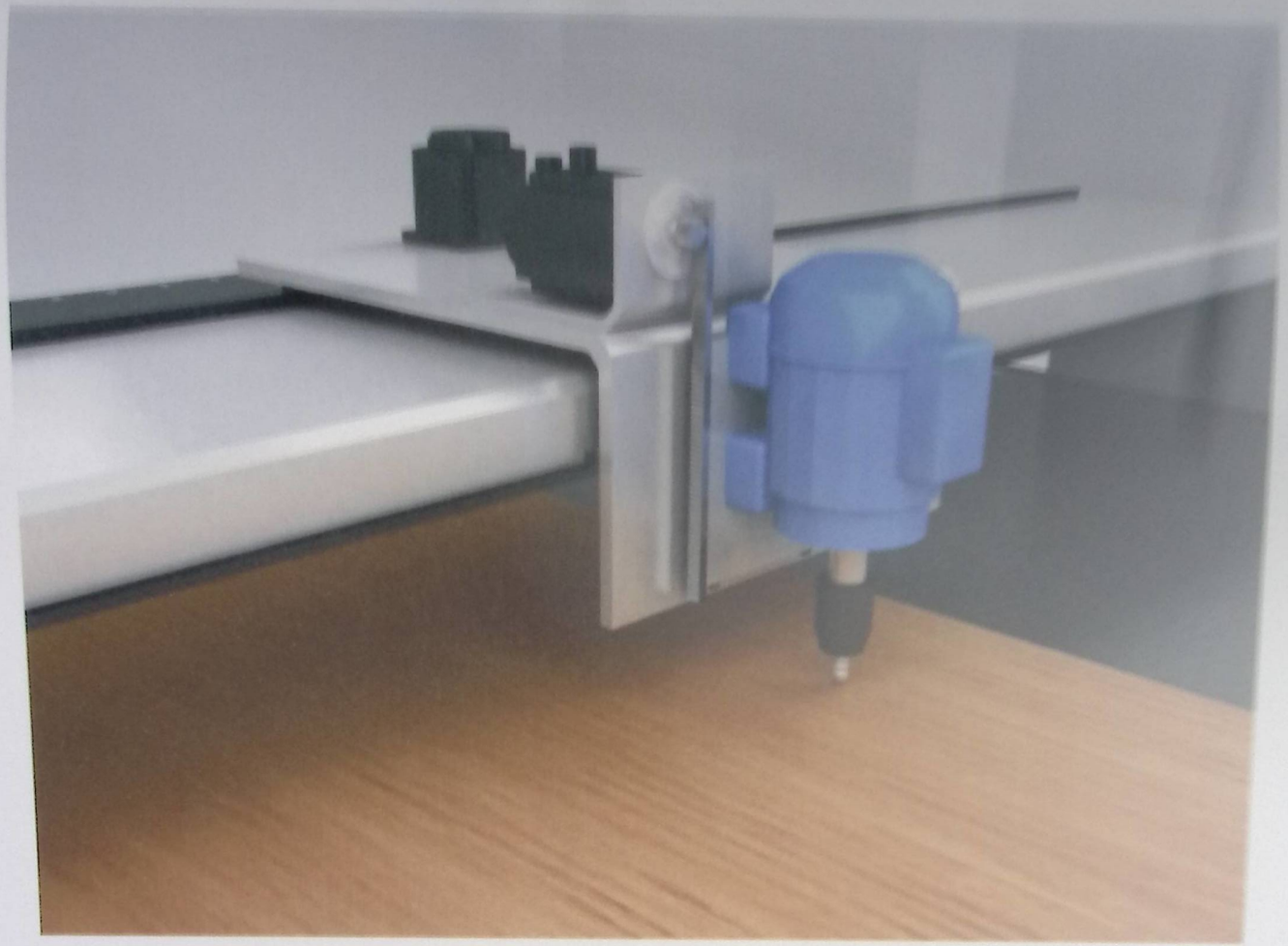
```
CHECKA2:    BTFSC PORTA, A2      ;SKIP NEXT INSTRUCTION IF A2 IS PRESSED
            GOTO CHECKA3         ;SET B2 = 0 IF THE A2 IS NOT PRESSED

            BTFSC PORTB, B2      ;COMPLEMENT B2
            GOTO SET_ZERO3
            GOTO SET_ONE3
SET_ZERO3:  BCF PORTB, B2
            GOTO AFTER3
SET_ONE3:   BSF PORTB, B2
AFTER3:     BCF PORTB, B3        ;MOVING IN THE POSITIVE DIRECTION

CHECKA3:    BTFSC PORTA, A3      ;CHECK IF A3 IS PRESSED OR NOT
            GOTO CHECKB6         ;SKIP NEXT INSTRUCTOIN IF A3 IS PRESSED
                                 ;GOTO TO SET B2 = 0 IF A3 IS NOT PRESSED
            BTFSC PORTB, B2      ;COMPLEMENT B2
            GOTO SET_ZERO4
            GOTO SET_ONE4
SET_ZERO4:  BCF PORTB, B2
            GOTO AFTER4
SET_ONE4:   BSF PORTB, B2
AFTER4:     BSF PORTB, B3

                                 ;CHECK IF B6 IS PRESSED OR NOT
CHECKB6:    BTFSC PORTB, B6      ;SKIP NEXT INSTRUCITON IF B6 IS PRESSED
            GOTO CHECKB7         ;CLEAR B4 IF THE BUTTON IS NOT PRESSED

            BTFSC PORTB, B4      ;COMPLEMENT THE VALUE OF B4
            GOTO SET_ZERO5
            GOTO SET_ONE5
SET_ZERO5:  BCF PORTB, B4
            GOTO AFTER5
SET_ONE5:   BSF PORTB, B4
AFTER5:     BCF PORTB, B5        ;FOR MOVING IN THE NEGATIVE DIRECTION

                                 ;CHECK IF B7 IS PRESSED OR NOT
CHECKB7:    BTFSC PORTB, B7      ;SKIP NEXT INSTRUCTION IF B7 IS PRESSED
            GOTO NEXT            ;CLEAR B4 IF B7 IS NOT PRESSED

            BTFSC PORTB, B4      ;COMPLEMENT B4
            GOTO SET_ZERO6
            GOTO SET_ONE6
SET_ZERO6:  BCF PORTB, B4
            GOTO AFTER6
SET_ONE6:   BSF PORTB, B4
AFTER6:     BSF PORTB, B5        ;THE NEGATIVE DIRECTION

NEXT:       CALL DELAY
            GOTO LOOP


DELAY:      MOVWF R0
DELAY1:     MOVLW COUNT1
            MOVWF R1
DELAY2:     MOVLW COUNT2
            MOVWF R2
            GOTO DELAY3
DELAY3:     DECFSZ R1
            GOTO DELAY2
            DECFSZ R0
            GOTO DELAY1
            RETURN
            END
```

APPENDIX D
Interfacing Circuit



Carving Machine General Overview

Carving Machine Close View



Carving Machine Close View