DEVELOPMENT OF MOTOR CONTROL USING GRAPHICAL USER INTERFACE

KHAIRUL ANUAR BIN ARIS

UNIVERSITY MALAYSIA PAHANG

UNIVERSITI MALAYSIA PAHANG

BORANG PENGESAHAN STATUS TESIS♦

JUDUL: **DEVELOPMENT OF MOTOR CONTROL USING GRAPHICAL USER INTERFACE**

SESI PENGAJIAN: **2007/2008**

Saya **KHAIRUL ANUAR BIN ARIS (850811-11-5225)**
(HURUF BESAR)

mengaku membenarkan tesis (Sarjana Muda/~~Sarjana~~ /~~Doktor Falsafah~~)* ini disimpan di Perpustakaan dengan syarat-syarat kegunaan seperti berikut:

1. Tesis adalah hakmilik Universiti Malaysia Pahang.
2. Perpustakaan dibenarkan membuat salinan untuk tujuan pengajian sahaja.
3. Perpustakaan dibenarkan membuat salinan tesis ini sebagai bahan pertukaran antara institusi pengajian tinggi.
4. **Sila tandakan (√)

☐	**SULIT**	(Mengandungi maklumat yang berdarjah keselamatan atau kepentingan Malaysia seperti yang termaktub di dalam AKTA RAHSIA RASMI 1972)
☐	**TERHAD**	(Mengandungi maklumat TERHAD yang telah ditentukan oleh organisasi/badan di mana penyelidikan dijalankan)
√	**TIDAK TERHAD**	

Disahkan oleh:

_____ (TANDATANGAN PENULIS)

_____ (TANDATANGAN PENYELIA)

Alamat Tetap:

NO 833 JALAN LIMBONG, KG. LIMBONG, 24000 CUKAI, KEMAMAN TERENGGANU.

MR. MUHAMMAD SHARFI BIN NAJIB
(Nama Penyelia)

Tarikh: **26 NOVEMBER 2007**

Tarikh: : **26 NOVEMBER 2007**

CATATAN:
- * Potong yang tidak berkenaan.
- ** Jika tesis ini SULIT atau TERHAD, sila lampirkan surat daripada pihak berkuasa/organisasi berkenaan dengan menyatakan sekali tempoh tesis ini perlu dikelaskan sebagai atau TERHAD.
- ♦ Tesis dimaksudkan sebagai tesis bagi Ijazah doktor Falsafah dan Sarjana secara Penyelidikan, atau disertasi bagi pengajian secara kerja kursus dan penyelidikan, atau Laporan Projek Sarjana Muda (PSM).

"I hereby acknowledge that the scope and quality of this thesis is qualified for the award of the degree of Bachelor of Electrical Engineering (Power Systems)"

Signature: _____

Name: MUHAMMAD SHARFI BIN NAJIB

Date: 26 NOVEMBER 2007

DEVELOPMENT OF MOTOR CONTROL USING GRAPHICAL USER INTERFACE

KHAIRUL ANUAR BIN ARIS

This thesis is submitted as partial fulfillment of the requirements for the award of the Bachelor Degree of Electrical Engineering (Power Systems)

Faculty of Electrical & Electronics Engineering
University Malaysia Pahang

NOVEMBER 2007

"All the trademark and copyrights use here in are property of their respective owner. References of information from other sources are quoted accordingly; otherwise the information presented in this report is solely work of the author."

Signature : _____

Author : KHAIRUL ANUAR BIN ARIS

Date : 26 NOVEMBER 2007

Specially dedicated to
my beloved family and those people who have guided and inspired me
throughout my journey of education.

ACKNOWLEDGEMENT

First and foremost, I am very grateful to the almighty ALLAH S.W.T for giving me this opportunity to accomplish my Final Year Project.

Firstly, I wish to express my deep gratitude to my supervisor, Mr. Muhammad Sharfi bin Najib for all his valuable guidance, assistance and support all through this work.

Secondly, I wish to thank lecturers and technicians, for their suggestions and support on this project. Their comments on this project are greatly appreciated. My thanks are also to all my friends who have involved and helped me in this project.

Most importantly I extend my gratitude to my parents who have encouraged me throughout my education and I will always be grateful for their sacrifice, generosity and love.

ABSTRACT

DC Motor control is very common in robotic application. The developments of this kind of project are widely used in most electronic devices nowadays. There are many application that have been developed based on motor control in electronic field such as in automation, Flexible Manufacturing System (FMS) and Computer Integrated Manufacturing (CIM). The purpose of this project is to develop the Graphical User Interface of Motor Control through MATLAB GUIDE, interface the MATLAB GUI with hardware via communication port and control the DC motor through MATLAB GUI. By using MATLAB GUIDE, it provides a set of tools which simplify the process of laying out and programming GUIs and interface with PIC via serial communication port to control the DC motor. The PIC is used to control motor. As a result, the DC motor is able to be controlled through MATLAB GUI and interface the MATLAB GUI with PIC via serial communication port.

ABSTRAK

Motor DC umumnya dikaitkan dengan bidang robotik dan pembangunan projek-projek yang berkaitan dalam bidang ini sangat meluas yang digunakan dalam kebanyakan peralatan elektrik hari ini. Banyak applikasi yang telah dibangunkan berdasarkan kawalan motor dalam bidang automasi seperti Flexible Manufacturing System(FMS) dan Computer Integrated Manufacturing(CIM). Tujuan projek ini adalah untuk membina grafik antaramuka pengguna untuk mengawal motor DC melalui MATLAB dan membina antaramuka antara MATLAB GUI dengan perkakasan elektronik melalui communication port. Dengan menggunakan MATLAB GUIDE, ia telah menyediakan peralatan yang mana set peralatan ini memudahkan pengguna dengan proses meletak dan membina program untuk grafik antaramuka pengguna untuk mengawal motor DC. PIC digunakan dalam projek ini adalah bertujuan untuk mengawal motor. Sebagai kesimpulanya, motor DC dapat dikawal melalui MATLAB GUI dan berantaramuka dengan perkakasan elektronik melalui serial communication port.

TABLE OF CONTENTS

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Overview

The serial port found on the back of the most PC and it is extremely useful for robotics work. Variety devices are configured to communicate via a serial port.

This Project is focus on designing the Graphical User Interface (GUI) through MATLAB to control the DC motor using PIC. The PIC is a programmable interface devices or controller between PC (MATLAB GUI) and the DC motor. The main contribution of this project is the interfacing of the MATLAB with PIC and Graphical User Interface (GUI).

The Peripheral Interface Controller (PIC) use in this project is as controller device between Personal Computer and the DC motor to control DC motor. The PIC is use because of wide availability and economical. Beside that PIC is a free development tools and can perform many function without needed extra circuitry. The PIC is program using the PICBasic Pro Compiler. The PicBasic Pro Compiler produces code that may be programmed into a wide variety of PICmicro

microcontrollers having from 8 to 84 pins and various on-chip features including A/D converters, hardware timers and serial ports. The purpose using MATLAB in creating the GUI is because it already has Graphical User Interface Development Environment (GUIDE) that provides a set of tools for creating GUI. These tools simplify the process of laying out and programming GUIs.

The GUI create in MATLAB with appropriate coding will control the DC motor via serial port that interface with the PIC. There are many advantage by using the DC motor, among that the DC motor has no adverse effect on power quality and the speed is proportional to the magnetic flux.

1.2 Objective

At the end of this Project:

i. Able to control DC motor through MATLAB GUI.
ii. Able to interface the MATLAB GUI with hardware using PIC.

The important part of this project is to interface the MATLAB GUI with the PIC. This part is done if the PIC produces a signal. The output from PIC will monitor by using the oscilloscope. After that the DC motor can be control via MATLAB GUI.

1.3 Scope of Project

The scopes of this project are laying out the GUI in MATLAB GUIDE and create programming for the GUI's. Secondly Prepare the PIC circuitry and serial

connection (DB9) circuit for interfacing part. And the third part is creating program for PIC using PICBasic Pro Compiler to control the DC motor.

1.4 Problem Statement

The main objective in this project to interface the MATLAB GUI with the PIC. It is a difficult part to develop the program for MATLAB and the PIC simultaneously to make the interfacing part. By using the PicBasic Pro Compiler software to develop programming to control DC motor, it can reduces the difficulty by comprises a list of statements that written in a programming language like assembler, C, or BASIC. With this opportunity, the men in charge do not have to take long time to written and troubleshoot the program.

1.5 Thesis Organization

This thesis consists of five chapters including this chapter. The contents of each chapter are outlined as follows;

Chapter 2 contains a detailed description each part of project. It will explain about the MATLAB GUIDE, PIC, and DC motor. Chapter 3 includes the project methodology. This will explain how the project is organized and the flow of the process in completing this project. Chapter 4 presents the expected result of simulation runs using MATLAB GUIDE. Finally the conclusions for this project are presented in Chapter 5.

CHAPTER 2

LITERATURE REVIEW

2.1 Graphical User Interface (GUI)

2.1.1 General Definition of GUI

A **graphical user interface** (or **GUI**, often pronounced "gooey"), is a particular case of user interface for interacting with a computer which employs graphical images and widgets in addition to text to represent the information and actions available to the user [4][5]. Usually the actions are performed through direct manipulation of the graphical elements.

The first graphical user interface was designed by Xerox Corporation's Palo Alto Research Center in the 1970s, but it was not until the 1980s and the emergence of the Apple Macintosh that graphical user interfaces became popular. One reason for their slow acceptance was the fact that they require considerable CPU power and a high-quality monitor, which until recently were prohibitively expensive [4].

A graphical user interface (GUI) is a pictorial interface to a program. A good GUI can make programs easier to use by providing them with a consistent appearance and with intuitive controls like pushbuttons, list boxes, sliders, menus, and so forth [2][4]. A true GUI includes standard formats for representing text and graphics [4]. The GUI should behave in an understandable and predictable manner, so that a user knows what to expect when he or she performs an action. For example, when a mouse click occurs on a pushbutton, the GUI should initiate the action described on the label of the button.

Many DOS programs include some features of GUIs, such as menus, but are not *graphics based*. Such interfaces are sometimes called *graphical character-based user interfaces* to distinguish them from true GUIs [4].

2.1.2 MATLAB GUI

A graphical user interface (GUI) is a graphical display that contains devices, or components, that enable a user to perform interactive tasks. To perform these tasks, the user of the GUI does not have to create a script or type commands at the command line. Often, the user does not have to know the details of the task at hand [1] [2] [16].

The GUI components can be menus, toolbars, push buttons, radio buttons, list boxes, and sliders — just to name a few. In MATLAB, a GUI can also display data in tabular form or as plots, and can group related components [1] [2] [3].

2.1.3 Operation of GUI

Each component, and the GUI itself, is associated with one or more user-written routines known as callbacks. The execution of each callback is triggered by a particular user action such as, mouse click, pushbuttons, toggle buttons, lists, menus, text boxes, selection of a menu item, or the cursor passing over a component and so forth [1] [2].

Clicking the button triggers the execution of a callback [1]. A mouse click or a key press is an event, and the MATLAB program must respond to each event if the program is to perform its function. For example, if a user clicks on a button, that event must cause the MATLAB code that implements the function of the button to be executed. The code executed in response to an event is known as a call back [1] [2].

This kind of programming is often referred to as event-driven programming. The event in the example is a button click. In event-driven programming, callback execution is asynchronous, controlled by events external to the software. In the case of MATLAB GUIs, these events usually take the form of user interactions with the GUI. The writer of a callback has no control over the sequence of events that leads to its execution or, when the callback does execute, what other callbacks might be running simultaneously [1].

Callbacks

- Routine that executes whenever you activate the uicontrol object
- Define this routine as a string that is a valid MATLAB expression or the name of an M-file
- The expression executes in the MATLAB workspace.

2.1.4 A brief introduction of GUIDE

GUIDE, the MATLAB graphical user interface development environment, provides a set of tools for creating graphical user interfaces (GUIs). These tools simplify the process of laying out and programming GUIs [1].

- GUIDE is primarily a set of layout tools
- GUIDE also generates an M-file that contains code to handle the initialization and launching of the GUI
  - This M-file also provides a framework for the implementation of the callbacks - the functions that execute when users activate a component in the GUI [1].

2.1.4.1Two Basic Task in Process of implementing a GUI

The two basic tasks in Process of implementing a GUI is first, laying out a GUI where MATLAB implement GUIs as figure windows containing various styles of uicontrol (User Interface) objects. The second task is programming the GUI, where each object must be program to perform the intended action when activated by the user of GUI [14].

2.2 DC Motors

2.2.1 Introduction

Electric motors are everywhere! In a house, almost every mechanical movement that you see around you is caused by a DC (direct current) electric motor. An electric motor is a device that transforms electrical energy into mechanical energy by using the motor effect [7] [8].

Every DC motor has six basic parts -- axle, rotor (a.k.a., armature), stator, commutator, field magnet(s), and brushes. In most common DC motors, the external magnetic field is produced by high-strength permanent magnets. The stator is the stationary part of the motor -- this includes the motor casing, as well as two or more permanent magnet pole pieces. The rotor rotates with respect to the stator. The rotor consists of windings (generally on a core), the windings being electrically connected to the commutator [7] [8].

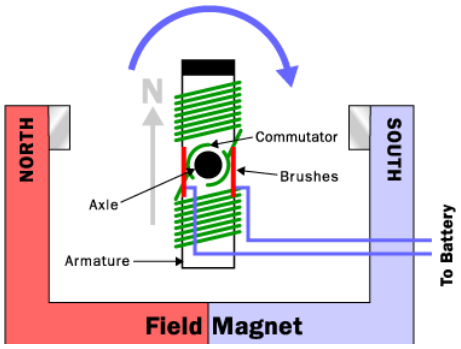


Figure 2.1: Part of an Electric Motor

Industrial applications use dc motors because the speed-torque relationship can be varied to almost any useful form -- for both dc motor and regeneration applications in either direction of rotation. Continuous operation of dc motors is

commonly available over a speed range of 8:1. Infinite range (smooth control down to zero speed) for short durations or reduced load is also common [6].

Dc motors are often applied where they momentarily deliver three or more times their rated torque. In emergency situations, dc motors can supply over five times rated torque without stalling (power supply permitting) [6].

Dc motors feature a speed, which can be controlled smoothly down to zero, immediately followed by acceleration in the opposite direction -- without power circuit switching. And dc motors respond quickly to changes in control signals due to the dc motor's high ratio of torque to inertia [6] [7].

2.2.2 The Advantages

The greatest advantage of DC motors may be speed control. Since speed is directly proportional to armature voltage and inversely proportional to the magnetic flux produced by the poles, adjusting the armature voltage and/or the field current will change the rotor speed [7].

- Today, adjustable frequency drives can provide precise speed control for AC motors, but they do so at the expense of power quality, as the solid-state switching devices in the drives produce a rich harmonic spectrum. The DC motor has no adverse effects on power quality [6] [7].

2.2.3 The drawbacks

- Power supply, initial cost, and maintenance requirements are the negatives associated with DC motors
- Rectification must be provided for any DC motors supplied from the grid. It can also cause power quality problems.
- The construction of a DC motor is considerably more complicated and expensive than that of an AC motor, primarily due to the commutator, brushes, and armature windings. An induction motor requires no commutator or brushes, and most use cast squirrel-cage rotor bars instead of true windings — two huge simplifications [6].

2.2.4 Type of DC Motor

2.2.4.1 Stepper motors

A **stepper motor** is a brushless, synchronous electric motor that can divide a full rotation into a large number of steps, for example, 200 steps. Thus the motor can be turned to a precise angle [7]. A stepper motor is an electromechanical device which converts electrical pulses into discrete mechanical movements and is a unique type of dc motor that rotates in fixed steps of a certain number of degrees. Step size can range from 0.9 to 90 degree [6] [7].

The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the length of rotation is directly related to the number of input pulses applied. The motors rotation has several direct relationships to these applied input pulses. The sequence of the applied pulses is directly related to the direction of

Motor shafts rotation [6] [8]. The stepper motors has an excellent response to start-up, stopping and reverse [7].

There are three main of stepper motor type. First is Permanent Magnet (PM) Motors second is Variable Reluctance (VR) Motors and the third is Hybrid Motors.

2.2.4.2 Brushless DC motors

- A **brushless DC motor** (**BLDC**) is an AC synchronous electric motor that from a modeling perspective looks very similar to a DC motor.
- In a BLDC motor, the electromagnets do not move; instead, the permanent magnets rotate and the armature remains static.
- In order to do this, the brush-system/commutator assembly is replaced by an
- Intelligent electronic controller. The controller performs the same power-distribution found in a brushed DC-motor, but using a solid-state circuit rather than a commutator/brush system [6].

2.2.4.3 Coreless DC motors

- Optimized for rapid acceleration, these motors have a rotor that is constructed without any iron core.
- Because the rotor is much lighter in weight (mass) than a conventional rotor formed from copper windings on steel laminations, the rotor can accelerate much more rapidly, often achieving a mechanical time constant under 1 ms.

- These motors were commonly used to drive the capstan(s) of magnetic tape drives and are still widely used in high-performance servo-controlled systems [6].

2.3 PIC Microcontroller

PIC is a family of Harvard architecture microcontrollers made by Microchip Technology, derived from the PIC1650 originally developed by General Instrument's Microelectronics Division[9] [10].

PICs are popular with developers due to their low cost, wide availability, large user base, extensive collection of application notes, availability of low cost or free development tools, and serial programming (and re-programming with flash memory) capability[9].

2.3.1 ORIGINS

1. The original PIC was built to be used with GI's new 16-bit CPU, the CP1600. While generally a good CPU, the CP1600 had poor I/O performance, and the 8-bit PIC was developed in 1975 to improve performance of the overall system by offloading I/O tasks from the CPU.
2. The PIC used simple microcode stored in ROM to perform its tasks, and although the term wasn't used at the time, it is a RISC design that runs one instruction per cycle (4 oscillator cycles).

3. In 1985 General Instruments spun off their microelectronics division, and the new ownership cancelled almost everything — which by this time was mostly out-of-date. The PIC, however, was upgraded with EPROM to produce a programmable channel controller, and today a huge variety of PICs are available with various on-board peripherals (serial communication modules, UARTs, motor control kernels, etc.) and program memory from 512 words to 32k words and more[9].

2.3.2 PIC Microcontroller Option

A PIC Microcontroller chip combines the function of microprocessor, ROM program memory, some RAM memory and input-output interface in one single package which is economical and easy to use [10][14].

The PIC – Logicator system is designed to be used to program a range of 8, 18, 28 pin reprogrammable PIC microcontroller which provide a variety of input – output, digital input and analogue input options to suit students project uses [10].

Reprogrammable “FLASH Memory” chips have been selected as the most economical for student use. If a student needs to amend to control system as the project is evaluated and developed, the chip can simply be taken out of the product and reprogrammed with an edited version of the floe sheet [10].

The PIC devices generally feature is sleep mode (power saving), watchdog timer and various crystal or RC oscillator configuration, or an external clock.

2.3.3 Variants

Within a series, there are still many device variants depending on what hardware resources the chip features [9].

- general purpose i/o pins
- internal clock oscillators
- 8/16 Bit Timers
- Internal EEPROM Memory
- Synchronous/Asynchronous Serial Interface USART
- MSSP Peripheral for I²C and SPI Communications
- Capture/Compare and PWM modules
- Analog-to-digital converters
- USB, Ethernet, CAN interfacing support
- external memory interface
- Integrated analog RF front ends (PIC16F639, and rfPIC)
- KEELOQ Rolling code encryption peripheral (encode/decode)

2.4 PIC Basic Pro Compiler

The PicBasic Pro Compiler (or PBP) makes it even quicker and easier to program Microchip Technology's powerful PICmicro microcontrollers (MCUs). The English-like BASIC language is much easier to read and write than the quirky Microchip assembly language [11].

The PicBasic Pro Compiler is "BASIC Stamp II like" and has most of the libraries and functions of both the BASIC Stamp I and II. Being a true compiler, programs execute much faster and may be longer than their Stamp equivalents. PBP is not quite as compatible with the BASIC Stamps as our original [11] [17].

ject

The PicBasic Pro Compiler produces code that may be programmed into a wide variety of PICmicro microcontrollers having from 8 to 84 pins and various on-chip features including A/D converters, hardware timers and serial ports [11].

2.5 LDmicro

LDmicro generates native code for certain Microchip PIC16 and Atmel AVR microcontrollers. Usually software for these microcontrollers is written in a programming language like assembler, C, or BASIC. A program in one of these languages comprises a list of statements. These languages are powerful and well-suited to the architecture of the processor, which internally executes a list of instructions. PLCs, on the other hand, are often programmed in `ladder logic.'

CHAPTER 3

METHODOLOGY

3.1 Introduction

This chapter presents the methodology of this project. It describes on how the project is organized and the flow of the steps in order to complete this project. The methodology is diverged in two parts, which is developing the hardware to interface with MATLAB. The other is developing the programming for MATLAB and the PIC to control DC motor.

3.2 Methodology

There are three mains method in order to develop this project. Before the project is developing using MATLAB, it is needed to do the study on MATLAB GUIDE and the hardware (especially PIC). The flowchart in Figure 3.1 illustrated the

sequence of steps for this project. The first method is developing GUI in MATLAB and programs every GUI component. Secondly is to develop PIC programming to control 5V DC and stepper motor. And lastly is hardware design which is use to interface with MATLAB GUI.

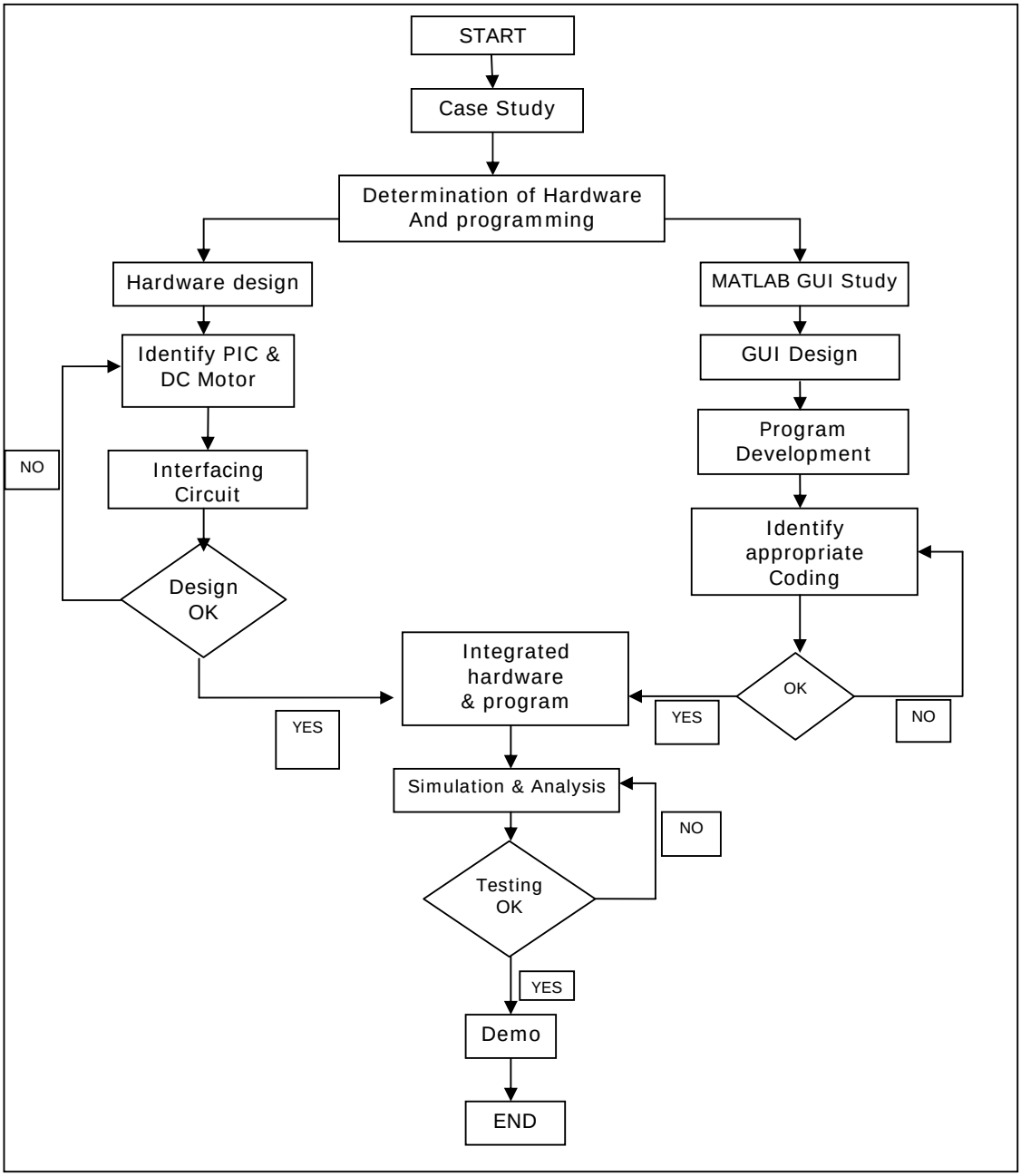


Figure 3.1: Flowchart for Whole Project

Figure 3.1 show the flow of the whole project. The project begins after registering the PSM title with doing case study about the project. The flow of the project is separate into two main tasks that are hardware design and MATLAB GUI design. In hardware design part flow, the main target is to create appropriate programming for PIC to interface with personal computer via serial port to control DC motor. The second part, the prior task is to develop program in MATLAB to interface with PIC and the DC motor. After that the both part is combine and do the analysis until achieve the needed objective. The main contribution of this project is to interface MATLAB GUI with the PIC.

3.2.1 Development MATLAB GUI Using MATLAB GUIDE

GUIDE, the MATLAB graphical user interface development environment, provides a set of tools for creating graphical user interfaces (GUIs). These tools simplify the process of laying out and programming GUIs.

There are 5 steps in build the MATLAB GUI. First Use a MATLAB tool called guide (GUI Development Environment) to layout the components that show in figure 3.2. This tool allows a programmer to layout the GUI, selecting and aligning the GUI components to be placed in it. The basic component of the MATLAB GUI is shown in Table 3.1.

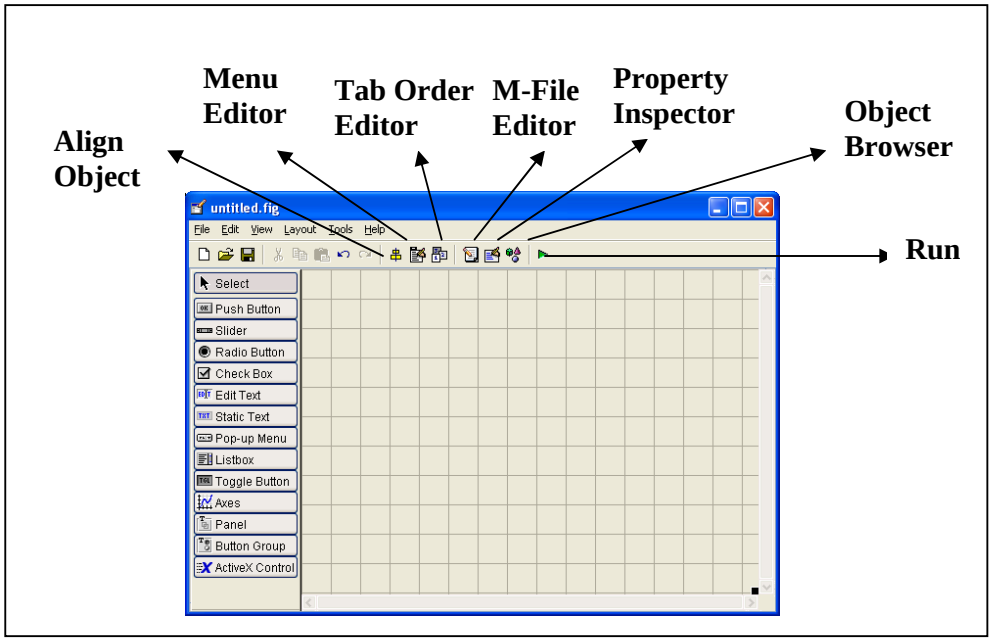


Figure 3.2: MATLAB GUIDE Layouts

Next is Use a MATLAB tool called the Property Inspector (built into guide) to give each component a name (a "tag") and to set the characteristics of each component, such as its color, the text it displays, and so on. After that, save the figure to a file. When the figure is saved, two files will be created on disk with the same name but different extents. The fig file contains the actual GUI that has been created, and the M-file contains the code to load the figure and skeleton call backs for each GUI element. These two files usually reside in the same directory. They correspond to the tasks of laying out and programming the GUI. When you lay out the GUI in the Layout Editor, your work is stored in the FIG-file. When you program the GUI, your work is stored in the corresponding M-file.

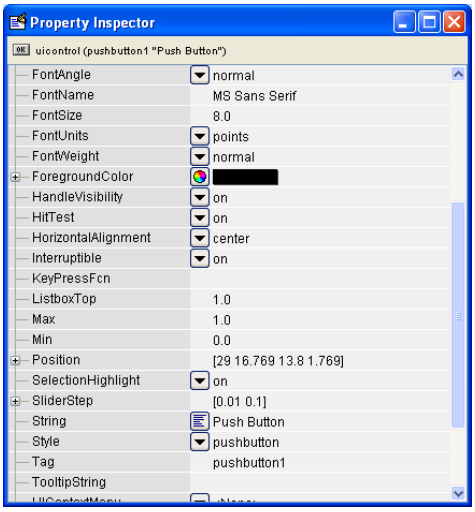


Figure 3.3: Property Inspector

Table 3.1: Basic MATLAB GUI Component [12]

Element	Created By	Description
Graphical Controls		
Pushbutton	`uicontrol`	A graphical component that implements a pushbutton. It triggers a callback when clicked with a mouse.
Toggle button	`uicontrol`	A graphical component that implements a toggle button. A toggle button is either "on" or "off," and it changes state each time that it is clicked. Each mouse button click also triggers a callback.
Radio button	`uicontrol`	A radio button is a type of toggle button that appears as a small circle with a dot in the middle when it is "on." Groups of radio buttons are used to implement mutually exclusive choices. Each mouse click on a radio button triggers a callback.
Check box	`uicontrol`	A check box is a type of toggle button that appears as a small square with a check mark in it when it is "on." Each mouse click on a check box triggers a callback.
Edit box	`uicontrol`	An edit box displays a text string and allows the user to modify the information displayed. A callback is triggered when the user presses the `Enter` key.
List box	`uicontrol`	A list box is a graphical control that displays a series of text strings. A user can select one of the text strings by single- or double-clicking on it. A callback is triggered when the user selects a string.
Popup menus	`uicontrol`	A popup menu is a graphical control that displays a series of text strings in response to a mouse click. When the popup menu is not clicked on, only the currently selected string is visible.
Slider	`uicontrol`	A slider is a graphical control to adjust a value in a smooth, continuous fashion by dragging the control with a mouse. Each slider change triggers a callback.
Static Elements		
Frame	`uicontrol`	Creates a frame, which is a rectangular box within a figure. Frames are used to group sets of controls together. Frames never trigger callbacks.
Text field	`uicontrol`	Creates a label, which is a text string located at a point on the figure. Text fields never trigger callbacks.
Menus and Axes		
Menu items	`uimenu`	Creates a menu item. Menu items trigger a callback when a mouse button is released over them.
Context menus	`uicontextmenu`	Creates a context menu, which is a menu that appears over a graphical object when a user right-clicks the mouse on that object.
Axes	`axes`	Creates a new set of axes to display data on. Axes never trigger callbacks.

After laying out the GUI component and set the property, the GUI will be look like in figure 3.4 for example according to the user creativity.

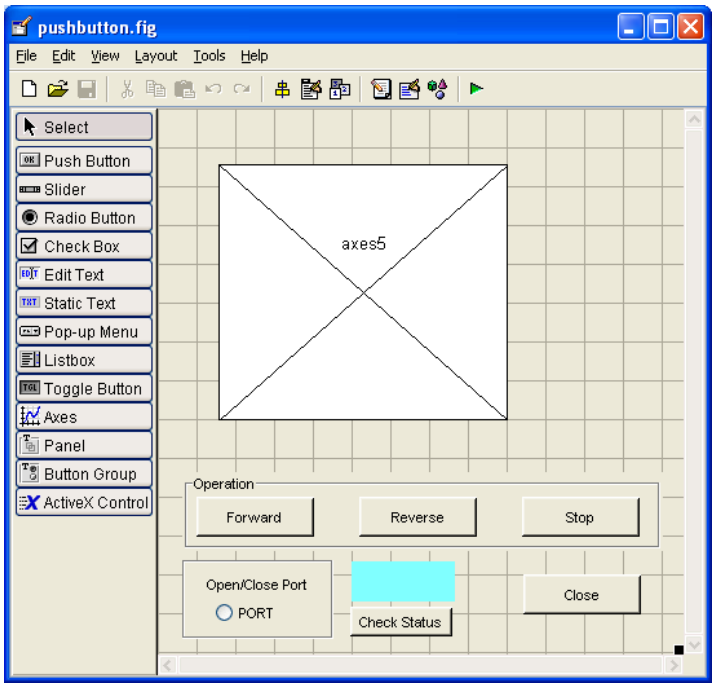


Figure 3.4: Example GUI

And finally write code to implement the behavior associated with each callback function in m-files show in figure 3.5. A callback is a function that writes and associates with a specific GUI component or with the GUI figure. It controls GUI or component behavior by performing some action in response to an event for its component. This kind of programming is often called event-driven programming. This last step is the difficult one and has to make an extra reading on how to write the coding before the GUI component can perform some task that user desire.

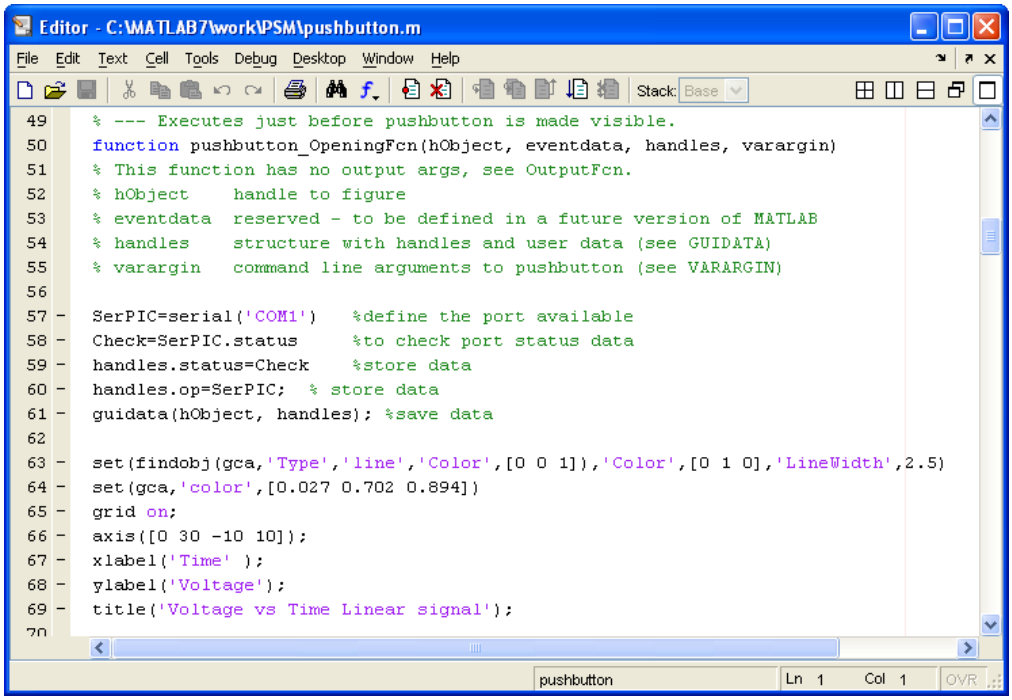


Figure 3.5: Example M-files for GUI

3.2.2 Build MATLAB Programming

After layed out the GUI, it need to program its behavior. The code is to write controls how the GUI responds to events such as button clicks, slider movement, menu item selection, or the creation and deletion of components. This programming takes the form of a set of functions, called callbacks, for each component and for the GUI figure itself.

A callback is a function that writes and associates with a specific GUI component or with the GUI figure. It controls GUI or component behavior by performing some action in response to an event for its component. This kind of programming is often called event-driven programming.

The GUI figure and each type of component have specific kinds of callbacks with which it can be associated. The callbacks that are available for each component are defined as properties of that component. Each kind of callback has a triggering mechanism or event that causes it to be called. The kind of callback is shown in table 3.2.

Table 3.2: Kind of Callback

Callback Property	Triggering Event	Components
DeleteFcn	Component deletion. It can be used to perform cleanup operations just before the component or figure is destroyed.	Axes, figure, button group, context menu, menu, panel, user interface controls
KeyPressFcn	Executes when the user presses a keyboard key and the callback's component or figure has focus.	Figure, user interface controls
ResizeFcn	Executes when a user resizes a panel, button group, or figure whose figure. Resize property is set to On.	Button group, figure, panel
SelectionChangeFcn	Executes when a user selects a different radio button or toggle button in a button group component.	Button group
WindowButtonDownFcn	Executes when you press a mouse button while the pointer is in the figure window.	Figure

WindowButtonMotionFcn	Executes when you move the pointer within the figure window.	Figure
WindowButtonUpFcn	Executes when you release a mouse button.	Figure
ButtonDownFcn	Executes when the user presses a mouse button while the pointer is on or within five pixels of a component or figure. If the component is a user interface control, its Enable property must be on.	Axes, figure, button group, panel, user interface controls
Callback	Component action. Executes, for example, when a user clicks a push button or selects a menu item.	Context menu, menu, user interface controls
CloseRequestFcn CreateFcn	Executes before the figure closes. Component creation. It can be use to initialize the component when it is created. It executes after the component or figure is created, but before it is displayed.	Figure Axes, figure, button group, context menu, menu, panel, user interface controls

The GUI M-file that GUIDE generates is a function file. The name of the main function is the same as the name of the M-file. For example, if the name of the M-file is mygui.m, then the name of the main function is mygui. Each callback in the file is a sub function of the main function. When GUIDE generates an M-file, it automatically includes templates for the most commonly used callbacks for each component. The major sections of the GUI M-file are ordered as shown in table 3.3.

Table 3.3: Major Sections of the GUI M-file [13]

Section	Description
Comments	Displayed at the command line in response to the `help` command. Edit these as necessary for your GUI.
Initialization	GUIDE initialization tasks. *Do not edit this code.*
Opening function	Performs your initialization tasks before the user has access to the GUI.
Output function	Returns outputs to the MATLAB command line after the opening function returns control and before control returns to the command line.
Component and figure callbacks	Control the behavior of the GUI figure and of individual components. MATLAB calls a callback in response to a particular event for a component or for the figure itself.
Utility/helper functions	Perform miscellaneous functions not directly associated with an event for the figure or a component.

GUIDE automatically includes two callbacks, the opening function and the output function, in every GUI M-file it creates. The opening function programming is importance in initialize the communication port in MATLAB GUI before it can transmit data to the PIC. The data send from MATLAB GUI to PIC is in decimal form and PIC will control the DC motor with the preset programming according to the data received. Here is the example programming in figure 3.6 and figure 3.7 to initialize and close communication port at the back of computer using radio button in MATLAB GUI. In figure 3.8 is example to transmit data to PIC.

```
48
49    % --- Executes just before pushbutton is made visible.
50    function pushbutton_OpeningFcn(hObject, eventdata, handles, varargin)
51    % This function has no output args, see OutputFcn.
52    % hObject    handle to figure
53    % eventdata  reserved - to be defined in a future version of MATLAB
54    % handles    structure with handles and user data (see GUIDATA)
55    % varargin   command line arguments to pushbutton (see VARARGIN)
56
57 -  SerPIC=serial('COM1')    %define the port available
58 -  Check=SerPIC.status      %to check port status data
59 -  handles.status=Check     %store data
60 -  handles.op=SerPIC;  % store data
61 -  guidata(hObject, handles); %save data
```

Figure 3.6: Initialize Communication Port

```
94    % --- Executes on button press in open_close_port.
95    function open_close_port_Callback(hObject, eventdata, handles)
96    % hObject    handle to open_close_port (see GCBO)
97    % eventdata  reserved - to be defined in a future version of MATLAB
98    % handles    structure with handles and user data (see GUIDATA)
99
100   % Hint: get(hObject,'Value') returns toggle state of open_close_port
101
102 - if (get(hObject,'Value')==get(hObject,'Max'));
103 -     SerPIC=handles.op  % retrieve data
104 -     set(SerPIC,'BaudRate',9600,'DataBits',8,'Parity','none','StopBits',1,'FlowControl','none');
105 -     fopen(SerPIC)
106 -     guidata(hObject,handles);     %save data    ;
107 - else
108 -     SerPIC=handles.op
109 -     fclose(SerPIC)
110 -     guidata(hObject,handles)
111
112 - end
113 - guidata(hObject,handles);
```

Figure 3.7: Open and Close of Communication Port

```
193   % --- Executes on button press in stop_PB.
194   function stop_PB_Callback(hObject, eventdata, handles)
195   % hObject    handle to stop_PB (see GCBO)
196   % eventdata  reserved - to be defined in a future version of MATLAB
197   % handles    structure with handles and user data (see GUIDATA)
198
199 -  SerPIC=handles.op   %retrieve data
200
201 -  m=1:0.1:1000;
202 -  n=-m;
203 -  c=m+n;
204 -  plot(c);
205 -  set(findobj(gca,'Type','line','Color',[0 0 1]),'Color',[0 1 0],'LineWidth',2.5)
206 -  set(gca,'color',[0.027 0.702 0.894])
207 -  grid on;
208 -  axis([0 30 -10 10]);
209 -  xlabel('Time' );
210 -  ylabel('Voltage');
211 -  title('Voltage vs Time Linear signal');
212
213 -  fprintf(SerPIC,'%s','031'); %tranmit data to PIC
```

Figure 3.8: Transmit data to PIC

In opening and closing the communication port the command fclose (SerPIC) is use to disconnect a serial port object from the device. The baud rate from MATLAB GUI must be set same with the baud rate in PIC before it can transmit and receive the data. For example if baud rate in MATLAB GUI is 9600bps, so the baud rate in PIC also 9600bps.

3.2.3 Build PIC programming

There many ways to program the PIC either the user can use LDmicro, assembly language or PICBasic Pro Compiler. The LDmicro use ladder diagram approach like PLC while PICBasic Pro compiler is English-like BASIC language and much easier to read and write than the quirky Microchip assembly language.

The data from MATLAB GUI is send to PIC in decimal form, so the PIC is program to read or receive the data also in decimal form. The communication

between MATLAB GUI and PIC is in standard asynchronous format where the device uses its own internal clock resulting in bytes that are transferred at arbitrary times. The baud rate is specifying according to MATLAB GUI. Some standard baud rates are listed in table 3.4. For PIC programming, 9600bps is using which same with the MATLAB GUI.

Table 3.4: List of Standard Baud Rate [11]

Baud Rate	Bits 0 - 12
300	3313
600	1646
1200	813
2400	396
4800	188
9600*	84
19200*	32

The input data at PIC that transmit from MATLAB GUI is set to PORTC.0 before it run certain program to control the DC motor. Here is the example to program the stepper motor run in clockwise and anticlockwise direction. If MATLAB send data '001', so the PIC will perform case 001 according the programming.

```
INCLUDE "bs2defs.bas"          'has some definition in it
DEFINE      OSC 8              'define the oscillator speed in MHz
SerI    VAR   PORTC.0          'define input port

x       VAR          BYTE

TRISA = %00000000              'set PortA as an output port

Start:
   portc = %00000000           'clear port C
   Serin2 SerI, 84, [dec3 B0]  'get three digit decimal number data from
                               MATLAB GUI

     SELECT CASE B0
```

```
        CASE 001                    'clockwise
        FOR x=1 to 80
        GOSUB qwe
        NEXT x

        CASE 002                    'anti clockwise
        FOR x=1 to 80
        GOSUB ewq
        NEXT x

GOTO Start

        qwe:
           porta=%00000101
           PAUSE 30
           porta=%00001001
           PAUSE 30
           porta=%00001010
           PAUSE 30
           porta=%00000110
           PAUSE 30

        RETURN

        ewq:
           porta=%00000101
           PAUSE 30
           porta=%00000110
           PAUSE 30
           porta=%00001010
           PAUSE 30
           porta=%00001001
           PAUSE 30

        RETURN

END
```

To program the PIC, make sure the oscillator that defines in programming is same as use at hardware to avoid instability during transmit and receive data. The SERIN2 command in the program support many different data modifier which may be mixed and matches freely within single SERIN2 statement to provide various input formatting. The modifier support is shown in table 3.5. The number 84 on

“Serin2 SerI, 84, [dec3 B0]” command is refer to baud rate that equal to 9600bps according table 3.4.

Table 3.5: Modifier Support by SERIN2 Command [11]

Modifier	Operation
BIN{1..16}	Receive binary digits
DEC{1..5}	Receive decimal digits
HEX{1..4}	Receive upper case hexadecimal digits
SKIP n	Skip n received characters
STR ArrayVar\n{\c}	Receive string of n characters optionally ended in character c
WAIT ( )	Wait for sequence of characters
WAITSTR ArrayVar{\n}	Wait for character string

In the PM type stepper motor, a permanent magnet is used for rotor and coils are put on stator. The stepper motor model which has 4-poles is shown in the figure 3.9. In case of this motor, step angle of the rotor is 90 degrees. The turn of the motor is controlled by the electric current which pours into X, X’ and Y, Y’. The direction of stepper run can be fixed according table 3.6.

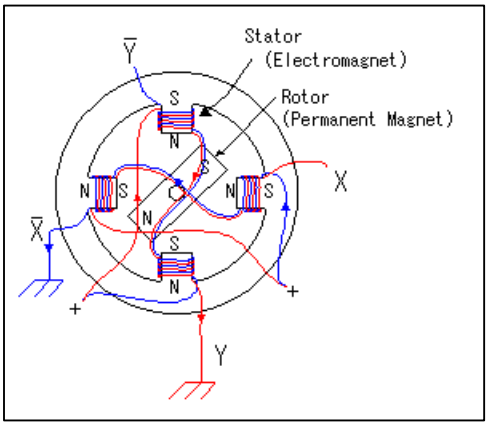


Figure 3.9: Construction of Stepper Motor

Table 3.6: Direction Control of Stepper Motor

Clockwise Control	**X**	**X'**	**Y**	**Y'**	Counter Clockwise Control	**X**	**X'**	**Y**	**Y'**
	0	1	0	1		0	1	0	1
	1	0	0	1		0	1	1	0
	1	0	1	0		1	0	1	0
	0	1	1	0		1	0	0	1
Step Angle	0º	90 º	180 º	270 º		0 º	-90 º	-180 º	-270 º

The command "PAUSE 30" on the programming will determine the rotation speed of stepper motor in millisecond. If the value of PUASE is decrease, it means the step of stepper motor is greater. The total rotation of stepper motor also can be set with user needed at "FOR x 1 to 80" command.

In this project the programming for PIC has been develop which can control four variable speed of motor either in clockwise or counters clockwise direction. All the speed and rotation of the motor can be control via MATLAB GUI. Beside can control stepper motor in same time the PIC also can control 5V DC motor.

The general flow of the PIC program is show in figure 3.10. The PIC will wait the data transfer from MATLAB GUI before it run a specific task to control the motor according the data transferred. For example, if MATLAB GUI sends three decimal number data "030", the PIC will run the task or specific program under CASE 030 in PIC. If the data send by MATLAB GUI is not valid, there is nothing happen to the PIC until the valid data received again.

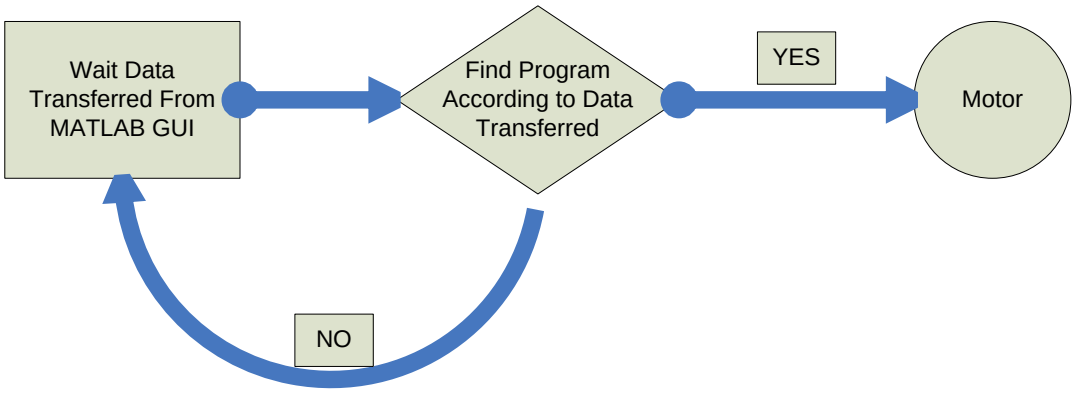


Figure 3.10: General PIC Program Flow

3.2.4 Hardware Installation

For hardware design, first is to design the power supply module which is to supply 5V fixed to PIC and max232 IC. Power supply module is importance to PIC and max232 to prevent damage if users give the higher input supply to device. The schematic diagram for power supply module is like in figure 3.11. Input to the power supply must greater than 7V to 7805 voltage regulator IC to achieve the 5V output supply to PIC and max232.

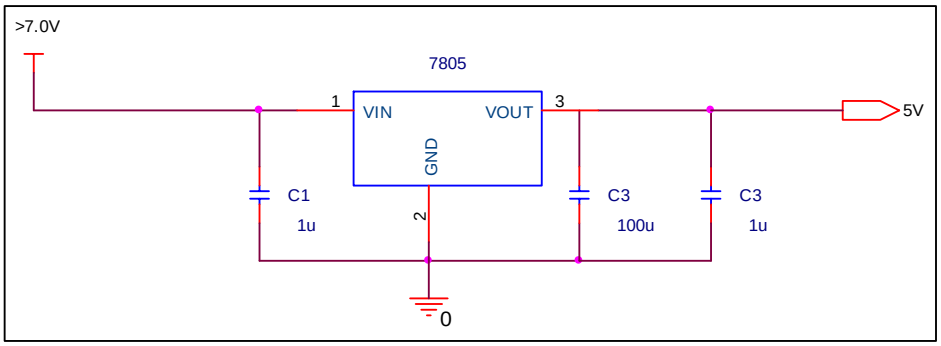


Figure 3.11: Power Supply Modules

Second is to design the connection from communication port (DB9 female connection) from computer to the device which is the pin assignment is shown in table 3.7 below and the figure of RS 232 communication port shown on figure 3.12. In fact, only three pins are required for serial port communications: one for receiving data, one for transmitting data, and one for the signal ground. The connection from computer to device only on pin 2, 3 and pin 5. The circuit in figure 3.13 shows the connection between RS232 with MAX232 and the PIC.

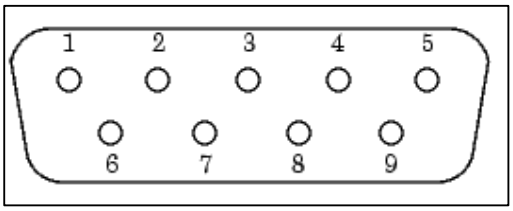


Figure 3.12: Pins and Signals Associated With the 9-pin Connector

Table 3.7: Serial Port Pin and Signal Assignments

Pin	Label	Signal Name	Signal Type
1	CD	Carrier	Detect Control
2	RD	Received Data	Data
3	TD	Transmitted Data	Data
4	DTR	Data Terminal Ready	Control
5	GND	Signal Ground	Ground
6	DSR	Data Set Ready	Control
7	RTS	Request to Send	Control
8	CTS	Clear to Send	Control
9	RI	Ring Indicator	Control

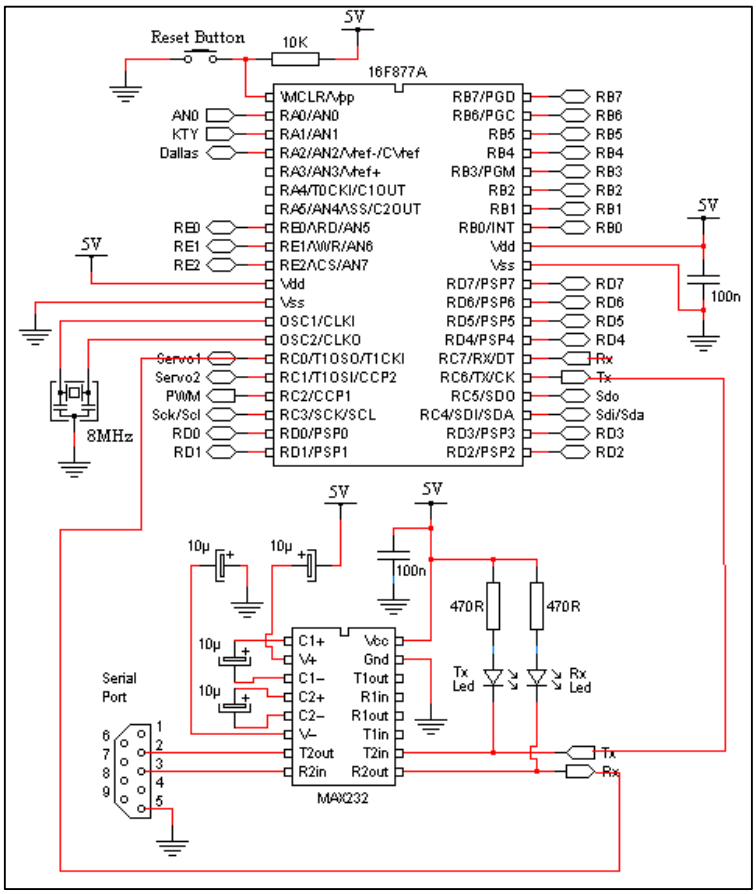


Figure 3.13: Serial Port Connection to PIC

In this project the output data from MAX232 is send directly to PIC at PORTC.0. This connection is depending on the PIC programming that has been developing before it can perform specific task according the data send from the MATLAB GUI. The oscillator use in the circuit diagram also same with the define one in the PIC programming to avoid instability.

The output on the PIC port is approximately 4.7 V low current which is cannot run the stepper motor or DC motor directly. So, to run the motor, switching approach is use by using additional source with high current supply. To done this method the Darlington transistor (C1815) is use like the circuit in figure 3.14. To run the DC motor in forward or reverse direction it has to use relay because it cannot directly control via PIC. In this project, the PORTA.0 to PORTA.3 will be use to

control the coil of stepper motor while PORTD.0 and PORTD.1 is use to control 5V DC motor like in figure 3.15. Figure 3.16 show the hardware installation use in this project in control the 5V DC motor and stepper motor. The hardware installation for this project is shown in figure 3.16.

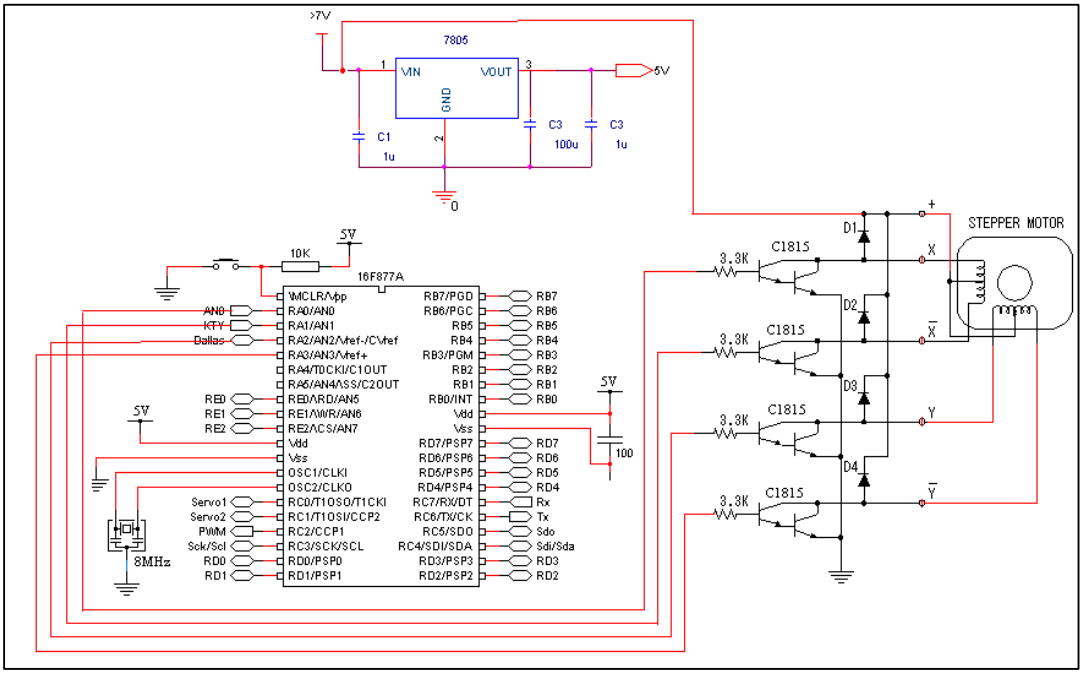


Figure 3.14: Stepper Motor and Switching Circuit

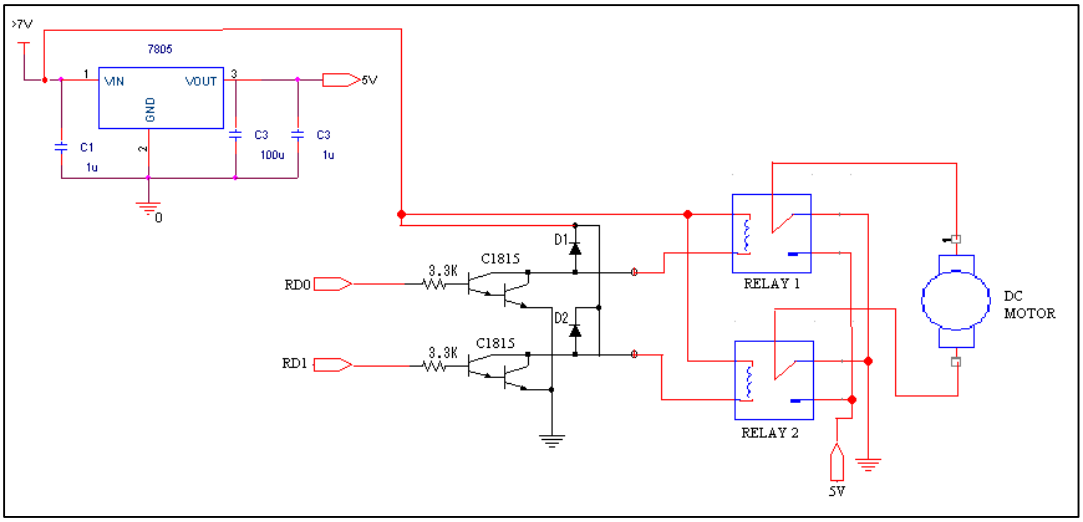


Figure 3.15: 5V DC Motor Connection

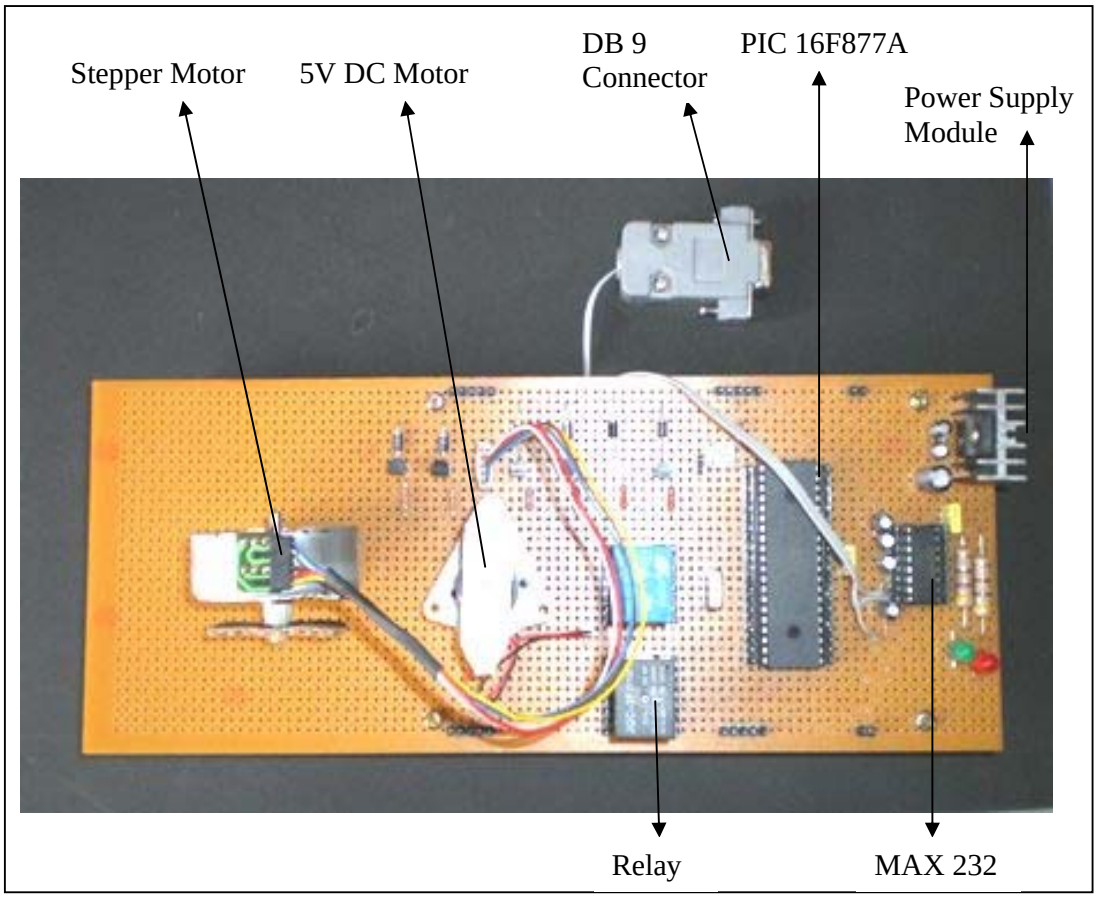


Figure 3.16: Hardware (Top View)

CHAPTER 4

RESULT DISCUSSION

4.1 Introduction

This chapter consists of the discussions on the results from the MATLAB GUI layout that has been developing using MATLAB Graphical User Interface Development Environment. The MATLAB GUI in this project can be divided to four parts. First part is main menu of the whole GUI. Second part is interfacing MATLAB GUI software. The third part is advance MATLAB GUI development and the last part is user information GUI.

4.2 Main Menu of the GUI

The main menu of the GUI in this project contain of four pushbutton which

link to motor control, general info about the project abstract and credit and lastly is exit button. The main menu of the GUI and info of the project is shown in figure 4.1. For motor control pushbutton will explain detail in the next sub chapter. In credit part shown in figure 4.2 contains the detail about the GUI developer and the supervisor. For the exit button user will ask about the confirmation either to exit the GUI or not. The confirmation figure is shown in figure 4.3.

Figure 4.1: Main menu of the GUI

Figure 4.2: Credit

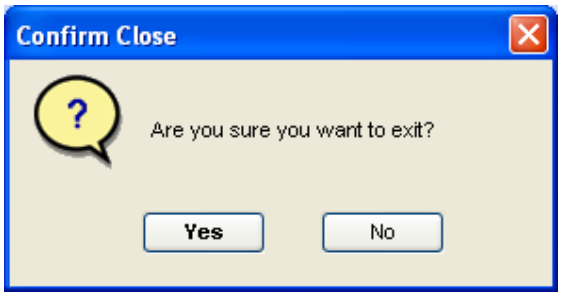


Figure 4.3: Exit Button Confirmations

4.3 Interface MATLAB GUI Software

For motor control part, it divides into two parts where the first part is interfacing software and the second part is advance GUI development for future. The first part of the motor control GUI is the main objective of this project where to interface between MATLAB GUI with the device (motor) to control the motor.. The figure of motor control menu is shown in figure 4.4. The interface software is developing only for 5V DC motor and basic stepper motor control. The rest is for advance development. In the menu motor control menu also, user can get the information to using this software and will discuss in the next chapter.

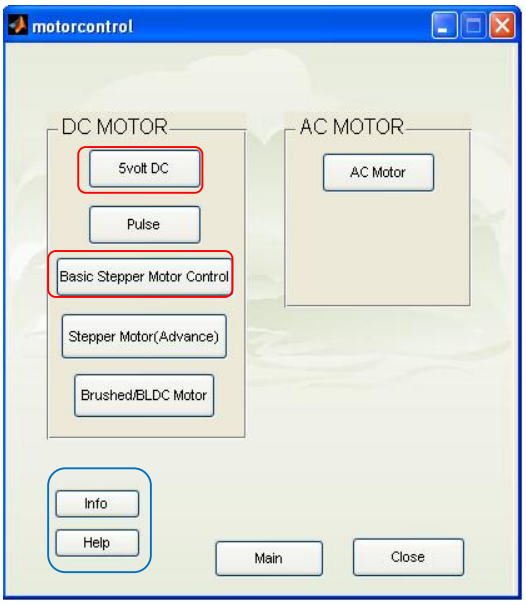


Figure 4.4: Motor Control Menu

The menu of the 5V DC motor is shown in figure 4.5. Before the motor can be control, the user has to tick the 'PORT' button in open/close port panel to initialize the port. If users not tick the button, the GUI cannot send the data to PIC. To check the status of the port, user only has to push the check status button. It will display either the port is opened or closed condition like in figure 4.6. For 5V DC motor the user can control either forward or reverse direction. The user also can stop the motor with click on Stop button in operation panel. The graph on the menu is only to give information about the output from PIC supply to the motor. For future development the graph will shown the actual voltage that supply to the motor directly and has close loop feedback.

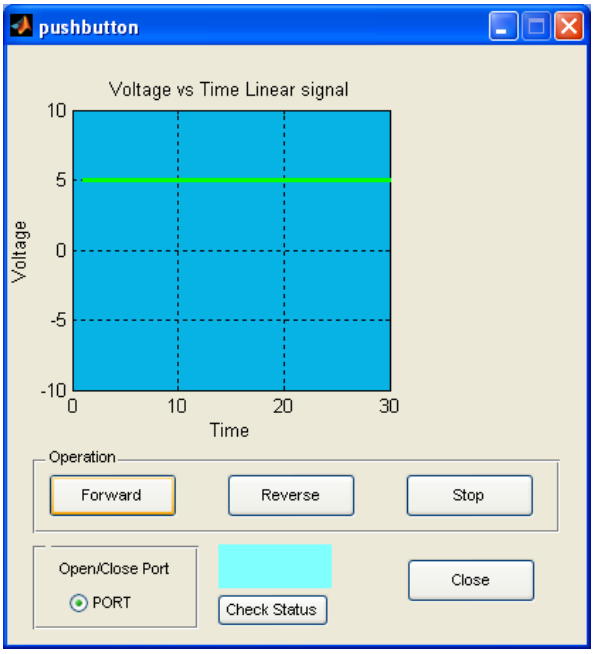


Figure 4.5: 5V DC Motor Menu

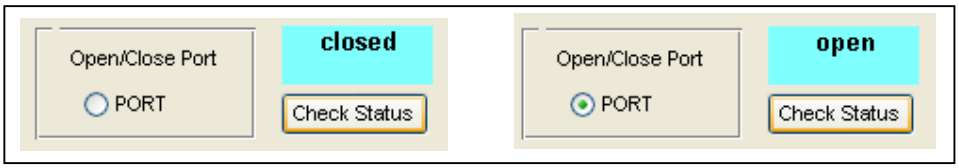


Figure 4.6: Communication Port Status

The second software that has been developed is for basic stepper motor control. In this software, the stepper motor can be control either in forward or reverse direction with four variable speeds where 1 is slowest and 4 is the fastest. The stepper motor also can be control to forward and reverse with one button click. This feature also can be control in four different speeds. Beside that, the stepper motor also can be control in random speed. The menu of basic stepper motor is shown in figure 4.7. The open/close port part is same like in 5V DC motor control.

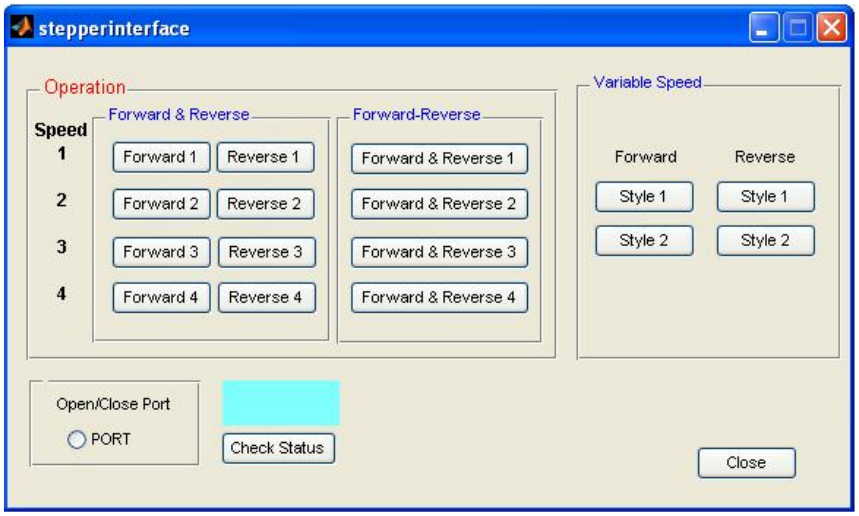


Figure 4.7: Basic Stepper Motor Control

For both motor controls, before user quit the GUI, the reminder warning will pop up like in figure 4.8. The user will remind about to close the port before exit in order to avoid error to run next interface GUI software. If errors happen, user must restart the MATLAB and run the GUI back.

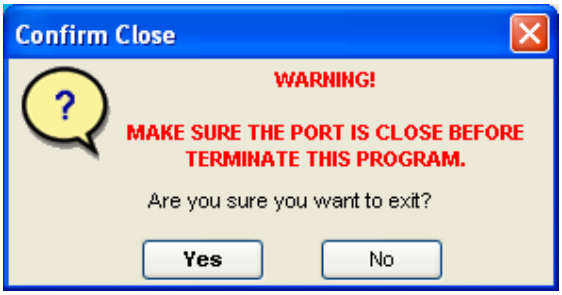


Figure 4.8: Warning pop up Menu

4.4 Advance Development of GUI

In this part the software is develop also to control the motor but in different way. But the development this kind of project need further study. This part contains pulse control to control either stepper motor or Brushed/Brushless DC motor shown

in figure 4.9, advance stepper motor control shown in figure 4.10 and Brushed/Brushless DC motor shown in figure 4.11.

For the pulse menu, the graph will plot the actual output that generate after the properties is set on the GUI menu. The output actually can be compare with the oscilloscope with the graph plot in the GUI. This software actually design to run either stepper motor or Brushed/BLDC motor that use PWM concept.

In the Brushed/BLDC motor control menu the user control PWM mode and frequency beside can control direction in clockwise or counter clockwise. PWM duty cycle is to change the speed of the motor in three decimal numbers where 255 is the maximum speed.

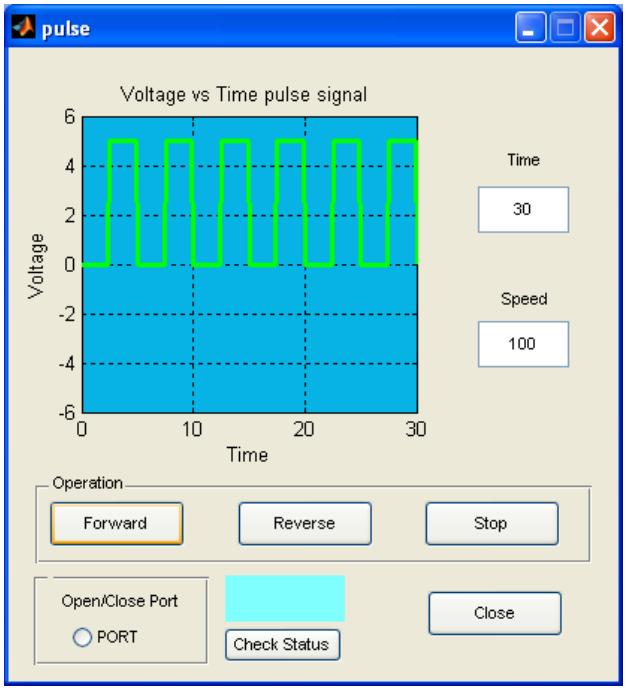


Figure 4.9: Pulse Control GUI Menu

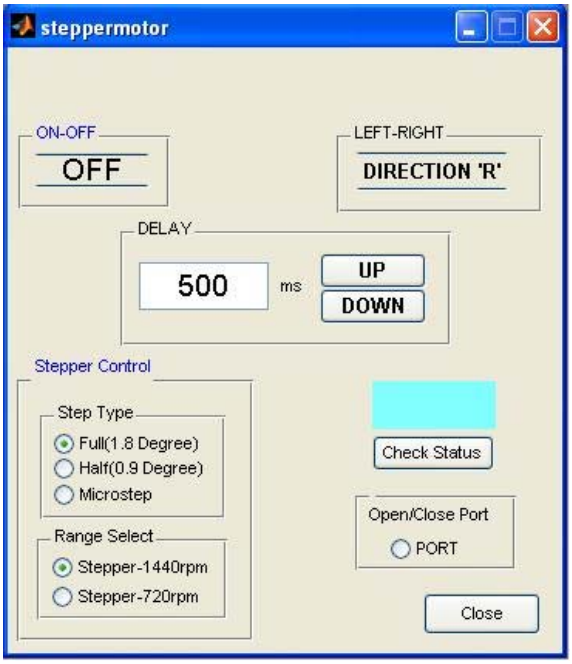


Figure 4.10: Advance Stepper Motor Control GUI Menu

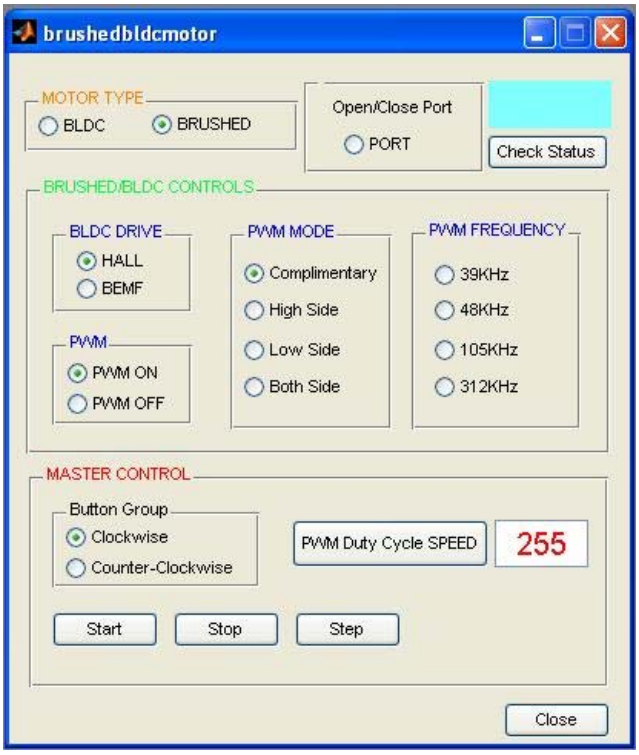


Figure 4.11: Brushed/Brushless DC Motor GUI Menu

4.5 User Information GUI

This part (mark with blue line in figure 4.4) provides the user manual as guidance to use these MATLAB GUI software. The manual is important for the first time user to get the information on how to operate the GUI in right way. The user can get the information on how to setting the port shown in figure 4.12, because if this software use in different computer, the communication port configuration also differ. So the GUI software cannot control the motor or in other word the interface between MATLAB GUI and device is failed. The data is not send to the PIC. Beside that user can get the information what to do before, during or after using the software shown in figure 4.13.

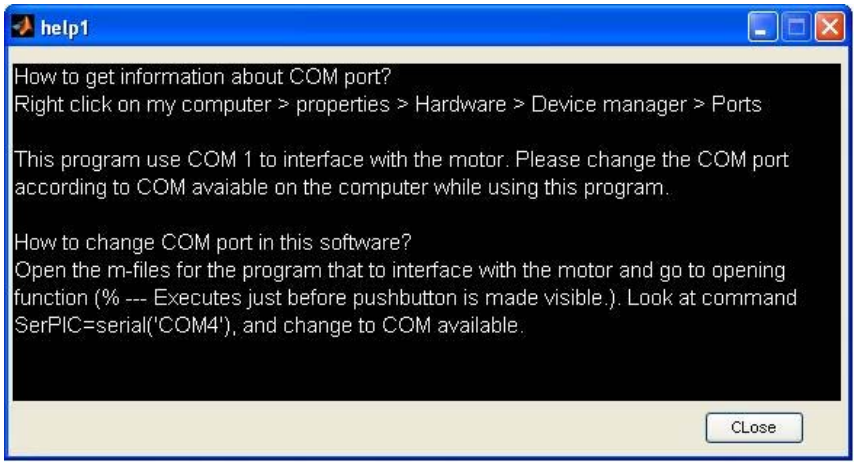


Figure 4.12: Help Menu

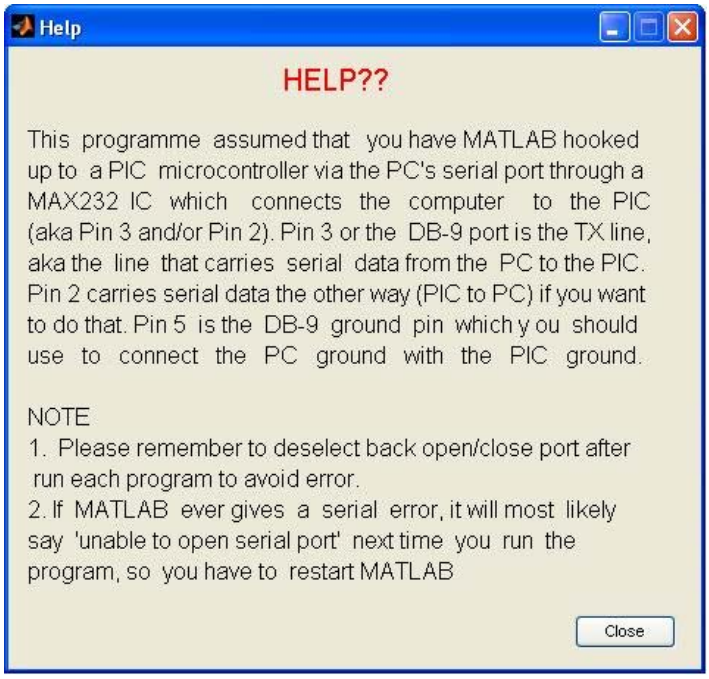


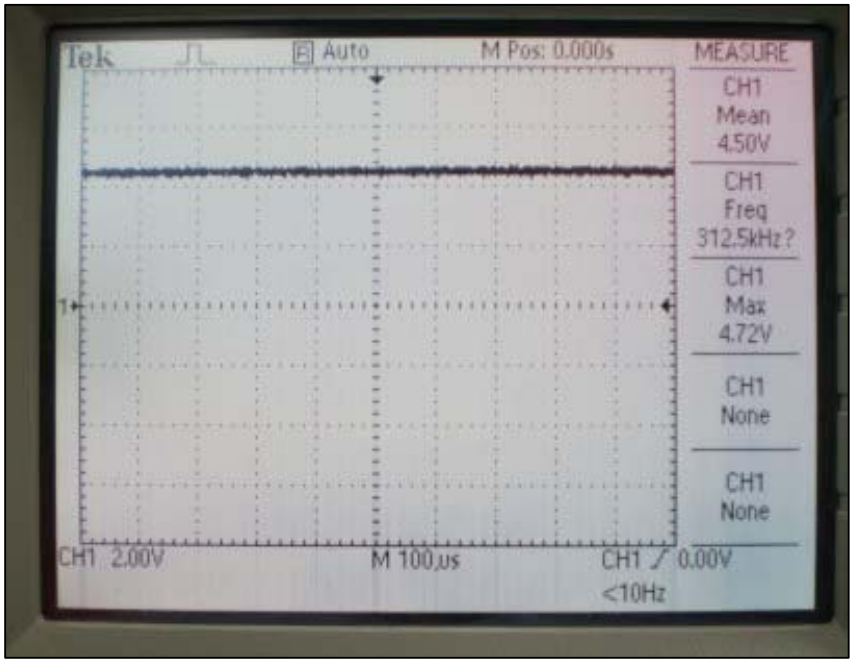
Figure 4.13: Info Menu

4.6 Observation of PIC Output

In this part, the PIC output observation is made to ports which control the 5V DC motor and the stepper motor. The observation of the output is monitor using oscilloscope.

4.6.1 5V DC Motor output Observation

The output for the motor in forward and reverse condition is shown in figure 4.14 where the output is approximately 4.72V maximum. The forward and reverse of the motor is control by additional circuit using relay. So the output for PORTD.0 and PORTD.1 is same and just to active the relay. In stop condition the output that has been monitor is shown in figure 4.15 where the output is nearly 0V.



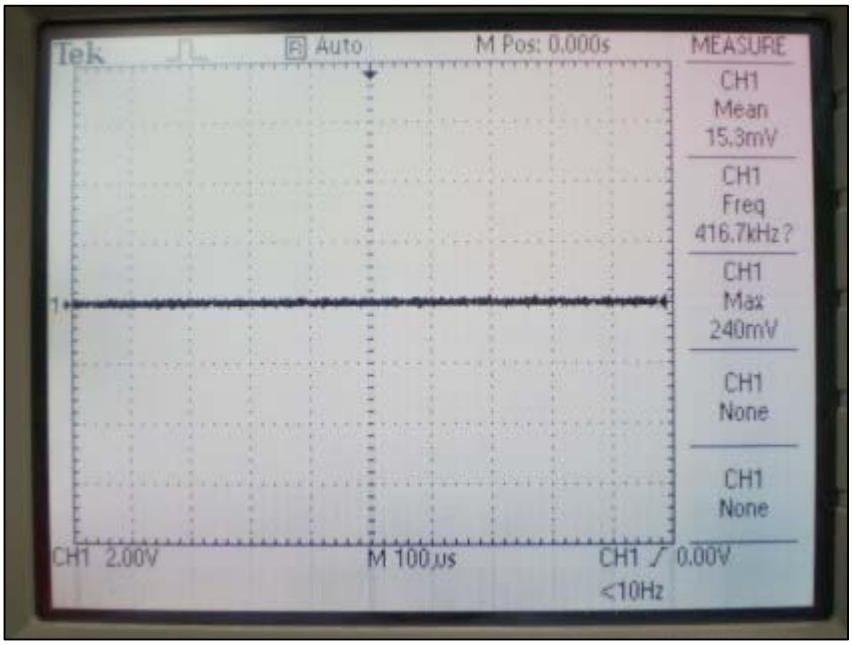


Figure 4.14: Output for Forward & Reverse 5V DC Motor

Figure 4.15: Output during Stop Condition

4.6.2 Stepper Motor Output Observation

This project use unipolar stepper motor. In a unipolar stepper motor, there are four separate electromagnets. To turn the motor, first coil "1" is given current, then it's turned off and coil 2 is given current, then coil 3, then 4, and then 1 again in a repeating pattern. Current is only sent through the coils in one direction; thus the name unipolar.

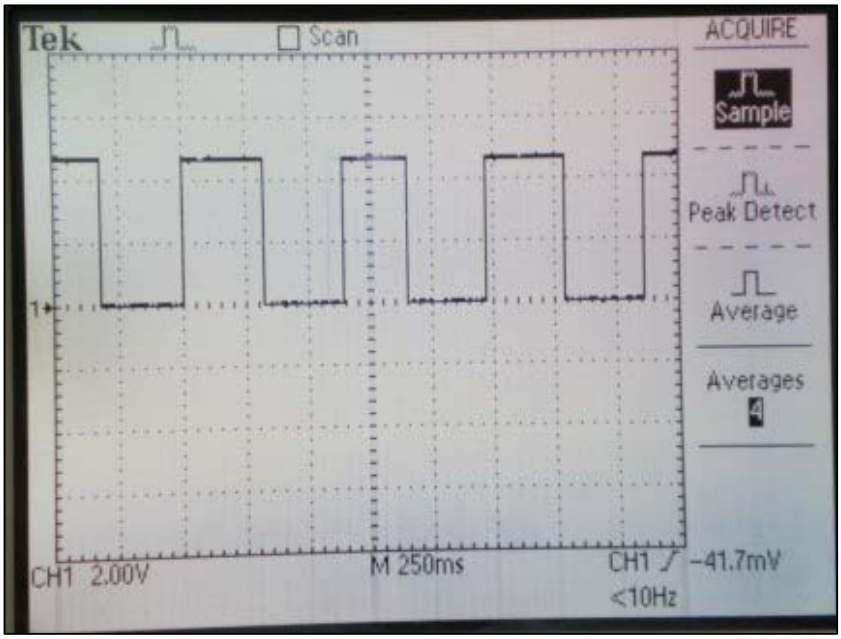


Figure 4.16: Speed 1 Output for Stepper Motor

To control stepper motor, the each of the coil must be supply with pulse width modulation (PWM). In this project there are four different width of the PWM where it determines the speed of the stepper motor. When the width of PWM supply is decrease or small, the speed of the stepper motor is increase and when the width of PWM is increase, the speed is decrease or slow. The output monitor using oscilloscope for PORTA in four different speeds is shown in figure 4.16 to figure 4.19.

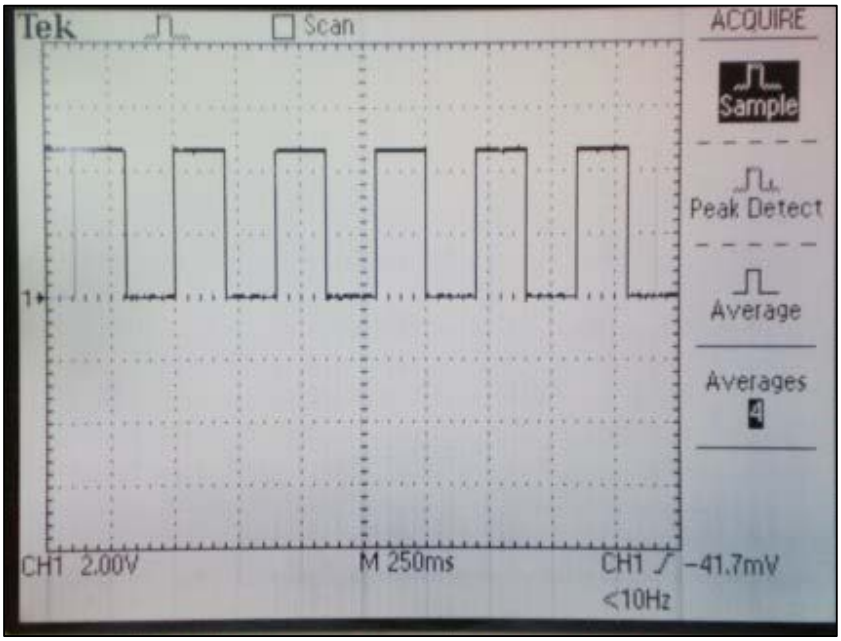


Figure 4.17: Speed 2 Output for Stepper Motor

The outputs for speed 1 that produce by PIC is like figure 4.16. The width of the PWM is program in PIC for 240ms gap between each on and off sequence. For speed 2 the PWM program in PIC for 160ms and for speed 3 the delay set to 90ms and lastly for speed 4 the delay set in PIC programming is 30ms.

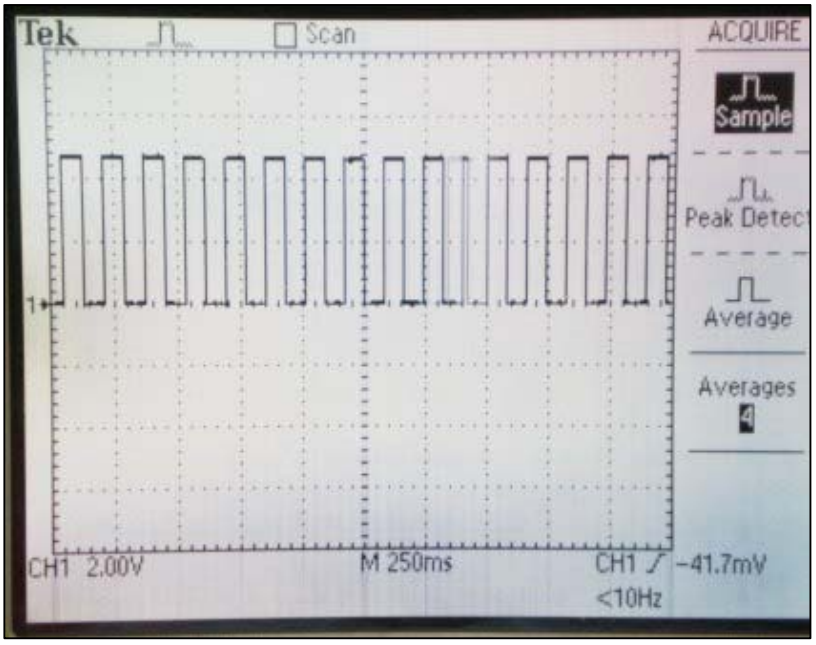


Figure 4.18: Speed 3 Output for Stepper Motor

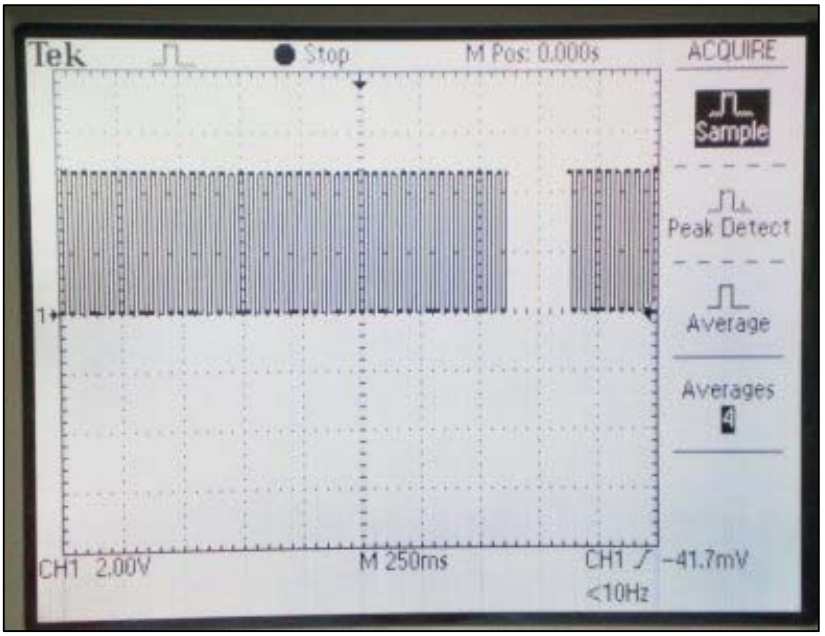


Figure 4.19: Speed 4 Output for Stepper Motor

CHAPTER 5

CONCLUSION AND RECOMMENDATION

5.1 Conclusion

The design and implementation of Motor control GUI has been presented in this project. The development of the MATLAB GUI using MATLAB GUIDE was done after detail study and analysis. Through the development of this project it has conclude that the MATLAB GUI can control the motor and interface with the device with the proper hardware installation and knowledge. The GUI seems easy to develop using simple pushbutton but it needs more knowledge and effort to do advanced programming on MATLAB GUI.

The objective of this project is to interface the MATLAB GUI and to control the DC motor is achieve. The main contribution of this project is interfacing the GUI with the device.

5.2 Future Recommendations

For the future recommendations, to improve this project, other features on GUI control can be added like slider to control the motor speed simultaneously with the slider change. For the information this project can be develop to control four stepper motor in time and can be use in store and retrieve application. Beside that other motor also can be added to be control through MATLAB GUI such as AC motor.

To make this project look more interesting, the close loop feedback from hardware is added. From this there are many things that we can develop such as the rotation or speed of the motor can be measure in MATLAB GUI. We also can include sensor to make specific task to detect object or detect change in surface condition if the user to apply in pick or place application using this software.

5.3 Costing and Commercialization

The cost of the project is divided into two parts. First part is for hardware cost and second part is for software. For hardware, it will cost approximately RM 100. For software cost it more on to get the license from MATLAB and usually the cost is high where the license must be renew by year.

This project can be used in picking and placing or store and retrieve application. Whereas commercially available software such as Flexible Manufacturing System and Computer Integrated Manufacturing but this project provides basic GUI capability for controlling that kind of the DC motor. This project approach of imparting advanced GUI capability to microcontrollers using MATLAB can be used to develop microcontroller-based low-cost control platforms. In addition, this approach can be used to impart GUI capability to any microcontroller that supports serial communication, such as the PIC series microcontrollers.

REFERENCE

[1] 17 January 2007, Citing Internet source URL http://www.mathworks.com/access/helpdesk/help/pdf_doc/matlab/buildgui.pdf

[2] Chapman, Stephen J, (2001) MATLAB Programming for Engineers, Brooks Cole.

[3] Creating Graphical User Interfaces (GUI's) with MATLAB
By Jeffrey A. Webb
OSU Gateway Coalition Member

[4] 19 January 2007, Citing Internet source URL http://www.webopedia.com/TERM/G/Graphical_User_Interface_GUI.html

[5] 20 January 2007, Citing Internet source URL http://en.wikipedia.org/wiki/Graphical_user_interface

[6] 20 January 2007, Citing Internet source URL http://en.wikipedia.org/wiki/Electric_motor

[7] Notes from subject - BEE2123 ELECTRICAL MACHINES
Prepared by Abu Zaharin Bin Ahmad

[8] 17 January 2007, Citing Internet source URL http://www.solarbotics.net/starting/200111_dcmotor/200111_dcmotor2.html

[9] http://en.wikipedia.org/wiki/PIC_microcontroller

[10] 18 January 2007, Citing Internet source URL
http://www.flhs.org.uk/Departments/technology/Word/PIC.htm

[11] PicBasic Pro Compiler, (2004). microEngineering Labs, Inc.
Available at: http://www.melabs.com

[12] Introduction to Graphical User Interface (GUI) MATLAB 6.5
By Prof. Abdulla Ismail Abdulla

[13] Creating Graphical User Interface Version 7
By The MathWork, Inc

[14] Yan-Fang Li, Saul Harari, Hong Wong, and Vikram Kapila (2004). Matlab-Based Graphical User Interface Development for Basic Stamp 2 Microcontroller Projects.
Department of Mechanical, Aerospace, and Manufacturing Engineering
Polytechnic University, Brooklyn, NY.

[15] Duane Hanselman & Bruce Littlefield (2005). Mastering MATLAB 7.
Pearson, Prentice Hall.

[16] Marc E. Herniter (2001). Programming In MATLAB. Northern Arizona University, Brooks/Cole.

[17] Robert DeMoyer and E. Eugene Mitchell (1999). Use of the MATLAB Graphical User Interface Development Environment for Some Control System Applications.

APPENDIX A

PIC Programming

```
INCLUDE "bs2defs.bas"
define  OSC 8
SerI var PORTC.0

p var byte
x var byte
y var byte
z var byte

TRISA = %00000000
TRISB = %00000000
TRISC = %00000000
TRISD = %00000000


Start:
   portc = %00000000
   Serin2 SerI, 84, [dec3 B0]

      select case B0

      case 001       'forward
      for x=1 to 80
      gosub qwe
      next x


      case 002       'reverse
      for x=1 to 80
      gosub ewq
      next x


      case 003       'forward
      for y=1 to 50
      gosub asd
      next y


      case 004        'reverse
      for y=1 to 50
      gosub dsa
      next y


      case 005        'froward
      for z=1 to 50
      gosub zxc
      next z


      case 006        'reverse
      for z=1 to 50
      gosub cxz
      next z


      case 007        'forward
      for z=1 to 50
      gosub rty
      next z


      case 008        'reverse
      for z=1 to 50
      gosub ytr
      next z


      case 010         'forward & reverse
      for x=1 to 50
      gosub qwe
      next x
      for y=1 to 50
      gosub ewq
      next y


      case 011
      for x=1 to 50
      gosub asd
      next x
      for y=1 to 50
      gosub dsa
      next y


      case 012
      for x=1 to 50
      gosub zxc
      next x
      for y=1 to 50
      gosub cxz
      next y


      case 013
      for x=1 to 50
      gosub rty
```

```
        next x
        for y=1 to 50
        gosub ytr
        next y


        case 020        'variable speed forward
        for x=1 to 50
        gosub qwe
        next x
        for y=1 to 50
        gosub zxc
        next y


        case 021        'variable speed forward
        for x=1 to 50
        gosub rty
        next x
        for y=1 to 50
        gosub asd
        next y

        case 022        'variable speed reverse
        for z=1 to 50
        gosub ewq
        next z
        for y=1 to 50
        gosub ytr
        next y


        case 023        'variable speed reverse
        for z=1 to 50
        gosub ytr
        next z
        for y=1 to 50
        gosub dsa
        next y


        CASE 030            'for dc motor
        high portd.0

        case 031
        portd = 0

        case 032
        high portd.1

        end select

goto start

        qwe:
           porta=%00000101
           pause 30
           porta=%00001001
           pause 30
           porta=%00001010
           pause 30
           porta=%00000110
           pause 30

        return

        ewq:
           porta=%00000101
           pause 30
           porta=%00000110
           pause 30
           porta=%00001010
           pause 30
           porta=%00001001
           pause 30

        return

        asd:
           porta=%00000101
           pause 90
           porta=%00001001
           pause 90
           porta=%00001010
           pause 90
           porta=%00000110
           pause 90

        return

        dsa:
           porta=%00000101
           pause 90
           porta=%00000110
           pause 90
           porta=%00001010
           pause 90
           porta=%00001001
           pause 90

        return

        zxc:
           porta=%00000101
           pause 160
           porta=%00001001
           pause 160
           porta=%00001010
           pause 160
           porta=%00000110
```

```
    pause 160

return

cxz:
    porta=%00000101
    pause 160
    porta=%00000110
    pause 160
    porta=%00001010
    pause 160
    porta=%00001001
    pause 160

return

rty:
    porta=%00000101
    pause 240
    porta=%00001001
    pause 240
    porta=%00001010
    pause 240
    porta=%00000110
    pause 240

return

ytr:
    porta=%00000101
    pause 240
    porta=%00000110
    pause 240
    porta=%00001010
    pause 240
    porta=%00001001
    pause 240

return

end
```

APPENDIX B

PIC 16F877 Data sheet

MICROCHIP

PIC16F7X7

28/40/44-Pin, 8-Bit CMOS Flash Microcontrollers with 10-Bit A/D and nanoWatt Technology

Low-Power Features:

- Power-Managed modes:
  - Primary Run (XT, RC oscillator, 76 µA, 1 MHz, 2V)
  - RC_RUN (7 µA, 31.25 kHz, 2V)
  - SEC_RUN (9 µA, 32 kHz, 2V)
  - Sleep (0.1 µA, 2V)
- Timer1 Oscillator (1.8 µA, 32 kHz, 2V)
- Watchdog Timer (0.7 µA, 2V)
- Two-Speed Oscillator Start-up

Oscillators:

- Three Crystal modes:
  - LP, XT, HS (up to 20 MHz)
- Two External RC modes
- One External Clock mode:
  - ECIO (up to 20 MHz)
- Internal Oscillator Block:
  - 8 user-selectable frequencies (31 kHz, 125 kHz, 250 kHz, 500 kHz, 1 MHz, 2 MHz, 4 MHz, 8 MHz)

Analog Features:

- 10-bit, up to 14-channel Analog-to-Digital Converter:
  - Programmable Acquisition Time
  - Conversion available during Sleep mode
- Dual Analog Comparators
- Programmable Low-Current Brown-out Reset (BOR) Circuitry and Programmable Low-Voltage Detect (LVD)

Peripheral Features:

- High Sink/Source Current: 25 mA
- Two 8-bit Timers with Prescaler
- Timer1/RTC module:
  - 16-bit timer/counter with prescaler
  - Can be incremented during Sleep via external 32 kHz watch crystal
- Master Synchronous Serial Port (MSSP) with 3-wire SPI™ and I^2C™ (Master and Slave) modes
- Addressable Universal Synchronous Asynchronous Receiver Transmitter (AUSART)
- Three Capture, Compare, PWM modules:
  - Capture is 16-bit, max. resolution is 12.5 ns
  - Compare is 16-bit, max. resolution is 200 ns
  - PWM max. resolution is 10 bits
- Parallel Slave Port (PSP) – 40/44-pin devices only

Special Microcontroller Features:

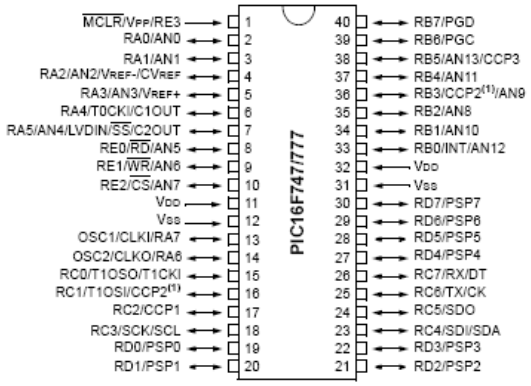
- Fail-Safe Clock Monitor for protecting critical applications against crystal failure
- Two-Speed Start-up mode for immediate code execution
- Power-on Reset (POR), Power-up Timer (PWRT) and Oscillator Start-up Timer (OST)
- Programmable Code Protection
- Processor Read Access to Program Memory
- Power-Saving Sleep mode
- In-Circuit Serial Programming™ (ICSP™) via two pins
- MPLAB® In-Circuit Debug (ICD) via two pins
- $\overline{MCLR}$ pin function replaceable with input only pin

Device	Program Memory (# Single-Word Instructions)	Data SRAM (Bytes)	I/O	Interrupts	10-bit A/D (ch)	Comparators	CCP (PWM)	MSSP		AUSART	Timers 8/16-bit
								SPI™	I^2C™ (Master)		
PIC16F737	4096	368	25	16	11	2	3	Yes	Yes	Yes	2/1
PIC16F747	4096	368	36	17	14	2	3	Yes	Yes	Yes	2/1
PIC16F767	8192	368	25	16	11	2	3	Yes	Yes	Yes	2/1
PIC16F777	8192	368	36	17	14	2	3	Yes	Yes	Yes	2/1

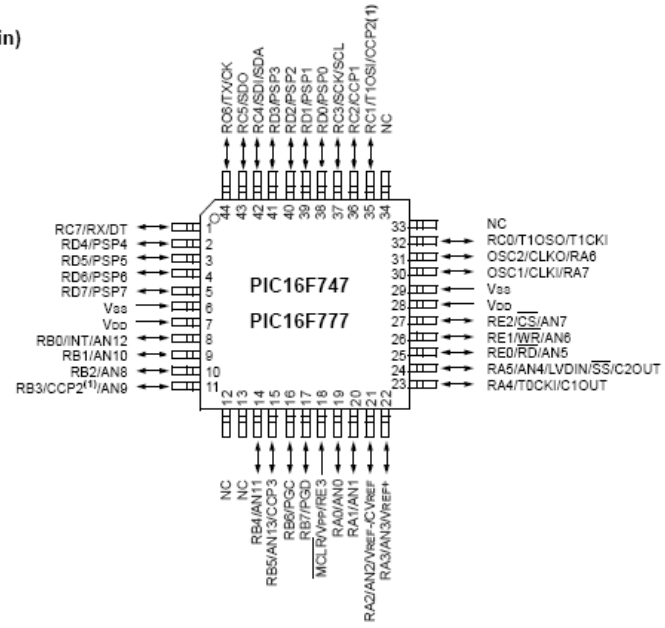
© 2004 Microchip Technology Inc.

Pin Diagrams (Continued)

PDIP (40-pin)



TQFP (44-pin)



Note 1: Pin location of CCP2 is determined by the CCPMX bit in Configuration Word Register 1.

1.0 DEVICE OVERVIEW

This document contains device specific information about the following devices:

- PIC16F737
- PIC16F747
- PIC16F767
- PIC16F777

PIC16F737/767 devices are available only in 28-pin packages, while PIC16F747/777 devices are available in 40-pin and 44-pin packages. All devices in the PIC16F7X7 family share common architecture with the following differences:

- The PIC16F737 and PIC16F767 have one-half of the total on-chip memory of the PIC16F747 and PIC16F777.
- The 28-pin devices have 3 I/O ports, while the 40/44-pin devices have 5.
- The 28-pin devices have 16 interrupts, while the 40/44-pin devices have 17.
- The 28-pin devices have 11 A/D input channels, while the 40/44-pin devices have 14.
- The Parallel Slave Port is implemented only on the 40/44-pin devices.
- Low-Power modes: RC_RUN allows the core and peripherals to be clocked from the INTRC, while SEC_RUN allows the core and peripherals to be clocked from the low-power Timer1. Refer to **Section 4.7 "Power-Managed Modes"** for further details.
- Internal RC oscillator with eight selectable frequencies, including 31.25 kHz, 125 kHz, 250 kHz, 500 kHz, 1 MHz, 2 MHz, 4 MHz and 8 MHz. The INTRC can be configured as a primary or secondary clock source. Refer to **Section 4.5 "Internal Oscillator Block"** for further details.
- The Timer1 module current consumption has been greatly reduced from 20 μA (previous PIC16 devices) to 1.8 μA typical (32 kHz at 2V), which is ideal for real-time clock applications. Refer to **Section 7.0 "Timer1 Module"** for further details.
- Extended Watchdog Timer (WDT) that can have a programmable period from 1 ms to 268s. The WDT has its own 16-bit prescaler. Refer to **Section 15.17 "Watchdog Timer (WDT)"** for further details.
- Two-Speed Start-up: When the oscillator is configured for LP, XT or HS, this feature will clock the device from the INTRC while the oscillator is warming up. This, in turn, will enable almost immediate code execution. Refer to **Section 15.17.3 "Two-Speed Clock Start-up Mode"** for further details.
- Fail-Safe Clock Monitor: This feature will allow the device to continue operation if the primary or secondary clock source fails by switching over to the INTRC.

The available features are summarized in Table 1-1. Block diagrams of the PIC16F737/767 and PIC16F747/777 devices are provided in Figure 1-1 and Figure 1-2, respectively. The pinouts for these device families are listed in Table 1-2 and Table 1-3.

Additional information may be found in the *"PICmicro® Mid-Range MCU Family Reference Manual"* (DS33023) which may be obtained from your local Microchip Sales Representative or downloaded from the Microchip web site. 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.

TABLE 1-1: PIC16F7X7 DEVICE FEATURES

Key Features	PIC16F737	PIC16F747	PIC16F767	PIC16F777
Operating Frequency	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz	DC – 20 MHz
Resets (and Delays)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)	POR, BOR (PWRT, OST)
Flash Program Memory (14-bit words)	4K	4K	8K	8K
Data Memory (bytes)	368	368	368	368
Interrupts	16	17	16	17
I/O Ports	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C	Ports A, B, C, D, E
Timers	3	3	3	3
Capture/Compare/PWM Modules	3	3	3	3
Master Serial Communications	MSSP, AUSART	MSSP, AUSART	MSSP, AUSART	MSSP, AUSART
Parallel Communications	—	PSP	—	PSP
10-bit Analog-to-Digital Module	11 Input Channels	14 Input Channels	11 Input Channels	14 Input Channels
Instruction Set	35 Instructions	35 Instructions	35 Instructions	35 Instructions
Packaging	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin QFN 44-pin TQFP	28-pin PDIP 28-pin SOIC 28-pin SSOP 28-pin QFN	40-pin PDIP 44-pin QFN 44-pin TQFP

© 2004 Microchip Technology Inc.

APPENDIX C

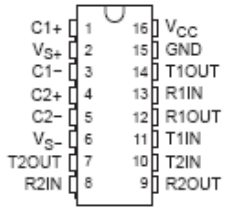
MAX232 Data Sheet

MAX232, MAX232I
DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047L – FEBRUARY 1989 – REVISED MARCH 2004

- Meets or Exceeds TIA/EIA-232-F and ITU Recommendation V.28
- Operates From a Single 5-V Power Supply With 1.0-μF Charge-Pump Capacitors
- Operates Up To 120 kbit/s
- Two Drivers and Two Receivers
- ±30-V Input Levels
- Low Supply Current . . . 8 mA Typical
- ESD Protection Exceeds JESD 22
  - 2000-V Human-Body Model (A114-A)
- Upgrade With Improved ESD (15-kV HBM) and 0.1-μF Charge-Pump Capacitors is Available With the MAX202
- Applications
  - TIA/EIA-232-F, Battery-Powered Systems, Terminals, Modems, and Computers

MAX232 . . . D, DW, N, OR NS PACKAGE
MAX232I . . . D, DW, OR N PACKAGE
(TOP VIEW)



description/ordering information

The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply TIA/EIA-232-F voltage levels from a single 5-V supply. Each receiver converts TIA/EIA-232-F inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V, a typical hysteresis of 0.5 V, and can accept ±30-V inputs. Each driver converts TTL/CMOS input levels into TIA/EIA-232-F levels. The driver, receiver, and voltage-generator functions are available as cells in the Texas Instruments LinASIC™ library.

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 70°C	PDIP (N)	Tube of 25	MAX232N	MAX232N
	SOIC (D)	Tube of 40	MAX232D	MAX232
		Reel of 2500	MAX232DR	
	SOIC (DW)	Tube of 40	MAX232DW	MAX232
		Reel of 2000	MAX232DWR	
	SOP (NS)	Reel of 2000	MAX232NSR	MAX232
−40°C to 85°C	PDIP (N)	Tube of 25	MAX232IN	MAX232IN
	SOIC (D)	Tube of 40	MAX232ID	MAX232I
		Reel of 2500	MAX232IDR	
	SOIC (DW)	Tube of 40	MAX232IDW	MAX232I
		Reel of 2000	MAX232IDWR	

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

LinASIC is a trademark of Texas Instruments.

PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

Copyright © 2004, Texas Instruments Incorporated

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

Function Tables

EACH DRIVER

INPUT TIN	OUTPUT TOUT
L	H
H	L

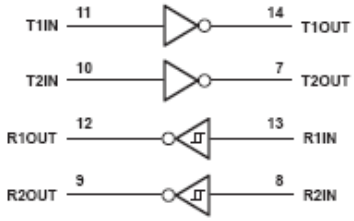
H = high level, L = low level

EACH RECEIVER

INPUT RIN	OUTPUT ROUT
L	H
H	L

H = high level, L = low level

logic diagram (positive logic)



absolute maximum ratings over operating free-air temperature range (unless otherwise noted)†

Input supply voltage range, V_{CC} (see Note 1) . . . . . −0.3 V to 6 V
Positive output supply voltage range, V_{S+} . . . . . V_{CC} − 0.3 V to 15 V
Negative output supply voltage range, V_{S-} . . . . . −0.3 V to −15 V
Input voltage range, V_I: Driver . . . . . −0.3 V to V_{CC} + 0.3 V
Receiver . . . . . ±30 V
Output voltage range, V_O: T1OUT, T2OUT . . . . . V_{S-} − 0.3 V to V_{S+} + 0.3 V
R1OUT, R2OUT . . . . . −0.3 V to V_{CC} + 0.3 V
Short-circuit duration: T1OUT, T2OUT . . . . . Unlimited
Package thermal impedance, θ_{JA} (see Notes 2 and 3): D package . . . . . 73°C/W
DW package . . . . . 57°C/W
N package . . . . . 67°C/W
NS package . . . . . 64°C/W
Operating virtual junction temperature, T_J . . . . . 150°C
Storage temperature range, T_{stg} . . . . . −65°C to 150°C

† Stresses beyond those listed under "absolute maximum ratings" may cause permanent damage to the device. These are stress ratings only, and functional operation of the device at these or any other conditions beyond those indicated under "recommended operating conditions" is not implied. Exposure to absolute-maximum-rated conditions for extended periods may affect device reliability.

NOTES: 1. All voltages are with respect to network GND.
2. Maximum power dissipation is a function of $T_J(max)$, θ_{JA}, and T_A. The maximum allowable power dissipation at any allowable ambient temperature is $P_D = (T_J(max) - T_A)/\theta_{JA}$. Operating at the absolute maximum T_J of 150°C can affect reliability.
3. The package thermal impedance is calculated in accordance with JESD 51-7.

recommended operating conditions

			MIN	NOM	MAX	UNIT
V_{CC}	Supply voltage		4.5	5	5.5	V
V_{IH}	High-level input voltage (T1IN,T2IN)		2			V
V_{IL}	Low-level input voltage (T1IN, T2IN)				0.8	V
R1IN, R2IN	Receiver input voltage				±30	V
T_A	Operating free-air temperature	MAX232	0		70	°C
		MAX232I	−40		85	

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature (unless otherwise noted) (see Note 4 and Figure 4)

PARAMETER		TEST CONDITIONS	MIN	TYP‡	MAX	UNIT
I_{CC}	Supply current	V_{CC} = 5.5 V, All outputs open, T_A = 25°C		8	10	mA

‡ All typical values are at V_{CC} = 5 V and T_A = 25°C.
NOTE 4: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

DRIVER SECTION

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature range (see Note 4)

PARAMETER			TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage	T1OUT, T2OUT	R_L = 3 kΩ to GND		5	7		V
V_{OL}	Low-level output voltage‡	T1OUT, T2OUT	R_L = 3 kΩ to GND			–7	–5	V
r_o	Output resistance	T1OUT, T2OUT	V_{S+} = V_{S-} = 0,	V_O = ±2 V	300			Ω
I_{OS}§	Short-circuit output current	T1OUT, T2OUT	V_{CC} = 5.5 V,	V_O = 0		±10		mA
I_{IS}	Short-circuit input current	T1IN, T2IN	V_I = 0				200	µA

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ The algebraic convention, in which the least-positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.
§ Not more than one output should be shorted at a time.
NOTE 4: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

switching characteristics, V_{CC} = 5 V, T_A = 25°C (see Note 4)

PARAMETER		TEST CONDITIONS	MIN	TYP	MAX	UNIT
SR	Driver slew rate	R_L = 3 kΩ to 7 kΩ, See Figure 2			30	V/µs
SR(t)	Driver transition region slew rate	See Figure 3		3		V/µs
	Data rate	One TOUT switching		120		kbit/s

NOTE 4: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

RECEIVER SECTION

electrical characteristics over recommended ranges of supply voltage and operating free-air temperature range (see Note 4)

PARAMETER			TEST CONDITIONS		MIN	TYP†	MAX	UNIT
V_{OH}	High-level output voltage	R1OUT, R2OUT	I_{OH} = –1 mA		3.5			V
V_{OL}	Low-level output voltage‡	R1OUT, R2OUT	I_{OL} = 3.2 mA				0.4	V
V_{IT+}	Receiver positive-going input threshold voltage	R1IN, R2IN	V_{CC} = 5 V,	T_A = 25°C		1.7	2.4	V
$V_{IT–}$	Receiver negative-going input threshold voltage	R1IN, R2IN	V_{CC} = 5 V,	T_A = 25°C	0.8	1.2		V
V_{hys}	Input hysteresis voltage	R1IN, R2IN	V_{CC} = 5 V		0.2	0.5	1	V
r_i	Receiver input resistance	R1IN, R2IN	V_{CC} = 5,	T_A = 25°C	3	5	7	kΩ

† All typical values are at V_{CC} = 5 V, T_A = 25°C.
‡ The algebraic convention, in which the least-positive (most negative) value is designated minimum, is used in this data sheet for logic voltage levels only.
NOTE 4: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

switching characteristics, V_{CC} = 5 V, T_A = 25°C (see Note 4 and Figure 1)

PARAMETER		TYP	UNIT
$t_{PLH(R)}$	Receiver propagation delay time, low- to high-level output	500	ns
$t_{PHL(R)}$	Receiver propagation delay time, high- to low-level output	500	ns

NOTE 4: Test conditions are C1–C4 = 1 µF at V_{CC} = 5 V ± 0.5 V.

APPENDIX D

Main Menu GUI Programming

```
function varargout = main(varargin)
% MAIN M-file for main.fig
%      MAIN, by itself, creates a new MAIN or raises the existing
%      singleton*.
%
%      H = MAIN returns the handle to a new MAIN or the handle to
%      the existing singleton*.
%
%      MAIN('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MAIN.M with the given input arguments.
%
%      MAIN('Property','Value',...) creates a new MAIN or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before main_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to main_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help main

% Last Modified by GUIDE v2.5 23-Oct-2007 00:44:22

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @main_OpeningFcn, ...
                   'gui_OutputFcn',  @main_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before main is made visible.
function main_OpeningFcn(hObject, eventdata, handles, varargin)
movegui ('center')
% This function has no output args, see OutputFcn.
% hObject    handle to figure
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to main (see VARARGIN)
% Choose default command line output for main
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes main wait for user response (see UIRESUME)
% uiwait(handles.figure1);

movegui('center')

% --- Outputs from this function are returned to the command line.
function varargout = main_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in pushbutton1.
function varargout=pushbutton1_Callback(h,eventdata,handles,varagin)
figure(motorcontrol)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

user_response = modaldlg1('Title','Confirm Close');
switch lower(user_response)
case 'no'
        % take no action
case 'yes'
        % Prepare to close GUI application window
        %         .
        %         .
        %         .
   close all

end



% --- Executes on button press in pushbutton3.
function varargout=pushbutton3_Callback(h,eventdata,handles,varagin)
figure(Info)
% hObject    handle to pushbutton3 (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton4.
function varargout=pushbutton4_Callback(h,eventdata,handles,varagin)
figure(credit)

% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1
[x,map]=imread('main','jpg');
image(x)
set(gca,'visible','off')
```

APPENDIX E

Motor Control Menu GUI Programming

```
function varargout = motorcontrol(varargin)
% MOTORCONTROL M-file for motorcontrol.fig
%      MOTORCONTROL, by itself, creates a new MOTORCONTROL or raises the existing
%      singleton*.
%
%      H = MOTORCONTROL returns the handle to a new MOTORCONTROL or the handle to
%      the existing singleton*.
%
%      MOTORCONTROL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in MOTORCONTROL.M with the given input arguments.
%
%      MOTORCONTROL('Property','Value',...) creates a new MOTORCONTROL or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before motorcontrol_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to motorcontrol_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help motorcontrol

% Last Modified by GUIDE v2.5 28-Oct-2007 14:19:18

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @motorcontrol_OpeningFcn, ...
                   'gui_OutputFcn',  @motorcontrol_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before motorcontrol is made visible.
function motorcontrol_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to motorcontrol (see VARARGIN)

% Choose default command line output for motorcontrol
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes motorcontrol wait for user response (see UIRESUME)
% uiwait(handles.figure1);

movegui('center')

% --- Outputs from this function are returned to the command line.
function varargout = motorcontrol_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

[x,map]=imread('bg1','jpg');
image(x)
set(gca,'visible','off')

% --- Executes on button press in pushbutton1.
function varargout=pushbutton1_Callback(h,eventdata,handles,varagin)
figure(pushbutton)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close(gcbf)
% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close (gcbf)

% --- Executes on button press in pushbutton4.
function varargout=pushbutton4_Callback(h,eventdata,handles,varagin)
figure(pulse)

% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(gcbf)

% --- Executes on button press in pushbutton5.
function varargout=pushbutton5_Callback(h,eventdata,handles,varagin)
```

```
figure(steppermotor)

% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(gcbf)


% --- Executes on button press in pushbutton6.
function varargout=pushbutton6_Callback(h,eventdata,handles,varagin)
figure(brushedbldcmotor)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
close(gcbf)


% --- Executes on button press in pushbutton7.
function varargout=pushbutton7_Callback(h,eventdata,handles,varagin)
figure(Help)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in help.
function varargout=help_Callback(h,eventdata,handles,varagin)
figure(help1)
% hObject    handle to help (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)



% --- Executes on button press in pushbutton9.
function varargout=pushbutton9_Callback(h,eventdata,handles,varagin)
figure(stepperinterface)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close(gcbf)


% --- Executes on button press in pushbutton10.
function varargout=pushbutton10_Callback(h,eventdata,handles,varagin)
figure(main)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close (gcbf)
```

APPENDIX F

5V DC Motor Control GUI Programming

```
function varargout = pushbutton(varargin)
% PUSHBUTTON M-file for pushbutton.fig
%      PUSHBUTTON, by itself, creates a new PUSHBUTTON or raises the existing
%      singleton*.
%
%      H = PUSHBUTTON returns the handle to a new PUSHBUTTON or the handle to
%      the existing singleton*.
%
%      PUSHBUTTON('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in PUSHBUTTON.M with the given input arguments.
%
%      PUSHBUTTON('Property','Value',...) creates a new PUSHBUTTON or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before pushbutton_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to pushbutton_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help pushbutton

% Last Modified by GUIDE v2.5 28-Oct-2007 14:25:49

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @pushbutton_OpeningFcn, ...
                   'gui_OutputFcn',  @pushbutton_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before pushbutton is made visible.
function pushbutton_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to pushbutton (see VARARGIN)

SerPIC=serial('COM1')   %define the port available
Check=SerPIC.status     %to check port status data
handles.status=Check    %store data
handles.op=SerPIC;  % store data
guidata(hObject, handles); %save data

set(findobj(gca,'Type','line','Color',[0 0 1]),'Color',[0 1 0],'LineWidth',2.5)
set(gca,'color',[0.027 0.702 0.894])
grid on;
axis([0 30 -10 10]);
xlabel('Time' );
ylabel('Voltage');
title('Voltage vs Time Linear signal');

% Choose default command line output for pushbutton
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes pushbutton wait for user response (see UIRESUME)
% uiwait(handles.figure1);

movegui('center')

% --- Outputs from this function are returned to the command line.
function varargout = pushbutton_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

% --- Executes on button press in open_close_port.
function open_close_port_Callback(hObject, eventdata, handles)
% hObject    handle to open_close_port (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of open_close_port

if (get(hObject,'Value')==get(hObject,'Max'));
   SerPIC=handles.op  % retrieve data
   set(SerPIC,'BaudRate',9600,'DataBits',8,'Parity','none','StopBits',1,'FlowControl','none');
   fopen(SerPIC)
   guidata(hObject,handles);     %save data    ;
else
   SerPIC=handles.op
   fclose(SerPIC)
   guidata(hObject,handles)

end
guidata(hObject,handles);

% --- Executes on button press in close_button.
```

```
function close_button_Callback(hObject, eventdata, handles)
% hObject    handle to close_button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

user_response = modaldlg2('Title','Confirm Close');
switch lower(user_response)
case 'no'
        % take no action
case 'yes'
        % Prepare to close GUI application window
        %               .
        %               .
        %               .
        delete(handles.figure1)
end

% --- Executes on button press in forward_PB.
function forward_PB_Callback(hObject, eventdata, handles)
% hObject    handle to forward_PB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op %retrieve data

m=1:0.1:1000;
n=-m;
c=m+n;
plot(c+5);
set(findobj(gca,'Type','line','Color',[0 0 1]),'Color',[0 1 0],'LineWidth',2.5)
set(gca,'color',[0.027 0.702 0.894])
grid on;
axis([0 30 -10 10]);
xlabel('Time' );
ylabel('Voltage');
title('Voltage vs Time Linear signal');

fprintf(SerPIC,'%s','031');
fprintf(SerPIC,'%s','030');

% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op %retrieve data
m=1:0.1:1000;
n=-m;
c=m+n;
plot(c-5);
set(findobj(gca,'Type','line','Color',[0 0 1]),'Color',[0 1 0],'LineWidth',2.5)
```

```
set(gca,'color',[0.027 0.702 0.894])
grid on;
axis([0 30 -10 10]);
xlabel('Time' );
ylabel('Voltage');
title('Voltage vs Time Linear signal');

fprintf(SerPIC,'%s','031');
fprintf(SerPIC,'%s','032');

% --- Executes on button press in stop_PB.
function stop_PB_Callback(hObject, eventdata, handles)
% hObject    handle to stop_PB (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op   %retrieve data

m=1:0.1:1000;
n=-m;
c=m+n;
plot(c);
set(findobj(gca,'Type','line','Color',[0 0 1]),'Color',[0 1 0],'LineWidth',2.5)
set(gca,'color',[0.027 0.702 0.894])
grid on;
axis([0 30 -10 10]);
xlabel('Time' );
ylabel('Voltage');
title('Voltage vs Time Linear signal');

fprintf(SerPIC,'%s','031');

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1

val = get(hObject,'Value');
str = get(hObject, 'String');
switch str{val};
case 'linear'
            handles.current_data = handles.peaks;
case 'pulse'
            handles.current_data = handles.membrane;


end
guidata(hObject,handles)

% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
```

```
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
SerPIC=handles.op
Check=handles.status
u=SerPIC.status

set(handles.text2,'String',u)
```

APPENDIX G

Stepper Motor Control GUI Programming

```
function varargout = stepperinterface(varargin)
% STEPPERINTERFACE M-file for stepperinterface.fig
%      STEPPERINTERFACE, by itself, creates a new STEPPERINTERFACE or raises the existing
%      singleton*.
%
%      H = STEPPERINTERFACE returns the handle to a new STEPPERINTERFACE or the handle
to
%      the existing singleton*.
%
%      STEPPERINTERFACE('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in STEPPERINTERFACE.M with the given input arguments.
%
%      STEPPERINTERFACE('Property','Value',...) creates a new STEPPERINTERFACE or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before stepperinterface_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to stepperinterface_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help stepperinterface

% Last Modified by GUIDE v2.5 28-Oct-2007 15:01:23

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @stepperinterface_OpeningFcn, ...
                   'gui_OutputFcn',  @stepperinterface_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
   gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before stepperinterface is made visible.
function stepperinterface_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to stepperinterface (see VARARGIN)

SerPIC=serial('COM1')   %define the port available
Check=SerPIC.status     %to check port status data
handles.status=Check    %store data
handles.op=SerPIC;  % store data
guidata(hObject, handles); %save data

% Choose default command line output for stepperinterface
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes stepperinterface wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = stepperinterface_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;


% --- Executes on button press in radiobutton1.
function radiobutton1_Callback(hObject, eventdata, handles)
% hObject    handle to radiobutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of radiobutton1
if (get(hObject,'Value')==get(hObject,'Max'));
   SerPIC=handles.op  % retrieve data
   set(SerPIC,'BaudRate',9600,'DataBits',8,'Parity','none','StopBits',1,'FlowControl','none');
   fopen(SerPIC)
   guidata(hObject,handles);     %save data    ;
else
   SerPIC=handles.op
   fclose(SerPIC)
   guidata(hObject,handles)

end
guidata(hObject,handles);

% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1
```

```
% --- Executes during object creation, after setting all properties.
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu2.
function popupmenu2_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu2 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu2


% --- Executes during object creation, after setting all properties.
function popupmenu2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end


% --- Executes on selection change in popupmenu3.
function popupmenu3_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu3 contents as cell array
%        contents{get(hObject,'Value')} returns selected item from popupmenu3


% --- Executes during object creation, after setting all properties.
function popupmenu3_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
```

```
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

% --- Executes on button press in close.
function close_Callback(hObject, eventdata, handles)
% hObject    handle to close (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

user_response = modaldlg2('Title','Confirm Close');
switch lower(user_response)
case 'no'
        % take no action
case 'yes'
        % Prepare to close GUI application window
        %               .
        %               .
        %               .
        delete(handles.figure1)
end

% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op %retrieve data
fprintf(SerPIC,'%s','007');

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','005');

% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','003');

% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','001');

% --- Executes on button press in pushbutton7.
```

```
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','008');

% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','006');

% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','004');

% --- Executes on button press in pushbutton10.
function pushbutton10_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','002');

% --- Executes on button press in pushbutton11.
function pushbutton11_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','013');

% --- Executes on button press in pushbutton12.
function pushbutton12_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','012');

% --- Executes on button press in pushbutton13.
function pushbutton13_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton13 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
```

```
fprintf(SerPIC,'%s','011');

% --- Executes on button press in pushbutton14.
function pushbutton14_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton14 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','010');

% --- Executes on button press in pushbutton15.
function pushbutton15_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton15 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','020');

% --- Executes on button press in pushbutton16.
function pushbutton16_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton16 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','021');

% --- Executes on button press in pushbutton17.
function pushbutton17_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton17 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','022');

% --- Executes on button press in pushbutton18.
function pushbutton18_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton18 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

SerPIC=handles.op
fprintf(SerPIC,'%s','023');

% --- Executes on button press in pushbutton19.
function pushbutton19_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton19 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
SerPIC=handles.op
Check=handles.status
y=SerPIC.status

set(handles.text8,'String',y)
```

APPENDIX H

Credit Menu GUI Programming

```
function varargout = Credit(varargin)
% CREDIT M-file for Credit.fig
%      CREDIT, by itself, creates a new CREDIT or raises the existing
%      singleton*.
%
%      H = CREDIT returns the handle to a new CREDIT or the handle to
%      the existing singleton*.
%
%      CREDIT('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in CREDIT.M with the given input arguments.
%
%      CREDIT('Property','Value',...) creates a new CREDIT or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before Credit_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to Credit_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Copyright 2002-2003 The MathWorks, Inc.

% Edit the above text to modify the response to help Credit

% Last Modified by GUIDE v2.5 21-Oct-2007 14:38:11

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Credit_OpeningFcn, ...
                   'gui_OutputFcn',  @Credit_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before Credit is made visible.
function Credit_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
% varargin   command line arguments to Credit (see VARARGIN)
movegui('center')
% Choose default command line output for Credit
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes Credit wait for user response (see UIRESUME)
% uiwait(handles.figure1);

whitebg('k')

% --- Outputs from this function are returned to the command line.
function varargout = Credit_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;

[a,map]=imread('nuar','jpg');
image(a)
set(gca,'visible','off')

% --- Executes during object creation, after setting all properties.
function axes5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes5

[x,map]=imread('mrsharfi','jpg');
image(x)
set(gca,'visible','off')

% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

close

function edit1_Callback(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit1 as text
%        str2double(get(hObject,'String')) returns contents of edit1 as a double


% --- Executes during object creation, after setting all properties.
function edit1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
```

```
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end

function edit2_Callback(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit2 as text
%        str2double(get(hObject,'String')) returns contents of edit2 as a double

% --- Executes during object creation, after setting all properties.
function edit2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc
   set(hObject,'BackgroundColor','white');
else
   set(hObject,'BackgroundColor',get(0,'defaultUicontrolBackgroundColor'));
end
```