EECS 388 Lab #10

Board-to-Board Communication

In this lab, you will establish UART based communication channels between the Pi 4 and the HiFive 1 boards.

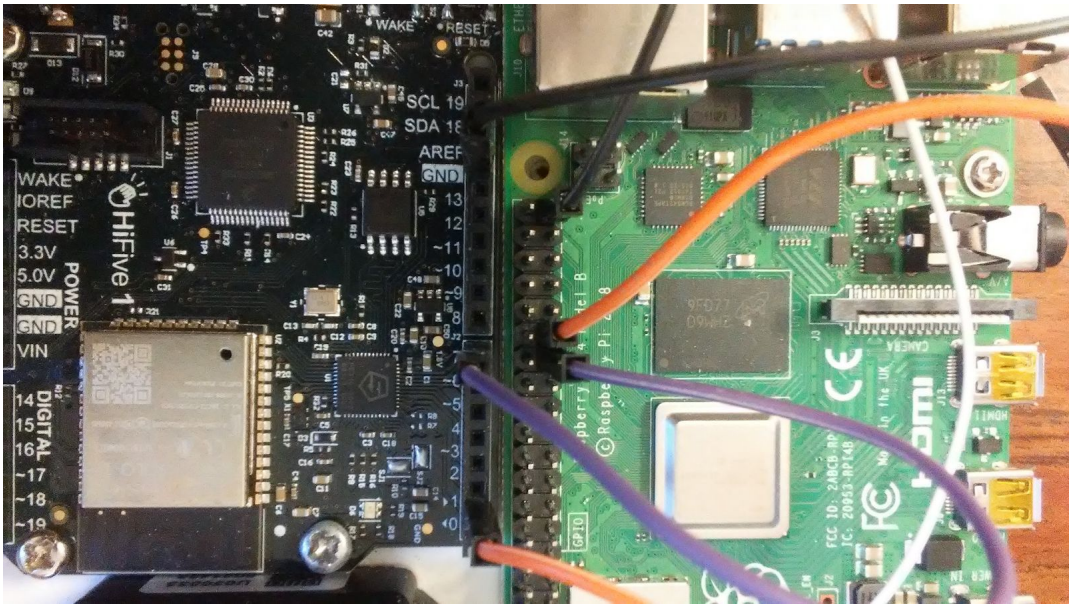
Part 0: Setup the UART connections (for TAs)

In this part, we will connect the HiFive1 and the Raspberry Pi 4 boards via two UART channels. **(Note that this step will be performed by the TAs.)**

The Pi 4 has 4 UARTs and we will use two of them (uart2 and uart3). Add the following line at the end of the `/boot/config.txt` file to enable uart2 and uart3.

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

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 1: 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 project skeleton as follows.

```
$ cd ~/Documents/PlatformIO
$ wget https://ittc.ku.edu/~heechul/courses/eecs388/l10-comm.tar.gz
$ tar zxvf l10-comm.tar.gz
```

Add the l10-interrupt folder into VSCode workspace.

Your task is to receive the data from HiFive1's UART1 channel and send the received data to UART0 channel. The following is a rough **pseudo code** of the task.

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

To implement the task, you may need to use the provided serial API shown in the following. Note that devid is 0 to access UART0, while it is 1 to access UART1.

```
void ser_setup(int devid);
int  ser_isready(int devid);
void ser_write(int devid, char c);
void ser_printline(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 the functions are doing, check `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;
}
```

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 (term1)
```
$ screen /dev/ttyAMA1 115200
```

Debug terminal (term2)
```
$ screen /dev/ttyAMA2 115200
```

Now, type any strings on the 'term1'.

If you programmed your HiFive 1 correct, you should see the message coming out from the 'term2' terminal.

Part 2: Programming the Raspberry Pi 4.

Instead of using terminals, you now run a python program on the Pi 4 to communicate with the HiFive1. Your task is to extend the dnn.py from the previous lab to be able to send the steering output to the `/dev/ttyAMA1` serial channel. The following **pseudo code** provides a general idea of the modifications you will need to make to dnn.py:

```
Open serial connections to /dev/ttyAMA1 and /dev/ttyAMA2
While True:
  image = camera.read()
  angle = dnn_inference(image)
  Write 'angle' to /dev/ttyAMA1
  Wait_till_next_period()
Close serial connections
```

To achieve the functionality from above, you need to use Python's pySerial API which can be used by importing the serial package:

```
import serial
```

With it, you should create two separate serial channels, one for writing to the HiFive1 over /dev/ttyAMA1 and another for debugging over /dev/ttyAMA2. Note that both channels should be opened with the baudrate 115200 bps.

```
ser1 = serial.Serial(...)
ser2 = serial.Serial(...)
```

The angles received from the DNN as it processes frames can then be sent to the HiFive1 by using the serial write() function:

```
ser1.write(...)
```

However, write() requires a byte value while the angle produced by the DNN is a float32 value, so you will have to convert the angle data in order to send it to the HiFive1. Finally, after all of the frames are processed, the serial connections can be closed by invoking the serial close() function:

```
ser1.close()
ser2.close()
```

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 — to HiFive1	28	SPI3 MISO/SCL6/RXD2
SPI4 MISO/RXD3/SCL3	29	30	GND
SPI4 MOSI/SDA4	31 — from HiFive1 (debug)	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