PROGRAMMABLE MULTIPLE STEPPER MOTOR CONTROL DEVICE WITH BASIC STAMP MICROCONTROLLER

- **Arun Dayal Udai**

INTRODUCTION

A number of control based applications used in manufacturing, hobby projects and automation processes including robotic operations require precise position control, where inherent precision of stepper motors have proved to be useful. Various robots like robotic arm, automated material handling robot, welding robots, camera positioning systems in robots, various parts of humanoid robot etc. are examples where some of the typical application of precise positioning and control are involved.

Stepper motors are preferred in some of above cases due to their high torque per step movement compare to the servo motors of similar size. Although, stepper motor has limitation of step resolution, it can be overcome by adding mechanical gearbox or by micro stepping. The stepper motor is easy to drive in open-loop, mechanically simple, and responds to a digital input which makes it possible to control directly by computer without any encoder. One of the unique characteristic of stepper motor is its holding torque, without any additional circuitry. Holding Torque is the minimum torque required to turn the motor from stand still position when energized. A DC servo motor is expensive in comparison to stepper motors. Moreover, use of servo motor requires additional cost for the complicated closed loop control system. Due to this, stepper motor becomes convenient and friendly for small sized projects with limited cost.

Parallel port control application has limitation on use of number of motors that can be controlled along with the desired communication speed at which they should be controlled.

In the present scheme, the complete motion sequence of the stepper motors can be embedded to the EEPROM of the circuit or can be run directly through computer for any random motion. Once the sequence is downloaded, the circuit may be detached from the computer and the sequence can be regenerated in cycle when stand alone.

If the torque requirement is pre-calculated precisely, the torque seldom increases the pre-calculated value and the chances of motor loosing its position are very less. In the presented circuit, a serial port interface circuit for controlling multiple stepper motors is used. The circuit is very flexible with digital logic that can be easily modified for any special needs with similar layouts and logic.

CIRCUIT DESIGN AND IMPLEMENTATION

A Serial port interface circuit is designed with serial connector shown in Figure 3. Once the sequence of motor steps is calculated, it is communicated via serial port from any of the PC's COM# Port (RS232) through Computer Program. This data is decoded by the microcontroller and it sends the pulses to the corresponding motor required for its forward or reverse driving. The first digit of the serial data contains the motor direction and the remaining digits form the motor number to be driven. E.g. a serial data 91 will be decoded as 9^{th} motor in forward direction and similarly 90 would mean the 9^{th} motor in reverse direction. As long as the motor is not moved by the microcontroller, program remains activated with the last phase status of the motor coils. This holds the motor in its current position. With the current layout eight motors can be controlled and may be very easily extended for sixteen motors by cascading of shift registers to be explained later.

Microcontroller

The microcontroller used is Parallax BASIC Stamp 2px24 due to its simplicity in its programming language i.e. PBASIC. It serves the purpose of synchronous serial communication between microcontroller and the shift register 74HC595. It has also a dedicated serial communication port for interfacing with computer which can take in asynchronous serial data from computer by a single SERIN command at 19200 bauds. Only three pins of the microcontroller are engaged in asynchronous serial communication. Two of the I/O pins are engaged for serial communication with the external EEPROM. Thus, 16 I/O pins of the microcontroller can serve for 4 non cascaded shift registers, which in turn can drive 8 stepper motors. A multiple number of shift registers can be cascaded together if more number of motors is required to be driven. If required, any remaining pins may be used for feedback purposes. The microcontroller chip is smaller in size

which is best suited for compact control applications. PBASIC Compiler and Basic stamp command guide can be downloaded from Parallax website www.parallax.com

4K x 8 (32K bit) Serial EEPROM 24LC32A

It makes the circuit capable of working offline without the computer attached to the circuit. It is used here for storing sequence or motor movements which is required for any particular operation. The microcontroller program reads this downloaded data and sends to the steppers during runtime. Communication to this is established through I2C Communication protocol through microcontroller using I2COUT and I2CIN Command.

Figure 1: Prototype of the scheme tested on a Biped Robot.

Sl. No.	Components	Details
1	BS2px24	Basic Stamp Microcontroller
2	74HC595	Serial in Parallel Out Shift Register with Latch
3	M42SP-5	Mitsumi Stepper Motor
4	ULN2803	Darlington Array Stepper Motor Drivers
5	Switch	2 Way DIL Slide Switch
6	Resistor	1KΩ, 220Ω, 10KΩ × 6, 4.7KΩ × 2
7	7809, 7805	9V and 5V Regulator
8	24LC32A	4K x 8 (32K bit) Serial EEPROM 24LC32A
9	Capacitors	0.1µF × 2 Ceramic, 1000µF Ceramic
10	Push Button	Push to On Switch for Reset Button
11	Power Supply	12V, Ampere rating will depend on number of motors used (3A approx.)
12	Carrier Board	Parallax Basic Stamp Super Carrier Board
Items listed in 5, 6, 8 and 9 is not required if Parallax Basic Stamp Super Carrier Board is used with 12V power supply. Gearbox is optional		

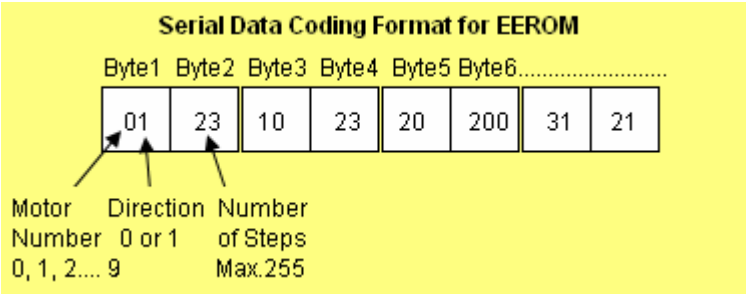

Figure 2: Serial Coding Format for EEPROM Storage

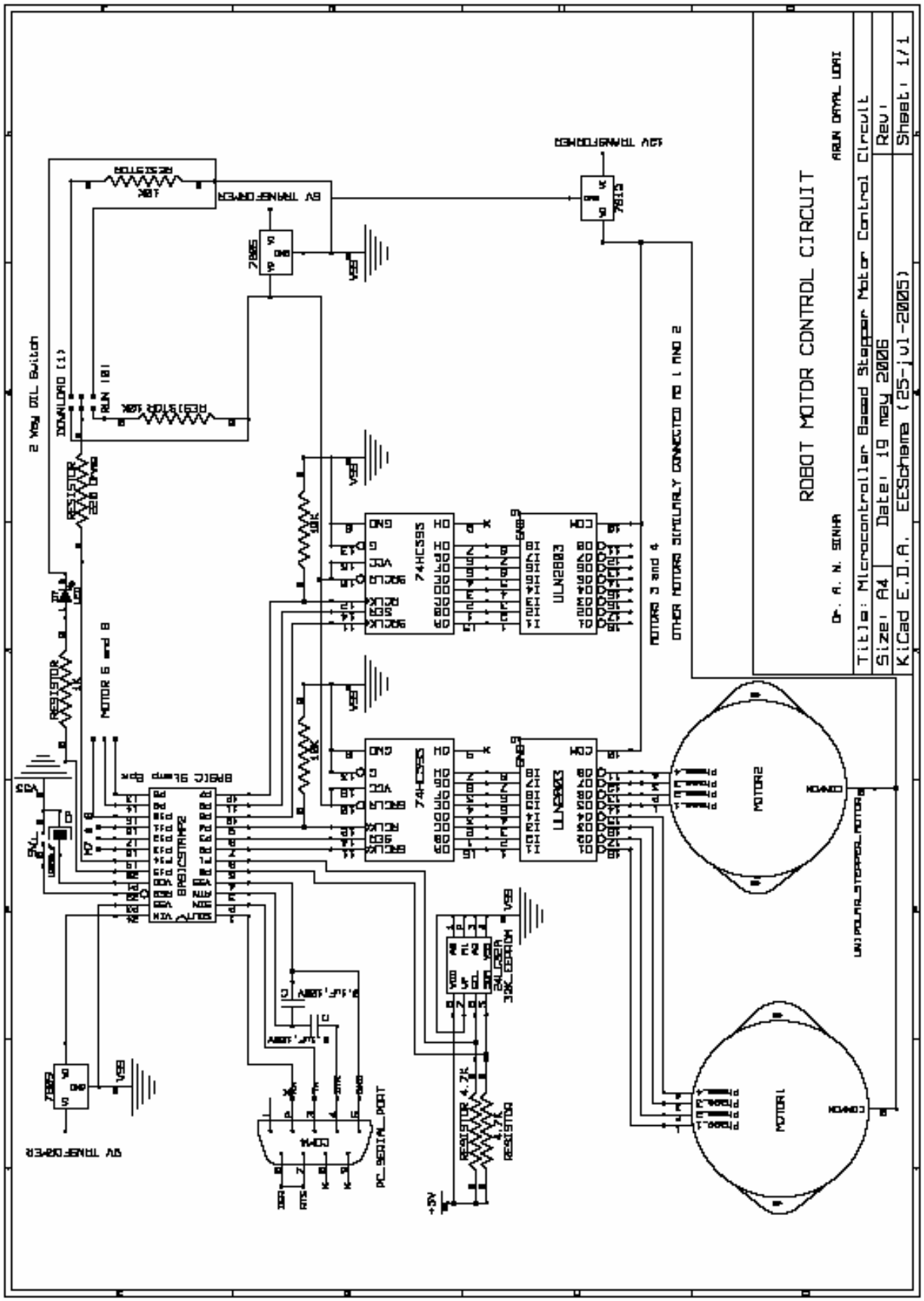

Figure 3: Stepper Motors Control Scheme

16 pin Serial-in parallel-out Shift Register with Latch 74HC595

This IC provides the 4 bit sequential pulse data required for driving each stepper motor. Having an 8 – bit output pins, it can drive two unipolar stepper motors at a time. Pin 11 receives the clock pulses and pin 14 receives the series data given by the microcontroller by a single SHIFTOUT command to any microcontroller pin. Pin 12 receives the Latch signal given by the single PULSEOUT command of the microcontroller that enables output pins to collect the finally prepared signal at the registers.

The IC is implemented to expand the output pins of the microcontroller. The chip uses only 3 data pins to drive 8 output pins when used in non cascaded form. Two 74HC595 IC's can be cascaded together to control 8 ×2 = 16 output pins, with same 3 input lines. Thus cascading is used to drive 4 motor with only 3 microcontroller pins.

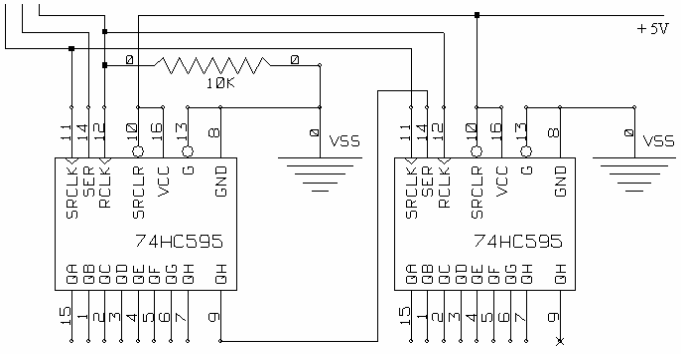

Figure 4: Cascading of two Shift Registers

Darlington Array Stepper Motor Drivers ULN2003/2803

The ULN2803A/ULN2003 contains eight/seven darlington transistors with common emitters and integral suppression diodes for inductive loads. Outputs may be paralleled for higher current capability for driving higher torque motors. It is used to convert the TTL logic 5V signal levels to 12V driving signals for driving motor actuators.

Unipolar Stepper Motor (MITSUMI M42SP-5)

Permanent Magnet Unipolar stepping motors (MITSUMI M42SP-5) is selected because of its compact size, low power consumption, high torque 26.5 mN·m/200pps and precise step angle of 7.5^o/step and a maximum of 365 pulses per second. The driving of unipolar motor is simpler as compared to bipolar motor which requires H-Bridge for its driving. Unipolar motor also supports precise half step or micro step driving. If a motion of step lesser than half step i.e. 3.75^o is required, additional reduction gear heads can be attached which will also increase the driving torque of the shaft. Voltage applied to the stepper motor remains constant, while the speed can be varied with the number of pulses per second applied through ULN2803. However, the torque of the motor varies inversely as per the speed which is shown in the graph below for MITSUMI M42SP-5 motor.

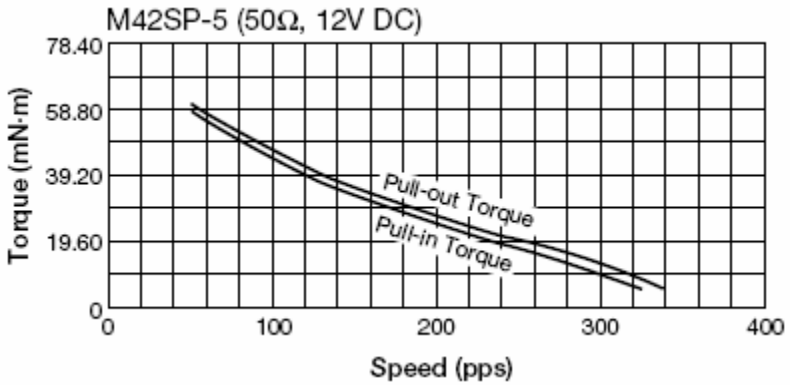

Figure 5: A Typical Speed – Torque characteristics of a stepper motor.

Based on the speed-torque characteristic of the stepper motor selected, maximum and minimum speed for the motion is to be decided. The variation of the pulse can be made easily as this is a digital process achieved through microcontroller. This is an added advantage over simple DC motor, where the torque characteristic is based on the voltage applied to its terminals that requires digital to analog converter for speed control.

SOFTWARE IMPLEMENTATION

A suitable Computer Program in any appropriate language can be used to send the serial data to the microcontroller. This data is coded in the format discussed before and sent to the microcontroller which receives it and stores it in the external EEPROM in download mode. As the PC COM Port can send the maximum numeric value of 255 i.e. FF in Hexadecimal or one byte, the maximum number of steps are limited to 255. A typical coding of motor data can be sent in two bytes of information. First byte consists of motor number and the direction, and second byte says the number of steps to turn in that direction. For example, 91 means motor number 9 to be rotated in clockwise direction. Similarly, the value 80 is to be decoded as motor number 8 and direction anticlockwise. The motor data is stored in EEPROM in groups of two bytes of information at two consecutive locations.

The microcontroller program consists of two parts. One is the download code which receives the motor data through PC and stores in its EEPROM. The second part of the microcontroller code reads the data from the EEPROM and sends the pulses to the required motor in decoded direction. There is a mode change switch implemented in the circuit PIN 14 of the microcontroller (refer Figure 3) which decides the download or working code to be functional based on the position of the switch. The indicator pin PIN 15 of the microcontroller fitted with a Light Emitting Diode (LED) is used to indicate the download completion while data transfer and cycle completion in running mode. In the running mode, if the switch position is changed to download mode, it acts to halt the motors.

If feedback is required to the computer, MAX232 chip can be used for communicating with Microcontroller. However, this requires two additional pins of the microcontroller I/O pins. It acts as a level shifter and converts the RS-232 level signals from our PC to the TTL levels required by the BASIC Stamp microcontroller.

If any gear is used it should have minimum backlash for positional accuracy. The gear box may contain clutch arrangement to enables easy resetting to the initial position.

PBASIC Microcontroller Code

```
'=====================================================================================
'   File......Serial IO.bpx
'   Purpose...Microcontroller Program for Controlling Stepper Motors
'   Author....Arun Dayal Udai
'   E-mail....arun_udai@yahoo.com
'
' {$STAMP BS2px}
' {$PBASIC 2.5}
'
' -------------------------------------------------------------------------------
' Program Description
' -------------------------------------------------------------------------------
' This program uses serial port to take input from the Computer, the motor number
' and the direction and sends the Phase patterns through serial communication to
' 74HC595 serial in parallal out shift register, which in turn send it to the
' ULN2803 Stepper motor drivers.
' -------------------------------------------------------------------------------
' I/O Definitions
' -------------------------------------------------------------------------------
SDA           CON    0                ' I2C serial data line
DataOut1       CON    2                 ' serial data out (1)(74HC595.14)
Latch1        CON    4                ' output latch (1)(74HC595.12)
```

```
Clock1          CON     6                ' shift clock (1)(74HC595.11)
DataOut2        CON     8                ' serial data out (2)(74HC595.14)
Latch2          CON     10               ' output latch (2)(74HC595.12)
Clock2          CON     12               ' shift clock (2)(74HC595.11)
DataOut3        CON     3                ' serial data out (3)(74HC595.14)
Latch3          CON     5                ' output latch (3)(74HC595.12)
Clock3          CON     7                ' shift clock (3)(74HC595.11)
DataOut4        CON     9                ' serial data out (4)(74HC595.14)
Latch4          CON     11               ' output latch (4)(74HC595.12)
Clock4          CON     13               ' shift clock (4)(74HC595.11)
ModePin         CON     14               ' Mode Change Pin
LEDPin          CON     15               ' Download Status Indicator Pin
' -----------------------------------------------------------------------------
' Constants
' -----------------------------------------------------------------------------
DevType         CON     %1010 << 4       ' device type
DevAddr         CON     %000 << 1        ' address = %000 -> %111
Rd2432          CON     DevType | DevAddr | 1   ' read from 24C32A
Wr2432          CON     DevType | DevAddr | 0   ' write to 24C32A
' -----------------------------------------------------------------------------
' Variables
' -----------------------------------------------------------------------------
sAddr     VAR     Byte
sAddr0    VAR     Byte
sAddr1    VAR     Byte
sAddr2    VAR     Byte
sAddr3    VAR     Byte
sAddr4    VAR     Byte
sAddr5    VAR     Byte
sAddr6    VAR     Byte
sAddr7    VAR     Byte
Phase     VAR     Nib
Phase0    VAR     Nib
Phase1    VAR     Nib
Phase2    VAR     Nib
Phase3    VAR     Nib
Phase4    VAR     Nib
Phase5    VAR     Nib
Phase6    VAR     Nib
Phase7    VAR     Nib
Motor_num VAR     Nib
Directn   VAR     Bit
eeAddr    VAR     Word          ' address: 0 - 4095
inVal1    VAR     Byte          ' input from EEPROM, MotorNum and Direction,
                                ' outVal in case of data Download mode
inVal2    VAR     Byte          ' input from EEPROM, Number of Steps
                                ' inVal in case of data download mode
counter   VAR     Byte
' -----------------------------------------------------------------------------
' EEPROM Data for half stepping of unipolar stepper motor
' -----------------------------------------------------------------------------
Steps  DATA  %1000, %1100, %0100, %0110, %0010, %0011, %0001, %1001
' -----------------------------------------------------------------------------
' Initialization
' -----------------------------------------------------------------------------
LOW   Latch1
LOW   Latch2
```

```
LOW   Latch3
LOW   Latch4
INPUT ModePin
' -----------------------------------------------------------------------------
' Program Code
' -----------------------------------------------------------------------------
IF (IN14 = 0)THEN
  GOTO Main
ELSE
  GOTO Download_Data
ENDIF

Main:
eeAddr = 0
DO
  I2CIN SDA, Rd2432, eeAddr.HIGHBYTE\eeAddr.LOWBYTE, [inVal1]
  eeAddr = eeAddr + 1
  I2CIN SDA, Rd2432, eeAddr.HIGHBYTE\eeAddr.LOWBYTE, [inVal2]
  Motor_num = (inVal1/10)
  FOR counter = 1 TO inVal2
    PAUSE 20
    ON Motor_num GOSUB Case_0, Case_1, Case_2, Case_3, Case_4, Case_5, Case_6, Case_7
  NEXT
  eeAddr = eeAddr + 1
  IF eeAddr > 447 THEN                ' 447 is the length of the serial_IO_data array.
    eeAddr = 0
    HIGH LEDPin
    PAUSE 500
    LOW LEDPin
  ENDIF
  halt:
  IF (IN14 = 1) THEN GOTO halt
LOOP

Download_Data:
  FOR eeAddr = 0 TO 447                                 ' Array Length
    SERIN 16, 16572, [WAIT("M"), DEC inVal1]
    I2COUT SDA, Wr2432, eeAddr.HIGHBYTE\eeAddr.LOWBYTE, [inVal1]       ' outVal
    PAUSE 10
    I2CIN SDA, Rd2432, eeAddr.HIGHBYTE\eeAddr.LOWBYTE, [inVal2]        ' inVal
    IF (inVal2 <> inVal1) THEN
      LOW LEDPin
      PAUSE 500
    ELSE
      HIGH LEDPin
    ENDIF
  NEXT
  LOW LEDPin
  END
' -----------------------------------------------------------------------------
' Subroutines
' -----------------------------------------------------------------------------
Case_0:
Directn = inVal1
sAddr = sAddr0
ON Directn GOSUB Step_Fwd, Step_Rev
sAddr0 = sAddr
```

```
Phase0 = Phase
SHIFTOUT DataOut1, Clock1, MSBFIRST, [Phase0\4, Phase1\4]
PULSOUT Latch1, 1
RETURN

Case_1:
Directn = inVal1 - 10
sAddr = sAddr1
ON Directn GOSUB Step_Rev, Step_Fwd
sAddr1 = sAddr
Phase1 = Phase
SHIFTOUT DataOut1, Clock1, MSBFIRST, [Phase0\4, Phase1\4]
PULSOUT Latch1, 1
RETURN

Case_2:
Directn = inVal1 - 20
sAddr = sAddr2
ON Directn GOSUB Step_Rev, Step_Fwd
sAddr2 = sAddr
Phase2 = Phase
SHIFTOUT DataOut2, Clock2, MSBFIRST, [Phase2\4, Phase3\4]
PULSOUT Latch2, 1
RETURN

Case_3:
Directn = inVal1 - 30
sAddr = sAddr3
ON Directn GOSUB Step_Fwd, Step_Rev
sAddr3 = sAddr
Phase3 = Phase
SHIFTOUT DataOut2, Clock2, MSBFIRST, [Phase2\4, Phase3\4]
PULSOUT Latch2, 1
RETURN

Case_4:
Directn = inVal1 - 40
sAddr = sAddr4
ON Directn GOSUB Step_Fwd, Step_Rev
sAddr4 = sAddr
Phase4 = Phase
SHIFTOUT DataOut3, Clock3, MSBFIRST, [Phase4\4, Phase5\4]
PULSOUT Latch3, 1
RETURN

Case_5:
Directn = inVal1 - 50
sAddr = sAddr5
ON Directn GOSUB Step_Fwd, Step_Rev
sAddr5 = sAddr
Phase5 = Phase
SHIFTOUT DataOut3, Clock3, MSBFIRST, [Phase4\4, Phase5\4]
PULSOUT Latch3, 1
RETURN

Case_6:
Directn = inVal1 - 60
```

```
sAddr = sAddr6
ON Directn GOSUB Step_Rev, Step_Fwd
sAddr6 = sAddr
Phase6 = Phase
SHIFTOUT DataOut4, Clock4, MSBFIRST, [Phase6\4, Phase7\4]
PULSOUT Latch4, 1
RETURN

Case_7:
Directn = inVal1 - 70
sAddr = sAddr7
ON Directn GOSUB Step_Rev, Step_Fwd
sAddr7 = sAddr
Phase7 = Phase
SHIFTOUT DataOut4, Clock4, MSBFIRST, [Phase6\4, Phase7\4]
PULSOUT Latch4, 1
RETURN

Step_Fwd:
sAddr = sAddr + 1 // 8
READ (Steps + sAddr), Phase
RETURN

Step_Rev:
sAddr = sAddr + 7 // 8
READ (Steps + sAddr), Phase
RETURN
```

Sample pseudo code to communicate with Microcontroller through serial port

```
function serialIO(serial_IO_data)
        s = serial('COM1');
        set(s, 'BaudRate', 19200);
        fopen(s);
        for i = 1: length(serial_IO_data)
           fprintf(s, 'M%d ', serial_IO_data(i));        //M denotes the byte seperator
           pause(0.1);
        end
        fclose(s);
        delete(s)
        clear s;
```

serial_IO_data is an array having format [10 34 20 22 31 23 41 32 50 89……] in the format explained above as first byte denoting motor number and direction and second byte as number of steps in that direction.

CIRCUIT OPERATION

In the stepper control system Figure 3, put the switch in download position (refer figure 3), press reset button and run the Serial IO program to send the data to the EEPROM. Put the switch in run mode and press reset button to execute the downloaded data to reproduce motion cyclically. Change the array length to the actual array length of the Serial_IO_data in the microcontroller program. Maximum length can be 4095 bytes, i.e. 32K for 32LC32A EEPROM. This may be replaced by 32LC256A or higher compatible IC's.