EE405 Electronic design lab weeks 4~5

Lab 2 Switch Circuit Control - Procedures

First Week - Problem 2A, 2B

Step 1. User LED0 control using sysfs and command line

- Follow the procedure below and become familiar with sysfs command for GPIO.

1. Connect and turn on hardware – PC and Beaglebone.

- Before starting, activate the cross-development environment using NFS and ssh.
- Please refer to Lab 1 Procedures for review.

2. Select User LED for test

- We will control LED0, User LED located in the Beaglebone Black board during the lab session.
- Default setting for LED0 is heartbeat signal. We will turn off/turn on or modify the signal throughout the experiment.

> ** Note
>
> $: PC directory
>
> # : beaglebone directory

3. Check sysfs file for User LEDs (as superuser)

- Search /sys/class:

```
# sudo su
# ls -F /sys/class
ata_device/      extcon/       mbox/         remoteproc/    tpmrm/
ata_link/        firmware/     mdio_bus/     rfkill/        tty/
ata_port/        gnss/         mem/          rtc/           typec/
backlight/       gpio/         misc/         scsi_device/   ubi/
bdi/             graphics/     mmc_host/     scsi_disk/     udc/
block/           hidraw/       mtd/          scsi_generic/  uio/
```

```
bsg/             hwmon/        net/           scsi_host/     usb_role/
devcoredump/     i2c-adapter/  phy/           sound/         vc/
devfreq/         i2c-dev/      power_supply/  spidev/        vtconsole/
devfreq-event/   ieee80211/    pps/           spi_master/    watchdog/
dma/             input/        ptp/           spi_slave/
drm/             iommu/        pwm/           thermal/
drm_dp_aux_dev/  leds/         regulator/     tpm/
```

- Found “leds”!

- Search /sys/class/leds:

```
# ls -F /sys/class/leds
beaglebone:green:usr0@ beaglebone:green:usr2@
beaglebone:green:usr1@ beaglebone:green:usr3@
```

- Here you see the directories for controlling each of the user LEDs.
- For User LEDs, this directory shows the system information by sysfs and GPIO.
- By default, USR0 flashes a heartbeat pattern. Let’s control USR0 by sysfs command.

13. Get access right to usr0 LED

- Go to the directory /sys/class/leds

```
# cd /sys/class/leds
# cd beaglebone\:green\:usr0
```

- Note that '\' should be included before each ':'.

- List directory

```
# ls -F
brightness  invert          power/      trigger
device@     max_brightness  subsystem@  uevent
```

- See what's in trigger

```
# cat trigger
```

```
none rfkill-any rfkill-none kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalock
 kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftllock kbd-shiftrl
```

```
ock kbd-ctrlllock kbd-ctrlrlock mmc0 mmc1 usb-gadget usb-host timer oneshot disk
-activity disk-read disk-write ide-disk mtd nand-disk [heartbeat] backlight gpio
 cpu cpu0 activity default-on panic netdev rfkill1 phy1rx phy1tx phy1assoc phy1r
adio
```

- This shows trigger can have many values.
- The present value is **heartbeat** (enclosed with '[]'). Check the LED, is it beating?
- You can stop the heartbeat via;

```
# echo none > trigger
```

- Heartbeat is stopped! Check by:

```
# cat trigger
[none] rfkill-any rfkill-none kbd-scrolllock kbd-numlock kbd-capslock kbd-kanalo
ck kbd-shiftlock kbd-altgrlock kbd-ctrllock kbd-altlock kbd-shiftllock kbd-shift
rlock kbd-ctrlllock kbd-ctrlrlock mmc0 mmc1 usb-gadget usb-host timer oneshot di
sk-activity disk-read disk-write ide-disk mtd nand-disk heartbeat backlight gpio
 cpu cpu0 activity default-on panic netdev rfkill1 phy1rx phy1tx phy1assoc phy1r
adio
```

14. Control on/off of usr0 LED

- Turn on/off USR0 LED

```
# echo 1 > brightness
# echo 0 > brightness
```

- USR0 LED is turned on and off!

15. Control periodic on/off of usr0 LED

- LED trigger with timer and 10% duty:

```
# echo timer > trigger
# echo 100 > delay_on
# echo 900 > delay_off
```

16. Return to heartbeat

- Start the heartbeat again

```
# echo heartbeat > trigger
```

- Success?

Step 2. Switching circuit control using sysfs and command line

- Before you start, you have to wire two LEDs with a switching circuit with a transistor array.
    - Please be aware of the burning Beaglebone board.
    - Refer to [Lab 2_Experiment Guide - Background 3. GPIO] for safe GPIO connection with the switching circuit.
- After then, control LEDs on the circuit using sysfs command.

20. Wire Two GPIOs to Two Light LEDs on the breadboard.

- Wire switching circuit to the Beaglebone Black board. Please refer to [Lab2 Experiment Guide - Background 3. GPIO] for wiring sequence.
- GPIO0_30 is connected to LED 1, and GPIO0_31 is connected to LED 2.

21. Access GPIO0_30 for Light 1 as root

- Check sysfs for gpio

```
# ls -F /sys/class/gpio
export    gpio117@  gpio26@  gpio39@  gpio51@  gpio7@   gpio79@     gpiochip32@
gpio10@   gpio12@   gpio27@  gpio4@   gpio60@  gpio70@  gpio8@      gpiochip64@
gpio11@   gpio13@   gpio3@   gpio44@  gpio61@  gpio71@  gpio80@     gpiochip96@
gpio110@  gpio14@   gpio32@  gpio45@  gpio62@  gpio72@  gpio81@     unexport
gpio111@  gpio15@   gpio33@  gpio46@  gpio63@  gpio73@  gpio86@
gpio112@  gpio19@   gpio34@  gpio47@  gpio65@  gpio74@  gpio87@
gpio113@  gpio2@    gpio35@  gpio48@  gpio66@  gpio75@  gpio88@
gpio114@  gpio20@   gpio36@  gpio49@  gpio67@  gpio76@  gpio89@
gpio115@  gpio22@   gpio37@  gpio5@   gpio68@  gpio77@  gpio9@
gpio116@  gpio23@   gpio38@  gpio50@  gpio69@  gpio78@  gpiochip0@
```

- Export GPIO0_30. Note that GPIO number = 0*32 + 30 = 30. Hence GPIO30:

```
# cd /sys/class/gpio
# echo 30 > export
# ls -F
export    gpio117@  gpio26@  gpio38@  gpio50@  gpio69@  gpio78@  gpiochip0@
gpio10@   gpio12@   gpio27@  gpio39@  gpio51@  gpio7@   gpio79@  gpiochip32@
gpio11@   gpio13@   gpio3@   gpio4@   gpio60@  gpio70@  gpio8@   gpiochip64@
gpio110@  gpio14@   gpio30@  gpio44@  gpio61@  gpio71@  gpio80@  gpiochip96@
gpio111@  gpio15@   gpio32@  gpio45@  gpio62@  gpio72@  gpio81@  unexport
gpio112@  gpio19@   gpio33@  gpio46@  gpio63@  gpio73@  gpio86@
gpio113@  gpio2@    gpio34@  gpio47@  gpio65@  gpio74@  gpio87@
gpio114@  gpio20@   gpio35@  gpio48@  gpio66@  gpio75@  gpio88@
gpio115@  gpio22@   gpio36@  gpio49@  gpio67@  gpio76@  gpio89@
gpio116@  gpio23@   gpio37@  gpio5@   gpio68@  gpio77@  gpio9@
```

22. Control Light 1 via GPIO0_30 [Right LED]

- Go to GPIO30 directory

```
# cd gpio30
```

- Set GPIO direction to output

```
# echo out > direction
# cat direction
out
```

- Turn on your own LED 1(Right LED).

```
# echo 1 > value // 3.3V, right LED turns on
# cat value
1
```

- Turn off your own LED 1.

```
# echo 0 > value // 4 mV. Right LED turns off
# cat value
0
```

23. Free GPIO0_30

```
# cd /sys/class/gpio
# echo 30 > unexport
```

24. Control Light 2 via GPIO0_31

- Repeat the same procedure for GPIO0_31 for Light 2 (Left light).

Step 3. Shell program for switching circuit (Problem 2B)

- For this time, we will control a switching circuit for LED lights through shell script.
    - Shell scripts will be given before the lab session; Write through and execute them.

31. make a subdirectory b_LED_ShellScript.

- Make a new subdirectory for shell programming

```
$ mkdir -p ~/DesignLab/2_SwitchControl/b_LED_ShellScript
```

- Use NFS to access the directory in the PC.

32. Test basic scripting

- Read and test:
- Beginners – Bash Scripting, https://help.ubuntu.com/community/Beginners/BashScripting.
- Especially, Sections "Scripting" to "Functions".

33. Test ui_control_lights.sh

- Note on Shell:
- "echo $SHELL" returns /bin/bash on Ubuntu PC & Bone (Hence we are using bash).
- Run the shell script code
- Make this shell script executable with 'chmod'.

```
# chmod a+x ui_control_lights.sh
```

- Run the executable shell script file(as a sudo mode)

```
# ./ui_control_lights.sh
```

- Enter user input repeatedly.
- Check whether the Light LEDs operate as commanded.

34. Test loop_control_lights.sh

- Complete a skeleton shell script code for loop_control_lights.sh.
- Make this shell script executable with 'chmod'.

```
# chmod a+x loop_control_lights.sh
```

- Run(as a sudo user)

```
# ./loop_control_lights.sh
```

- Record elapsed time for M (many) loops.
- Record the result for the final report.

Second Week

Step 4. C++ Program for Switching Circuit Control (Problem 2C)

35. Make a subdirectory c_LED_CppProgram.

- Make a subdirectory

```
$ mkdir -p ~/DesignLab/2_SwitchControl/c_LED_CppProgram
```

36. Edit files

- We need four files:
- gpio_control.hpp: Define gpio control functions [Given already]
- gpio_control.cpp: Actual body of gpio control functions
- test_light_control.cpp: Test light control along with user input.
- loop_light_control.cpp: Test time consumption of the ON/OFF light control loop for N times.
    - .hpp file will be given.
    - Skeleton code for .cpp files will be given. Please complete the functions and control codes.

```
// File gpio_control.hpp
// Function definitions for gpio_control.cpp

#define MAX_BUF 64 /* For the max length of string */

int gpio_export(unsigned int gpio); // gpio means gpio number (0 to 127)
/* - input : GPIO port number to export
   - function that exports GPIO port */

int gpio_unexport(unsigned int gpio);
/* - input : GPIO port number to unexport
   - function that unexports GPIO port */

int gpio_set_dir(unsigned int gpio, unsigned int out); // out = 0: in. out = 1:
out.
/* - input : GPIO port number to set direction / desired direction(out : 1, in :
0)
  - function sets the direction of the exported GPIO port */

int gpio_fd_open(unsigned int gpio); //Returns file descriptor fd
/* - input : GPIO port number to get fd
   - output : fd(file descriptor; information used to write values to the
configured GPIO)
  - function which opens the GPIO port and get the file descriptor info */
```

```
int gpio_fd_close(int fd);
/* - function which closes the GPIO port using file descriptor information */

int gpio_fd_set_value(int fd, unsigned int value); // value can be 0 or 1
/* - input : GPIO port file descriptor / desired value(1: ON, 0: OFF)
   - Sample code to turn on/off the GPIO (IN CASE OF out direction) */
```

37. Compile

- Cross-compile C++ files to generate executables.
    - You might use VSCode to generate CMakeLists.txt and cross-compile. Please refer to cmake on VSCode on the Lab1 Procedure.

38. Run on Bone as root (using NFS).

- Does LED lights operate as expected?
- Run executables on the Beaglebone Black board as root.
- Record the result for the final report.

Step 5. C++ Program for Solenoid Magnet (Problem 2D)

39. Wire the GPIO port with Solenoid magnet on the breadboard.

- Connect Bone P8/P9 to Resistor, Transistor array, Resistor, and solenoid magnet on Breadboard.
- Use GPIO0_30 to connect the solenoid magnet.

  ** In this case, you must use an external power supply to power the solenoid magnet.

  ** Please check the appropriate Voltage/Current/Power for the solenoid magnet(12V / 0.26A / 2.5W), and set the input voltage and current for the external power supply.

40. Make a subdirectory d_GripperControl.

- Make a subdirectory

```
$ mkdir -p ~/DesignLab/2_SwitchControl/d_GripperControl
```

41. Edit files

- To utilize it later, we will make a C++ file in which the solenoid magnet control functions are defined; Open / turn on / turn off / close the solenoid magnet GPIO.
- We need 3 files.
- gripper_control.hpp: Define functions to control solenoid magnet [Given already]
- gripper_control.cpp: Actual body of solenoid control functions
- gripper_test.cpp: Test the functions defined in gripper_control.cpp
- You must design your .cpp code using functions defined in gpio_control.cpp

1. Design gripper_control.cpp with given gripper_control.hpp file

```
// File gripper_control.hpp
// Function definitions for gripper_control.cpp

int gripper_open(unsigned int gpio);
// open the GPIO port for the solenoid magnet
// input : GPIO port number
// output : file descriptor of GPIO port

int gripper_on(unsigned int fd_gpio);
// turn on the solenoid magnet
// input : GPIO port number

int gripper_off(unsigned int fd_gpio);
// turn off the solenoid magnet
```

```
// input : GPIO port number

int gripper_close(unsigned int gpio, unsigned int fd_gpio);
// close the GPIO port for the solenoid magnet
// input : GPIO port number, file descriptor for GPIO
```

42. Compile

- Cross-compile C++ files to generate executables 'gripper_test'.
    - You might use VSCode to generate CMakeLists.txt and cross-compile. Please refer to cmake on VSCode on the Lab1 Procedure.

43. Run on Bone as root (using NFS).

- Run generated C++ executable on the Beaglebone Black board as root.
- Does the solenoid magnet operate as expected?