Assignment No. A-1

Roll No:

Aim :

Develop an application using Beeglebone Black/ ARM Cortex A5 development board to simulate the operations of LIFT.

Software Required :

- Linux Operating System
- GCC Compiler.

Hardware Required :

- Beaglebone Black/ ARM Cortex Processor
- Lift 1+3 elevator kit
- Interfacing cables

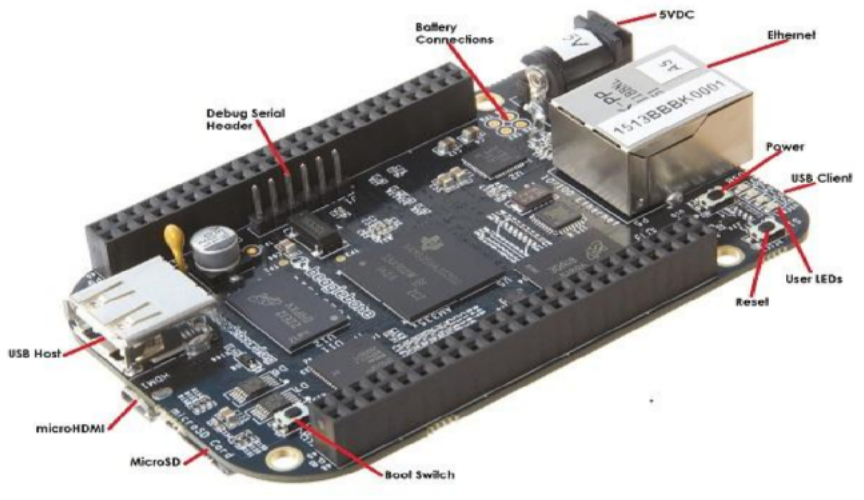
Theory :

Beeglebone Black/ ARM Cortex development board

The BeagleBoard is a low-power open-source hardware single-board computer produced by Texas Instruments in association with Digi-Key and Newark element14. The BeagleBoard was also designed with open source software development in mind, and as a way of demonstrating the Texas Instrument's OMAP3530 system-on-a-chip. The board was developed by a small team of engineers as an educational board that could be used in colleges around the world to teach open source hardware and software capabilities

LIFT Operation Simulation

On the left side of the screen switches appear and on the right side of the screen there are LEDs. The lift facility has a total of the inputs. The contactors are designated and the light indicators. By clicking on the switch the corresponding actuator is activated. There are also contactors for the lift



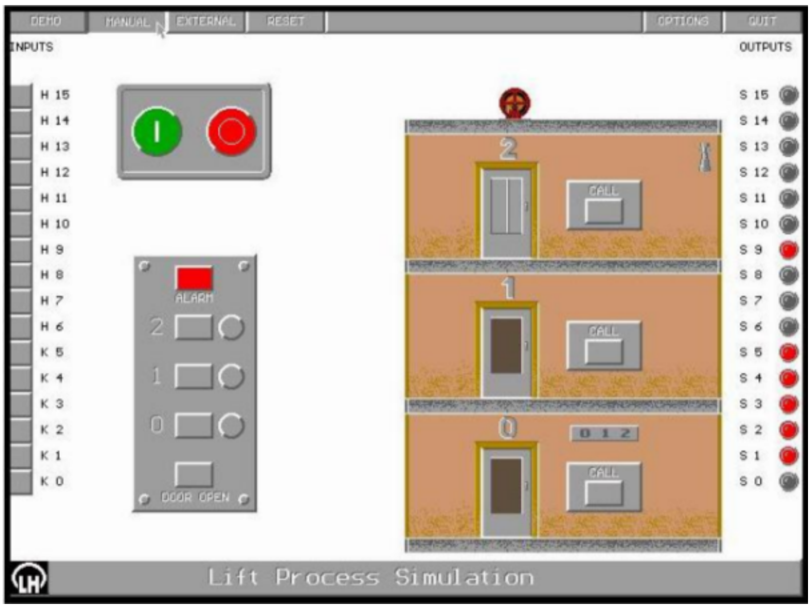
motor, which moves the cage up and down of the various floors. The light indicators are located within the cage in the floor select buttons and directly to the right of the buttons to indicate the last floor arrived at. The switch numbers are identical to the input numbers of the PC adapter. The sensor or switch statuses of the facility are indicated via LEDs on the right side. All facility outputs are generally designated with switches and are set when the operating elements are activated using when the lift cage activates a floor limit switch.

Floor Selection of the Lift

There are two operating elements for the selection of the desired floor. One operating element is the lift call button on the destination floor and the second is the floor selection button inside the cabin. If one of the buttons is activated, the floor selection is stored.

Determining the Up/Down Direction of the Lift

The up/down direction of the cage is dependent on the current position of the lift and the requested destinations. The lift destinations are stored. For the position it suffices to know whether the cage was last on the ground floor or on the second floor. The cage goes down if someone wants to reach the ground floor, but has not yet reached it or when the cage was on the second floor and the destination selected was the 1 st floor. When travelling down, lift up operation must still be inhibited so that the cage does not continuously

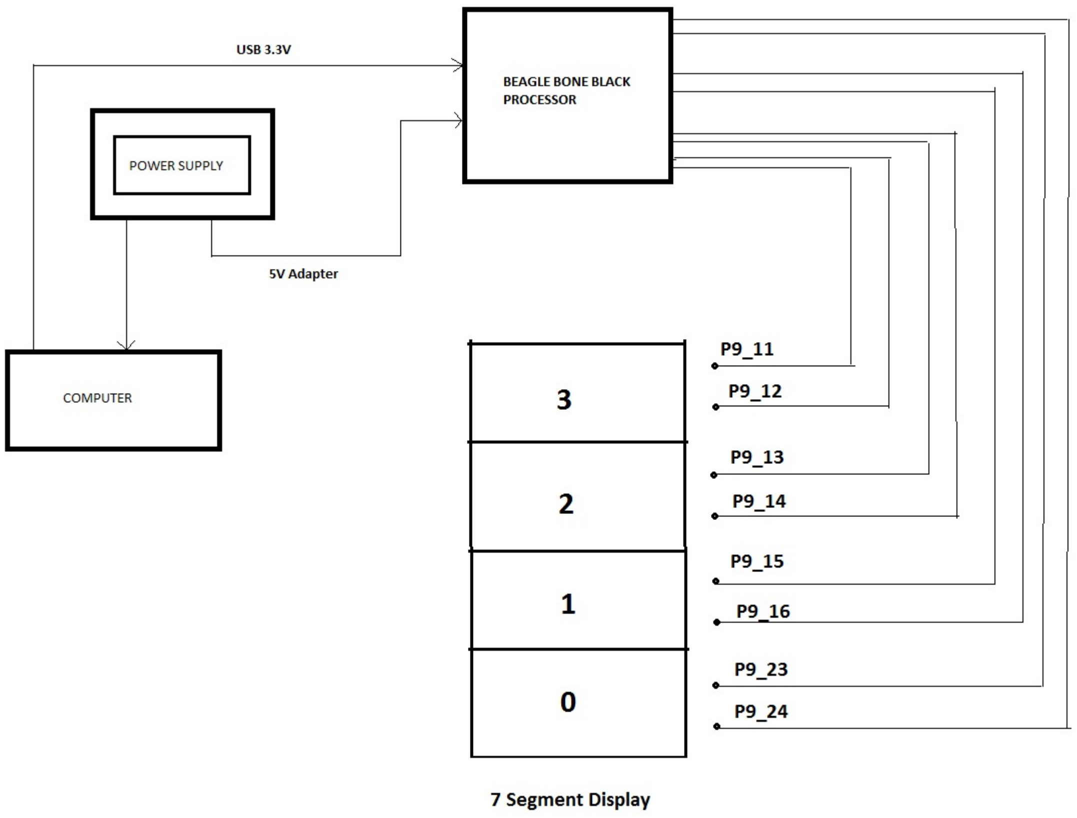


change directions.
This results in a control whereby the previous direction, even with a stopover, still has the first priority; prior instructions are treated with preference. The lift up direction becomes active when someone requests the 2nd floor as destination, but has not yet arrived there or when the lift was down on the ground floor and the 1st floor was selected as the destination.

Algorithm

- Start the kit.
- The lift is by default on the ground floor .
- Take user input (Floor Number).
- Change the seven segment display accordingly .
- Repeat the previous step until we reach the destination .
- Exit.

Interfacing Diagram



Mathematical Model

Let S be a set such that

S={s, e, i, o, f, DD, NDD, success, failure}

s= initial state

e = end state

i= input of the system.

o= output of the system.

f= functions

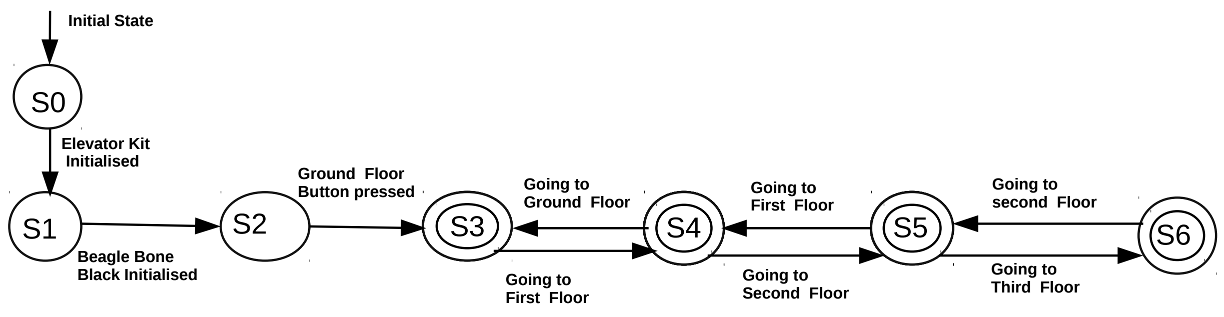


Figure 1: State Diagram

DD-deterministic data it helps identifying the load store functions or assignment functions.

NDD- Non deterministic data of the system S to be solved.

Success-desired outcome generated.

Failure-Desired outcome not generated or forced exit due to system error.

States:{ S0 ,S1 , S2 , S3 , S4 , S5 }

S0: initial State (Power supply)

S1: Elevator

S2: Beagle Board Black

S3: Final State (Ground floor state)

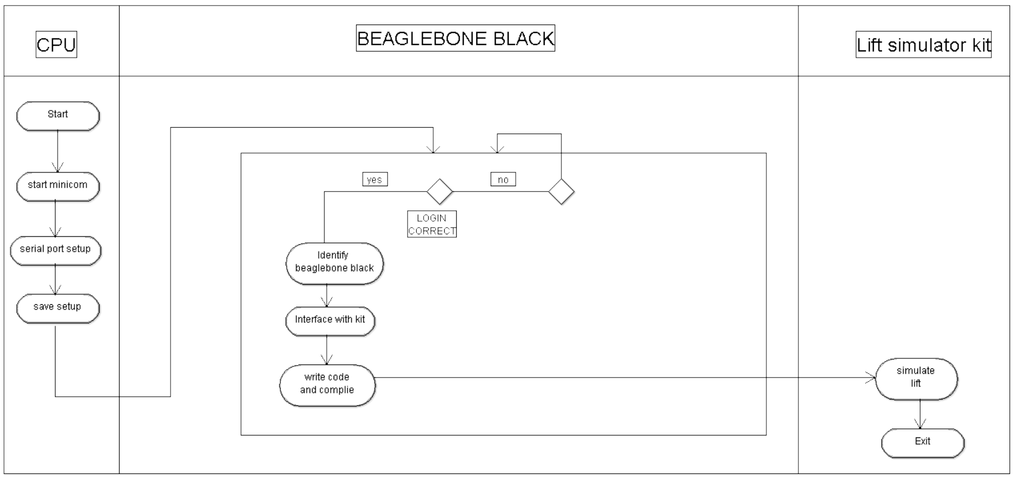
S4: Final State (First floor state)

S5: Final State (Second floor)

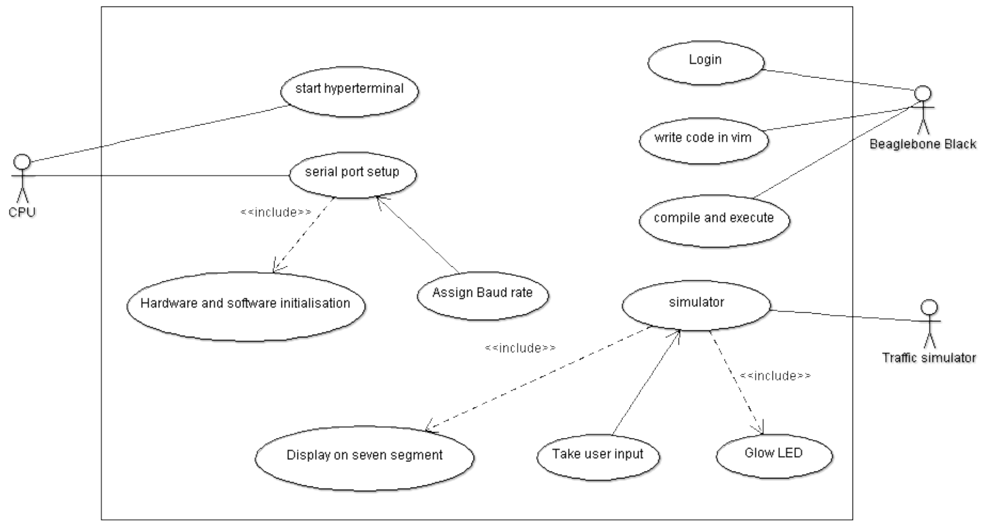
S6: Final State (Third floor)

UML Diagrams :

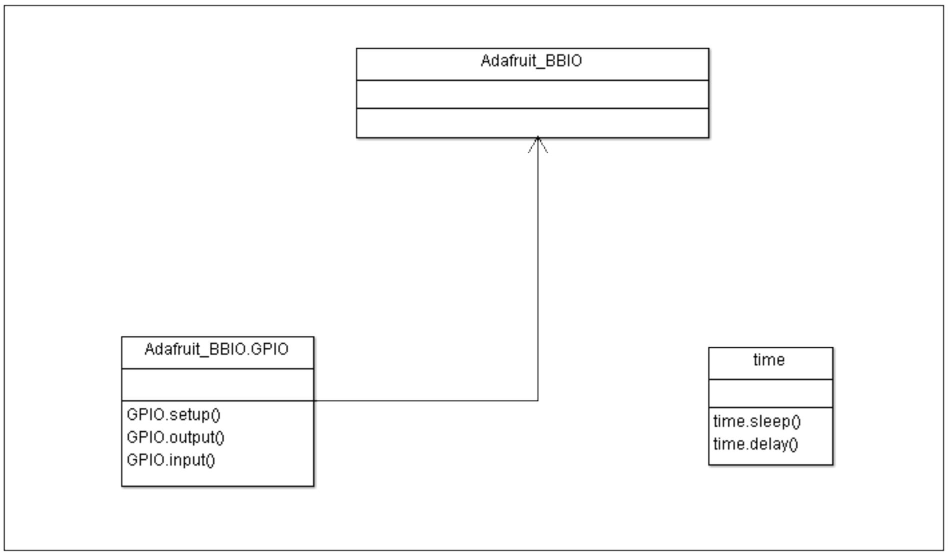
Activity Diagram



Use-case Diagram



Class Diagram



Conclusion :

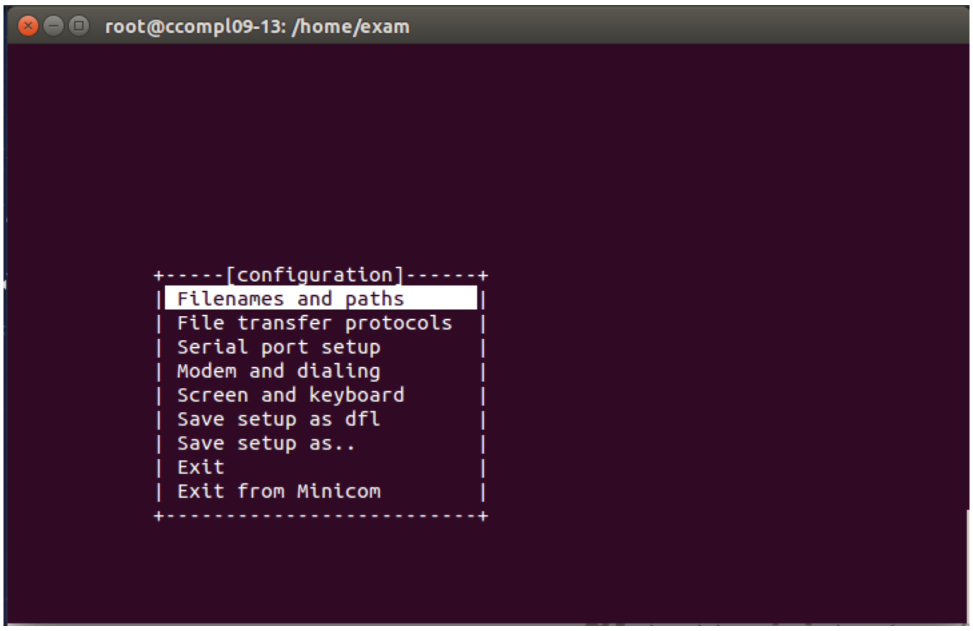
Hence, we have successfully simulated Lift operations using Beeglebone Black/ ARM Cortex development board.

Course Outcomes :

Course Outcomes	Tick [√]
Ability to perform multi-core, Concurrent and Distributed Programming	
Ability to perform Embedded Operating Systems Programming	
Ability to write Software Engineering Document	
Ability to perform Concurrent and Distributed Programming	

Output (Screenshots)

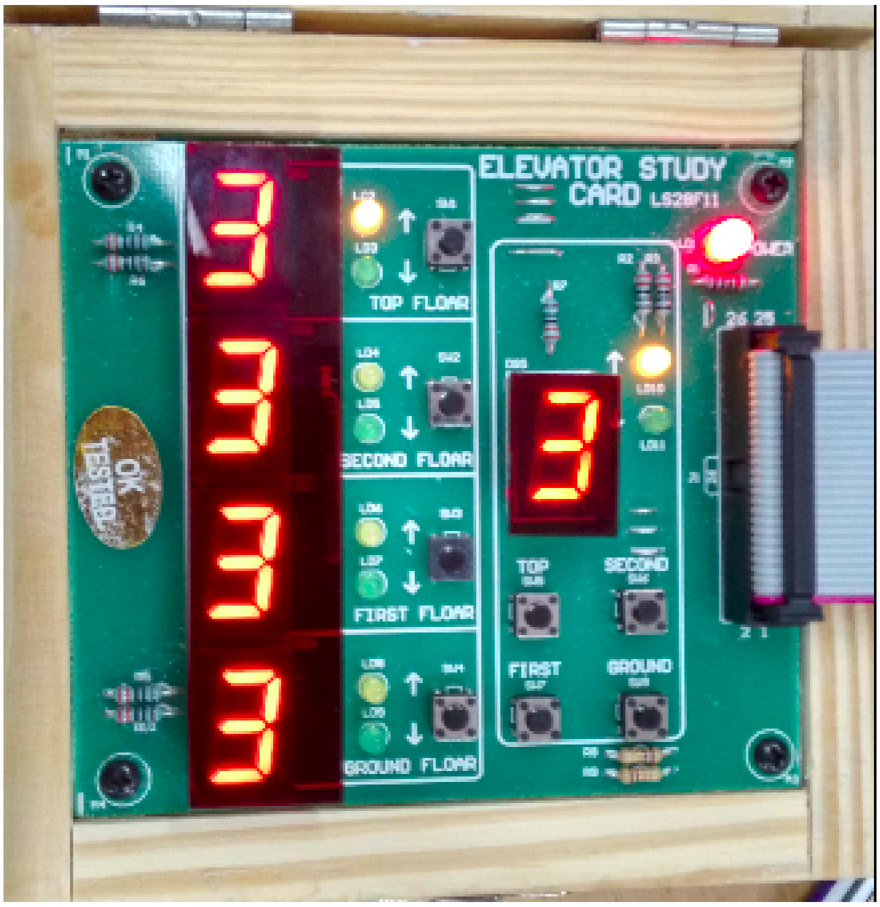
Minicom Terminal



```
root@ccompl09-13: /home/exam

+-----[configuration]------+
| Filenames and paths      |
| File transfer protocols  |
| Serial port setup        |
| Modem and dialing        |
| Screen and keyboard      |
| Save setup as dfl        |
| Save setup as..          |
| Exit                     |
| Exit from Minicom        |
+--------------------------+
```

Output Example



Code :

```
import Adafruit_BBIO.GPIO as GPIO
import time

led_pins = ['P9_23', 'P9_24', 'P9_11', 'P9_12', 'P9_13', 'P9_14', 'P9_15', 'P9_1
seg = ['P8_11', 'P8_12', 'P8_13', 'P8_14', 'P8_15', 'P8_16', 'P8_17', 'P8_18']
switch = ['P8_7', 'P8_8', 'P8_9', 'P8_10']

zero = ['P8_11', 'P8_12', 'P8_13', 'P8_14', 'P8_15', 'P8_16']
one = ['P8_12', 'P8_13']
two = ['P8_11', 'P8_12', 'P8_14', 'P8_15', 'P8_17']
three = ['P8_11', 'P8_12', 'P8_13', 'P8_14', 'P8_17']

for i in range(len(led_pins)):
GPIO.setup(led_pins[i], GPIO.OUT)
GPIO.setup(seg[i], GPIO.OUT)

for j in range(len(switch)):
GPIO.setup(switch[j], GPIO.IN)

def led_clear():
for i in range(len(led_pins)):
GPIO.output(led_pins[i], GPIO.LOW)

def seg_clear():
for i in range(len(seg)):
GPIO.output(seg[i], GPIO.HIGH)

def seg_disp(b):
if b==0:
seg_clear()
for i in range(len(zero)):
GPIO.output(zero[i], GPIO.LOW)
if b==1:
seg_clear()
for j in range(len(one)):
GPIO.output(one[j], GPIO.LOW)
if b==2:
seg_clear()
for k in range(len(two)):
```

```
GPIO.output(two[k], GPIO.LOW)
if b==3:
seg_clear()
for l in range(len(three)):
GPIO.output(three[l], GPIO.LOW)

num=0
var1=0
var2=0
var3=0
var4=0
var5=0
var6=var7=var8=var9=var10=0
p=1

def led(m,ch):
if((m>=0) & (m<=3)):
if ch == 0:
led_clear()
GPIO.output("P9_23",GPIO.HIGH)
GPIO.output("P9_24",GPIO.HIGH)
elif ch == 1:
led_clear()
GPIO.output("P9_15",GPIO.HIGH)
elif ch == 2:
led_clear()
GPIO.output("P9_13",GPIO.HIGH)
elif ch == 3:
led_clear()
GPIO.output("P9_11",GPIO.HIGH)
elif m>3:
if ch == 3:
led_clear()
GPIO.output("P9_11",GPIO.HIGH)
elif ch == 2:
led_clear()
GPIO.output("P9_14",GPIO.HIGH)
elif ch == 1:
led_clear()
GPIO.output("P9_16",GPIO.HIGH)
elif ch == 0:
```

```
led_clear()
GPIO.output("P9_23",GPIO.HIGH)
GPIO.output("P9_24",GPIO.HIGH)

while True:
#print(GPIO.input("P8_7"),GPIO.input("P8_8"),GPIO.input("P8_9"),GPIO.input("P8_1
if (GPIO.input("P8_10")==0 and var1==0 ):#0
seg_disp(0)
if var6==1:
led(4,0)
else:
led(0,0)
time.sleep(1)
var2=var6=0
var1=1
var4=var5=var3=p=1
var7=var8=var9=1
elif (GPIO.input("P8_8")==0 and var6==0):#1
seg_disp(1)
if (var3==1 and p==0):
led(4,1)
else:
led(0,1)
time.sleep(1)
num=0
var1=var3=var7=p=0
var8=var6=var5=1
elif (GPIO.input("P8_9")==0 and var3==0):#2
seg_disp(2)
if (var5 == 1 and p==1):
led(4,2)
else:
led(0,2)
time.sleep(1)
var6=var5=var4=p=0
var3=var1=1
elif (GPIO.input("P8_7")==0 and var5==0):#3
seg_disp(3)
led(0,3)
time.sleep(1)
var1=var6=var5=var4=p=1
```

```
var3=var9=var8=0
elif (GPIO.input("P8_8")==0 and var8==0):#3-1
seg_disp(2)
led(4,2)
time.sleep(1)
seg_disp(1)
led(4,1)
time.sleep(1)
var1=var3=var7=0
var8=var5=var9=1
elif (GPIO.input("P8_7")==0 and var7==0):#1-3
seg_disp(2)
led(0,2)
time.sleep(1)
seg_disp(3)
led(0,3)
time.sleep(1)
var3=var8=var9=0
var1=var5=var7=var4=1
elif (GPIO.input("P8_10")==0 and var9==0):#3-0
seg_disp(2)
led(4,2)
time.sleep(1)
seg_disp(1)
led(4,1)
time.sleep(1)
seg_disp(0)
led(4,0)
time.sleep(1)
var1=var3=var5=var9=p=var7=1
var2=var6=0
elif (GPIO.input("P8_7")==0 and var2==0 ):#0-3
seg_disp(1)
led(0,1)
time.sleep(1)
seg_disp(2)
led(0,2)
time.sleep(1)
seg_disp(3)
led(0,3)
time.sleep(1)
```

```
var1=var2=var4=var5=var6=var7=1
var3=var8=var9=0
elif (GPIO.input("P8_9")==0 and var2==0):#0-2
seg_disp(1)
led(0,1)
time.sleep(1)
seg_disp(2)
led(0,2)
time.sleep(1)
var4=var5=var6=p=0
var2=var1=var3=1
elif (GPIO.input("P8_10")==0 and var4==0):#2-0
seg_disp(1)
led(4,1)
time.sleep(1)
seg_disp(0)
led(4,0)
time.sleep(1)
var1=var3=var5=var7=p=1
var8=var9=1
var2=var6=0
```