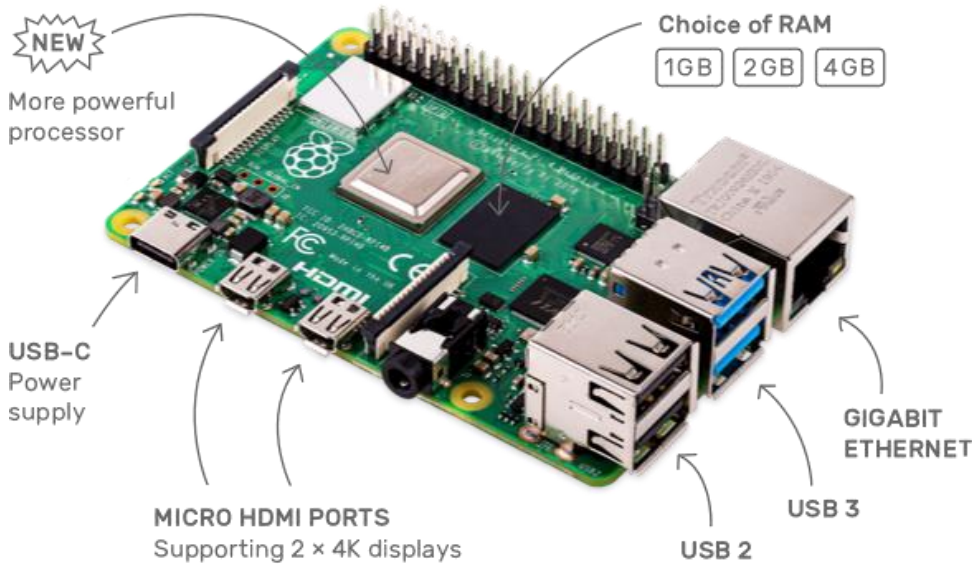
EECS 388 Lab #6

Intro to Embedded Linux & Board-to-Board Communication

In this lab, you will first get familiarized with the Raspberry Pi 4 and then establish UART based communication channels between the Pi 4 and the HiFive1 board.



Unlike the HiFive1 micro-controller you used, the Raspberry Pi 4 in front of you is essentially a small PC, which runs a general purpose operating system, Linux, complete with a desktop environment. You can basically do anything that you would expect to do on a Linux based PC. On the Raspberry Pi 4, using its computing power, you will later (Final Project) run a deep learning model for vision based real-time control.

Part 0: Setup your account in the Raspberry Pi 4.

Before you do any of such fancy things, you first need to setup your account that you will use for the rest of the semester. Because each Pi needs to be shared by six students, we created the following six accounts.

```
Login id:
      lab1        for M  10:00 - 11:45 a.m. class
      lab2        for M  03:00 - 04:45 p.m. class
      lab3        for Tu 09:00 - 10:45 a.m. class
```

```
lab4        for Tu 12:30 - 02:15 p.m. class
lab5        for W  03:00 - 04:45 p.m. class
lab6        for Th 09:00 - 10:45 a.m. class
lab7        for Th 12:30 - 02:15 p.m. class
lab8        for F  01:00 - 02:45 a.m. class
```

You need to choose the right account based on which lab section you belong to. For example, if you are in 10:00 a.m. Monday lab section, your account name will be 'lab1', while it will be 'lab6' if you are in the Thursday morning lab.

The TA will tell you the initial login password. Once you login to the Pi 4, the first thing you need to do is to change the password as follows, so that only you can access the files in our account. In the Terminal:

```
$ passwd
```

Note that, throughout the rest of the semester, you need to use the same Pi 4 because your account is local to the specific Pi 4 board.

Part 1: Getting familiar with the commonly used Linux tools

Task 1.1 Hello World in C

Use your favorite editor (vi, emacs, or anything), edit the following "Hello World!" in C and python:
Note: For vi, to begin typing you first press the letter `i` to enter INSERT mode. Once you have finished, press `Esc` and then type `:x` to save the file and return to the terminal.

```
$ vi hello.c
#include <stdio.h>
int main()
{
  printf("Hello World!\n");
  return 0;
}
$ gcc hello.c
$ ./a.out
Hello World!
```

Task 1.2. Hello World in Python

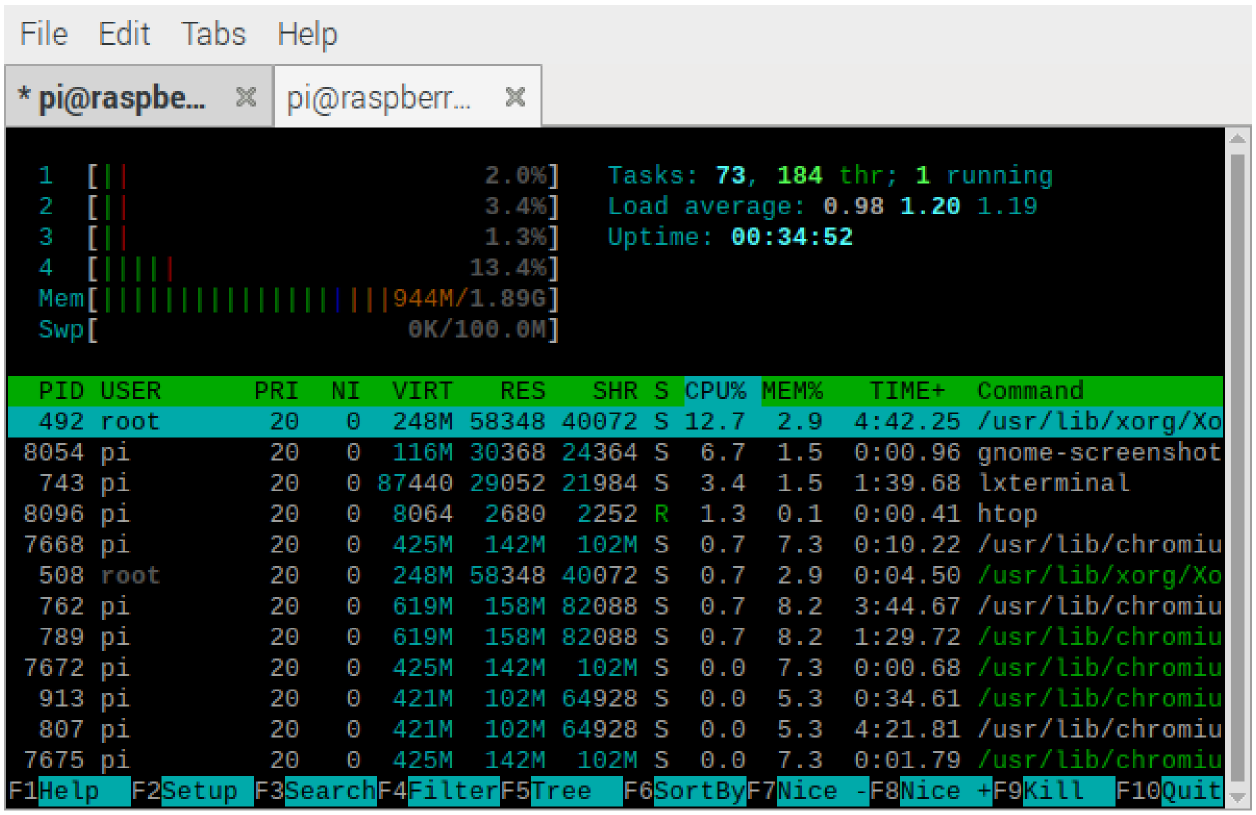
You can do the same "Hello World!" with python as follows. By executing the following command, the python interpreter will open.

```
$ python
Python 2.7.16 (default, Apr  6 2019, 01:42:57)
[GCC 8.2.0] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print ("Hello World!")
Hello World!
```

Task 2. System monitoring

Let's check what programs are currently running on the Pi 4 using `htop`. Note that Pi 4 has 4 cores and 2 GB of memory. You can visually see what programs are running on which CPU cores and how much they are being used.

```
$ htop
```



```
  1  [||                       2.0%]   Tasks: 73, 184 thr; 1 running
  2  [||                       3.4%]   Load average: 0.98 1.20 1.19
  3  [||                       1.3%]   Uptime: 00:34:52
  4  [|||||                   13.4%]
  Mem[|||||||||||||||||||944M/1.89G]
  Swp[                    0K/100.0M]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
  492 root       20   0  248M 58348 40072 S 12.7  2.9  4:42.25 /usr/lib/xorg/Xo
 8054 pi         20   0  116M 30368 24364 S  6.7  1.5  0:00.96 gnome-screenshot
  743 pi         20   0 87440 29052 21984 S  3.4  1.5  1:39.68 lxterminal
 8096 pi         20   0  8064  2680  2252 R  1.3  0.1  0:00.41 htop
 7668 pi         20   0  425M  142M  102M S  0.7  7.3  0:10.22 /usr/lib/chromiu
  508 root       20   0  248M 58348 40072 S  0.7  2.9  0:04.50 /usr/lib/xorg/Xo
  762 pi         20   0  619M  158M 82088 S  0.7  8.2  3:44.67 /usr/lib/chromiu
  789 pi         20   0  619M  158M 82088 S  0.7  8.2  1:29.72 /usr/lib/chromiu
 7672 pi         20   0  425M  142M  102M S  0.0  7.3  0:00.68 /usr/lib/chromiu
  913 pi         20   0  421M  102M 64928 S  0.0  5.3  0:34.61 /usr/lib/chromiu
  807 pi         20   0  421M  102M 64928 S  0.0  5.3  4:21.81 /usr/lib/chromiu
 7675 pi         20   0  425M  142M  102M S  0.0  7.3  0:01.79 /usr/lib/chromiu
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

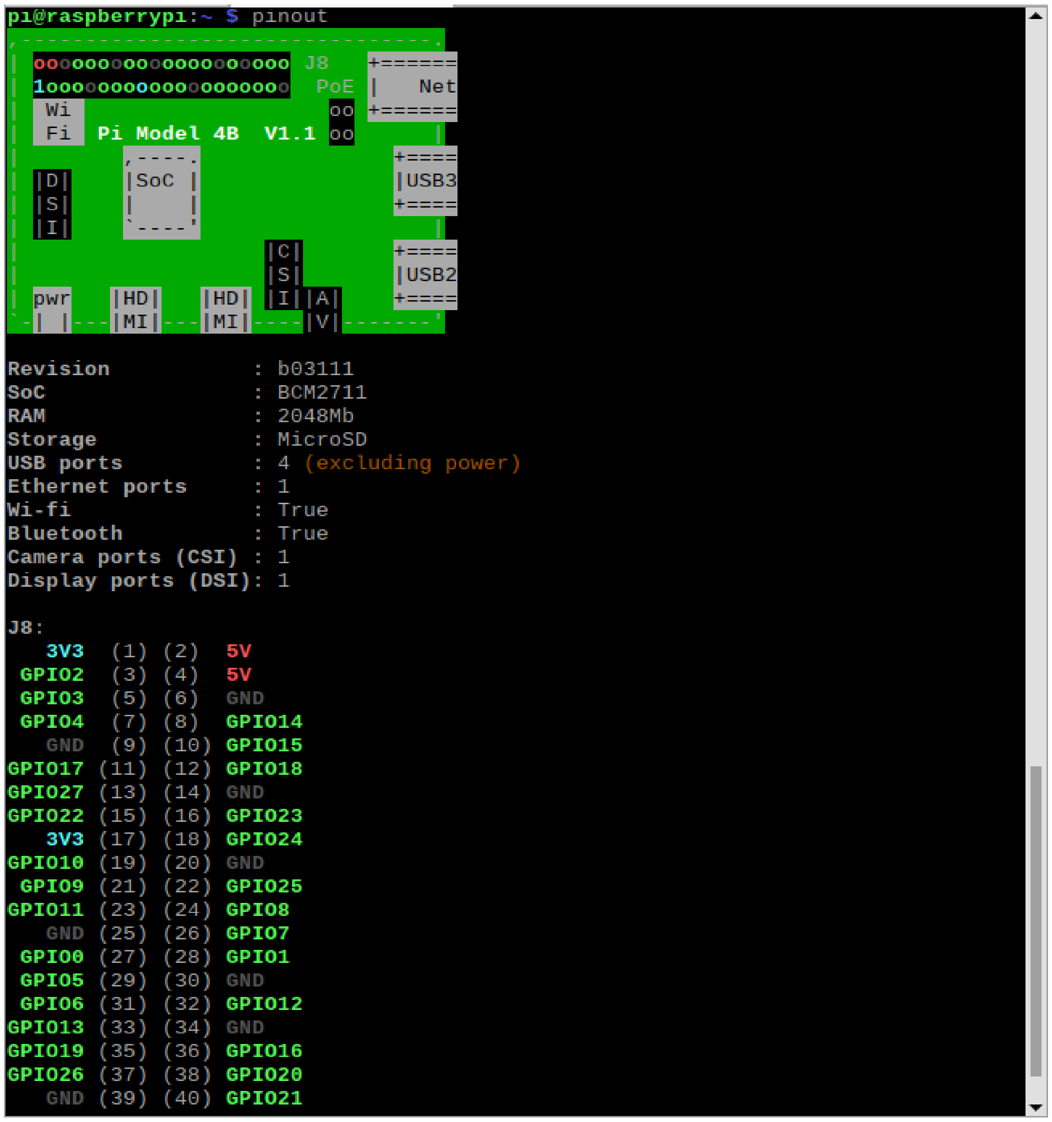
Now, open another terminal window (ctrl + alt + t) and create/run the following simple C program and see how it is shown in the htop screen on the first terminal window.

```
$ vi cpuhog.c
int main()
{
  while(1);
  return 0;
}
$ gcc cpuhog.c -o cpuhog
$ ./cpuhog
```

Task 3. Other Tools

There are a couple of very useful Raspberry Pi specific tools. The pinout is such a program. You may need the tool in the future when you connect the Pi 4 to other sensors and the HiFive 1.

```
$ pinout
```



```
pi@raspberrypi:~ $ pinout
,--------------------------------.
| oooooooooooooooooooo J8   +======
| 1ooooooooooooooooooo  PoE |   Net
|  Wi                    oo +======
|  Fi  Pi Model 4B  V1.1 oo      |
|        ,----.               +====
| |D|    |SoC |               |USB3
| |S|    |    |               +====
| |I|    `----'                    |
|                   |C|       +====
|                   |S|       |USB2
| pwr   |HD|   |HD| |I||A|    +====
`-| |---|MI|---|MI|----|V|-------'

Revision           : b03111
SoC                : BCM2711
RAM                : 2048Mb
Storage            : MicroSD
USB ports          : 4 (excluding power)
Ethernet ports     : 1
Wi-fi              : True
Bluetooth          : True
Camera ports (CSI) : 1
Display ports (DSI): 1

J8:
   3V3  (1) (2)  5V
 GPIO2  (3) (4)  5V
 GPIO3  (5) (6)  GND
 GPIO4  (7) (8)  GPIO14
   GND  (9) (10) GPIO15
GPIO17 (11) (12) GPIO18
GPIO27 (13) (14) GND
GPIO22 (15) (16) GPIO23
   3V3 (17) (18) GPIO24
GPIO10 (19) (20) GND
 GPIO9 (21) (22) GPIO25
GPIO11 (23) (24) GPIO8
   GND (25) (26) GPIO7
 GPIO0 (27) (28) GPIO1
 GPIO5 (29) (30) GND
 GPIO6 (31) (32) GPIO12
GPIO13 (33) (34) GND
GPIO19 (35) (36) GPIO16
GPIO26 (37) (38) GPIO20
   GND (39) (40) GPIO21
```

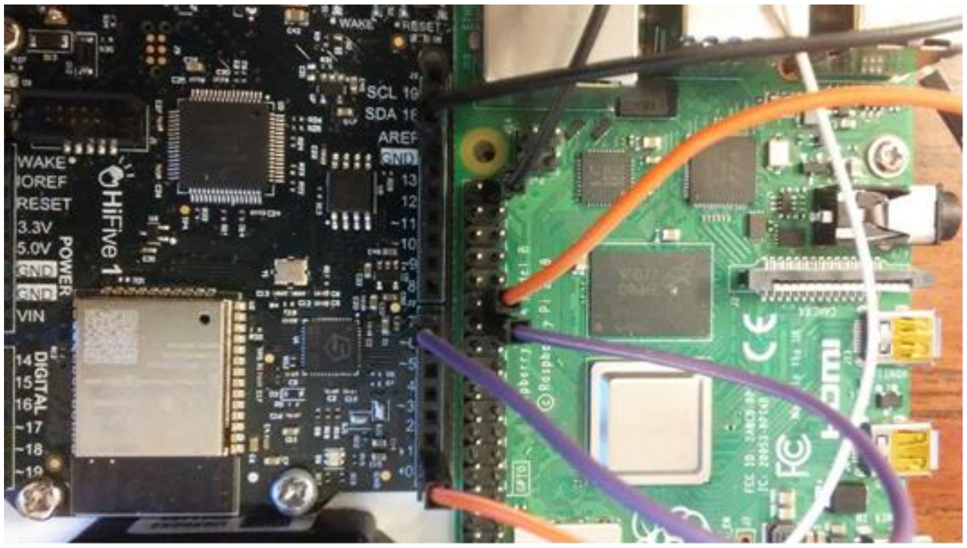
Part 2: Setup the UART connections

In this part, we will connect the HiFive1 and the Raspberry Pi 4 boards via two UART channels.

While the Pi 4 has 4 UARTs, we will only be using two of them today (uart2 and uart3). Examine the file `/boot/config.txt` for the following two lines enabling uart2 and uart3, add them to the file if they are not present:

```
dtoverlay=uart2,115200
dtoverlay=uart3,115200
```

If you needed to add these two lines, reboot the Raspberry Pi. After rebooting the system, `/dev/ttyAMA1` and `/dev/ttyAMA2` will be created.



Connect HiFive's UART1 RX (pin7) to Raspberry Pi 4's UART2 TX (pin 27). This is the main communication line between the Pi 4 and the HiFive1. From the Pi 4, you can access the channel via `/dev/ttyAMA1`.

For debugging of HiFive 1, connect HiFive1's UART0 TX (pin1) to Pi 4's UART3 RX (pin 29). From the Pi 4, it can be accessed via /dev/ttyAMA2.

In summary, you will be able to access the following two files from the Pi 4.

```
/dev/ttyAMA1      Pi 4 → HiFive1: Send steering angle to HiFive1 (uart1).
/dev/ttyAMA2      HiFive1 → Pi 4: Receive HiFive1's console (uart0) output
```

Part 3: Programming the HiFive1

In this part of the lab, you will program the HiFive1 to receive data from the Pi 4.

On your PC (not Pi 4), download the starter code.

```
// Download tar from Blackboard into ~/Documents/PlatformIO
$ cd ~/Documents/PlatformIO
$ tar zxvf lab6-comm.tar.gz
$ code lab6-comm/l6-comm/
```

Your task is to receive the data from HiFive1's UART1 channel and send the received data to UART0 channel **as a null-terminated string**. The following is rough **pseudo code** for the task.

```
while (1) {
  if (is UART1 ready?) {
    data = read from UART1.
    print data to UART0.
  }
}
```

To implement the task, you should use the provided serial API shown in the following. Note that `devid` is 0 to access UART0, while it is 1 to access UART1. For this lab, the two functions `ser_printline` and `ser_readline` will be especially helpful (since you need to send the data as a string).

```
void ser_setup(int devid);
int  ser_isready(int devid);
void ser_write(int devid, char c);
void ser_println(int devid, char *str);
char ser_read(int devid);
int  ser_readline(int devid, int n, char *str);
```

In particular, you may need to use `ser_isready()` function to check whether a given UART channel has pending data to read. To better understand what a specific function is doing, you can always check both `eecs388_lib.h` and `eecs388_lib.c` files.

```
int  ser_isready(int devid)
{
  uint32_t regval = *(volatile uint32_t *)(UART_ADDR(devid) + UART_IP);
  return regval;
}
```

How to test your HiFive code: Once you finish programming the HiFive1, **switch to the Raspberry Pi 4** and open two terminals: one for sending data to the HiFive1, and one for seeing the debug message output from the HiFive1.

Sender's terminal (terminal 1):

```
$ screen /dev/ttyAMA1 115200
```

Debug terminal (terminal 2):

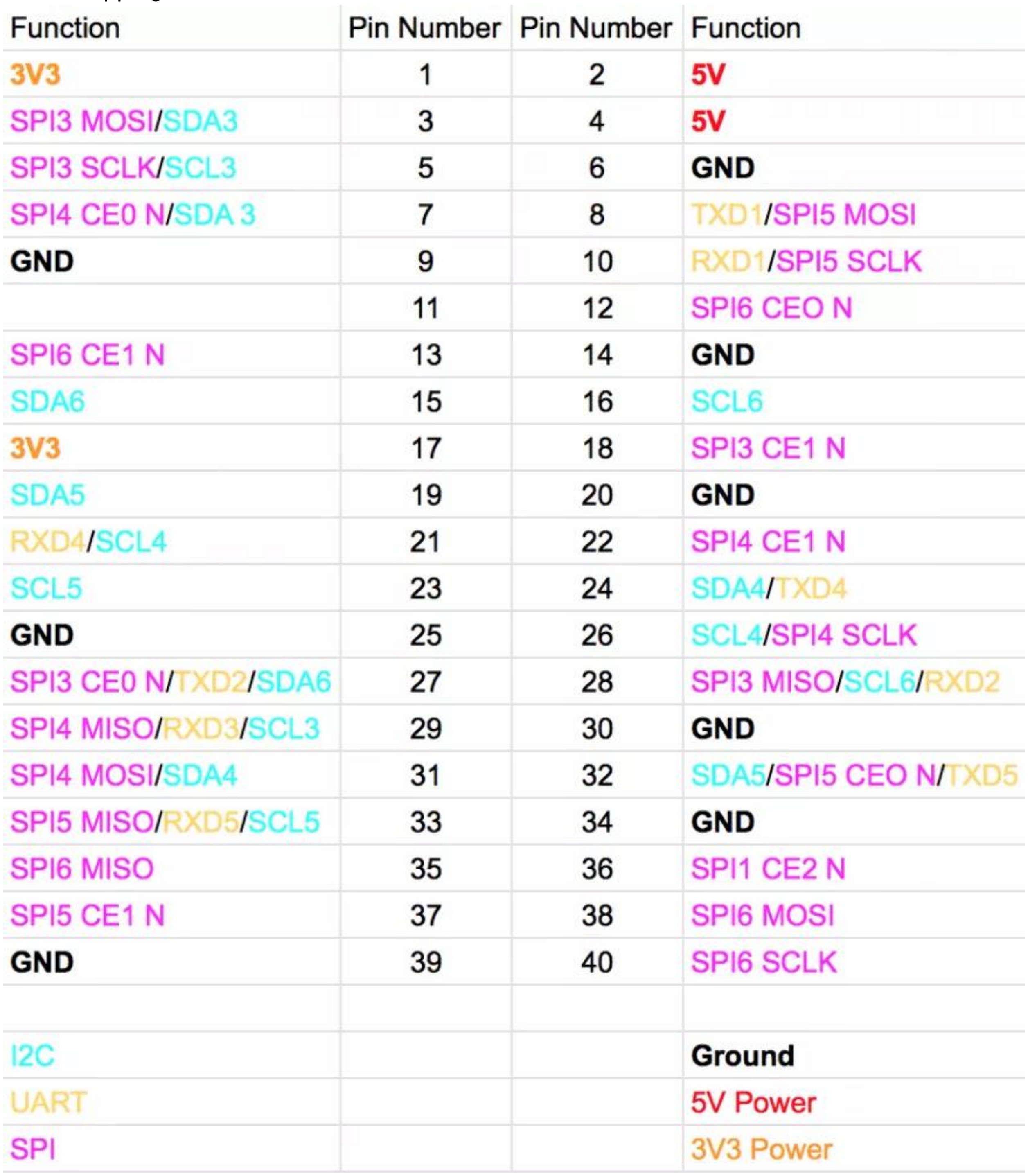
```
$ screen /dev/ttyAMA2 115200
```

Now, type any string on the ‘terminal 1’. If you programmed your HiFive 1 correct, you should see the message coming out from the ‘terminal 2’ terminal.

Once you are finished, show your work to the TA and then submit to blackboard.

Appendix

GPIO mapping of Pi 4.



Function	Pin Number	Pin Number	Function
3V3	1	2	5V
SPI3 MOSI/SDA3	3	4	5V
SPI3 SCLK/SCL3	5	6	GND
SPI4 CE0 N/SDA 3	7	8	TXD1/SPI5 MOSI
GND	9	10	RXD1/SPI5 SCLK
	11	12	SPI6 CEO N
SPI6 CE1 N	13	14	GND
SDA6	15	16	SCL6
3V3	17	18	SPI3 CE1 N
SDA5	19	20	GND
RXD4/SCL4	21	22	SPI4 CE1 N
SCL5	23	24	SDA4/TXD4
GND	25	26	SCL4/SPI4 SCLK
SPI3 CE0 N/TXD2/SDA6	27	28	SPI3 MISO/SCL6/RXD2
SPI4 MISO/RXD3/SCL3	29	30	GND
SPI4 MOSI/SDA4	31	32	SDA5/SPI5 CEO N/TXD5
SPI5 MISO/RXD5/SCL5	33	34	GND
SPI6 MISO	35	36	SPI1 CE2 N
SPI5 CE1 N	37	38	SPI6 MOSI
GND	39	40	SPI6 SCLK
I2C			Ground
UART			5V Power
SPI			3V3 Power

Source: https://learn.pi-supply.com/make/raspberry-pi-4-pinout