



# Electrical and Computer Engineering

**ECE453**

## **Altera SoC Introduction**



## Overview

The DE1-SOC development platform is designed around a Altera Cyclone® V SE 5CSEMA5F31C6N SoC (System-on-Chip). This is a dual core 800MHz Cortex-A9 processor with 1GB of DDR3 memory, a 2 port USB host, and a 1 gigabit Ethernet PHY.

A good question to ask is how does this processor differ from the microcontrollers you've experienced in your previous course work? The simplest way to answer that is to say that the Cortex-A9 family of ARM processors is much more powerful, but also much more complex. The Cortex-A9 family supports a memory management unit (MMU) that will allow us to implement virtual memory and support many of the constructs required for simultaneous multithreading. The gigabit Ethernet PHY allows for the device to support complex network data transactions. These capabilities allow the Altera SOC to handle an entire range of computing environments that a typical microcontroller simply cannot. As is often the case, adding additional capabilities leads to additional complexities.

Due to the added complexity, we will not simply write our own C code to configure and utilize the Altera SoC and its set of peripherals. The development time to do that would be far too time consuming to cover in a single semester. Instead, we will need to leverage the Linux kernel and GNU filesystem to help us to manage many of the complexities that arise

## 1. DE1-SOC Linux Setup

### 1.1. Downloading the Linux Filesystem

In order to familiarize ourselves with the DE1-SOC board, we will boot the DE1-SOC using the a custom Linux image configured for the DE1-SOC.

0. [Windows] Download ECE453\_4gb.img to C:\shared\default\_image

NOTE: The Linux VM has a shared folder that allows access to the files from both Windows and Linux. The [Windows] notation indicates that you are using the Windows Host when entering those commands. Instructions that begin with [Linux-VM] will be issued in the Linux VM. Instructions that begin with [DE1-SoC] will be issued in the serial terminal of the DE1-SoC.

### 1.2. Transferring Linux Filesystem to Your MicroSD Card

In order to boot the filesystem image on the DE1-SOC, we will need to transfer the image a microSD card using a Linux VM (virtual machine) found on the lab PCs. This virtual machine provides a Ubuntu 16.04 environment where most of our Linux software development will take place. You will use the Linux VM to properly format the SD card with the Linux filesystem.

1. [Windows] Start the Linux VM by double clicking the VMWare Workstation Logo.



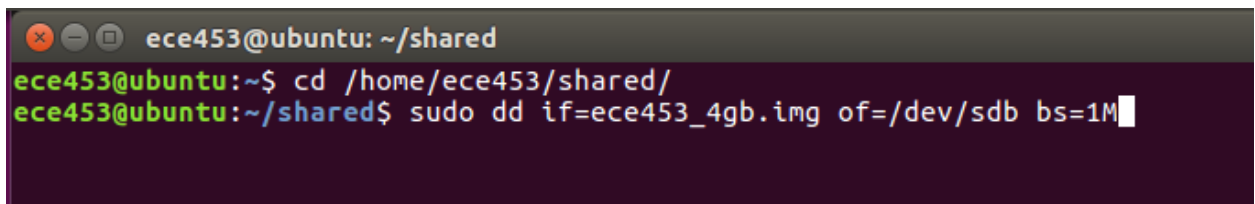
2. Plug in the microSD card using the microSD to USB adapter into your PC

3. [Windows] The Linux VM needs to have access to the microSD card. This can be done by selecting Player -> Removable Devices -> Genesys -> Connect
4. [Linux-VM] Log into the Linux VM. Your TA will supply you with the login credentials.
5. [Linux-VM] Open a Terminal command prompt



6. [Linux-VM] The following Linux commands can be used to transfer the filesystem image to the microSD card.

```
sudo dd if=ece453_4gb.img of=/dev/sdb bs=1M
```

A screenshot of a terminal window. The title bar shows 'ece453@ubuntu: ~/shared'. The terminal has a dark purple background. The first line shows the prompt 'ece453@ubuntu:~\$' followed by the command 'cd /home/ece453/shared/'. The second line shows the prompt 'ece453@ubuntu:~/shared\$' followed by the command 'sudo dd if=ece453\_4gb.img of=/dev/sdb bs=1M' with a white cursor at the end.

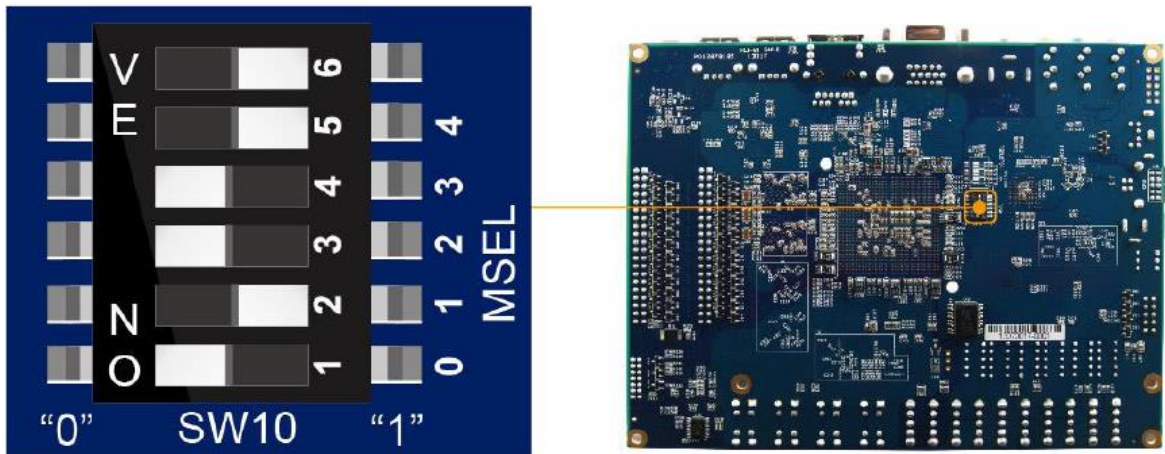
```
ece453@ubuntu: ~/shared
ece453@ubuntu:~$ cd /home/ece453/shared/
ece453@ubuntu:~/shared$ sudo dd if=ece453_4gb.img of=/dev/sdb bs=1M
```

NOTES: The names of the files that you need to unmount may be different on your system. Use tab complete to determine how your microSD card was labeled. unmount all filesystems under /media/ece453.

The dd command stands for disk dump. We supply in input source with the “if=” and the output source using “of=”. The microSD card in this case is /dev/sdb. You can type “man dd” for more information on the dd command

### 1.3. Booting Linux

7. When the dd command finishes, unplug the microSD-to-USB adapter and plug the microSD card into the DE1-SOC.
8. Make sure that the DE1-SOC is configured for so that MSEL[4..0] are set to 01010. **The image below shows a different setting, so make sure you use 01010!**



Source: DE1-SoC\_Getting\_Started\_Guide.pdf

9. Plug in the supplied power cord, USB-B, and mini USB cable into the DE1-SOC.
10. [\[Windows\]](#) Open Putty or TeraTerm and connect to the COM port for the Altera SoC. This should be COM3 or greater. Connect at 115200 baud.

## 1.4. Configuring U-Boot

In order for the Ethernet interface to function properly on the DE1-SOC, we need to set the MAC address to an address that is allowed on the CAE network. This will allow us to transfer files between the Linux VM and the DE1-SoC.

You will only need to configure the network interface once. The exception here is if you re-image the SD card, you will need to re-run these commands.

### 1.4.1. Setting the MAC address

11. Connect the mini-USB cable, power cord, and then power on the DE1-SoC. Press a key within 5 seconds to drop into the uBoot boot loader. From the boot loader prompt, execute the following commands. **Be sure to use the MAC address found on the label for your DE1-SoC board.**

```
[DE1-SoC] SOCFPGA_CYCLONE5 # setenv ethaddr 00:0a:35:XX:00:00
```

12. Boot the Board

```
[DE1-SoC] SOCFPGA_CYCLONE5 # run bootcmd
```

13. **[Windows]** In the serial console, you should observe that the DE1-SoC boots to a Linux prompt. When you reach the login prompt, enter the username '**ece453**'. The password is '**TMS320**'

## 1.5. Configuring the Altera SoC

The Cortex-A9 architecture is an ARM core that is used in many different vendors' microprocessors. Each vendor customizes the peripheral set attached to the A9 core to meet the computing requirements for their desired market.

What is unique about the Altera SoC (and Xilinx Zynq) A9 variants is that in addition to the Cortex-A9 core, the Altera Soc provides an FPGA fabric that can be used to design custom peripheral devices. The peripheral devices are designed in Verilog (or VHDL) and can be accessed from the Cortex-A9 core. The FPGA fabric allows for the Altera SoC to be easily customized to meet a broader range of applications than a standard Cortex-A9 core itself. This portion of the lab will show you how to add a new peripheral device to the DE1-SoC.

You will add a custom module to the FPGA fabric to control the LEDs, push buttons, and switches found on the DE1-SoC. In later labs, you will also add support to interface with the ECE453 mezzanine card that your partners are designing.

14. [Windows] Download the ECE453 SoC distribution from the course website. Unzip the contents into C:\shared\lab1c
15. [Windows] Double click the Quartus Prime project (.qpf) found in C:\shared\lab1c\SoC\ECE453
16. [Windows] In order to add a new peripheral device to our design, we will need to use the Qsys tool. You can find Qsys under "Tools" on the top menu. When the Qsys window opens, select soc\_system.qsys from the Open dialog.

You should see the following representation of the base Altera SoC system. The ARM-A9 processor is (called hps\_0) in the structure below.

System: soc\_system Path: hps\_0

Use	Connections	Name	Description	Export	Clock	Base	Er
<input checked="" type="checkbox"/>		<b>hps_0</b> Arria V/Cyclone V Hard Processor System					
		f2h_cold_reset_req	Reset Input	hps_0_f2h_cold_reset_r...			
		f2h_debug_reset_req	Reset Input	hps_0_f2h_debug_reset...			
		f2h_warm_reset_req	Reset Input	hps_0_f2h_warm_reset...			
		f2h_stm_hw_events	Conduit	hps_0_f2h_stm_hw_ev...			
		spim0	Conduit	hps_0_spim0			
		spim0_sclk_out	Clock Output	hps_0_spim0_sclk_out	hps_0_spim...		
		uart1	Conduit	hps_0_uart1			
		i2c2_sd_in	Clock Input	hps_0_i2c2_scl_in	exported		
		i2c2_clk	Clock Output	hps_0_i2c2_clk	hps_0_i2c2_clk		
		i2c2	Conduit	hps_0_i2c2			
		memory	Conduit	memory			
		hps_io	Conduit	hps_0_hps_io			
		h2f_reset	Reset Output	hps_0_h2f_reset			

17. [Windows] If you scroll to the end of the System Contents using the vertical scroll bar, you can see a custom ece453 peripheral located in the programmable logic of the SoC. This is the peripheral that you will modify as part of this class. You should **NOT** modify anything in Qsys.

<input checked="" type="checkbox"/>		<b>ece453_0</b> ece453	Reset Output	Double-click to export				
		avalon_slave_0	Avalon Memory Mapped Slave	Double-click to export	[dock_sink]	0x0000_0000	0x0000_003f	
		dock_sink	Clock Input	Double-click to export	[dock_sink]			
		dock_reset	Reset Input	Double-click to export	[dock_sink]			
		interrupt_sender	Interrupt Sender	Double-click to export	[dock_sink]			
		gpio_in	Conduit	ece453_0_gpio_in	[dock_sink]			
		gpio_out	Conduit	ece453_0_gpio_out	[dock_sink]			

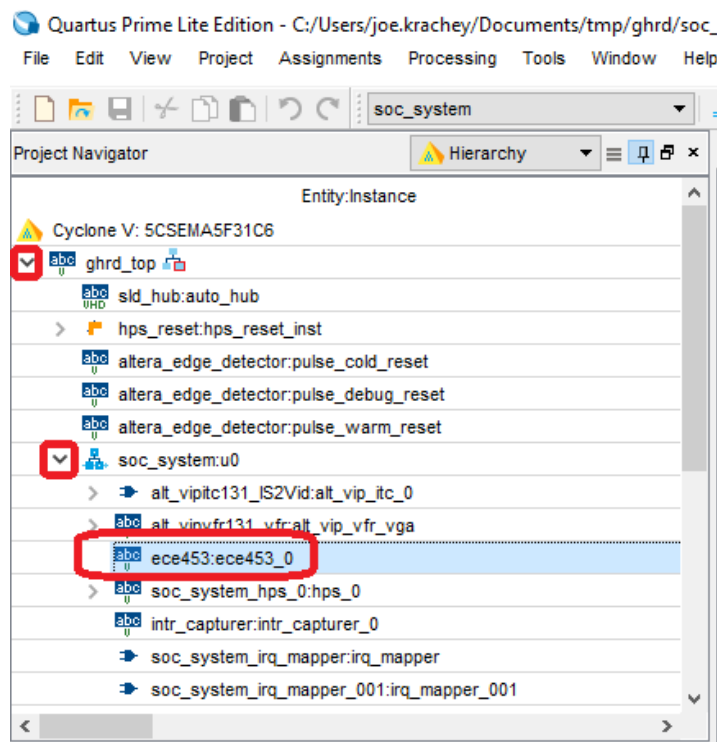
18. [Windows] Select the “Generate HDL” button near the lower right hand corner of the screen and then select “Generate”. This process takes about a minute and will finish with some warnings, but should not have any errors.
19. [Windows] Select the “Finish” button to close Qsys. Quartus is going to issue a warning message to you after Qsys closes. You can hit OK and ignore this message.
20. [Windows] Double click on the ghrd\_top.v file in the project navigator. ghrd\_top.v is the Verilog file that is used to describe how the HPS and FPGA cores are interconnected.
21. [Windows] If you look at line 406 of ghrd\_top.v, you will see that the ece453 peripheral is connected to a set of switches, LEDs, and push buttons on the SoC. In addition to these peripheral devices, you will also notice that there are other signals coming out of the ece453 module connected to several GPIO pins. These GPIO pins are selected so they correctly interface with the ECE453 mezzanine card you built in a previous lab.



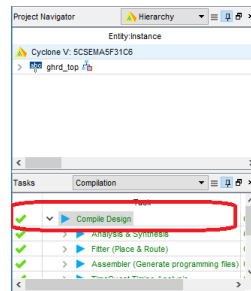
The supplied project has already defined the pin locations for these.

IO Device	Pin Names
Red LEDs	LEDR[9:0]
Slide Swithces	SW[9:0]
Push Buttons	KEY[3:0]
Mezzanine Connector 0	GPIO_0[35:0]
Mezzanine Connector 1	GPIO_1[35:0]

22. [Windows] The GPIO pins connected to LEDR[9:0], SW[9:0], and KEY[3:0] are controlled by the custom ece453 peripheral located in the programmable logic of the SoC. You can view the ece453 peripheral by examining the modules Verilog code. You can open the Verilog file by expanding the ghrd\_top instance in the Project Navigator, then expanding soc\_system:u0, and then double clicking on ece453:ece453\_0. Do **NOT** modify this file in this lab!



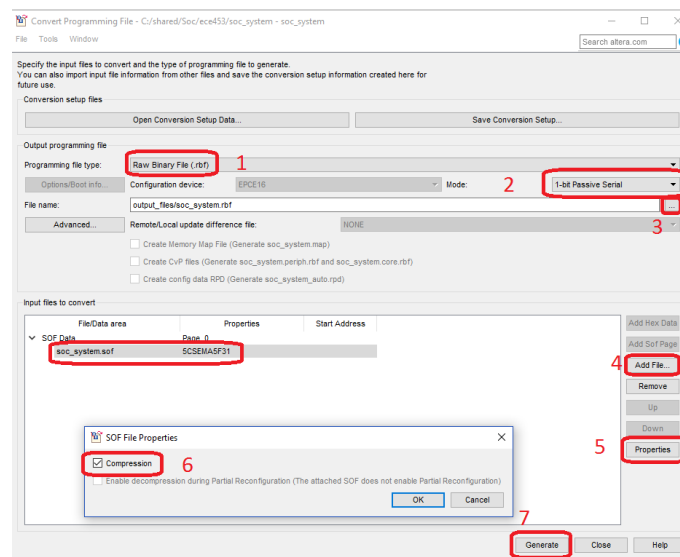
23. [Windows] Once this is done, you can double click on the “Compile Design” entry in the Tasks pane on the left. It will take roughly 10 minutes to build your system.



24. [Windows] You now need to convert the default programming file type (.sof) to a raw binary format (.rbf). This is done using File→Convert Programming Files.

In the window that pops up,

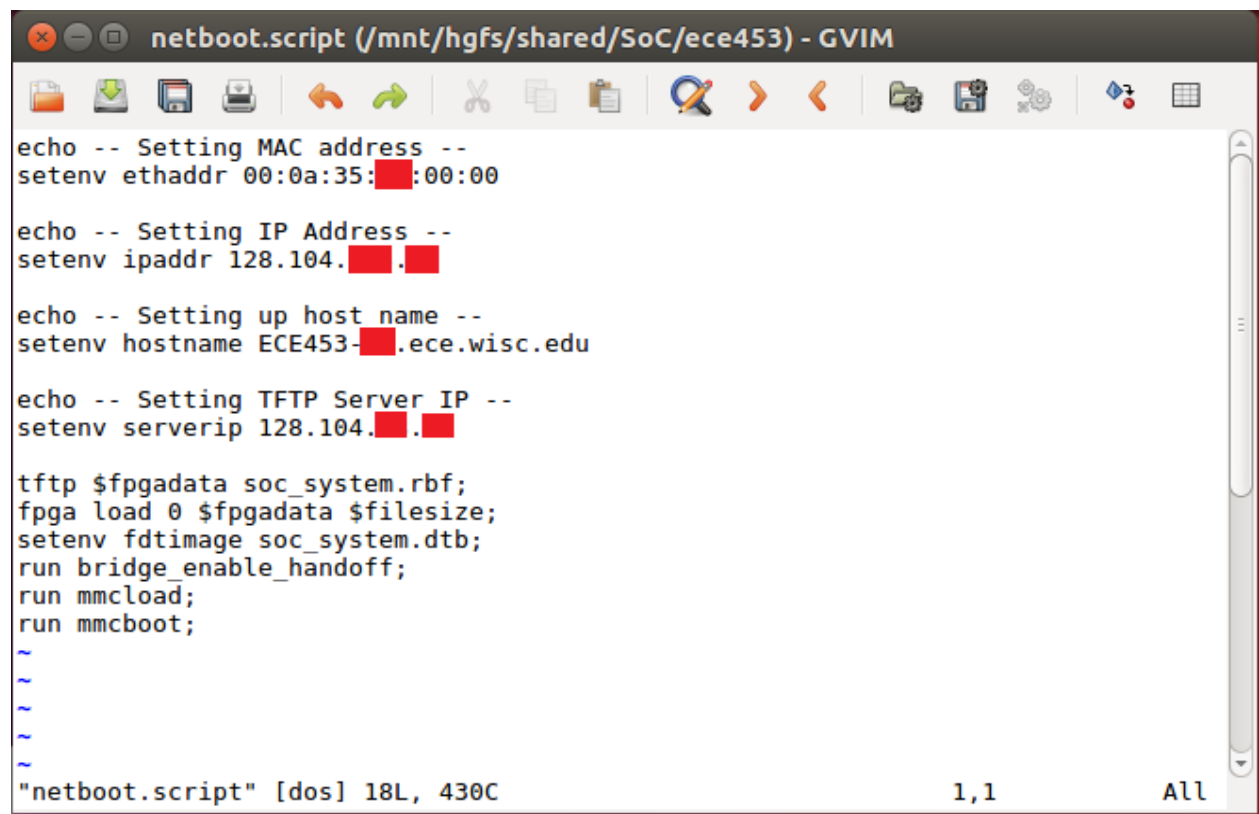
1. Select a programming file type as Raw Binary File
2. Select mode 1-bit Passive.
3. Select a file name of output\_files/soc\_system.rbf
4. Select Add File and navigate to output\_files/soc\_system.sof
5. Click on Properties
6. Select Compression
7. Generate the File



## 1.6. Programming the DE1-SoC

Now that you have your DE1-SoC design built, you will need to program the DE1-SoC and test if we are able to communicate with the GPIO pins that we have added to the design. Instead of modifying the SD card each time we generate a new image for the DE1-SoC, we are going to configure U-boot so that it downloads the new programming file from your Lab PC. In future labs and in your project, you will modify the ECE453 peripheral module. Configuring u-boot to program the DE1-SoC using the TFTP server will be a convenient way to “activate” any changes you make by simply rebooting the DE1-SoC.

25. Plug in an Ethernet cable into the DE1-SoC.
26. [Linux-VM] Edit netboot.script in /home/ece453/shared/lab1c/SoC/u-boot to match the network information for your DE1-SoC. You will need to modify the information in the red boxes to match your DE1-SoC.



```
netboot.script (/mnt/hgfs/shared/SoC/ece453) - GVIM

echo -- Setting MAC address --
setenv ethaddr 00:0a:35: :00:00

echo -- Setting IP Address --
setenv ipaddr 128.104. :

echo -- Setting up host name --
setenv hostname ECE453- .ece.wisc.edu


echo -- Setting TFTP Server IP --
setenv serverip 128.104. :

tftp $fpgadata soc_system.rbf;
fpga load 0 $fpgadata $filesize;
setenv fdtimage soc_system.dtb;
run bridge_enable_handoff;
run mmcload;
run mmcboot;

~
~
~
~

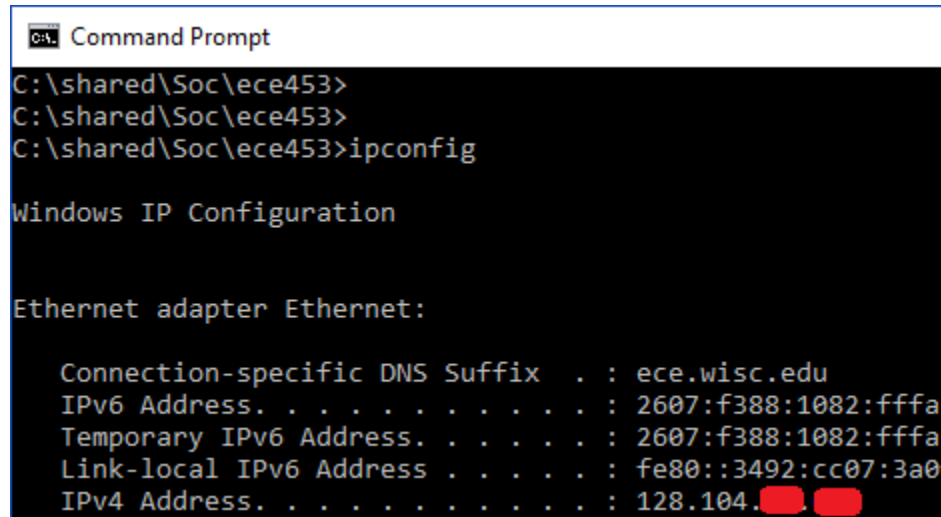
"netboot.script" [dos] 18L, 430C 1,1 All
```

- a. Your MAC address can be found on the top of your DE1-SoC board.
- b. Your IP address can be found by running ‘ifconfig eth0’ in the serial terminal connected to the DE1-SoC. NOTE: Do NOT use the IP address below! Use the one returned in the serial terminal!!!



```
ece453@de1-soc:~$ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0a:35:cb:00:00
          inet addr:128.104.183.70  Bcast:128.104.183.255  Mask:255.255.252.0
```

- c. Your Host name can be found on the top of your DE1-SoC board
- d. To find the IP address of the TFTP server, open a command prompt in Windows and type 'ipconfig'.



```

C:\shared\Soc\ece453>
C:\shared\Soc\ece453>
C:\shared\Soc\ece453>ipconfig

Windows IP Configuration

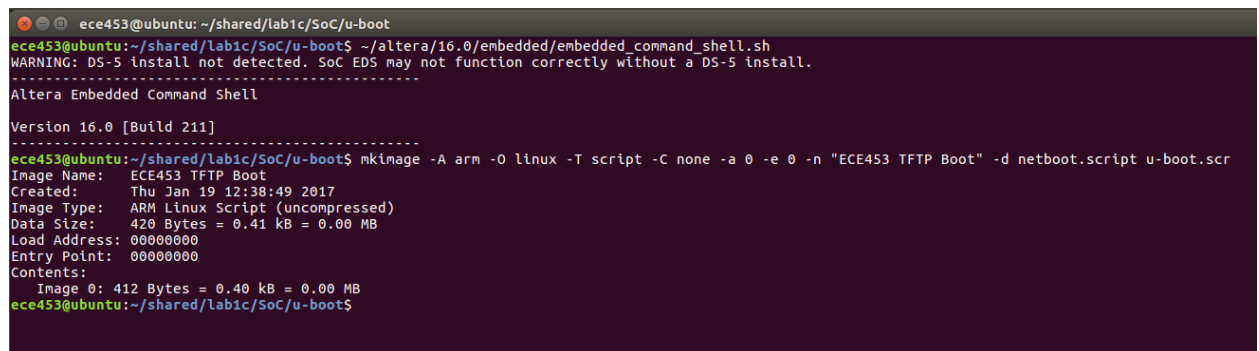
Ethernet adapter Ethernet:

    Connection-specific DNS Suffix  . : ece.wisc.edu
    IPv6 Address. . . . .           : 2607:f388:1082:fffa
    Temporary IPv6 Address. . . . . : 2607:f388:1082:fffa
    Link-local IPv6 Address . . . . . : fe80::3492:cc07:3a0
    IPv4 Address. . . . .           : 128.104.
  
```

27. [Linux-VM] Save netboot.script
28. [Linux-VM] You will now compile netboot.script into a u-boot script image that will execute these commands found in the script each time u-boot is started.

```
~/altera/16.0/embedded/embedded_command_shell.sh
```

```
mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "ECE453
TFTP Boot" -d netboot.script u-boot.scr
```



```

ece453@ubuntu: ~/shared/lab1c/SoC/u-boot
ece453@ubuntu:~/shared/lab1c/SoC/u-boot$ ~/altera/16.0/embedded/embedded_command_shell.sh
WARNING: DS-5 install not detected. SoC EDS may not function correctly without a DS-5 install.
-----
Altera Embedded Command Shell
Version 16.0 [Build 211]
-----
ece453@ubuntu:~/shared/lab1c/SoC/u-boot$ mkimage -A arm -O linux -T script -C none -a 0 -e 0 -n "ECE453 TFTP Boot" -d netboot.script u-boot.scr
Image Name:   ECE453 TFTP Boot
Created:      Thu Jan 19 12:38:49 2017
Image Type:   ARM Linux Script (uncompressed)
Data Size:    420 Bytes = 0.41 kB = 0.00 MB
Load Address: 00000000
Entry Point:  00000000
Contents:
  Image 0: 412 Bytes = 0.40 kB = 0.00 MB
ece453@ubuntu:~/shared/lab1c/SoC/u-boot$
  
```

29. [DE1-SoC] Stop Linux from the serial terminal.

```
sudo halt
```

```
ece453@de1-soc:~$ sudo halt
[sudo] password for ece453:
[ OK ] Stopped target Timers.
[ OK ] Stopped target Graphical Interface.
Stopping Light Display Manager...
```

30. Power Off the DE1-SoC

31. Remove the microSD card from the DE1-SoC and plug in the microSD card using the microSD to USB adapter into your PC

32. [Linux-VM] Copy u-boot.scr to the microSD card.

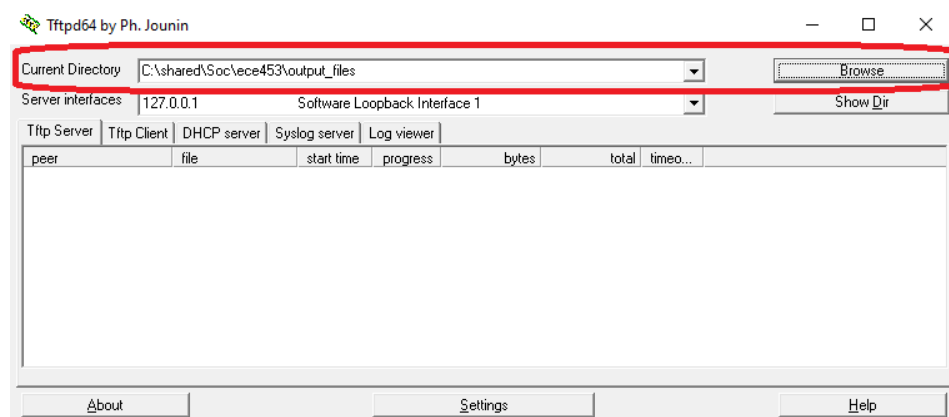
```
cp u-boot.scr /media/ece453/ALTERA/u-boot.scr
```

33. [Linux-VM] Eject the microSD card.

```
ece453@ubuntu: ~/shared/lab1c/SoC/u-boot
ece453@ubuntu:~/shared/lab1c/SoC/u-boot$ cp u-boot.scr /media/ece453/ALTERA/u-boot.scr
```

34. [Windows] U-Boot is now configured to network boot, but you still need to enable the TFTP server on your PC.

Open the TFTP64 server shortcut on your Windows PC. When the TFTP64 application opens, click on the “Browse” button and navigate to C:\shared\SoC\ece453\output\_files.



35. Plug the microSD card into the DE1-SoC and power the board on.

36. [Windows] From the serial terminal, you should see that u-boot pulls the programmable logic programming file (soc\_system.rbf) from the TFTP server and then boots to Linux.

## 2. Testing FPGA Controlled GPIO Pins From Linux Command Prompt

Now that you have successfully booted into Linux, let's take a look at how we can read and write to the GPIO controlled by the FPGA from the Linux operating system. The GPIO drivers that you compiled into the Linux kernel expose the GPIO pins through a virtual filesystem called the sysfs. The sysfs filesystem is used to configure specific hardware by reading and writing to a set of virtual files.

Before we can access the GPIO pins from the Linux Kernel, we need to compile a Linux kernel driver that has been written specifically for the ECE453 programmable logic module that was built into the DE1-SoC image. We will examine how to write the kernel driver in a different lab.

37. [\[Linux-VM\]](#) Compile the kernel driver

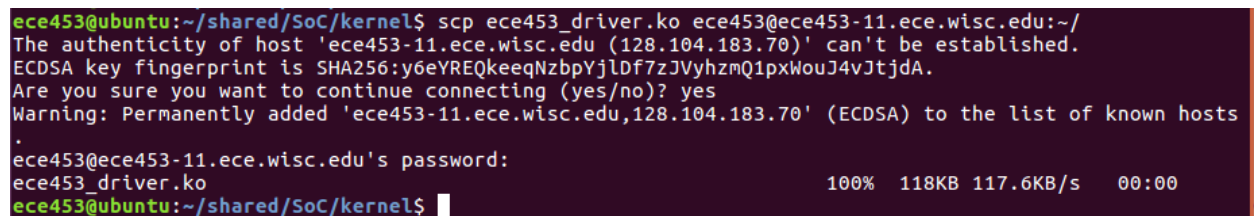
```
cd /home/ece453/shared/lab1c/kernel/ece453
make
```



```
ece453@ubuntu: ~/shared/lab1c/kernel/ece453
ece453@ubuntu:~$ cd /home/ece453/shared/lab1c/kernel/ece453
ece453@ubuntu:~/shared/lab1c/kernel/ece453$ make
make -C ~/linux-socfpga/ M=/home/ece453/shared/lab1c/kernel/ece453 modules
make[1]: Entering directory '/home/ece453/linux-socfpga'
CC [M] /home/ece453/shared/lab1c/kernel/ece453/ece453_driver.o
/home/ece453/shared/lab1c/kernel/ece453/ece453_driver.c: In function 'ece453_probe':
/home/ece453/shared/lab1c/kernel/ece453/ece453_driver.c:308:1: warning: label 'err_release_region' defined but not used [-Wunused-label]
err_release_region:
^
Building modules, stage 2.
MODPOST 1 modules
CC /home/ece453/shared/lab1c/kernel/ece453/ece453_driver.mod.o
LD [M] /home/ece453/shared/lab1c/kernel/ece453/ece453_driver.ko
make[1]: Leaving directory '/home/ece453/linux-socfpga'
ece453@ubuntu:~/shared/lab1c/kernel/ece453$
```

38. [\[Linux-VM\]](#) Copy the kernel driver to the DE1-SoC (be sure to fill in the XX with your DE1-SoC information!)

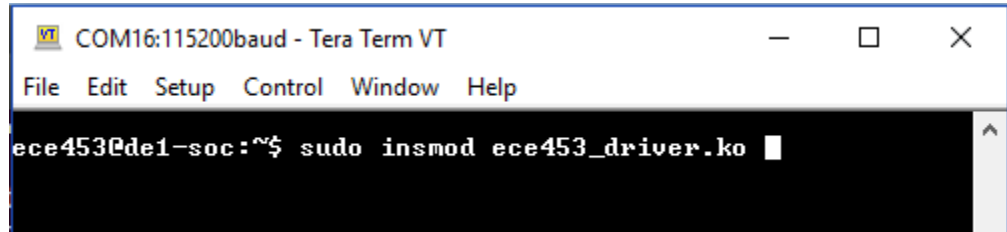
```
scp ece453_driver.ko ece453@ece453-XX.ece.wisc.edu:~/
```



```
ece453@ubuntu:~/shared/SoC/kernel$ scp ece453_driver.ko ece453@ece453-11.ece.wisc.edu:~/
The authenticity of host 'ece453-11.ece.wisc.edu (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.ece.wisc.edu,128.104.183.70' (ECDSA) to the list of known hosts
ece453@ece453-11.ece.wisc.edu's password:
ece453_driver.ko                                100% 118KB 117.6KB/s 00:00
ece453@ubuntu:~/shared/SoC/kernel$
```

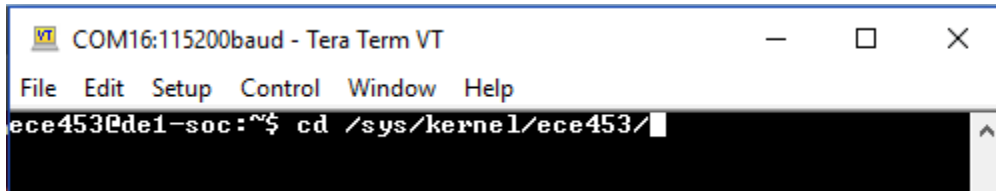
39. [DE1-SoC] From the serial terminal, load the kernel driver into the kernel. Loading the ece453\_driver into the kernel is going to create three virtual files that will give you access to the registers found in the ECE453 custom peripheral in the programmable logic of the DE1-SoC.

```
sudo insmod ece453_driver.ko
```

A screenshot of a Tera Term VT serial terminal window. The title bar reads 'COM16:115200baud - Tera Term VT'. The menu bar includes 'File', 'Edit', 'Setup', 'Control', 'Window', and 'Help'. The terminal text shows the prompt 'ece453@de1-soc:~\$' followed by the command 'sudo insmod ece453\_driver.ko' and a cursor. A vertical scrollbar is visible on the right side of the terminal window.

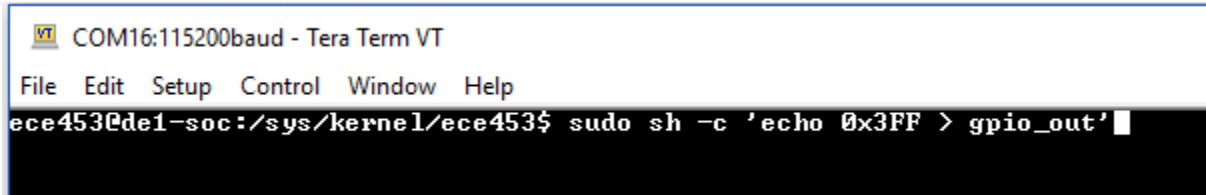
40. [DE1-SoC] Enter the directory that contains the virtual files used to control GPIO pins and lists the files that are found.

```
cd /sys/kernel/ece453/
```

A screenshot of a Tera Term VT serial terminal window. The title bar reads 'COM16:115200baud - Tera Term VT'. The menu bar includes 'File', 'Edit', 'Setup', 'Control', 'Window', and 'Help'. The terminal text shows the prompt 'ece453@de1-soc:~\$' followed by the command 'cd /sys/kernel/ece453/' and a cursor. A vertical scrollbar is visible on the right side of the terminal window.

41. [DE1-SoC] From the command line, turn all of the LEDs on by writing to the file used to control the red LEDs on the DE1-SoC.

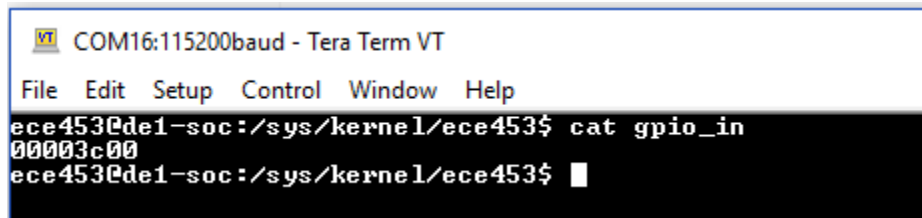
```
sudo sh -c 'echo 0x3FF > gpio_out'
```



```
VT COM16:115200baud - Tera Term VT
File Edit Setup Control Window Help
ece453@de1-soc:/sys/kernel/ece453$ sudo sh -c 'echo 0x3FF > gpio_out'
```

42. [DE1-SoC] From the command line, read from the file connected push buttons.

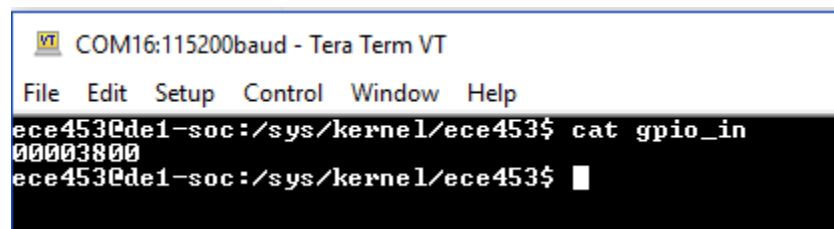
```
cat gpio_in
```



```
VT COM16:115200baud - Tera Term VT
File Edit Setup Control Window Help
ece453@de1-soc:/sys/kernel/ece453$ cat gpio_in
00003c00
ece453@de1-soc:/sys/kernel/ece453$
```

43. [DE1-SoC] When pressing KEY[0], read from the file connected push buttons and observe the change in value.

```
cat gpio_in
```



```
VT COM16:115200baud - Tera Term VT
File Edit Setup Control Window Help
ece453@de1-soc:/sys/kernel/ece453$ cat gpio_in
00003800
ece453@de1-soc:/sys/kernel/ece453$
```



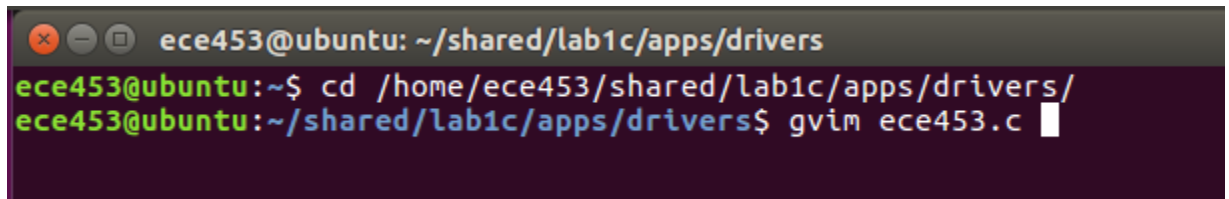
### 3. Example Linux Application to Control LEDs and Push Buttons

The fact that we can control the GPIO pins from the command prompt is a good thing, but in order for you to use the GPIO pins in a meaningful way, you need to be able to write an application that is able to control the GPIO pins.

Controlling the GPIO pins can be accomplished using basic file operations on the same virtual files that you modified from the command line. The Linux kernel provides function calls that allow you to read and write to a file. Below is a summary of the basic file operations you will need to control the FPGA GPIO pins from a Linux application

44. [\[Linux-VM\]](#) Open the driver file that will be used to read and write the GPIO pins. Two functions have been provided that will allow you to read/write to the registers in the ECE453 programmable module used to control the GPIO pins.

```
cd /home/ece453/shared/lab1c/apps/drivers/  
gvim ece453.c
```



```
ece453@ubuntu: ~/shared/lab1c/apps/drivers  
ece453@ubuntu:~$ cd /home/ece453/shared/lab1c/apps/drivers/  
ece453@ubuntu:~/shared/lab1c/apps/drivers$ gvim ece453.c
```

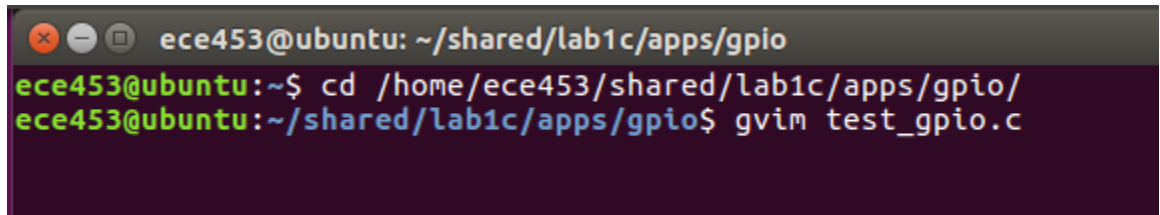
The key thing to gather from these files is that you can communicate with hardware by reading and writing to files that are associated with the custom ECE453 module that was built into the DE1-SoC Image. Accessing a file is accomplished by providing the file name as the 1<sup>st</sup> parameter to the open() function. The second parameter to the open function indicated if you can read, write, or read and write a file.

The open function returns a file handle that will then allow you to access the hardware from user space. We will cover how to write a kernel driver that creates these files in a different lab.

Close the files once you are done examining them.

45. [Linux-VM] Now let's examine a user application that calls the two functions in ece453.c to control the GPIO pins.

```
cd /home/ece453/shared/lab1c/apps/gpio/  
gvim test_gpio.c
```



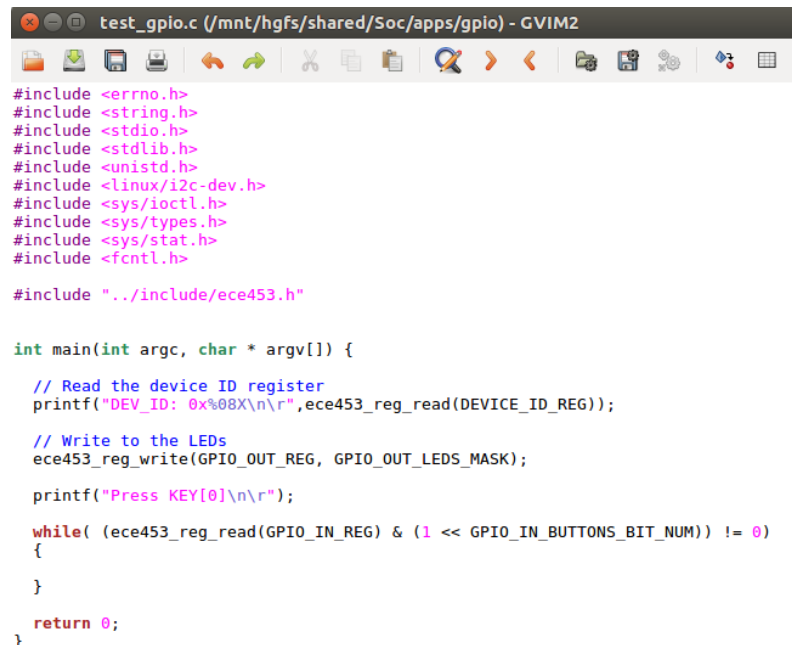
A terminal window titled 'ece453@ubuntu: ~/shared/lab1c/apps/gpio' shows the following commands and output:

```
ece453@ubuntu:~$ cd /home/ece453/shared/lab1c/apps/gpio/  
ece453@ubuntu:~/shared/lab1c/apps/gpio$ gvim test_gpio.c
```

The entirety of the application is shown below. The application makes calls to the two functions found in /home/ece453/shared/lab1c/apps/drivers/ece453.c.

Remember, when you loaded the ece453\_driver into the kernel, it created several virtual files that allowed you to access the ECE453 custom peripheral. The file paths for these files have been defined for you in /home/ece453/shared/lab1c/apps/include/ece453.h. The application makes use of these file paths to read or write a given register.

When you're done examining test\_gpio.c, you can close this file.



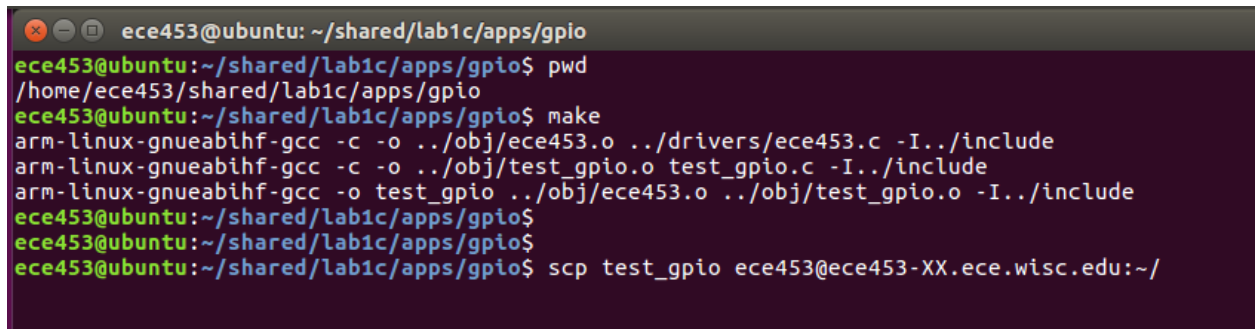
The screenshot shows the source code of test\_gpio.c in a GVIM2 editor window. The code includes standard C headers and a custom header for the ECE453 peripheral. The main function reads a device ID register, writes to LED registers, and enters a loop waiting for a key press.

```
test_gpio.c (/mnt/hgfs/shared/Soc/apps/gpio) - GVIM2  
  
#include <errno.h>  
#include <string.h>  
#include <stdio.h>  
#include <stdlib.h>  
#include <unistd.h>  
#include <linux/i2c-dev.h>  
#include <sys/ioctl.h>  
#include <sys/types.h>  
#include <sys/stat.h>  
#include <fcntl.h>  
  
#include "../include/ece453.h"  
  
int main(int argc, char * argv[]) {  
  
    // Read the device ID register  
    printf("DEV_ID: 0x%08X\n", ece453_reg_read(DEVICE_ID_REG));  
  
    // Write to the LEDs  
    ece453_reg_write(GPIO_OUT_REG, GPIO_OUT_LEDS_MASK);  
  
    printf("Press KEY[0]\n");  
  
    while( (ece453_reg_read(GPIO_IN_REG) & (1 << GPIO_IN_BUTTONS_BIT_NUM)) != 0 )  
    {  
  
    }  
  
    return 0;  
}
```

46. [Linux-VM] You will now compile the application and copy it to the DE1-SoC.

```
cd /home/ece453/shared/lab1c/apps/gpio/  
make
```

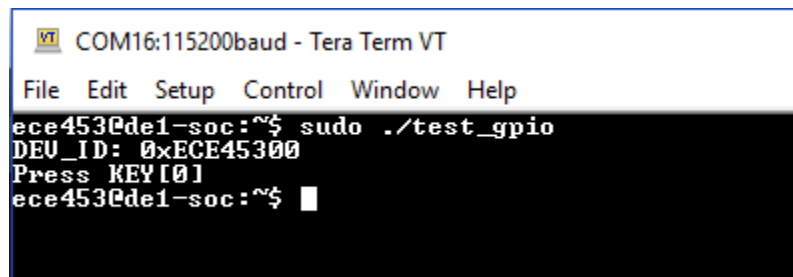
```
scp test_gpio ece453@ece453-XX.ece.wisc.edu:~/
```

A terminal window titled 'ece453@ubuntu: ~/shared/lab1c/apps/gpio' with a dark purple background. It shows the following commands and output:

```
ece453@ubuntu:~/shared/lab1c/apps/gpio$ pwd  
/home/ece453/shared/lab1c/apps/gpio  
ece453@ubuntu:~/shared/lab1c/apps/gpio$ make  
arm-linux-gnueabi-gcc -c -o ../obj/ece453.o ../drivers/ece453.c -I../include  
arm-linux-gnueabi-gcc -c -o ../obj/test_gpio.o test_gpio.c -I../include  
arm-linux-gnueabi-gcc -o test_gpio ../obj/ece453.o ../obj/test_gpio.o -I../include  
ece453@ubuntu:~/shared/lab1c/apps/gpio$  
ece453@ubuntu:~/shared/lab1c/apps/gpio$  
ece453@ubuntu:~/shared/lab1c/apps/gpio$ scp test_gpio ece453@ece453-XX.ece.wisc.edu:~/
```

47. [DE1-SoC] In the serial terminal, run the application. It should print out the information in the Device ID register, set all the red LEDs on, and then wait for you to press KEY[0].

```
cd  
sudo ./test_gpio
```

A serial terminal window titled 'COM16:115200baud - Tera Term VT' with a black background. It shows the following output:

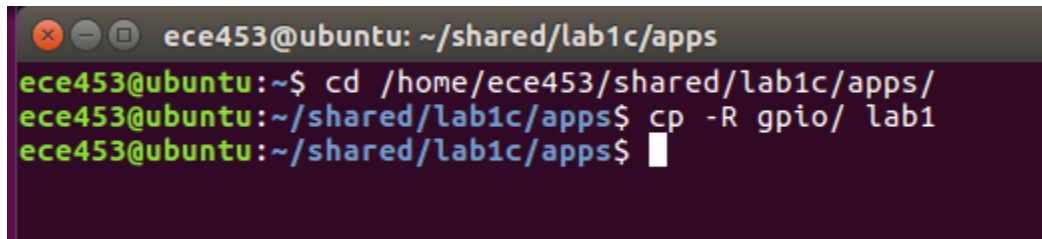
```
File Edit Setup Control Window Help  
ece453@de1-soc:~$ sudo ./test_gpio  
DEV_ID: 0xECE45300  
Press KEY[0]  
ece453@de1-soc:~$ █
```

## 4. Writing Your Own Linux Application

Now you will write your own application that moves which LED is on based on which push buttons are pressed. The application will make use of the files already provided to you.

48. Make a copy of the gpio directory called lab1.

```
cd /home/ece453/shared/lab1c/apps
cp -R gpio/ lab1/
```

A terminal window with a dark purple background. The title bar shows 'ece453@ubuntu: ~/shared/lab1c/apps'. The terminal text shows the user navigating to the directory and copying the 'gpio' directory into 'lab1'.

49. Use an editor to modify test\_gpio.c so that the application initializes the LEDs so that the right most red LED is on.

Examine the buttons once per second. If KEY[1] is pressed, the LED that is on should move to the left. Every time KEY[0] is pressed, the LED that is on should move to the right. If the LED that is on cannot move any further to the left or right, do NOT alter the state of the LEDs. If both buttons are pressed, do nothing.

When you are done modifying your file, compile the application, test it on the DE1-SoC, and debug any errors that you encounter.

LEDR[9:0] are mapped to GPIO\_OUT\_REG[9:0]. KEY[3:0] are mapped to GPIO\_IN[13:10]. Feel free to make use of the macros found in ece453.h when writing your application.

50. **Demonstrate to your TA that your Linux application is functioning correctly.**