ECE 484W

Lab Assignment #3
Communicating between the DE10-Standard (VEEK-MT2S) and Computer

Due by date listed on Canvas

Turn in your report via Canvas (each individual needs to do it; accumulated 10 points deduction for every 12 hours window late submission)

Turn in your video clip (less than one minute; each individual needs to do it) via Canvas to demonstrate each required task (Accumulated 10 points deduction for every 12 hours window late submission). If you video clip does not demonstrate each required task, you will lose 10 points for each missing task. If you do not submit your video clip, we will not grade your report.

If you submit your report/video clip multiple times, I will grade the last attempt and your submission time will be the time for your last attempt

Requirements:

Establish communication between your Qt GUI from Assignment #1 and VEEK-MT2S board via the virtual RS-232 port (mini-B USB port). The following functionalities are required:

1. When you move the brightness and contrast sliders in your Qt GUI on PC, the new values should be transmitted to the FPGA board and displayed on the 7-segment displays. Both values should be reset between 0 to 99.
2. You need to use source/version control software (any free software is acceptable) when collaborating on writing the code. In the Section of Design Methodology, you need to describe how your team use source/version control.

Important Notes:

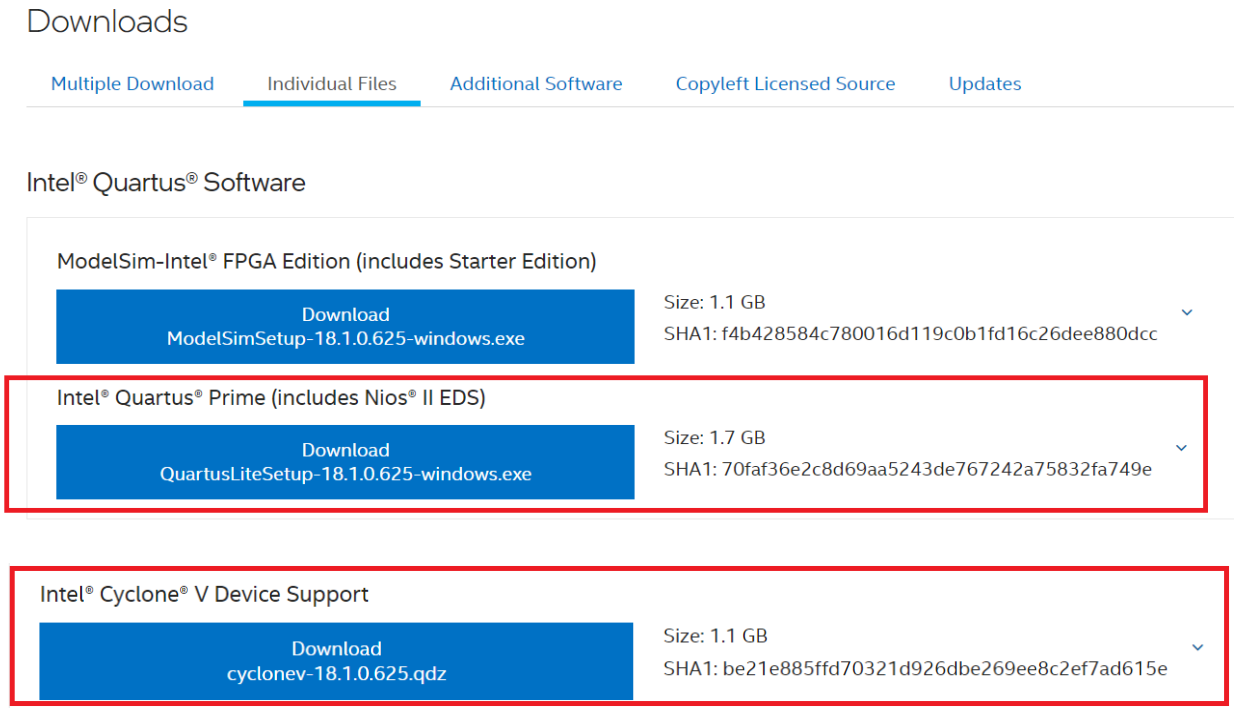
- A **router will be necessary** for this assignment and the remaining ones.
- Python or C++ code must be written for the receiver program developed on the board.
- This tutorial assumes starting everything from scratch, you may some of these things downloaded already (check version numbers).
- There may be errors when trying to compile u-boot on Windows 10…
- The virtual RS232 port might automatically be recognized upon connecting, meaning you can skip installing the drivers.
- **DE-10 Standard is the front of the board**, the LCD screen on the back is considered the VEEK-MT2S and the Linux boot image will have to be re-flashed to this for the next assignment.
- The sample Python code for operating the 7-segment displays on the board here is also attached.

Contents

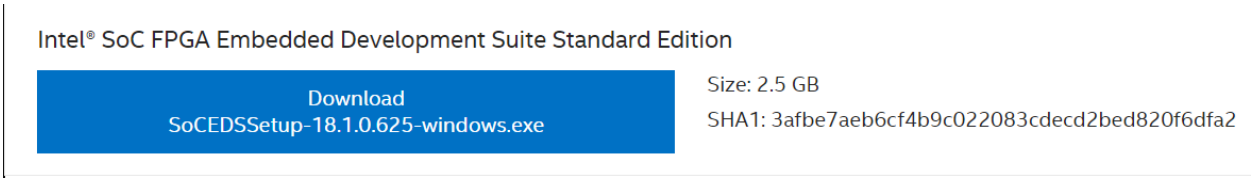
Downloads:

For software on Intel, you MAY need to first create an account and login to the website.

- Install Quartus Prime Lite Edition Design Software Version 18.1 for Windows and the Cyclone V device support boxed in red below
  (If you already have 18.1 installed, please ignore this step)



- Download and install Intel Quartus Prime Lite Edition Design Software Version 18.1 for Windows (choose DS-5 install at the end)



- Download the DE10 Standard CD
- Download the SD Card Linux image
- Download Win32 Disk Imager
- Download PuTTY

Flashing the SD Card:

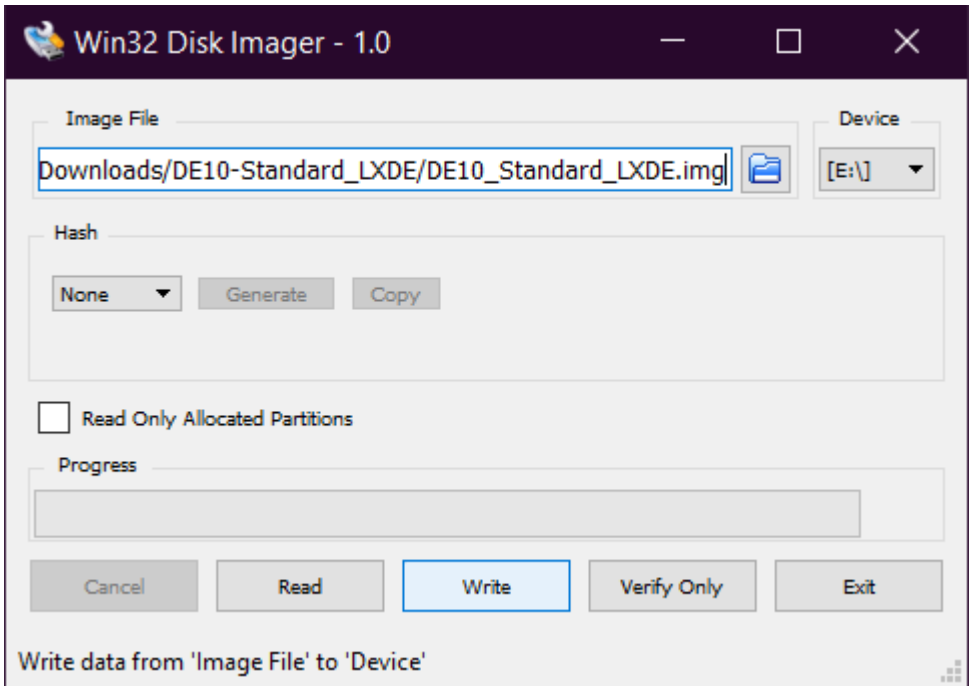
1. Insert the SD Card into your computer (there are adapters in the DE10 box if your computer doesn't support microSD)

2. Open Win32 Disk Imager and in the "Device" dropdown select the drive of the SD card defined by your computer ("E" in this case)

3. For "Image File" select the .img file **unzipped** from the previously installed SD Card Linux Image, "DE10-Standard_LXDE.zip"

4. Click "Write" and wait for it to finish
    - Confirm that writing to a physical device can corrupt it
    - Should take ~ 10 mins
    - Ignore any file explorer windows that pop up saying to format the disk

5. Eject the SD Card and put it back in the VEEK-MT2S board



Connecting the RS232 Port/Establishing Connection from PC to Board:

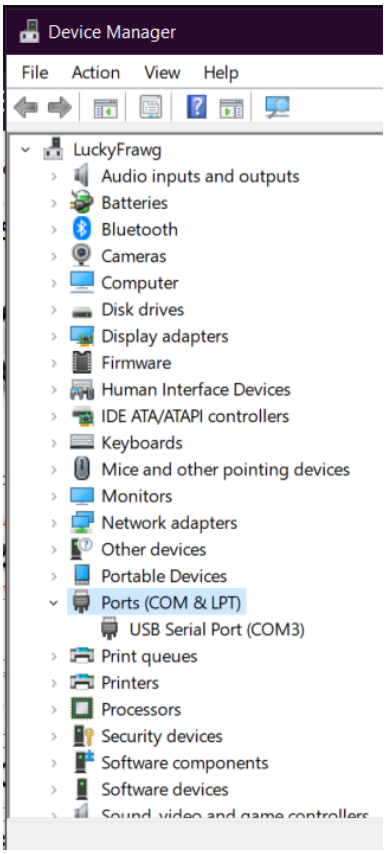
Make sure to use the Mini-B to USB cable to connect the board to the PC

1. Press “Windows Key” + “X” → Device Manager → Ports(COM & LPT) – about midway down → expand it and note the name of the serial port (COM?)

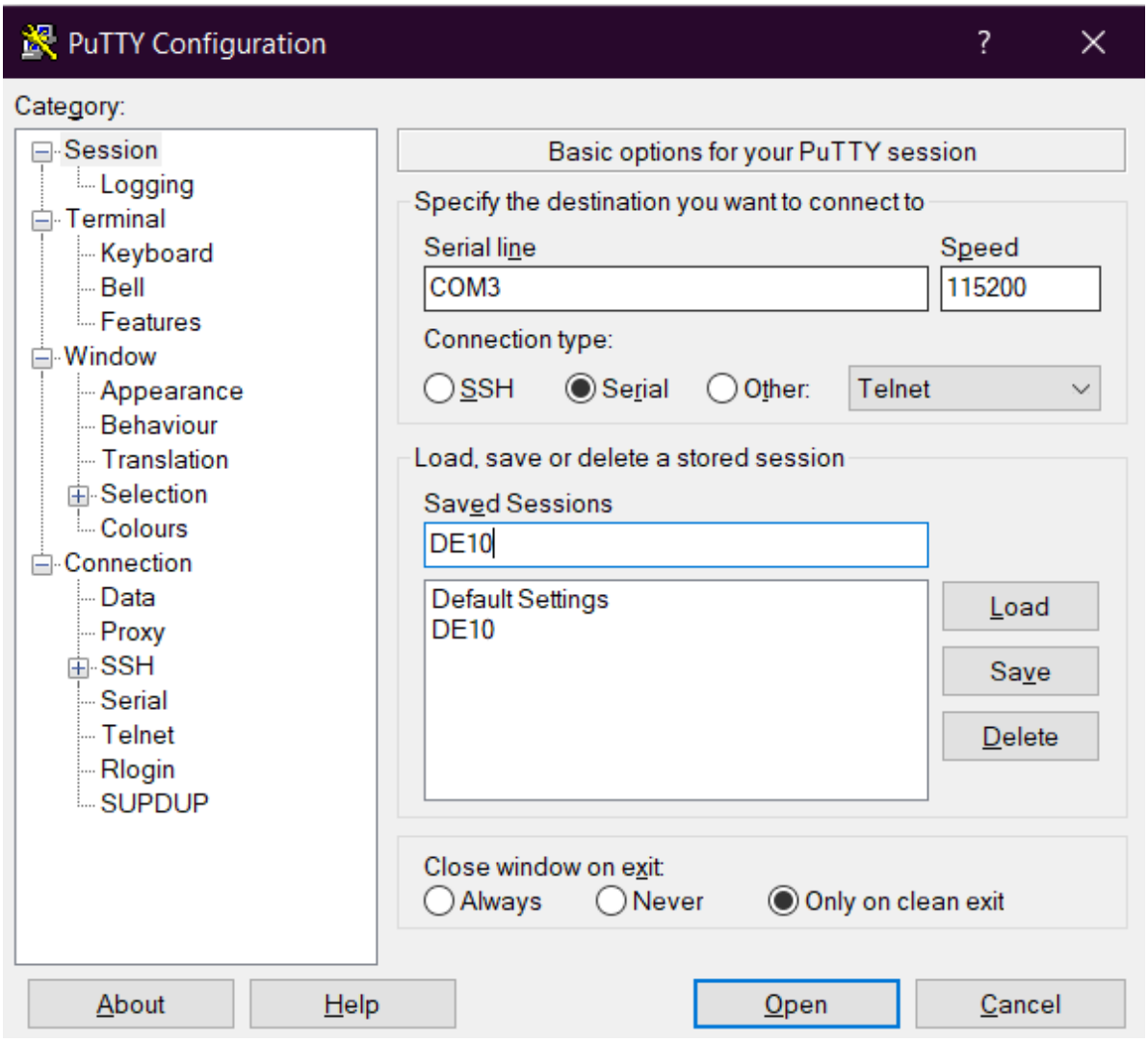
   If no port is found:
2. Download the drivers for the virtual RS232 port (mini-B USB port)

3. Unzip and launch the FTDI driver setup, click next until successfully installed

4. Retry above and double check you are using the Mini-B cable and that the connections are secure



5. Open PuTTY, on the main screen change the Connection Type to Serial, change the Serial Line to the previously noted serial port name above (COM3 in this case), and change the baud rate to 115200
    - This will have to be done each time you want to establish connection from the PC to board.
        - Recommended: Name a “Saved Session” and save it to easily set the settings for future assignments.



6. Connect the DE10 board and click “Open”, it should open a terminal window

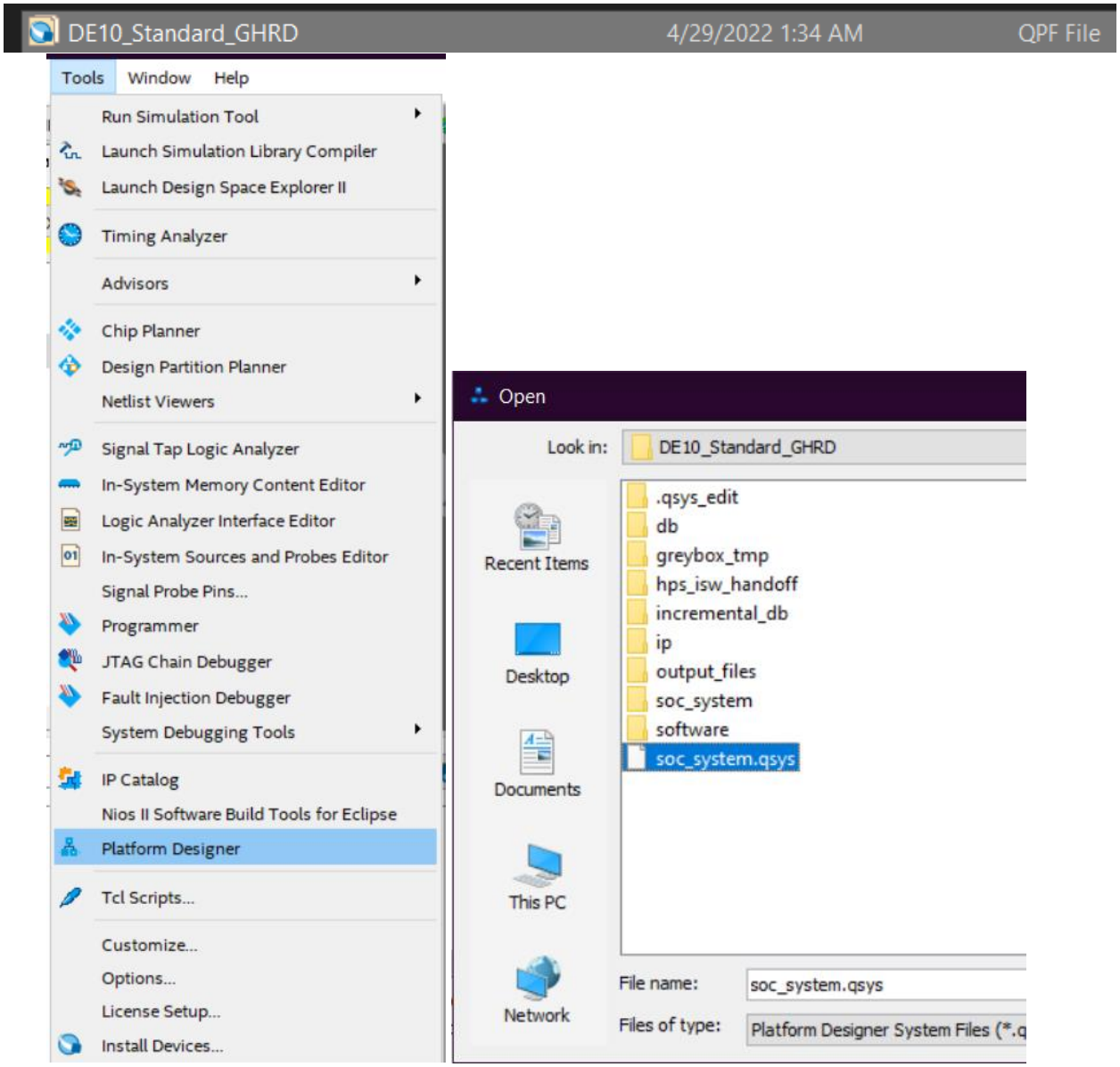
7. Press the red button on the board to power it on, you should see output in the terminal window about U-boot. Linux doesn’t necessarily boot here.
    - If U-boot info was displayed, you can power off the board and close PuTTY for now.

Quartus – Platform Designer (Modifying the GHRD Example Project):

1. Go into the DE 10-Standard_v.1.3.0_SystemCD file you downloaded and navigate to: “Demonstration/SoC_FPGA/DE10_Standard_GHRD/”

2. Open “DE10_Standard_GHRD.qpf”, which should open Quartus

3. After loading, go into Tools → Platform Designer (or Qsys)



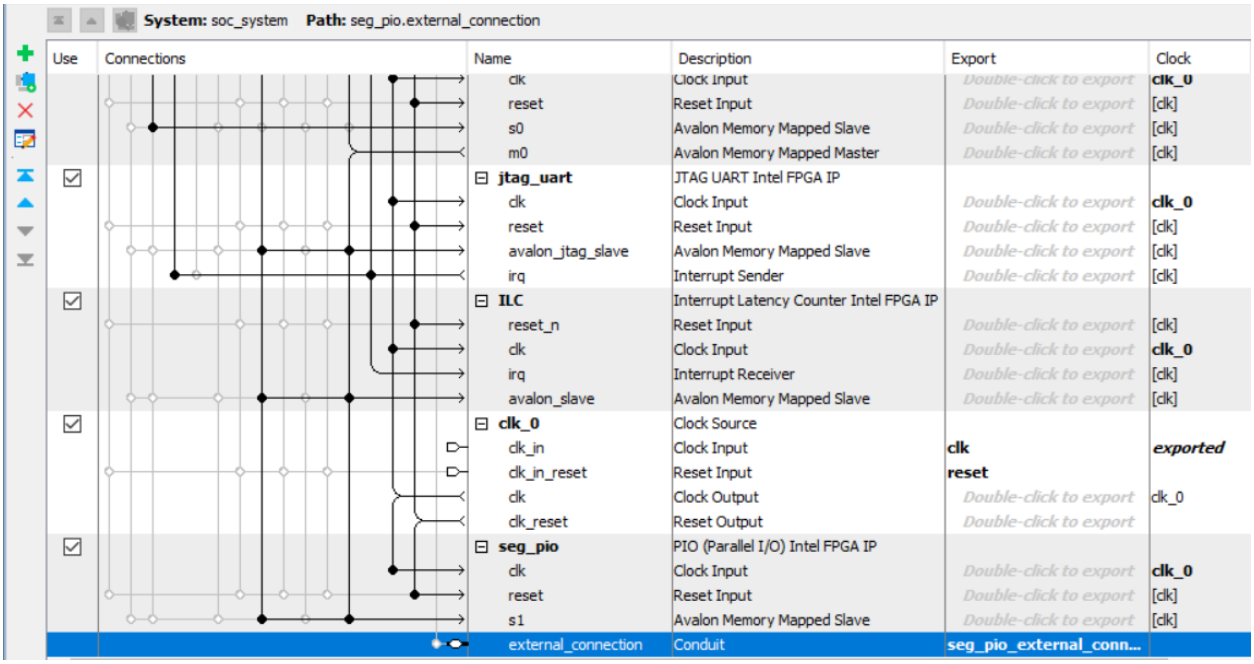
4. Platform Designer will automatically show a file open dialog, choose the existing “soc_system.qsys” file

The connection between the HPS and FPGA is already established, you just need to add the addressable IO to the 7-segment displays. You should see an example of these connection with the **led_pio** module located in the list.

5. Follow suit and add another parallel IO by going to the top left in the IP Catalog and searching for “pio”, you’ll see a module named “PIO (Parallel I/O) …”, double click it to add it to the system.

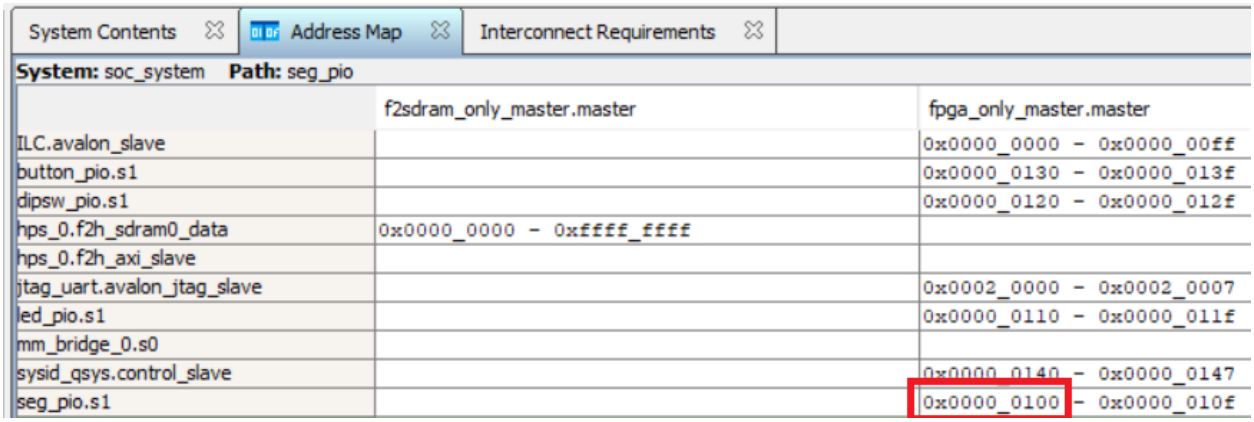
6. In the settings for the new PIO module, set the number of bits you need for the amount of 7-segment displays you will use…
    - Example: If the brightness slider goes from 0-99 then you need 2 digits (2 7-segment displays)
    - Similarly, if the contrast slider goes from 0-99 you need another 2 digits (2 7-degment displays)
    - This results in needing need 4*7 bits. The direction should be Output.

7. Click Finish.



8. Connection mapping for new pio above:
    - Renamed the newly created pio to “seg_pio”
    - Connected the module to replicate the above connections in “led_pio”
    - Double clicked to export the external_connection conduit. Take note of the name in the export, in this case “seg_pio_external_connection”

9. Go to System → Assign Base Addresses then go to View → Address Map and note the base address for the new pio, boxed in red below:



System Contents | Address Map | Interconnect Requirements

System: soc_system **Path:** seg_pio

	f2sdram_only_master.master	fpga_only_master.master
ILC.avalon_slave		0x0000_0000 - 0x0000_00ff
button_pio.s1		0x0000_0130 - 0x0000_013f
dipsw_pio.s1		0x0000_0120 - 0x0000_012f
hps_0.f2h_sdram0_data	0x0000_0000 - 0xffff_ffff	
hps_0.f2h_axi_slave		
jtag_uart.avalon_jtag_slave		0x0002_0000 - 0x0002_0007
led_pio.s1		0x0000_0110 - 0x0000_011f
mm_bridge_0.s0		
sysid_qsys.control_slave		0x0000_0140 - 0x0000_0147
seg_pio.s1		0x0000_0100 - 0x0000_010f

The base address above will be useful when writing the program for the board when needing to access the 7-segment displays

10. Click File → Save and it will process all the modified files before you can close.
11. Click Generate → Generate HDL → Generate (keep ALL defaults)
12. When it's done generating, click Close → Finish.
13. Close the Platform Designer.
14. Quartus will warn about the .qip and .sip files that were generated, just ignore this since they're already imported.

Quartus – Programming (Modifying the GHRD Example Project):

The newly created & exported external connection conduit must be connected to the actual 7-segment display location on the FPGA. Thankfully the pins to map the 7-segment displays are already put in place, you just need to hook them up to the external connection created in the Platform Designer.

1. Back in the main Quartus screen, double click on the “Top Level Design” on the left, named “DE10_Standard_GHRD”

2. You can follow how “led_pio_external_connection” sets up its connection, but here’s what you need to insert:
    - Add an internal vector of signals with the bit width of the PIO module added in a previous step ([27:0] is equivalent to (27 downto 0)):
        - Defining it in this manner means the programming done in Qt will need to add the values for contrast and brightness together as a string. Then, two of the values need to be correlated to brightness and the other two to contrast.
        - It is possible to define two separate PIO modules from above and 2 signals of 13:0, the method shown here is only an example.

```
212  //=======================================================
213  //  REG/WIRE declarations
214  //=======================================================
215    wire  hps_fpga_reset_n;
216    wire [3:0] fpga_debounced_buttons;
217    wire [6:0] fpga_led_internal;
218    wire [27:0] fpga_seg_internal;
219    wire [2:0] hps_reset_req;
220    wire       hps_cold_reset;
221    wire       hps_warm_reset;
222    wire       hps_debug_reset;
223    wire [27:0] stm_hw_events;
224    wire       fpga_clk_50;
225  // connection of internal logics
226    assign LEDR[9:1] = fpga_led_internal;
227    assign HEX0[6:0] = fpga_seg_internal[6:0];
228    assign HEX1[6:0] = fpga_seg_internal[13:7];
229    assign HEX2[6:0] = fpga_seg_internal[20:14];
230    assign HEX3[6:0] = fpga_seg_internal[27:21];
231    assign stm_hw_events    = {{4{1'b0}}, SW, fpga_led_internal, fpga_seg_internal, fpga_deboun
232    assign fpga_clk_50=CLOCK_50;
```

3. Note: fpga_seg_internal is also added to the stm_hw_events array (~line 231)

4. Assign the internal signal to the pins of the 7-segment displays, configured as HEX0-5 near the top of the file. If you look closely at your board, you will see this printed on the PCB. See my choice of displays above.

5. Now hook this internal signal up to the exported connection made in the Platform Designer similar to the led_pio external connection export:

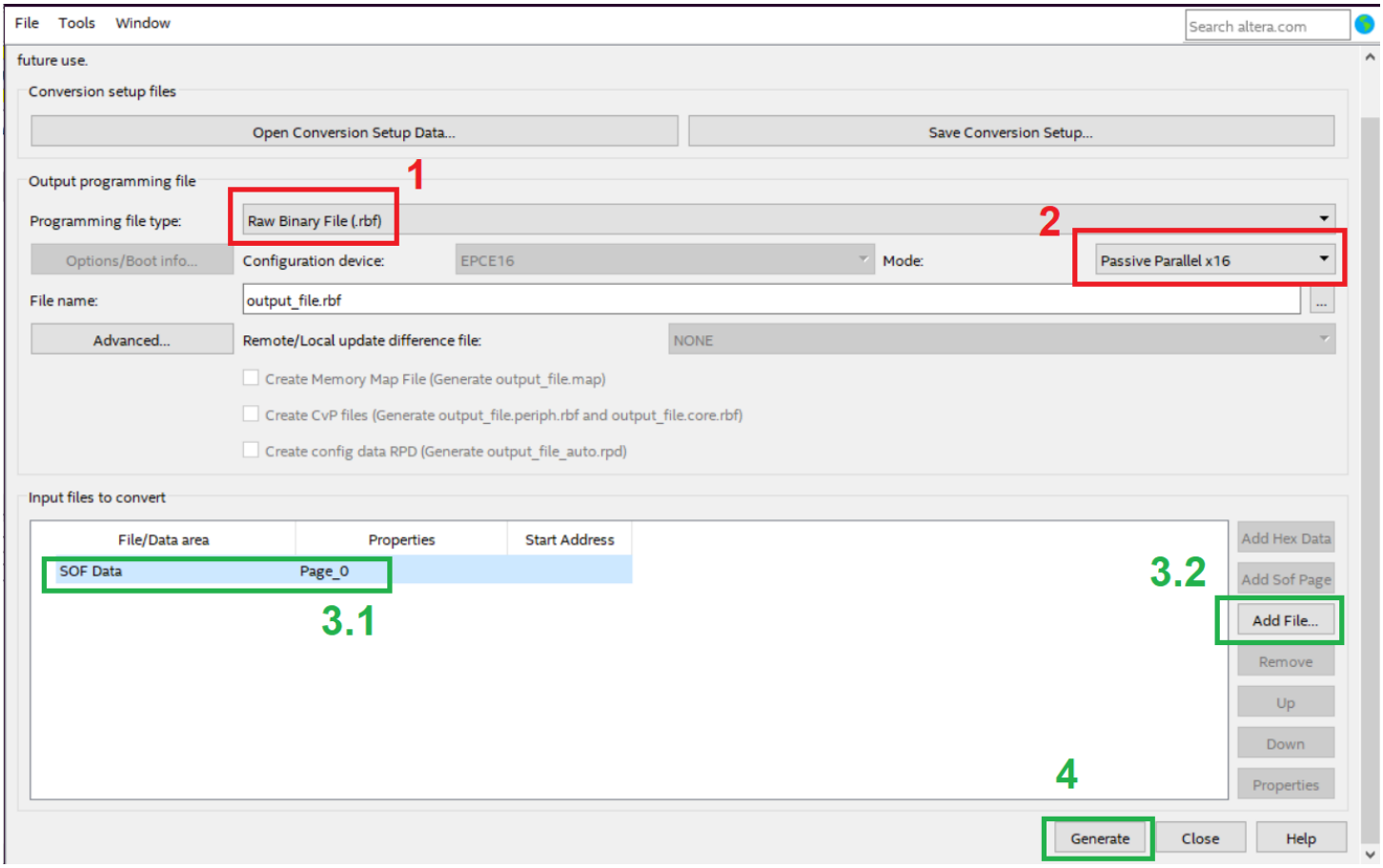
```
323          .hps_0_hps_io_hps_io_gpio_inst_GPIO54  ( HPS_KEY),
324          .hps_0_hps_io_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT),  /
325
326
327          .led_pio_external_connection_export    ( fpga_led_internal ),
328          .seg_pio_external_connection_export    ( fpga_seg_internal ),
329          .dipsw_pio_external_connection_export  ( SW ),
```

The naming for this follows ".<export_name>_export" because in the Platform Designer, ".export" is a property of the external connection "seg_pio_external_connection", which gets interpreted to "seg_pio_external_connection_export"

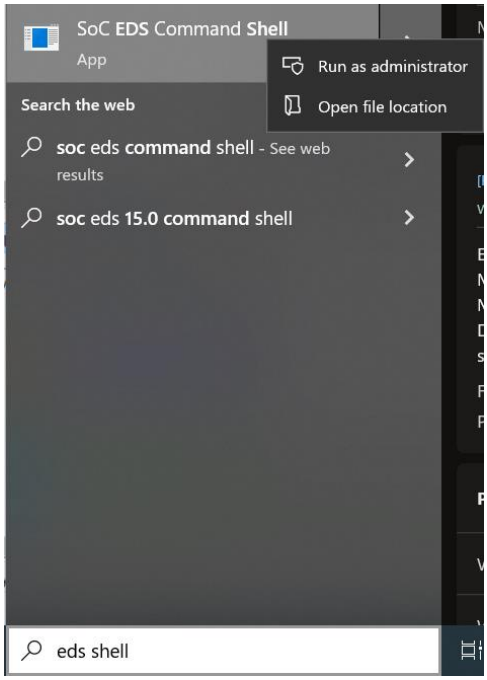
6. Finally, you can save the file you edited and press the play button, compiling the whole project. (~10 - 15+ mins)

Compiling the New System:

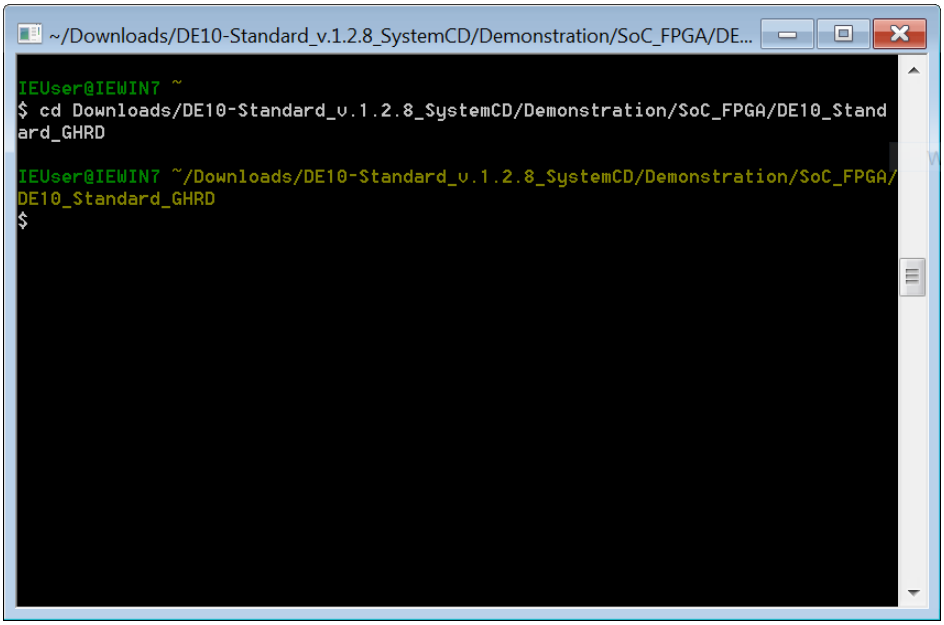
1. In the Quartus project, after compilation…go to File → Convert Programming Files…:
    1. Change the Programming file type to .rbf
    2. Change the Mode to Passive Parallel x16
    3. Click "SOF Data" →"Add File…", go to the output_files folder and choose DE10_Standard_GHRD.sof file.
    4. Click Generate then Close.



2. After the .rbf file is created, run the EDS shell with administrator privileges:



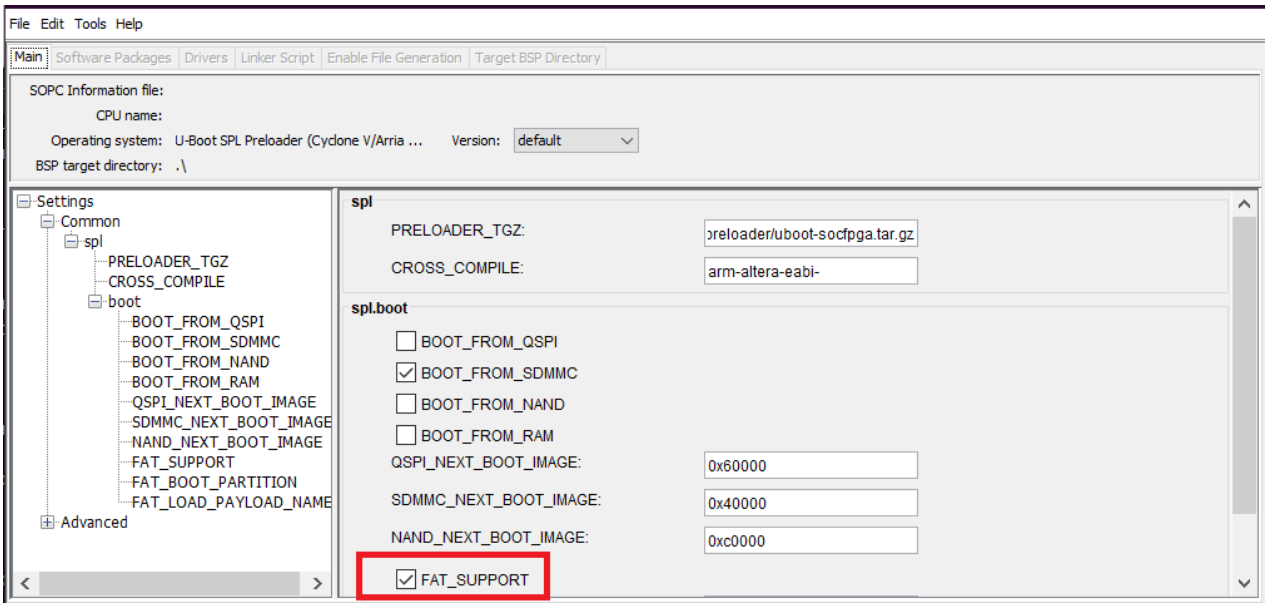
3. Navigate to the project directory you modified and compiled above in the EDS shell:
    - If it is within your downloads folder, you may copy and paste the following command:
    - cd ~/Downloads/DE10-Standard_v.1.3.0_SystemCD/Demonstration/SoC_FPGA/DE10_Standard_GHRD



4. Now run the command “bsp-editor &” and wait until the window pops up

5. In the window, click File → New HPS BSP, choose the Preloader settings directory by opening the “hps_isw_handoff” folder → “soc_system_hps_0”.
    - Beside “Operating system:”, “U-Boot SPL Preloader (Cyclone V…” should be visible
    - Click Ok

6. Now the only setting you must change is checking the FAT_SUPPORT checkbox here:



7. Click Generate then Exit

8. Navigate to the newly generated directory by typing the command “cd software/spl_bsp/”

Compiling U-boot:

This is where most students on Windows encounter errors

1. To build uboot, run the command “make && make uboot”, this will take a while…
    - Receiving “tar: Error opening archive: Failed to open '/cygdrive/c/intelFPGA/18.1/embedded/host_tools/altera/preloader/uboot-socfpga.tar.gz'”?
        - Type /usr/bin/make and press enter
        - Once it finishes run “make uboot”

- Still having errors?
  - Try commenting out lines 219/220 of the preloader make file
    - Run “tar zxf /cygdrive/c/intelFPGA/18.1/embedded/host_tools/altera/preloader/uboot-socfpga.tar.gz” then “make”
  - Try replacing “$(UNTAR) $(shell cygpath --unix "$(if $1,$1,$(if $<,$<,$(error ERROR: no input provided to gnu make function untar_recipe)))")”
    with “$(UNTAR) $(if $1,$1,$(if $<,$<,$(error ERROR: no input provided to gnu make function untar_recipe)))”

Transfer the New System + U-boot:

1. Reconnect the SD card and take note of the drive letter, e.g. E:/ (ignore any popups telling you to format the disk)
   - Make sure you’re still in the “... software/spl_bsp” directory in the EDS shell

2. To copy the preloader onto the SD card type in the shell:
   - “alt-boot-disk-util.exe -p preloader-mkpimage.bin -a write -d e”
     - “e” is whatever drive letter you had from the first step
   - “Altera Boot Disk Utility was successful” should be outputted

3. Now to copy U-boot and the .rbf:
   - type “cp uboot-socfpga/u-boot.img /cygdrive/e” and
   - “cp ../../output_files/output_file.rbf /cygdrive/e/soc_system.rbf”
   - Again, “e” is the drive letter from the first step
     - Being told no such file or directory?
       - Go to your file explorer and go to the directory with the DE10 CD -> Demonstration -> SoC_FPGA -> DE10_Standard_GHRD
       - Look for “output_file.rbf”, once found, drag it into the output_files folder and run the above command

4. To finalize, type the command “sync”. Eject the SD Card. You can also close EDS and any Quartus programs.

Boot + Ethernet Setup:

1. Plug the SD card back, connect the RS232 Mini-B USB cable to the board and computer, and reopen PuTTY

2. Look at the MSEL switches (6 switches on the bottom right of the board) and make sure they are all in the ON (‘0’) position.
    - This has to do with booting the specific .rbf file setup

The ethernet MAC address isn’t set, which we need for communication.

3. Press the red button and wait for the U-boot 5 second timeout to appear, press any key to stop it and get to the U-boot menu.

4. Run the commands:
    - “setenv eth1addr 00:11:22:33:44:55” and “saveenv”
        - (“Timeout on data busy” is okay).

5. Cycle the red button and let it go past the 5 second timeout this time.

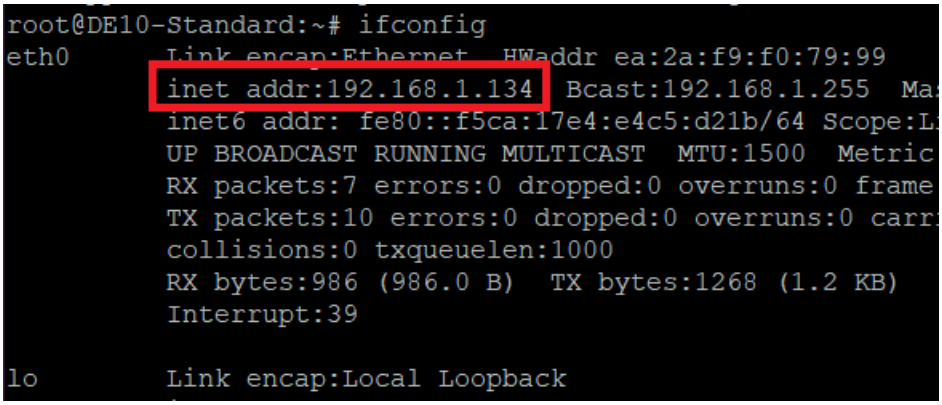
6. Linux should be booting until it gets to a login screen.
    - Username: root
    - Password is nothing, just press enter

7. Linux should now be set up to write the program that can modify the 7-segment displays… Finally...

Talk to the Router:

1. Find any spare router that has an ethernet jack, connect the board to it with an RJ45 (ethernet) cable

2. In PuTTY, type “ifconfig”, the IP address after “inet addr” is the IP address you will need to use for your Qt program and Python/C++ program on the board
    - Each time the board is powered off, this will change unless reserved for the DE10-Standard board in your router configuration settings (typically type 192.168.1.1 in browser)



Tips for writing the program on the board:

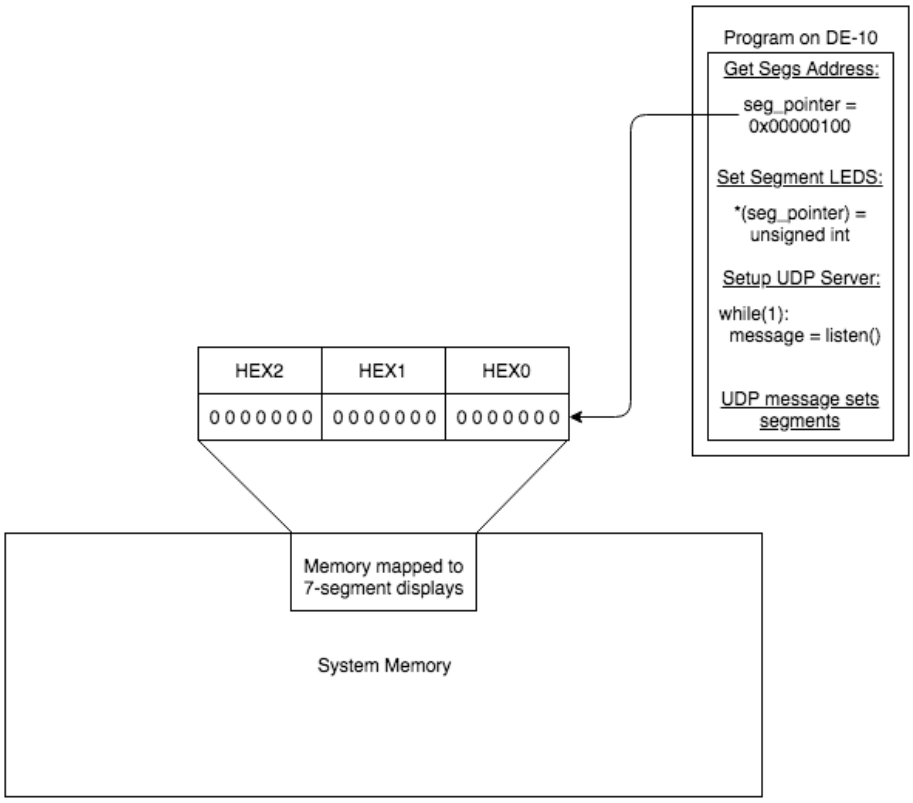
- Use a text editor such as vim to create the program, one example would be typing “vim lab3.py”, the program can then be run with “python lab3.py”

- You have two options that I’m aware of when writing code for the board itself. One is to write in C, which I realized recently that the Linux on the board actually has gcc installed which makes it possible to compile C code straight from the terminal. Another option that I opted for because of its simplicity is using Python, which fully works on the board.

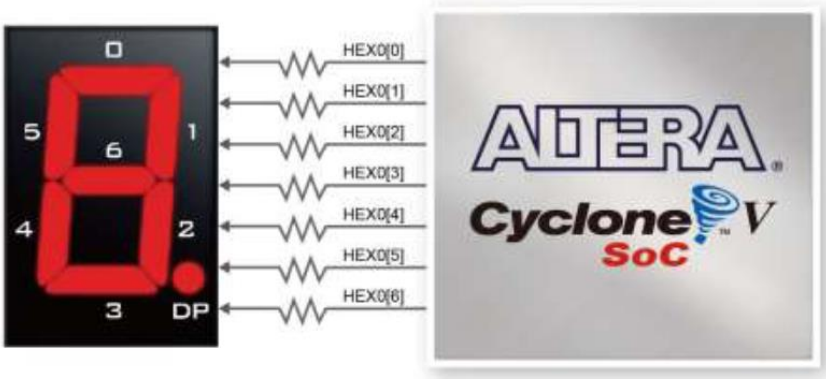
- You can see the code I wrote for my implementation here
    - Once running this, the board should display 50 | 50 across 4 LEDs
    - Try to study this code and fully understand it, if you can, modify it to be your own and explain how it works in the report

- A note about the different addresses you see in the code, I was able to base the addressing in this code off of an example given in the CD. You can find this at “CD/Demonstration/SoC_FPGA/HPS_FPGA_LED/main.c”, which will help you along if you choose to write a C program

- To illustrate what is happening in the code a little better I made a small graph:



- This is more or less pseudocode but essentially what is happening.
- One more thing, the bits that you set correspond to the image below, [6:0]
- The board utilizes common anode LEDs so each LED segment is lit when a “0” is written to a segment as can be observed in the above Python code.



QT tips:

- Once your computer is connected to the router, it can communicate with the board using its IP address. When you set up the UDP server on the board, it requires a port number to bind. The same IP address and port number will be used to connect on the QT side to the board.

- QT has a native library for networking, specifically a UDP client. This function will come especially in handy (http://doc.qt.io/qt-5/qudpsocket.html#writeDatagram-2)

- Majority of the previously developed Qt code can remain; however, it must be able to communicate with the UDP server created on-board. This requires the board's IP and port address to be initialized to establish the connection. This will require using <QtNetwork> and to implement this in the code, the .pro file should be edited to include "QT += network" at the top of the file.

- The parameter for a datagram to be written to the socket is that it has to be a string, in Qt, this can be accomplished using the QByteArray library.