

Assignment No. B-8

Roll No:

Aim :

Develop a network based application by setting IP address on BeagleBoard/ ARM Cortex A5.

Software Required :

- Linux Operating System
- GCC Compiler.

Hardware Required :

- Beaglebone Black/ ARM Cortex Processor
- Interfacing cables

Theory :

Sockets

Sockets are endpoints of a bidirectional communication channel . Sockets may communicate within a process , between two different processes on different machines. Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest. To create a socket, you must use the `socket.socket()` function available in socket module, which has the general syntax:

`s = socket.socket (socket_family, socket_type, protocol=0)`

Here is the description of the parameters:

- **socket_family:** This is either `AF_UNIX` or `AF_INET`, as explained earlier.
- **socket_type:** This is either `SOCK_STREAM` or `SOCK_DGRAM`.
- **protocol:** This is usually left out, defaulting to 0.

Once you have socket object, then you can use required functions to create your client or server program. Following is the list of functions required:

Table 1: Socket Terms

Term	Description
Domain	The family of protocols that will be used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
Type	The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
Protocol	Typically zero, this may be used to identify a variant of a protocol within a domain and type
Port	Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

Table 2: **Socket Server Methods :**

Method	Description
s.bind()	This method binds address (hostname, port number pair) to socket.
s.listen()	This method sets up and start TCP listener.
s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).

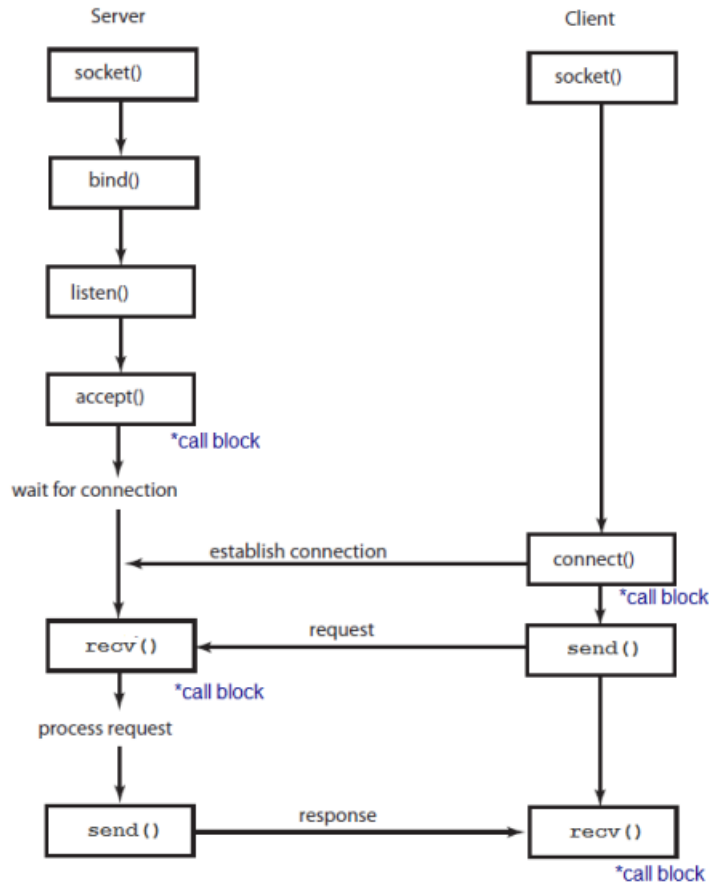
Table 3: **Client Server Methods :**

Method	Description
s.connect()	This method actively initiates TCP server connection.

Table 4: **General Socket Methods :**

Method	Description
s.recv()	This method receives TCP message
s.send()	This method transmits TCP message
s.recvfrom()	This method receives UDP message
s.sendto()	This method transmits UDP message
s.close()	This method closes socket
s.gethostname()	Returns the hostname.

Basic Client-Server Working



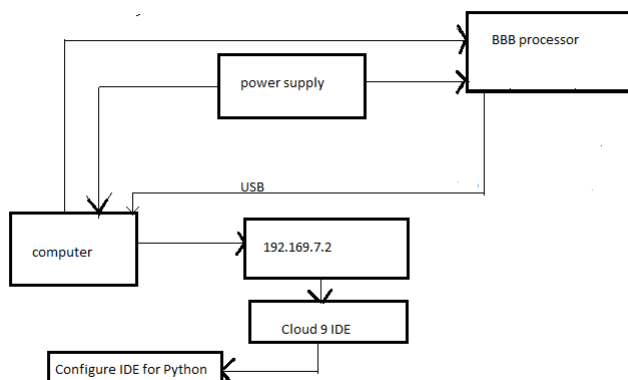
By default, the IP address of the Beaglebone Black is set dynamically using the DHCP server on our network. That is very useful; however, if we wish to map the Beaglebone to an external port (virtual server) on your network, or you wish to force your Beaglebone to have a specific identifiable location in your home you need to change our IP address to be static. The static address of BEAGLE BLACK BONE and Linux system should be same so it is useful to communicate between them.

Algorithm

- connect usb cable to BeagleBone Black and PC.

- Go to network interfaces of BeagleBone Black and change IP address of eth0.
- Connect BeagleBone Black to PC by LAN cable
- Type IP address of BeagleBone Black in browser
- Select Cloud9E.
- Write Server code in Cloud9E
- Write Client code in gedit in PC
- Run Server
- Run Client
- Exit

Interfacing Diagram



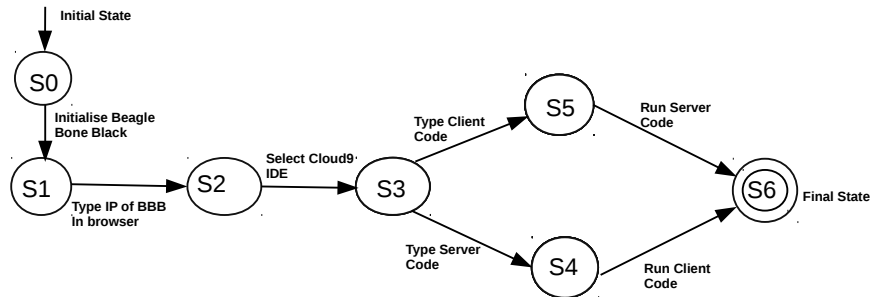
Mathematical Model

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.

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

Success-Client-Server communicate successfully

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: BeagleBone initialisation

S2: BeagleBone Black connected to Network

S3: Connected to Cloud9E

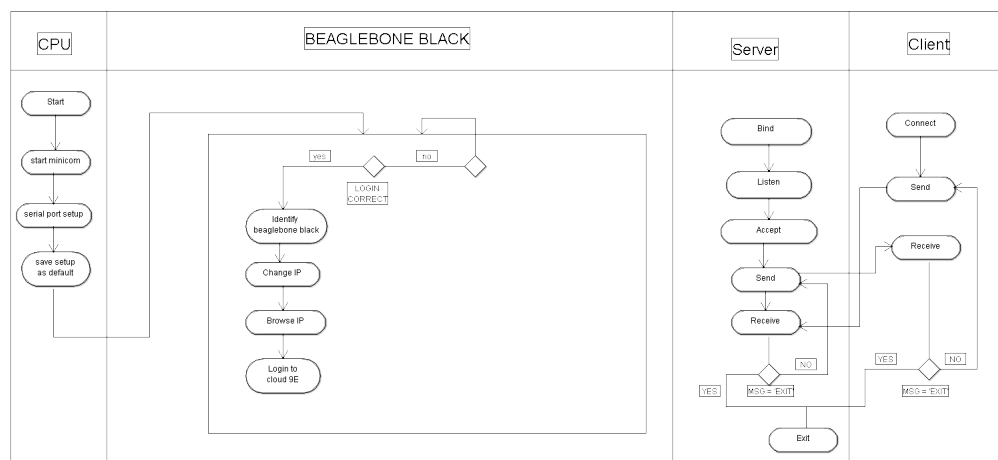
S4: Server code

S5: Client Code

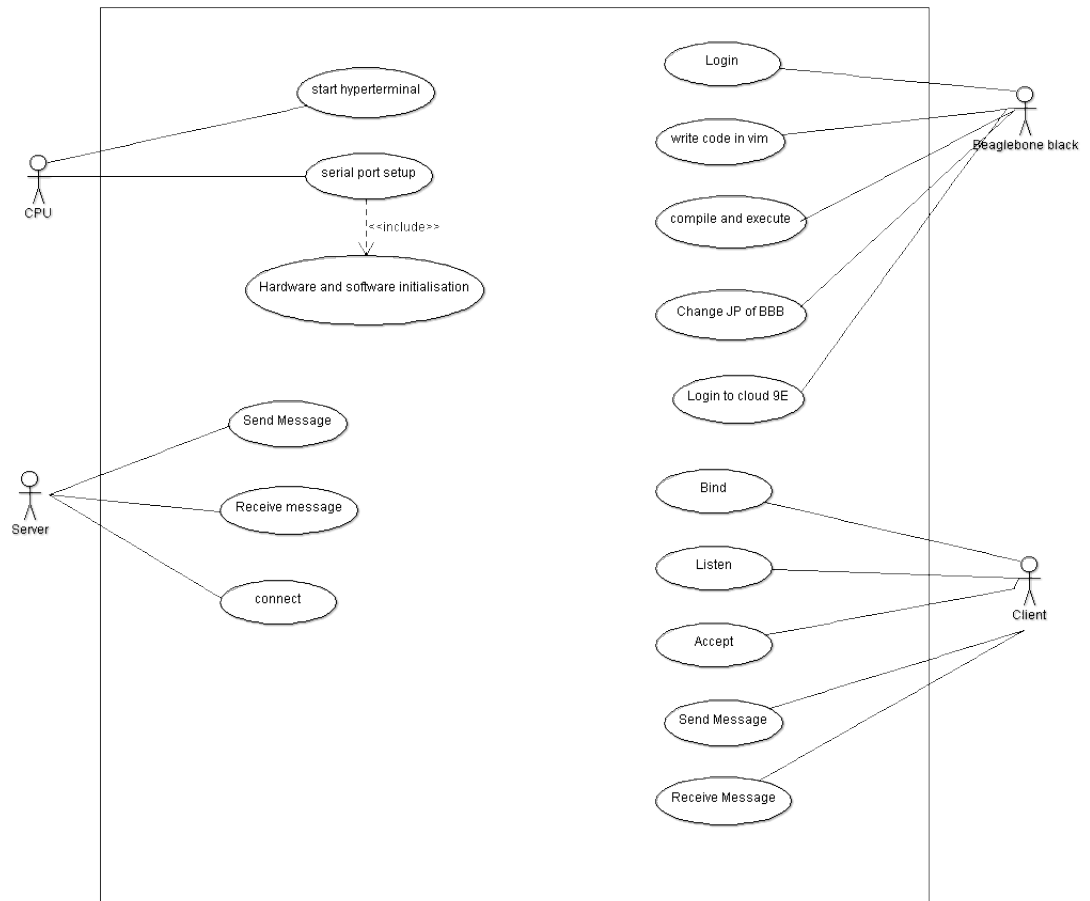
S6 : Server and Client Communication (Final State)

UML Diagrams :

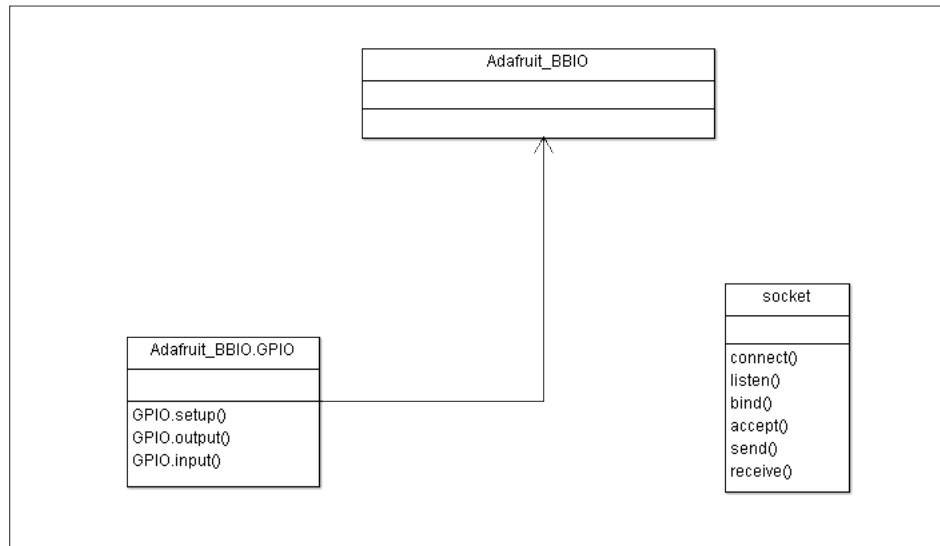
Activity Diagram



Use-case Diagram



Class Diagram



CONCLUSION :

Hence, we develop network application like chat application using static IP address of BEAGLE BLACK BONE.

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	

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   |  
+-----+
```

BeagleBoard.org - bone101 - Mozilla Firefox

BeagleBoard.org - bone101 x cloud9 - Cloud9 x cloud9 - Cloud9 x +

192.168.7.2/Support/bone101/ Google

See [updates](#) for the step-by-step guide.

Information about getting the source code for the image shipped with your board can be found at [beagleboard.org/angstrom](#), along with instructions for rebuilding it.

BoneScript interactive guide

BoneScript is a JavaScript library to simplify learning how to perform physical computing tasks using your embedded Linux. This web page is able to interact with your board to provide an interactive tutorial.

Example

```

1 var b = require('bonescript');
2 b.pinMode('USR0', b.OUTPUT);
3 b.pinMode('USR1', b.OUTPUT);
4 b.pinMode('USR2', b.OUTPUT);
5 b.pinMode('USR3', b.OUTPUT);
6 b.digitalWrite('USR0', b.HIGH);
7 b.digitalWrite('USR1', b.HIGH);
8 b.digitalWrite('USR2', b.HIGH);
9 b.digitalWrite('USR3', b.HIGH);
10 setTimeout(restore, 2000);
11

```

Running the above example will cause all of your LEDs to light up at once for a couple of seconds.

To learn more about BoneScript, please continue exploring [this interactive guide](#).

Cloud9 IDE

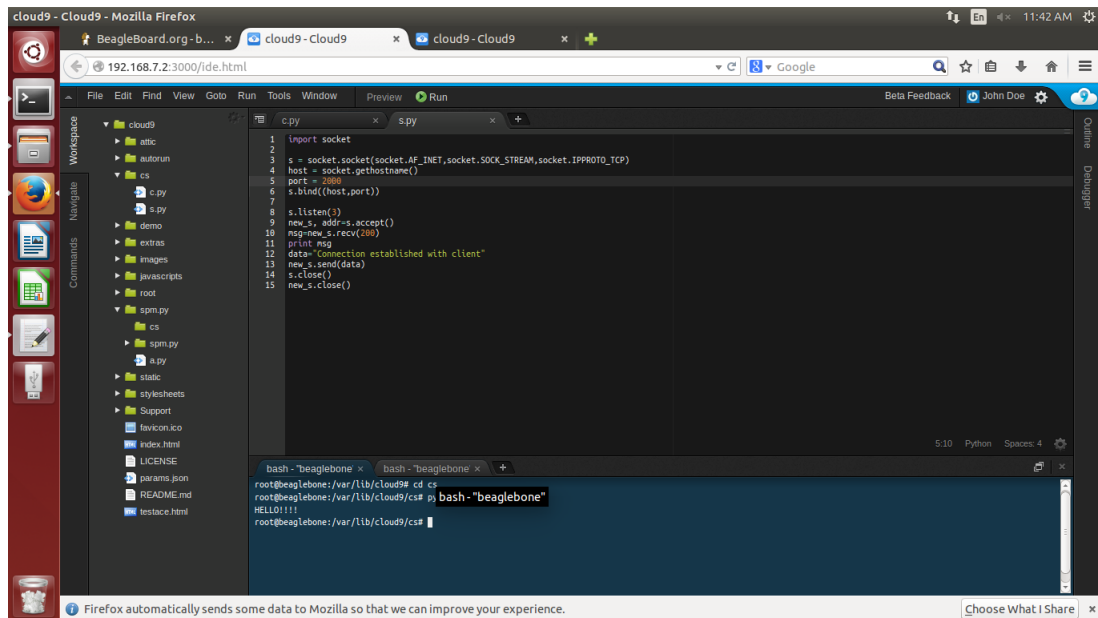
To begin editing programs that live on your board, you can use the Cloud9 IDE.

Click on the "Cloud9 IDE" link above to start the editor.

Firefox automatically sends some data to Mozilla so that we can improve your experience.

Choose What I Share

Server Example

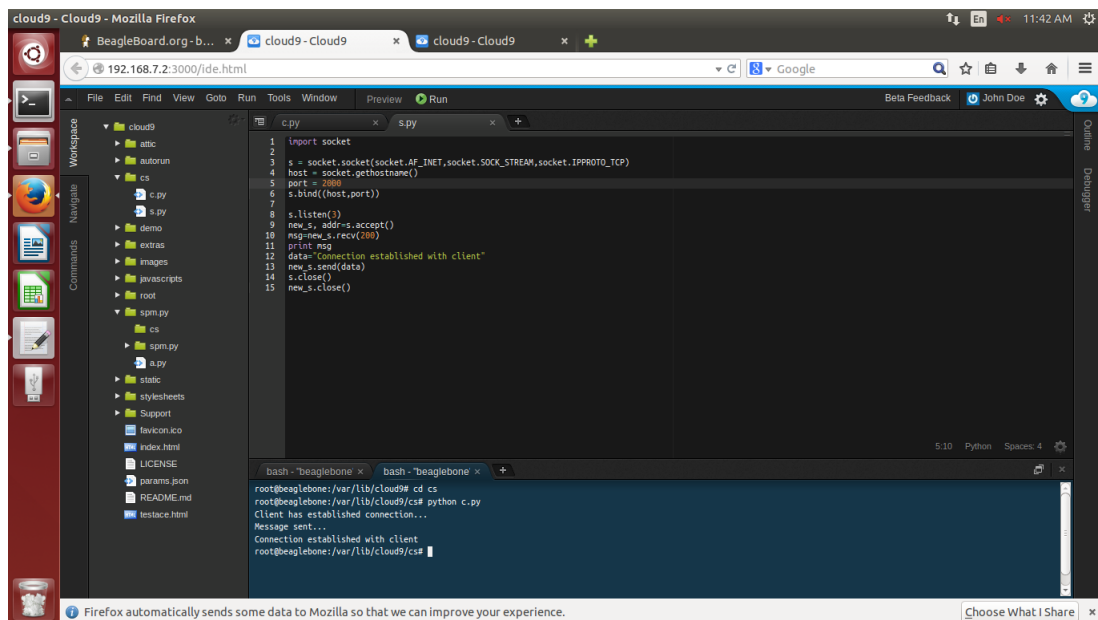


The screenshot shows a web IDE interface with a file explorer on the left, a code editor in the center, and a terminal at the bottom. The file explorer shows a project structure with folders like 'cloud9', 'altic', 'autorun', 'cs', 'demo', 'extras', 'images', 'javascripts', 'root', 'spm.py', 'static', 'stylesheets', 'Support', 'favicon.ico', 'index.html', 'LICENSE', 'params.json', 'README.md', and 'testace.html'. The code editor displays a Python script named 'c.py' that sets up a socket server on port 2000. The terminal shows the command 'bash - "beaglebone"' and the output 'HELLO!!!!'.

```
1 import socket
2
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP)
4 host = socket.gethostname()
5 port = 2000
6 s.bind((host, port))
7
8 s.listen()
9 new_s, addr = s.accept()
10 msg = new_s.recv(1024)
11 print msg
12 data = "Connection established with client"
13 new_s.send(data)
14 s.close()
15 new_s.close()
```

```
bash - "beaglebone"
root@beaglebone:/var/lib/cloud9# cd cs
root@beaglebone:/var/lib/cloud9/cs# python c.py
HELLO!!!!
root@beaglebone:/var/lib/cloud9/cs#
```

Client Example



The screenshot shows the same web IDE interface as the previous one, but with a different script in the code editor. The file explorer and terminal are the same. The code editor displays a Python script named 'c.py' that sets up a socket client to connect to a server on port 2000. The terminal shows the command 'bash - "beaglebone"' and the output 'Client has established connection...'.

```
1 import socket
2
3 s = socket.socket(socket.AF_INET, socket.SOCK_STREAM, socket.IPPROTO_TCP)
4 host = socket.gethostname()
5 port = 2000
6 s.bind((host, port))
7
8 s.listen()
9 new_s, addr = s.accept()
10 msg = new_s.recv(1024)
11 print msg
12 data = "Connection established with client"
13 new_s.send(data)
14 s.close()
15 new_s.close()
```

```
bash - "beaglebone"
root@beaglebone:/var/lib/cloud9# cd cs
root@beaglebone:/var/lib/cloud9/cs# python c.py
Client has established connection...
Message sent...
Connection established with client
root@beaglebone:/var/lib/cloud9/cs#
```

Server Code :

```
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM,socket.IPPROTO_TCP)
host = socket.gethostname()
port = 12348
s.bind((host,port))
s.listen(5)
while (True) :
    new_s, addr=s.accept()
    print 'Got connection from proxy....sending data...'
    user = new_s.recv(200)
    print 'User : ',user
    new_s.send("CONNECTION WAS SUCCESSFULL....")
s.close()
new_s.close()
```

Client Code :

CLIENT CODE

```
import socket
s = socket.socket(socket.AF_INET,socket.SOCK_STREAM,socket.IPPROTO_TCP)
host = socket.gethostname()
port = 12347
print "CONNECTING ....."
s.connect((host, port))
me = raw_input('ROHAN : ')
s.sendall(me)
server = s.recv(200)
print 'SERVER : ',server
s.close()
```