

Assignment No. A-2

Roll No:

Aim :

Develop an application using Beaglebone Black/ ARM Cortex A5 development board to simulate the working of signal lights.

Software Required :

- Linux Operating System
- GCC Compiler.

Hardware Required :

- Beaglebone Black/ ARM Cortex Processor
- Traffic light simulation kit
- Interfacing cables

Theory :

The BeagleBone is a 1GHz credit card sized computer that runs Linux operating system and has extensive input/output capabilities to control and measure the world around it. The performance is comparable to a 'smart phone'. Using a 32-bit ARM processor, the BeagleBone Black can serve webpages interact with a cell-phone, control motors and act as the brains of a robot Initially intended for academic use, the BeagleBone computer has gain popularity with the maker movement, tinkerer and the interested. The powerful computer is used in new applications for industrial, automation, education, and arts and imaginative and evens some quirky projects. The low cost provides easy accessibility to extensive technology. All this fits in a small Altoid mint tin!

The first time you plug in the BeagleBone it will appear as a 'USB thumb drive' with the required drivers in the Driver folder (Windows, Mac OSX and Linux versions). (Takes about 20 seconds) There Are Many Ways To Connect To The BeagleBone:

- Serial port (J1)

- ssh (using USB cable)
- Stand Alone -keyboard + display
- VNC - Virtual Network Connection
- WebBrowser (Cloud9)

Advantages of BeagleBone Black

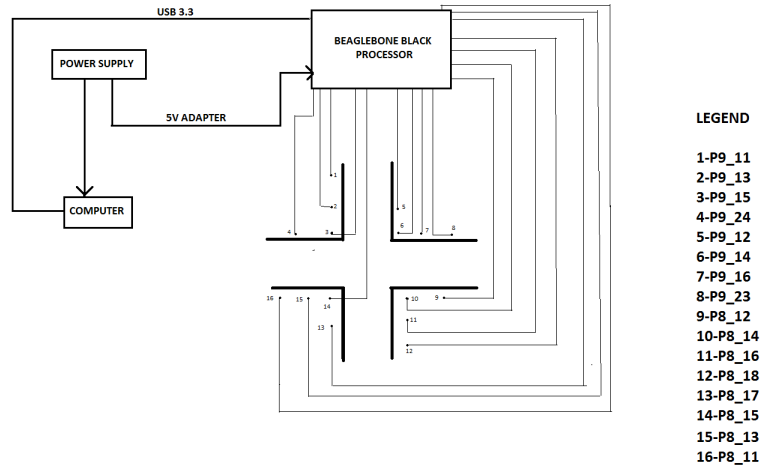
- A low-cost but fully capable platform - BBB offers an incredible value for developers of all ability levels
- Small and power efficient - advanced 1GHz, ARMv7-based processor with low power needs in a compact package
- System-on-a-chip (SOC) capabilities - all standard computing functions built-in
- Simple set-up - included mini-USB cable
- Wide range of IO options - 2x 46 pin headers allow for a wide range of connectivity options
- New REV C version - flash memory doubled to 4GB, Debian version of Linux onboard
- Embest commercial BB clone - makes larger scale embedded projects a reality

Traffic Light Controller Interface Kit

- 4 junction Road, 4 set of Red, Green and Yellow LEDs
- Pedestrian Crossings bi-colour LEDs
- Control switch to control day and night operations
- Closed wooden cabinet

Traffic Light Control

Traffic lights, which may also be known as stoplights, traffic lamps, traffic signals, signal lights, robots or semaphore, are signaling devices positioned at road intersections, pedestrian crossings and other locations to control competing flows of traffic.



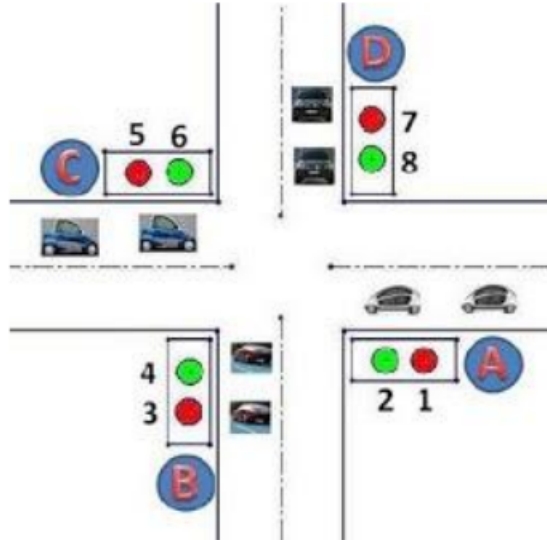


Figure 1: Traffic light intersection scenario

About the colors of Traffic Light Control

Traffic lights alternate the right of way of road users by displaying lights of a standard color (red, yellow/amber, and green), using a universal color code (and a precise sequence to enable comprehension by those who are color blind). In the typical sequence of colored lights:

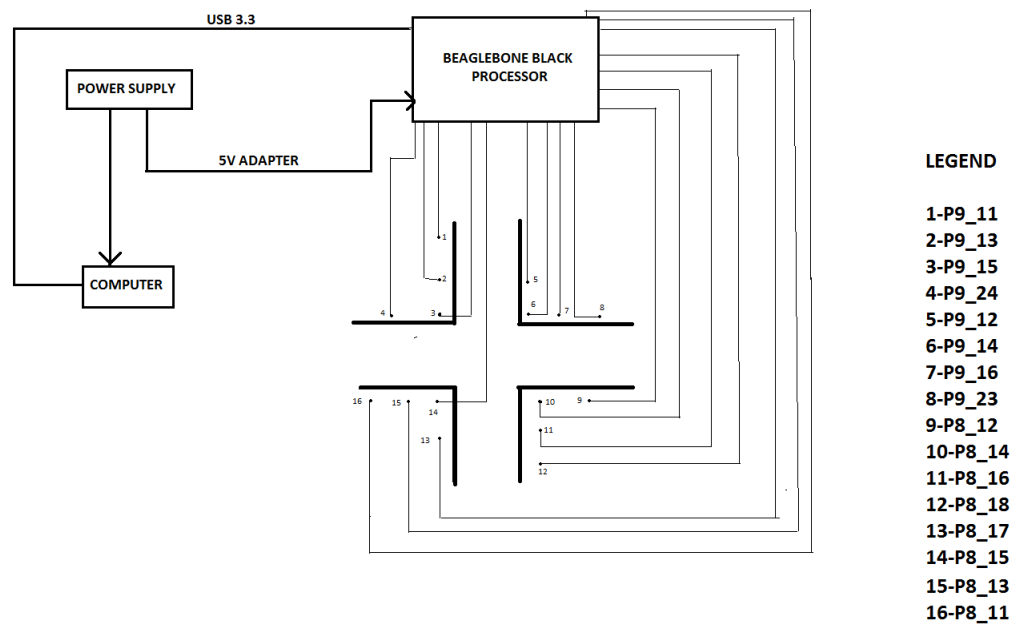
- Illumination of the green light allows traffic to proceed in the direction denoted,
- Illumination of the yellow/amber light denoting, if safe to do so, prepare to stop short of the intersection, and
- Illumination of the red signal prohibits any traffic from proceeding.

Usually, the red light contains some orange in its hue, and the green light contains some blue, for the benefit of people with red-green color blindness, and "green" lights in many areas are in fact blue lenses on a yellow light (which together appear green).

Algorithm

1. Start the Kit .
2. Start with section 1
 - Set red led to high
 - delay(time)
 - set red led to low
 - set yellow led to high
 - delay(time)
 - set yellow led to low
 - set green to high
 - delay(time)
 - set green to low
3. Repeat step 2 for section 2 ,3 and 4
4. 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

DD-Deterministic Data it helps identifying the load store functions or assignment functions. As we are trying to simulate the Traffic control system, our LED's will in DD state if they are in ON state when system is working properly and traffic is not deadlocked.

NDD- It is Non deterministic data of the system S to be solved. Here only Non deterministic state is pedestrian walk

Success-If Simulator works successfully to simulate proper working of Traffic Signal System

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

States: $\{ S0, S1, S2, S3, S4, S5, S6 \}$

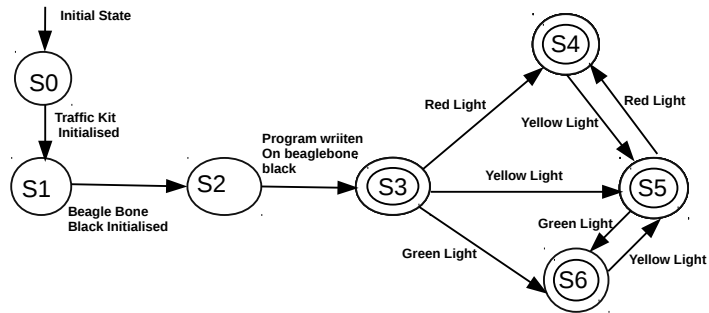
S0: initial State (Power supply)

S1: Monitor Display editor (Write and Design application using gedit)

S2: BeagleBone Black

S3: Traffic controller kit

S4: Final State (Red signal)

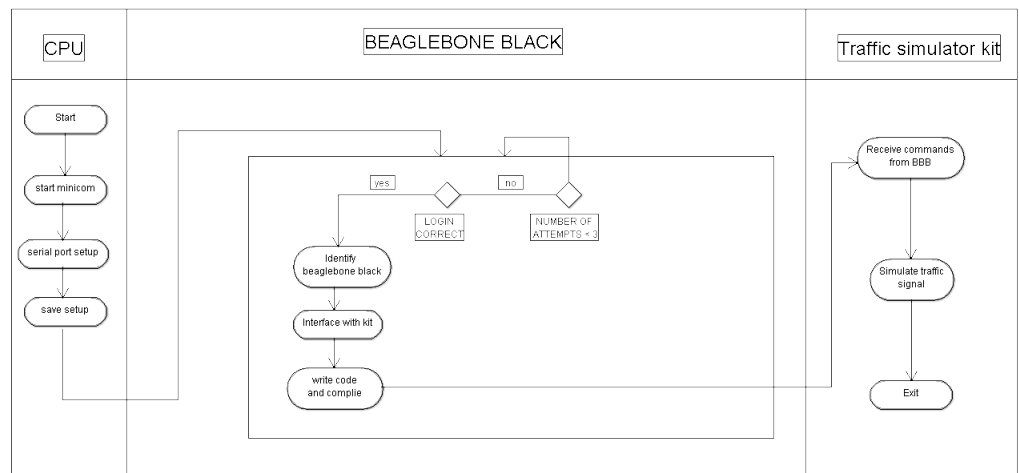


S5: Final State (Yellow signal)

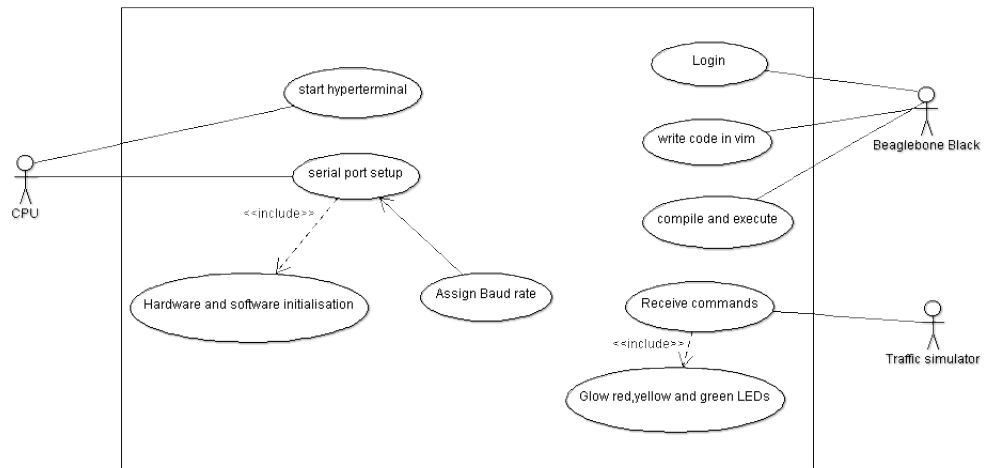
S6: Final State (Green signal)

UML Diagrams :

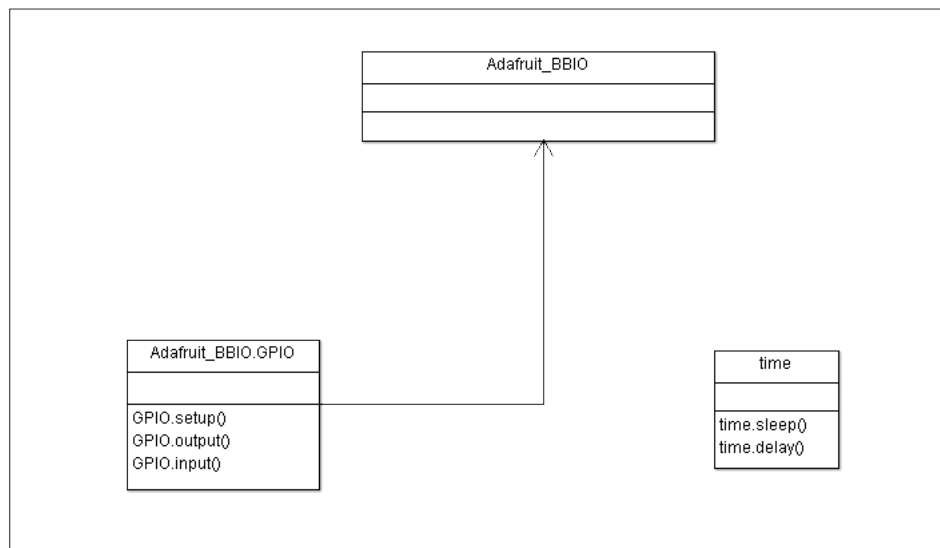
Activity Diagram



Use-case Diagram



Class Diagram



CONCLUSION :

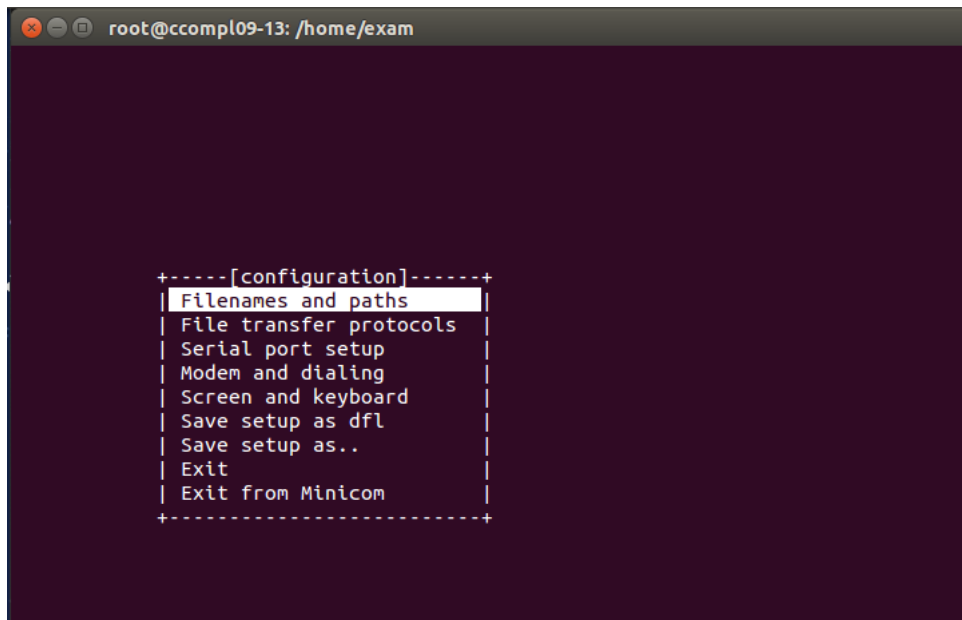
In this experiment we have studied simulation of traffic control system using ARM cortex processor with Beaglebone black board interface .

Course Outcomes :

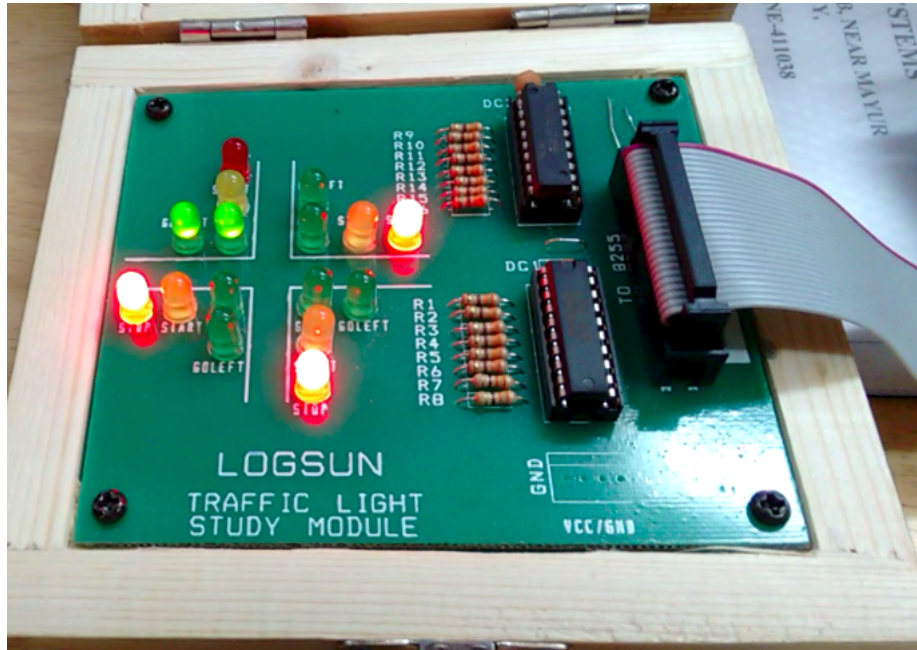
Course Outcomes	Tick [✓]
Ability to perform multi-core, Concurrent and Distributed Programming	
Ability to perform Embedded Operating Systems Programming	
Ability to write Software Engineering Document	
Ability to perform Concurrent and Distributed Programming	

OUTPUT (Screenshots)

Minicom Terminal

A screenshot of a Minicom terminal window. The title bar shows 'root@ccompl09-13: /home/exam'. The terminal has a dark purple background. A menu is displayed with the title '+-----[configuration]-----+'. The menu items are: 'Filenames and paths' (highlighted with a white background), 'File transfer protocols', 'Serial port setup', 'Modem and dialing', 'Screen and keyboard', 'Save setup as dfl', 'Save setup as..', 'Exit', and 'Exit from Minicom'. The menu is enclosed in a dashed white border.

Output Example



Code :

```
import Adafruit_BBIO.GPIO as GPIO
import time

GPIO.setup("P9_11",GPIO.OUT)
GPIO.setup("P9_12",GPIO.OUT)
GPIO.setup("P9_13",GPIO.OUT)
GPIO.setup("P9_14",GPIO.OUT)
GPIO.setup("P9_15",GPIO.OUT)
GPIO.setup("P9_16",GPIO.OUT)
GPIO.setup("P9_23",GPIO.OUT)
GPIO.setup("P9_24",GPIO.OUT)
GPIO.setup("P8_11",GPIO.OUT)
GPIO.setup("P8_12",GPIO.OUT)
GPIO.setup("P8_13",GPIO.OUT)
GPIO.setup("P8_14",GPIO.OUT)
GPIO.setup("P8_15",GPIO.OUT)
GPIO.setup("P8_16",GPIO.OUT)
GPIO.setup("P8_17",GPIO.OUT)
GPIO.setup("P8_18",GPIO.OUT)

GPIO.output("P9_11",GPIO.LOW)
GPIO.output("P9_12",GPIO.LOW)
GPIO.output("P9_13",GPIO.LOW)
GPIO.output("P9_14",GPIO.LOW)
GPIO.output("P9_15",GPIO.LOW)
GPIO.output("P9_16",GPIO.LOW)
GPIO.output("P9_23",GPIO.LOW)
GPIO.output("P9_24",GPIO.LOW)
GPIO.output("P8_11",GPIO.LOW)
GPIO.output("P8_12",GPIO.LOW)
GPIO.output("P8_13",GPIO.LOW)
GPIO.output("P8_14",GPIO.LOW)
GPIO.output("P8_15",GPIO.LOW)
GPIO.output("P8_16",GPIO.LOW)
GPIO.output("P8_17",GPIO.LOW)
GPIO.output("P8_18",GPIO.LOW)

while(1):
    ''' 1 CASE : '''
    GPIO.output("P9_15",GPIO.HIGH)
```

```

GPIO.output("P9_24",GPIO.HIGH)
GPIO.output("P9_23",GPIO.HIGH)
GPIO.output("P8_11",GPIO.HIGH)
GPIO.output("P8_18",GPIO.HIGH)
time.sleep(5)
GPIO.output("P9_15",GPIO.LOW)
GPIO.output("P9_24",GPIO.LOW)
GPIO.output("P9_23",GPIO.LOW)
GPIO.output("P9_13",GPIO.HIGH)
GPIO.output("P9_16",GPIO.HIGH)
    time.sleep(2)
    GPIO.output("P9_13",GPIO.LOW)
GPIO.output("P9_16",GPIO.LOW)
GPIO.output("P9_11",GPIO.HIGH)
GPIO.output("P9_12",GPIO.HIGH)
GPIO.output("P9_14",GPIO.HIGH)
time.sleep(5)
GPIO.output("P9_12",GPIO.LOW)
GPIO.output("P9_14",GPIO.LOW)
GPIO.output("P8_18",GPIO.LOW)
GPIO.output("P8_16",GPIO.HIGH)
GPIO.output("P9_16",GPIO.HIGH)
    time.sleep(2)
    GPIO.output("P9_23",GPIO.HIGH)
GPIO.output("P8_14",GPIO.HIGH)
GPIO.output("P8_12",GPIO.HIGH)
GPIO.output("P8_16",GPIO.LOW)
GPIO.output("P9_16",GPIO.LOW)
time.sleep(5)
GPIO.output("P8_12",GPIO.LOW)
GPIO.output("P8_14",GPIO.LOW)
GPIO.output("P8_11",GPIO.LOW)
GPIO.output("P8_13",GPIO.HIGH)
GPIO.output("P8_16",GPIO.HIGH)
    time.sleep(2)
    GPIO.output("P8_13",GPIO.LOW)
GPIO.output("P8_16",GPIO.LOW)
GPIO.output("P8_18",GPIO.HIGH)
GPIO.output("P8_15",GPIO.HIGH)
GPIO.output("P8_17",GPIO.HIGH)
    time.sleep(5)

```

```
GPIO.output("P8_13",GPIO.HIGH)
GPIO.output("P9_13",GPIO.HIGH)
GPIO.output("P8_15",GPIO.LOW)
GPIO.output("P8_17",GPIO.LOW)
GPIO.output("P9_11",GPIO.LOW)
time.sleep(2)
```