Electrical and Computer Engineering

ECE453

Linux Applications: I2C and SPIDEV

1. Overview

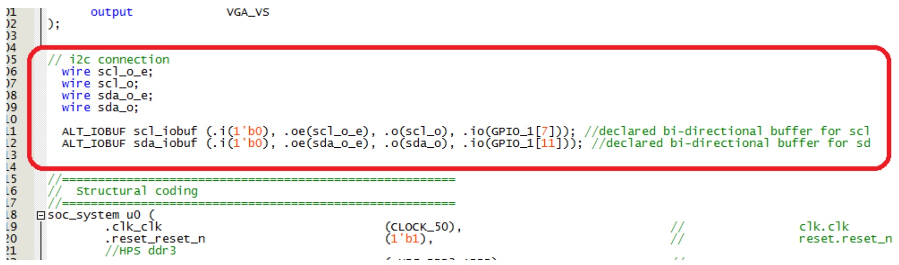
The purpose of this lab is to demonstrate how you can communicate with devices connected to the SPI and I2C interfaces provided by the DE1-SoC. Many integrated circuits communicate using SPI and I2C. This lab will demonstrate how to communicate with these devices using the SPI and I2C peripherals that are part of the ARM core in the Altera SoC.

2. Testing DE1-SoC Setup

1. Copy the following directories to C:\shared from the lab3c distribution file

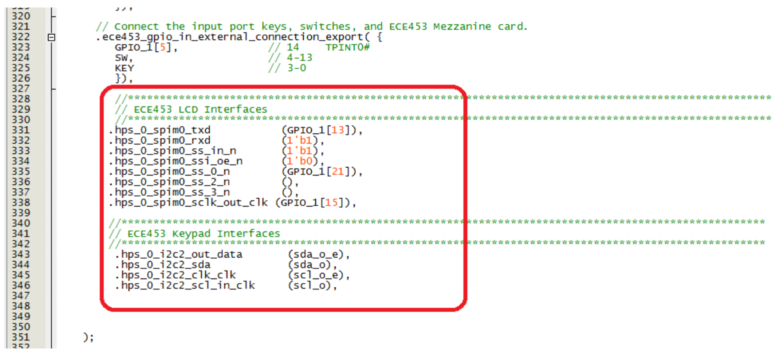
2. The DE1-SoC has been configured so that SPIM0 and I2C2 can be routed through the FPGA. If you were to look at the Quartus project, you would see connections to the top level Verilog file to specify which pins each of these interfaces is connected to.

   Directly above the declaration of the soc_system u0, the following lines of code were added that allow the I2C pins to properly behave as open drain pins.



```
        output              VGA_VS
);

// i2c connection
  wire scl_o_e;
  wire scl_o;
  wire sda_o_e;
  wire sda_o;

  ALT_IOBUF scl_iobuf (.i(1'b0), .oe(scl_o_e), .o(scl_o), .io(GPIO_1[7])); //declared bi-directional buffer for scl
  ALT_IOBUF sda_iobuf (.i(1'b0), .oe(sda_o_e), .o(sda_o), .io(GPIO_1[11])); //declared bi-directional buffer for sd

//=======================================================
//  Structural coding
//=======================================================
soc_system u0 (
        .clk_clk                         (CLOCK_50),                    //              clk.clk
        .reset_reset_n                   (1'b1),                        //              reset.reset_n
        //HPS ddr3
```

At the end of the declaration of soc_system u0, add the following lines of code that connect the specific pins of both the I2C and SPI interfaces to the pins on the external connector. The GPIO pin numbers were determined by which pins the I2C interface on the mezzanine card were connected to.



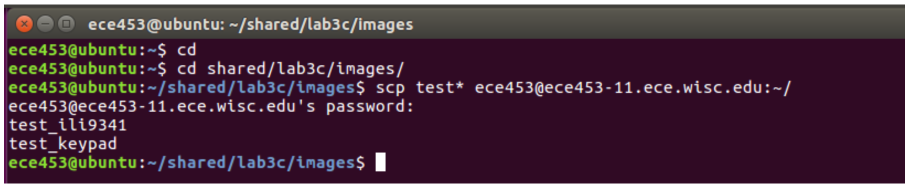
```
    });

    // Connect the input port keys, switches, and ECE453 Mezzanine card.
    .ece453_gpio_in_external_connection_export( {
       GPIO_1[5],                // 14    TPINT0#
       SW,                       // 4-13
       KEY                       // 3-0
       }),

       //*****************************************************************************************
       // ECE453 LCD Interfaces
       //*****************************************************************************************
      .hps_0_spim0_txd          (GPIO_1[13]),
      .hps_0_spim0_rxd          (1'b1),
      .hps_0_spim0_ss_in_n      (1'b1),
      .hps_0_spim0_ssi_oe_n     (1'b0),
      .hps_0_spim0_ss_0_n       (GPIO_1[21]),
      .hps_0_spim0_ss_2_n       (),
      .hps_0_spim0_ss_3_n       (),
      .hps_0_spim0_sclk_out_clk (GPIO_1[15]),

      //*****************************************************************************************
      // ECE453 Keypad Interfaces
      //*****************************************************************************************
       .hps_0_i2c2_out_data       (sda_o_e),
       .hps_0_i2c2_sda            (sda_o),
       .hps_0_i2c2_clk_clk        (scl_o_e),
       .hps_0_i2c2_scl_in_clk     (scl_o),


  );
```

The details of why these settings were changed were taken from this Altera Design Note.
https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/an/an706.pdf

3. Power on your DE1-SoC. Be sure that you have connected the serial debug cable and ethernet cables. You should also have previously completed the configuration of the Ethernet interface in the u-boot script file.

4. Before you look at how to write an application that uses the SPI or I2C interface, let's test your hardware setup with a known working application. We will copy the files in the lab3c distribution to your DE1-SoC.

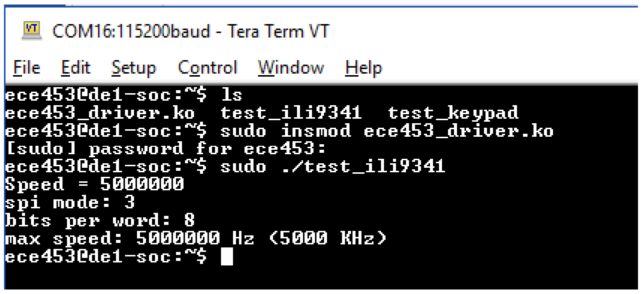
```
cd
cd shared/lab3c/images/
scp test* ece453@ece453-XX.ece.wisc.edu:~/
```



```
ece453@ubuntu:~$ cd
ece453@ubuntu:~$ cd shared/lab3c/images/
ece453@ubuntu:~/shared/lab3c/images$ scp test* ece453@ece453-11.ece.wisc.edu:~/
ece453@ece453-11.ece.wisc.edu's password:
test_ili9341
test_keypad
ece453@ubuntu:~/shared/lab3c/images$ 
```

5. On the DE1-SoC, run the test for the LCD. You will need to load the ece453 kernel module that you compiled in lab 1. Both the keypad and LCD have GPIO pins controlled by the ece453 module.

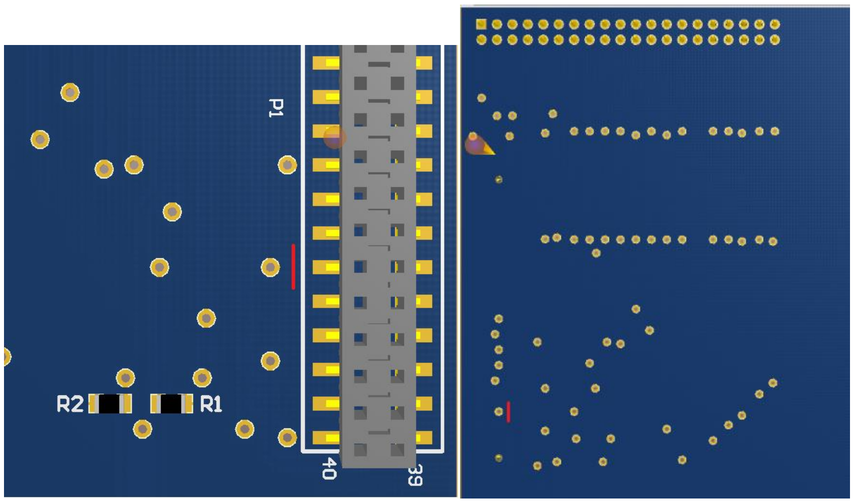
   You should observe some text print out on the display. You will also see the LCD display some text very, very slowly.


COM16:115200baud - Tera Term VT
File Edit Setup Control Window Help

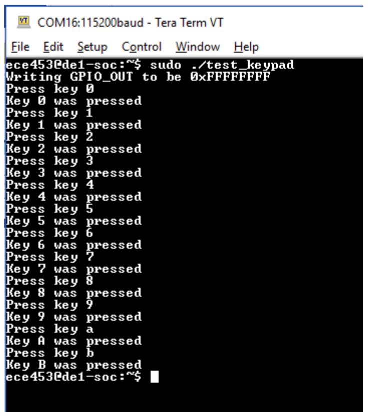
```
ece453@de1-soc:~$ ls
ece453_driver.ko  test_ili9341  test_keypad
ece453@de1-soc:~$ sudo insmod ece453_driver.ko
[sudo] password for ece453:
ece453@de1-soc:~$ sudo ./test_ili9341
Speed = 5000000
spi mode: 3
bits per word: 8
max speed: 5000000 Hz (5000 KHz)
ece453@de1-soc:~$ 
```

6. Spring 2017 – Cut Wires on I2C connection near LCD connector (red lines).

   Top Side   Bottom Side



7. On the DE1-SoC, run the test for the keypad. The test will prompt you to touch one key at a time until you have verified that all of the keys have been pressed.

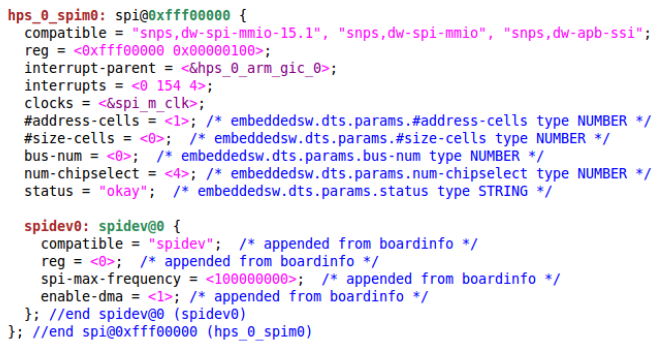

COM16:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
ece453@de1-soc:~$ sudo ./test_keypad
Writing GPIO_OUT to be 0xFFFFFFFF
Press key 0
Key 0 was pressed
Press key 1
Key 1 was pressed
Press key 2
Key 2 was pressed
Press key 3
Key 3 was pressed
Press key 4
Key 4 was pressed
Press key 5
Key 5 was pressed
Press key 6
Key 6 was pressed
Press key 7
Key 7 was pressed
Press key 8
Key 8 was pressed
Press key 9
Key 9 was pressed
Press key a
Key A was pressed
Press key b
Key B was pressed
ece453@de1-soc:~$
```

8. If either of these two applications does not work, please review the steps above and consult your instructor.

3. Completing the LCD Application

The LCD on the ECE453 mezzanine board is a SPI based LCD. Writing to a SPI based device requires two drivers in the Linux kernel: a vendor specific driver for the low level hardware details (Design Ware SPI controller) and a more generic interface called SPIDEV. The lower level driver is specified in the device tree using the “compatible” filed and is used to configure all the nitty gritty details of the SPI controller.   The image below shows how the SPI interface connected to the LCD is described in the device tree.

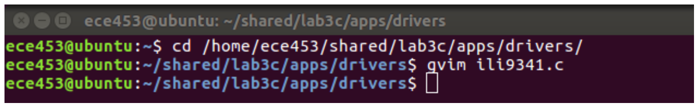


```
hps_0_spim0: spi@0xfff00000 {
  compatible = "snps,dw-spi-mmio-15.1", "snps,dw-spi-mmio", "snps,dw-apb-ssi";
  reg = <0xfff00000 0x00000100>;
  interrupt-parent = <&hps_0_arm_gic_0>;
  interrupts = <0 154 4>;
  clocks = <&spi_m_clk>;
  #address-cells = <1>; /* embeddedsw.dts.params.#address-cells type NUMBER */
  #size-cells = <0>;  /* embeddedsw.dts.params.#size-cells type NUMBER */
  bus-num = <0>;  /* embeddedsw.dts.params.bus-num type NUMBER */
  num-chipselect = <4>; /* embeddedsw.dts.params.num-chipselect type NUMBER */
  status = "okay";  /* embeddedsw.dts.params.status type STRING */

  spidev0: spidev@0 {
    compatible = "spidev";  /* appended from boardinfo */
    reg = <0>;  /* appended from boardinfo */
    spi-max-frequency = <100000000>;  /* appended from boardinfo */
    enable-dma = <1>; /* appended from boardinfo */
  }; //end spidev@0 (spidev0)
}; //end spi@0xfff00000 (hps_0_spim0)
```

The subentry in the device tree snippet above attaches the SPIDEV driver to SPIM0. The SPIDEV driver provides a more generic SPI API that can be accessed from a user space application. This is an abstraction layer of abstraction that allows you to write a user space application without knowing any of the underlying hardware details of the SPI controller. You will examine a mostly completed driver for the LCD that takes advantage of the SPIDEV driver.

9. Open the LCD driver using the following commands

```
cd /home/ece453/shared/lab3c/apps/drivers/
gvim ili9341.c
```



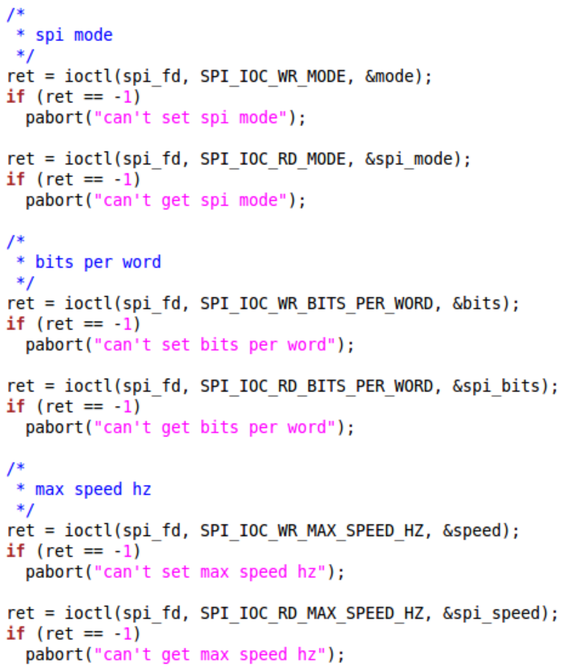
```
ece453@ubuntu: ~/shared/lab3c/apps/drivers
ece453@ubuntu:~$ cd /home/ece453/shared/lab3c/apps/drivers/
ece453@ubuntu:~/shared/lab3c/apps/drivers$ gvim ili9341.c
ece453@ubuntu:~/shared/lab3c/apps/drivers$
```

10. In “TM_ILI9341_Open” we can take a look and see how the SPI interface is configured.

    In order to specify which device you are communicating with, the user must call the “open” function and specify which SPI device is being opened. In you case, this will be **/dev/spidev32766.0**

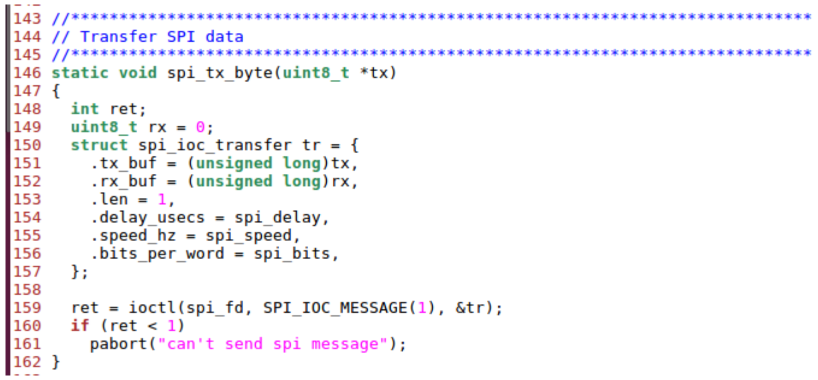
    Once the device is opened, the user space application can use a set of IO Control, or IOCTLs, functions that allow us to configure the SPI bus. IOCTLs are special functions that are used to communicate or configure hardware devices.

    An IOCTL consists of issuing a request code to the file handle indicating which operation you want to accomplish. If a driver implements IOCTLs, it must define it’s own set of request codes. The following snippet of code shows you how to configure the SPI Mode, Bits Per Word, and max speed using the SPIDEV IOCTLs. The first parameter is the file handle, the second parameter is the request code, and the third parameter is normally data being read or written.



```
/*
 * spi mode
 */
ret = ioctl(spi_fd, SPI_IOC_WR_MODE, &mode);
if (ret == -1)
  pabort("can't set spi mode");

ret = ioctl(spi_fd, SPI_IOC_RD_MODE, &spi_mode);
if (ret == -1)
  pabort("can't get spi mode");

/*
 * bits per word
 */
ret = ioctl(spi_fd, SPI_IOC_WR_BITS_PER_WORD, &bits);
if (ret == -1)
  pabort("can't set bits per word");

ret = ioctl(spi_fd, SPI_IOC_RD_BITS_PER_WORD, &spi_bits);
if (ret == -1)
  pabort("can't get bits per word");

/*
 * max speed hz
 */
ret = ioctl(spi_fd, SPI_IOC_WR_MAX_SPEED_HZ, &speed);
if (ret == -1)
  pabort("can't set max speed hz");

ret = ioctl(spi_fd, SPI_IOC_RD_MAX_SPEED_HZ, &spi_speed);
if (ret == -1)
  pabort("can't get max speed hz");
```

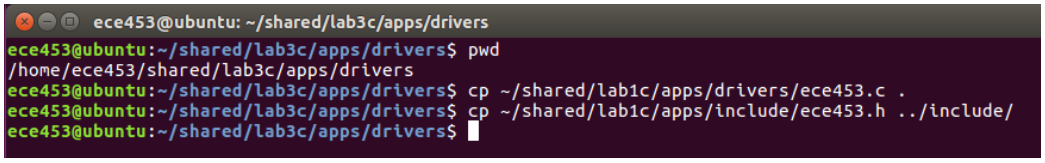
11. If you search for **`spi_tx_byte`** you can examine how data is sent to over the SPI bus. SPIDEV provides a data structure used by the IOCTL command. Filling out each of the fields of the struct indicate to the SPIDEV what data to transmit. In this case, setting "len" to 1 indicates that a single byte will be transmitted. This data structure is then passed to the SPIDEV IOCTL



```
//*****************************************************************************
// Transfer SPI data
//*****************************************************************************
static void spi_tx_byte(uint8_t *tx)
{
  int ret;
  uint8_t rx = 0;
  struct spi_ioc_transfer tr = {
    .tx_buf = (unsigned long)tx,
    .rx_buf = (unsigned long)rx,
    .len = 1,
    .delay_usecs = spi_delay,
    .speed_hz = spi_speed,
    .bits_per_word = spi_bits,
  };

  ret = ioctl(spi_fd, SPI_IOC_MESSAGE(1), &tr);
  if (ret < 1)
    pabort("can't send spi message");
}
```

You can find further documentation on SPIDEV in ~/linux-socfpga/Documentation/spi. In fact, lcd.c was based off of an example driver, spidev_test.c, found in this directory.

12. Copy the your ece453 driver files into the lab3c directory.



```
ece453@ubuntu: ~/shared/lab3c/apps/drivers
ece453@ubuntu:~/shared/lab3c/apps/drivers$ pwd
/home/ece453/shared/lab3c/apps/drivers
ece453@ubuntu:~/shared/lab3c/apps/drivers$ cp ~/shared/lab1c/apps/drivers/ece453.c .
ece453@ubuntu:~/shared/lab3c/apps/drivers$ cp ~/shared/lab1c/apps/include/ece453.h ../include/
ece453@ubuntu:~/shared/lab3c/apps/drivers$ 
```

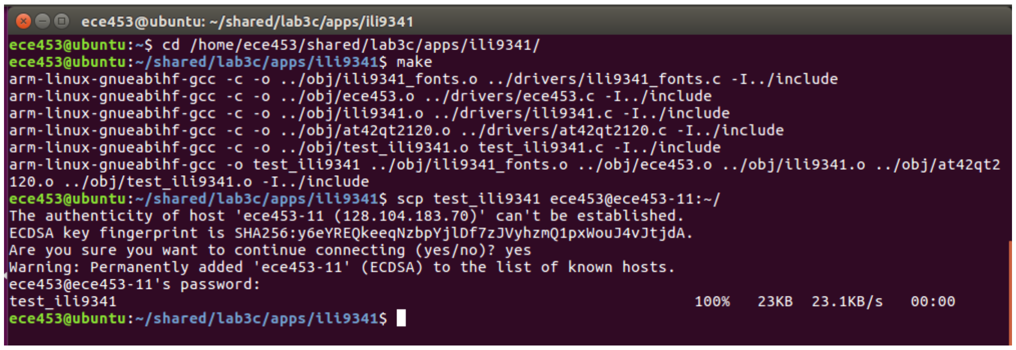
13. The ili9341 driver is 95% complete for you. This driver is nearly identical to the LCD driver you have seen in ECE353, so the details of how the LCD will not be covered here.

    What you will need to do is add the necessary lines of code to set the values the three GPIO pins (LCD_RST, LCD_CD, BckLite). Use the knowledge you gained in Lab 1 along with examining ghrd_top_level.v in the Quartus project to determine how to map each pin to a GPIO chip in the sys filesystem.

    - Complete the 6 functions near the top of the file that are used to set the value (0 or 1) to the specified GPIO pin.

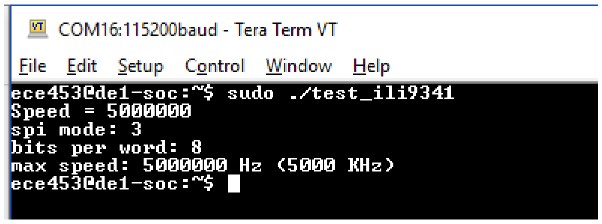
14. When you have completed the driver, compile the test application. Fix any syntax errors that you encounter.

```
cd /home/ece453/shared/lab3c/apps/ili9341/
make
scp test_ili9341 ece453@ece453-XX:~/
```



```
ece453@ubuntu:~$ cd /home/ece453/shared/lab3c/apps/ili9341/
ece453@ubuntu:~/shared/lab3c/apps/ili9341$ make
arm-linux-gnueabihf-gcc -c -o ../obj/ili9341_fonts.o ../drivers/ili9341_fonts.c -I../include
arm-linux-gnueabihf-gcc -c -o ../obj/ece453.o ../drivers/ece453.c -I../include
arm-linux-gnueabihf-gcc -c -o ../obj/ili9341.o ../drivers/ili9341.c -I../include
arm-linux-gnueabihf-gcc -c -o ../obj/at42qt2120.o ../drivers/at42qt2120.c -I../include
arm-linux-gnueabihf-gcc -c -o ../obj/test_ili9341.o test_ili9341.c -I../include
arm-linux-gnueabihf-gcc -o test_ili9341 ../obj/ili9341_fonts.o ../obj/ece453.o ../obj/ili9341.o ../obj/at42qt2
120.o ../obj/test_ili9341.o -I../include
ece453@ubuntu:~/shared/lab3c/apps/ili9341$ scp test_ili9341 ece453@ece453-11:~/
The authenticity of host 'ece453-11 (128.104.183.70)' can't be established.
ECDSA key fingerprint is SHA256:y6eYREQkeeqNzbpYjlDf7zJVyhzmQ1pxWouJ4vJtjdA.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ece453-11' (ECDSA) to the list of known hosts.
ece453@ece453-11's password:
test_ili9341                                                  100%   23KB  23.1KB/s   00:00
ece453@ubuntu:~/shared/lab3c/apps/ili9341$ 
```

15. On the DE1-SoC, run the test for the LCD. You should observe some text print out on the display.


COM16:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
ece453@de1-soc:~$ sudo ./test_ili9341
Speed = 5000000
spi mode: 3
bits per word: 8
max speed: 5000000 Hz (5000 KHz)
ece453@de1-soc:~$ 
```

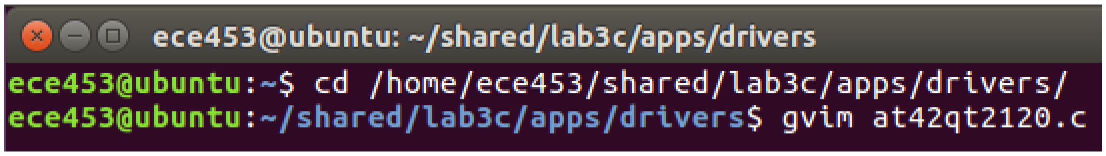
4. Completing the Keypad Application

The Keypad on the ECE453 mezzanine board is an I2C based IC. Communicating with a I2C based device requires you to use a different set of IOCTL command supported by the I2C bus in Linux. The IOCTL commands allow you to access the I2C bus through the /dev/i2c-X. The X indicates which I2C bus you are communicating with. In our case, we are using /dev/i2c-2.

You will use the following IOCTL commands to interface with the devices on i2c-2.

```
//Open the file handle with read/write privileges
i2c_fh = open(I2C_DEV, O_RDWR);

// Set the slave address
ioctl(i2c_fh, I2C_SLAVE, AT42QT2120_SLAVE_ADDR) ;

// close the device
close(i2c_fh);

// Write 2 bytes of data

if (write(i2c_fh, data, 2) != 2) {

   perror("Failed to Write data");

}

// Read 1 Byte of data
if (read(i2c_fh, &data, 1) != 1) {

   perror("Failed to read data");

}
```

16. Open the at42qt2120 driver file.



```
ece453@ubuntu: ~/shared/lab3c/apps/drivers
ece453@ubuntu:~$ cd /home/ece453/shared/lab3c/apps/drivers/
ece453@ubuntu:~/shared/lab3c/apps/drivers$ gvim at42qt2120.c
```

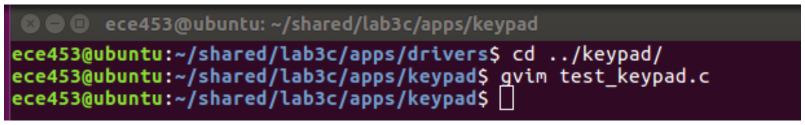
17. Complete the following functions.

```
static void     at42qt2120_assert_reset(void);
static void     at42qt2120_deassert_reset(void);
void            at42qt2120_write(uint8_t reg, uint8_t value);
uint8_t         at42qt2120_read(uint8_t reg);
```

<u>Remember and I2C read requires you to write the address of the register first, then read the data!</u>

18. Open the application file that will be used to test the touch sensor. This test file uses the functions you completed in at42qt2120.c. This test application has been completed to show you how to use the functions you have written. When you are done examining this file you can close this file.

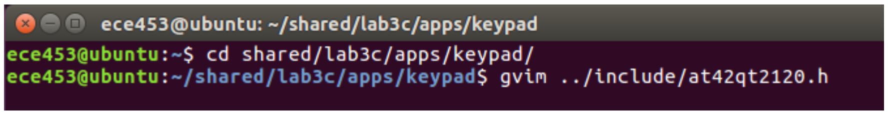
```
cd ../keypad/
gvim test_keypad.c
```



```
ece453@ubuntu: ~/shared/lab3c/apps/keypad
ece453@ubuntu:~/shared/lab3c/apps/drivers$ cd ../keypad/
ece453@ubuntu:~/shared/lab3c/apps/keypad$ gvim test_keypad.c
ece453@ubuntu:~/shared/lab3c/apps/keypad$ 
```

19. Before you compile the application, you will need to define the I2C slave address of the AT42QT2120 and to define the register addresses that return the status of the keypad. You will need to properly define the macros in this file where you see the words `// ADD CODE.` You can find this information in the AT42QT2120 data sheet.

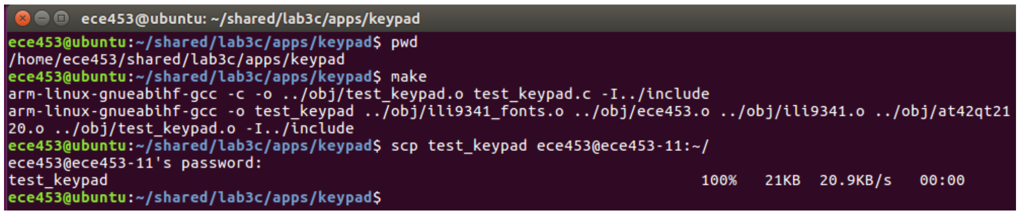
```
gvim ../include/at42qt2120.h
```



```
ece453@ubuntu: ~/shared/lab3c/apps/keypad
ece453@ubuntu:~$ cd shared/lab3c/apps/keypad/
ece453@ubuntu:~/shared/lab3c/apps/keypad$ gvim ../include/at42qt2120.h
```

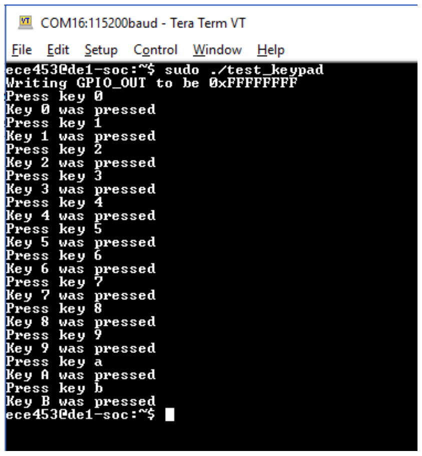
20. When you are done, you can compile the application and fix any compile errors you encounter. After the compile process is successful, copy the application to the DE1-SoC.

```
pwd
make
scp test_keypad ece453@ece453-11:~/
```



```
ece453@ubuntu: ~/shared/lab3c/apps/keypad
ece453@ubuntu:~/shared/lab3c/apps/keypad$ pwd
/home/ece453/shared/lab3c/apps/keypad
ece453@ubuntu:~/shared/lab3c/apps/keypad$ make
arm-linux-gnueabihf-gcc -c -o ../obj/test_keypad.o test_keypad.c -I../include
arm-linux-gnueabihf-gcc -o test_keypad ../obj/ili9341_fonts.o ../obj/ece453.o ../obj/ili9341.o ../obj/at42qt21
20.o ../obj/test_keypad.o -I../include
ece453@ubuntu:~/shared/lab3c/apps/keypad$ scp test_keypad ece453@ece453-11:~/
ece453@ece453-11's password:
test_keypad                                                    100%   21KB  20.9KB/s   00:00
ece453@ubuntu:~/shared/lab3c/apps/keypad$
```

21. On the DE1-SoC, run the test for the keypad. The test will prompt you to touch one key at a time until you have verified that all of the keys have been pressed.


COM16:115200baud - Tera Term VT
File Edit Setup Control Window Help

```
ece453@de1-soc:~$ sudo ./test_keypad
Writing GPIO_OUT to be 0xFFFFFFFF
Press key 0
Key 0 was pressed
Press key 1
Key 1 was pressed
Press key 2
Key 2 was pressed
Press key 3
Key 3 was pressed
Press key 4
Key 4 was pressed
Press key 5
Key 5 was pressed
Press key 6
Key 6 was pressed
Press key 7
Key 7 was pressed
Press key 8
Key 8 was pressed
Press key 9
Key 9 was pressed
Press key a
Key A was pressed
Press key b
Key B was pressed
ece453@de1-soc:~$ 
```

Demonstrate to your TA that your Linux applications are functioning correctly.

5. What's Next

Up to this point, you've written applications that interface with device drivers that haven written by other developers. The major differentiator of the Altera SoC is that you can define custom hardware peripherals in the FPGA portion of the SoC. In order to communicate with this custom peripheral, we will need to write a custom Linux device driver so that you can utilize this feature of the Altera SoC. In the next lab, you will develop a custom Verilog FPGA module and device driver for the the WS2812B LEDs found on the ECE453 mezzanine card.