PARALLEL-PORT CONTROLLER USING MATLAB

■ DEVANAND T.

Many projects have been published in EFY on parallel port and how to program it using C, C++ and Visual Basic. Here we attempt the same using Matlab. The advantage of Matlab is that it does not require any special device driver to run in Windows XP. It has rich 'help' documents and a large number of user community in Internet, such as Kluid and Matlab Central, which makes even an absolute beginner create an application quickly.

Described here are two simple application programs in Matlab for controlling eight LEDs and a unipolar stepper motor through the parallel port. Since the files are bulky, you need a PC with at least 512MB RAM to run Matlab.

Parallel port

The parallel port of the PC is a 25-pin, D-type connector, commonly known as LPT (line printer terminal) port, having three different ports, namely, data port, control port and status port. Access to the parallel port is via 25-pin, D-type female connector (DB25) as shown in Fig. 1. This female connector is normally available at the back side of your PC.

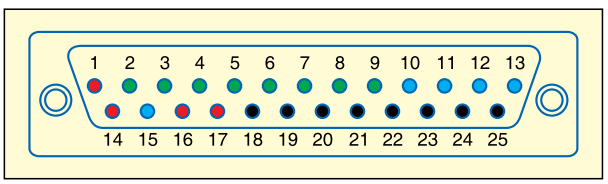

Fig. 1: 25-pin parallel-port connector

TABLE I
Pin Details of 25-pin, D-type Parallel-Port Connector

Pin	Register	Signal bit	Signal direction	Level	Hardware
1	Control	C 0	Output	Low	Inverted
2	Data	D 0	Output	High	Direct
3	Data	D 1	Output	High	Direct
4	Data	D 2	Output	High	Direct
5	Data	D 3	Output	High	Direct
6	Data	D 4	Output	High	Direct
7	Data	D 5	Output	High	Direct
8	Data	D 6	Output	High	Direct
9	Data	D 7	Output	High	Direct
10	Status	S 6	Input	High	Direct
11	Status	S 7	Input	Low	Inverted
12	Status	S 5	Input	High	Direct
13	Status	S 4	Input	High	Direct
14	Control	C 1	Output	Low	Inverted
15	Status	S 3	Input	High	Direct
16	Control	C 2	Output	High	Direct
17	Control	C 3	Output	Low	Inverted
18-25	Ground				

Note: Control register can be used as output as well as input

In order to access any port, you should have its port ID, i.e., the base address of the port. Here the data port has its base address as '0378' in hex format. The addresses of the three ports are in sequential order. So if the address of the data port is '0×378,' the addresses of the status port and the control port will be '0×378+1' and '0×378+2,' respectively.

Not all the 25 pins of the port are used for every application. Normally, the eight data lines (pins 2 through 9) along with the signal ground line are used. Pins 2 through 9 form the data port, thus making it an 8-bit wide port. It is a unidirectional port. That is, it can be used only to output the data to the connected device. Pins 1, 14, 16 and 17 form the control port. It is a bidirectional port, i.e., it can be used both to input and output the data to and from a device. Pins 10 to 13 and 15 form the status port. It is a unidirectional port, i.e., it can be used only for data input from a device. The various pin functions of the parallel port are shown in the Table I.

Circuit description

Fig. 2 shows the combined interfacing circuit for controlling the LEDs and the stepper motor through the parallel port. Here, LED1 through LED8 are connected to pins 2 through 9, respectively, of the parallel port. You can check these pins for proper working by clicking 'all on' and 'all off' button from the software (refer Fig. 3). All

PARTS LIST

Semiconductors:
IC1 - 7812 12V regulator
IC2-IC5 - MCT2E optocoupler
IC6 - ULN2003 buffer/driver
D1-D4 - 1N4001 rectifier diode

Resistors (all ¼-watt, ±5% carbon):
LED1-LED9 - 5mm light-emitting diode
R1 - 1-kilo-ohm
R2-R9 - 220-ohm

Capacitors:
C1 - 1000µF, 35V electrolytic
C2 - 0.1µF ceramic

Miscellaneous:
CON1 - 25-pin, D-type, parallel-port male connector
X1 - 230V AC primary to 15V, 1A secondary transformer
STM - 55SIM-25DAYG, 7.5 degrees/step, 12V stepper motor

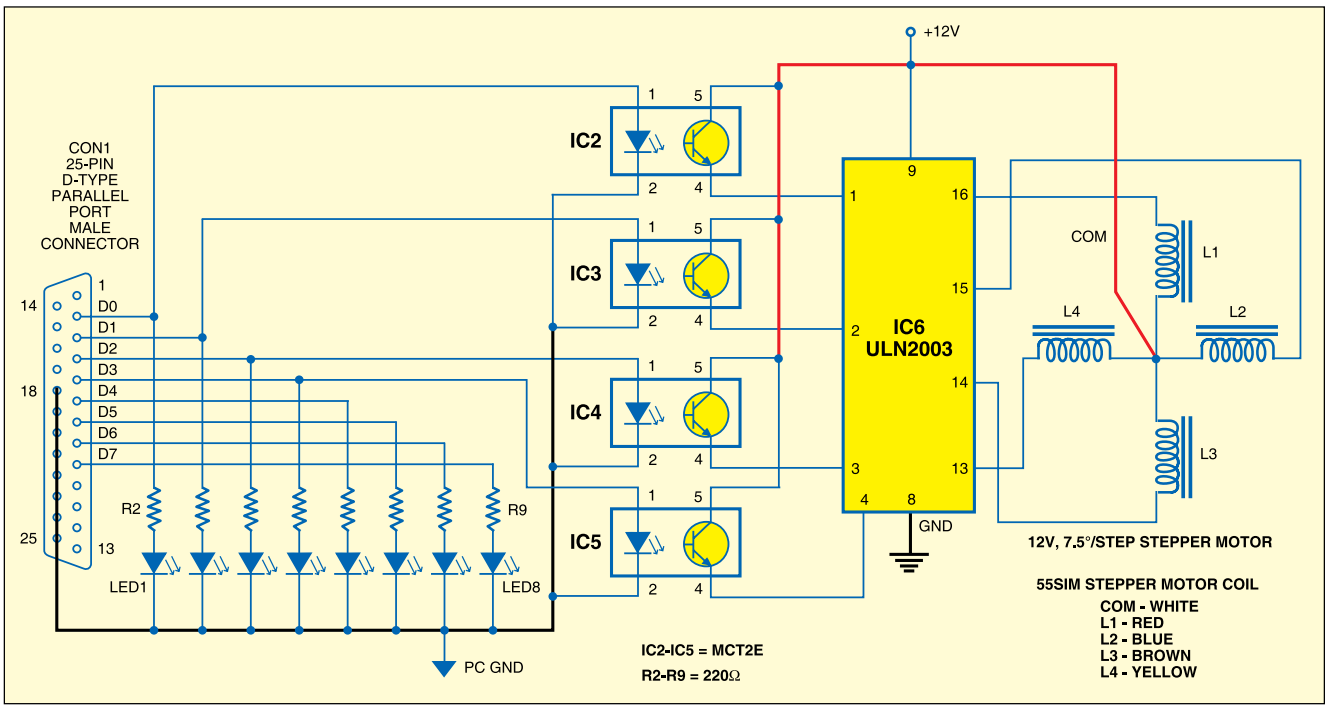

Fig. 2: Interfacing circuit for controlling LEDs and stepper motor

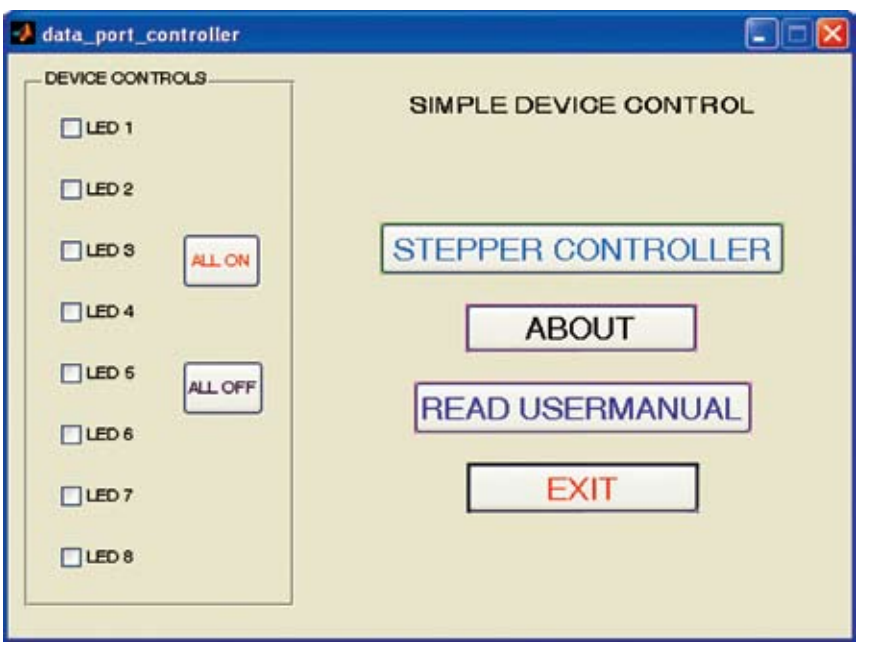

Fig. 3: Screenshot of LED control program output

the eight LEDs will glow on clicking 'all on' button and turn off on clicking 'all off' button. Each individual LED can also be made to glow/turn off by ticking the corresponding checkbox. The LED control application program works without external power supply in the circuit.

There is another application program to control the number of rotations, direction (clockwise or anticlockwise) and rotational speed of a stepper motor. Optocoupler MCT2E ICs (IC2 through IC5) are recommended for interfacing the stepper motor to the parallel port. Each of these ICs has an internal LED and a photo-transistor.

The external circuitries of the internal LED and photo-transistor of IC MCT2E differ. The internal LED is driven by the signal from the parallel port, whereas a separate 12V supply powers the photo-transistor (refer Fig. 4). Therefore it completely isolates the parallel port from the rest of the interfacing circuit. This ensures safety of the parallel port as it can get damaged in case of shorting of the external circuit.

IC ULN2003 (IC6) is used to drive the stepper motor. It is a high-voltage, high-current Darlington transistor array. It consists of seven npn Darlington pairs that feature high-voltage outputs with internal common-cathode clamp diodes for switching inductive loads. The collector-current rating of a single Darlington pair is 500 mA. The Darlington pairs can be paralleled for higher current capability. Other applications of this IC include relay driver, lamp driver, display driver, line driver and logic buffer.

Stepper motor

Different types of stepper motors with various specifications are available in

TABLE II
Colour Codes of the Wires for 55SIM-25DAYG Stepper Motor

Colour	Identification
White	Common
Red	L1
Blue	L2
Brown	L3
Yellow	L4

the market. The motor used here is a 55SIM-25DAYG unipolar stepper motor rated at 12V DC, with step angle of 7.5° per pulse. It is shown in Fig. 5.

This stepper motor has five wires for connection to the circuit. Out of the five wires, four are coming out from the coils of the stepper motors and the fifth is the centre-tapped common terminal.

Correct identification and proper connection of these wires are important for successful assembly of the stepper motor. The unipolar stepper motor has two centre-tapped coils. Their centre-tapped terminals are joined to make one common terminal. If you have a multimeter, adjust it in 200-ohm range and check resistances between **wires**. The resistance between the common terminal and the rest of the coil terminals will be around 36 ohms. The resistance between two coil terminals will be around 72 ohms. Connect these coils to the output terminals of the driver circuit in sequence.

To make the project more user-friendly, colour codes of wires for the 55SIM-25DAYG stepper motor are shown in Table II. Note that for the same ratings, different manufacturers may have their own coil specifications.

Matlab program

This program consists of 14 M-files and four figure files. However, only three main programs (parport.m, data_port_controller.m and data.m) have been explained and listed here. Matlab help files are very comprehensive. To get general help, type 'doc' command against '>>' prompt in 'command' window.

Matlab has a toolbox called DAQ (data aquisition). The 'dio = digitalio ('adaptor', ID)' creates the digital I/O object (dio) for the specific adaptor and for the hardware device with device identifier ID defined with an integer or a string. An ID can be LPT1, LPT2 or LPT3 only. LPT1 corresponds to dataport address 0×378.

So typing "dio=digitalio('parallel', 'LPT1')" on Matlab command line gives an output as shown in Fig. 6. 'addline' command adds hardware to the digital I/O object. The syntax is:

```
line = addline(obj, hwline, port,
'direction')
```

The 'putvalue(data, 1)' command makes LED1 glow.

The above codes are defined in 'parport.m' program as given below:

```
dio = digitalio('parallel', 'LPT1');
data1 = addline(dio, 0:7, 0, 'out');
putvalue(data1, value); %where value
b/w 0 and 255
```

Matlab comes with a 'Guide' tool.

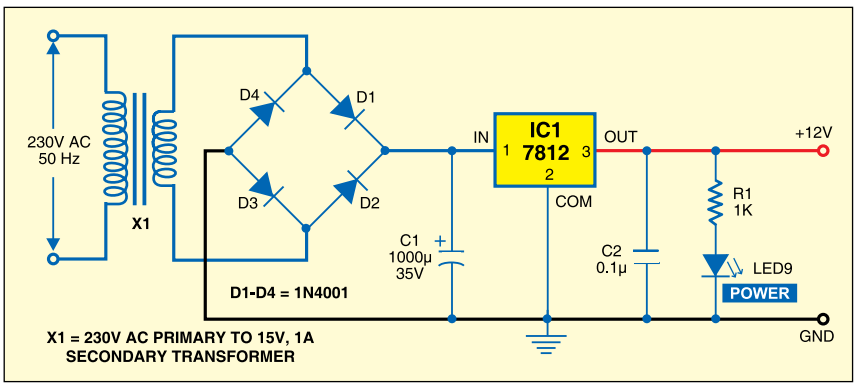

Fig. 4: 12V power supply

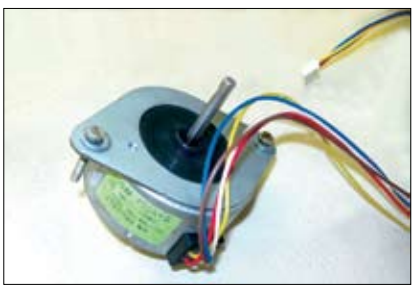

Fig. 5: Screenshot of stepper motor

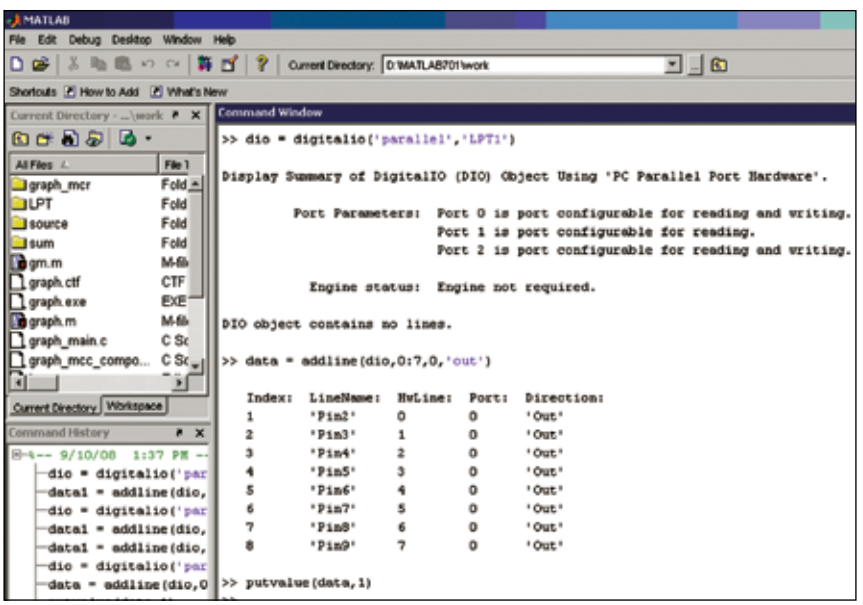

Fig. 6: Screenshot of Matlab command window

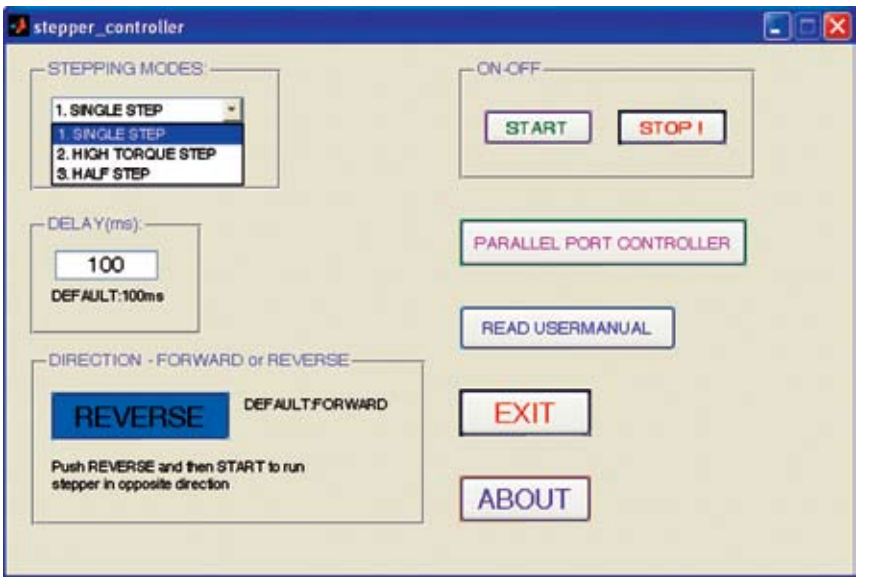

Fig. 7: Screenshot of stepper motor controller program output

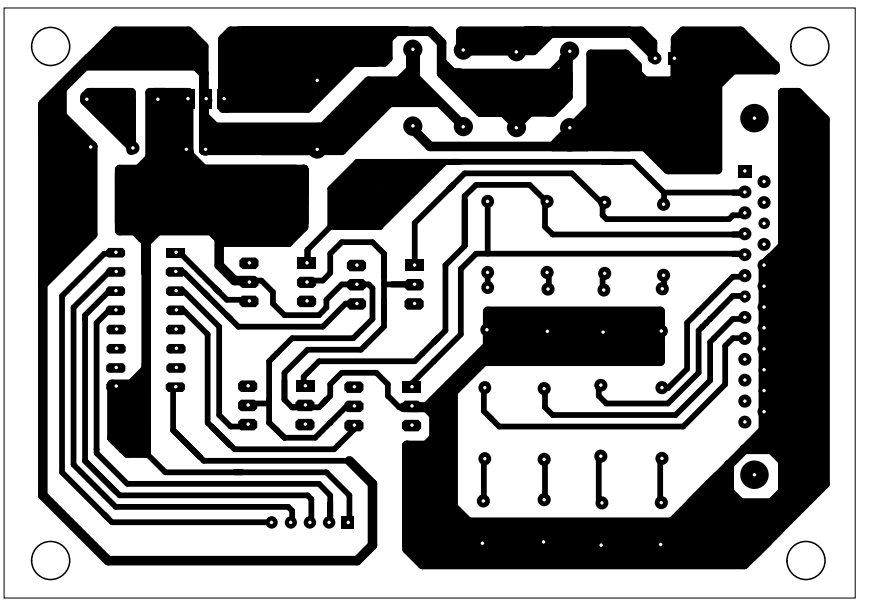

Fig. 8: Actual-size, single-side PCB for the parallel-port controller using Matlab

Guide (short for 'graphical user interface design environment') is similar to Visual Basic (VB) environment. We know that VB is an event-driven programming language which helps users to implement graphic interfaces.

Guide provides a set of tools for creating graphical user interfaces (GUIs). Just type 'Guide' against the prompt in Matlab command window. Select 'Blank GUI' option in the next window to launch Guide.

Using 'Layout' editor option from Guide, you can populate a GUI by clicking and dragging GUI components—such as axes, panels, buttons, text fields, sliders and so on—into the layout area. You can also create menus and context menus for the GUI. From the layout editor, you can resize the GUI, modify and align components, set tab order, view a hierarchical list of the component objects and set other GUI options.

Guide provides controls like push-button, slider, radio button, checkbox, edit text, static text, pop-up menu, list box, toggle-button, table, axes, panel, button group and ActiveX control. It even generates a skeleton M-file (plus figure file with a '.fig' extension) which programmers can further use to link other events through 'callback' function. The callback may be an event after mouse click on a button.

'callback' function is used in 'data_port_controller.m' program as given below:

```
function checkbox1_Callback(hObject,
eventdata, handles)
```

'checkbox1_Callback( )' is a callback function executed while you check and uncheck the checkboxes corresponding to LED1 through LED8 as shown in Fig. 3.

Variables can be used without declaration

```
handles.output = hObject;
handles.i1=1-handles.i1;
data
parport
guidata(hObject, handles);
```

Here, 'handles.i1' is a global variable. Every click on the checkbox makes 'handles.i1' toggled. 'data' is a script file which has only one line of code:

```
handles.data = handles.i1 * 1 +
handles.i2 * 2 + handles.i3 * 4 +
handles.i4 * 8 + handles.i5*16 +
handles.i6*32 + handles.i7 * 64 +
handles.i8 * 128 ;
```

'handles.data' is a global variable which updates a new value and then 'parport.m' script file is executed to output the value at the parallel port. Script file executes a sequence of commands rather than accepting or returning parameters. For example, the file generated by Guide is not a script file but a function m-file. Examples of function file include most commands in Matlab, such as sine and plot.

Two more important commands are 'set' and 'get.' For example, 'checkbox1' value sets 'handles.i1' as follows:

```
set(handles.checkbox1, 'value',
handles.i1);
```

In 'stepper_controller.m' file, the direction control variable returns the

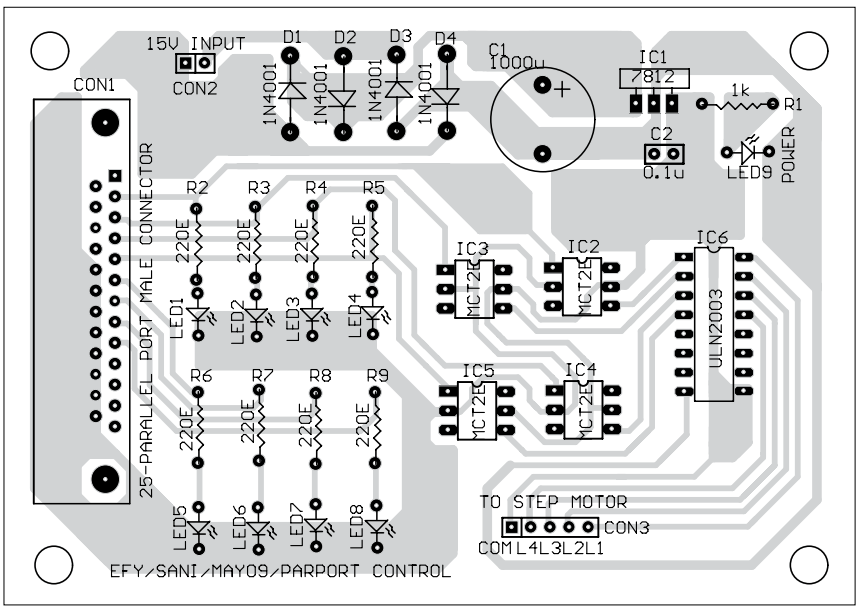

Fig. 9: Component layout for the PCB

toggle state of 'togglebutton1' and gives the value to 'handles.dir' using 'get' command:

```
handles.dir=get(handles.
togglebutton1, 'Value');
```

Running the program

Install Matlab 7.0.1 software in your system. Copy 'LPT port' folder from this month's EFY-CD to 'work' folder. Click the Matlab icon. Navigate to 'LPT port' folder from 'Current Directory' section, which is just on top of 'Command' window. To run the program, type 'data_port_controller' in 'Command' window.

The control of LED has been explained earlier. To control the stepper motor, click 'Stepper Controller' button in the program output shown in Fig. 3. You will get the program output as shown in Fig. 7. Here, you can control the stepper motor in three modes: single step, high torque and half step. Also the motor can be controlled either in forward or reverse direction. By default, the program is designed to rotate the stepper motor in reverse direction with 100ms delay speed. You can change these parameters as per your requirement.

Generating standalone program. To run the program without Matlab, create a standalone '.exe' file from 'M-file' and port it to other computers using MCR (Matlab component run time). For that, configure the Matlab compiler as follows: Type 'mbuild –setup' against the prompt. Matlab will ask for a few compiler options. Select the default Matlab compiler. Then type 'mcc –m mfilename.m –o exefilename'. Here 'mfilename' must be of function 'm-file' and 'exename' is the filename of the standalone executable file that will be generated. This command generates a few other files along with standalone '.exe' file. Now, you can run the executable file on a PC without Matlab by copying the generated files to that computer.

Construction

Assemble this project on any general-purpose PCB or on the PCB shown here in Figs 8 and 9. After mounting all the components on the PCB, fix the 25-pin male connector (CON1) on the PCB as shown in the layout. Use a 9-wire, 50cm FRC cable with 25-pin female connector on one end and male connector on the other end to connect between the board and the PC. Identify the stepper motor wires and connect the common wire to 12V and the rest of the coils in sequence as shown in the schematic diagram.

EFY note. All the relevant files of this project have been included in this month's EFY-CD.

DATA_PORT_CONTROLLER.M

```
function varargout = data_port_controller(varargin)
% DATA_PORT_CONTROLLER M-file for data_port_controller.fig
%      DATA_PORT_CONTROLLER, by itself, creates a new DATA_PORT_CONTROLLER or raises the existing
%      singleton*.
%
%      H = DATA_PORT_CONTROLLER returns the handle to a new DATA_PORT_CONTROLLER or the handle to
%      the existing singleton*.
%
%      DATA_PORT_CONTROLLER('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in DATA_PORT_CONTROLLER.M with the given input arguments.
%
%      DATA_PORT_CONTROLLER('Property','Value',...) creates a new DATA_PORT_CONTROLLER or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before data_port_controller_OpeningFunction gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to data_port_controller_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

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

% Last Modified by GUIDE v2.5 28-Sep-2008 10:23:03

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @data_port_controller_OpeningFcn, ...
                   'gui_OutputFcn',  @data_port_controller_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin & isstr(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 data_port_controller is made visible.
function data_port_controller_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
data_port_controller (see VARARGIN)

handles_i
handles.data=0;
% Choose default command line output for
data_port_controller
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

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


% --- Outputs from this function are re-
turned to the command line.
function varargout = data_port_controller_
OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning out-
put 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 han-
dles structure
varargout{1} = handles.output;

parport
%putvalue(data1,0)%initializes to zero at
every pin , XP normally has a value 255 at
data port

% --- Executes on button press in pushbut-
ton1.
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)

about_me

% --- Executes on button press in pushbut-
ton2.
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)
close

% --- Executes on button press in check-
box1.
function checkbox1_Callback(hObject, event-
data, handles)
% hObject      handle to checkbox1 (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 checkbox1

handles.output = hObject;
handles.i1=1-handles.i1;
data
parport

guidata(hObject, handles);
% --- Executes on button press in check-
box2.
function checkbox2_Callback(hObject, event-
data, handles)
% hObject      handle to checkbox2 (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 checkbox2
handles.output = hObject;
handles.i2=1-handles.i2;
data
parport
guidata(hObject, handles);

% --- Executes on button press in check-
box3.
function checkbox3_Callback(hObject, event-
data, handles)
% hObject      handle to checkbox3 (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 checkbox3
handles.output = hObject;
handles.i3=1-handles.i3;
data
parport
guidata(hObject, handles);

% --- Executes on button press in check-
box4.
function checkbox4_Callback(hObject, event-
data, handles)
% hObject      handle to checkbox4 (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 checkbox4
handles.output = hObject;
handles.i4=1-handles.i4;
data
parport
guidata(hObject, handles);

% --- Executes on button press in check-
box5.
function checkbox5_Callback(hObject, event-
data, handles)
% hObject      handle to checkbox5 (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 checkbox5
handles.output = hObject;
handles.i5 = 1-handles.i5;
data
parport
guidata(hObject, handles);

% --- Executes on button press in check-
box6.
function checkbox6_Callback(hObject, event-
data, handles)
% hObject      handle to checkbox6 (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 checkbox6
handles.output = hObject;
handles.i6=1-handles.i6;
data
parport
guidata(hObject, handles);

% --- Executes on button press in check-
box7.
function checkbox7_Callback(hObject, event-
data, handles)
% hObject      handle to checkbox7 (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 checkbox7
handles.output = hObject;
handles.i7=1-handles.i7;
data
parport
guidata(hObject, handles);

% --- Executes on button press in check-
box8.
function checkbox8_Callback(hObject, event-
data, handles)
% hObject      handle to checkbox8 (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 checkbox8
handles.output = hObject;
handles.i8=1-handles.i8;
data
parport
guidata(hObject, handles);

% --- Executes on button press in pushbut-
ton3.
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)
stepper_controller
close data_port_controller


% --- Executes on button press in pushbut-
ton4.
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)
 open('usermanual.pdf')


% --- Executes on button press in pushbut-
ton5.
function  pushbutton5_Callback(hObject,
eventdata, handles)
% 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)
handles.i1=1;
handles.i2=1;
handles.i3=1;
handles.i4=1;
handles.i5=1;
handles.i6=1;
handles.i7=1;
handles.i8=1;
data
parport
set(handles.checkbox1,  'value',handles.
i1);
set(handles.checkbox2,  'value',handles.
i2);
set(handles.checkbox3,  'value',handles.
i3);
set(handles.checkbox4,  'value',handles.
i4);
set(handles.checkbox5,  'value',handles.
i5);
set(handles.checkbox6,  'value',handles.
i6);
set(handles.checkbox7,  'value',handles.
i7);
set(handles.checkbox8,  'value',handles.
i8);

guidata(hObject, handles);

% --- Executes on button press in pushbut-
ton6.
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)
handles_i
data
parport
set(handles.checkbox1,  'value',handles.
i1);
set(handles.checkbox2,  'value',handles.
i2);
set(handles.checkbox3,  'value',handles.
i3);
set(handles.checkbox4,  'value',handles.
i4);
set(handles.checkbox5,  'value',handles.
i5);
set(handles.checkbox6,  'value',handles.
i6);
set(handles.checkbox7,  'value',handles.
i7);
set(handles.checkbox8,  'value',handles.
i8);

guidata(hObject, handles);
```