

Lift Controller

by *Shahriar Nirjon* and his friends

1. Introduction:

A Lift Controller is a computer automated control that operates the motor and other mechanical aspects of a lift. Its minimal responsibilities include: reading user requests through switches, driving a motor to move the lift at appropriate floors and a control hardware for opening and closing the lift door. We completed this project as a requirement in the coursework of CSE 408N, Computer Interfacing Sessional. Out of several options, “A Lift Controller” seemed to have a uniform importance in several intricate aspects, e.g., Implementation of an algorithm, taking inputs correctly and timely, processing the inputs by feeding them into the algorithm, and dispatching necessary commands through the outputs. Several other features imposed by the coursework (described in the following section: **Problem Description**) made the project a bit more challenging. Above all, it was a fairly attractive project to us considering the limited time available for implementation and also the availability of resources in the local market.

2. Problem Description:

The core requirement of this project was to implement and control a hardware circuit that can control a physical lift. The controlling of this circuit was supposed to be accomplished by interfacing through a PC parallel port. Inputs were to be processed by applying the standard Elevator Algorithm. The lift was moved by a stepper motor. Outputs to control the stepper motor were also issued by the parallel port.

Switches were supposed to have toggling capability, i.e., in case of a mistake made by the user, the command could be taken back. The processing logic of the lift algorithm was as follows:

1. For a request from within the lift to open the door, the lift would stop and serve the request. The same would happen if someone presses to go in the same direction as the lift's current direction and the lift is also in that floor.
2. For a request from a nearest floor in the current direction to go farther in the same direction, the request was to be served.
3. If no request like (2) could be found, then a request from the switches within the lift to go in the current direction was served.
4. If no request like (3) could be found, then a request from the farthest floor in the current direction to go in the opposite direction would be served.
5. If no request like (4) could be found, then the current direction of movement would be reversed.

6. The processing logic would advance differentially, i.e, each command to the motor would move it exactly one floor up or down. Then the whole state of the inputs would again be scanned at that floor to make a new decision.

There was another important configuration in this project. The lift could be configured to move between two specified floors. This setup was made from the software. An operator could also operate the lift by issuing up/down commands from the software.

Also, the lift had to be robust in case of power failures. When the power returned, the computer program was supposed to bring the lift safely to the ground floor. This was to be performed through some form of sensors, as we had used metal contacts. The same course of events would take place in case of an ALARM signal from within the lift. Finally, the lift was to be precise in stopping at appropriate floor levels.

3. Details Design

3.1 Block Diagram

The block diagram of the lift controller is given below. The diagram shows that User sends request to the CPU via input switches. According to user request CPU drives the LED output and controls the motor. LEDs indicate current lift position and the motor physically moves the lift. When user request is served the input switches are reset. Both the controllers are driven by power supply.

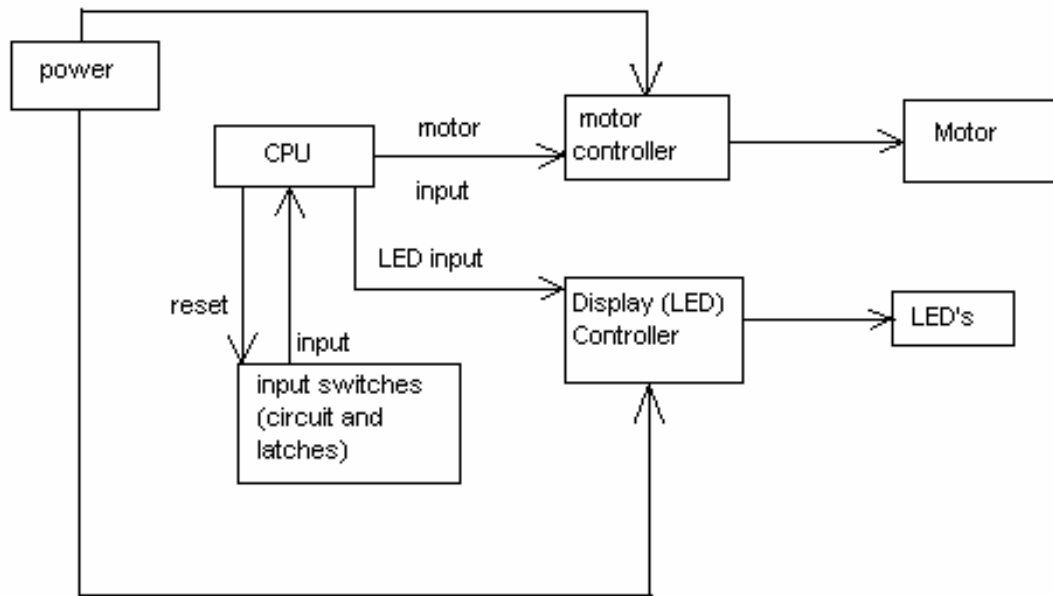


Figure: Block diagram of lift controller

3.2 Details Diagram

The details diagram for different components of the lift controller are described in the following sections.

- Input Operation
- Output Operation
- Full Diagram

3.2.1 Input Operation

3.2.1.1 Switch Operation

Inputs to the lift controller are taken from switches. For a building 4 floors there are 6 switches outside the lift and 6 switches inside the lift. Switch operations are:

- Each switch is a push button
- T Flip-flop is used to hold current state. So, in case of mistake we toggle the switch.
- Led output for status of each switch
- Resistors to control current
- Reset to clear request

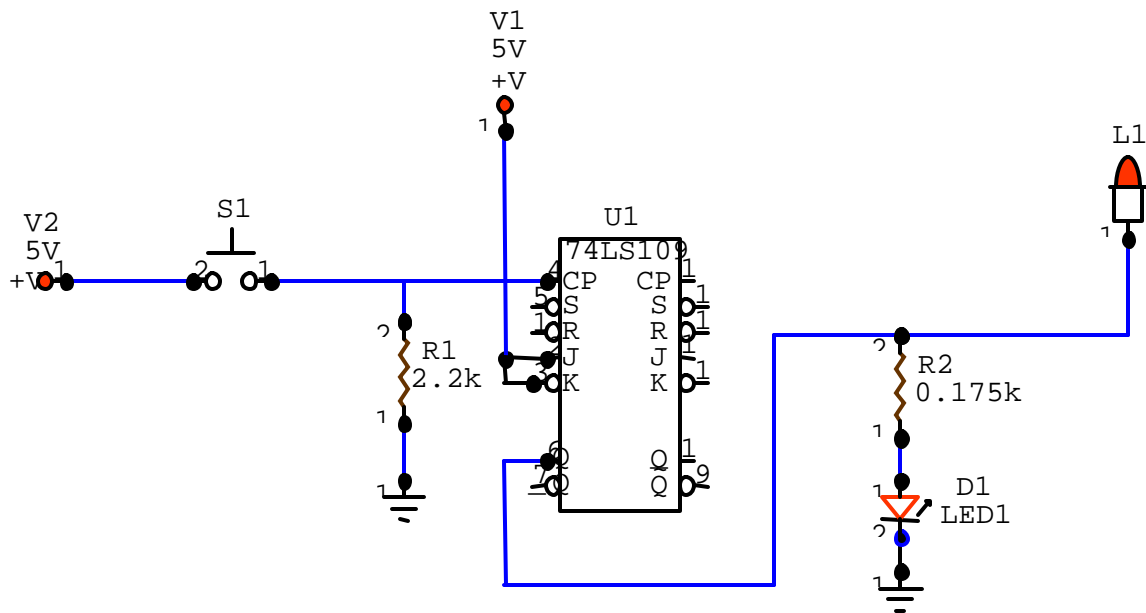


Figure : Switch Implementation

3.2.1.2 Input Circuit Specification

There are 16 inputs to be taken which are 12 switches, lift position (2 bits), lift direction and initial ground floor sensor. We use the following parallel port pins for input:

- 4 Input Pins
 - ☐ 10 (Ack)
 - ☐ 11 (Busy)
 - ☐ 12 (Paper out)
 - ☐ 13 (Select)

Since there are 4 pins available we read 4 times to get 16 inputs. We use MUX for this selection. The selection for the mux comes from the output pins 1(D₀) and 2(D₁).

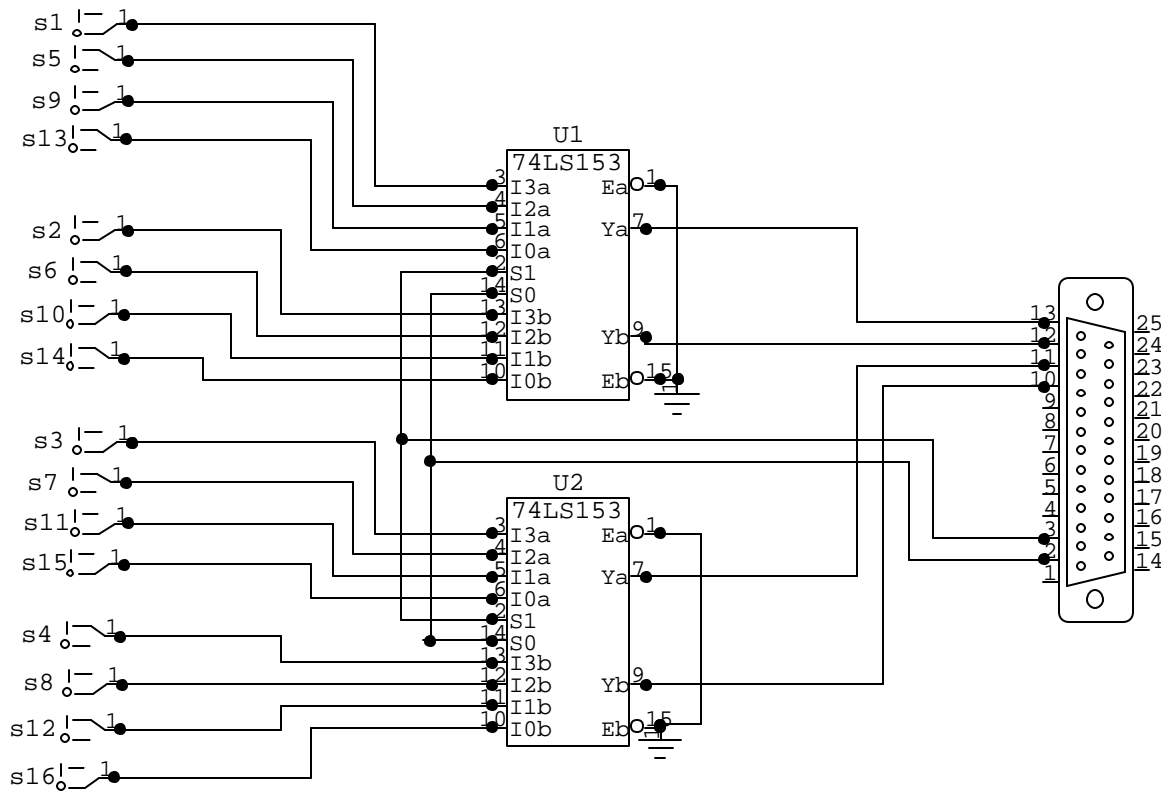


Figure: Input Circuit Specification

3.2.2 Output Operation

The following outputs are to be given from the CPU:

1. Control Lift
 - Stepper motor to control motion of lift
2. Position, Status and Direction of the lift
 - Led Output
3. Feedback to reset switch

There are 8 output pins 2-9 (D_0 - D_7). D_0 , D_1 are used for Mux selection. So, pins D_4 - D_7 are used for output. We need

- Latch to multiplex output
- Decoder to decode output
- Universal Darlington Pair Array

- To control stepper motor

3.2.2.1 Position and Direction of the lift

Position of the lift are encoded to 2 bits since there are 4 floors and for door open/close and direction 2 extra bits are needed to output. So, 4 output pins are used for this purpose. A decoder is used to decode the lift position and reset logic for the push buttons are shown in figure. For simplicity we only show the reset logic for 3 switches.

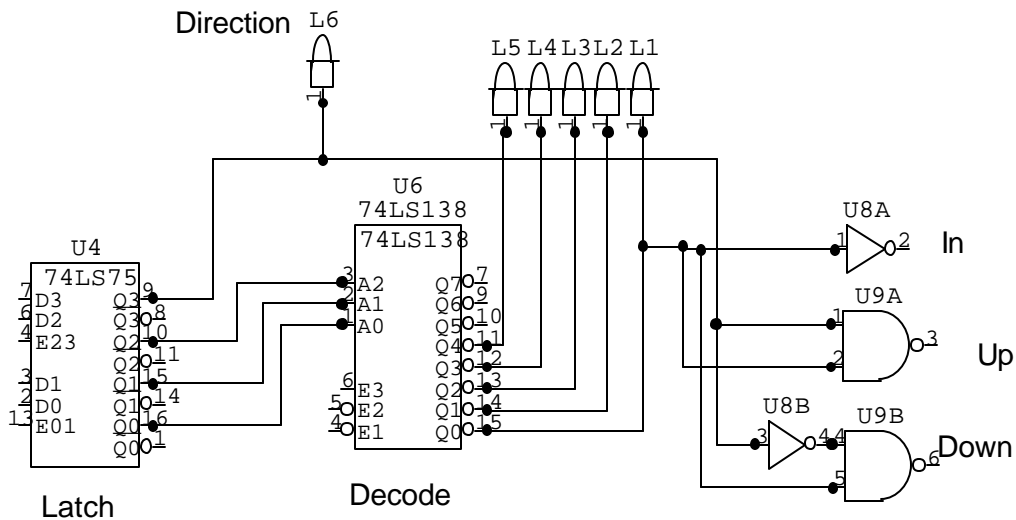


Figure: Position and Direction of the lift

3.2.2.2 Motor Control

Universal Darlington Pair array (ULN 2003) is used to drive the motor. It is interfaced to output by a latch and the outputs of ULN2003 drives motor. A zener diode is used to safeguard the circuit against any inductive kick during power off of the motor.

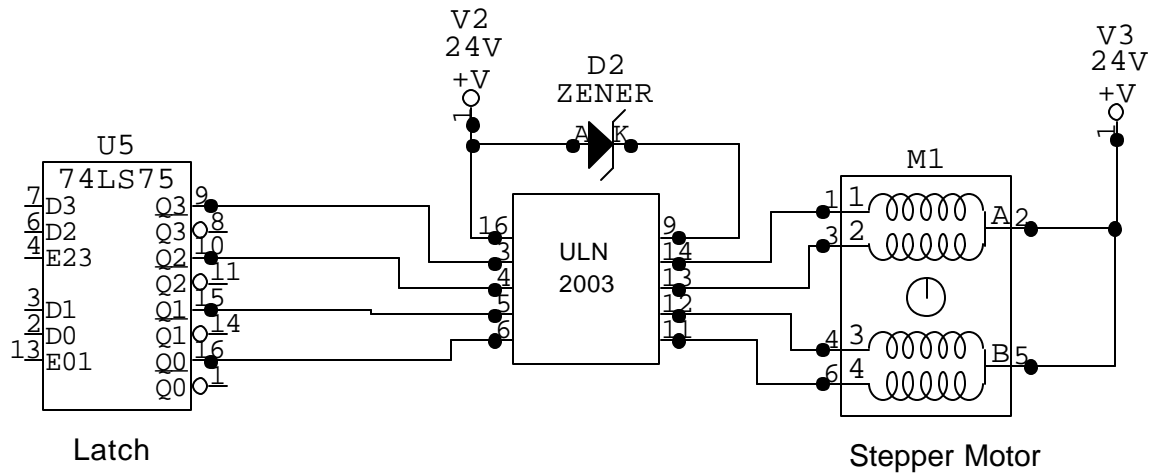


Figure: Motor Control

3.2.2.3 Output Circuit

The output circuit consists of two sections: one is led output and the other is stepper motor controlling. So, to multiplex output latches are used for each component. The extra 2 pins are used for the enable input of the latches. The latch enables are controlled in such a way that appropriate latch is enabled at the appropriate time.

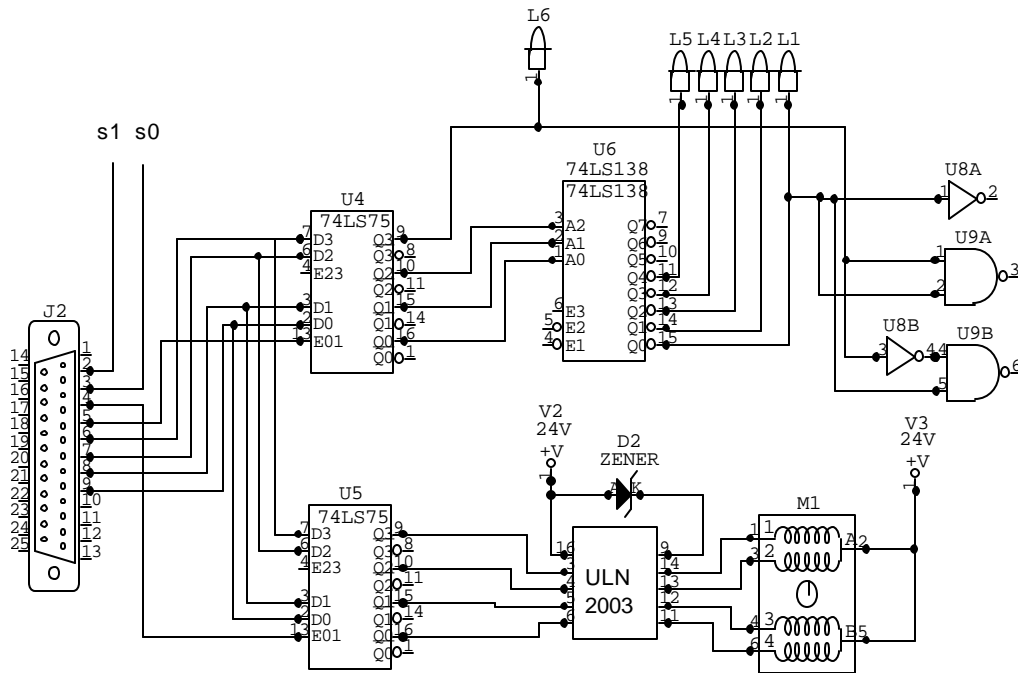


Figure: Output Circuit

4. Hardware Specification

The details for the hardware used are given below. The IC numbers item name and the pieces required are given in the table.

Item Name	IC Number	Pieces
Push Button Switch	N/A	12
Dual J-K Flip Flop	74LS109	6
Resistor (2.2 k, 0.175 k)	N/A	24
LED	N/A	12 + 4 + 1
Dual 4 X 1 Mux	74LS153	2
Latch	7475	2
3 X 8 Decoder	74138	1
Universal Stepper Motor Controller	ULN 2003	1
Zener Diode	N/A	1
24 volt Stepper Motor	N/A	1

5. Software Description

In this section we will look closely at the software portion. At first we describe the overall software portion part by part with sample codes then the full commented code is presented.

5.1 The State

- We captured the current condition of the lift into a C++ structure that we named NODE. It looks like this:

```
struct NODE{
    int inLift[4]; /*condition of switches inside the lift*/
    int up[4]; /*condition of up switches outside the lift*/
    int down[4]; /*condition of down switches outside the lift*/
    int doorClose; /*used for ground sensor purpose*/
    int doorOpen; /*condition of door open switch inside lift*/
    int alarm; /*condition of alarm switch inside lift*/
    int floor; /*current floor*/
    int dir; /*current direction*/
    void clear(); /*zeros all the variables*/
    int isSame(NODE n); /*check if 2 NODE are identical*/
    void set(int arr[4][4]); /*set NODE using arr[4][4]*/
};
```

5.2 Taking Input

5.2.1 The Basics of taking Input

- Input is taken from LPT1 port register 0x379. In windows 98, input taking is done by using the following library function of turbo C:

```
unsigned char x;
x = inportb(0x379);
```

- Port pins are mapped with the 0x379 register as follows:

0x379:

Pin 11	Pin 10	Pin 12	Pin 13	X	X	X	X
--------	--------	--------	--------	---	---	---	---

Remember, Pin 11 gives inverted input.

5.2.2 Taking all 16 Inputs

- Using only 4 pins we have taken a total of 16 inputs. We've done it by reading 4 times the 0x379 register. Before reading the register we supply the 2 selection bits of the MUXs using output port.
- Using our readPort() function we read out 4 specific input at a time, like this:

```

unsigned char outputPort;
/*reads 4 specific inputs into arr[4], variable 'sel' is used to output MUX selection
bits*/
void readPort(unsigned char sel, int arr[4]){
    unsigned char x;
    outputPort &= 0xfc; /*clearing 2 least significant bits*/
    outputPort |= sel; /*sel = [0,3]*/
    outportb(0x378, outputPort);
    delay(10);
    x = inportb(0x379);
    arr[0] = (x & 0x40)?1:0;
    arr[1] = (x & 0x80)?0:1;
    arr[2] = (x & 0x20)?1:0;
    arr[3] = (x & 0x10)?1:0;
    return ;
}

```

- We have saved all 16 inputs into int arr[4][4] which holds information like this:

Up0	Door Open	InLift0	Alarm
Up1	Down1	InLift1	S1 (mux)
Up2	Down2	InLift2	S0 (mux)
Door Close	Down3	InLift3	Direction

- Our readState() function calls readPort() 4 times and reads all 16 inputs into arr[4][4]:

```

NODE readState(){
    NODE r;
    int arr[4][SZ];

```

```

        unsigned char i;

        for(i=0;i<4;i++){
            readPort(i, arr[i]);
        }
        r.clear();
        r.set(arr);
        return r;
    }

```

5.3 Controlling the Stepper Motor

5.3.1 Rotating Stepper Motor

- Using ULN-2003 it's easy to rotate a stepper motor. After connecting the motor to the ULN-2003 all we have to do is to excite each of the 4 pins of it sequentially.

5.3.2 Our Implementation

- The command byte we write to register 0x378:

Motor rotate command byte:

Sq3	Sq2	Sq1	Sq0	0	MLE	0	0
-----	-----	-----	-----	---	-----	---	---

MLE : Motor Latch Enable

Sq(0-3): Motor Sequence

- Our movelift(int dir) function rotates the motor clockwise or anti-clockwise according to direction parameter:

```

void moveLift(int dir){
    int i,j;
    unsigned char ff[12]={0x10,0x14,0x10,0x20, 0x24,0x20,0x40,0x44,
        0x40,0x80,0x84,0x80};

```

```

        if(dir==1){
            /*we need 28 steps to move one floor*/
            for(i=0;i<28;i++){
                for(j=0;j<12;j++){
                    outportb(0x378, ff[j]);
                    delay(15);
                }
            }
        }
        else{
            for(i=0;i<28;i++){
                for(j=11;j>=0;j--){
                    outportb(0x378, ff[j]);
                    delay(15);
                }
            }
        }
    }
}

```

5.4 Resetting Switches and LED outputs

- The command byte we write to register 0x378:

command byte:

Fl1	Fl0	Door	Dir	OLE	0	0	0
-----	-----	------	-----	-----	---	---	---

Fl(1-0): Current Floor

Door: Showing Door Condition (1 = Open)

Dir: Showing Lift Direction (1=up)

OLE: Latch Enable

- Our outputLiftPos() functions as follows:

/*x = current floor, dr = lift direction, door = door condition*/

```
void outputLiftPos(int x, unsigned int dr,unsigned int door){
```

```
    unsigned char tmp = (unsigned char)x;
```

```
    outputPort = 0;
```

```

        outputPort = tmp<<6;
        outputPort |= (dr<<4);
        outputPort |= (door<<5);
        outportb(0x378, outputPort);
        delay(30);
        outputPort |= (1<<3);
        outportb(0x378, outputPort);
        delay(30);
        outputPort = 0;
        outportb(0x378, outputPort);
    }

```

5.5 The Logic Behind

5.5.1 Door Simulation

- We simply show the door open/close simulation, using following code segment:

```

outputLiftPos(curState.floor, curState.dir, 1);
printf("\nOpening Door ...\n");
delay(DOOR_DELAY);
printf("Closing Door ... \n");
outputLiftPos(curState.floor, curState.dir, 0);

```

5.5.2 State Transition

- When Lift is moving up: The Lift will move one step up further if any of the following conditions hold: (maxf = maximum floor)
 - curState.up[i] == 1, for i = curState.floor+1 to maxf-2
 - curState.inLift[i] ==1, for i = curState.floor+1 to maxf-1
 - curState.down[i] ==1, for i = maxf to curState.floor+1
- When Lift is going down: The Lift will go one step down further if any of the following conditions hold: (minf = lowest floor)
 - curState.down[i] == 1, for i = curState.floor-1 to minf+1;
 - curState.inLift[i] == 1, for i = curState.floor-1 to minf;
 - curState.up[i] == 1, for i=minf to curState.floor-1

5.5.3 Initial Pull Down

- When the software starts running, the lift is pulled down until it reaches ground floor. We have a sensor input (namely doorClose) that helps us to determine when to stop pulling.
- Our init() function that is invoked at the beginning of the software:

```
void init(){
    do{
        curState = readState();
        delay(10);
        st1 = readState();
        if(st1.doorClose==1)
            break;
        moveDowninit();
    }while(1);
    curState.clear();
    fd=fu=0;
    outputLiftPos(0,1,0);
}
```

5.6 Manual mode operation

- While the software is running we can enter the manual operation mode (we call it mode2). In this mode, all the inputs are inhibited except ground sensing input. Lift can now be operated by software commands.
- The options available in this mode are:
 - Move Lift Up
 - Move Lift Down
 - Go back to previous mode
 - Pull Lift down to ground
 - Set maximum floor limit
 - Set minimum floor limit
- By setting up maximum & minimum limits we can customize our lift operation such that the lift works within the specified range of floor (minf to maxf). Requests that may take the lift out of the specified range are simply ignored.

6. User Manual:

The lift is very much amenable to use. Anyone who wants to have a service must press the appropriate switch. For example if anyone wants to go down from 2nd floor he must press the down button placed at 2nd floor. Once a button is pressed the associated LED will glow, which is an indication that the button press has been acknowledged. Any wrong switch press can be corrected by repressing that switch. That is once you pressed a

wrong switch, you can correct it at once by repressing that switch. Associated LED will be put off to denote that the switch is disabled. Once you have pressed the correct switch the lift will come to your floor as early as possible and gate will be opened for 5 seconds. After 5 seconds the gate will be closed and the gate open switch should be pressed to reopen the gate. After getting inside there will be a number of switches each of which denotes specific floor. To get down or up to a desired floor the switch with desired number must be pressed, which will cause the lift to go to that floor and open the gate. The gate will be kept open for 5 seconds and to reopen the gate you must press the open gate switch. Once the gate is opened in your desired floor you can easily get off and go to your business.

Besides the manual use of the lift, there are some administrative uses of our lift. The lift can be controlled fully by software. To control the lift by software you must have access to the location where the lift control program is running. Pressing a 'x' on keyboard while lift control program running will move you to administrative mode where you will be given the following prompt:

Enter choice:

1. up
2. down
3. exit
4. alarm
5. set min value
6. set max value

Next operations are clear from the prompt. Pressing 1 will cause the lift to go to one floor up than current floor. Pressing 2 will cause the lift to move down by one floor. Here any error such as pressing 1 when lift is at the topmost floor or pressing 2 when lift is at the down most floor may cause disastrous effect. Exit or alarm will cause the lift to go to ground floor and start the lift at normal mode. The most interesting part of the administrative works is 5 and 6. By pressing 5, the minimum value below which the lift won't move and by pressing 6 the maximum value above which the lift won't ride. So if the range is configured to $[m, n]$ then the lift would not go below m or go above n where m is the minimum value n is the maximum value and $n \geq m$. this range will be valid even if lift is entered into normal mode.

7. Problems faced and adopted solutions

1. Our input circuitry contains about 16 switches. But the parallel port has only 4 status pins that can be used to enter switch status into the PC. This was one of the

major hurdles that we faced at the initial phase of our project. The solution that we have adopted is to use MUX's for selecting between different inputs. There are four MUX's connected to the four status pins. As a result a total of 16 switch inputs can be read in four iterations via the MUX's by varying the selection inputs. If our system is expanded and requires more switch input's, then another level of MUX's can be used to incorporate the additional switches.

2. Another problem that we faced during our project is that one of the status pins gave inverted output. We discovered this trivial inconsistency while we were testing the status pin outputs via the `inportb()` function. This problem was solved by inverting the bit value of that particular status pin after the status pins had been read into the PC through the parallel port.
3. The push button switches that we used in our input circuitry were not of superior quality. As a result we had to take the possibility of switch debouncing into consideration. Debouncing was handled via software by reading the switch status continuously until the switch input values became consistent.
4. The stepper motor that we used did not have any well defined documentation on the correct sequence of pulses required to drive the motor in both directions. We had to try various motor connections in order to discover the correct connections required to rotate the motor in clockwise and anticlockwise directions.
5. One of the major hurdles that we faced while physically developing a lift structure was that of converting the rotational motion of the stepper motor to the translation motion of the lift. It is not possible to find out the exact amount of rotational motion required to move the lift from one floor to another floor without rigorous mathematical analysis. Since we did not have any exact mathematical formulation at our hand, we applied trial and error in order to derive the approximate amount of rotational motion that is proportional to the desired translation motion.
6. The stepper motor that we used in the first place was not heavy enough to drive the wooden frame structure that we had built to simulate lift motion. As a result we had to find a heavier stepper motor that was capable of driving the lift structure.

8. Further direction:

The chance of improving anything never ends. There are some good choices to improve our lift. They are discussed below,

There is no door control logic in our program. So, a door control logic will be spendid. To add a door control logic in our control program one can use the door open indicator to control the motor related to door and close it after some time. The hardware should be added also.

Whenever there is a door of a lift, there must be a sensor so that door does not closes while anything is passing through it. To add the sensors one can use the IR LED-sensor pairs which can be placed in two parts of the door and control the door so that it does not get closed until the sensor-LED line is blocked by something.

A lift should always be able to sense the load on it and give proper signals to warn people in case of overloads. A load cell can be employed for this.

The lift can be detected in each floor. Currently in our program the only one indicator for the ground floor is used. So, the lift goes to ground floor whenever the program is restarted or after some problem. If detector is added in each floor then the amount of movement can be reduced by moving the lift to the nearest floor.

Some voice annotation may also be added to lift for the relief of the users from determining the current floor from LED display. Whenever the lift reaches to a destination there may be a voice notification saying that “The lift is in 5th floor”. That will enable the user to get off even if he is unconscious or blind.

9. Conclusion

Lift controller embeds within it several other project scopes as we mentioned in the section *Future Direction*. If implemented properly and with some of those improvements, we can use such automated lift controllers in many important government and business premises for efficient, safe and sound performance. Lesser installation cost and better performance can make it a good choice for our country.

Appendix

1. Source Code

```
# include <stdio.h>
# include <string.h>
# define DELAY 1000
# define SZ 10
# define DOOR_DELAY 5000

# include <stdlib.h>
# include <dos.h>
# include <conio.h>

unsigned char outputPort; //0=s0,1=s1;
unsigned char inputPort;

int fd,fu;                /*flags to record current floor buttons*/
int maxf, minf;

struct NODE{
    int inLift[SZ];
    int up[SZ];
    int down[SZ];
    int doorClose;
    int doorOpen;
    int alarm;
    int floor;
    int dir;
    void clear();
    int isSame(NODE n);
    void set(int arr[][SZ]);
};

void NODE::clear(){

    memset(inLift, 0, sizeof(inLift));
    memset(up, 0, sizeof(up));
    memset(down, 0, sizeof(down));
    doorClose = doorOpen = 0;
}
int NODE::isSame(NODE n){
    int i;
    int fg=1;
    for(i=0;i<SZ;i++){
        if(inLift[i] != n.inLift[i]){
            printf("inLift %d (changed)\n", i);
```

```

        fg = 0;
    }
    if(up[i] != n.up[i]){
        printf("up %d (changed)\n", i);
        fg=0;
    }
    if(down[i] != n.down[i]) {
        printf("down %d (changed)\n", i);
        fg=0;
    }
}
if(doorOpen != n.doorOpen){
    printf("doorOpen (changed)\n");
    fg= 0;
}
if(doorClose != n.doorClose){
    printf("doorClose (changed)\n");
    fg= 0;
}
if(alarm != n.alarm){
    printf("alarm (changed)\n");
    fg= 0;
}
return fg;
}

```

```

void NODE::set(int arr[][SZ]){
    int i;
    for(i=0;i<3;i++){
        up[i] = arr[i][0];
    }
    doorClose = arr[3][0];
    doorOpen = arr[0][1];
    for(i=1;i<4;i++){
        down[i] = arr[i][1];
    }
    for(i=0;i<4;i++){
        inLift[i] = arr[i][2];
    }
    alarm = arr[0][3];
    floor = arr[1][3]*2 + arr[2][3];
    dir = arr[3][3];
}

```

```
NODE st1, st2, curState;
```

```
void show(NODE r)
```

```
{
    int i,j;
    char z[32][32];
    for(i=0;i<32;i++)for(j=0;j<32;j++)z[i][j]=' ';
    for(i=0;i<4;i++){
        z[i][0] = r.up[3-i]+'0';
    }

    for(i=0;i<4;i++){
        z[i][6] = r.down[3-i]+'0';
    }

    for(i=0;i<4;i++){
        z[i][12] = r.inLift[3-i]+'0';
    }
    printf("Up   Down  Inside\n");
    for(i=0;i<4;i++){
        for(j=0;j<15;j++){
            putchar(z[i][j]);
        }
        printf("\n");
    }
    printf("\n");
    printf("Range: [%d,%d]\n", minf+1, maxf+1);
    printf("Floor: %d\n", r.floor+1);
    printf("Dir  : %s\n", r.dir==1?"UP":"DOWN");
    printf("Alarm: %s\n", r.alarm==1?"ON": "OFF");
    printf("\n");
}
```

```
void moveLift(int dd)
```

```
{
    int i,j;
    unsigned char ff[12]={0x10,0x14,0x10,0x20, 0x24,0x20,0x40,0x44,
        0x40,0x80,0x84,0x80};

    if(dd==1){
        for(i=0;i<29;i++){
            for(j=0;j<12;j++){

                outportb(0x378, ff[j]);
                delay(10);
            }
        }
    }
}
```

```

        }
    }
    else{
        for(i=0;i<28;i++){
            for(j=11;j>=0;j--){
                outportb(0x378, ff[j]);
                delay(10);
            }
        }
    }
}

```

```

void moveDowninit()
{
    int i,j;
    unsigned char ff[12]={0x10,0x14,0x10,0x20, 0x24,0x20,0x40,0x44,
        0x40,0x80,0x84,0x80};

        for(j=11;j>=0;j--){
            outportb(0x378, ff[j]);
            delay(10);
        }
}

```

```

void readPort(unsigned char sel, int arr[SZ])
{
    unsigned char x;

    outputPort &= 0xfc;
    outputPort |= sel;

    outportb(0x378, outputPort);
    delay(10);
    x = inportb(0x379);

    arr[0] = (x & 0x40)?1:0;
    arr[1] = (x & 0x80)?0:1;
    arr[2] = (x & 0x20)?1:0;
    arr[3] = (x & 0x10)?1:0;

}

```

```

NODE readState()
{
    NODE r;
    int arr[4][SZ];
    unsigned char i;

    for(i=0;i<4;i++){
        readPort(i, arr[i]);
    }

    r.clear();
    r.set(arr);

    return r;
}

void outputLiftPos(int x, unsigned int dr,unsigned int door)
{
    unsigned char tmp = (unsigned char)x;
    outputPort = 0;

    outputPort = tmp<<6;
    outputPort |= (dr<<4);
    outputPort |= (door<<5);
    outportb(0x378, outputPort);
    delay(30);
    outputPort |= (1<<3);
    outportb(0x378, outputPort);
    delay(30);
    outputPort = 0;
    outportb(0x378, outputPort);
}

void init();

void action()
{
    if(curState.alarm == 1){
        sound(700);
        delay(1000);
        nosound();
        init();
        outputLiftPos(0, 1, 1);
        printf("\nOpening Door ...\n");
        delay(DOOR_DELAY);
    }
}

```

```

        printf("Closing Door ... \n");
        outputLiftPos(0, 1, 0);
        return;
    }
    if(curState.doorOpen==1)
    {
        outputLiftPos(curState.floor, curState.dir, 1);
        printf("\nOpening Door ...\n");
        delay(DOOR_DELAY);
        printf("Closing Door ... \n");
        outputLiftPos(curState.floor, curState.dir, 0);
        printf("# OPENNING DOOR ...\n");
        return;
    }
    if(curState.inLift[curState.floor]==1
        || (curState.up[curState.floor] && curState.dir)
        || (curState.down[curState.floor] && (1-curState.dir)))
    {
        //open door cmd

        outputLiftPos(curState.floor, curState.dir, 1);
        printf("\nOpening Door ...\n");
        delay(DOOR_DELAY);
        printf("Closing Door ... \n");
        outputLiftPos(curState.floor, curState.dir, 0);
    }

    if(curState.dir==1)//up
    {
        int i;
        for(i=curState.floor+1;i<3 && i<maxf;i++)
        {
            if(curState.up[i]==1)
            {
                //move up
                //output lift position
                moveLift(1);
                outputLiftPos(curState.floor+1, 1,0);

                if(curState.up[curState.floor+1]==1||
                    curState.inLift[curState.floor+1]==1)
                {
                    //open door cmd
                    outputLiftPos(curState.floor+1, 1, 1);
                    printf("\nOpening Door ...\n");
                }
            }
        }
    }

```



```

        delay(DOOR_DELAY);
        printf("Closing Door ... \n");
        outputLiftPos(curState.floor+1, 1, 0);
    }
    return;
}
}
for(i=curState.floor+1;i<=3 && i <= maxf;i++)
{
    if(curState.inLift[i]==1)
    {
        //move up
        //output lift position
        moveLift(1);
        outputLiftPos(curState.floor+1, 1,0);
        if(curState.inLift[curState.floor+1]==1)
        {
            //open door cmd
            outputLiftPos(curState.floor+1, 1, 1);
            printf("\nOpening Door ...\n");
            delay(DOOR_DELAY);
            printf("Closing Door ... \n");
            outputLiftPos(curState.floor+1, 1, 0);
        }
        return;
    }
}
for(i=maxf;i>=curState.floor+1;i--)
{
    if(curState.down[i]==1)
    {
        //move up
        //output lift position
        moveLift(1);
        outputLiftPos(curState.floor+1, 1,0);
        if((curState.floor+1)!=i)
            return;
        if(curState.down[curState.floor+1]==1||
           curState.inLift[curState.floor+1]==1)
        {
            //open door cmd
            outputLiftPos(curState.floor+1, 1, 1);
            printf("\nOpening Door ...\n");
            delay(DOOR_DELAY);
            printf("Closing Door ... \n");
            outputLiftPos(curState.floor+1, 0, 0);
        }
    }
}

```

```

        //curState.down[curState.floor+1]=0;
    }
    return;
}

}

//output direction down
delay(100);
fd=curState.down[curState.floor];
outputLiftPos(curState.floor, 0,0);
return;

}
else if(curState.dir==0) // down
{
    int i;
    for(i=curState.floor-1;i>0 && i > minf;i--)
    {
        if(curState.down[i]==1)
        {
            //move down
            //output lift position
            moveLift(0);
            outputLiftPos(curState.floor-1, 0,0);
            if(curState.down[curState.floor-1]==1||
                curState.inLift[curState.floor-1]==1)
            {
                //open door cmd
                outputLiftPos(curState.floor-1, 0, 1);
                printf("\nOpening Door ...\n");
                delay(DOOR_DELAY);
                printf("Closing Door ... \n");
                outputLiftPos(curState.floor-1, 0, 0);
            }
            return;
        }
    }
}
for(i=curState.floor-1;i>=0 && i >= minf;i--)
{
    if(curState.inLift[i]==1)
    {
        //move down
        //output lift position
        moveLift(0);
        outputLiftPos(curState.floor-1, 0, 0);
    }
}

```

```

        if(curState.inLift[curState.floor-1]==1)
        {

            outputLiftPos(curState.floor-1, 0, 1);
            printf("\nOpening Door ...\n");
            delay(DOOR_DELAY);
            printf("Closing Door ... \n");
            outputLiftPos(curState.floor-1, 0, 0);
            //open door cmd
        }
        return;
    }
}
for(i=minf;i<=curState.floor-1;i++)
{
    if(curState.up[i]==1)
    {
        //move down
        //output lift position
        moveLift(0);
        outputLiftPos(curState.floor-1, 0, 0);
        if((curState.floor-1)!=i)
            return;

        if(curState.up[curState.floor-1]==1||
            curState.inLift[curState.floor-1]==1)
        {
            //open door cmd
            outputLiftPos(curState.floor-1, 0, 1);
            printf("\nOpening Door ...\n");
            delay(DOOR_DELAY);
            printf("Closing Door ... \n");
            outputLiftPos(curState.floor-1, 1, 0);
        }
        return;
    }
}

//output direction up
delay(100);
fu=curState.up[curState.floor];
outputLiftPos(curState.floor, 1, 0);
return;
}
}

```

```

void init()
{

    do{
        curState = readState();
        delay(10);
        st1 = readState();
        if(st1.doorClose==1)
            break;
        moveDowninit();
    }while(1);
    curState.clear();
    fd=fu=0;
    outputLiftPos(0,1,0);

}

```

```

void mode2(char ch)
{
    int tmp;
    if(ch != 'x')return;
    clrscr();
    printf("Entering manual mode ...\n");
    while(1){
        printf("1.up\n2.down\n3.exit\n4.alarm\n5.set max\n6.set min\n\n");
        printf("Choice: ");
        ch = getche();
        if(ch=='1'){
            moveLift(1);
        }
        else if(ch=='2'){
            moveLift(0);
        }
        else if(ch=='3'){
            printf("\n");
            init();
            break;
        }
        else if(ch=='4'){
            init();
        }
        else if(ch=='5'){
            printf("\nnew max value(1-4): ");

```

```

        scanf("%d", &tmp);
        if(tmp<1||tmp>4||tmp<minf){
            printf("bad choice...\n");
        }
        else{
            maxf = tmp-1;
        }
    }
    else if(ch=='6'){
        printf("\nnew min value(1-4): ");
        scanf("%d", &tmp);
        if(tmp<1||tmp>4||tmp>maxf){
            printf("bad choice...\n");
        }
        else{
            minf = tmp-1;
        }
    }
}

printf("\n");
}

}

int main()
{
    init();
    clrscr();

    maxf = 3;
    minf = 0;

    while(1){

        if(kbhit()){
            char ch = getch();
            printf("%c\n", ch);
            if(ch=='0')break;
            mode2(ch);
        }

        st1 = readState();

        delay(100);
    }
}

```

```

    st2 = readState();

    if(!st1.isSame(st2))
        continue;

    st2.up[st2.floor]=(st2.up[st2.floor] || fu)?1:0;
    st2.down[st2.floor]=(st2.down[st2.floor] || fd)?1:0;;
    fu=fd=0;
    curState=st2;
    show(curState);
    if(curState.floor==0)
    {
        do{
            curState = readState();
            delay(10);
            st1 = readState();
            if(st1.doorClose==1)
                break;
            moveDowninit();
            delay(10);
        }while(1);

    }

    action();
}

return 0;
}

```