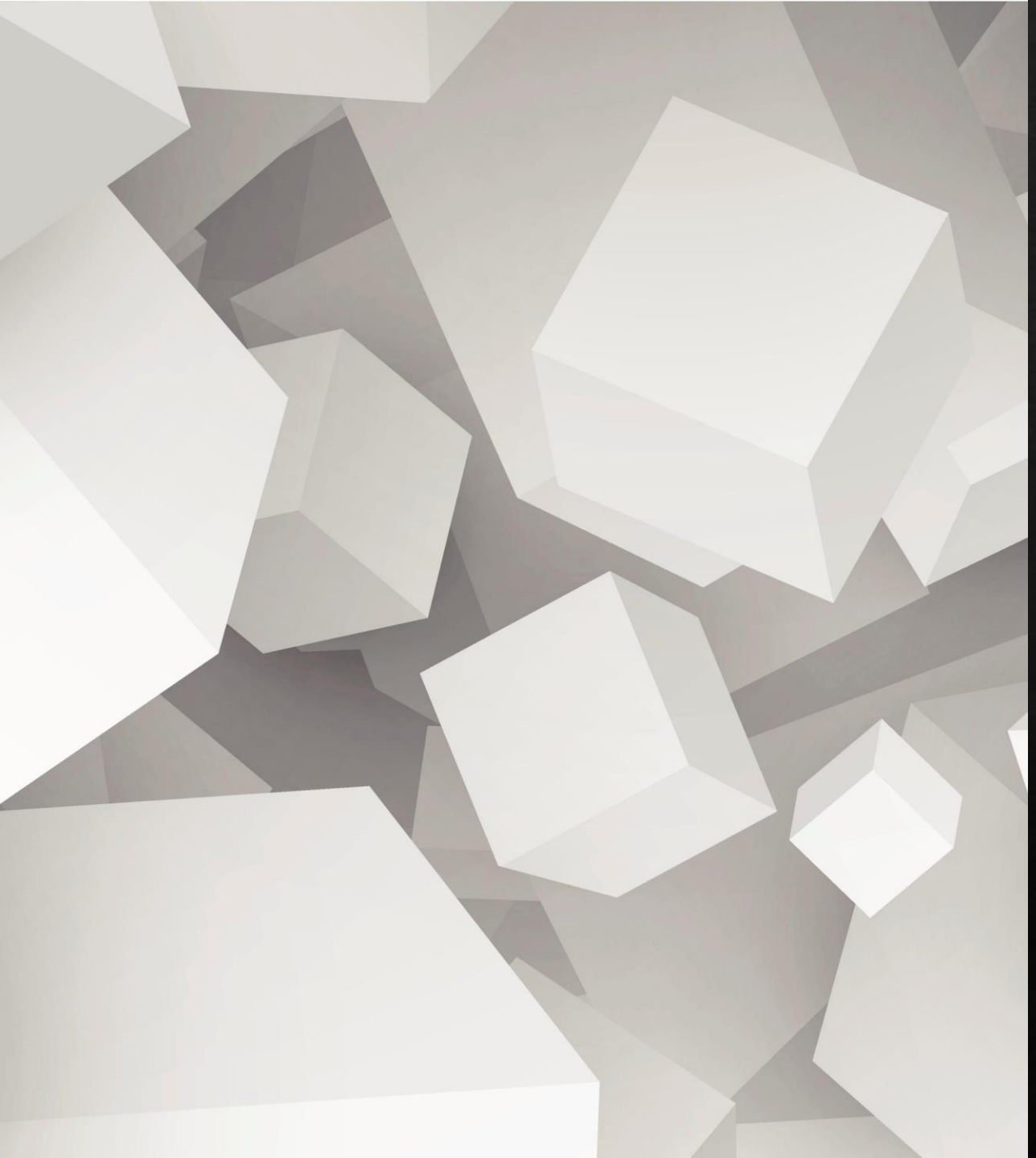
Interfacing with BeagleBone Wireless : Part II

- Zain Rajani (c0752681)

Group # 8



Outline

- Until Now
- Task Objective
- Task Introduction
- Task Requirement
- Interface Programming
- Wire Interface
- Conclusion
- References

Until Now.....

- Until now we have tried to complete the phase I of the interfacing which was to get the LED, IR sensor merged (interfaced) with the Beaglebone Black Wireless
- For the interface we used the user made header files and tried to use the POSIX (**P**ortable **O**perating **S**ystem **I**nterface) API function library such as the unistd.h (**Uni**x **St**andar**d** Header) inorder to get all the processes suspended and showing some of the power management features which can be adopted.
- We also tried to free up the resources which used the memory space for example we tried to free up the pins and libraries which may be used during the execution of the code.
- Using the header files we tried to control the GPIO pins of the BB-WI and made a complete interface in C programming Language.
- Lastly, we demonstrated the process of interface live and showed that whenever a letter arrives in the letterbox the sensor reads low (being active high in nature) and showed that even if there is a single letter in the letter box the LED will be set high

Task Objective

- To interface the GSM (**G**lobal **S**ystem for **M**obile Communication) with the **B**eaglebone **B**lack **Wi**reless (BB-WI)
- To make the coding done in part I of the interfacing compatible with the objective mentioned above
- The interfacing should be able to send SMS (Short Message Service) when a letter arrives in the letter box.

Task Introduction

- In the previous meeting we saw how we broke the interfacing of sensors and peripherials into various tasks.
- In today's task we try to accomplish the part 2 of the interfacing process i.e. we try to interface the GSM Module (**G**lobal **S**ystem for **M**obile Communication) with Beaglebone Black Wireless.
- This task is intended to complete in 2 stages first one being just interfacing the BB-WI with GSM Module and in the second stage being trying to accommodate the GSM code into part 1 of the interfacing code.
- For this purpose of task we intend to use Arduino based GSM sheild which is having the specification of SIM900. The connection is intended to be completed using the UART (**U**niversal **A**synchronous **R**eciever **T**ransmitter)
- We send the messages using the GSM Module to the desired mobile number which the user choose to recieve the message

Task Requirements: Hardware

- In order to accomplish this one must have the following hardware present with you:
  - BeagleBone Black Wireless (BB-WI)
  - Connecting Wires
  - Wall Adapters
  - GSM Modules
  - Micro USB Cable*
  - Part 1: Sensor and LED**

** Required in case of power not sufficient*

*** Required to demonstrate the Part 1 and Part 2 together*

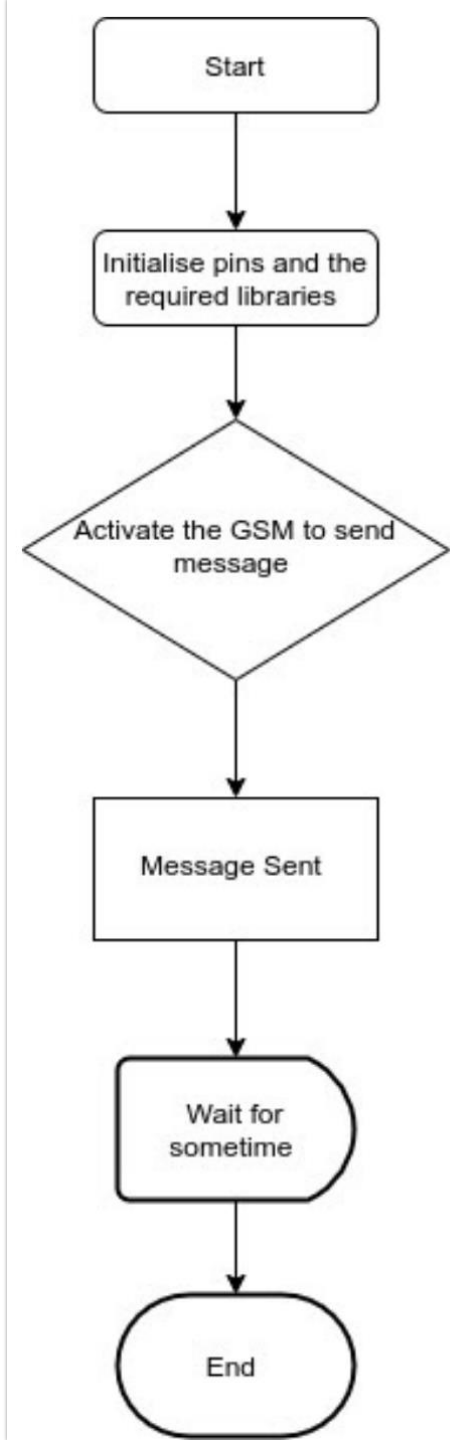
Task Requirements: Software

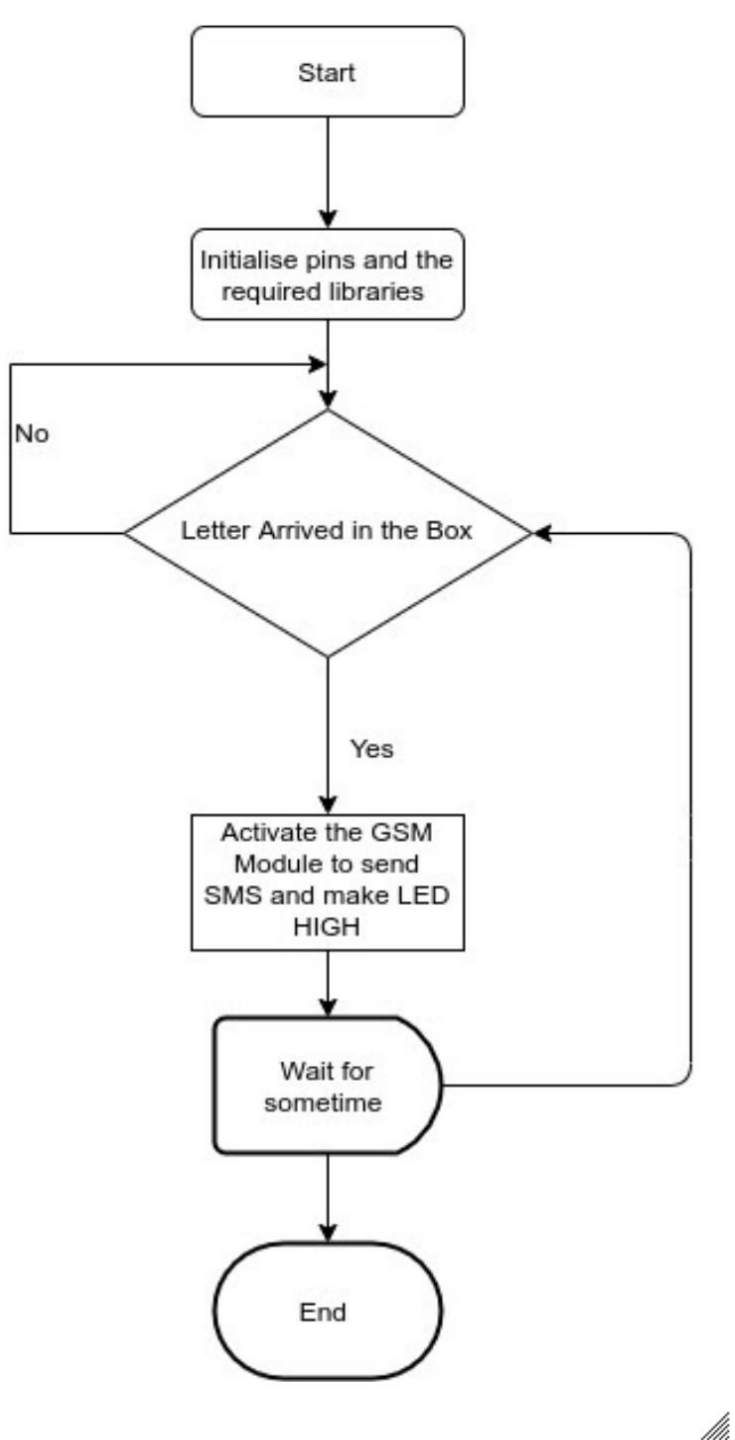
- As for our task objective since we need to program the GSM module to send messages to the desired number via the BeagleBone Black Wireless. For the same reason we need to control the pins thus the pins need to be programmed.
- As mentioned in the proposal and previous meetings we might use Eclipse if the coding cannot be performed on the Beagle bone Wireless itself but since this task can be completed using the nano editor present and the GCC (GNU Compiler Collection)
- To finalize we require the following software tools:
  - Debian OS (Installed on the BB-WI)
  - GCC (GNU [Not Unix] Compiler Collection) to run the code
  - Nano Editor (To write the code)
  - Any required libraries for making interfacing simpler

Getting Ready to Code

- Programming the BB-WI can be very simple as many users over the time has created various headers and function to access the GPIO very easily and hence code them using some simple function name.
- As mentioned in the previous meetings we used some of the user defined header files and their respective functions to gain the control of the GPIO pins and so was the same library files were downloaded and installed on the Beagle bone Black Wireless.
- For the current task we have tried to see if we could get appropriate C library for the GSM Arduino Sheild, but it was difficult to find it. We have discussed this more in detail in the Troubleshooting section.
- Thus, we finally decided to code this task in python and then finally integrated with C programming using appropriate command.

Interface Programming : Flow Chart





Interface Programming: Library Download

- As mentioned, we are going to use some python programming inorder to achieve this task thus we required some libraries such as the pyserial library
- "`pyserial`" library provided support for serial communication over a variety of devices. In our case we want to communicated serial with the BB-WI to control and program the pins.
- Our first step would be to install this library using the "`pip`" (Python Install Packages /Preferred Installer Program) command. By using pip one can install the required packages and libraries. It is a kind of package manager for python language. To install any required library in python using pip we use the command "`pip install some-package-name`"

Interface Programming: Library Download

```
debian@beaglebone:~$ sudo apt update
Get:1 http://deb.debian.org/debian buster InRelease [121 kB]
Get:2 http://repos.rcn-ee.com/debian buster InRelease [3,063 B]
Get:3 http://deb.debian.org/debian buster-updates InRelease [51.9 kB]
Get:4 http://deb.debian.org/debian-security buster/updates InRelease [65.4 kB]
Get:5 http://repos.rcn-ee.com/debian buster/main armhf Packages [1,624 kB]
Get:6 http://deb.debian.org/debian buster/main armhf Packages [7,698 kB]
Get:7 http://deb.debian.org/debian buster/contrib armhf Packages [40.4 kB]
Get:8 http://deb.debian.org/debian buster/non-free armhf Packages [62.0 kB]
Get:9 http://deb.debian.org/debian buster-updates/main armhf Packages [7,844 B]
Get:10 http://deb.debian.org/debian buster-updates/non-free armhf Packages [604
B]
Get:11 http://deb.debian.org/debian-security buster/updates/main armhf Packages
[239 kB]
```

```
debian@beaglebone:~$ pip --version
pip 18.1 from /usr/lib/python2.7/dist-packages/pip (python 2.7)
debian@beaglebone:~$ pip install pyserial
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting pyserial
  Downloading https://files.pythonhosted.org/packages/0d/e4/2a744dd9e3be04a0c090
7414e2a01a7c88bb3915cbe3c8cc06e209f59c30/pyserial-3.4-py2.py3-none-any.whl (193k
B)
    100% |████████████████████████████████| 194kB 399kB/s
Installing collected packages: pyserial
Successfully installed pyserial-3.4
debian@beaglebone:~$ ./message
Message sentdebian@beaglebone:~$ exit
logout
Connection to 192.168.2.44 closed.
```

```
zain@zain-xps-13-7390: ~
File  Edit  View  Search  Terminal  Help
9 packages can be upgraded. Run 'apt list --upgradable' to see them.
debian@beaglebone:~$ sudo apt install python-pip
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  gir1.2-glib-2.0 libgirepository-1.0-1 libjs-sphinxdoc libjs-underscore
  libpython-all-dev python-all python-all-dev python-asn1crypto
  python-cffi-backend python-configparser python-crypto python-cryptography
  python-dbus python-entrypoints python-enum34 python-gi python-ipaddress
  python-keyring python-keyrings.alt python-secretstorage python-setuptools
  python-six python-wheel python-xdg
Suggested packages:
  python-crypto-doc python-cryptography-doc python-cryptography-vectors
  python-dbus-dbg python-dbus-doc python-enum34-doc python-gi-cairo
  gnome-keyring libkf5wallet-bin gir1.2-gnomekeyring-1.0 python-gdata
  python-keyczar python-secretstorage-doc python-setuptools-doc
The following NEW packages will be installed:
  gir1.2-glib-2.0 libgirepository-1.0-1 libjs-sphinxdoc libjs-underscore
  libpython-all-dev python-all python-all-dev python-asn1crypto
  python-cffi-backend python-configparser python-crypto python-cryptography
  python-dbus python-entrypoints python-enum34 python-gi python-ipaddress
  python-keyring python-keyrings.alt python-pip python-secretstorage
  python-setuptools python-six python-wheel python-xdg
```

Interface Programming: Coding

- As last time preparing a flow chart helped us getting the flow of logic in making the program and the same has been done this time. First we tried to get a simple code to make sure that GSM is working through BB-WI next we try to make it fit with part I of the interface.
- As usual in C we import libraries first but in python we import packages and required libraries from package if you which one you want to use. It gives clarity to the reader. In our case we pyserial (Serial Communication), time (Power Management [sleep]) and curses (to indicate end of message)
- Once all libraries are imported we are ready to interface GSM Arduino Sheild with BB-WI and using this code we try to modify the code already completed in Part 1 of Interfacing to make a complete project

Communication Protocol: UART

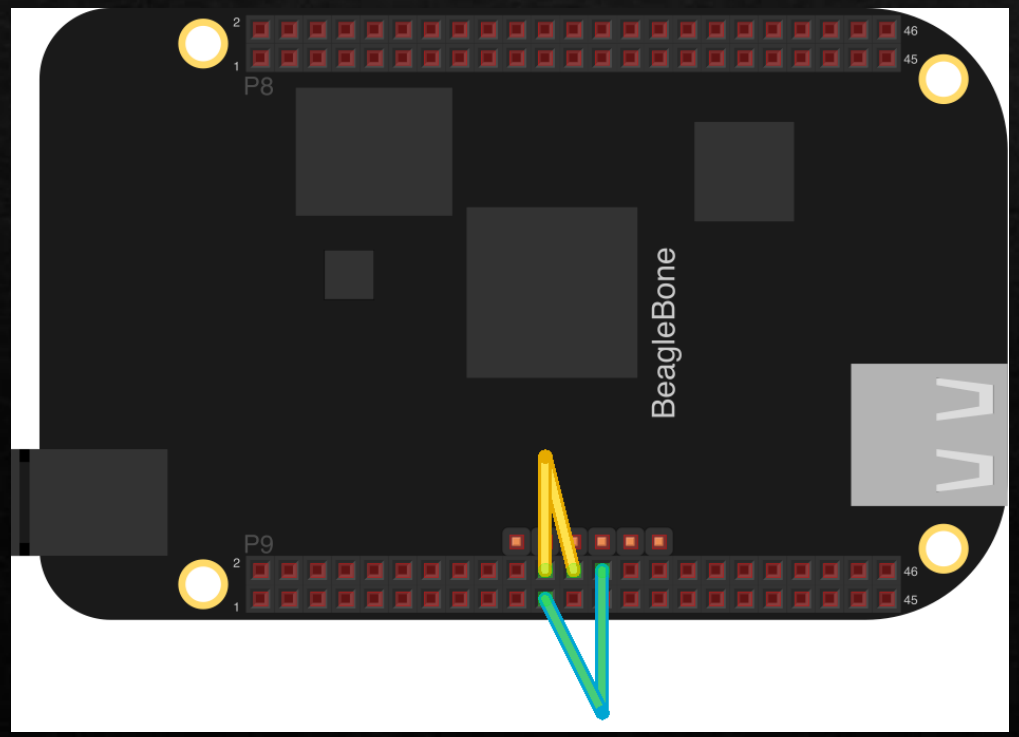
- Before getting started to code we first need to understand the UART protocol and its presence in the BB-WI.
- Firstly UART stands for **U**niversal **A**synchronous **R**eciver **T**ransmitter. This is the most common protocol used for full-duplex serial communication. It is a single LSI (**L**arge **S**cale **I**ntegration) chip designed to perform asynchronous communication. This device sends and receives data from one system to another system.
- Protocols like SPI (**S**erial **P**eripheral **I**nterface) and USB (**U**niversal **S**erial **B**us) are used for fast communication. When the high-speed data transfer is not required UART is used. It is a cheap communication device with a single transmitter/receiver. It requires a single wire for transmitting the data and another wire for receiving.
- If we view the complete details we understand that the block daigram of this protocol is the same at the reciever and transmitter end consisting of logical blocks, shift registers and hold registers with baud rate generator as the interface between the two.

Communication Protocol: UART

- The baud generator decides the rate at which the transmission and reception takes place thus one needs to make sure when programming that the BB-WI and GSM both are set at the same baud rate. This issue also has been discussed later.
- Now let us look at the BB-WI and try to find the UART ports. If we look at the pinout daigram of the BB-WI which is the same as the BBB (BeagleBone Black) we see that we have six UART ports available on the BB-WI.
- On the BeagleBone Black, it's only the /dev/ttyO0 that is enabled by default, and it is coupled to the serial console. The other serial ports must be enabled before they can be used. The newer debian version gives the option to enable it within the programming itself.
- To find the list of UART available on your device one needs to type `ls -l /dev/ttyO*`

Communication Protocol: UART

- We can test the UART Communication if it is working by following method




```
import Adafruit_BBIO.UART as UART
import serial

UART.setup("UART1")

ser = serial.Serial(port = "/dev/ttyO1",
baudrate=9600)
ser.close()
ser.open()
if ser.isOpen():
print "Serial is open!"
ser.write("Hello World!")
ser.close()
```

Communication Protocol: UART

```
root@beaglebone:~# python
Python 2.7.3 (default, May 29 2017,
21:25:00)
[GCC 4.7.3 20130205 (prerelease)] on
linux2
Type "help", "copyright", "credits" or
"license" for more information.
>>> import Adafruit_BBIO.UART as UART
>>> UART.setup("UART1")
>>> UART.setup("UART2")
>>> exit()

#first terminal window:
minicom -b 9600 -D /dev/ttyO1

#second terminal window:
minicom -b 9600 -D /dev/ttyO2
```

	RX	TX	CTS	RTS	Device	Remark
UART0	J1_4	J1_5			/dev/ttyO0	BeagleBone Black only
UART1	P9_26	P9_24	P9_20	P9_19	/dev/ttyO1	
UART2	P9_22	P9_21	P8_37	P8_38	/dev/ttyO2	
UART3		P9_42	P8_36	P8_34	/dev/ttyO3	TX only
UART4	P9_11	P9_13	P8_35	P8_33	/dev/ttyO4	
UART5	P8_38	P8_37	P8_31	P8_32	/dev/ttyO5	

**Note this table will help when writing codes*

Interface Programming: Coding

debian@beaglebone: ~

File Edit View Search Terminal Help

GNU nano 3.2 pysms.py

```
from time import sleep
import serial
from curses import ascii

##set serial
ser = serial.Serial()

##Set port connection to USB port GSM modem
ser.port = '/dev/ttyO1'

## set older phones to a baudrate of 9600 and new phones and 3G modems to 115200
## ser.baudrate = 9600
ser.baudrate = 115200
ser.timeout = 1
ser.open()
def sendsms(number,text):
    ser.write('AT+CMGF=1\r\n')
    sleep(2)
    ser.write('AT+CMGS="%s"\r\n' % number)
    sleep(2)
    ser.write('%s' % text)
    sleep(2)
    ser.write(ascii.ctrl('z'))


sendsms('+16477868334','Letter Arrived in the Box')
```

^G Get Help ^O Write Out ^W Where Is ^K Cut Text ^J Justify ^C Cur Pos
^X Exit ^R Read File ^\ Replace ^U Uncut Text ^T To Spell ^_ Go To Line

debian@beag

File Edit View Search Terminal Help

GNU nano 3.2 callpy

```
#include<stdio.h>
#include<stdlib.h>

int main()
{
    system("/usr/bin/python2.7 pysms.py");
    printf("Message sent\n");
    return(0);
}
```

** Left side python code for GSM and right-side integration of python in C language*

Wire Interface: GSM Module

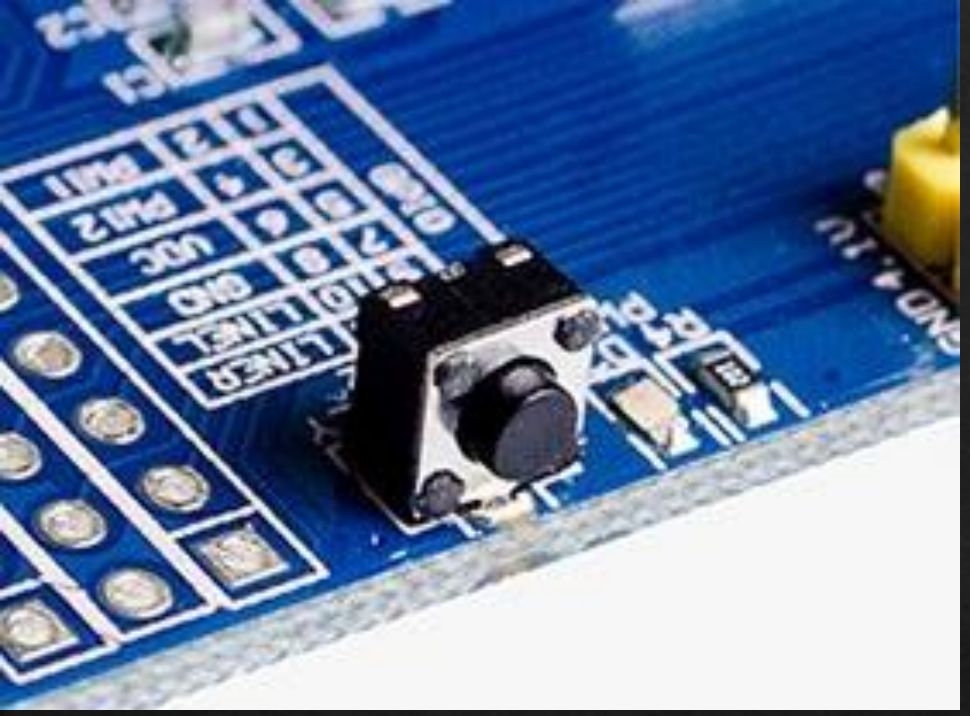
- Having a knowledge about how the UART protocol and making our program ready we now must make the physical (hardware) connection before we dump the code on the BB-WI to test as to if the interface is working properly.
- To make the connections we will take the help of schematic captures that we made in previous task (meetings). We see that we have used the UART4 ports of the BB-WI.
- A point to make here is to make sure that we connect the TX (BB-WI) with the RX (SIM900 GSM Module UART Side)
- The GSM module used can sending/receive text and call using the antenna attached to it. If one wants to monitor time, then it can do so using the RTC present by simply connecting a CR1220 battery at the back of the shield.
- It performs the various using the AT (Attention) Commands.

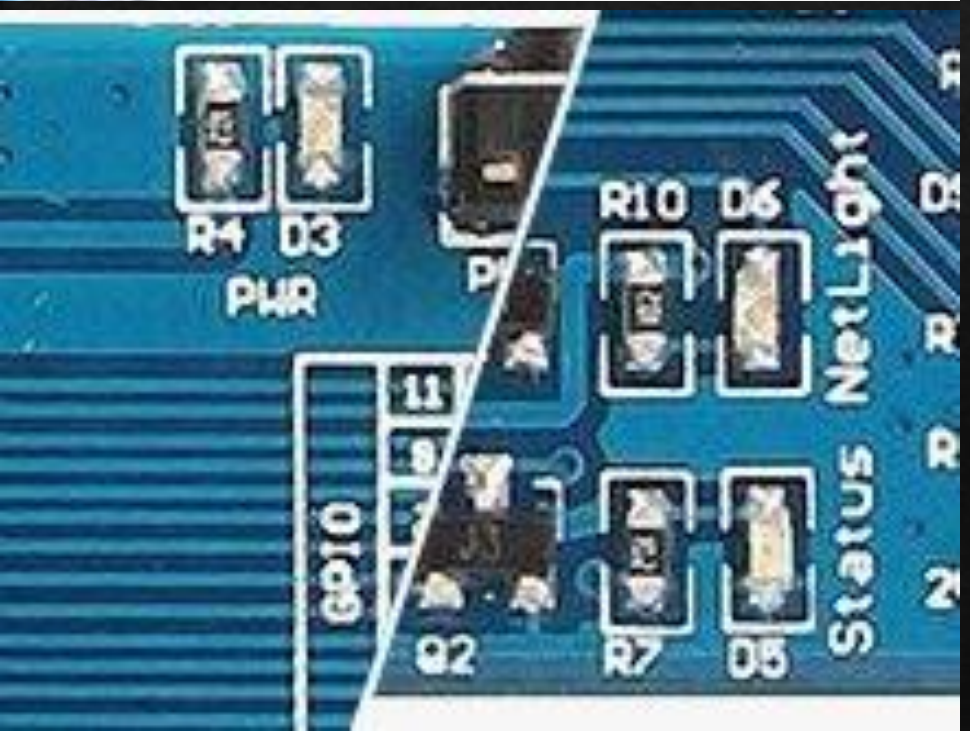
AT Commands: Brief Introduction

- AT commands are instruction used to control a modem. Every command starts with "AT" or "at" this is the reason why modem commands are called AT commands.
- Along with commands desired for the dial up modem the commands also support the GSM/GPRS modems and cell phones which include SMS related commands like AT+CMGS (Send SMS Message), AT+CMSS (Send Message from Storage), AT+CMGL (List SMS Messages) and AT+CMGR (Read SMS Messages).
- The "AT" at the start of the command informs the modem about the start of the command line. It is not a part of the command. All the GSM commands are extended commands as they carry a "+" sign ahead of them.

AT Commands: Brief Introduction

- For our task purposes we only need to send SMS to the desired number
- Some of the other task that can be performed by the GSM/GPRS modem or Cell phone include:
  - Get information about the phone/ modem (+CGMI, +CGMM, +CGSN, +CGMR)
  - Get information about the subscriber like IMSI number (+CIMI)
  - Get current status
  - Send and receive fax
  - SMS service
  - Read, write and search phonebook entries and many more





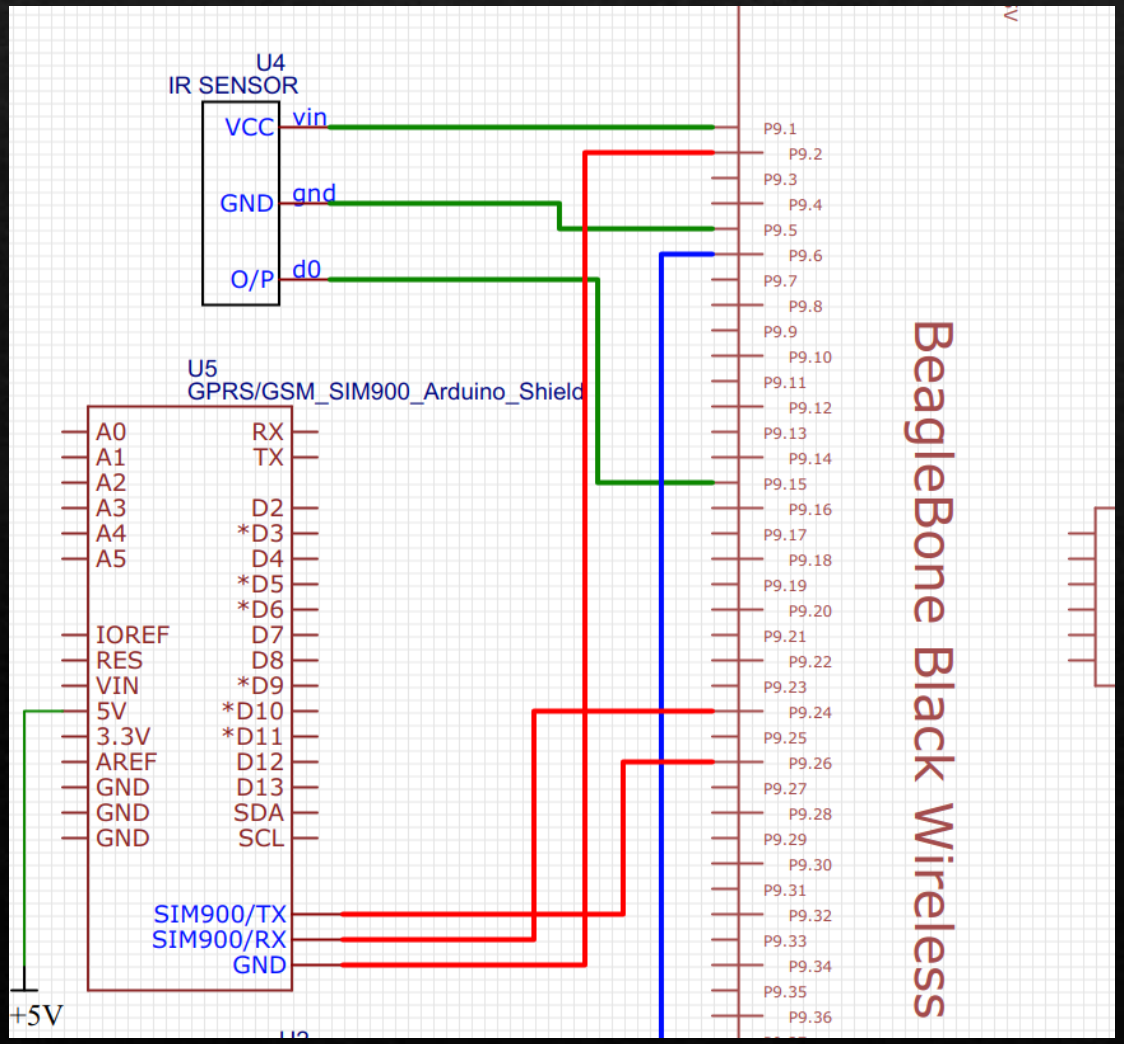
Wire Interface: GSM Module

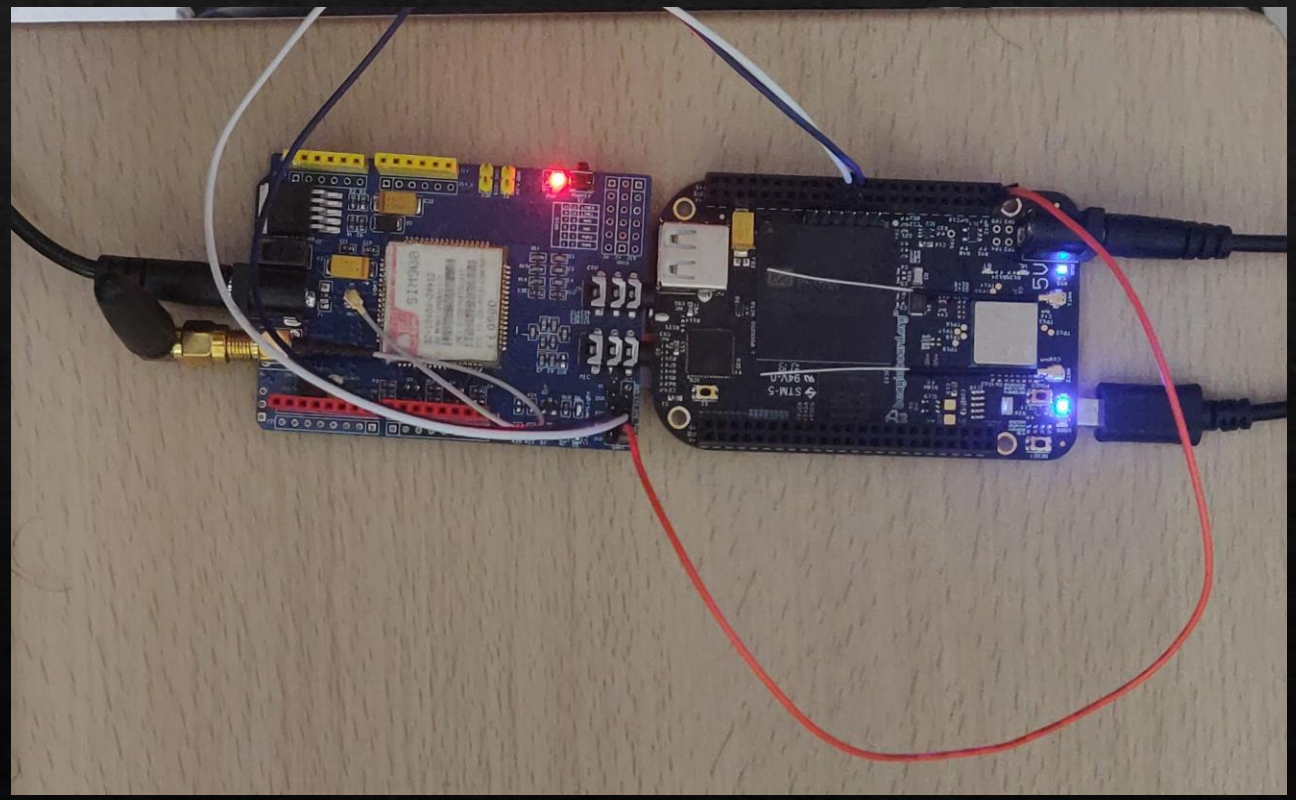
- Once we have the knowledge about the internal working of the GSM now, we can start the connection to the GSM module. Before that lets understand how to get the GSM module itself working.
- Firstly we need to power up the GSM Module using an 5 V adapter (Barrel jack). Next, we need to check for the LED Status. If the no status LED are on then we need to hold and press the perpendicular power key button for approx. 2 seconds

Wire Interface: GSM Module

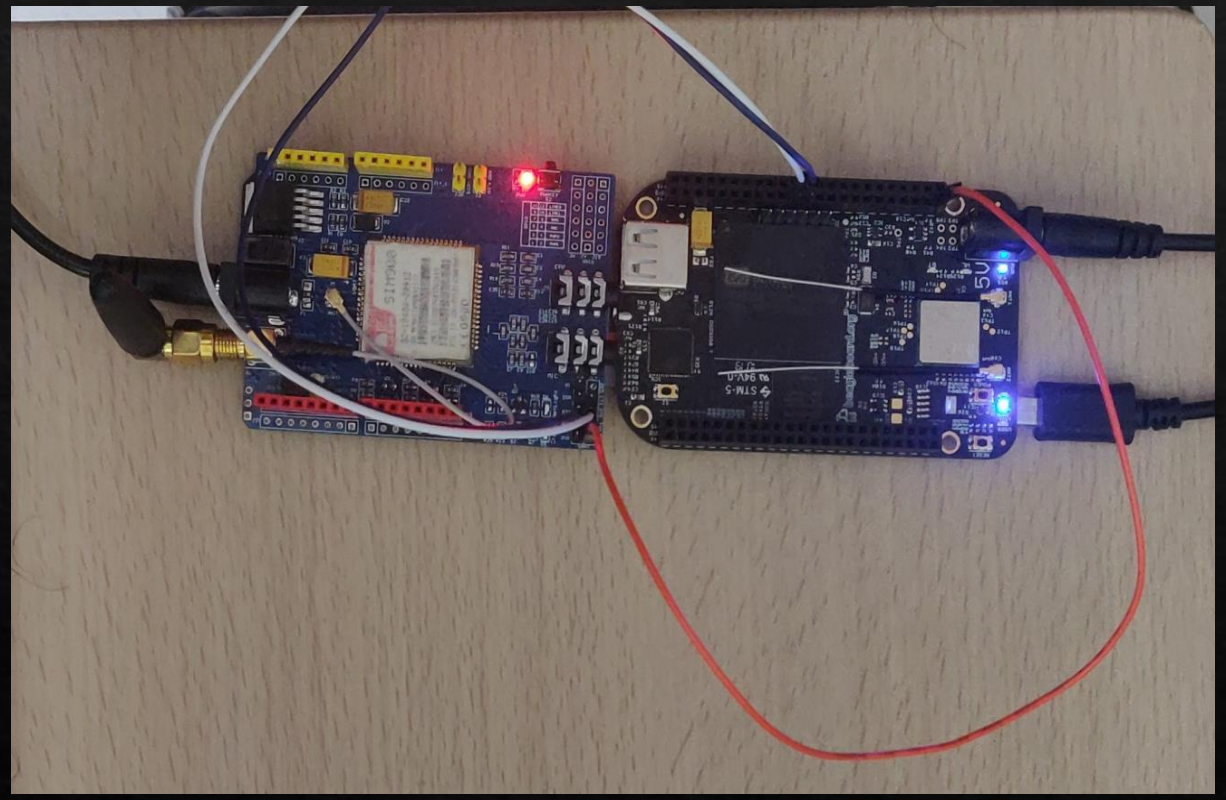
- PWR: This LED is connected to the shield's power supply line. If this LED is on, the shield is receiving power.
- Status: This LED indicates SIM900's working status. If this LED is on, the chip is in working mode.
- Netlight: This LED indicates the status of your cellular network. It'll blink at various rates to show what state it's in.
  - off: The SIM900 chip is not running
  - 64ms on, 800ms off: The SIM900 chip is running but not registered to the cellular network yet.
  - 64ms on, 3 seconds off: The SIM900 chip is registered to the cellular network & can send/receive voice and SMS.
  - 64ms on, 300ms off: The GPRS data connection you requested is active.
- When supplying power externally through an adapter make sure that you have made the switch on the external mode.

Wire Interface: GSM Module

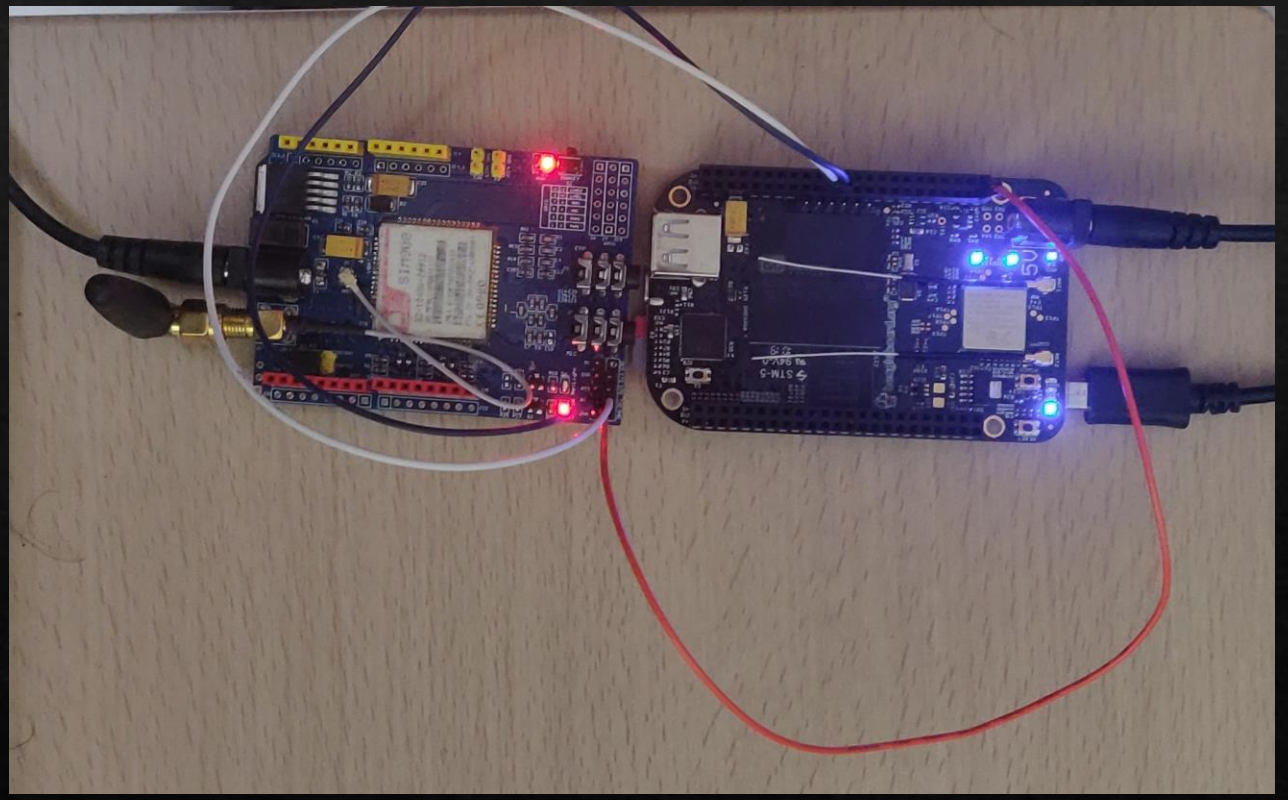




Execution

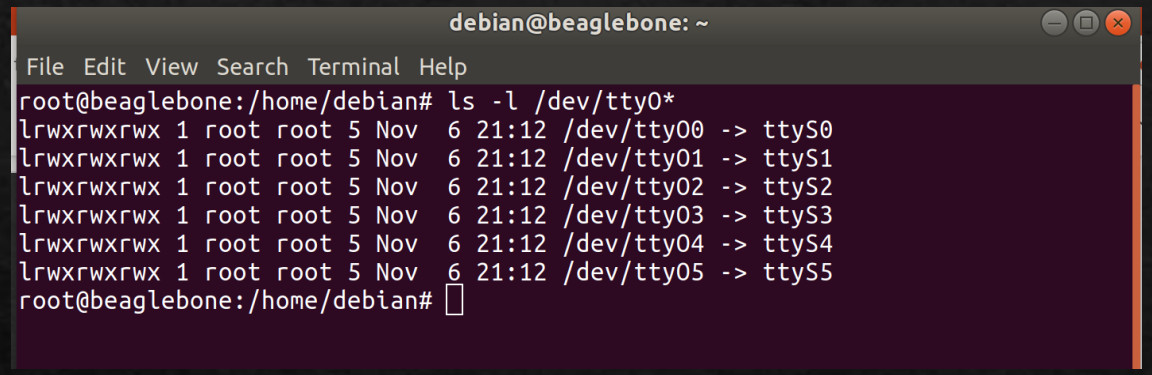


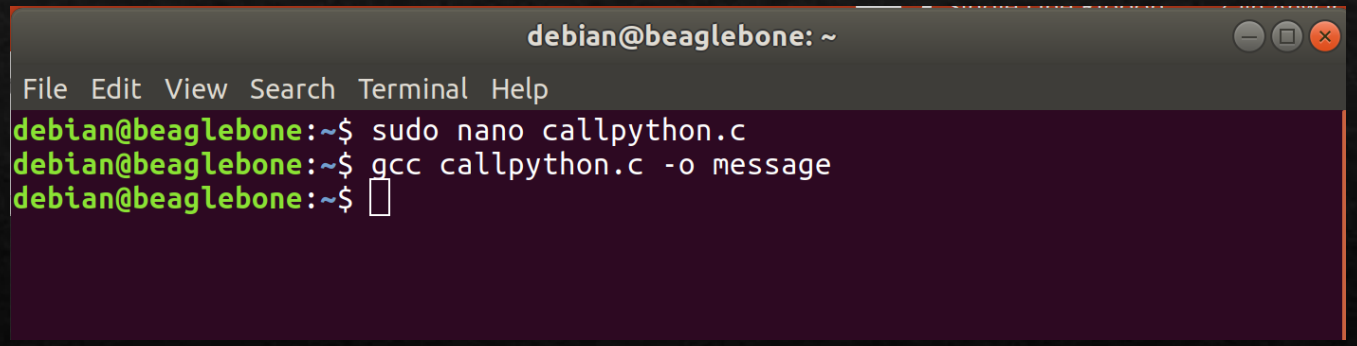
Initial Setup of GSM Module with BB-WI

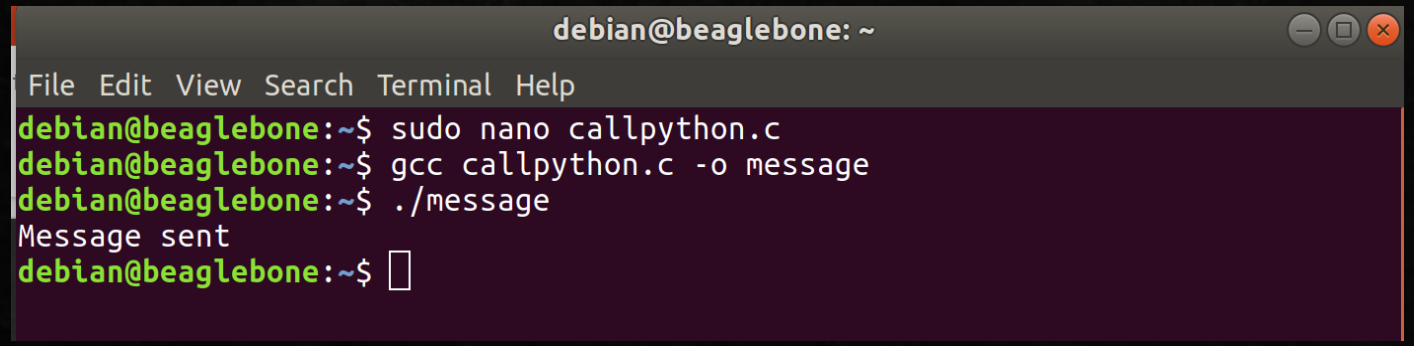


Setup of GSM Module when the it is connected to the network and working perfectly fine along with BB-WI

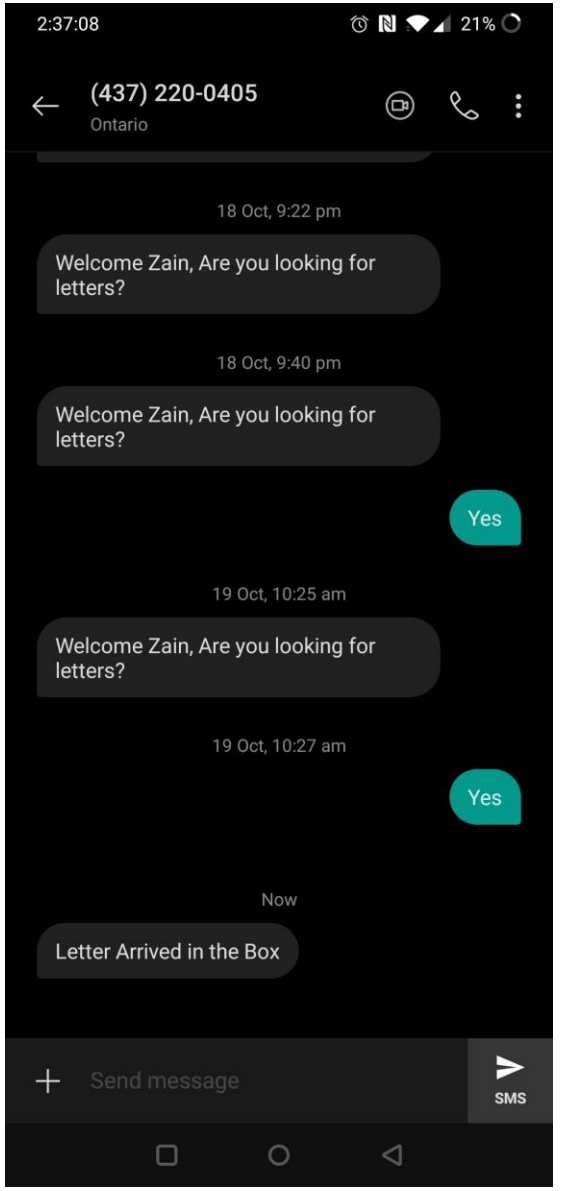
Execution


debian@beaglebone: ~
File Edit View Search Terminal Help
root@beaglebone:/home/debian# ls -l /dev/ttyO*
lrwxrwxrwx 1 root root 5 Nov 6 21:12 /dev/ttyO0 -> ttyS0
lrwxrwxrwx 1 root root 5 Nov 6 21:12 /dev/ttyO1 -> ttyS1
lrwxrwxrwx 1 root root 5 Nov 6 21:12 /dev/ttyO2 -> ttyS2
lrwxrwxrwx 1 root root 5 Nov 6 21:12 /dev/ttyO3 -> ttyS3
lrwxrwxrwx 1 root root 5 Nov 6 21:12 /dev/ttyO4 -> ttyS4
lrwxrwxrwx 1 root root 5 Nov 6 21:12 /dev/ttyO5 -> ttyS5
root@beaglebone:/home/debian#


debian@beaglebone: ~
File Edit View Search Terminal Help
debian@beaglebone:~$ sudo nano callpython.c
debian@beaglebone:~$ gcc callpython.c -o message
debian@beaglebone:~$


debian@beaglebone: ~
File Edit View Search Terminal Help
debian@beaglebone:~$ sudo nano callpython.c
debian@beaglebone:~$ gcc callpython.c -o message
debian@beaglebone:~$ ./message
Message sent
debian@beaglebone:~$

Execution


2:37:08
21%
(437) 220-0405
Ontario
18 Oct, 9:22 pm
Welcome Zain, Are you looking for letters?
18 Oct, 9:40 pm
Welcome Zain, Are you looking for letters?
Yes
19 Oct, 10:25 am
Welcome Zain, Are you looking for letters?
19 Oct, 10:27 am
Yes
Now
Letter Arrived in the Box
Send message
SMS

Troubleshooting

- As in the previous task we faced some difficulties the same was faced during this task which we shall discuss here.
- During the interfacing we connected the trasmitter pin to the transmitter and reciever pin to the reciever thus it couldn’t send message and by changing the connection suitably we were able to get it working
- While coding the module we couldn't find a suitable C library thus we had to deviate to python programming which was fast and easy to use and code. But we had to install appropriate libraries and set the baud rates which was not known thus later found that baud rate for mordern phone was 115200 and other phones with 3G and modem work at baud rate of 9600.

Troubleshooting

- Since the entire code would be written in C for the project thus, we had to find a way to integrate it into the C language. This was resolved using the system() command which is part of the stdlib.h header file.
- The GSM module still possess an issue at times as it causes problem though the program executes perfectly and displays a message sent output. But the output is not seen on the screen at times. This may cause a problem during the complete demonstration of the project.

Conclusion

- From the task prespective we could see that we are able to send messages to the mobile user when a letter arrives via the GSM module successfully
- The drawbacks of the system include that one cannot send what is the count of the letters in box via text message as the code that sends message is in python programming and counting letter code is in the C programming language
- Also GSM may sometimes may not catch or connect to the network also in the era of Cloud storage the GSM may soon be seen to vanish. This is a concern even while demonstration of this project.

References

(n.d.). Retrieved November 09, 2020, from https://www.tutorialspoint.com/system-function-in-c-cplusplus

Cooper, J. (n.d.). Setting up IO Python Library on BeagleBone Black. Retrieved November 09, 2020, from https://learn.adafruit.com/setting-up-io-python-library-on-beaglebone-black/uart

Curses Programming with Python¶. (n.d.). Retrieved November 09, 2020, from https://docs.python.org/3/howto/curses.html

In-Depth: Send Receive SMS & Call with SIM900 GSM Shield & Arduino. (2019, December 13). Retrieved November 09, 2020, from https://lastminuteengineers.com/sim900-gsm-shield-arduino-tutorial/

Molloy, D. (2019). *Exploring BeagleBone: Tools and techniques for building with embedded Linux*. Indiapolis, Indiana: Wiley.

Pringle-Wood, C. (n.d.). Colins44/gsm-sms. Retrieved November 09, 2020, from https://github.com/colins44/gsm-sms

Python time sleep() Method. (n.d.). Retrieved November 09, 2020, from https://www.tutorialspoint.com/python/time_sleep.htm

Serial ports / UART. (n.d.). Retrieved November 09, 2020, from http://beaglebone.cameon.net/home/serial-ports-uart

UART Communication Protocol - How it works? (2020, June 16). Retrieved November 09, 2020, from https://www.codrey.com/embedded-systems/uart-serial-communication-rs232/