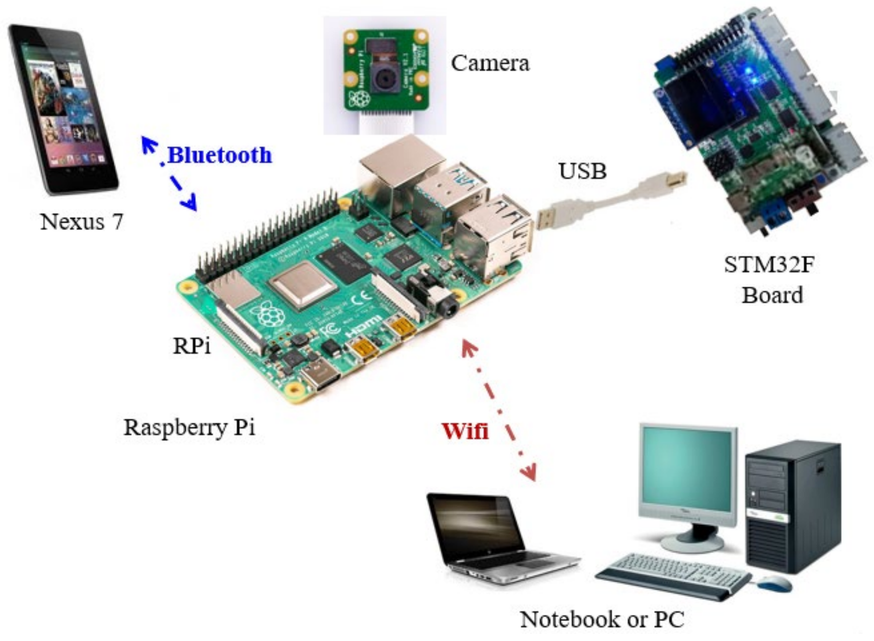


Figure 1: MDP System modules

Acknowledgement

This set of notes is adapted from the rpi setup guide created by A/P Nicholas Vun.

System Overview

The modules used in the MDP is as shown in Figure 1 above.

The Raspberry Pi runs the Raspberry Pi operating system[1] and forms the main platform of the system, interfacing with the rest of the components as shown in figure above. The RPI version used maybe RPI3 or RPI4 depending on the hardware availability.

The RPI will be setup as a wireless access point (AP) such that other computing devices can interface with it through Wifi link. However, during system development, its wired Ethernet connection may be connected to the LAN network point in the Lab, to provide a gateway to the Internet for the other computing devices.

RPI's interface with the Android Tablet will be through Bluetooth connection using the rfcomm protocol. RPI interface with the microcontroller-based board will be through the USB cable using the UART over USB serial interface (ttyACMx/ttyUSBx). The Android tablet used (as of writing) is the Samsung Galaxy A7 Lite and microcontroller board used is the STM32F processor board with on board motor drivers.

For initial development work, such as to configure the network interface as well as downloading software packages, the RPI can be connected to a local monitor[2] and keyboard through its USB ports such that it can be operated as a standalone computing platform. However, it will eventually be configured to be accessed remotely through network (either Wifi or Ethernet) using Secure Shell (SSH) remote login.

Note: Each project group will be assigned a PC in the Lab for the MDP. Students are advised to save their work in their own USB drive. The PC environment in the lab is virtualized and all work saved in the PC will be lost after a system reboot or when you logout from the PC.

To save the files you create on RPI, you can install the Samba Server on the RPI, which allow you to transfer the files over the network and store them on your own PC or thumb drive. You can also use VNC Viewer to perform file transfers. **Note that you will need to enable VNC in the RPI configuration to allow PC VNC viewer apps to communicate with RPI**.

Figure 2 below shows the main components and interfaces on the Raspberry Pi 4 development board provided for this project. Its main processor is based on a 64-bit quad-core ARM Cortex-A72 CPU. RPI board was originally developed for teaching/learning computer programming purpose but has since been widely used in many practical embedded applications.

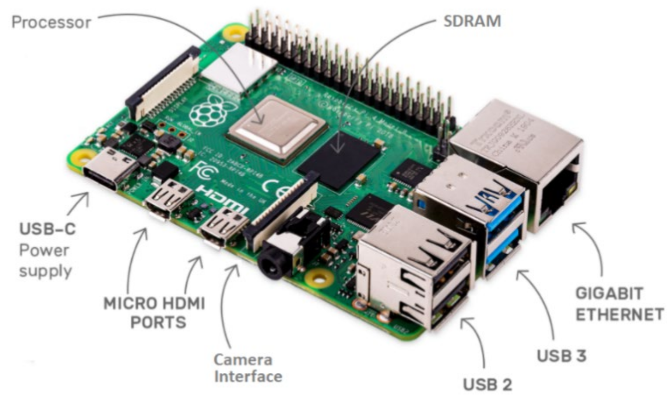


Figure 2: RPI4

Raspberry Pi OS (or Raspbian if you are using earlier version)
The RPI operating system used in RPI is based on the Debian Linux. Although it comes with a GUI, it is not very convenient for development work. Hence you will instead frequently use the Linux text command-based interface (remotely through SSH). As such, you will need to be familiar with some of the commonly use Linux commands (e.g. sudo, apt_get install, chmod etc.) as well as one of the command line text editor (e.g. nano or vi) when using the RPI.
Note that the installation steps in this guide is verified with the Raspberry Pi OS (version 10), also known as the Buster. The latest Raspberry Pi OS is version 11 (Bullseye).

Local monitor
The RPI will need to be connected to the Internet most of the time during the development work. However, before the Wifi AP is up and running, RPI will be assigned a dynamic IP address by the DHCP server used in the SCSE Lab's LAN. However, we need to know the IP address to use the remote SSH through the network. Hence it is likely that you will need to connect the local monitor to observe the IP allocated by the DHCP server during the bootup.

IP address
If the IP address is not shown in the bootup message, you could enable it through the **/etc/rc.local** script file (which may be already enabled by default). However, sometime the DHCP server is too slow to issue the IP address, before the RPI completes the bootup, and hence will not be able to show the IP address allocated. For such case, you will need the local keyboard and monitor to login and check by using the **ifconfig eth0** command.

Wireless Access Point and Internet gateway
The most convenient setup is to not connect the local monitor (and keyboard) but be able to SSH RPI through the network. For this, we need a fixed IP address to access RPI. This can be done by setting up the RPI as a **Wifi Access Point** which uses a **fixed static IP** address. We can then use IP forwarding to send the IP packets to its wired Ethernet, which is connected to the LAN (which in turn has a dynamic IP address allocated by NTU network) and form a gateway to the Internet.

Power Supply in earlier version of RPI (RPI3 and below)
Earlier version of RPI power all devices through its USB port, which is in turn itself powered through its Micro USB connector. However, RPI contains an onboard polyfuse whose resistance will increase as it gets hot. This is used to protect the RPI by limiting the total current allowable (about 750mA maximum) for the RPI. Beyond this current level, the fuse's resistance will increase, and cause the supply voltage to drop below the operating voltage of the devices connected to the RPI. This will sometime cause seemingly 'strange' intermittence problems, whereby the device may stop operating although it is OK initially. As such, check the voltage level of the RPI if you encounter intermittence problems during system operation that seems to recover by itself.

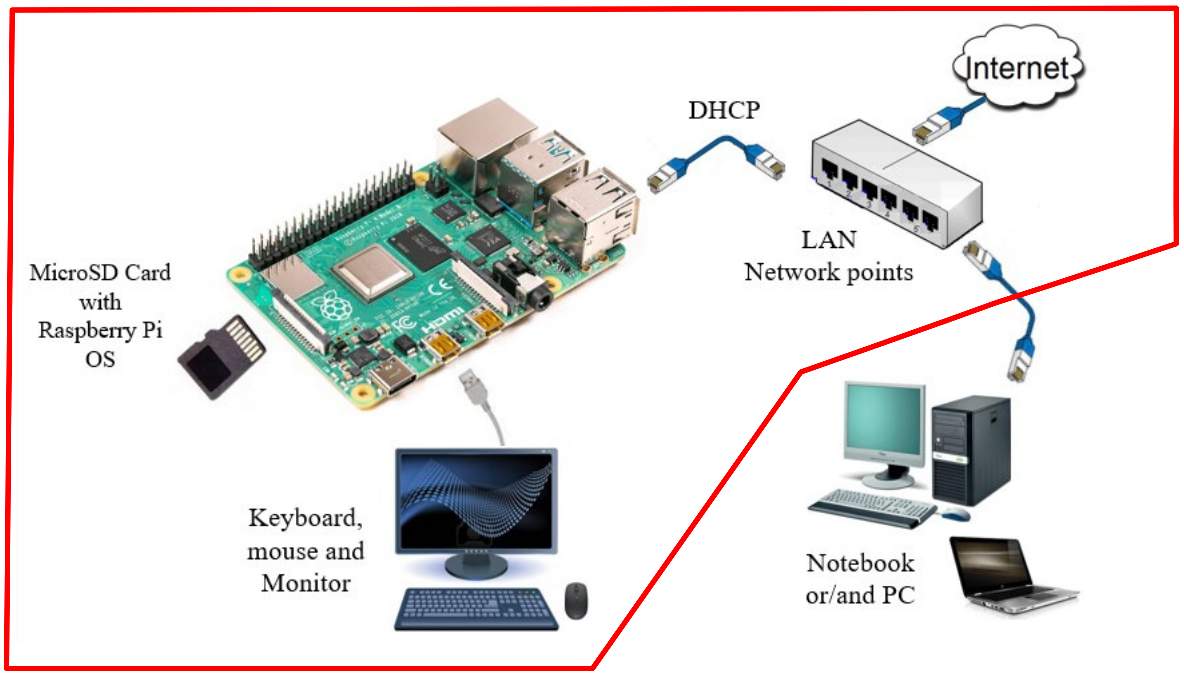
Speed Throttling
The operating system will throttle the CPU speed if processor temperature increase beyond a certain threshold, exact temperature depends on the RPI hardware and the OS version. So do keep the RPI cool, but please don't ice it or pour water over it 😊

The main **content** of this document is summarized as follows:

A. Setting up the RPI with Raspberry Pi OS
B. Remote access RPI using SSH
C. Setting up the RPI as a Wifi access point, and a gateway to the Internet via forwarding the Wlan0 traffic to Eth0 and vice versa
D. Setting up the Bluetooth on RPI and connecting to Android via BT SPP
E. Information on the USB interface between RPI and STM32F Board
F. Other useful packages to have on RPI
    1. Apache server to provide web-based GUI
    2. Samba server to transfer file over network

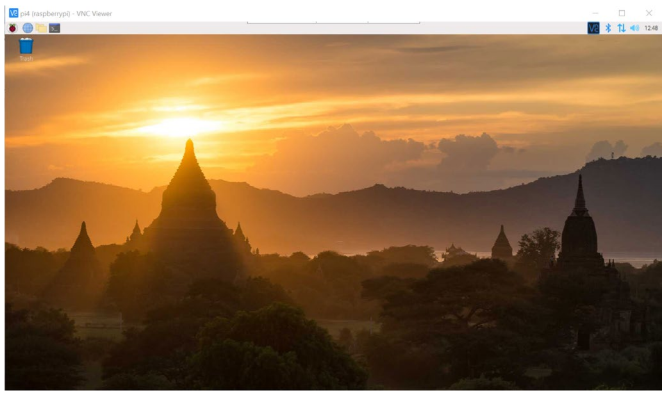
A. Booting up the RPI

To begin with, the RPI should be connected to a monitor, keyboard and mouse, as well as the network point in the Lab for internet access as shown below.

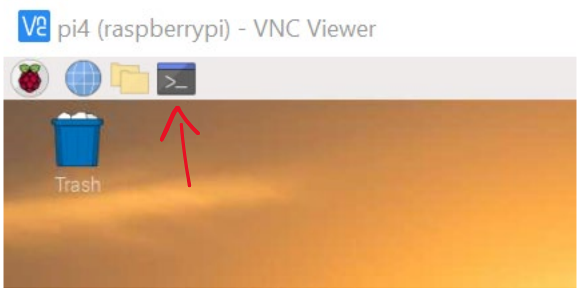

Internet
DHCP
LAN
Network points
MicroSD Card with Raspberry Pi OS
Keyboard, mouse and Monitor
Notebook or/and PC

(i) It is likely that the Micro SD card already contains an OS (e.g. installed by previous batch of MDP students), but it **should be re-formatted** and install with a new copy of the Raspberry Pi OS - See instruction in Appendix).

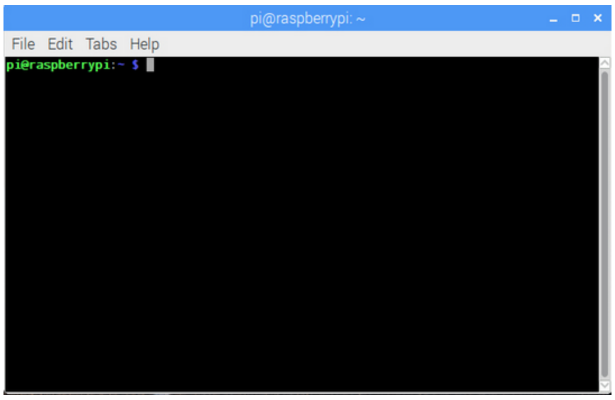
(ii) Plug the Micro SD card into RPI (beneath the board), apply power to the RPI through the USB-C connector. The RPI should boot up and displays series messages on the local monitor. Eventually it will boot into a GUI screen below. The screen below is the default wallpaper for Buster.



(iii) However, you will find that many time it will be more convenient to use the RPI in the Command Line terminal mode (also known as console) for development work. To do so, click the black icon showing a console on the top left-hand corner of the display.



This will open a **terminal** window – also known as the command line **console**, waiting for user to enter the necessary command.

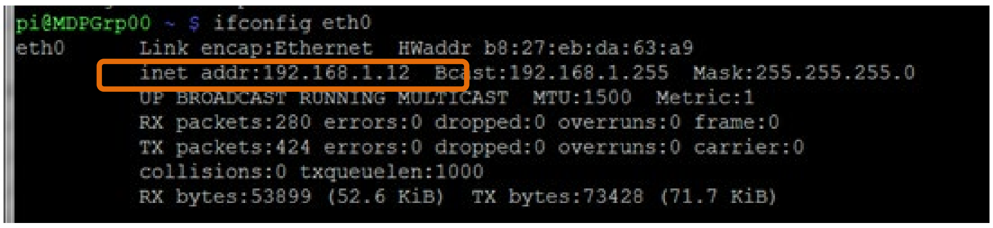


```
pi@raspberrypi:~ $
```

B. Remote SSH client

(i) The first thing to check once the RPI is operating is to make sure that it has been allocated an IP address through the wired Ethernet connection to the LAN in the Lab the LAN in the Lab.

In the command line console, types the Linux command: **hostname -I** or **ifconfig eth0** to check whether an IP address has been assigned to the RPI. (It should be if the connection to the LAN network point is done properly.)



```
pi@MDPGrp00 ~ $ ifconfig eth0
eth0      Link encap:Ethernet  HWaddr b8:27:eb:da:63:a9
          inet addr:192.168.1.12  Bcast:192.168.1.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:280 errors:0 dropped:0 overruns:0 frame:0
          TX packets:424 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:53899 (52.6 KiB)  TX bytes:73428 (71.7 KiB)
```

This IP address is dynamically assigned by the NTU DHCP server through the LAN in the Lab. As this is a dynamically assigned IP, you would need to recheck this every time you reboot the RPI, as it is probably changed every time you reboot RPI.

You can also check whether your RPI has connection to the Internet by accessing the web browser.

Note: in version STRETCH of RPI OS (version 9), the Ethernet port Network interface may appear with name formed from a prefix en followed by x (indicating MAC) and its MAC address e.g. enxb827eb123456. This fortunately reverted to the usual WLAN0, ETH0 etc from Buster (version 10) onwards.

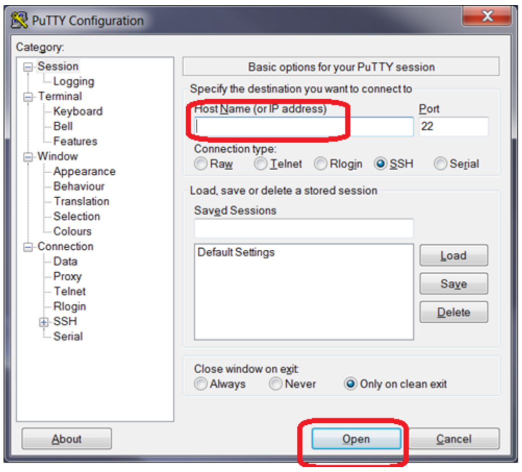
Important notes before proceeding:

- NTU server seemed to be allocating RPI to a different subnet these days. If RPI and your PC is in a different subnet i.e. the first three sets of digits are different XXX.YYY.ZZZ.NNN, then your SSH client may not be able to see RPI's IP and SSH will fail.
- Please skip B.(ii)-(iv), complete (v) and proceed to setup the Wifi AP in Section C.
- When your PC is connected to RPI's wifi AP, your PC will be able to SSH to RPI as they will be in the same subnet (Steps B.(ii)-(iv)).
- If you are doing steps B.(ii)-(iv) at home with your home router, the SSH procedure will work since your RPI and your PC will be in the same subnet.

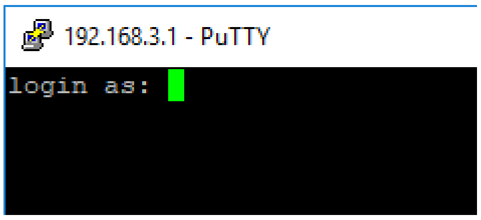
(ii) Once the **eth0's IP address** is known (value will depend on the LAN's DHCP setup), you can then use another computer on the network to remotely access the RPI, using a SSH Client program. For MDP, the program PuTTY should already be installed on the Windows PC in the Lab.

It is likely that you need to enable the SSH server on the RPI - latest RPI OS has SSH disabled by default. To do so, on the RPI board, execute the command **sudo raspi-config** and select **Interface Options** and select Enable SSH server. You can enable the **VNC** and **Camera** option at the same time.

On the networked computer, execute the PuTTY program and configure it with the RPI's IP address. Click Open. A console screen will appear on the remote PC . Log in using the default username and password (i.e. **pi** and **raspberry**). Or the password you modified during the OS installation. You can save the session to facilitate future access.


PuTTY Configuration

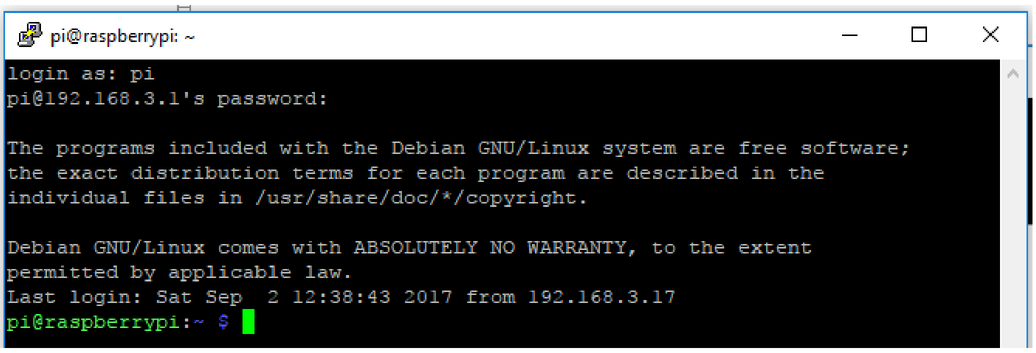
Basic options for your PuTTY session
Specify the destination you want to connect to
Host Name (or IP address):
Port: 22
Connection type: Raw Telnet Rlogin [✓ SSH] Serial
Load, save or delete a stored session
Saved Sessions: Default Settings
Close window on exit: Always Never [✓ Only on clean exit]


192.168.3.1 - PuTTY

```
login as:
```

(iii) You should now have remote network access to the RPI through the LAN using the desktop PC, which means the local monitor and keyboard is not necessary anymore. Note that multiple SSH sessions can be initiated for multiple users to log into the RPI simultaneously, if required.

(iv)



```
pi@raspberrypi: ~
login as: pi
pi@192.168.3.1's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Sep  2 12:38:43 2017 from 192.168.3.17
pi@raspberrypi:~ $
```

Note: While the local keyboard/monitor/mouse is now not essential once remote SSH is established, you will still need to have the local monitor connected during the following setups so that you can observe the bootup messages to check the dynamically assigned IP address, and find potential causes of problems encountered. The keyboard and monitor can be disconnected once the Wifi access point and internet gateway is successfully setup.

From this point onward, we will be using the remote computer to complete the procedures for the rest of this guide.

(v) Before we proceed further, let's prepare the system for the subsequent configurations. First execute the following commands:

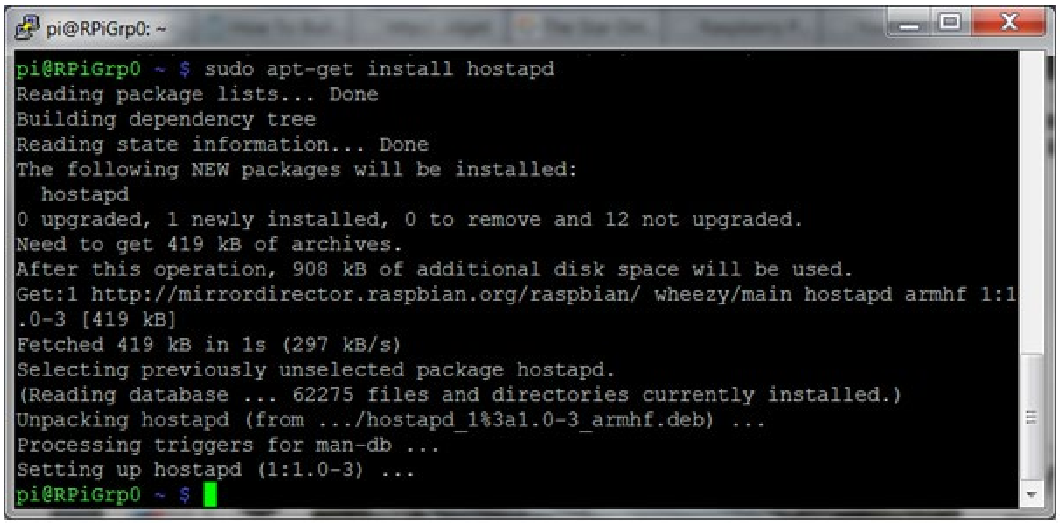
sudo apt-get update
sudo apt-get upgrade

This is to update and upgrade the program package lists (from a server on the internet) such that we will always install the most recent version of the program when using the **apt-get** command. Reboot the RPI after getting the upgrade. Note that you may have already done this during the OS installation.

Next, we will download and install two programs ('packages' in Linux) that are to be used later in the setup, by issuing the following Linux commands in the SSH client console.

(a) First is the Host Access Point package:

sudo apt-get install hostapd



```
pi@RPiGrp0: ~
pi@RPiGrp0 ~ $ sudo apt-get install hostapd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  hostapd
0 upgraded, 1 newly installed, 0 to remove and 12 not upgraded.
Need to get 419 kB of archives.
After this operation, 908 kB of additional disk space will be used.
Get:1 http://mirrordirector.raspbian.org/raspbian/ wheezy/main hostapd armhf 1:1
.0-3 [419 kB]
Fetched 419 kB in 1s (297 kB/s)
Selecting previously unselected package hostapd.
(Reading database ... 62275 files and directories currently installed.)
Unpacking hostapd (from .../hostapd_1%3a1.0-3_armhf.deb) ...
Processing triggers for man-db ...
Setting up hostapd (1:1.0-3) ...
pi@RPiGrp0 ~ $
```

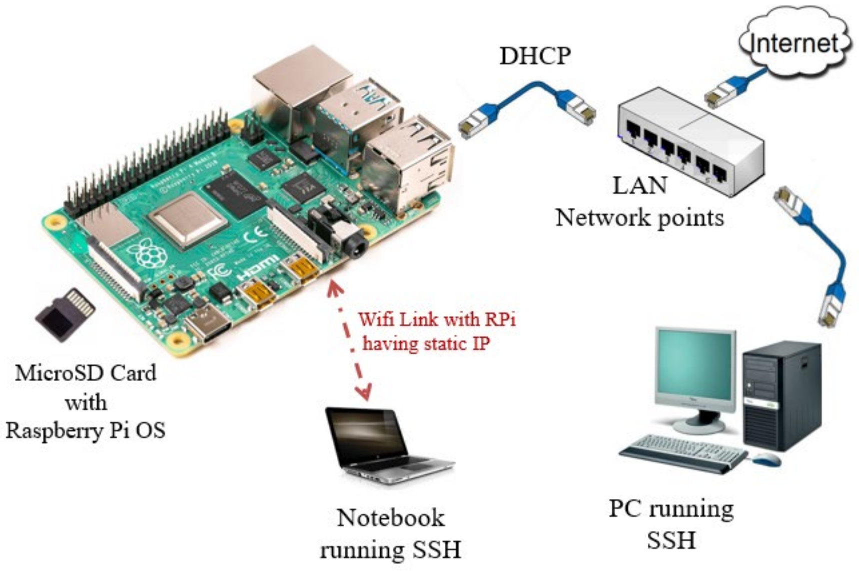
(b) Next is the DHCP package:

```
sudo apt-get install dnsmasq
```

Although these two packages are now installed in the RPI system, they need to be further configured and enabled before they can be activated in the system.

C. Setting up of the RPI as a Wireless Access Point with gateway to the Internet

In the MDP proposed system, the RPI is to remotely accessible through a Wifi link as well as a Bluetooth link. The section describes how the RPI can be setup as a wireless access point i.e. a Wifi hotspot. To do so, several programs need to be further installed on RPI, follows by the proper network configuration.


DHCP
Internet
LAN
Network points
Wifi Link with RPi having static IP
MicroSD Card with Raspberry Pi OS
Notebook running SSH
PC running SSH

C.1 – Setting up hostapd

The main program needed for setting up the RPI as a wireless access point is **hostapd** , which has been downloaded and installed earlier. This is a user space program, as opposed to kernel space program in Linux, that will run in the background (called **daemon**) and provide the wireless host access point service. You should google to find out more detail about this program.

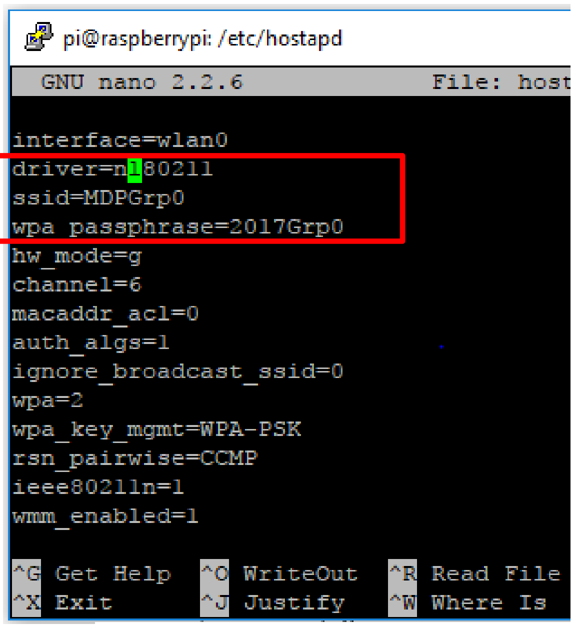
(i) This program can be easily downloaded and installed automatically, with the RPI connected to the internet through eth0, by simply issuing the following Linux command in the SSH client console:

```
sudo apt-get install hostapd
```

(ii) Once the hostapd is installed, a **hostapd** binary file would be created in a /usr/sbin/ subdirectory as follows: **/usr/sbin/hostapd** , together with other necessary files, such as those found in the subdirectory **/etc/hostapd/** **.**

(iii) Next create a (new) configuration file **hostapd.conf** to configure hostapd. You can use the Linux command line text editor program, **nano** to edit it, using the command as shown below. Google 'Linux nano' for detail of the program usage.

```
sudo nano /etc/hostapd/hostapd.conf
```



```
pi@raspberrypi: /etc/hostapd
  GNU nano 2.2.6                File: host

interface=wlan0
driver=nl80211
ssid=MDPGrp0
wpa passphrase=2017Grp0
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_key_mgmt=WPA-PSK
rsn_pairwise=CCMP
ieee80211n=1
wmm_enabled=1

^G Get Help  ^O WriteOut  ^R Read File
^X Exit      ^J Justify   ^W Where Is
```

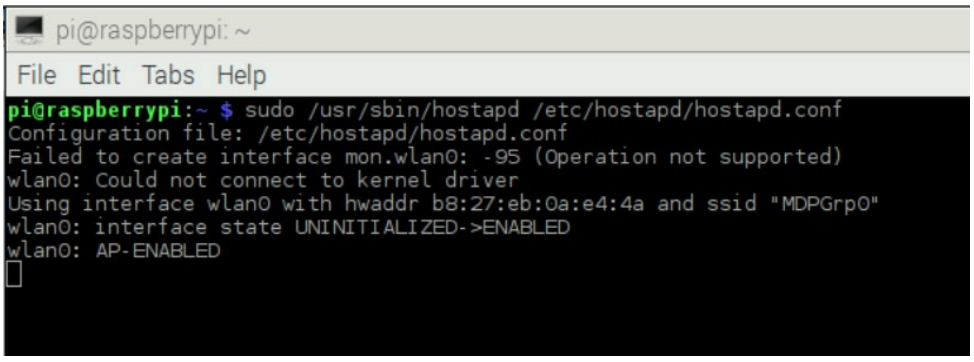
The configuration includes the driver (should be NL80211, letters all in lower case), the ssid of the Wifi hotspot and its password. It is suggested that each group use its group number within the ssid such that the Wifi signal can be clearly identified later.

Examples:
Group 1 should set the **ssid=MDPGrp1**
Group 3 should set the **ssid=MDPGrp3**

(v) To verify the setup, execute the command.

sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf

A series of messages should appear in the console, showing the access point is enabled.



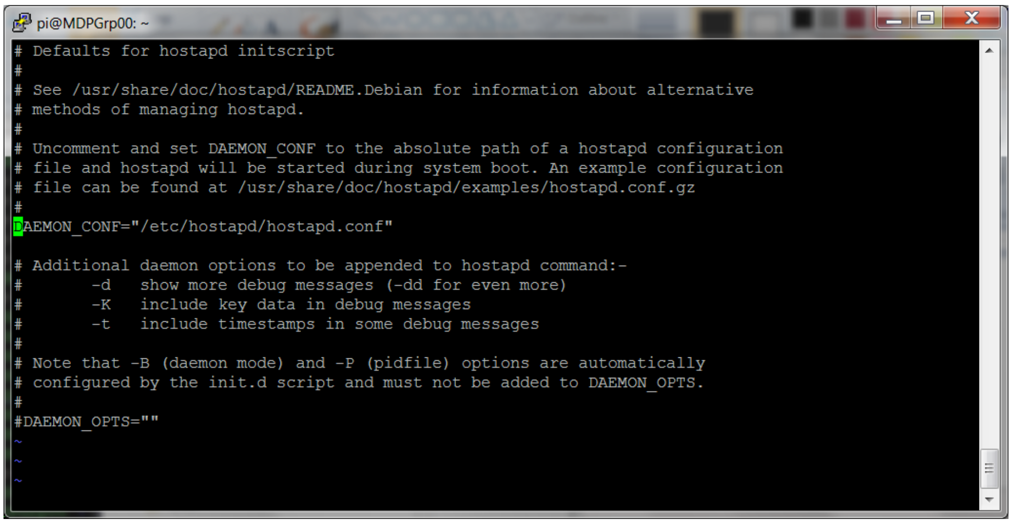
```
pi@raspberrypi: ~
File  Edit  Tabs  Help
pi@raspberrypi:~ $ sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf
Configuration file: /etc/hostapd/hostapd.conf
Failed to create interface mon.wlan0: -95 (Operation not supported)
wlan0: Could not connect to kernel driver
Using interface wlan0 with hwaddr b8:27:eb:0a:e4:4a and ssid "MDPGrp0"
wlan0: interface state UNINITIALIZED->ENABLED
wlan0: AP-ENABLED
```

Use your notebook/mobile phone wifi to scan for the network, the wifi ssid should appear on the screen if it is setup successfully.

(iv) Next edit the file **/etc/default/hostapd** and change the line to point to the configuration file just created as follows:

Change **#DAEMON_CONF=""**

to **DAEMON_CONF="/etc/hostapd/hostapd.conf"**



```
# Defaults for hostapd initscript
#
# See /usr/share/doc/hostapd/README.Debian for information about alternative
# methods of managing hostapd.
#
# Uncomment and set DAEMON_CONF to the absolute path of a hostapd configuration
# file and hostapd will be started during system boot. An example configuration
# file can be found at /usr/share/doc/hostapd/examples/hostapd.conf.gz
#
DAEMON_CONF="/etc/hostapd/hostapd.conf"

# Additional daemon options to be appended to hostapd command:-
#       -d   show more debug messages (-dd for even more)
#       -K   include key data in debug messages
#       -t   include timestamps in some debug messages
#
# Note that -B (daemon mode) and -P (pidfile) options are automatically
# configured by the init.d script and must not be added to DAEMON_OPTS.
#
#DAEMON_OPTS=""
~
~
~
```

(vi) To start the hostapd program, you can use the following command:

sudo service hostapd start

Note: If there is an error message such as

"Failed to start hostapd.service: Unit hostapd.service is masked"

Try the following commands:

```
sudo systemctl unmask hostapd
sudo systemctl enable hostapd
sudo systemctl start hostapd
```

C.2 – Setting up the Wireless interface wlan0

Before we configure the DHCP program that we installed earlier, we will first assign a static IP address to the Wifi interface, which is indicated as **wlan0** in Linux. (The wired Ethernet interface is referred to as **eth0**, as seen earlier). This is done by requesting the DHCP server to assign one as follows:

(i) **sudo nano /etc/dhcpcd.conf**

and add the following lines to the end of the file. In nano editor, you can go to the end of the file quickly with CTRL+W followed by CTRL+V.

```
interface wlan0
static ip_address=192.168.50.1/24
nohook wpa_supplicant
```

The **/24** is to indicate that the netmask to use is of 24-bit size, which is equivalent to 255.255.255.0.

Note that recent Linux (and hence RPI OS) by default uses dhcpcd to manage network interfaces, and the file **/etc/network/interfaces** should not be used anymore for network configuration.

The important setting is the IP for the Wifi, which must be unique among all the groups. The suggested value to used is **192.168.group_number.group_number**, or similar

i.e. Group 1 uses	192.168.1.1		
Group 2 uses	192.168.2.2	or	192.168.2.1
Group 40 uses	192.168.40.40	or	192.168.40.1

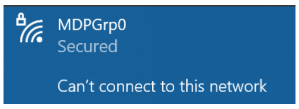
IP addresses 192.168.x.x is reserved for 'private' network, which should not be exposed to the Internet directly. Tunnelling using IP forwarding is used instead to access Internet from private network. For our case, the wlan0 is a private network, traffic is forwarded to eth0 via IP forwarding. Eth0 has internet connection.

Note: If you have difficulty establishing network connection for **eth0** using dhcp, try connecting the Ethernet cable to the LAN network point before boot-up

(iii) Restart the RPI using the command **sudo shutdown –r 0**, and the network interfaces should then be running and configured with IP address indicated in the **dhcpcd.conf** file. Check this using the command **ifconfig** after login.

The hostapd should also start running. This should be enabled by default after the hostapd is installed. Otherwise run the hostapd service command as before.

Use a Wifi-enable PC/notebook to scan for the Wifi signal. The RPI Wifi signal probably would be detected at this stage but can't be connected as there is no IP addressed being issued after connection.


MDPGrp0
Secured
Can't connect to this network

To do so, we need to setup a DHCP server on the RPI board, which we will do in the next step.

C.3 – Installing and setting up a DHCP server for the Wifi access point (dnsmasq)

The section is to setup a DHCP server on the RPI. The program that we will use here is the **dnsmasq**, which has been downloaded and installed earlier. But you can use other appropriate ones as you prefer – they are available through many references on RPI sites on internet.

(i) With the dnsmasq package installed earlier, there is a new file created: **/etc/dnsmasq.conf** , which contains a lot of irrelevant configurations. So, it is easier for us to start from scratch, but save the default configuration file before we make any changes: sudo mv /etc/dnsmasq.conf /etc/dnsmasq.conf.org . Create a new config file after saving the original.

```
sudo nano /etc/dnsmasq.conf
```

Key in the following lines in this file. In this example, it is to provide IP addresses between **192.168.50.11** and **192.168.50.11** for the wlan0 interface.

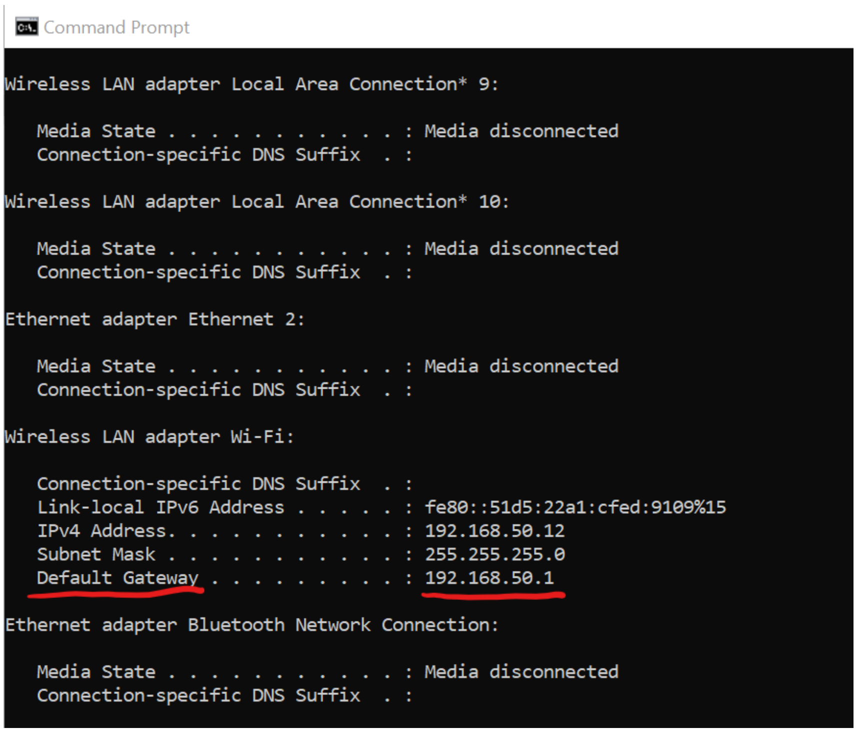
```
interface=wlan0
bind-dynamic
domain-needed
bogus-priv
dhcp-range=192.168.50.11,192.168.50.30,255.255.255.0,24h
dhcp-option=option:dns-server, 8.8.8.8
```

Note that the IP addresses setting used here correspond to the 192.168.50.x used in the Wifi's IP example earlier. Each group should hence set the IP addresses accordingly in this file.

(ii) Restart the dnsmasq package

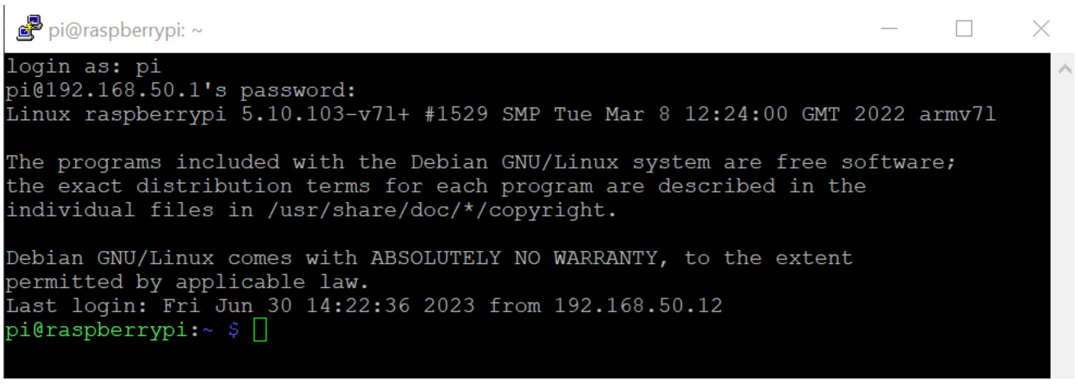
sudo systemctl start dnsmasq

Use the Notebook's Wifi connection to scan and connect to the RPI SSID, and upon successful connection, a dynamically generated IP address will be assigned by the DHCP just set up. Depending on which PC OS you are using, you can issue the equivalent of the IFCONFIG command in Linux. Windows equivalent command is IPCONFIG. Issue that in the windows command box and you will be able to see the assigned IP and the gateway IP. In the figure below, the IP assigned to the PC is 192.168.50.12 and RPI (Gateway) IP is 192.168.50.1



```
Wireless LAN adapter Local Area Connection* 9:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Local Area Connection* 10:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Ethernet adapter Ethernet 2:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :

Wireless LAN adapter Wi-Fi:

   Connection-specific DNS Suffix  . :
   Link-local IPv6 Address . . . . . : fe80::51d5:22a1:cfed:9109%15
   IPv4 Address. . . . . . . . . . . : 192.168.50.12
   Subnet Mask . . . . . . . . . . . : 255.255.255.0
   Default Gateway . . . . . . . . . : 192.168.50.1

Ethernet adapter Bluetooth Network Connection:

   Media State . . . . . . . . . . . : Media disconnected
   Connection-specific DNS Suffix  . :
```

At this juncture, using your notebook, you should also be able to SSH to RPI using its Wifi IP address i.e. 192.168.50.1.



```
login as: pi
pi@192.168.50.1's password:
Linux raspberrypi 5.10.103-v7l+ #1529 SMP Tue Mar 8 12:24:00 GMT 2022 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Fri Jun 30 14:22:36 2023 from 192.168.50.12
pi@raspberrypi:~ $
```

C.4 – Setting up IP forwarding for gateway through eth0

So far, we have setup the RPI as a wireless access point with a DHCP that can provide IP address to device wirelessly connected to it. And this is what we will need for the MDP mobile robotic setup.

However, during the code and system development, we would like to also access the internet to download other packages and sample codes through the internet. Rather than having a separate internet connection, we can setup the RPI as a Wifi AP and allow data to be forwarded between RPI's WLAN0 and ETH0. Since ETH0 has access to the internet, all devices that are connected to the RPI via its Wifi AP will have access to internet as well.

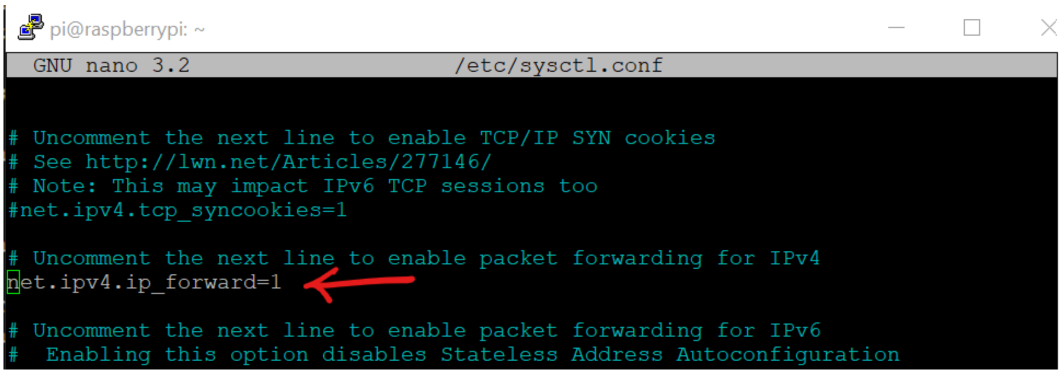
IP forwarding is to be used such that packet received at the Wifi interface wlan0 will be forward to the Ethernet interface eth0 for internet access. Remember that this is useful during the development work, but will not be needed for the eventual robotic setup since there won't be any wired connection to the robot.

(i) To enable IP forwarding in the kernel, we can use the following Linux command:

sudo sh –c "echo 1 > /proc/sys/net/ipv4/ip_forward"

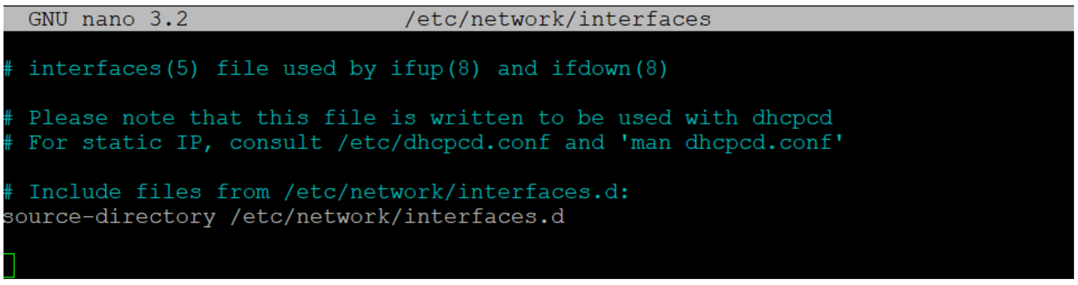
But to make this setup automatically on boot, we can edit the file **/etc/sysctl.conf** and uncomment the following line in file:

net.ipv4.ip_forward=1



```
pi@raspberrypi: ~
  GNU nano 3.2                    /etc/sysctl.conf


# Uncomment the next line to enable TCP/IP SYN cookies
# See http://lwn.net/Articles/277146/
# Note: This may impact IPv6 TCP sessions too
#net.ipv4.tcp_syncookies=1

# Uncomment the next line to enable packet forwarding for IPv4
net.ipv4.ip_forward=1

# Uncomment the next line to enable packet forwarding for IPv6
#  Enabling this option disables Stateless Address Autoconfiguration
```

(ii) Check that there are no options or configuration parameters in the /etc/network/interfaces file. You should only see the following lines in the file. Any additional configurations should be removed. Do **duplicate a copy of the original /etc/network/interfaces file** before doing any modifications.



```
  GNU nano 3.2                  /etc/network/interfaces

# interfaces(5) file used by ifup(8) and ifdown(8)

# Please note that this file is written to be used with dhcpcd
# For static IP, consult /etc/dhcpcd.conf and 'man dhcpcd.conf'

# Include files from /etc/network/interfaces.d:
source-directory /etc/network/interfaces.d
```

(iii) NFtables & IPtables setup. Next the rules need to be added that will allow any device connected to the access point to be able to use the network or internet connected to eth0. This is done with IP Table rules for older OS's Buster, Stretch and Jessie. Bullseye onwards use Nftables. Instructions below are for OS Buster, Stretch and Jessie. Please refer to resources online for Bulleyes configuration (if you chose to use Bullseye). One of the references can be found in https://www.raspberryconnect.com/projects/65-raspberrypi-hotspot-accesspoints/168-raspberry-pi-hotspot-access-point-dhcpcd-method

These tables will need to be loaded every time the Raspberry Pi starts up.

First create the file for the IP table rules.

sudo nano /etc/iptables-buster

Add the lines below to the file

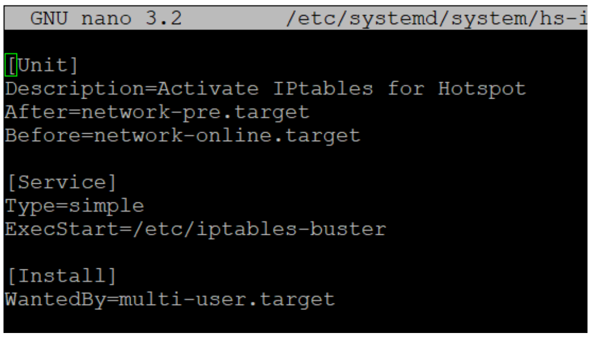
```
#!/bin/bash

iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT

iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```

Update the file attributes to executable

sudo chmod +x /etc/iptables-buster

Create the IP table service file, update the file with the following information, save and exit.

sudo nano /etc/systemd/system/buster-iptables.service



```
  GNU nano 3.2           /etc/systemd/system/hs-i

[Unit]
Description=Activate IPtables for Hotspot
After=network-pre.target
Before=network-online.target

[Service]
Type=simple
ExecStart=/etc/iptables-buster

[Install]
WantedBy=multi-user.target
```

The service file needs to be activated at every boot up. For that we will use the system and service manager in linux (systemd). The following command enable the hs-iptables service.

sudo systemctl enable buster-iptables

(iv) Reboot the RPI board: **sudo reboot**

Once the wifi network is detected, connect to it and SSH to the RPI board using its Wifi's IP address. The connection should show that it has internet access now.

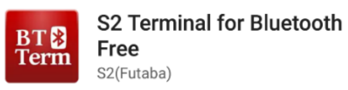
From this point onward, the local monitor and keyboard are not necessary anymore, as we can always SSH to the RPI board using the Wifi link, which has a fixed IP address.

D. Setting up the RPI Bluetooth link with Galaxy A7 Lite Tablet.

RPI comes with a built-in Bluetooth transceiver which can be used to connect to another Bluetooth device such as the Galaxy A7 tablet. Here, we will try to establish a Bluetooth connection between the RPI and A7 tablet, communicating using the Serial Port profile (SPP). Note that the Bluetooth protocol used here is the Classic Bluetooth, not the Bluetooth Low energy (BLE), which is an entirely different protocol.

D.1 Preparation work

(a) To be able to verify the proper configuration of the RPI bluetooth connection, we need a bluetooth application (APP) to run on the Galaxy A7 that utilizes the SPP. You will eventually develop such an APP for your MDP system. For the initial setup verification purpose, we will install a free APP (from the Play Store) on the A7 tablet. In this document, the APP used for the setup verification is the Serial Bluetooth Terminal. We have previously used the BT-Term but it appears to be missing from Google Playstore. You can use the BT Term app too if you can find it, or any other app that supports Bluetooth SPP.

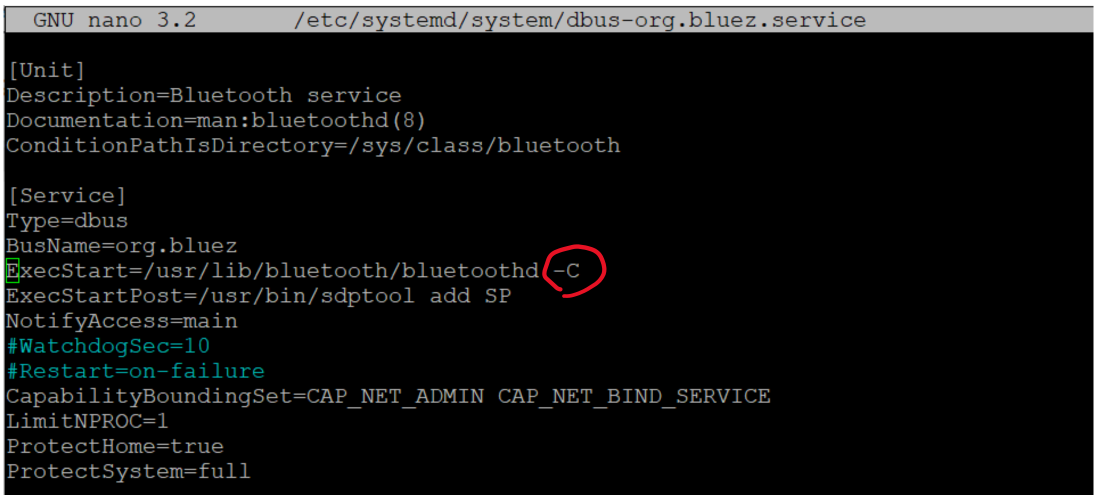


Download and install the Serial Bluetooth Serial APP. Note that we will need to pair the A7 with RPI via Bluetooth first.

(b) For successful interfacing with A7, the RPI bluetooth daemon process need to be run in the compatible mode. This can be done through the file **/etc/systemd/system/dbus-org.bluez.service** as follows.

sudo nano /etc/systemd/system/dbus-org.bluez.service

Add '**-C**' at the end of the '**ExecStart=**' line.

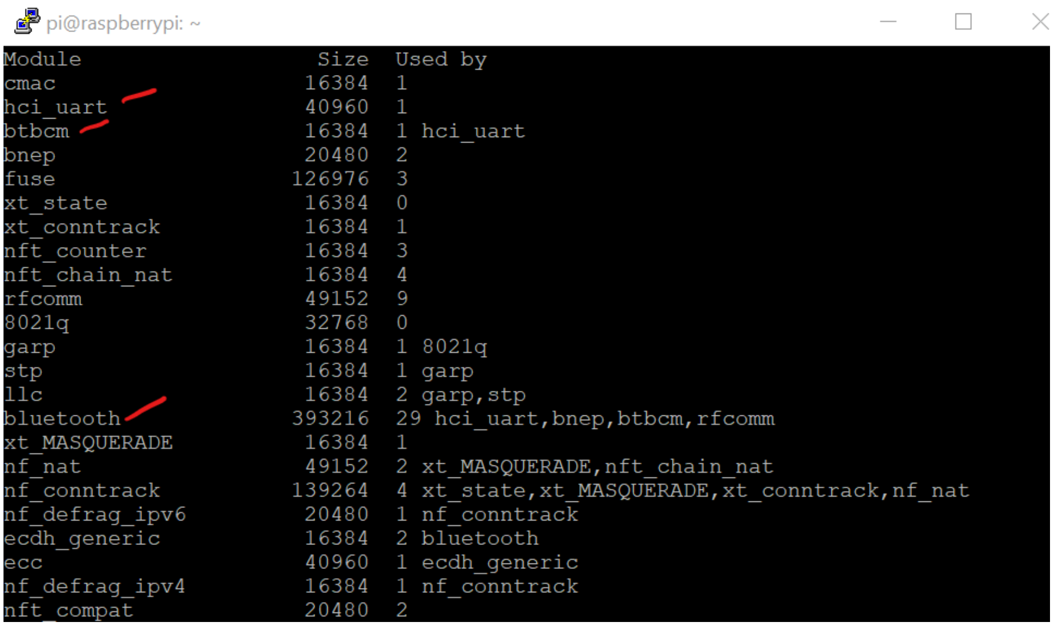


```
  GNU nano 3.2         /etc/systemd/system/dbus-org.bluez.service

[Unit]
Description=Bluetooth service
Documentation=man:bluetoothd(8)
ConditionPathIsDirectory=/sys/class/bluetooth

[Service]
Type=dbus
BusName=org.bluez
ExecStart=/usr/lib/bluetooth/bluetoothd -C
ExecStartPost=/usr/bin/sdptool add SP
NotifyAccess=main
#WatchdogSec=10
#Restart=on-failure
CapabilityBoundingSet=CAP_NET_ADMIN CAP_NET_BIND_SERVICE
LimitNPROC=1
ProtectHome=true
ProtectSystem=full
```

Then restart the RPI: **sudo reboot** .

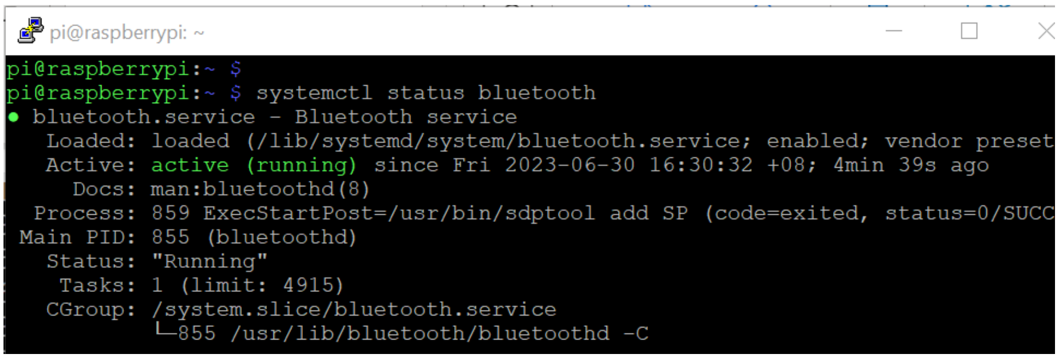
D.2 Checking the RPI Bluetooth status

It is likely that the RPI Bluetooth drivers are already up and running upon boot-up (i.e. by default). You can check its drivers by typing through the SSH terminal, the command: **lsmod**
There should be three drivers as shown above.



```
pi@raspberrypi: ~
Module                  Size  Used by
cmac                   16384  1
hci_uart               40960  1
btbcm                  16384  1 hci_uart
bnep                   20480  2
fuse                  126976  3
xt_state               16384  0
xt_conntrack           16384  1
nft_counter            16384  3
nft_chain_nat          16384  4
rfcomm                 49152  9
8021q                  32768  0
garp                   16384  1 8021q
stp                    16384  1 garp
llc                    16384  2 garp,stp
bluetooth             393216  29 hci_uart,bnep,btbcm,rfcomm
xt_MASQUERADE          16384  1
nf_nat                 49152  2 xt_MASQUERADE,nft_chain_nat
nf_conntrack          139264  4 xt_state,xt_MASQUERADE,xt_conntrack,nf_nat
nf_defrag_ipv6         20480  1 nf_conntrack
ecdh_generic           16384  2 bluetooth
ecc                    40960  1 ecdh_generic
nf_defrag_ipv4         16384  1 nf_conntrack
nft_compat             20480  2
```

Next check the status of the Bluetooth interface, use the command:



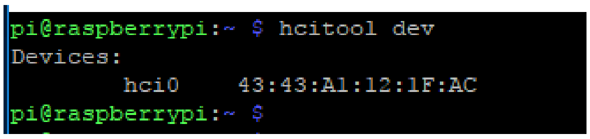
```
pi@raspberrypi: ~
pi@raspberrypi:~ $
pi@raspberrypi:~ $ systemctl status bluetooth
● bluetooth.service - Bluetooth service
   Loaded: loaded (/lib/systemd/system/bluetooth.service; enabled; vendor preset
   Active: active (running) since Fri 2023-06-30 16:30:32 +08; 4min 39s ago
     Docs: man:bluetoothd(8)
  Process: 859 ExecStartPost=/usr/bin/sdptool add SP (code=exited, status=0/SUCC
 Main PID: 855 (bluetoothd)
   Status: "Running"
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/bluetooth.service
           └─855 /usr/lib/bluetooth/bluetoothd -C
```

It should indicate that the Bluetooth service status is "**Running**".
(Otherwise, issue the command **sudo service bluetooth start**)

D.3 Configuring the RPI Bluetooth interface with A7

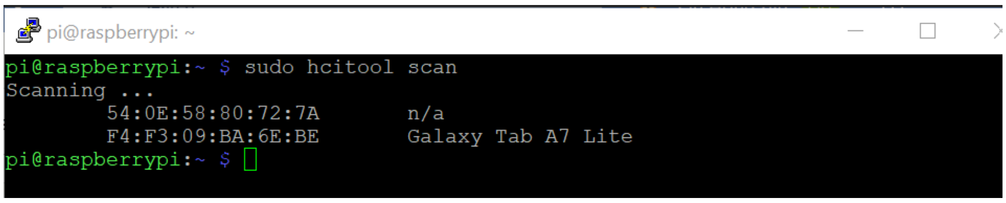
We will be using the **hcitool** command to configure the Bluetooth interface.
(Note: At this stage, you might want to also run the **bluetoothctl** utility on another terminal that can monitor the bluetooth activities – see D.7)

Type **hcitool dev** and this will show the hardware device address of the RPI.



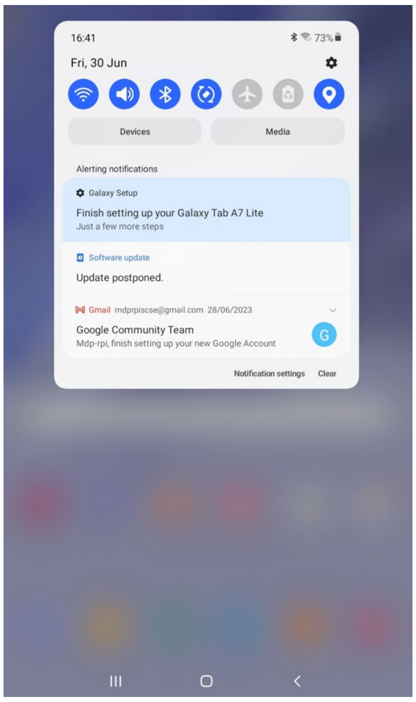
```
pi@raspberrypi:~ $ hcitool dev
Devices:
        hci0    43:43:A1:12:1F:AC
pi@raspberrypi:~ $
```

We can scan for active discoverable Bluetooth devices nearby using the command **sudo hcitool scan**



```
pi@raspberrypi: ~
pi@raspberrypi:~ $ sudo hcitool scan
Scanning ...
        54:0E:58:80:72:7A       n/a
        F4:F3:09:BA:6E:BE       Galaxy Tab A7 Lite
pi@raspberrypi:~ $ 
```

The above screenshot shows that a Galaxy A7 tablet has been detected. Take note of the hardware device address of your A7. If your RPI is not able to detect your A7, turn-off, and then turn-on the Bluetooth on your A7, and check that it is made discoverable (i.e. visible) to nearby devices. The control dashboard can be brought out by swiping down from the top.



Alerting notifications

Galaxy Setup
Finish setting up your Galaxy Tab A7 Lite
Just a few more steps

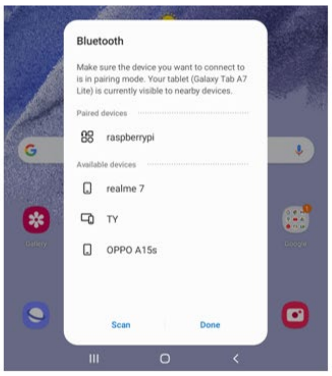
Software update
Update postponed.

Gmail mdprpiscse@gmail.com 28/06/2023
Google Community Team
Mdp-rpi, finish setting up your new Google Account

Note that while the RPI can detect the A7, the A7 may not be able to detect the RPI, such as when the RPI is not set to discoverable. To make the RPI also visible to A7, use the command

sudo hciconfig hci0 piscan

On the A7, refresh (or turn-off and turn-on) the Bluetooth control, which will then detect visible devices nearby, including the RPI. Press (and hold for a while) on the 'raspberrypi' to make A7 paired with the RPI. Figure below illustrates the pairing process. Note that you may be prompted to allow pairing on both the A7 and the RPI side.



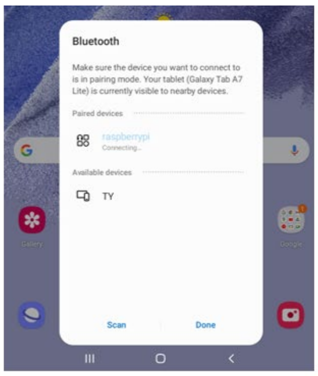
Bluetooth

Make sure the device you want to connect to is in pairing mode. Your tablet (Galaxy Tab A7 Lite) is currently visible to nearby devices.

Paired devices
raspberrypi

Available devices
realme 7
TY
OPPO A15s

Bluetooth scan



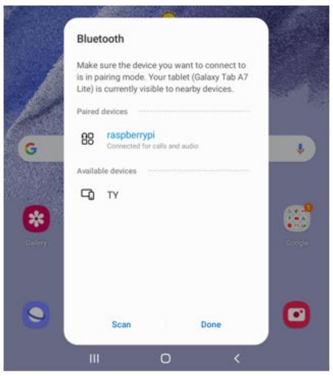
Bluetooth

Make sure the device you want to connect to is in pairing mode. Your tablet (Galaxy Tab A7 Lite) is currently visible to nearby devices.

Paired devices
raspberrypi
Connecting...

Available devices
TY

Initiate Pairing



Bluetooth

Make sure the device you want to connect to is in pairing mode. Your tablet (Galaxy Tab A7 Lite) is currently visible to nearby devices.

Paired devices
raspberrypi
Connected for calls and audio

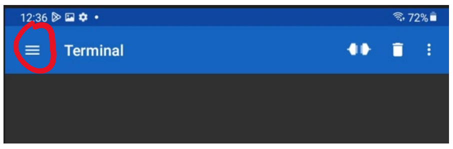
Available devices
TY

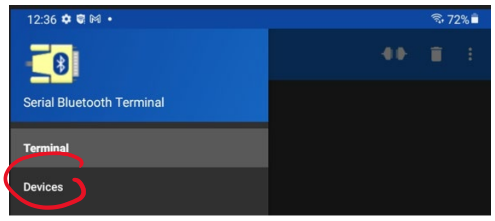
Paired

D.4 Bluetooth connection between RPI and N7 using SPP

At this stage, the A7 is ready to communicate with the RPI. We will try to establish a connection between the A7 and RPI using the Bluetooth's Serial Port Profile (SPP). Upon successful connection, the two devices will be able to communicate via virtual COM port (UART protocol).

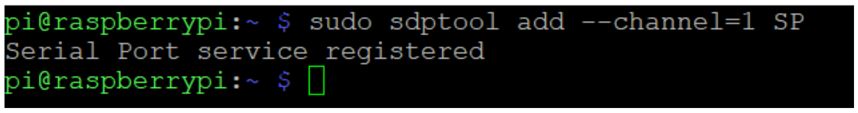
On the A7, run the Serial Bluetooth Terminal APP that we have installed earlier on, which uses SPP. Click on the 3-bar on the top left-hand corner and click on "Devices".





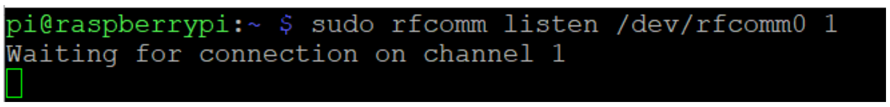
On RPI, add a SP profile for channel number 1 (you can try other channels but only the latest channel initialised will be valid.

sudo sdptool add --channel=1 SP



```
pi@raspberrypi:~ $ sudo sdptool add --channel=1 SP
Serial Port service registered
pi@raspberrypi:~ $ 
```

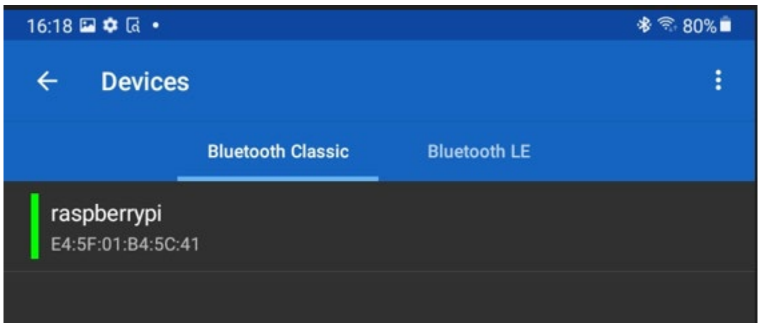
We then issue the command to place the RPI listening at channel 1, using the rfcomm listen command.



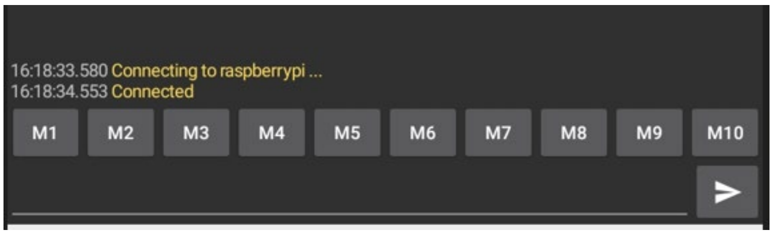
```
pi@raspberrypi:~ $ sudo rfcomm listen /dev/rfcomm0 1
Waiting for connection on channel 1
```

This configure the RPI as a Server, listening (waiting) for a Client (i.e. A7) to connect through channel 1, which is mapped to a node file /dev/rfcomm0. The /dev/rfcomm0 file will be created only after this command is executed successfully.

On the A7 tablet, select raspberrypi that was paired earlier on.

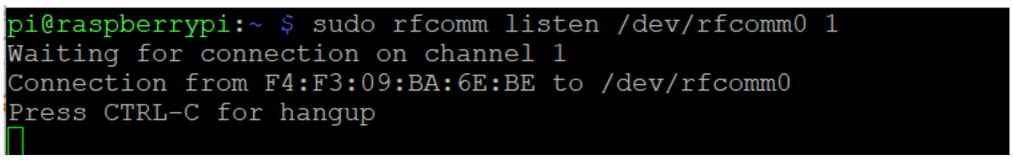


This will make A7 request to establish a connection with RPI, as shown below



```
16:18:33.580 Connecting to raspberrypi ...
16:18:34.553 Connected
```

The terminal display in the RPI will also show that the connection is achieved successfully.

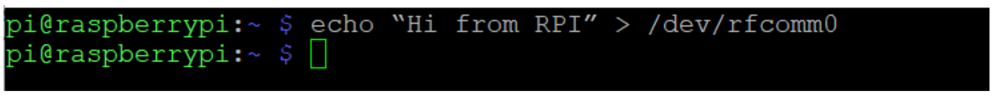


```
pi@raspberrypi:~ $ sudo rfcomm listen /dev/rfcomm0 1
Waiting for connection on channel 1
Connection from F4:F3:09:BA:6E:BE to /dev/rfcomm0
Press CTRL-C for hangup
```

D.5 Testing

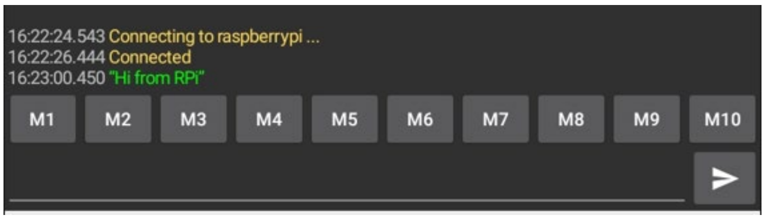
To test the connection, open a new SSH terminal and type

echo "Hi from RPI" > /dev/rfcomm0



```
pi@raspberrypi:~ $ echo "Hi from RPI" > /dev/rfcomm0
pi@raspberrypi:~ $
```

A message should appear on the A7 tablet screen

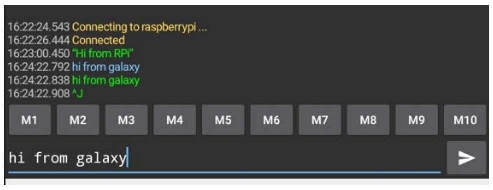


```
16:22:24.543 Connecting to raspberrypi ...
16:22:26.444 Connected
16:23:00.450 "Hi from RPi"
```

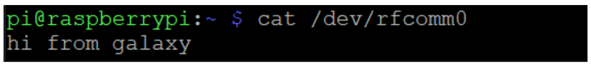
Similarly we can send message from A7 to RPI using its on-screen keyboard.
First open another terminal console. Get the RPI ready to display data in the 'file' /dev/rfcomm0 by typing the following command in the terminal:

cat /dev/rfcomm0

Then on the A7 Serial Bluetooth Terminal App, type a message (such as "hi from galaxy"). The message should appear on the RPI console display too.

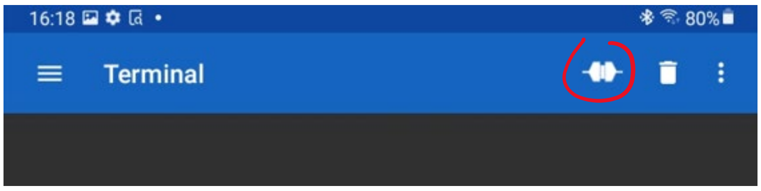


```
16:22:24.543 Connecting to raspberrypi ...
16:22:26.444 Connected
16:23:00.450 "Hi from RPi"
16:24:22.792 hi from galaxy
16:24:22.838 hi from galaxy
16:24:22.908 ^J
```



```
pi@raspberrypi:~ $ cat /dev/rfcomm0
hi from galaxy
```

Disconnect A7 from the RPI by pressing the on-screen DISCONNECT icon.



Note: It is observed that after you disconnect the Bluetooth connection, you may need to re-start the rfcomm by issuing the command.

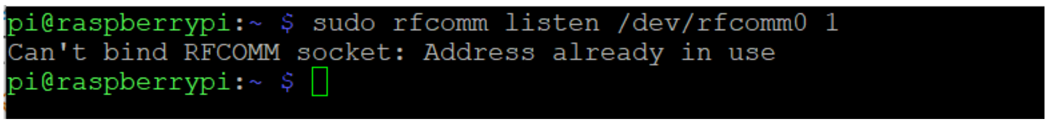
sudo systemctl start rfcomm

follow by

sudo rfcomm listen /dev/rfcomm0 1

Note that if you have implemented the steps outlined in D.6 (Automate the Bluetooth connection), then you would not need to start and listen for the rfcomm during after the RPI first bootup, as the two commands are already executed by the script file you configured in D.6. However, in event that you disconnect the rfcomm connection (at RPI or Android side), you will need to start and listen the rfcomm again.

There are times where you apply too many "sudo rfcomm listen" commands to the same node/channel, in which you will get the following error prompt

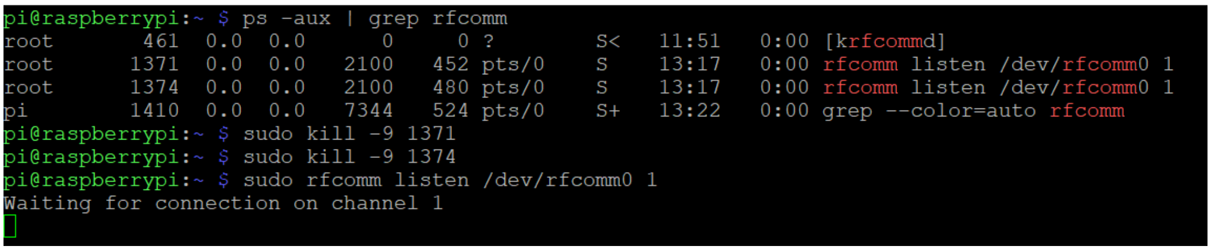


```
pi@raspberrypi:~ $ sudo rfcomm listen /dev/rfcomm0 1
Can't bind RFCOMM socket: Address already in use
pi@raspberrypi:~ $ 
```

You can either reboot your RPI or kill the numerous rfcomm listen commands you have executed. To know the PID of all the rfcomm listen commands, issue the following commands. The kill command can also be used to kill any processes running in the system but do use with care. 'grep' filters the output of the ps -aux command and shows you only those outputs with the word 'rfcomm'.

sudo ps –aux | grep rfcomm

sudo kill -9 <PID>



```
pi@raspberrypi:~ $ ps -aux | grep rfcomm
root       461  0.0  0.0      0     0 ?        S<   11:51   0:00 [krfcommd]
root      1371  0.0  0.0   2100   452 pts/0    S    13:17   0:00 rfcomm listen /dev/rfcomm0 1
root      1374  0.0  0.0   2100   480 pts/0    S    13:17   0:00 rfcomm listen /dev/rfcomm0 1
pi        1410  0.0  0.0   7344   524 pts/0    S+   13:22   0:00 grep --color=auto rfcomm
pi@raspberrypi:~ $ sudo kill -9 1371
pi@raspberrypi:~ $ sudo kill -9 1374
pi@raspberrypi:~ $ sudo rfcomm listen /dev/rfcomm0 1
Waiting for connection on channel 1
```

D.6 Automate the Bluetooth connection

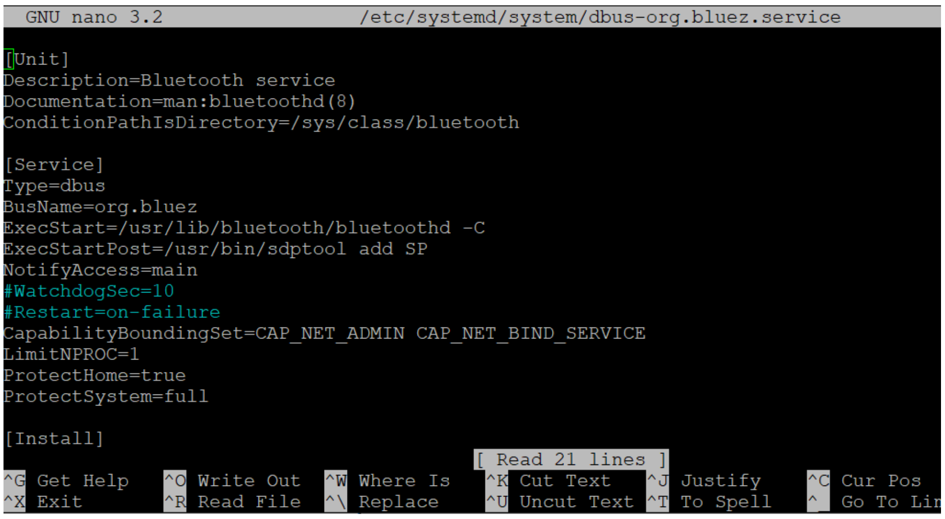
Once you have verified that the Bluetooth connection can operate as expected, you can automate the connection setup process upon boot-up by doing the following.

(i) Add a new “ExecStartPost=” line immediate after the “ExecStart”’ line in the file **/etc/systemd/system/dbus-org.bluez.service**

ExecStartPost=/usr/bin/sdptool add SP

This is to add a SP profile for first available channel, typically channel number 1. If you want to be specific, you can use

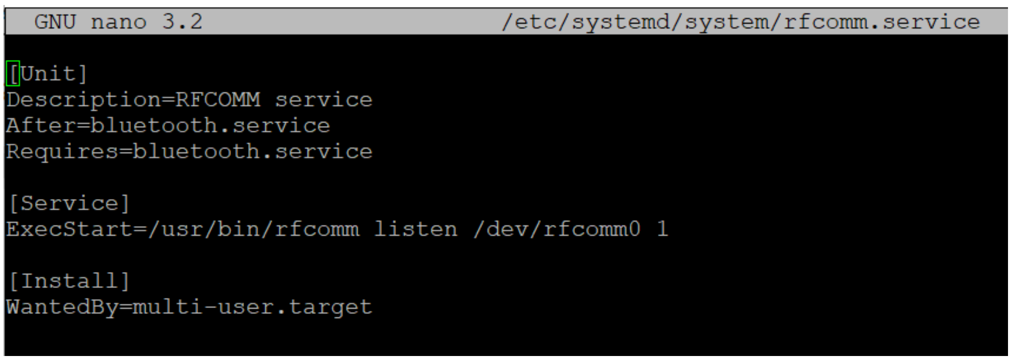
ExecStartPost=/usr/bin/sdptool add –channel=1 SP



```
  GNU nano 3.2                    /etc/systemd/system/dbus-org.bluez.service

[Unit]
Description=Bluetooth service
Documentation=man:bluetoothd(8)
ConditionPathIsDirectory=/sys/class/bluetooth

[Service]
Type=dbus
BusName=org.bluez
ExecStart=/usr/lib/bluetooth/bluetoothd -C
ExecStartPost=/usr/bin/sdptool add SP
NotifyAccess=main
#WatchdogSec=10
#Restart=on-failure
CapabilityBoundingSet=CAP_NET_ADMIN CAP_NET_BIND_SERVICE
LimitNPROC=1
ProtectHome=true
ProtectSystem=full

[Install]
                                        [ Read 21 lines ]
^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 Lin
```

(ii) Create a file to automate the launching of the rfcomm service upon boot up, using content as shown below

sudo nano /etc/systemd/system/rfcomm.service



```
  GNU nano 3.2                      /etc/systemd/system/rfcomm.service

[Unit]
Description=RFCOMM service
After=bluetooth.service
Requires=bluetooth.service

[Service]
ExecStart=/usr/bin/rfcomm listen /dev/rfcomm0 1

[Install]
WantedBy=multi-user.target
```

To activate the service file so that it starts at every boot up, issue the following command

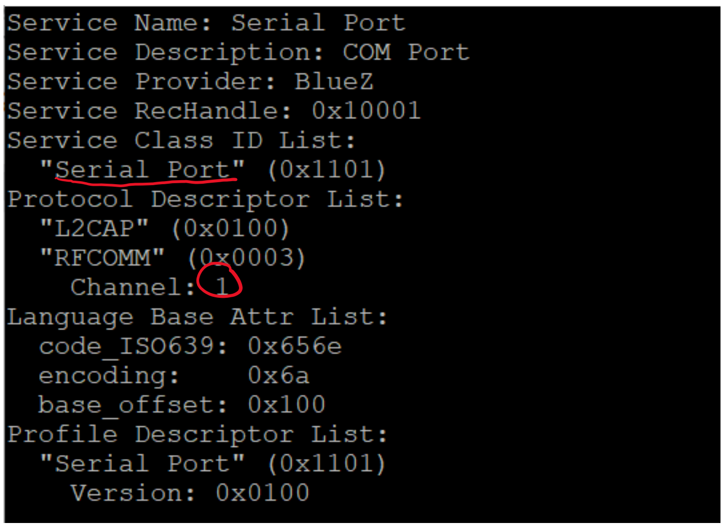
sudo systemctl enable rfcomm

If for some reason you need to disable this service then issue the following command

sudo systemctl disable rfcomm

(iii) Restart RPI. Upon boot up and paring of the bluetooth, you can check whether the SPP service is launched using the command. Scroll through the output of the sdptool browse and you will be able to locate a service for “Serial port”.

sudo sdptool browse local



```
Service Name: Serial Port
Service Description: COM Port
Service Provider: BlueZ
Service RecHandle: 0x10001
Service Class ID List:
  "Serial Port" (0x1101)
Protocol Descriptor List:
  "L2CAP" (0x0100)
  "RFCOMM" (0x0003)
    Channel: 1
Language Base Attr List:
  code_ISO639: 0x656e
  encoding:    0x6a
  base_offset: 0x100
Profile Descriptor List:
  "Serial Port" (0x1101)
    Version: 0x0100
```

(iv) If everything appears OK up to this stage, connect the A7 Serial Bluetooth Terminal as had been done earlier - and it should work.

D.7 Monitoring RPI Bluetooth status using bluetoothctl (Optional)

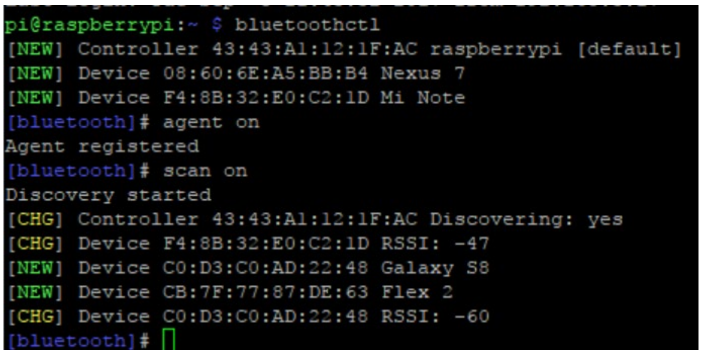
This is a handy Bluetooth utility that you can use to monitor the connection status of the Bluetooth interface operation in real time.

Open another SSH terminal, and execute the command: **bluetoothctl**
This utility is part of the **bluez** protocol stack package. If it is not available, it can be installed using:

sudo apt-get install bluez

Once the utility is running, you can start the monitoring by typing the commands:

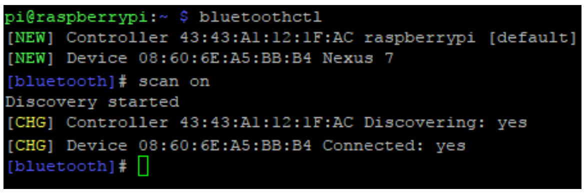
agent on
scan on



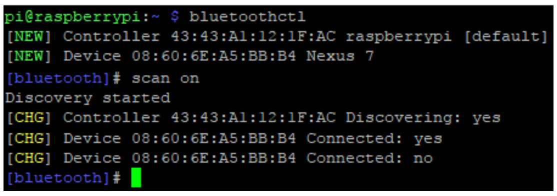
```
pi@raspberrypi:~ $ bluetoothctl
[NEW] Controller 43:43:A1:12:1F:AC raspberrypi [default]
[NEW] Device 08:60:6E:A5:BB:B4 Nexus 7
[NEW] Device F4:8B:32:E0:C2:1D Mi Note
[bluetooth]# agent on
Agent registered
[bluetooth]# scan on
Discovery started
[CHG] Controller 43:43:A1:12:1F:AC Discovering: yes
[CHG] Device F4:8B:32:E0:C2:1D RSSI: -47
[NEW] Device C0:D3:C0:AD:22:48 Galaxy S8
[NEW] Device CB:7F:77:87:DE:63 Flex 2
[CHG] Device C0:D3:C0:AD:22:48 RSSI: -60
[bluetooth]#
```

If there is any active Bluetooth device detected, it will be displayed on screen, (such as the A7, Mi Note, Galaxy and Flex devices shown above), together with their hardware device addresses.

It will also display the connection status, which is continuous updated in real time.

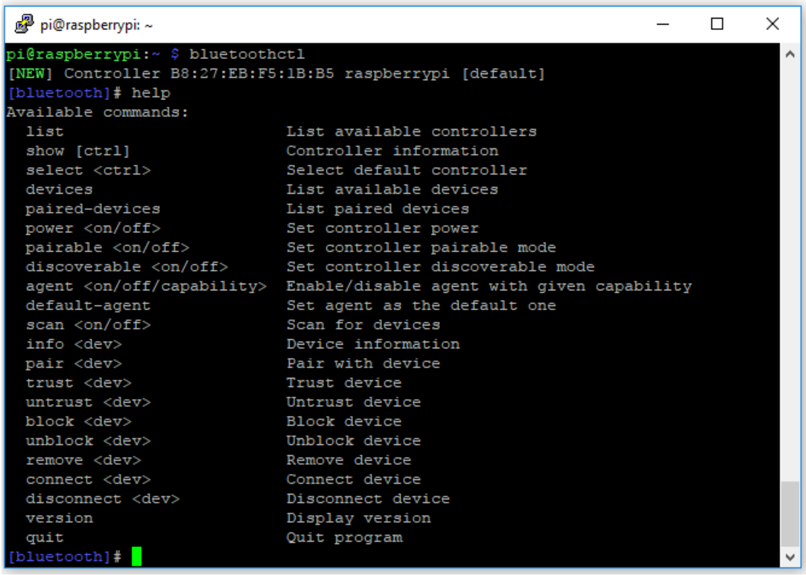


```
pi@raspberrypi:~ $ bluetoothctl
[NEW] Controller 43:43:A1:12:1F:AC raspberrypi [default]
[NEW] Device 08:60:6E:A5:BB:B4 Nexus 7
[bluetooth]# scan on
Discovery started
[CHG] Controller 43:43:A1:12:1F:AC Discovering: yes
[CHG] Device 08:60:6E:A5:BB:B4 Connected: yes
[bluetooth]#
```



```
pi@raspberrypi:~ $ bluetoothctl
[NEW] Controller 43:43:A1:12:1F:AC raspberrypi [default]
[NEW] Device 08:60:6E:A5:BB:B4 Nexus 7
[bluetooth]# scan on
Discovery started
[CHG] Controller 43:43:A1:12:1F:AC Discovering: yes
[CHG] Device 08:60:6E:A5:BB:B4 Connected: yes
[CHG] Device 08:60:6E:A5:BB:B4 Connected: no
[bluetooth]#
```

Type ‘help’ to see the list of commands available in this program.



```
pi@raspberrypi:~ $ bluetoothctl
[NEW] Controller B8:27:EB:F5:1B:B5 raspberrypi [default]
[bluetooth]# help
Available commands:
  list                        List available controllers
  show [ctrl]                 Controller information
  select <ctrl>               Select default controller
  devices                     List available devices
  paired-devices              List paired devices
  power <on/off>              Set controller power
  pairable <on/off>           Set controller pairable mode
  discoverable <on/off>       Set controller discoverable mode
  agent <on/off/capability>   Enable/disable agent with given capability
  default-agent               Set agent as the default one
  scan <on/off>               Scan for devices
  info <dev>                  Device information
  pair <dev>                  Pair with device
  trust <dev>                 Trust device
  untrust <dev>               Untrust device
  block <dev>                 Block device
  unblock <dev>               Unblock device
  remove <dev>                Remove device
  connect <dev>               Connect device
  disconnect <dev>            Disconnect device
  version                     Display version
  quit                        Quit program
[bluetooth]#
```

E. Setting up RPI Camera.

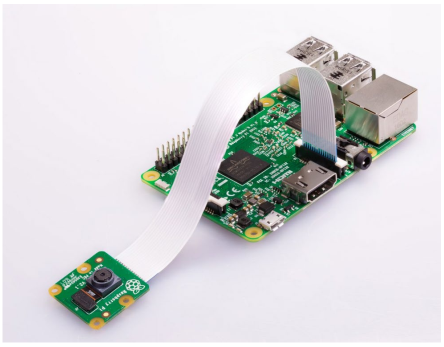
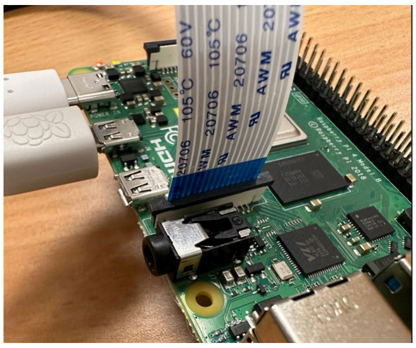
The RPI camera module used is v2.1. The details can be found in the following URL

https://projects.raspberrypi.org/en/projects/getting-started-with-picamera

Note that the Python Picamera module is currently not, by default, compatible with the latest version of Raspberry Pi OS (Bullseye). To use the Picamera module, you will need to enable legacy support for the camera. More details in the URL above.

Connection to RPI

The Camera is connected to the RPI via a ribbon cable. Check the figure below for the orientation of the cable.

Sample code

Camera preview

```python
from picamera import PiCamera
from time import sleep

camera = PiCamera()

camera.start_preview()
sleep(5)
camera.stop_preview()
```

Still pictures

```python
camera.start_preview()
sleep(5)
camera.capture('/home/pi/Desktop/image.jpg')
camera.stop_preview()
```

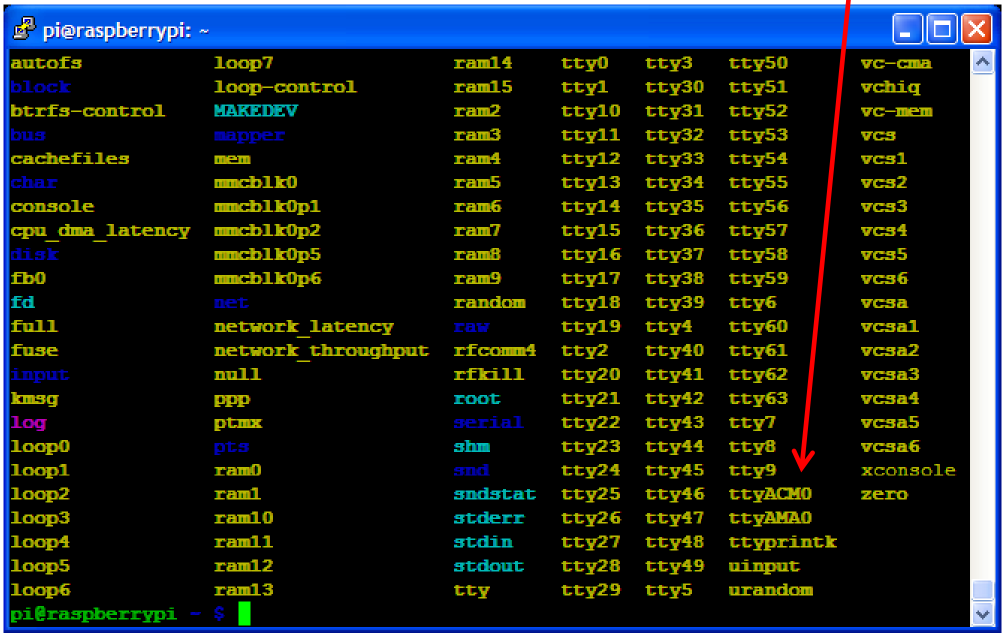
Video Recording

```python
camera.start_preview()
camera.start_recording('/home/pi/Desktop/video.h264')
sleep(5)
camera.stop_recording()
camera.stop_preview()
```

F. Setting up of the USB interface between RPI and STM32F Board.

Communication between the RPI and STM32F Board will be through their USB ports. For the STM32F Board, it is recommended to use its UART2 port that is accessible through the Micro USB port located at the middle of the board (the middle one among the three USB ports).

The STM32F Board provides its UART serial interface through its USB port running at the baudrate of 115200. Communication between the two devices can hence be done through the ttyACMx/ttyUSBx interface file that is automatically created on connection (typical /dev/ttyACMx or /dev/ttyUSBx in RPI).



```
pi@raspberrypi: ~
autofs           loop7              ram14    tty0   tty3   tty50     vc-cma
block            loop-control       ram15    tty1   tty30  tty51     vchiq
btrfs-control    MAKEDEV            ram2     tty10  tty31  tty52     vc-mem
bus              mapper             ram3     tty11  tty32  tty53     vcs
cachefiles       mem                ram4     tty12  tty33  tty54     vcs1
char             mmcblk0            ram5     tty13  tty34  tty55     vcs2
console          mmcblk0p1          ram6     tty14  tty35  tty56     vcs3
cpu_dma_latency  mmcblk0p2          ram7     tty15  tty36  tty57     vcs4
disk             mmcblk0p5          ram8     tty16  tty37  tty58     vcs5
fb0              mmcblk0p6          ram9     tty17  tty38  tty59     vcs6
fd               net                random   tty18  tty39  tty6      vcsa
full             network_latency    raw      tty19  tty4   tty60     vcsa1
fuse             network_throughput rfcomm4  tty2   tty40  tty61     vcsa2
input            null               rfkill   tty20  tty41  tty62     vcsa3
kmsg             ppp                root     tty21  tty42  tty63     vcsa4
log              ptmx               serial   tty22  tty43  tty7      vcsa5
loop0            pts                shm      tty23  tty44  tty8      vcsa6
loop1            ram0               snd      tty24  tty45  tty9      xconsole
loop2            ram1               sndstat  tty25  tty46  ttyACM0   zero
loop3            ram10              stderr   tty26  tty47  ttyAMA0
loop4            ram11              stdin    tty27  tty48  ttyprintk
loop5            ram12              stdout   tty28  tty49  uinput
loop6            ram13              tty      tty29  tty5   urandom
pi@raspberrypi ~ $ 
```

Data can hence be sent and displayed through the ttyACM0 file. To demonstrate the data transfer, the STM32F Board needs to run a corresponding program. (This will be left as an exercise for the student to work on).

G. Other packages

Other handy programs to have on the RPI are the following.

(i) Apache web server on RPI to host webpage for access through Wifi.

sudo apt-get install apache2

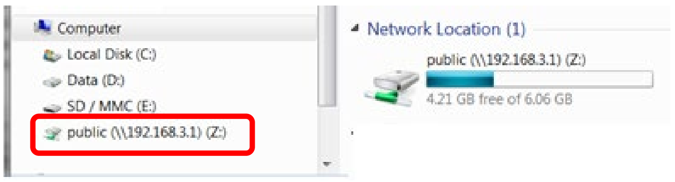
Use a web browser on the connected notebook and key in the RPI's IP address
http://192.168.50.1

(ii) Samba Server on RPI to allow file transfer from the RPI and network computer.

sudo apt-get install samba samba-common-bin

(see http://elinux.org/R-Pi_NAS for further configuration detail)

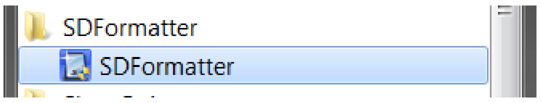
With Samba server on RPI, user on another computer will be able to read and write to it as a networked drive which appears to be locally-attached but is actually attached to the RPI. For simple file transfer, you can also use VNC.



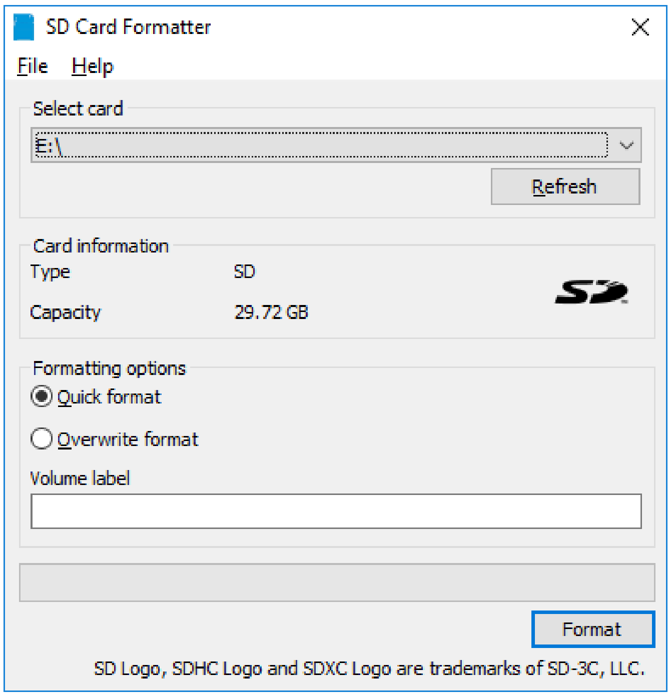
Appendix A: Installing a new Raspbian to the MicroSD card

Formatting the micro-SD card.

To begin, the MicroSD card will need to be formatted first, using the Windows PC in the Lab. This is to be done using the SDFormatter program, which should already be installed on the PC in the Lab. If not download the SDFormatter program (google for it) and install it on the Windows PC.



Insert the SD card into the PC. Run the SDFormatter. Select the SD card and format it.



SD Card Formatter

File Help

Select card: E:\

Refresh

Card information
Type: SD
Capacity: 29.72 GB

Formatting options
(●) Quick format
( ) Overwrite format

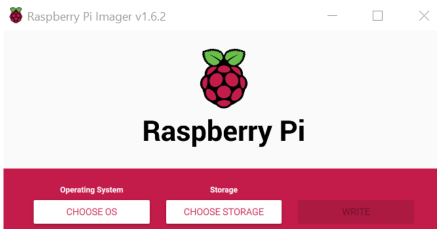
Volume label

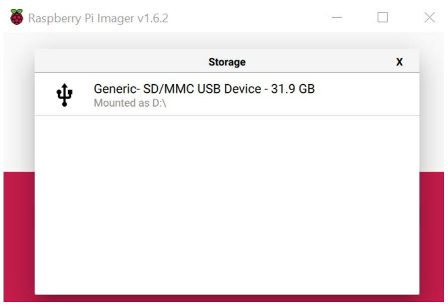
Format

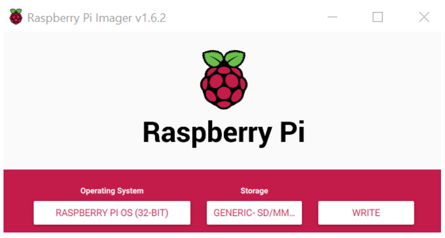
SD Logo, SDHC Logo and SDXC Logo are trademarks of SD-3C, LLC.

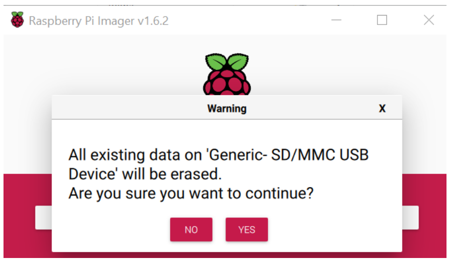
Installing latest Raspberry Pi OS using Raspberry Pi Imager

(i) Go to https://www.raspberrypi.org/software/
(ii) Download and install Raspberry Pi Imager
(iii) Put the micro-SD card in the computer (through a SD card reader). Put the SD card you'll use with your Raspberry Pi into the reader and run Raspberry Pi Imager.
(iv) The default (as of writing) is the Bullseye OS (version 11). Choose Buster OS instead (version 10). The Buster installation can be found under 'Legacy' OS. Choose the desktop version.

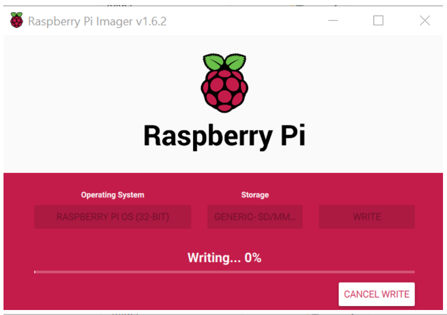


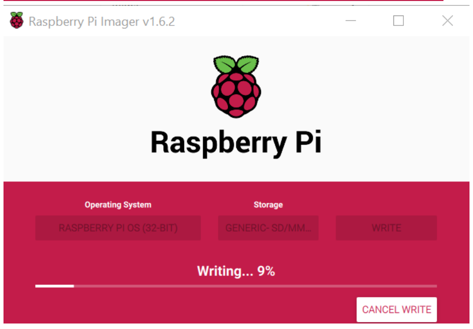

Raspberry Pi Imager v1.6.2
Storage
Generic- SD/MMC USB Device - 31.9 GB
Mounted as D:\

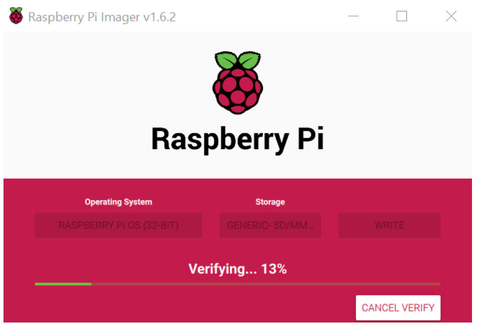


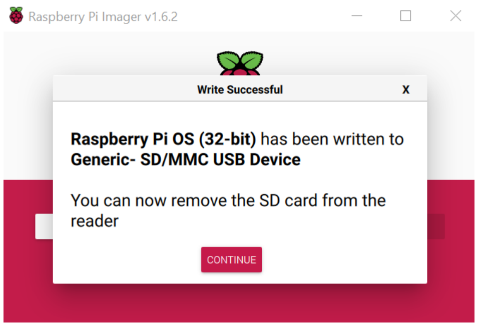


All existing data on 'Generic- SD/MMC USB Device' will be erased.
Are you sure you want to continue?









Raspberry Pi OS (32-bit) has been written to **Generic- SD/MMC USB Device**

You can now remove the SD card from the reader

(v) Plug the card into RPI, apply power to the RPI through the micro-USB cable. The RPI should boot up and displays messages on the local monitor. For Buster OS, the system may proceed to perform the update and upgrade procedure to download the latest version of the software libraries and drivers.

(vi) Depending on whether you have made any changes to the configuration settings during the installation, the default hostname, login, and password are as follows:

Hostname: raspberrypi
Username: **pi**
Password: **raspberry**

(You probably would want to change the **Hostname** to something that match your group, like **MDPGrpxx**)

(vii) You have a fully functional computer (i.e. RPI board with ARM CPU) running the Linux OS. Raspberry Pi OS is based on the Linux's Debian distribution. As such, students will need to be familiar with some basic Linux commands to operate the RPI system, which will be introduced as we move along in this guide.
Note: raspi-config can be subsequently executed from console using the Linux command: **sudo raspi-config** when needed. After you finish your changes to the raspi-config, you should reboot your RPI using the following Linux command: **sudo reboot**)

Appendix B: Script file to test Bluetooth connection

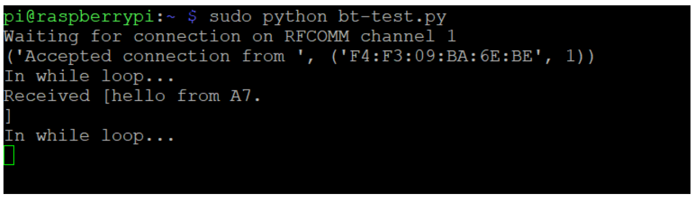
Create the following script file on RPI: **sudo nano bt-test.py**
This code is for the Server. You may need to install bluez

Python2.x: sudo apt install python-bluez
Python3.x: sudo apt install python3-bluez

The UUID is the service ID that the Client use to determine if the service is the one it is looking for. So this should match between the Server and Client.

```
# file: rfcomm-server.py
# auth: Albert Huang <albert@csail.mit.edu>
# desc: simple demonstration of a server application that uses RFCOMM sockets
#
# $Id: rfcomm-server.py 518 2007-08-10 07:20:07Z albert $


#!/usr/bin
from bluetooth import *
server_sock = BluetoothSocket(RFCOMM)
server_sock.bind(("",PORT_ANY))
server_sock.listen(1)
port = server_sock.getsockname()[1]
uuid = "94f39d29-7d6d-437d-973b-fba39e49d4ee"
advertise_service( server_sock, "MDP-Server",
                   service_id = uuid,
                   service_classes = [ uuid, SERIAL_PORT_CLASS ],
                   profiles = [ SERIAL_PORT_PROFILE ],
#                  protocols = [ OBEX_UUID ]
                  )
print("Waiting for connection on RFCOMM channel %d" % port)
client_sock, client_info = server_sock.accept()
print("Accepted connection from ", client_info)
try:
    while True:
        print ("In while loop...")
        data = client_sock.recv(1024)
        if len(data) == 0: break
        print("Received [%s]" % data)
        client_sock.send(data + " i am pi!")
except IOError:
    pass
print("disconnected")
client_sock.close()
server_sock.close()
print("all done")
```

Output when receiving a text string “hello from A7” from the A7 tablet.



```
pi@raspberrypi:~ $ sudo python bt-test.py
Waiting for connection on RFCOMM channel 1
('Accepted connection from ', ('F4:F3:09:BA:6E:BE', 1))
In while loop...
Received [hello from A7.
]
In while loop...
```

A sample corresponding Client-side code is as follows (**not verified**)

```
# file: rfcomm-client.py
# auth: Albert Huang <albert@csail.mit.edu>
# desc: simple demonstration of a client application that uses RFCOMM sockets
#       intended for use with rfcomm-server
#
# $Id: rfcomm-client.py 424 2006-08-24 03:35:54Z albert $

from bluetooth import *
import sys

addr = None

if len(sys.argv) < 2:
    print "no device specified.  Searching all nearby bluetooth devices for"
    print "the SampleServer service"
else:
    addr = sys.argv[1]
    print "Searching for SampleServer on %s" % addr

# search for the SampleServer service
uuid = "94f39d29-7d6d-437d-973b-fba39e49d4ee"
service_matches = find_service( uuid = uuid, address = addr )

if len(service_matches) == 0:
    print "couldn't find the SampleServer service =("
    sys.exit(0)

first_match = service_matches[0]
port = first_match["port"]
name = first_match["name"]
host = first_match["host"]

print "connecting to \"%s\" on %s" % (name, host)

# Create the client socket
sock=BluetoothSocket( RFCOMM )
sock.connect((host, port))

print "connected.  type stuff"
while True:
    data = raw_input()
    if len(data) == 0: break
    sock.send(data)

sock.close()
```

Appendix C: Miscellaneous notes on RPI system

The following is a collection of random notes on RPI. The information may or may not apply to Buster OS since there are updates to both hardware and software over the years.

The primary configuration file for the RPI is /boot/config.txt. Changes that you make using the raspi-config tool are reflected in this fi le. You can manually edit this file (e.g., **sudo nano /boot/config.txt**) to enable/disable bus hardware, overclock the processors, and so on.

the bootloader stages pass control to the kernel after it has been decompressed into memory. The kernel then mounts the root file system. The kernel's last step in the boot process is to call systemd init (/sbin/init on the RPI with Raspbian Jessie), which is the first user-space process that is started, and the next topic that is discussed.

The systemd System and Service Manager
A *system and service manager* starts and stops services (e.g., web servers, Secure Shell [SSH] server) depending on the current state of the RPI (e.g., starting up, shutting down). The *systemd* system and service manager is a recent and somewhat controversial addition to Linux that aims to replace, and remain backward compatible with *System V (SysV) init*. One major drawback of SysV init is that it starts tasks in series, waiting for one task to complete before beginning the next, which can lead to lengthy boot times. The systemd system is enabled by default in Debian 8/Raspbian 8 (Jessie). It starts up system services in parallel, helping to keep boot times short, particularly on multicore processors such as the RPI 2/3.

systemd uses *service fi les*, which have a .service extension to configure how the different services should behave on startup, shutdown, reload, and so on; see the /lib/systemd/system directory.

You can use the systemctl command to inspect and control the state of systemd. If called with no arguments, it provides a full list of the services that are running on the RPI (use the spacebar to page, and Q to quit

We used the systemctl services in the configuration of wifi and rfcomm. Refer to the corresponding sections above on systemctl can be used to automate certain configurations upon bootup.

There are tons of information on the internet related to the Raspberry Pi. Hence use Google to search for relevant information when you encounter problem or look for solutions.

Wifi:
https://www.raspberryconnect.com/projects/65-raspberrypi-hotspot-accesspoints/168-raspberry-pi-hotspot-access-point-dhcpcd-method

https://raspberrypi.stackexchange.com/questions/37920/how-do-i-set-up-networking-wifi-static-ip-address-on-raspbian-raspberry-pi-os

https://learn.pi-supply.com/make/how-to-setup-a-wireless-access-point-on-the-raspberry-pi/

https://thepi.io/how-to-use-your-raspberry-pi-as-a-wireless-access-point/

Bluetooth:
https://scribles.net/setting-up-bluetooth-serial-port-profile-on-raspberry-pi/

https://stackoverflow.com/questions/14618277/rfcomm-without-pairing-using-pybluez-on-debian

https://raspberrypi.stackexchange.com/questions/120665/python-import-bluetooth-module

https://people.csail.mit.edu/albert/bluez-intro/