

Raspberry Pi Based LCD Clock

Design Description

and

User's Manual

19-November-2025



Table of Contents

1	Introduction	3
1.1	List of Hardware Components	3
1.2	Location of Software Components	3
1.3	List of Software Files	4
2	Photographs of Hardware Components	5
2.1	Photographs of Prototype Unit	5
2.2	Photographs of Production Unit	6
3	Wiring Diagrams	7
4	Software Operational Overview	9
4.1	Software Processes and Communication Queues	9
4.2	Configuration File Setup	10
4.2.1	Accessing the RPi Desktop and Connecting to a Wi-Fi Network	10
4.2.2	Accessing the RPi Terminal Window	11
4.2.3	Editing the Configuration File with the Built-In Nano Editor.	11
4.2.4	Copying Files from the RPi to a Flash Drive	12
4.3	Starting the Server Manually	12
4.4	Starting the Server Automatically at Boot Time	13
4.5	Starting the Command Line Client	14
4.6	Starting the GUI Client	15
4.7	Starting the Clock	16
4.8	Memory Utilization	18
5	Appendix 1. A Python Based Client-Server Architecture	19
5.1	Introduction	19
5.2	A Well-Known Client Server	19
5.3	Closing a Client and Stopping the Server	19
5.4	Server's Handling of Unexpected Events	20
5.5	Connection Types	21
5.6	A Potential Pitfall – Firewall	22
5.7	Functional Call Tree	24
6	Appendix 2. SSH, SCP and other Handy RPi Commands	25
6.1	Installing Python3	25
6.2	Installing Python Packages	25
6.3	Enabling SSH Sessions	25
6.4	Establishing and Closing SSH Sessions:	26
6.5	Copying Files to/from an RPi Using SCP:	26
6.6	Setting up an SSH/SCP Pass Key:	26
6.7	Enabling NTP Synchronization:	27
6.8	Determining Currently Running Python Scripts:	27
6.9	Determining Open TCP Ports:	27

Table of Figures

Figure 1.	Functional Wiring Diagram	7
Figure 2.	Physical Wiring Diagram.....	8
Figure 3.	Communication Queues	9
Figure 4.	Raspberry Pi Connection Points.....	10
Figure 5.	Raspberry Desktop	10
Figure 6.	ifconfig Command Output Screenshot	11
Figure 7.	Example cfg.cfg File Screenshot	12
Figure 8.	Example ps Command Output Screenshot	12
Figure 9.	Crontab Screenshot.....	13
Figure 10.	Command Line Client Screenshot.....	14
Figure 11.	GUI Client Screenshot 1	15
Figure 12.	GUI Client Screenshot 2.....	15
Figure 13.	GUI Client Screenshot 3.....	16
Figure 14.	GUI Client Screenshot 4.....	17
Figure A15.	Connection Types	21
Figure A16.	Functional Call Tree	24

Table of Photographs

Photograph 1 - Prototype Front View	5
Photograph 2 - Prototype Back View 1	5
Photograph 3 - Prototype Back View 2	5
Photograph 4 - Production Front View	6
Photograph 5 - Production Back View 1	6
Photograph 6 - Production Back View 2	6

1 Introduction

This document explains the hardware and software for a Raspberry Pi clock with six 240x320 LCDs. After reading, you'll know how to build it and understand its operation.

1.1 List of Hardware Components

Hardware components, with Amazon links, are listed below.

Item	Link	Cost	Comment
1. Raspberry Pi 4B (RPi):	RPi	~ \$65	-
2. RPi SD card (OS):	SD_Card	~ \$14	Download SD OS imager here
3. RPi Power Supply:	PWR	~ \$10	-
4. LCD Displays:	LCD	~ \$15	Each, six required
5. Mini Breadboard:	MiniBB	~\$ 8	-
6. Breadboard Jumpers:	JMP	~\$13	Breadboard-to Breadboard
7. RPi Jumpers:	JMP2	~\$ 6	RPi-to-Breadboard

Total cost: ~\$206.

1.2 Location of Software Components

This document and (most of) the clock's software can be found here:

<https://github.com/sgarrow/spiClock>

Some of the clock's code is common with another, second, project. To prevent having to copy/paste changes/bug-fixes back and forth between projects, this common code resides in a third project. The common code can be found here:

<https://github.com/sgarrow/sharedClientServerCode>

Just as a matter of completeness the above-mentioned second project that also uses the shared code is an RPi sprinkler controller. Its documentation and (most of) its software can be found here:

<https://github.com/sgarrow/sprinkler2>

Both the clock and sprinkler run within a client/server architecture. It is the Client and Server files/functionality that are common.

1.3 List of Software Files

```

+---spiClock                                --+ ALL CODE RESIDES IN THIS ROOT DIRECTORY.
|   client.py                                | Only these files need to be on the client machine.
|   clientCustomize                          | The last 2 (cfg.*) also need to be on the server
|   gui.py                                    | machine (RPi). Cfg.cfg contains IP addresses, etc.,
|   cfg.py                                    | that need to be shared between the client/server
|   cfg.cfg                                    | Files below only need to be on the RPi.
|
|   server.py                                | Clients send commands, via a socket, to the server.
|   serverCustomize.py                      | The server looks up the command in a dictionary and
|   cmdVectors.py                          | vectors to the worker function. In the dictionary,
|                                           | the command string serves as the "key" and the
|                                           | associated "value" is the worker function.
|
|   startStopClock.py                      | Worker functions for the Clock Stop/Stop commands
|       clockProcess.py                    | reside in startStopClock.py. These functions spawn/
|       lcdProcess.py                      | terminate separate processes, running concurrently
|                                           | on separate cores, that increment the clock counter
|                                           | & push data out to the LCDs respectively.
|
|   makeScreen.py                          | Screens pushed to LCDs need to be made before the
|   styleMgmtRoutines.py                   | clock starts. Making a screen creates a file. When
|                                           | that file is loaded and pushed to an LCD it results
|                                           | in, e.g., a white 4 character being displayed on a
|                                           | black background. Management Routines allow for
|                                           | for changing styles, choosing a nighttime style,
|                                           | setting the times to automatically switch styles.
|
|   spiRoutines.py                         | These files contain the worker functions for the
|   swUpdate.py                             | remaining commands and various helper routines.
|   utils.py
|   cmds.py
|   testRoutines.py
|   fileIO.py
|   rpiShellCmds.py                        --+
|
+----digitScreenStyles                    --+ ALL SCREEN STYLES RESIDE IN THIS SUBDIRECTORY
|   blackOnWhite.pickle                    | blackOnWhite is the default and can't be deleted -
|   greyOnBlack.pickle                    | other styles can be. New styles can be created at
|   orangeOnTurquoise.pickle              | will. Each file contains an image for all digits.
|   turquoiseOnOrange.pickle              |
|   whiteOnBlack.pickle                    --+
|
+----fonts                                --+ ALL FONTS RESIDE IN THIS SUBDIRECTORY
|   Font00.ttf                             --+ This file contains the font used for all digits.
|
+----pics                                --+ ALL PICTURES RESIDE IN THIS SUBDIRECTORY
|   240x320b.jpg                          | Using the appropriate command the LCDs will
|   240x320c.jpg                          | momentarily display a set of six 240x320 jpg
|   240x320d.jpg                          --+ images.

```

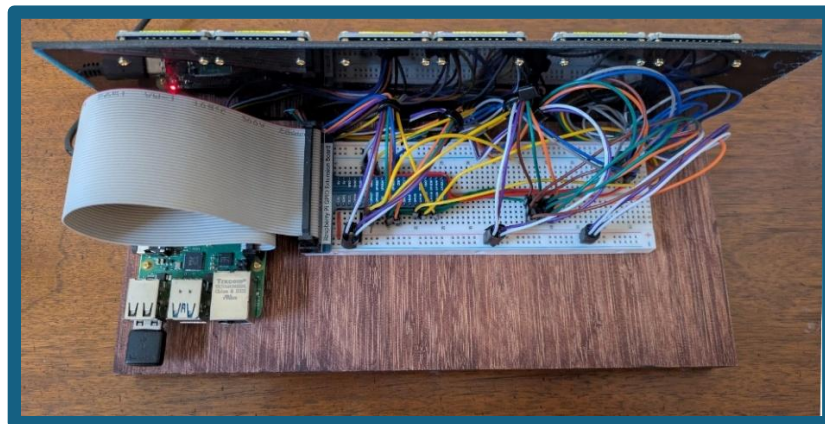
Note: the **bold underlined** files are in the shared GitHub Repository, the remaining files are in the clock repository.

2 Photographs of Hardware Components

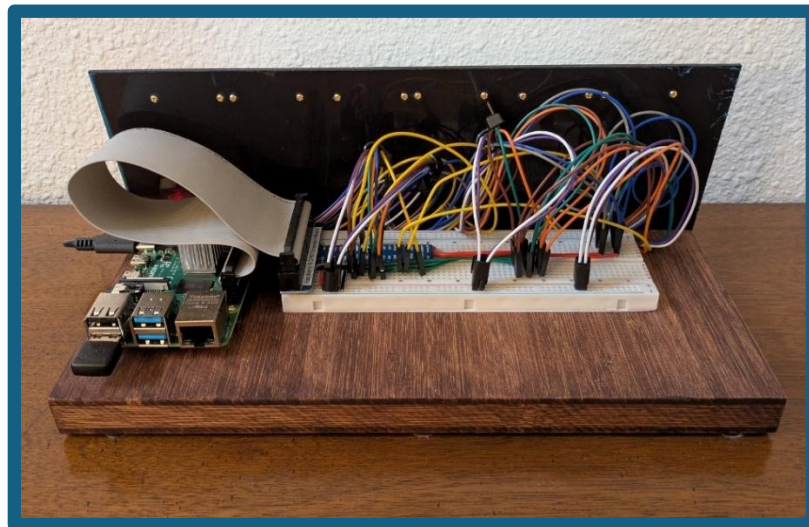
2.1 Photographs of Prototype Unit



Photograph 1 - Prototype Front View

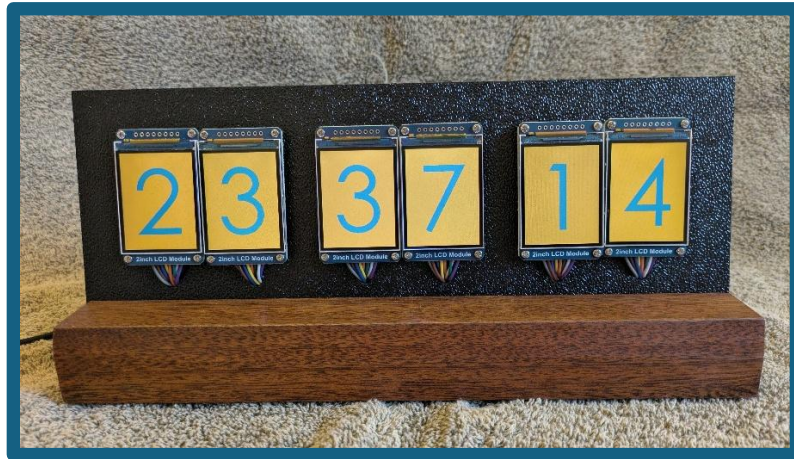


Photograph 2 - Prototype Back View 1

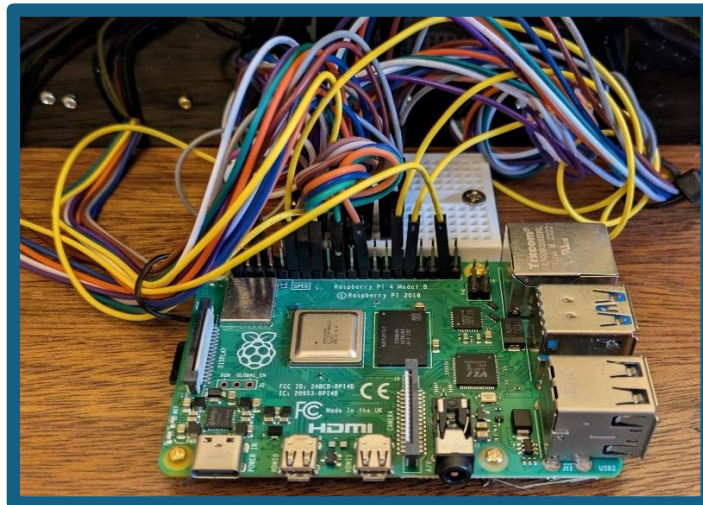


Photograph 3 - Prototype Back View 2

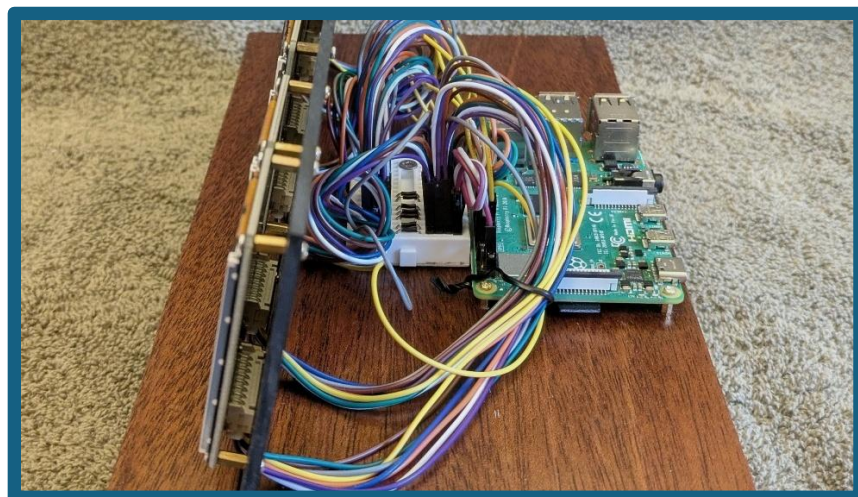
2.2 Photographs of Production Unit



Photograph 4 - Production Front View



Photograph 5 - Production Back View 1



Photograph 6 - Production Back View 2

3 Wiring Diagrams

The RPi talks to the LCDs over an SPI interface. An introduction to SPI can be found here:

https://en.wikipedia.org/wiki/Serial_Peripheral_Interface

Each LCD has eight pins, with seven shared among all units. For instance, pin 19 (Data In) from the RPi connects to every LCD, so data sent through it reaches all displays. However, only the LCD with a low Chip Select (CS) pin will respond. CS pins are unique per LCD, and the RPi sets only one LCD's CS line low at a time.

Functional and Physical wiring diagrams are provided in Figures 1 and 2, respectively.

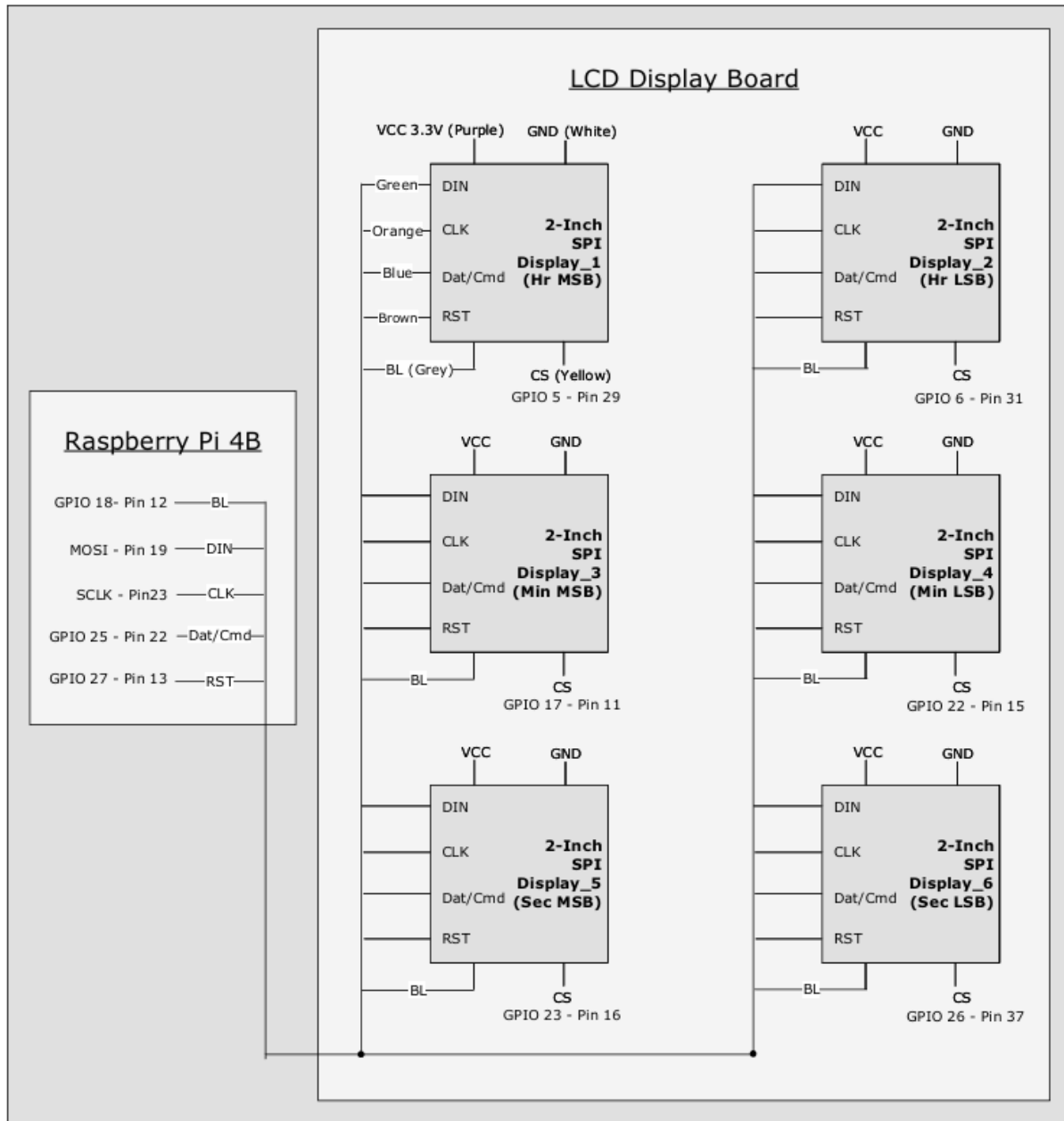


Figure 1. Functional Wiring Diagram

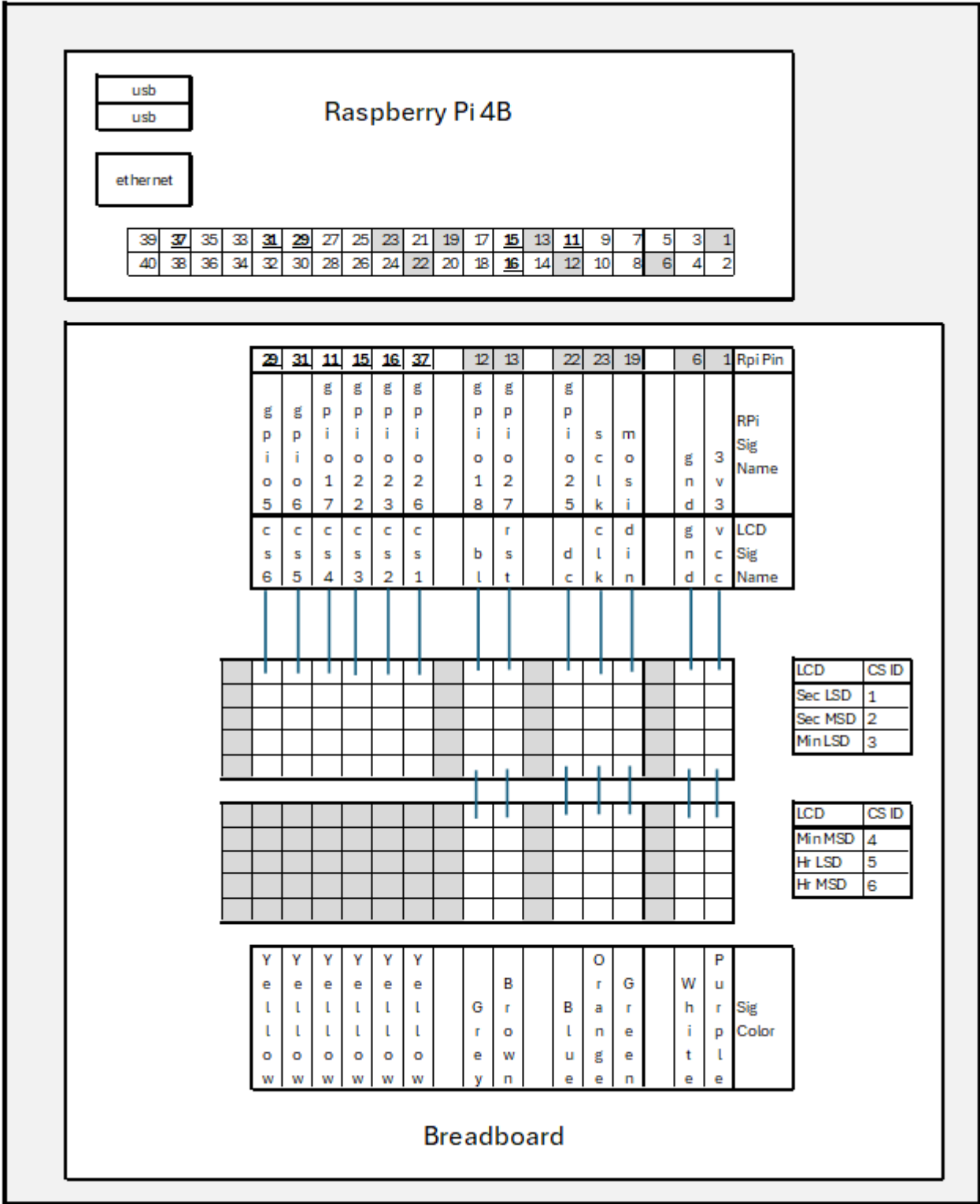


Figure 2. Physical Wiring Diagram

4 Software Operational Overview

The RPi runs two programs: a server and a clock. A remote client (on a PC or phone) connects via socket over LAN or the internet to send commands. The server relays these commands to the clock for control. See Appendix 1 for details on this implementation.

4.1 Software Processes and Communication Queues

The clock uses three cores: one for the server (Main Process), one for counting time (Clock Process), and one for managing displays (LCD Process). These processes interact via four communication queues—two for commands and two for responses. Figure 3 shows a simplified diagram of this setup.

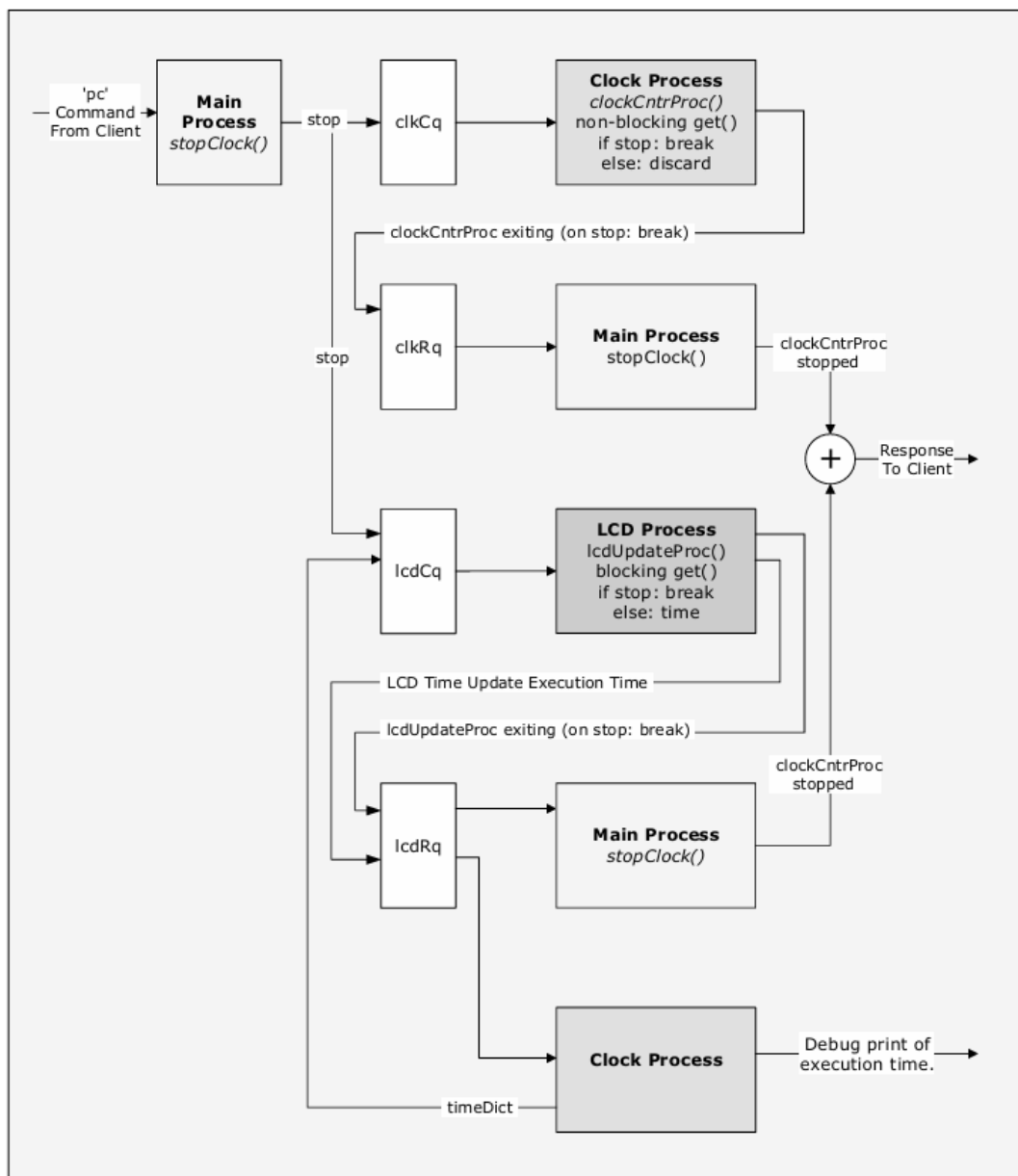


Figure 3. Communication Queues

4.2 Configuration File Setup

The client and server must share details like IP address and port numbers to communicate. Users need to enter this information in `cfg.cfg`.

4.2.1 Accessing the RPi Desktop and Connecting to a Wi-Fi Network

The file `cfg.cfg` is stored on the RPi's Solid-State Drive. To edit it, connect an HDMI display and keyboard/mouse to the RPi as shown in Figure 4, then power on the device and access the desktop.

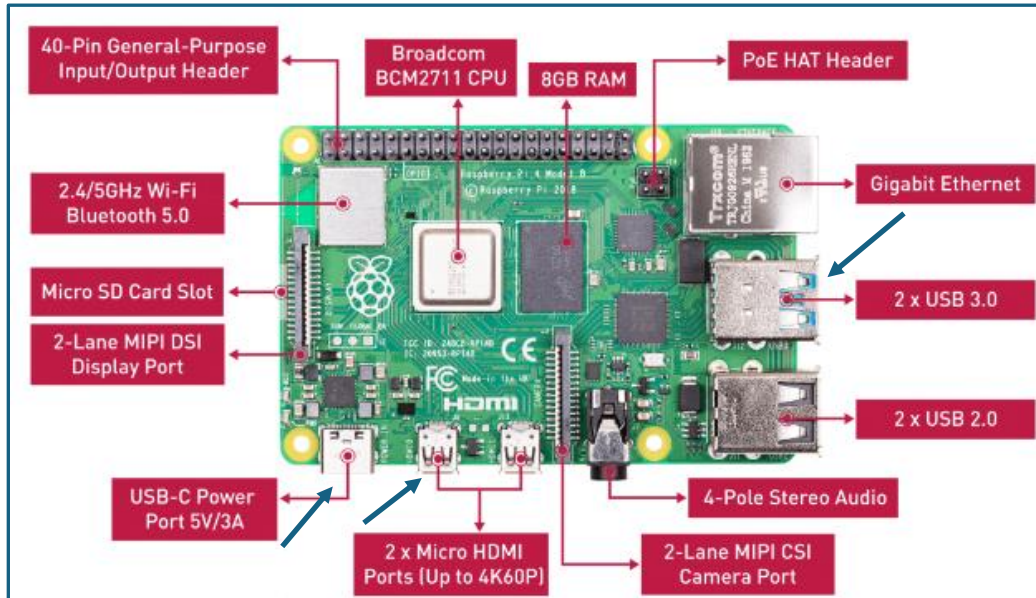


Figure 4. Raspberry Pi Connection Points

Once the display and mouse/keyboard are connected apply power via the USB-C connector. Once the RPi has booted the desktop shown in Figure 5 will be displayed.

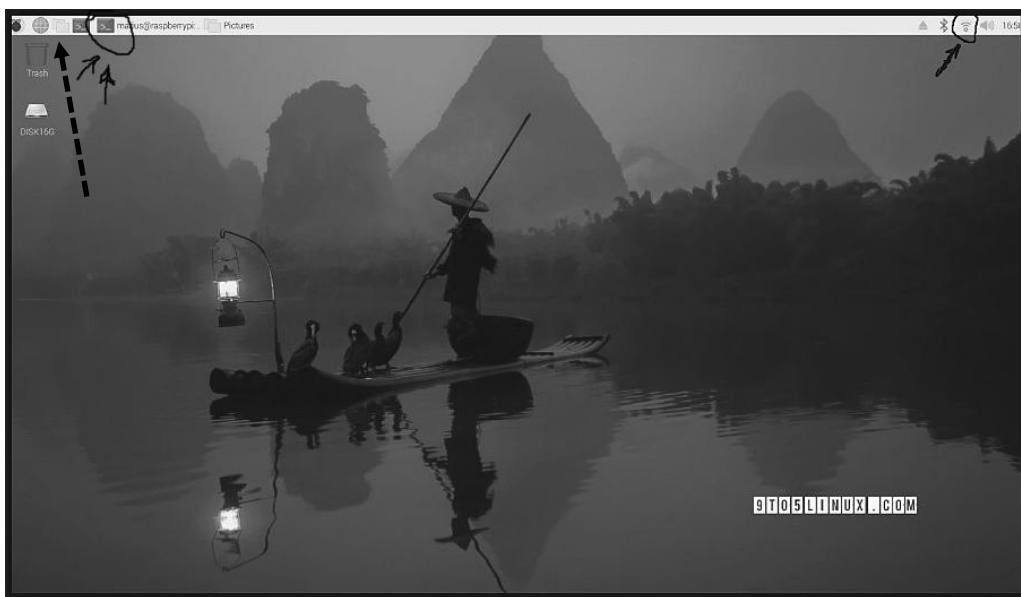
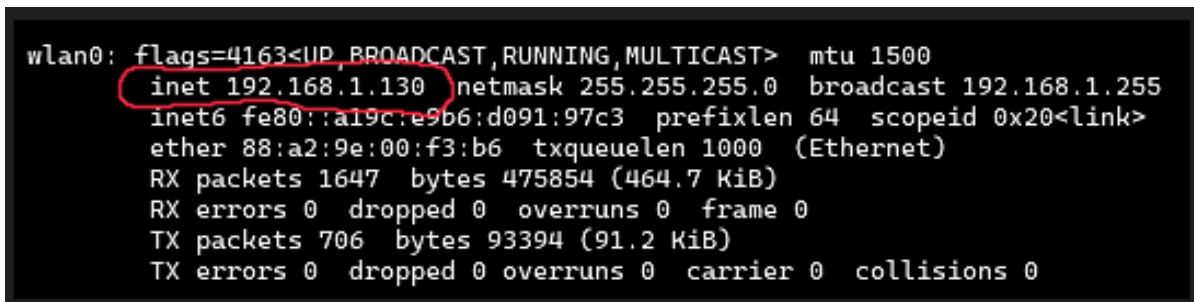


Figure 5. Raspberry Desktop

Click the Wi-Fi icon, designated above by a single arrow on the right-hand side, select the Wi-Fi network and enter the password. Henceforth the RPi will automatically connect to this network whenever it is available.

4.2.2 Accessing the RPi Terminal Window

After connecting to the LAN, open a terminal by clicking on the icon, designated above by the double arrow, and enter the command **ifconfig**. An example output of the ifconfig command is shown in Figure 6. Take note of the inet address as it will be added to cfg.cfg.



```
wlan0: flags=4163<UP, BROADCAST, RUNNING, MULTICAST> mtu 1500
    inet 192.168.1.130 netmask 255.255.255.0 broadcast 192.168.1.255
    inet6 fe80::a19c:e9b6:d091:97c3 prefixlen 64 scopeid 0x20<link>
    ether 88:a2:9e:00:f3:b6 txqueuelen 1000 (Ethernet)
    RX packets 1647 bytes 475854 (464.7 KiB)
    RX errors 0 dropped 0 overruns 0 frame 0
    TX packets 706 bytes 93394 (91.2 KiB)
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0
```

Figure 6. ifconfig Command Output Screenshot

4.2.3 Editing the Configuration File with the Built-In Nano Editor.

In a terminal window, enter the following command to navigate to the appropriate RPi directory:

```
cd python/spiClock
```

The following command opens cfg.cfg in the nano editor (mouse doesn't work use arrow keys):

```
nano -e cfg.cfg                      Note: An example cfg.cfg is provided in Figure 7.
```

Edit an existing line or add a new line. The RPi name will be entered when starting the server (on the RPi) and when starting the client (on a remote machine, either a PC or a phone).

Enter a port number. Ports between 8000 and 9000 are safe to use. Enter the Lan address obtained by the ifconfig command. Enter your router's IP address. The router IP address can be obtained via website www.whatismyip.com. Choose an enter your desired password.

Note that the password contained in the cfg.cfg file is the password that the client will send to the server when attempting to establish a connection. It is not the "user" password for the RPi itself. The RPi password for the RPi itself is used when, for example, when attempting, to establish an SSH connection to the RPi.

Save the file by entering the following commands:

```
ctrl-x                      # To begin the editor exiting process.
y                            # Answer to the "save-file" prompt.
<RETURN>                    # Answer to the "save as cfg.cfg" (default) prompt.
```

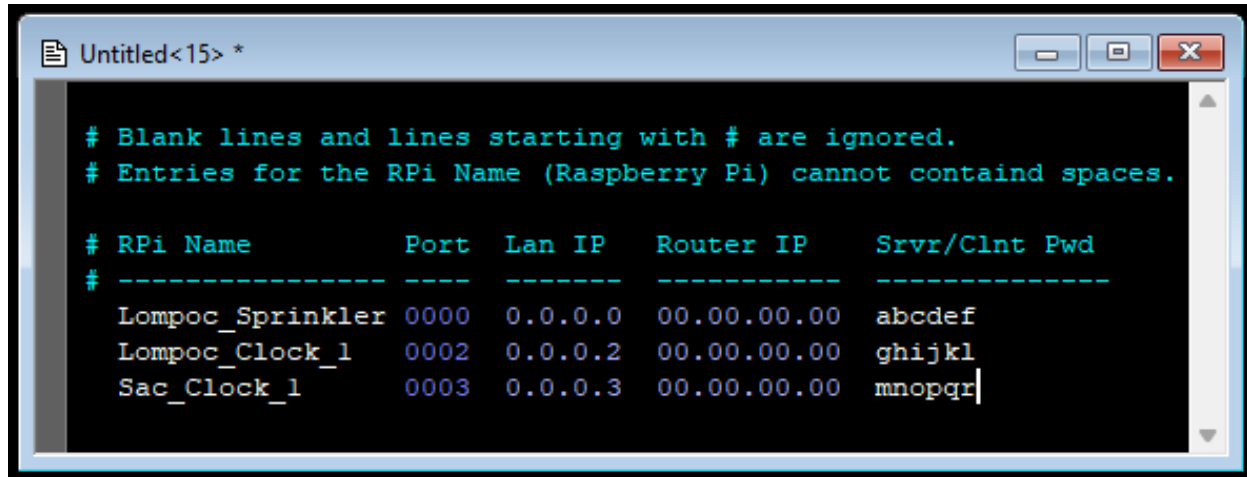


Figure 7. Example cfg.cfg File Screenshot

4.2.4 Copying Files from the RPi to a Flash Drive.

As mentioned in section 1.3, there are six files that need to be placed on the machine that will run the client. Insert a Flash Drive into a USB Port on the RPi and, using the RPi File Explorer (click on the File Explorer Icon designated by the dotted arrow in Figure 5. Raspberry Desktop), copy the following six files to it:

`client.py`, `clientCustomize.py`, `gui.py`, `cfg.py`, `cfg.cfg`

4.3 Starting the Server Manually

Start the server on the RPi by entering the following command in an RPi terminal window:

```
python3 server.py <RPi Name>
```

Replace <RPi Name> with the appropriate name from the `cfg.cfg` file.

Note that the RPi may be configured to start the server automatically at boot time. If so, attempting to start the server manually will not work – an error message displayed. If this error occurs, then the server must be terminated by either using the “kill” command or by running the client directly on the RPi.

To use the kill command the Process IDs (pid) of the server must first be determined. To determine the pids issue the following command in an RPi terminal:

```
ps aux | grep python
```

An example output of this command is shown in Figure 8, below.

```
pi@rasp3:~ $ ps aux |grep python
pi          584  0.0  0.1  2324  1396 ?        Ss   01:58   0:00 /bin/sh -c /bin/sleep 30; cd /home/
pi/python/  python3 server.py Sac_Clock_1 >>serverLog.txt 2>>serverException.txt
pi          1638  9.2  4.6 216748 43064 ?        Sl   01:58   0:01 python3 server.py Sac_Clock_1
pi          1661  0.0  3.6 216760 34140 ?        Sl   01:58   0:00 python3 server.py Sac_Clock_1
pi          1685  0.0  0.2  6092   2016 pts/0    S+   01:58   0:00 grep --color=auto python
pi@rasp3:~ $
```

Figure 8. Example ps Command Output Screenshot

Note there are two lines associated with server.py (their pid are 1638 and 1661). There are two (or possibly more) pids because of the multi-processing nature of the server/clock – both processes must be killed. Kill the processes with the following two commands (substitute the pids with those shown on the actual output):

```
kill -9 1638
kill -9 1661
```

To terminate the server with the client instead issue the following command in the appropriate directory (probably /python/spiClock, use the cd cmd to navigate there) in an RPi terminal window:

```
python3 client.py Sac_Clock_1
```

Replace **Sac_Clock_1** as appropriate from the output of the ps command. Once the client connects issue the “ks” command.

4.4 Starting the Server Automatically at Boot Time

Cron is a shell command for scheduling a job (i.e. command or shell script) to run periodically at a fixed time, date, interval, or at boot-time.

Open the cron scheduling file for editing on the RPi enter the following command in a terminal window:

```
crontab -e
```

Add the line shown at the bottom of the screenshot provided in Figure 9 and then save the file by entering the following commands:

```
ctrl-x      # To begin the editor exiting process.
Y           # Answer to the “save-file” prompt.
<RETURN>    # Answer to the “save as” prompt.
```

Cycle power to the RPi and the clock will start after about 30 seconds. The 30 sleep is required to ensure that the RPi has completely booted (including connecting to the Wi-Fi) before the server script is started.

```

pi@rasp3: ~
GNU nano 7.2 /tmp/crontab.pd7Tlw/crontab
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow  command
@reboot /bin/sleep 30; cd /home/pi/python/spiClock && python3 server.py Sac_Clock_1 >>serverLog.txt 2>>serverException.txt
^G Help      ^O Write Out ^W Where Is  ^K Cut       ^T Execute  ^C Location  M-U Undo    M-A Set Mark
^X Exit      ^R Read File ^_ Replace   ^U Paste     ^J Justify  ^/_ Go To Line M-E Redo    M-G Copy

```

Figure 9. Crontab Screenshot

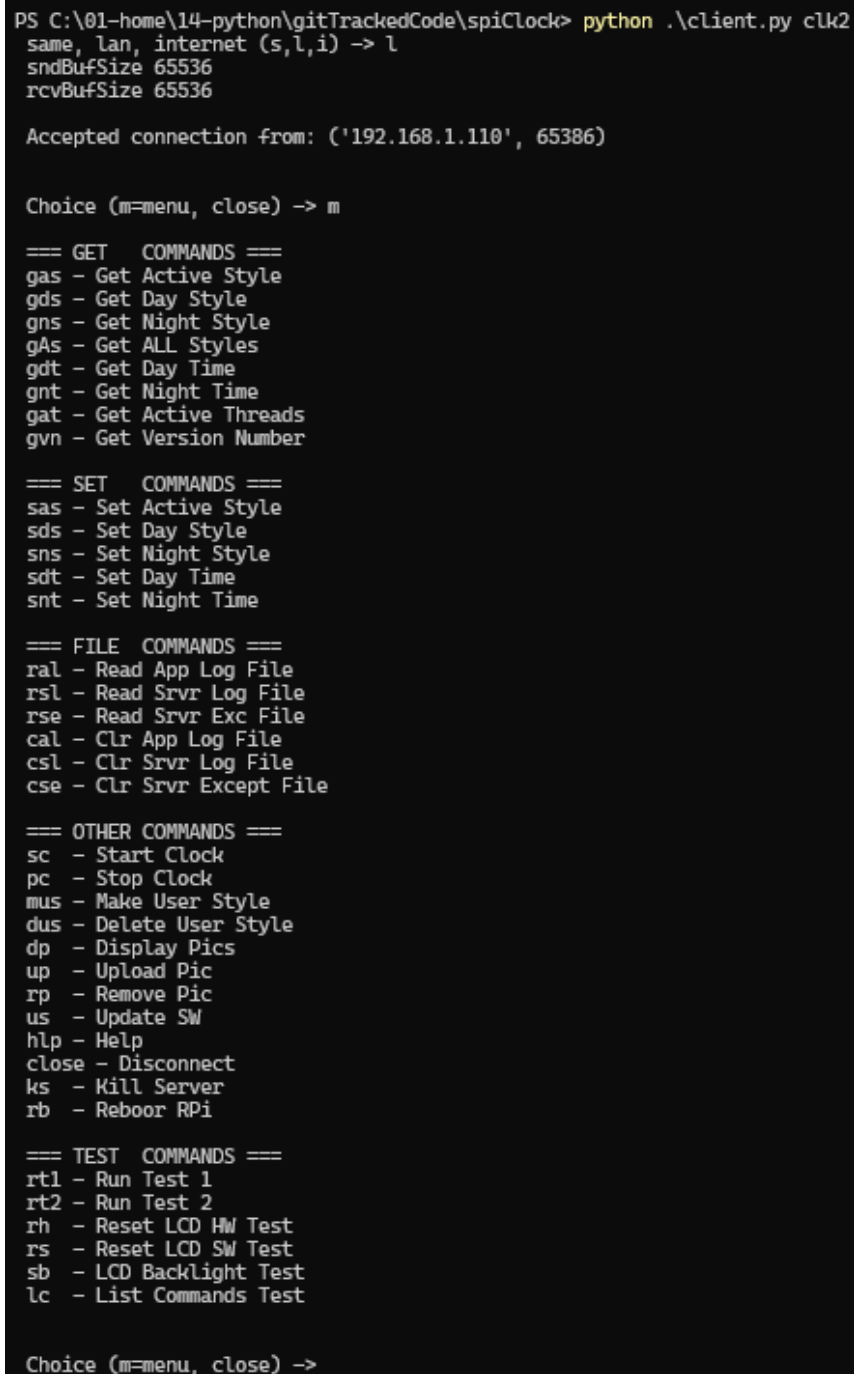
4.5 Starting the Command Line Client

Start the command line client on a remote machine by entering the following command:

```
python client.py <id>
```

Replace <id> as appropriate, see Section “Configuration File”, below

Once the server accepts the connection a prompt is presented. Enter ‘m’ at the prompt to get a list of available commands. A screen shot of this is provided in Figure 10, below.



```
PS C:\01-home\14-python\gitTrackedCode\spiClock> python .\client.py clk2
same, lan, internet (s,l,i) -> l
sndBufSize 65536
rcvBufSize 65536

Accepted connection from: ('192.168.1.110', 65386)

Choice (m=menu, close) -> m

=== GET  COMMANDS ===
gas - Get Active Style
gds - Get Day Style
gns - Get Night Style
gAs - Get ALL Styles
gdt - Get Day Time
gnt - Get Night Time
gat - Get Active Threads
gvn - Get Version Number

=== SET  COMMANDS ===
sas - Set Active Style
sds - Set Day Style
sns - Set Night Style
sdt - Set Day Time
snt - Set Night Time

=== FILE COMMANDS ===
ral - Read App Log File
rsl - Read Srvr Log File
rse - Read Srvr Exc File
cal - Clr App Log File
csl - Clr Srvr Log File
cse - Clr Srvr Except File

=== OTHER COMMANDS ===
sc  - Start Clock
pc  - Stop Clock
mus - Make User Style
dus - Delete User Style
dp  - Display Pics
up  - Upload Pic
rp  - Remove Pic
us  - Update SW
hlp - Help
close - Disconnect
ks  - Kill Server
rb  - Reboor RPi

=== TEST  COMMANDS ===
rtl - Run Test 1
rt2 - Run Test 2
rh  - Reset LCD HW Test
rs  - Reset LCD SW Test
sb  - LCD Backlight Test
lc  - List Commands Test

Choice (m=menu, close) ->
```

Figure 10. Command Line Client Screenshot

4.6 Starting the GUI Client

Start the GUI client with the following command:

```
python gui.py.
```

When the screen shown in Figure 11 is displayed, click on the appropriate UUT (this is the equivalent of the <id> parameter specified when starting the Command Line Client). Note that the list of available UUTs will be those listed in the `cfg.cfg` file.

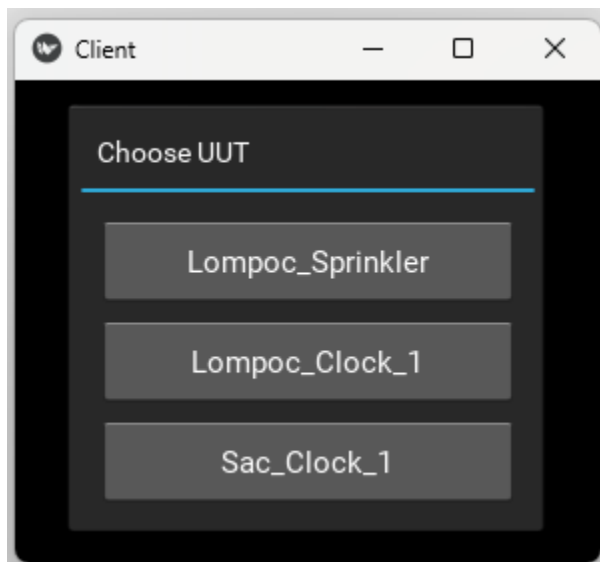


Figure 11. GUI Client Screenshot 1

Once a UUT is selected the screen shown in Figure 12 will be displayed. Select the desired/appropriate connection type.

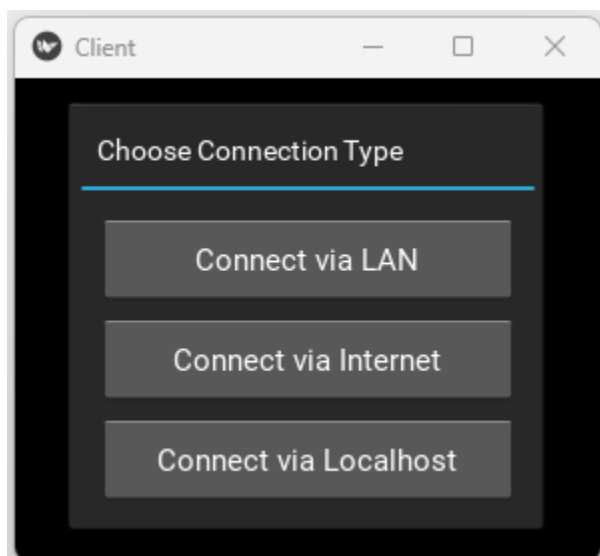


Figure 12. GUI Client Screenshot 2

When the GUI client is started and a connection is accepted by the server the 'm' command is automatically issued. The response is used to populate the GUI's buttons (GUI is 'built' at run time). A screen shot of the GUI is provided in Figure 13, below.

The 'build at run time' architecture is what allows the sprinkler and clock applications to use the same GUI.

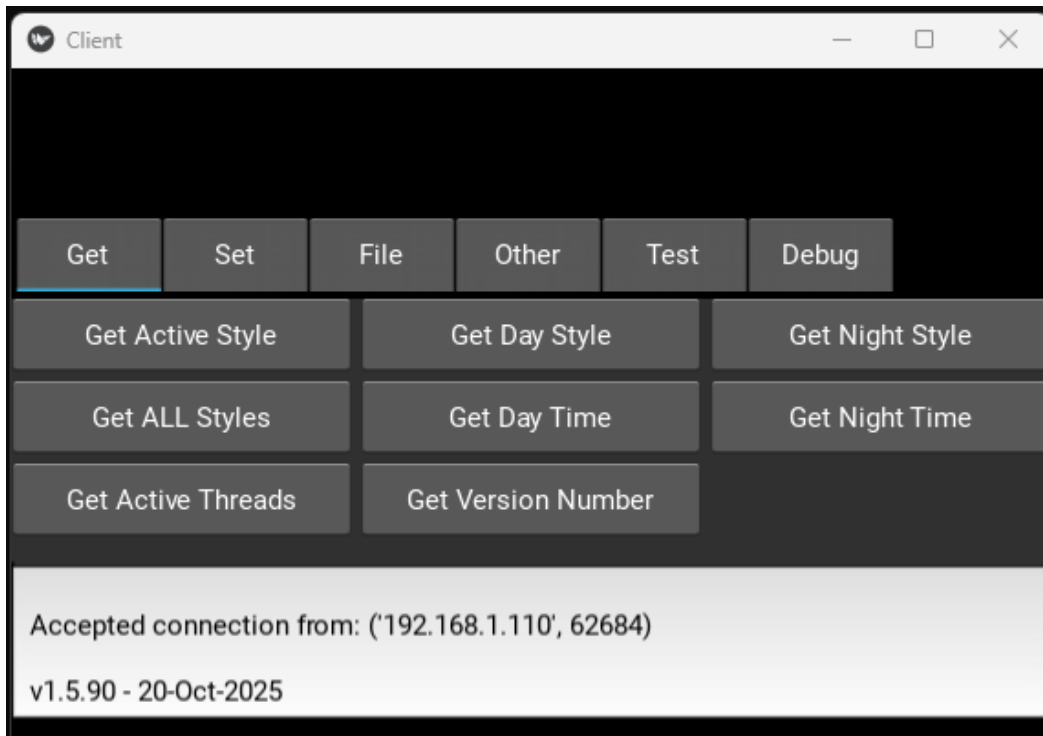


Figure 13. GUI Client Screenshot 3

4.7 Starting the Clock

If using the command line client the clock can be started with the "sc" command. There are 3 possible usages of this command:

```
Usage type 1:      sc
Usage type 2:      sc 121314
Usage type 3:      sc 121314 x
```

Type 1 starts the clock immediately using as an initial time a value obtained by a call to a native python function `datetime.now`.

Type 2 delays starting the clock until `datetime.now()` returns the time specified. In the above example that time is 12 13 14 but any time can be specified. If an invalid start time is specified (e.g., 12 13 65 or 12 13 ab) then a start time of 23 59 59 is used.

Types 1 and 2 are essentially alike in practice; the main difference is that type 1's start time may be off by up to one second if `datetime.now()` is called right before the second changes.

In type 2 a call is made to `.now()` every 0.2 seconds until a match is detected to the value it returns and the specified time.

Thus, the max start time error between type 1 and 2 are 1 sec and 0.2 sec, respectively.

Type 3 starts the clock at the exact time specified, regardless of the actual time. If an invalid start time is given, it defaults to 23:59:59. This is useful when the Raspberry Pi is not connected to a wireless LAN. More details follow below.

The RPi does not have a Real Time Clock (RTC).

At boot time, assuming no internet connection, the RPi starts its internal, SW based, time tracking function (that `datetime.now()` accesses) with a start time that is equal to the time it saved on its last power off event.

If the RPi is on and set to the correct time without internet, it counts time accurately—after one hour, 1 o'clock becomes 2 o'clock. However, if it's turned off for four hours, when restarted, it will still read 2 o'clock instead of the actual 6 o'clock.

If an internet connection IS available at boot time, then the RPi starts its internal time tracking function with an initial value obtained from the internet.

Note that this spiClock's SW only accesses the RPi's internal time tracking function (via the call to `datetime.now()`) at clock start time (when the `sc` command is first entered). After that this clock keeps track of the time all on its own using the python `sleep` (1 second) native internal function.

The sleep function varies by about ± 1 millisecond from the target time, potentially causing errors of up to 1.5 minutes daily. To address this, a high-precision counter measures actual sleep duration and the nominal sleep time is adjusted accordingly. This adjustment is made every 20 seconds.

If using the GUI, just enter the optional parameter in the text box before clicking the “Start Clock” button as shown in Figure 14, below.

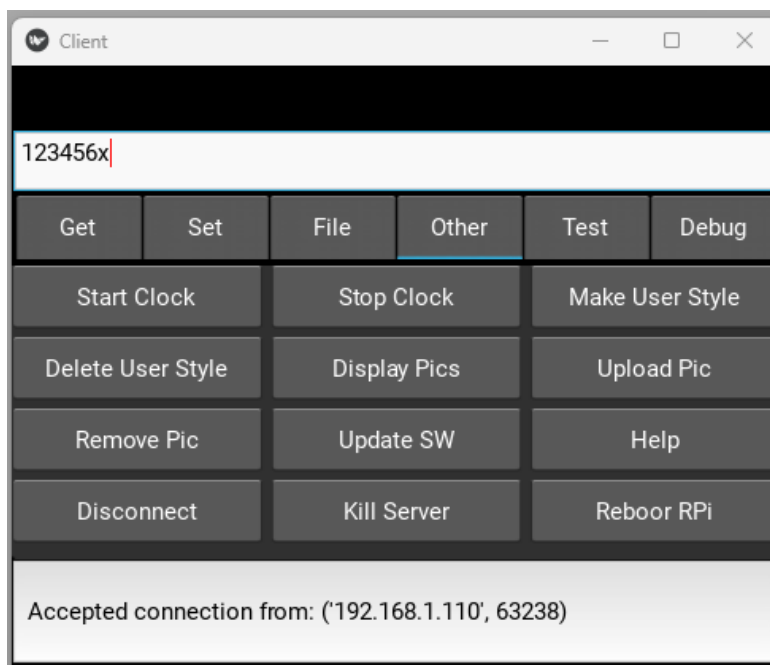


Figure 14. GUI Client Screenshot 4

4.8 Memory Utilization

The Amazon Link provided earlier for the RPi is a link to a 4GB (RAM) version of the RPi. However, the application can run on a 1GB version. Below is a screen capture of the output of four issuances of the `free --mega` Linux command. The first command was issued right after boot where only the OS is running, the second after the server was started, the third after a client was connected and the fourth after the clock was started.

The data shows that after everything is up and running there is still over 76MB of free RAM. Plenty for dozens of more clients to connect.

```
pi@rasp3:~ $ free --mega    # After Boot
```

	total	used	free	shared	buff/cache	available
Mem:	<u>950</u>	359	<u>152</u>	24	520	591
Swap:	536	0	536			

```
pi@rasp3:~ $ free --mega    # After Starting Server
```

	total	used	free	shared	buff/cache	available
Mem:	950	389	<u>100</u>	24	542	560
Swap:	536	0	536			

```
pi@rasp3:~ $ free --mega    # After Connecting Client
```

	total	used	free	shared	buff/cache	available
Mem:	950	387	<u>102</u>	24	542	562
Swap:	536	0	536			

```
pi@rasp3:~ $ free --mega    # After Starting Clock
```

	total	used	free	shared	buff/cache	available
Mem:	950	410	<u>76</u>	24	545	539
Swap:	536	0	536			

```
pi@rasp3:~ $
```

5 Appendix 1. A Python Based Client-Server Architecture

5.1 Introduction

This client-server architecture is written in the Python programming language. A client-server architecture is a structure where multiple clients request services from a centralized server and separates user interaction from data processing.

To start the server type "python server.py <device>" on the command line.

To connect a command line client to the server type "python client.py <device>" on the command line.

To connect a GUI client to the server type "python gui.py" on the command line.

A client can be run on the same machine as the server (in a different command window) or on a different machine entirely.

5.2 A Well-Known Client Server

Servers are things that respond to requests. Clients are things that make requests.

A web browser is a type of client that can connect to servers that "serve" web pages - like the Google server.

When a web browser (a client) connects to the Google Server and sends it a request (e.g., send me a web page containing a bunch of links related to "I'm searching this") the Google Server will respond to the request by sending back a web page.

Requests are sent in "packets" over connections called "sockets". Included in the request is the IP address of the client making it - that's how the server knows where to send the response back to. A given machine has one IP address, so if more than one instance of a web browser is open on a single machine how is it that the response ends up in the "right" web browser and not the other browser? Port number.

5.3 Closing a Client and Stopping the Server

Every client has a unique (IP, port) tuple. The server tracks every client by (IP, port). The server maintains a list of (IP, port) for all active clients.

Each client has two unique things associated with it - (1) a socket and (2) an instance (a thread) running the client's handling function

When a client issues a "close" command, its (IP, port) is removed from the list and as a result the `handleClient`'s infinite loop is exited thereby causing its socket to be closed and its thread to terminate.

When a client issues a "ks" (kill server) command, not only does that client terminate but all other clients terminate as well. Furthermore the "ks" command causes the server itself (it's still waiting for other clients to possibly connect) to terminate.

Upon receipt of the ks command the server (1) sends a message to all clients (including the one sent the command) indicating that the server is shutting down so that the client will exit gracefully, (2) terminates all clients and then finally (3) the server itself exits.

Additional details related to client connection types and to function calling sequences are provided in figures 1 and 2.

5.4 Server's Handling of Unexpected Events

If a user clicks the red X in the client window (closes the window) that client unexpectedly (from the server's viewpoint) terminates. This contrasts with the client issuing the close or ks command where the server is explicitly notified of the client's termination. An unexpected termination results in a sort of unattached thread and socket that may continue to exist even when the server exits. This situation is rectified by two try/except blocks in function `handleClient`. Two are needed because it was empirically determined the Window and Linux systems seem to block (waiting for a command from the associated client) in different places.

5.5 Connection Types

Clients can connect to the sever under three different scenarios as described in Figure A15.

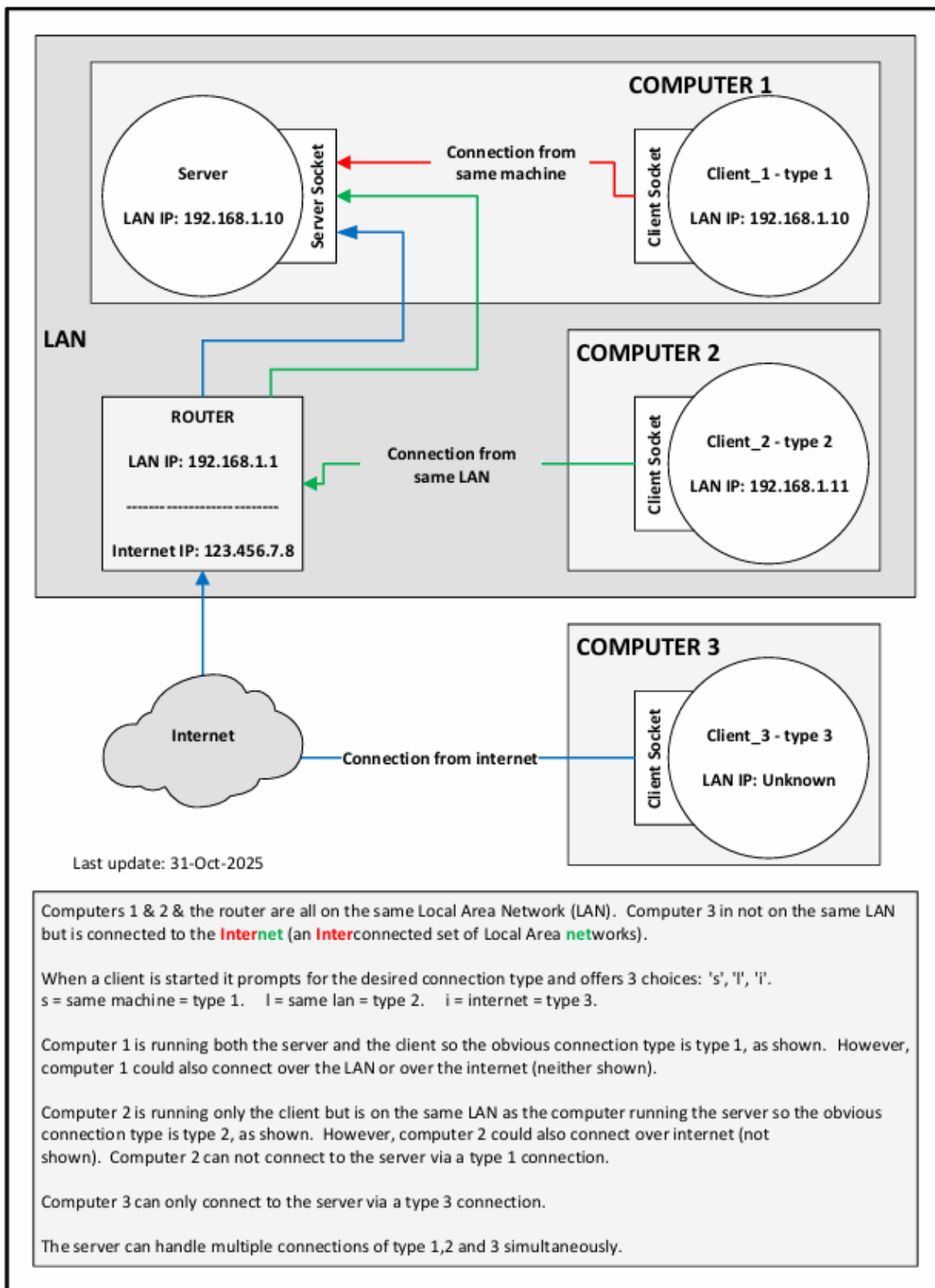


Figure A15. Connection Types

For connection type 2, in addition to the port number, the IP of the server needs to be known. The address can be found via the ifconfig command in a command window open on the machine that will be running the server.

For connection type 3, in addition to the port number, the external IP of the router needs to be known. The router's external IP address can be found using by going to the following web page on a browser: <https://whatismyipaddress.com/>

The use of connection type 3 also requires port forwarding to be set up on the router. An example is shown below. The example shows forwarding port 1234 (substitute 1234 with whatever port is configured) to port 1234 for IP address 192.168.1.10. Substitute 192.168.1.10 with whatever the IP address of the machine running the server is. Again, this address can be obtained via use of the ipconfig command. Since only one port number needs to be forwarded the start and end port numbers are the same.

It seems weird that a port number needs to be forwarded to that same number, but it does.

Service Name	External Start Port	External End Port	Internal Start Port	Internal End Port	Internal IP address
My Service Name	1234	1234	1234	1234	192.168.1.10

5.6 A Potential Pitfall – Firewall

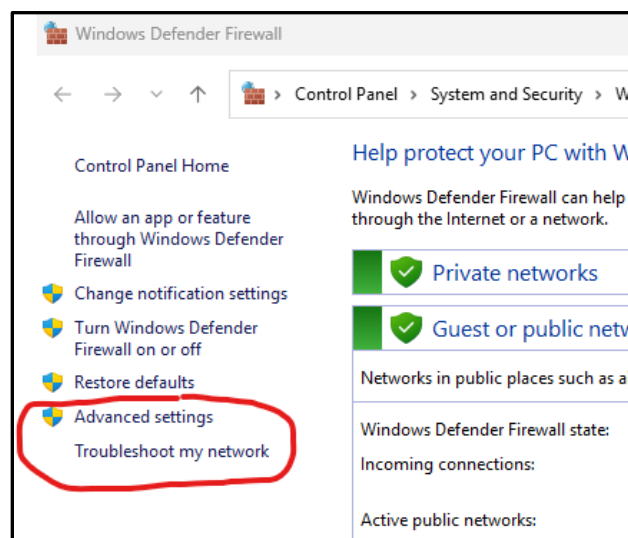
The above was all initially done with the server running on a Raspberry Pi. The Raspberry Pi runs Linux and by default its firewall is disabled. As such, all connection types worked only by performing the "SOME ASSEMBLY REQUIRED" steps outlined above.

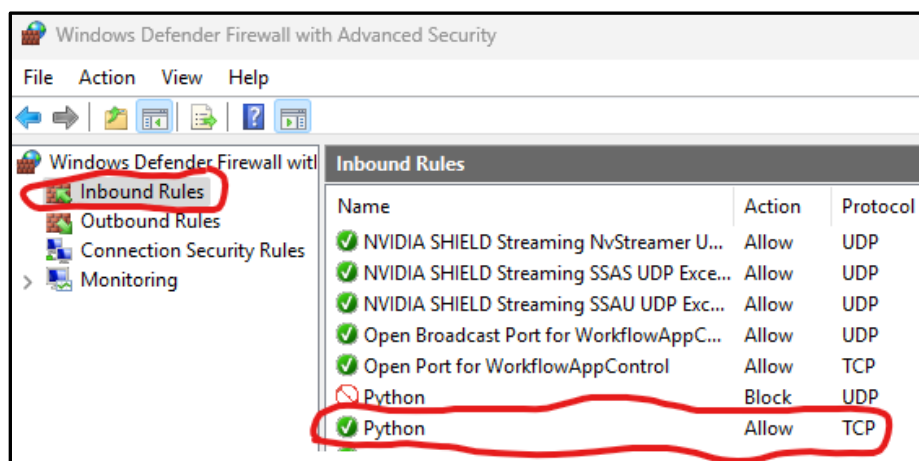
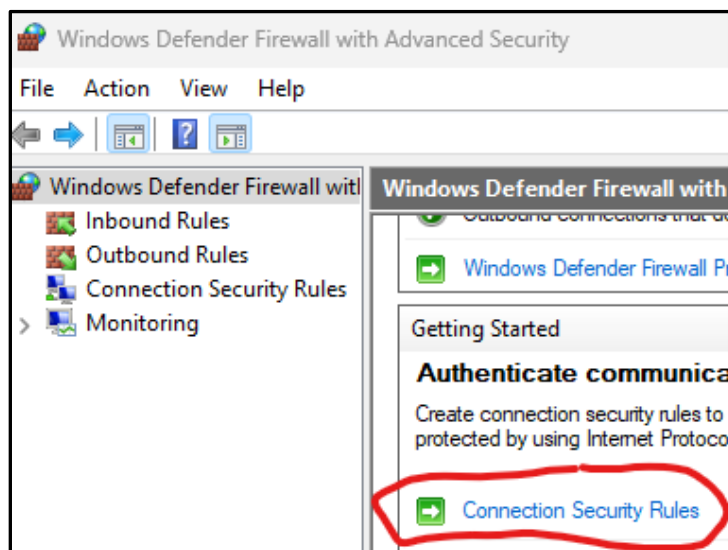
On windows to get all connection types working, specifically connection type 3, the Windows firewall will need to be changed to allow incoming python TCP connections.

These are the basic steps:

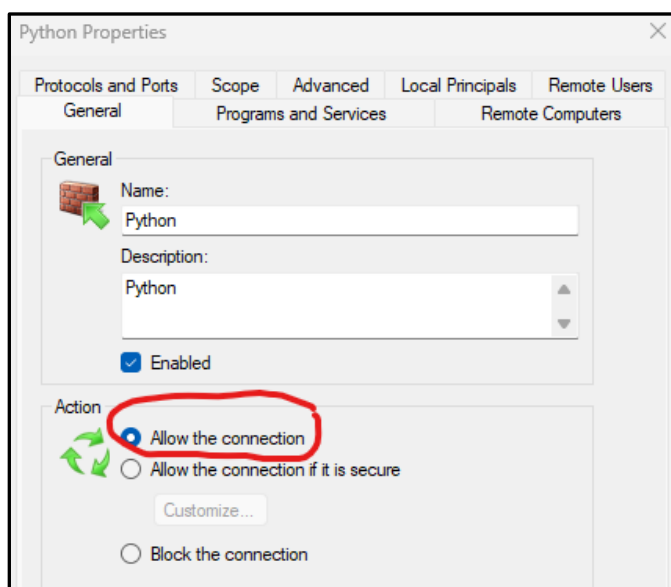
Control Panel\System and Security\Windows Defender Firewall ---> Advanced settings ---> Connection Security Rules ---> Inbound Rules

Screen shots for these various steps are provided below.





Right click on the Incoming Rule for Python TCP Protocol, select the General Tab and change to Allow the connection.



5.7 Functional Call Tree

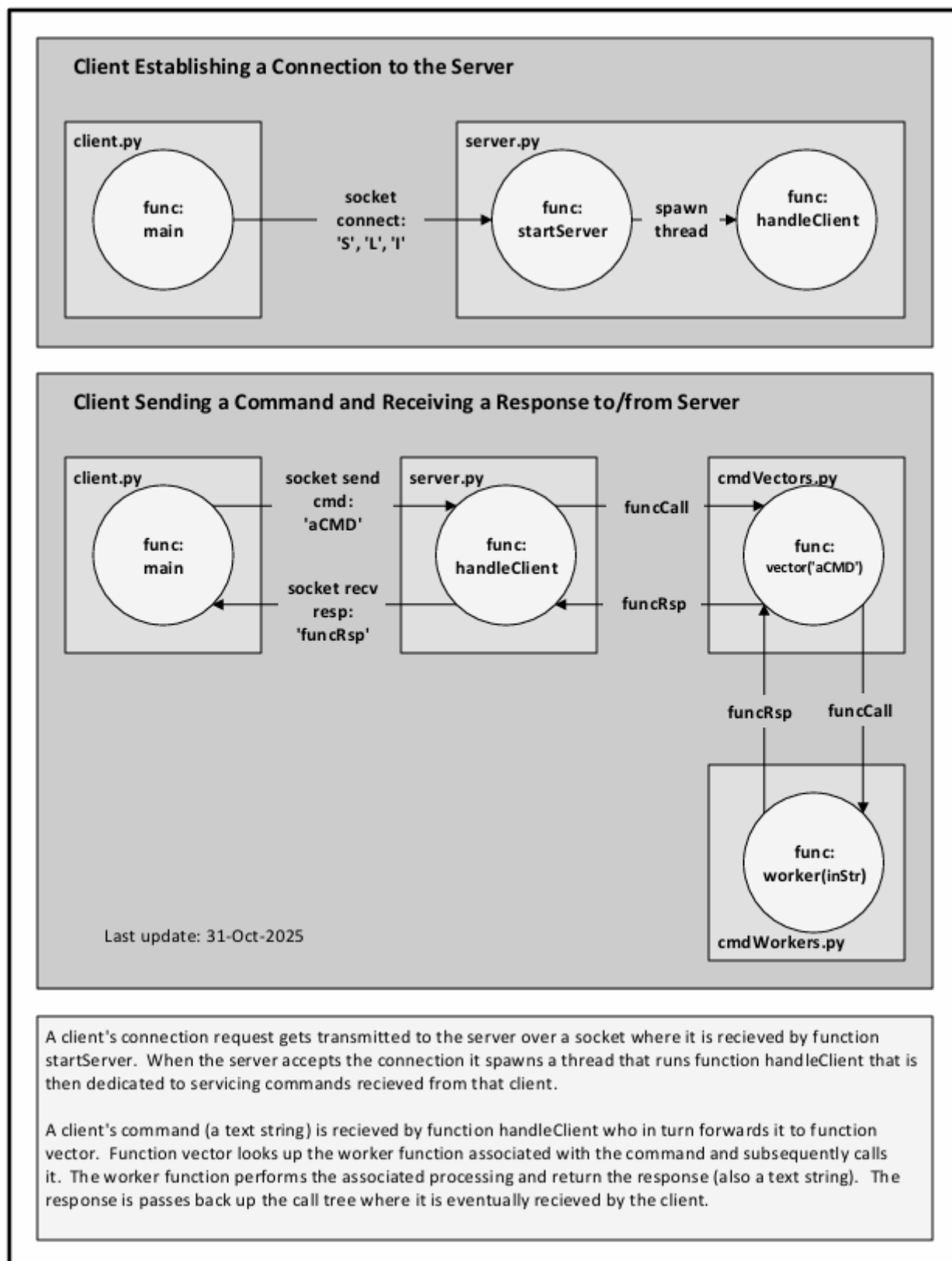


Figure A16. Functional Call Tree

6 Appendix 2. SSH, SCP and other Handy RPi Commands

6.1 Installing Python3

Most Linux distributions, including the Raspberry Pi distribution, come with python 2, not python 3 installed. To install python 3 on the Raspberry Pi, issue the following commands in a terminal window on the RPi:

```
Sudo apt update
sudo apt install python3 idle3
```

6.2 Installing Python Packages

Most python packages are installed using “pip”. yaml (Yet Another Markup Language) is an exception. To install yaml use the following command:

```
sudo apt install python3-yaml
```

To upgrade pip (Pip Installs Packages) to the latest version issue the following command:

```
python3 -m pip install --upgrade pip
```

To install a typical python package using pip use the following command:

```
pip install <package_name>
```

6.3 Enabling SSH Sessions

Opening a terminal on your Raspberry Pi is simple, but you need to have a monitor, keyboard, and mouse connected directly to it. If your Raspberry Pi and a remote PC are both on the same local network—or connected with an Ethernet cable—you can access a Raspberry Pi terminal window on your PC using SSH (Secure Shell). By default, SSH is disabled. To enable it, follow the steps below. Once SSH is enabled, it will stay enabled even after restarting the device.

- Connect:** Your Raspberry Pi to a monitor, keyboard and mouse.
- Open:** The Raspberry Pi Configuration tool: Click the Raspberry Pi icon (menu) > Preferences > Raspberry Pi Configuration.
- Navigate:** To the "Interfaces" tab.
- Enable:** SSH: Select "Enabled" next to the SSH option.
- Click:** "OK" to save the changes.
- Reboot:** The Raspberry Pi for the changes to take effect.

6.4 Establishing and Closing SSH Sessions:

The command to start an SSH session varies by connection type—Ethernet or LAN. For LAN, use the RPi's IP address, which you can find by running `ifconfig`` on a terminal window directly on the RPi with a monitor, keyboard, and mouse.

Establishing a connection over ethernet cable:	<code>ssh pi@raspberrypi.local</code>
Establishing a connection over LAN:	<code>ssh pi@12.34.56.78</code>
Closing a connection:	<code>~.</code>

6.5 Copying Files to/from an RPi Using SCP:

SCP (Secure Copy Protocol) is a command-line utility used for secure file transfers over a network. Again, use the RPi's actual IP address in the command.

Copy a specified file from PCs current directory to a specified RPi directory:
<code>scp .\myFile.py pi@12.34.56.78:~/python/spiClock</code>
Copy all .py files from PCs current directory to a specified RPi directory:
<code>scp *.py pi@12.34.56.78:~/python/spiClock</code>
Copy a specified file from the RPi PCs to the PC's current directory (.):
<code>scp pi@192.168.1.130:~/python/spiClock/pics/240x320a.jpg .</code>

6.6 Setting up an SSH/SCP Pass Key:

SSH and SCP commands prompt for a password each time, which can be inconvenient. Setting up a passkey eliminates the need to enter a password. Below is the setup procedure.

First, on the PC, create a key and copy it to RPi (replace stang with appropriate username and use correct IP address of the RPi):

```
ssh-keygen
scp -v "C:\Users\stang\.ssh\*.pub" pi@192.168.1.120:~/.
```

Next, on the RPi put the key in the right place:

```
mkdir -p ~/.ssh
chmod 700 ~/.ssh
mv ~/id_ed25519.pub ~/.ssh/
cat ~/.ssh/id_ed25519.pub >> ~/.ssh/authorized_keys
chmod 600 ~/.ssh/authorized_keys
rm ~/.ssh/id_ed25519.pub # Clean up (optional)
```

Finally, restart SSH on the RPi:

```
sudo systemctl restart ssh
```

Now when SSH-ing you will be prompted for the passphrase you used when you created the key so essentially, we're no better off than before. To not have to enter that every time follow the procedure provided below.

Run PowerShell as Administrator

Click on the Start Menu and search for PowerShell.

Right-click on Windows PowerShell and select Run as administrator.

Now, run:

```
Set-Service ssh-agent -StartupType Automatic
Start-Service ssh-agent
```

If no errors appear, run:

```
ssh-add ~/.ssh/id_ed25519
```

Verify the key is loaded using:

```
ssh-add -l
```

6.7 Enabling NTP Synchronization:

NTP synchronization is the process of using the Network Time Protocol (NTP) to align the clocks of networked computer systems to a single, accurate time source, like an atomic clock. This process ensures that all devices on a network agree on the time, which is crucial for accurate logging, security, and coordination of events like file updates or transactions.

NTP synchronization is enabled by default on a Raspberry Pi. However, if it has been disabled or becomes disabled for any reason, it can be re-enabled using the following command:

```
sudo timedatectl set-ntp true
```

6.8 Determining Currently Running Python Scripts:

To list all, if any, Python scripts are running use the following command:

```
ps aux | grep python
```

6.9 Determining Open TCP Ports:

To list all open ports, use one of the following commands:

```
ss -lntp
```

```
Netstat | grep -E 'tcp|State'
```