Please find below a detailed explanation for Pi setup. You should make sure that you Pi is connected properly before starting.

GDB Server Connection

To establish a connection to the GDB server you need to open a terminal and type **JLinkGDBServer** (Case Sensitive). A successful connection should have the same results as **Figure 1.** If you don't have same results than you should disconnect and reconnect again the PI. If you are still facing issues you should change your machine. After you are done minimize the terminal.

```
J-Link Host interface:          USB
J-Link script:                  none
J-Link settings file:           none
------Target related settings------
Target device:                  unspecified
Target interface:               JTAG
Target interface speed:         1000kHz
Target endian:                  little

Connecting to J-Link...
J-Link is connected.
Firmware: J-Link ARM V8 compiled Sep 22 2014 23:26:43
Hardware: V8.00
S/N: 268005455
OEM: SEGGER-EDU
Feature(s): FlashBP, GDB
Checking target voltage...
Target voltage: 3.31 V
Listening on TCP/IP port 2331
Connecting to target...
J-Link found 1 JTAG device, Total IRLen = 4
JTAG ID: 0x4BA00477 (Cortex-A8)
Connected to target
Waiting for GDB connection...
```

Figure 1: GDB Server

To quit gdb server type **Ctrl + z**

Code Loading

For code loading first you should download the template folder from D2L and put it in your desktop inside cpsc359 folder. Please follow the steps bellow in order:

1. Extract your template folder inside the cpsc359 folder.
2. Open another terminal and navigate to the template folder.
3. Type in terminal command: **make all** (this is responsible for compiling your code)
4. Type in terminal command: **arm-none-eabi-gdb build/output.elf**

5. Type in terminal command: **target remote localhost:2331**
6. Type in terminal command: **load**
7. Type in terminal command: **j _start**. This will execute your code
8. You should have the same results as in **figure 2**

```
make: Nothing to be done for 'all'.
[asarhan@pi-ta1 templateRPi2]$ arm-none-eabi-gdb build/output.elf
GNU gdb (GNU Tools for ARM Embedded Processors) 7.6.0.20140529-cvs
Copyright (C) 2013 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.htr
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show copying"
and "show warranty" for details.
This GDB was configured as "--host=i686-linux-gnu --target=arm-none-eabi".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /home/grads/asarhan/Desktop/CPSC_359_Winter_2017/temp
i2/build/output.elf...done.
(gdb) target remote localhost:2331
Remote debugging using localhost:2331
0x00000000 in ?? ()
(gdb) load
Loading section .init, size 0x4 lma 0x8000
Loading section .text, size 0x9c lma 0x8004
Start address 0x8004, load size 160
Transfer rate: 17 KB/sec, 80 bytes/write.
(gdb) j _start
Continuing at 0x8000.
```

Figure 2: Code Execution

- To exit code execution press **Ctrl+c**
- To quit gdb type **quit** then **y.**

UART Initialization

For this you should make sure you download the new template folder which contain the UART class. You can find it in the same folder of Assignment 2 on D2L. You should now use this template instead of the previous one.

Figure 3: UART Interface

ttyUSB0

Open a third screen ant type **screen /dev/~~ttyol~~ 115200** then press **enter.** You should have same results as in **Figure 3**. In case you have an error (see **Figure 4)** then you should unplug all USB cables and plug them again run **screen /dev/ttyol 115200** again; It should now work.

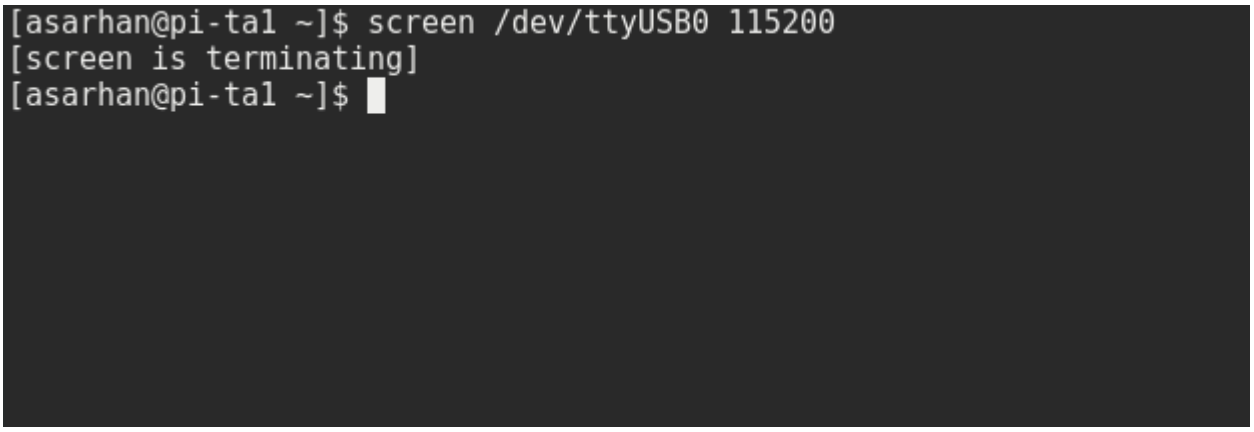

Figure 4: UART Error

Note:

In order not to unplug and plug all USB cables every time (you will face this a lot). Remove each one separately and then run **screen /dev/ttyol 115200;** if it works then you will know which USB cable to unplug every time you face this issue.

To see values stored in registers you should type **info registers** as shows in figure 5

```
Continuing at 0x8000.
^C
Program received signal SIGTRAP, Trace/breakpoint trap.
haltLoop$ () at source/main.s:17
17              b               haltLoop$
(gdb) info registers
r0             0x6      6
r1             0x2461b6c0       610383552
r2             0x61b6c0 6403776
r3             0x0      0
r4             0x0      0
r5             0x0      0
r6             0x0      0
r7             0x0      0
r8             0x0      0
r9             0x0      0
r10            0x0      0
r11            0x0      0
r12            0x0      0
sp             0x8000   0x8000 <_start>
lr             0x8038   32824
pc             0x8010   0x8010 <haltLoop$>
cpsr           0x600000d3       1610612947
(gdb)
```

Figure 5: Info Registers