PRL Technical Note

Graphical User Interface for Stepper Motor based Filter Wheel control

T.A. Rajesh

July 2008

Physical Research Laboratory
Navrangpura, Ahmedabad - 380009

Contents

1. Introduction

The Stepper motors are used for precision positioning control in many applications like floppy drives, printers, process control instruments, robotic, machine tool control, etc. It requires a controller to generate a pulse sequence for its proper operation. The stepper motor driver unit can have (1) discrete components (2) microprocessor or microcontroller or (3) computer based circuit, which can accurately control the rotation direction, speed and the number of revolution of the stepper motor. This technical note illustrates a graphical user interface (GUI) for stepper motor based filter wheel controller developed in house.

The stepper motor, for its proper operation, requires a pre-programmed pulse sequence and necessary hardware driver. In the present discussion, the stepper motor hardware driver is made up of Darlington driver TP122 and the pulse sequence is generated using the computer. In the computer we can have either a digital input output (DIO) add-on card or use the existing parallel (LPT) port for outputting the pulse sequence to the stepper motor hardware driver unit. The present work is built using the parallel port for generating the pulse sequence.

The GUI is built using visual basic 6.0. The GUI allows the user to select the Stepper Motor ID (SMID), the rotation direction (CW or CCW), the stepping mode (half or full), inter step delay and the steeping loop. It also allows running the stepper motor in scanning mode in coordination with the encoder unit. The filter wheel encoder signal is fed into the computer through the same LPT (0X379H) port. The computer issues the free running signal to the stepper motor until the programmed encoder ID signal is detected and decoded back at the computer. The GUI is presently built to operate four stepper motors, but with the hardware driver unit, it can be upgraded to operate eight stepper motors.

The computer LPT port is opto isolated from the hardware driver unit as a safety precaution. The intelligence involved in the GUI is such that it keeps track of the user input parameters and then controls the flow of the program sequence to maintain the proper operation of the stepper motors. The present GUI is not based on hyper threading architecture and so only one stepper motor can be operated at a time.

2. Stepper Motor - Overview

A stepper motor is a brushless, synchronous electromechanical device that converts electrical pulses into discrete mechanical movements. The shaft or spindle of a stepper motor rotates in discrete step increments when electrical command pulses are applied to it in the proper sequence. 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. The speed of the motor shafts rotation is directly related to the frequency of the input pulses and the degree of rotation is directly related to the number of input pulses applied. The stator carries the magnetic field, which causes the rotor to align itself with the magnetic field. Sequentially energizing or "stepping" the stator coils can alter the magnetic field, which generates rotary motion.

Stepper Motor Advantages

a. The rotation angle of the motor is proportional to the input pulse.

b. Precise positioning and repeatability of movement since good stepper motors have an accuracy of 3 – 5% of a step and this error is non cumulative from one step to the next.

c. Excellent response to starting/ stopping/reversing.

d. It is possible to achieve very low speed synchronous rotation with a load that is directly coupled to the shaft.

e. A wide range of rotational speeds can be realized, as the speed is proportional to the frequency of the input pulses.

f. Because of the incremental nature of command and motion, stepper motors are easily adaptable to digital control applications.

g. No serious stability problems exist, even under open-loop control.

h. Torque capacity and power requirements can be optimized and electronic

switching can control the response.

i. Brushless construction has obvious advantages.

Stepper Motor Disadvantages

a. Resonance can occur if not properly controlled.

b. Not easy to operate at extremely high speeds.

c. They have low torque capacity compared to DC motors.

d. They have limited speed (limited by torque capacity and by pulse-missing problems due to faulty switching systems and drive circuits).

e. They have high vibration levels due to stepwise motion.

f. Large errors and oscillations can result when a pulse is missed under open-loop control.

Stepper Motor Switching Sequence

The stepper motor can be operated in three different stepping modes, namely, full-step, half-step, and micro step.

Full-Step

The stepper motor uses a four-step switching sequence. Here both phases of the motor are

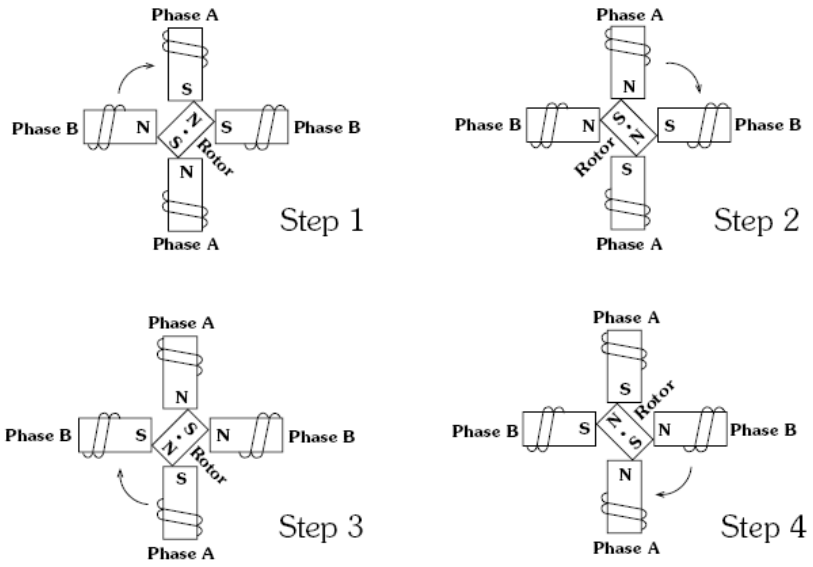


Figure 1. Full-stepping 90° step angle

always energized. However, only the polarity of one phase is switched at a time, as shown in figure 1. With two phases on stepping the rotor aligns itself between the "average" north and "average" south magnetic poles. Since both phases are always on, this method gives 41.4% more torque than "one phase on" stepping, but with twice the power input.

Half-Step

Another switching sequence for the stepper motor is called an eight-step or half step sequence. The main feature of this switching sequence is the doubling in the resolution of the stepper motor by causing the rotor to move half the distance it does when the full-step switching sequence is used. This means that a 200-step motor, which has a resolution of

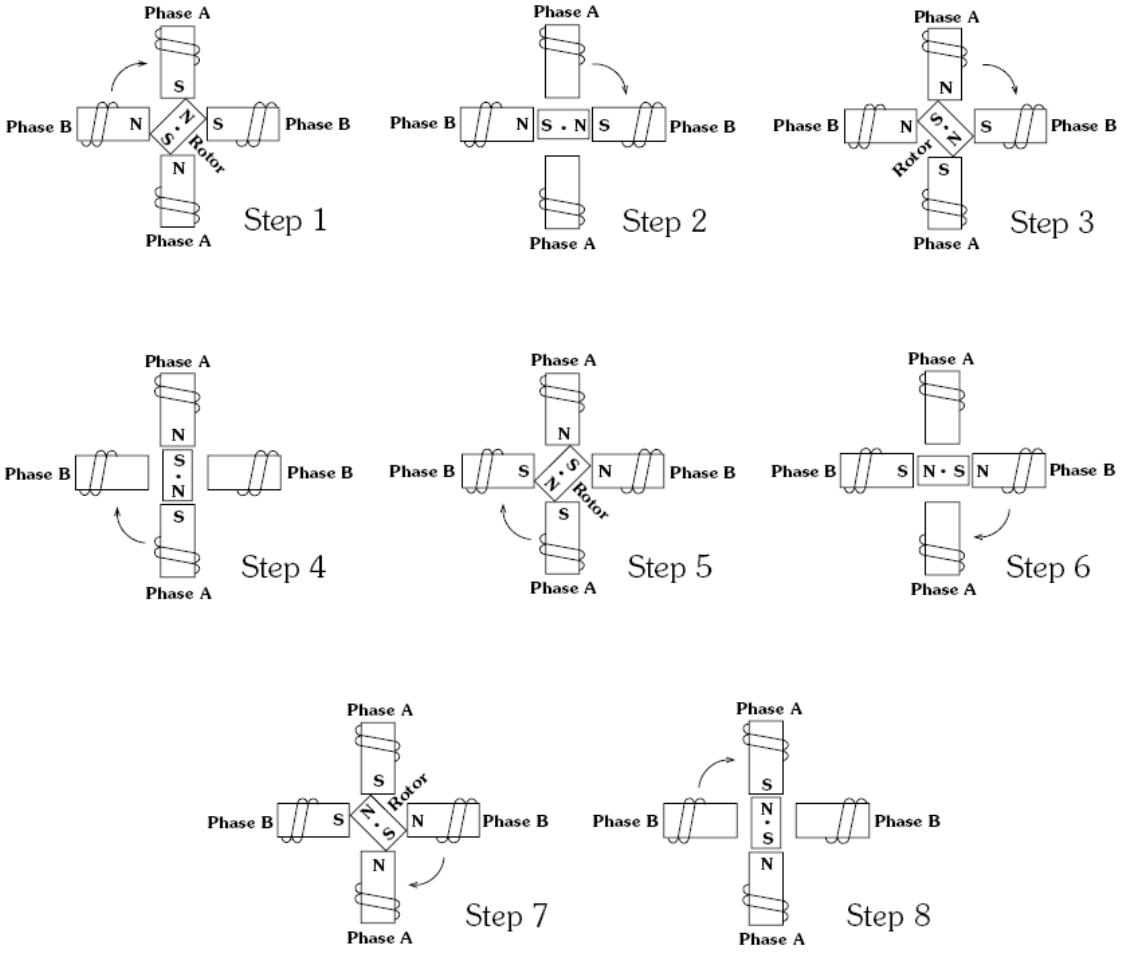


Figure 2. Half stepping 90° step angle is reduced to 45° with half-stepping.

1.8 degree, will have a resolution of 400 steps and 0.9 degree. The half-step switching sequence requires a special stepper motor controller, but it can be used with a standard hybrid motor. The way the controller gets the motor to reach the half-step is to energize both phases at the same time with equal current.

In this mode the stepper's full step angle is half. For example, a 90 degree stepping motor would move 45 degree. on each half step, figure 2. However, half stepping typically results in a 15% - 30% loss of torque depending on step rate when compared to the two phases on stepping sequence. Since one of the windings is not energized during each alternating half step, less electromagnetic force is exerted on the rotor resulting in a net loss of torque.

Micro Step Mode

The full-step and half step motors tend to be slightly jerky in their operation as the motor moves from step to step. The amount of resolution is also limited by the number of physical poles that the rotor can have. The amount of resolution (number of steps) can be increased by manipulating the current that the controller sends to the motor during each step. The current can be adjusted so that it looks similar to a sine wave. The current sent to each of the four sets of windings is timed so that there is always a phase difference with each other. The fact that the current to each individual phase increases and decreases like a sine wave and that is always out of time with the other phase will allow the rotor to reach hundreds of intermediate steps.

3. Hardware driver and control for Stepper Motor

The stepper motor hardware system controller consists of opto isolator cum buffer unit, selector cum control unit, drive unit, stepper motor with the filter wheel and filter wheel encoder unit as shown in figure 3.

We are operating the stepper motor in the unipolar configuration. The output pulse sequences for the clockwise and anticlockwise rotation of the stepper motor are generated using the computer parallel port. The parallel port consists of (1) Data port (2) Status port and (3) Control port. Each port has different address for communication. Here, data port

(0X378H) is used for sending the sequence of pulses to the stepper motor and the status port (0X379H) is used to get the status of filter wheel encoder. The output of the parallel port is normally TTL logic levels. The port can sink and source current around 12mA. The best bet is to use a buffer, so the least current is drawn from the parallel port.

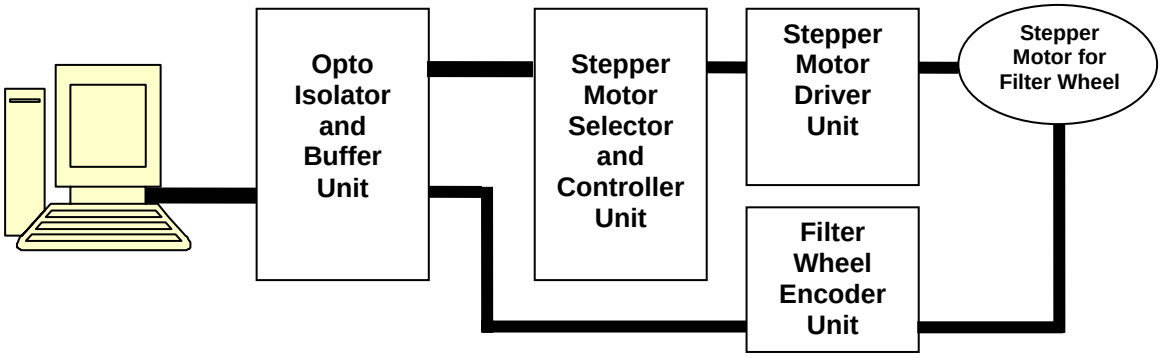


Fig 3: Schematic for Filter wheel control

The parallel port is isolated from the inductive load (stepper motor) using the optocoupler MCT2E as shown in figure 4. The optocouplers outputs are buffered using hex

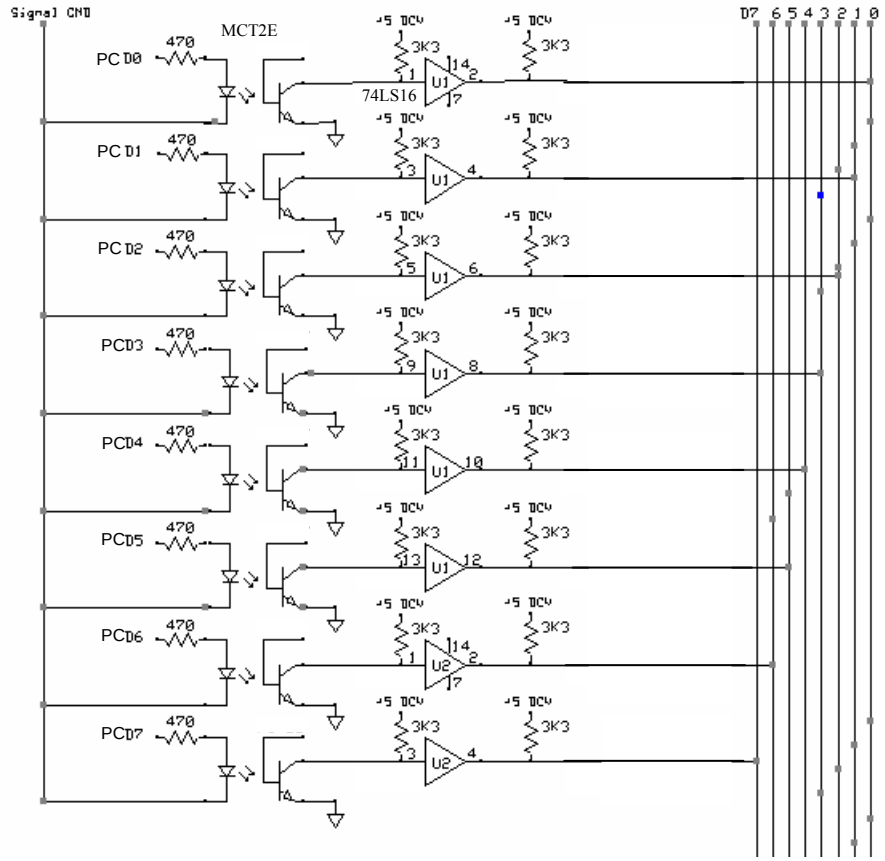


Fig 4: Optocoupler MCT2E port isolator circuit

inverter buffer 74LS16 to get the user data on the data bus, D0-D7. The present stepper motor selector unit is designed to control four stepper motors. The data bits D4-D6 are used for the selection of the stepper motor and D0-D3 bits are used for the stepper motor sequence control. The multi stepper motor controller is designed using Decoder/Demultiplexer 74LS138 as shown in figure 5. It decodes one-of-eight lines; based upon the conditions at the three binary select inputs and the three enable inputs. Two active-low and one active-high enable inputs reduce the need for external gates or inverters when expanding. The Decoder/Demultiplexer 74LS138 can be used to control

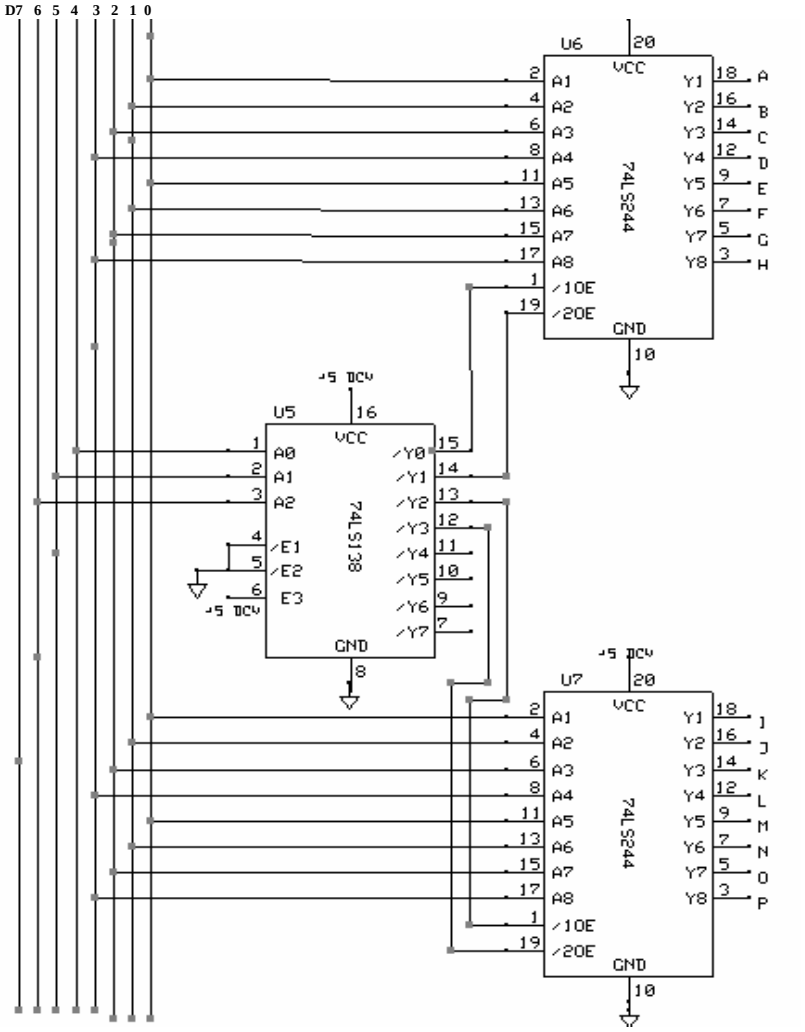


Fig. 5: 74LS138 based multi Stepper motor contoller circuit

a maximum of eight motor and in the present circuit, as shown in figure 5, it has been designed and tested for four motors only. The Decoder/Demultiplexer 74LS138 activates the respective buffer 74LS244 depending upon the input data format. The buffer 74LS244 output bits A-D, E-H, I-L and M-P generate the rotating sequence for stepper motor 1, 2, 3 and 4 respectively. Each of the 7LS4244 buffers is interfaced to the stepper motor driver unit through the transparent latch 74LS373 as shown in figure 6. The data bus is common to all the stepper motor driver latch circuit and the Decoder/Demultiplexer 74LS138 activates the respective stepper motor depending upon the input data. The present circuit describes the sequence control for a single stepper motor only.

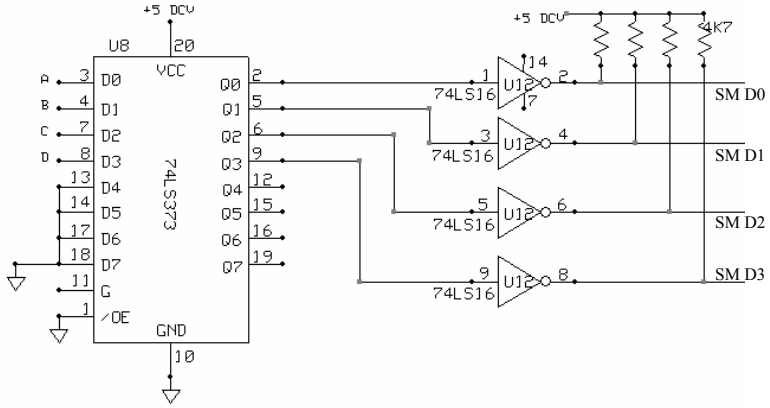


Fig. 6: 74373 latch circuit for Stepper motor driver

The data bits D0-D7 of parallel port being used to control the stepper motor for half and full mode stepping are listed in table 1 and 2 respectively. D0-D3 data bits are used for stepper motor rotation sequence, D4-D6 data bits are used for stepper motor selection and D6-D7 data bits are not used in the present configuration. The computer generates the desired address at the parallel port, the data is now floated on the hardware data bus where the decoder circuit activates the respective stepper motor latch and then issues the pulse sequence to the same stepper motor for the desired operation. The transparent latch 74LS373 also helps in operating the stepper motor in lock mode, where the stepper motor can be locked electronically in its last position by applying a low logic to transparent latch 74LS373 enable input (G). The lock signal can be issued from the computer as per the user requirement. In present design the stepper motor is not operated in the lock mode.

Stepper Motor	Spare	SM ID			SM Sequence				Address
	D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	0	0	1	0	1	5
	0	0	0	0	0	0	0	1	1
	0	0	0	0	1	0	0	1	9
	0	0	0	0	1	0	0	0	8
	0	0	0	0	1	0	1	0	10
	0	0	0	0	0	0	1	0	2
	0	0	0	0	0	1	1	0	6
	0	0	0	0	0	1	0	0	4
1	0	0	0	1	0	1	0	1	21
	0	0	0	1	0	0	0	1	17
	0	0	0	1	1	0	0	1	25
	0	0	0	1	1	0	0	0	24
	0	0	0	1	1	0	1	0	26
	0	0	0	1	0	0	1	0	18
	0	0	0	1	0	1	1	0	22
	0	0	0	1	0	1	0	0	20
2	0	0	1	0	0	1	0	1	37
	0	0	1	0	0	0	0	1	33
	0	0	1	0	1	0	0	1	41
	0	0	1	0	1	0	0	0	40
	0	0	1	0	1	0	1	0	42
	0	0	1	0	0	0	1	0	34
	0	0	1	0	0	1	1	0	38
	0	0	1	0	0	1	0	0	36
3	0	0	1	1	0	1	0	1	53
	0	0	1	1	0	0	0	1	49
	0	0	1	1	1	0	0	1	57
	0	0	1	1	1	0	0	0	56
	0	0	1	1	1	0	1	0	58
	0	0	1	1	0	0	1	0	50
	0	0	1	1	0	1	1	0	54
	0	0	1	1	0	1	0	0	52

Table 1: Half step mode data bit pattern

Stepper motor	Spare	SM ID			SM Sequence				Address
	D7	D6	D5	D4	D3	D2	D1	D0	
0	0	0	0	0	0	1	0	1	5
	0	0	0	0	0	1	1	0	6
	0	0	0	0	1	0	1	0	10
	0	0	0	0	1	0	0	1	9
1	0	0	0	1	0	1	0	1	21
	0	0	0	1	0	1	1	0	22
	0	0	0	1	1	0	1	0	26
	0	0	0	1	1	0	0	1	25
2	0	0	1	0	0	1	0	1	37
	0	0	1	0	0	1	1	0	38
	0	0	1	0	1	0	1	0	42
	0	0	1	0	1	0	0	1	41
3	0	0	1	1	0	1	0	1	53
	0	0	1	1	0	1	1	0	54
	0	0	1	1	1	0	1	0	58
	0	0	1	1	1	0	0	1	57

Table 2: Full step mode data bit pattern

The driving current of the computer parallel port is of the order of few mA. A driver circuit being used to amplify this driving current in order to drive the coil of a stepper motor.

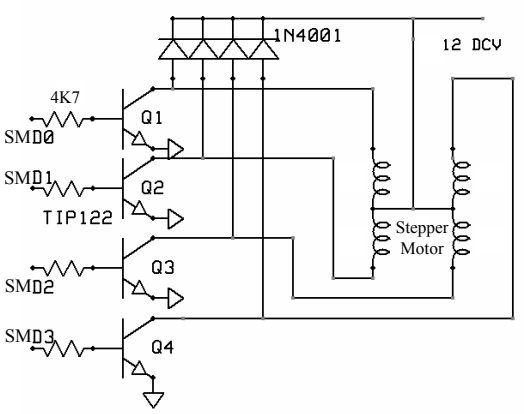


Fig. 7: TIP122 based Stepper motor driver circuit

The darlington drivers TIP122 are used for driving the stepper motor as shown in figure 7. Diodes IN4001 are used between the power supply and collector on the chip, to absorb reverse (or "back") EMF from the magnetic field collapsing when motor coils are switched off. In the present work, the stepper motor of make: STEP-SYN, type: SS-32-DC, specifications: 12V, 2A, 200 steps per revolution and torque 2Kg-cm is being used. The darlington drivers TIP122 based stepper motor driver circuit can be operated from 2.5 to 12 V with the respective stepper motor ratings. The same circuit has been tested successfully to operate the stepper motor TRW PN: 15629.001 of rating 2.5 VDC and 1.25A.

4. Software driver and control for Stepper Motor

The GUI is written in Microsoft Visual Basic 6.0 (VB6.0). The VB6.0 is a rapid application development tool, which allows programmers to create windows application in very little time. The software uses a window Dynamic Link Library (DLL) “Inpout32” to give direct access to hardware ports from user level programs. This DLL have the following features

1) Works seam less with all versions of windows (WIN 98, NT, 2000 and XP)

2) Using a kernel mode driver embedded in the DLL

3) No special software or driver installation required
4) Driver will be installed and configured when the DLL is loaded
5) No special APIs required except two functions Inp32 and Out32

The Inpout32.dll can work with all the windows versions without any modification in user code or the DLL itself. The DLL will check the operating system version when functions are called, and if the operating system is WIN9X, the DLL will use _inp() and _outp() functions for reading/writing through the parallel port. On the other hand, if the operating system is WIN NT, 2000 or XP, it will install a kernel mode driver and talk to parallel port through that driver. The user code will not be aware of the OS version on which it is running. The functions in the DLL are implemented in two source files, "inpout32drv.cpp" and "osversion.cpp". “osversion.cpp” checks the version of operating system. "inpout32drv.cpp" does installing the kernel mode driver, loading it, writing/ reading parallel port. The two functions exported from inpout32.dll are (1) 'Inp32', reads data from a specified parallel port register and (2) 'Out32', writes data to specified parallel port register. The other functions implemented in Inpout32.dll are

1) 'DllMain', called when dll is loaded or unloaded. When the dll is loaded, it checks the OS version and loads hwinterface.sys if needed.
2) 'Closedriver', close the opened driver handle, called before unloading the driver.
3) 'Opendriver', open a handle to hwinterface driver.
4) ‘Start’, starts the hwinterface service using Service Control Manager APIs.
5) 'SystemVersion' Checks the OS version and returns appropriate code.

The software generates the pulse sequence depending on the user input parameters SM ID, Step mode, stepping delay, stepping loop etc as seen in figure 8. The Inpout32.dll helps in putting the GUI generated pulse sequence at the parallel port. The GUI when loads initially, it puts the computer parallel port in the default mode. After putting the specified pulse sequence at the parallel port the GUI puts the parallel port again in its default state. The specified pulse sequence is available at the parallel port for a very short interval (few mill sec) in order to protect the port from any damage.

When the GUI is loaded it detects the hardware controller at its parallel port as shown in figure 8. The step delay with 1ms was tested with our filter wheel application, but the system can be upgraded for high speed requirement. The GUI also features stepper motor

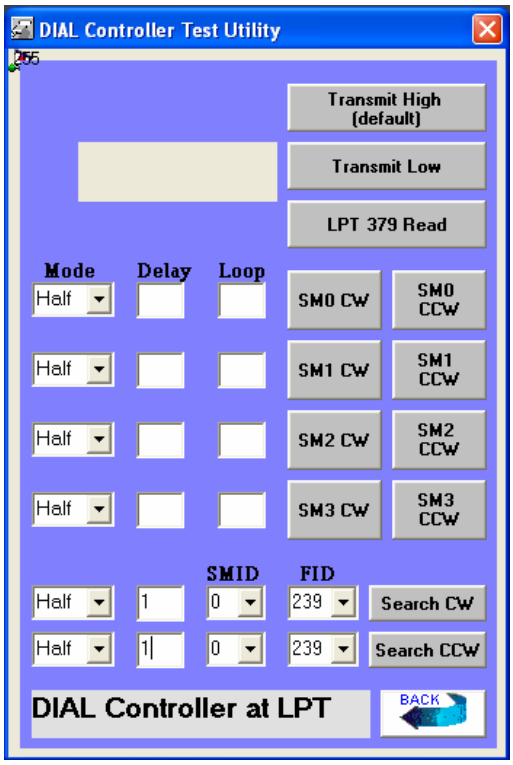


Fig. 8: Multi Stepper motor controller GUI screens shot

encoder section, where it reads the present load position. It search's the specified stepper motor position by reading the present motor position and comparing the status with the specified value, if it doesn't matches the GUI issues the next rotation command and again compare the motor position. When the specified and the present stepper motor position are same then the GUI issues the command to stop the rotation. The GUI contains the following Inputs or Control for the user

1. **Mode**: The stepper motor operating mode; Half or Full can be selected using this drop down combo box.
2. **Delay**: The stepper motors inter step timing can be set through this text box. The entered value is in ms and the minimum allowed value is 1ms.
3. **Loop**: The number of steps of the stepper motor rotation can be set through this text box. The entered value is an integer figure.

4. **SMx CW**: This command buttons enables the stepper motor (x=0to3) to rotate in clockwise direction with the specified mode, delay and loop.
5. **SMx CCW**: This command buttons enables the stepper motor(x=0to3) to rotate in counter clockwise direction with the specified mode, delay and loop.
6. **SMID**: The user can select the stepper motor as per the requirement using this drop down combo box.
7. **FID**: The user can select the optical filter in the optical path using this drop down combo box.
8. **Search CW**: This command button enables the stepper motor to rotate and search the required FID position in the clockwise direction.
9. **Search CCW**: This command button enables the stepper motor to rotate and search the required FID position in the counter clockwise direction.
10. **Transmit High (default)**: This command button allows the user to transmit data 255 at the parallel port, for the testing purpose.
11. **Transmit Low**: This command button allows the user to transmit data 0 at the parallel port, for the testing purpose.
12. **LPT 379 Read**: This command button read the data on the LPT status bits from the address 0x379.

5. Encoder for Filter Wheel

The anodized aluminum filter wheel of diameter 28cm, thickness 3mm, weight 210gms enclosed in an anodized aluminum box of dimension 34 x 34 x 4 cm. The filter wheel can hold eight 50.8mm filters. The motorised filter wheel allows to accurately switching between these eight filters. The motor is coupled to the filter wheel through a gear drive mechanism of ratio 1:10. It includes an optical edge sensor to index the position of the wheel to provide a reference position for the software as shown in figure 9. The filter wheel encoder is designed using 8 Input Priority Encoder 74LS148 chip. The filter

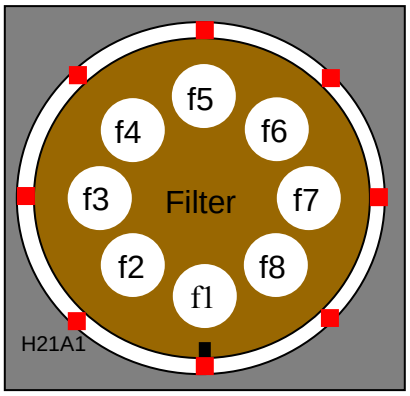


Fig. 9: Filter wheel with eight 50.8mm filters enclosed in an aluminum box

position is sensed using optical interrupter switch sensor H21A1 as shown in figure 10. The optical interrupter switch sensor H21A1 consist of a gallium arsenide infrared

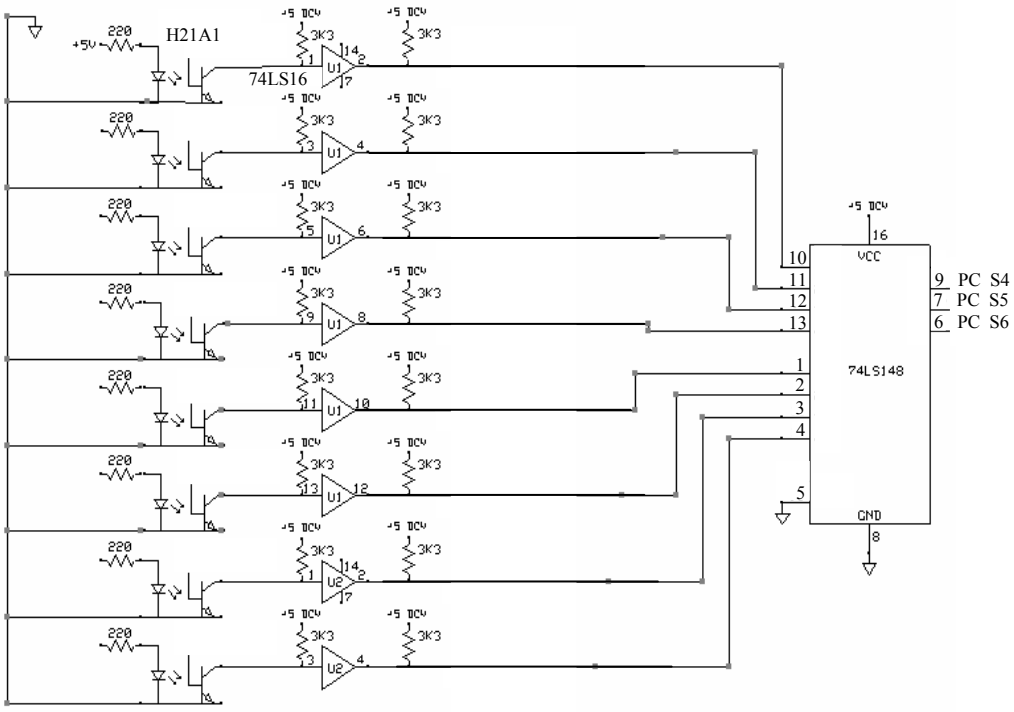


Fig. 10: Encoder circuit

emitting diode coupled with a silicon phototransistor in a plastic housing. The packaging system is designed to optimise the mechanical resolution, coupling efficiency, ambient light rejection and reliability. The gap in the housing provides a means of interrupting the signal with an opaque material, switching the output from an “ON” to an “OFF” state.

The optical interrupter switch sensor H21A1 outputs are buffered using hex Invertor Buffer 74LS16 to fed data into 8 Input Priority Encoder 74LS148 chip. The priority encoder 74LS148 output data bits are fed to the computer parallel status port S4-S6 (0X379H). The software using 'Inp32' function reads the LPT status port data. The digital buffers and inverters were used to match the digital signal as per the priority encoder requirement. The each optical interrupter switch status generates a fixed data pattern 143, 159, 175, 191, 207, 223 and 239D. The each data pattern corresponds to the individual filter on the filter wheel. The software is configured for the available data patterns and the system search's the motor position with reference to the set and the read data pattern.

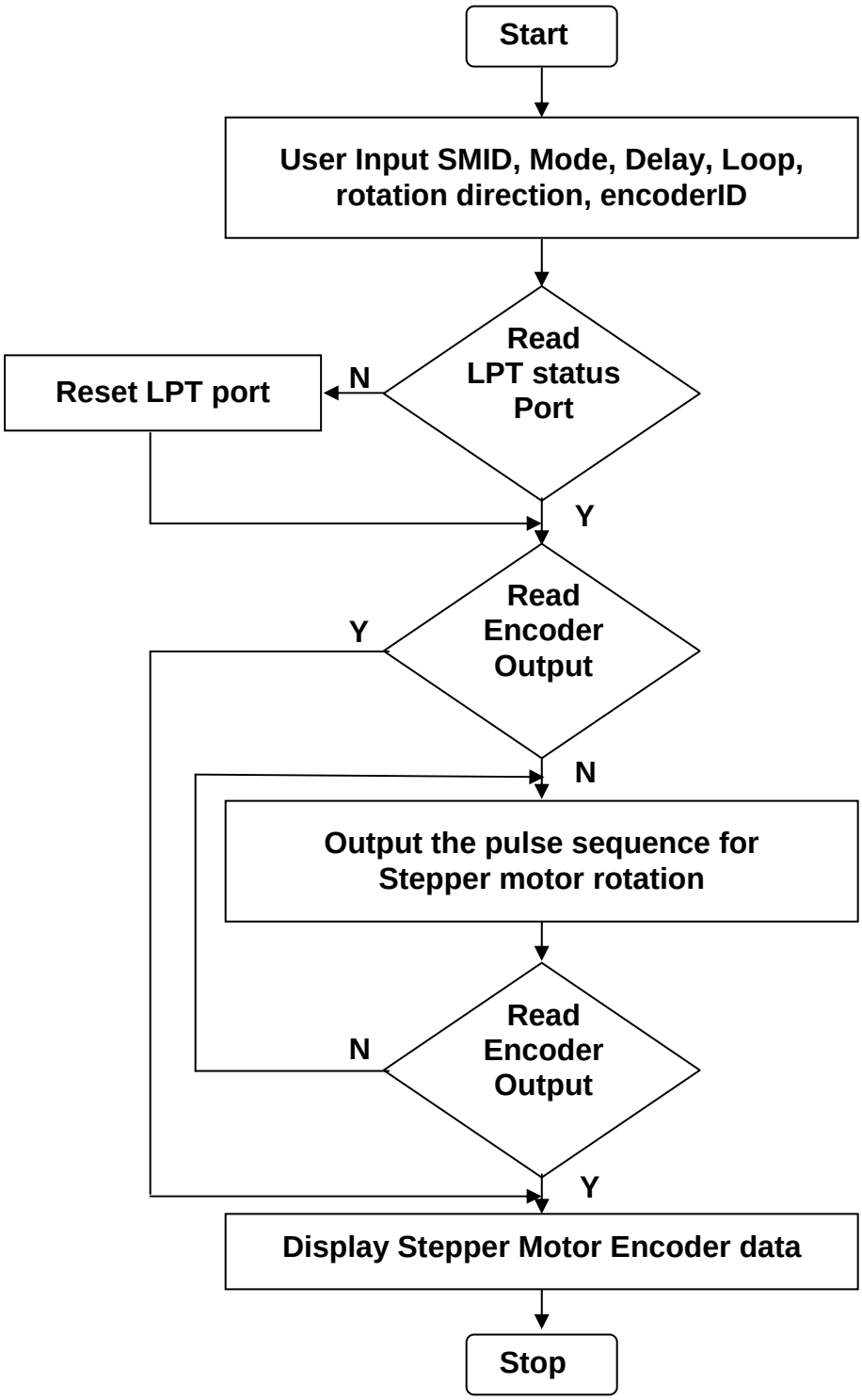
7. Summary

The present note discuss about the controlled rotation of the stepper motorised filter wheel using the windows platform based GUI. It discusses for only four-stepper motor control but the hardware and software controller can be upgraded to control more stepper motors. The program checks for the hardware and software status every time the user initiates any rotation sequence. It is a user-friendly windows package to be used with the described hardware controller. The program in the distribution form is available with the author for any user. The discussed work is only for any low speed application, in order to achieve a high speed operation the drive mechanism has to be redesigned for the user specific application.

Acknowledgement

I am thankful to all the members of the Space and Atmospheric Sciences Division of PRL who has helped directly or indirectly in carrying out this work on Computer parallel port based Stepper motor controlled Filter Wheel.

Appendix: A - Flowchart



Appendix: B - VB 6.0 Source Code

**********LPT form**

```
Dim Self As Boolean
Private Sub full_step_CW(b As Integer)
For i = 0 To 3

    Select Case b
      Case 0
      Out PortAddress, full_step(i)
      Case 16
      Out PortAddress, full_step(i) + 16
      Case 32
      Out PortAddress, full_step(i) + 32
      Case 48
      Out PortAddress, full_step(i) + 48
   End Select
   Sleep step_delay
Next i
End Sub
Private Sub full_step_CCW(b As Integer)
For i = 3 To 0 Step -1
   Select Case b
      Case 0
      Out PortAddress, full_step(i)
      Case 16
      Out PortAddress, full_step(i) + 16
      Case 32
      Out PortAddress, full_step(i) + 32
      Case 48
      Out PortAddress, full_step(i) + 48
   End Select
   Sleep step_delay
Next i
End Sub
Private Sub half_step_CW(b As Integer)

For i = 0 To 7
   Select Case b
      Case 0
      Out PortAddress, half_step(i)
      Case 16
      Out PortAddress, half_step(i) + 16
      Case 32
      Out PortAddress, half_step(i) + 32
      Case 48
      Out PortAddress, half_step(i) + 48
   End Select
   Sleep step_delay
Next i
End Sub
Private Sub half_step_CCW(b As Integer)
For i = 7 To 0 Step -1
   Select Case b
      Case 0
      Out PortAddress, half_step(i)
      Case 16
      Out PortAddress, half_step(i) + 16
      Case 32
      Out PortAddress, half_step(i) + 32
      Case 48
      Out PortAddress, half_step(i) + 48
   End Select
   Sleep step_delay
Next i
End Sub
Private Sub condition_half_step_CCW(ByVal b,
ByVal c)
If Inp(PortAddress1) <> c Then
For i = 7 To 0 Step -1
   Select Case b
      Case 0
      Out PortAddress, half_step(i)
      Case 16
      Out PortAddress, half_step(i) + 16
      Case 32
      Out PortAddress, half_step(i) + 32
      Case 48
      Out PortAddress, half_step(i) + 48
   End Select
   Sleep step_delay
Next i
End If
End Sub
Private Sub condition_full_step_CCW(ByVal b,
ByVal c)
If Inp(PortAddress1) <> c Then
For i = 3 To 0 Step -1
   Select Case b
      Case 0
      Out PortAddress, full_step(i)
      Case 16
      Out PortAddress, full_step(i) + 16
      Case 32
      Out PortAddress, full_step(i) + 32
      Case 48
      Out PortAddress, full_step(i) + 48
   End Select
   Sleep step_delay
Next i
End If

End Sub

Public Sub condition_half_step_CW(ByVal b,
ByVal c)

If Inp(PortAddress1) <> c Then
```

```
   For i = 0 To 7
      Select Case b
         Case 0
         Out PortAddress, half_step(i)
         Case 48
         Out PortAddress, half_step(i) + 48
      End Select
      Sleep step_delay
   Next i
End If
End Sub
Private Sub condition_full_step_CW(ByVal b,
ByVal c)
If Inp(PortAddress1) <> c Then
For i = 0 To 3
   Select Case b
      Case 0
      Out PortAddress, full_step(i)
      Case 16
      Out PortAddress, full_step(i) + 16
      Case 32
      Out PortAddress, full_step(i) + 32
      Case 48
      Out PortAddress, full_step(i) + 48
   End Select
   Sleep step_delay
Next i
End If
      Label7.Caption = Inp(PortAddress1)

End Sub

Private Sub Command1_Click() '//SM0 CW
step_delay = 0
motor_loop = 0

If Text1.Text <> "" Then
   If Text2.Text <> "" Then
      step_delay = Val(Trim$(Text1.Text))
      motor_loop = Val(Trim$(Text2.Text))
      Out PortAddress, 255
      For j = 1 To motor_loop
         Select Case Combo1.Text
            Case "Half"
            Call half_step_CW(0)
            Case "Full"
            Call full_step_CW(0)
         End Select
      Next j

   Else
      MsgBox "Loop Field Empty !!"
   End If
Else
MsgBox "Delay Field Empty !!"
End If
         Case 16
         Out PortAddress, half_step(i) + 16
         Case 32
         Out PortAddress, half_step(i) + 32
   Out PortAddress, 255
End Sub

Private Sub Command13_Click() '//Search CW
step_delay = 0
a1 = Val(Trim$(Combo5.Text))
Select Case Val(Trim$(Combo6.Text))
   Case 0
      a = 0
   Case 1
      a = 16
   Case 2
      a = 32
   Case 3
      a = 48
End Select

If Text9.Text <> "" Then
      step_delay = Val(Trim$(Text9.Text))
      Out PortAddress, 255

      Do
         Select Case Combo7.Text
            Case "Half"
            Call condition_half_step_CW(ByVal a,
ByVal a1)
            Case "Full"
            Call condition_full_step_CW(ByVal a,
ByVal a1)
         End Select
      'ext11.Text = Inp(PortAddress1)

      Loop Until Inp(PortAddress1) = a1

Else
MsgBox "Delay Field Empty !!"
End If
   Out PortAddress, 255


End Sub

Private Sub Command14_Click() '//Search CCW
step_delay = 0
a1 = Val(Trim$(Combo9.Text))
Select Case Val(Trim$(Combo8.Text))
   Case 0
      a = 0
   Case 1
      a = 16
   Case 2
      a = 32
```

```
   Case 3
      a = 48
End Select

If Text10.Text <> "" Then
      step_delay = Val(Trim$(Text10.Text))
      Out PortAddress, 255

      Do
         Select Case Combo10.Text
            Case "Half"
            Call   condition_half_step_CCW(ByVal
a, ByVal a1)
            Case "Full"
            Call    condition_full_step_CCW(ByVal
a, ByVal a1)
         End Select


      Loop Until Inp(PortAddress1) = a1

Else
MsgBox "Delay Field Empty !!"
End If
   Out PortAddress, 255
End Sub


Private Sub Command9_Click() '//SM0 CCW
step_delay = 0
motor_loop = 0
If Text1.Text <> "" Then
   If Text2.Text <> "" Then
      step_delay = Val(Trim$(Text1.Text))
      motor_loop = Val(Trim$(Text2.Text))
      Out PortAddress, 255
      For j = 1 To motor_loop
         Select Case Combo1.Text
            Case "Half"
            Call half_step_CCW(0)
            Case "Full"
            Call full_step_CCW(0)
         End Select
      Next j

   Else
      MsgBox "Loop Field Empty !!"
   End If
Else
MsgBox "Delay Field Empty !!"
End If
   Out PortAddress, 255

End Sub
Private Sub Command5_Click() '//SM1 CW
step_delay = 0
motor_loop = 0
If Text3.Text <> "" Then
   If Text4.Text <> "" Then
      step_delay = Val(Trim$(Text3.Text))
      motor_loop = Val(Trim$(Text4.Text))
      Out PortAddress, 255
      For j = 1 To motor_loop
         Select Case Combo2.Text
            Case "Half"
            Call half_step_CW(16)
            Case "Full"
            Call full_step_CW(16)
         End Select
      Next j

   Else
      MsgBox "Loop Field Empty !!"
   End If
Else
MsgBox "Delay Field Empty !!"
End If
   Out PortAddress, 255
End Sub

Private Sub Command10_Click() '//SM1 CCW
step_delay = 0
motor_loop = 0
If Text3.Text <> "" Then
   If Text4.Text <> "" Then
      step_delay = Val(Trim$(Text3.Text))
      motor_loop = Val(Trim$(Text4.Text))
      Out PortAddress, 255
      For j = 1 To motor_loop
         Select Case Combo2.Text
            Case "Half"
            Call half_step_CCW(16)
            Case "Full"
            Call full_step_CCW(16)
         End Select
      Next j

   Else
      MsgBox "Loop Field Empty !!"
   End If
Else
MsgBox "Delay Field Empty !!"
End If
   Out PortAddress, 255


End Sub
Private Sub Command6_Click() '//SM2 CW
step_delay = 0
motor_loop = 0
If Text5.Text <> "" Then
   If Text6.Text <> "" Then
```

```
      step_delay = Val(Trim$(Text5.Text))
      motor_loop = Val(Trim$(Text6.Text))
      Out PortAddress, 255
      For j = 1 To motor_loop
         Select Case Combo3.Text
         Case "Half"
         Call half_step_CW(32)
         Case "Full"
         Call full_step_CW(32)
         End Select
      Next j

   Else
      MsgBox "Loop Field Empty !!"
   End If
Else
MsgBox "Delay Field Empty !!"
End If
   Out PortAddress, 255

End Sub

Private Sub Command11_Click() '//SM2 CCW
step_delay = 0
motor_loop = 0
If Text5.Text <> "" Then
   If Text6.Text <> "" Then
      step_delay = Val(Trim$(Text5.Text))
      motor_loop = Val(Trim$(Text6.Text))
      Out PortAddress, 255
      For j = 1 To motor_loop
         Select Case Combo3.Text
         Case "Half"
         Call half_step_CCW(32)
         Case "Full"
         Call full_step_CCW(32)
         End Select
      Next j

   Else
      MsgBox "Loop Field Empty !!"
   End If
Else
MsgBox "Delay Field Empty !!"
End If
   Out PortAddress, 255


End Sub
Private Sub Command7_Click() '//SM3 CW
step_delay = 0
motor_loop = 0
If Text7.Text <> "" Then
   If Text8.Text <> "" Then
      step_delay = Val(Trim$(Text7.Text))
      motor_loop = Val(Trim$(Text8.Text))
      Out PortAddress, 255
      For j = 1 To motor_loop
         Select Case Combo4.Text
         Case "Half"
         Call half_step_CW(48)
         Case "Full"
         Call full_step_CW(48)
         End Select
      Next j

   Else
      MsgBox "Loop Field Empty !!"
   End If
Else
MsgBox "Delay Field Empty !!"
End If
   Out PortAddress, 255

End Sub

Private Sub Command12_Click() '//SM3 CCW
step_delay = 0
motor_loop = 0
If Text7.Text <> "" Then
   If Text8.Text <> "" Then
      step_delay = Val(Trim$(Text7.Text))
      motor_loop = Val(Trim$(Text8.Text))
      Out PortAddress, 255
      For j = 1 To motor_loop
         Select Case Combo4.Text
         Case "Half"
         Call half_step_CCW(48)
         Case "Full"
         Call full_step_CCW(48)
         End Select
      Next j

   Else
      MsgBox "Loop Field Empty !!"
   End If
Else
MsgBox "Delay Field Empty !!"
End If
   Out PortAddress, 255
End Sub

Private Sub Command2_Click() '// LPT 379
Read
Print Inp(PortAddress1)
Sleep 50
Select Case Inp(PortAddress1)
   Case 127
   Label1.Caption = "LPT Port open"
   Case 223
   Label1.Caption = "EPSON Printer at LPT"
   Case 255
```

```
   Label1.Caption = "DIAL Controller at LPT"
End Select
End Sub

Private Sub Command3_Click() '//Tx High(Default)
Out PortAddress, 255
Sleep 100
End Sub

Private Sub Command4_Click() '//Tx Low
Out PortAddress, 0
Sleep 100
End Sub


Private Sub Command8_Click() '//Back
LPT_Form.Enabled = False
LPT_Form.Visible = False
Form1.Enabled = True
Unload LPT_Form
End Sub


Private Sub Form_Load()
i = 0
j = 0
PortAddress = &H378
PortAddress1 = &H379

half_step(0) = 5
half_step(1) = 1
half_step(2) = 9
half_step(3) = 8
half_step(4) = 10
half_step(5) = 2
half_step(6) = 6
half_step(7) = 4

full_step(0) = 5
full_step(1) = 6
full_step(2) = 10
full_step(3) = 9

End Sub
```

References

1. Parallel Port Complete, Jan Axelson, Penram Publications
2. Visual Basic 6 Programming Black Book, Steven Holzner, Coriolis Publications
3. Visual Basic 6 Programming Bible, Eric A. Smith, Valor Whisler and Hank Marquis, Wiley Publications
4. Visual Basic 6 Desktop Applications, Michael Mckelvy, BPB Publications
5. www.msdn.microsoft.com/vbasic
6. www.devarticles.com/c/b/Visual-Basic
7. www.vbexplorer.com/VBExplorer
8. www.vb-helper.com
9. www.beyondlogic.org
10. Microprocessor Data Handbook, BPB Publications